

AN ACTIVITY THEORY-BASED ARCHITECTURE TO
ENHANCE CONTEXT-AWARE COLLABORATION IN
SOFTWARE DEVELOPMENT IN THE CLOUD

STANLEY FRANCIS EWENIKE

STAFFORDSHIRE UNIVERSITY

A thesis submitted in fulfilment of the requirement of Staffordshire University for the
degree of Doctor of Philosophy

2023

Acknowledgment

This PhD journey has been quite a learning curve and a roller-coaster of emotional fluctuations. Some days were very exciting and brimming with motivation and confidence. Some days were quite frustrating, where one is surrounded by a sea of information but still clueless. Some days brought me close to tears, while some other days brought a glimmer of hope. Some days, I felt like I knew it all and had everything figured out. Other days, I felt lost and in over my head. Words cannot describe all the days, nor the emotions. However, at the end of the journey, nothing is left, except the feeling of gratitude. It is finished!

I am grateful to: God Almighty, to my supervisors - Professor Elhadj Benkhelifa and Professor Claude Chibelushi, for their mentorship and guidance; to the external and internal examiners for their time and feedback on this piece of work; to my family - parents, wife, siblings and even my babbling 3-month-old son who doubled as my intermittent biological alarm clock during the last days of my thesis write-up; to my colleagues and friends – Siyakha Mthunzi, Oluwasegun Adedugbe, David Odebade, Chigozie Onyekaba, Thomas Welsh; and to Staffordshire University. A special thanks to Russell Campion, Clare Stanier and Uchitha Jayawickrama. And to anyone not mentioned here, just know you are not forgotten. Your support is probably too much to fit on a page. May God bless everyone. And me too. Let me catch my breath, and then, on to the next learning curve, because life is one un-ending school. It is finished!

Table of Contents

1	Introduction	15
1.1	Background	15
1.2	Key research motivation	18
1.3	Research questions	19
1.4	Research aims	20
1.5	Research objectives	20
1.6	Research contributions	20
1.7	Summary	20
2	Research philosophy and methodology	22
2.1	Introduction	22
2.2	Research philosophy	25
2.3	Underlying principles of a research philosophy	26
2.4	Positivism vs Interpretivism	27
2.5	Adopted research philosophy and methodology for this research project	29
2.6	Analysis of methods and justification	30
2.7	Summary	31
3	Literature review.....	32
3.1	Introduction	32
3.2	Methodology.....	34
3.3	Review of software engineering trends and its relevance	38
3.4	Extending Boehm’s software engineering trends’ diagram	41
3.5	Software development process and models	44
3.6	Gaps and challenges facing collaborative software development in the cloud.....	51
3.7	Impact analysis of some of the more prominent gaps and proposed recommendations.....	65
3.7.1	Need for cloud-based collaborative software development architectures with explicit theoretical foundation	66
3.7.2	Need for effective methods for capturing and representing contexts and other related data in a cloud-agnostic format for generation of actionable insights.....	68
3.7.3	Need for effective ways of managing complexity throughout cloud-based collaborative software development process	69
3.7.4	Need for standards and adequate metrics for benchmarking cloud-based collaborative development and testing	70

3.8	Summary	70
4	Conceptual foundations.....	73
4.1	Introduction	73
4.2	Cloud computing overview	74
4.3	A SWOT analysis of cloud computing	76
4.4	Collaboration overview.....	79
4.5	Key dimensions for collaboration	80
4.5.1	Coordination	81
4.5.2	Communication.....	82
4.5.3	Balance of member contributions	83
4.6	Classification of approaches for enhancing collaboration	83
4.6.1	Classification based on empirically measured activities within software development process.....	84
4.6.2	Classification derived from objectives of activities	85
4.6.3	Classification based on software development process characteristics	86
4.6.4	Classification based on analysis of interactions between all aspects of the process	88
4.7	Context awareness overview.....	89
4.8	Relevance of context-awareness and proposed process for application of contextual information to collaborative software development process in the cloud	91
4.9	Summary	99
5	Theoretical framework	101
5.1	Introduction	101
5.2	Related work.....	104
5.3	A formal process for adoption of an appropriate theoretical basis	108
5.3.1	Overview of the proposed formal process	109
5.3.2	The problem scenario	110
5.3.3	Criteria for theoretical foundation	111
5.3.4	Question.....	111
5.3.5	Parameters.....	112
5.3.6	Assumptions.....	113
5.3.7	Initial conditions.....	113
5.3.8	Modelling the process.....	113
5.4	Application of the proposed formal process	116

5.5	Cross-sectional review of relevant theories	117
5.5.1	Information Foraging Theory (IFT).....	118
5.5.2	Game Theory.....	119
5.5.3	Complexity Theory	121
5.5.4	Actor-Network Theory (ANT).....	122
5.5.5	Activity Theory (AT).....	123
5.6	Analysis and justification for Activity theory as theoretical basis for cloud-based collaborative software development.....	129
5.7	Developing an AT-based framework for enhancing context-aware collaboration in cloud-based software development.....	135
5.7.1	Step 1: Define use case scenario	135
5.7.2	Step 2: Define requirements.....	137
5.7.3	Step 3: Identify Activity theory concepts/components to leverage.....	137
5.7.4	Step: Mapping of Activity theory concepts to cloud-based software development aspects.....	142
5.7.5	Step: Define baseline activity structure for collaborative software development process	153
5.8	Summary	157
6	The Architecture	158
6.1	Introduction	158
6.2	Overview of an Architecture	160
6.3	Review of architecture patterns	162
6.3.1	Layered architecture pattern.....	162
6.3.2	Service-oriented architecture (SOA) pattern.....	163
6.3.3	Microservices pattern	163
6.3.4	SOA architectural patterns vs. Microservices architectural patterns.....	164
6.4	Modelling the architecture	170
6.5	Architecture description of activity scenario for collaborative software development process in the cloud	174
6.6	High-level architecture components	175
6.7	Activity and activity sequence decomposition	178
6.8	Operation-condition sequences for activity	180
7	Architecture implementation and evaluation	193
7.1	Introduction	193
7.2	Evaluating and validating an architecture	193

7.3	Architecture Implementation approach	195
7.4	Software requirements specifications (SRS) for POC implementation of architecture.....	198
7.5	Development and deployment	200
7.6	Evaluation of POC implementation and functionality performance	202
7.6.1	Case study Test Scenario:	203
7.6.2	Implemented service, roles, and activity sequence for POC	205
7.7	Operation-condition sequences for case study scenario	206
8	Conclusion and future work.....	208
9	References	212
10	Appendices.....	239
A.	List of publications	239
	Publications.....	239
	Seminar presentation	239
B.	POC IMPLEMENTATION – SOURCE CODE	240
	Clients.....	240
	Providers	245
	Models	248
	Services	249
	HTTP (Controllers, middleware, requests).....	258
	Configuration	277
	Console.....	300
	Exceptions	300
	Public.....	301
	API	302
	Database	305
	Tests	306
C.	POC IMPLEMENTATION OF FUNCTIONALITY.....	307
	Source code for API call	316
	#YAML source script for test scenario (Plugin tool used – Blazemeter)	318
	Metadata for Test scenario.....	320
	Check-Login-functionality.jmx source file.....	320

List of Figures

Figure 1: An illustration of typical factors affecting direction of a research project.	22
Figure 2: Highlighting “what” influences a researcher's initial approach towards research. .	23
Figure 3: Adapted approach used for design of appropriate research methodology for this project (Easterby-Smith et al., 2012)	25
Figure 4: Characteristic principles underlying research philosophy and design.	27
Figure 5: Barriers to collaboration in Cloud-based software development process.....	33
Figure 6: Decade survey of relevant collaborative software development within cloud context, grouped by year.....	36
Figure 7: A flowchart representation of the adopted method.....	37
Figure 8: A timeline of software engineering trends spanning six decades (Boehm, 2006a).	39
Figure 9: Typical makeup of a Software development project	51
Figure 10: representation of Cloud computing characteristics based on NIST definition (Badger, Lee et al., 2012)	74
Figure 11: A view of the Cloud computing architecture (Zafar et al., 2017)	75
Figure 12: Service provisioning of Cloud Computing (Whaiduzzaman et al., 2014)	76
Figure 13: A view of Cloud challenges & Issues.....	77
Figure 14: Key dimensions for examining and assessing collaboration needs in collaborative cloud-based software development processes	83
Figure 15: Classification based on empirically measured development activities	85
Figure 16: Classification based on objectives of development activities	86
Figure 17: Classification based on both internal and external characteristics of the development process.....	88
Figure 18: Classification based on analysis of interactions between all development process aspects	90
Figure 19: Classification of context.....	91
Figure 20: Understanding context-awareness requirements for cloud-based collaborative software development.....	93
Figure 21: Adapted key dimensions for collecting, categorizing, analysing, and applying contextual information	96
Figure 22: Applying contextual information to the Collaborative Software Development process	97
Figure 23: Classification of theory use.....	102
Figure 24: Systematic protocol for identifying relevant literature	105
Figure 25: Relevance of an adequate theoretical foundation for cloud-based collaborative software development.....	112
Figure 26: Flowchart for a formal process for adoption of an appropriate theoretical basis	115
Figure 27: key focus points and impact areas for cloud-based collaborative software development based on output of formal theoretical process applied in Sections 5.1 – 5.4, in line with Gregor’s taxonomy (Gregor, 2006; Gregor & Jones, 2007)	117
Figure 28: First generation of Activity Theory	129
Figure 29: object-oriented transitive relationship between object and activity instance	130

Figure 31 A representation of time context between activity instances	134
Figure 32 Conceptualizing the problem scenario	136
Figure 33 Initial working requirements model for stakeholder collaboration	137
Figure 34 Hierarchical breakdown of the Activity	139
Figure 35 Hierarchical breakdown of the Subject component of Activity theory	140
Figure 36 Hierarchical breakdown of the Object component of Activity theory	141
Figure 37 Hierarchical breakdown of the Tool component of Activity theory	141
Figure 38 Visualizing future context-aware collaborative software development process enabled at the core by solid theoretical foundation	146
Figure 39 Adopted mapping of main software development process components to activity theory components.....	147
Figure 40 AT-based conceptualization of distributed interacting activity systems in the cloud	147
Figure 41 Mapping proposed theoretical framework - Initial architecture block diagram for existing system.....	148
Figure 42 Zooming in on Mapping proposed theoretical framework - Initial architecture block diagram for existing system: Presentation layer.....	149
Figure 43 Zooming in on Mapping proposed theoretical framework - Initial architecture block diagram for existing system: Activity layer	150
Figure 44 Zooming in on Mapping proposed theoretical framework - Initial architecture block diagram for existing system: Managed cloud platform layer	151
Figure 45 Mapping proposed theoretical framework to existing system – architecture block diagram	152
Figure 46 Distinction between Activity levels.....	154
Figure 47 Proposed Activity baseline structure (schema) for designing/creating a collaborative software development activity	154
Figure 48 Modelling context-aware development process sequence	155
Figure 49 Object-task specification transformation sequence.....	156
Figure 50 Prominent identified use cases for the proposed platform	160
Figure 51 Context model for proposed AT-based architecture for enhancing context-aware collaboration cloud-based software development process	171
Figure 52 Database service categories	177
Figure 53 Service decomposition & structure	182
Figure 54 Project stakeholders, Cloud frontend and API Gateway	183
Figure 55 Cloud frontend, API Gateway, Cloud Activity service, and Stakeholder service ...	184
Figure 56 API Gateway and cloud software development services	185
Figure 57 API gateway Cloud collaboration services, Cloud infrastructure services and Cloud activity analysis services	186
Figure 58 Datastores for Cloud software development services	187
Figure 59 Datastores for Cloud collaboration services, Cloud infrastructure services and Cloud activity analysis services	187
Figure 60 Event-driven interservice communication for Activity layer	188
Figure 61 Messaging Middleware pattern (Publish/Subscribe)	189

Figure 62 AT Cloud-based collaborative software development process workflow (cross-functional flow chart).....	190
Figure 63 CFTM_Test workflow	191
Figure 64 PM_Stakeholder Workflow.....	192
Figure 65: Cloud activity workflow	203
Figure 66 Summary of the research journey	209

List of Tables

Table 1: Summary of differences between Positivist and Interpretivist philosophical approaches (Weber, 2004)	28
Table 2: Query strings for systematic literature search	35
Table 3 Software engineering trends post 2010 till date.	42
Table 4 Cross-sectional comparison of software development models	46
Table 5 summary of the gaps covering collaboration in both traditional and cloud-based collaborative software development.....	55
Table 6 A Survey of cross-section of notable open-source industry tools/platforms towards Cloud-based SDLC process	60
Table 7: SWOT analysis of Cloud Computing	78
Table 8: Adaptation of Zachman's framework for definition of context data and levels	95
Table 9 Adapted contradiction matrix for analysis of contextual information for an object in cloud-based software development.....	98
Table 10 Adapted contradiction matrix for representation of activities and tasks in cloud-based software development	98
Table 11 Employing matrix multiplication in adapting contradiction matrix for applying contextual information to activities and tasks in cloud-based software development.....	99
Table 12 Overview of steps for searching, selecting, and deduping relevant articles based on use of relevant keywords.....	105
Table 13 Cross-sectional summary of categories of common approaches for selecting theoretical foundations in the field of software development	107
Table 14 Proposed process' variables and description	112
Table 15 4-point Likert scale for the formal process for selecting theoretical foundation...114	
Table 16 Summary of evaluated theories and their matches to appropriate importance or use category for cloud-based collaborative software development:.....	126
Table 17 Mapping AT to collaborative software development components.....	144
Table 18 Summary of gaps in cloud-based collaborative software development (Ewenike et al., 2010, 2017b)	158
Table 19 comparative summary of common software architecture patterns (Richards, 2015b)	166
Table 20 Components of context model for proposed AT-based architecture for enhancing context-aware collaboration cloud-based software development process.....	172
Table 21 Summary best practice design and evaluation criteria (Bruegge & Dutoit, 2004) .194	
Table 22 Architecture implementation approach	196
Table 23: Test cases for sample feature to test.....	204

List of abbreviations

ABBREVIATION	MEANING
ALMA	Architecture Level Modifiability Analysis
ANT	Actor Network Theory
API	Application Programming Interface
AT	Activity Theory
ATAM	Architecture Trade-off Analysis Method
AT-based architecture	Activity Theory-based Architecture
AWS	Amazon Web Services
CCSD	Cloud-based Collaborative Software Development
CFTM	Cross-Functional Team Members
CSD	Collaborative Software Development
CU	Community Users
DFX	Design For Excellence
EC2	Elastic Compute Cloud
FOSS	Free Open-Source Software
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IFT	Information Foraging Theory
LAMP	Linux, Apache, MySQL, and PHP
MSMQ	Microsoft Messaging Queuing
PCSDE	Process-Centred Software Development Environments
PFIS	Programmer Flow by Information Scent
PHP	Hypertext Pre-processor
PM	Project Manager

POC	Proof-Of-Concept
R&D	Research & Development
REDIS	Remote Dictionary Server
REST	Representational State Transfer
SAAM	Software Architecture Analysis Method
SDLC	Software Development Lifecycle
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SRS	Software Requirements Specifications
SVM	Software Version Management
SWOT	Strengths, Weaknesses, Opportunities, Threats
TWQ	Teamwork Quality Model
UDAO	User Data Access Object
UDM	User Delegate Module
UO	User object

Abstract

This research study reviews collaborative software development and assesses the impact of cloud computing in this domain. This is with a view towards identifying challenges to effective context-aware collaboration, as well as opportunities, risks, and potential benefits that could come from a well-defined structured leverage of cloud capabilities. Findings from systematic review of literature indicate that adoption of cloud computing played a significant part in bringing about trends such as: movement of traditional applications and processes to the cloud; cloud development environments; increased distribution in teams and resources; increased diversity in requirements; changes in how software is developed, tested, deployed, accessed, and maintained. These trends have in turn introduced factors such as: massive scale; additional layers of complexity in abstraction levels, entity characteristics and entity relationships within the development process. This additional layer of complexity translates into increase in contexts i.e., information that can be used to characterize states of entities. This is in addition to existing traditional complexity i.e., measure of proportionality of activities and tasks within the process.

Some notable efforts towards improving collaboration in software development in the cloud include: transitioning development environments, tools and teams to the cloud; provision of code repositories and version control functionality to support collaboration between developers; provision of platforms to enhance collaboration between developers and end-users in early stages of the process via registered project campaigns and targeted questionnaires; provision of platforms with integrated social networking tools. However, an essential missing piece for more effective context-aware collaboration in the process is, the need for ways of addressing resultant complexity from cloud adoption and capturing actionable contexts. Capturing and communicating contextual information can help improve awareness and understanding and facilitate role-based coordination of distributed team members including users, and not just developers. This would ensure all stakeholders are always on the same page even if not in same location, across all phases of development.

The main aim of this research study is to apply a new architecture framework underpinned by the right theoretical foundations, capable of leveraging cloud capabilities, harnessing contexts and addressing complexity to enhance context-aware collaboration in cloud-based software development. To achieve this aim, knowledge gleaned from the systematic

literature review and the gap-impact analysis was thematized and synthesized to provide optimal recommendations to serve as roadmap guide for the development and evaluation carried out, and subsequent knowledge contributions. Key dimensions were adapted, along with development of classifications for approaches to enhancing collaboration in software development in the cloud. The key dimensions created were for - assessing collaboration needs; definition of context data and levels; collecting, categorizing, analysing, and applying contextual information to tasks, activities, and stages within software development in the cloud. These dimensions and classifications are useful for identification of reliable ways of measuring collaboration and success factors, as well as managing complexity and ensuring synchronous regularity of process and understanding within the development process in the cloud. A formal process was proposed to aid selection of an appropriate theoretical basis and assembling of a theoretical framework and methodology to underpin the architecture for enhancing context-aware collaboration in cloud-based software development. This was necessary due to the current lack of a *de-facto* architecture method for cloud-based software development. An activity theory-based architecture has been designed and developed, along with a Proof-of-Concept (POC) implementation that leverages cloud capabilities, for evaluation of the architecture. This architecture presents a novel approach for enhancing collaboration in software development in the cloud due to its underlying activity theory-based tenets that considers 'activity' as the unit of analysis, and ideal for activity systems and ease of identification of congruencies and contradictions present or capable impacting related components of the activity system and its ecosystem. The conclusions for this research study, limitations and future research directions have been discussed at the end of this thesis work.

1 Introduction

1.1 Background

Today's global economy is characterised by organisations that are increasingly decentralised, geographically distributed, with diverse workforce that must share resources in day-to-day operations. Almost all organisations need software for different day-to-day operations, for different reasons, and, in different scenarios (Murthy & Suma, 2017; Mourad et al., 2020). The goal of the software development process is to create software that can be utilized for different scenarios as required (Johnson & Ekstedt, 2016). There are many aspects and stakeholders in the software development process, that come together to ensure software developed is fit for intended purpose i.e., meets users' needs or stated requirements. Collaborative software development process refer to how all stakeholders and aspects within a software development project, work together throughout the development process to achieve a desired final goal or outcome (Mistrík et al., 2010).

Central to collaborative software development process, are activities which result in creation of knowledge-based artefacts, requiring contributions from multiple persons or teams (Whitehead et al., 2010). These activities can be defined as actions of individuals or members of the development teams that can be measured based on characteristics such as quantity, correctness of execution, complexity of tasks within activities, speed or ease of execution, etcetera (Lindsjörn et al., 2018). These individuals or team members are connected by interactions that can be studied via frequency and intensity. Both empirical and anecdotal evidence emphasize that the success of the software development process depend not only on quantity of activities and correctness of activities carried out by the individuals involved; but also on the quality of collaboration or interactions between the individuals involved in the process (Weimar et al., 2017; Lindsjörn et al., 2016; Mistrík et al., 2010).

The advent of cloud computing has impacted the nature of interactions individuals and teams, as well as, the process of developing software (Kannan, 2012). One of such impact of cloud computing on the software development process is the movement of development environments to the cloud. This has contributed to software development process in the cloud and the potential to benefit from the ability of the cloud to provide a substrate platform for collaboration support. This support can be via: on-demand broad network access to a pool

of decentralised or distributed resources that can be quickly provisioned for development and testing activities (Fylaktopoulos et al., 2016a); relative reduction in time and effort needed to set up high availability development and testing environments (Hiremath & Patil, 2015); provision of cloud-based applications and tools for communication, collaboration and resource sharing on the fly, for distributed teams (Oberhauser, 2013a, 2014).

However, in addition to such benefits, moving the software development process to the cloud has brought about increase in complexity and contexts that need to be duly considered (Hiremath & Patil, 2015). The various components that make up a distributed team collaborating within the software development process in the cloud needs to be able to have access to relevant data pertaining to every component or entity, at all times. This will facilitate creation or improvement of shared understanding, as well as, inform technical improvements and decisions (Fylaktopoulos et al., 2016a). This data is referred to as contextual information.

Contextual information can help to create and improve understanding across roles within the development process, as well as, improve shared meaning about the state of entities, activities, tasks and artefacts involved in cloud-based software development process (Omoronyia et al., 2010). It can also be used for making managerial and technical decisions and improvements. Context-awareness provides the ability to be able to obtain and process contextual information that can help to ascertain state of entities within the development process, and support proactive, timely actions and operations (van Engelenburg et al., 2019). However, these contextual information are sometimes underestimated, ignored, or not given enough consideration, with possible consequences including (Mistrík et al., 2010):

- i. undermining of collaboration in the cloud-based development process
- ii. gaps between theoretical benefits of collaborative software development in the cloud and practical challenges to an effective collaborative development process (Omoronyia et al., 2010)
- iii. negative impact on the ability to facilitate a reproducible, context-aware development process in the cloud.

Notable efforts in cloud-based collaborative software development has been mainly in areas of: asynchronous collaboration; collaboration in isolated aspects of the development process such as coding activities; use of open-source tools for contributing, improving, and managing code; design of context-based systems for complex environments; and leveraging of social

networking as an enabler (Mahmood & Saeed, 2013; van Engelenburg et al., 2019; Fylaktopoulos et al., 2016). Leveraging key characteristics of the cloud present a research area of potential synergies for collaborative software development process. Prominent among such characteristics is the ability to provide on-demand network access to a configurable ample pool of resources (Mahmood & Saeed, 2013), and capability of the cloud to be used as a platform for effective collaboration, communication and data exchange (Barenji et al., 2021; Ramis et al., 2016). These key characteristics present potential for effective synchronous collaboration and management of knowledge-based artefacts in cloud-based software development.

The advent and increased adoption of cloud computing, has brought about movement of traditional applications and development environments to the cloud (Mistik et al., 2016). Results of this include: introduction and increase in tool heterogeneity; distribution of services and teams; changes in how software is developed, tested, maintained, accessed and stored (Guha & Al-Dabass, 2010); increased availability of operational data for development and deployment insights; API-driven infrastructure instances; spawning and stopping of cloud-based development instances (Cito et al., 2015), increased diversity and differences in team makeup, operations and communication (Sangwan et al., 2020). All the above-mentioned, create a need to investigate gaps and impact, as well as ascertain potentially better ways to leverage the cloud for more synchronous, context-aware collaboration. Without appropriate research and development efforts, attempts at cloud-based software development risk increase in the number of defects and issues that could result within cloud-based software development process (eds. Z. Mahmood & S. Saeed, 2013). This would ultimately affect the quality and usability of software, as well as release time.

Existing literature have shown that leveraging the above-mentioned cloud capabilities can improve collaboration and efficiency (Rauch et al., 2016a; Valilai & Houshmand, 2013; Golightly et al., 2016). With adequate research and development efforts, areas of synergy could be identified towards a more collaborative software development process for creating applications with features and performance tailored for best results in the cloud. Cloud computing has the potential to be an important substrate for support and improvement of collaboration within the software development process via its ability to provide on-demand broad network access to ample pool of resources (Fylaktopoulos et al., 2016a), whilst at the

same time, reducing cost of developing, testing, deploying and maintaining software in distributed organizations.

1.2 Key research motivation

The advent of relatively modern trends and paradigms such as grid computing, virtualization, cloud computing, etc., have contributed to increasing distribution (geographic), decentralisation (control) and diversity in modern organisations (Sitaram & Manjunath, 2011; Boehm, 2006a; Dillon et al., 2010; Boehm, 2010; Bojanova et al., 2013). These trends increased the reach of organisations, reduced cost, created new revenue streams, and improved business models and opportunities (Chhabra et al., 2010; Gai & Li, 2012). However, these trends also ushered in some challenges e.g., availability and privacy (Patidar et al., 2012) and increased emphasis on need for better ways of facilitating and supporting efficient collaboration. In today's global economy, teams and individuals within the workforce must share resources, collaborate in day-to-day operations, improve productivity and business agility, as well as reduce cost and waste. This is the situation faced by software teams in organisations with traditional or convention software development processes (Mistrík et al., 2010).

Though cloud computing offers a lot of upsides for the software development process e.g., greater degree of abstraction of underlying infrastructure, and elastic allocation of resources, the area of cloud-based collaborative software development is still under-exploited in terms of enhancing effective collaboration (Benfenatki et al., 2014; Chhabra et al., 2010). It remains a viable area in need of academic research efforts (Rellermeyer et al., 2013). The following points outlined below, provide motivational basis for this research project:

Firstly, though software engineering is a well-established discipline for the design and development of software; software engineering trends and adoption of cloud computing, necessitate a review of the current software development process in the cloud (Mahmood & Saeed, 2013; Krishna & Jayakrishnan, 2013). Preliminary research reveals the need for adaptation of the current software development process for more effective context-aware, collaboration and management in the cloud. This need has been further exacerbated with the increase in distribution of teams and resources in the cloud due to effects of globalisation (Haig-Smith & Tanner, 2016). Leveraging the characteristics of the cloud as an

enabler would help to create: improved understanding across roles, technical improvements, and better shared meaning within entities(Mistrík et al., 2010; Benfenatki et al., 2014)

Secondly, a review of current industry offerings reveals a need for context-aware cloud-based collaborative software development architectures with explicit theoretical foundation(Chhabra et al., 2010). A growing school of thought that has gained traction points to the existence of two main classes of challenges in this need category and their unique distinction (Ralph et al., 2013; Johnson & Ekstedt, 2016; Zahedi et al., 2017; Adolph & Kruchten, 2013; Clear, 2009; Benfenatki et al., 2014; Chhabra et al., 2010). The first class of challenges are those that exist because of fundamental flaws in the suitability of the existing process methodology for a more distributed process in the cloud, hence the need for a more suitable process methodology grounded in theoretical foundations(Panigrahi et al., 2017). The second class of challenges in this need category alludes to challenges that exist as a result of unsuitability of architectures and development environments for cloud based development and the mismatch between the these structures and the process methodologies they are supposed to enable (Cico & Cico, 2019). This results in a lack of de-facto standard for cloud-based software development architectures (Gill & Chana, 2012).

Lastly, there is a need for more efficient interoperability, awareness, and communication, to enhance context-aware collaboration within the cloud-based software development process(Nogueira et al., 2016). This requires standardised approaches and models that can help to ensure both data and applications are interoperable across distributed setups(Andres et al., 2021).

1.3 Research questions

The questions this research seeks to answer are as follows:

- i. Does the gradual shift in the way applications are accessed, utilized, and stored in the cloud imply a need to review current software engineering methodologies for the software development process?
- ii. What are the challenges to context-aware, collaboration in software development in the cloud and how can these be addressed using suitable theoretical concepts or foundation?

- iii. How can cloud computing be further leveraged for more efficient collaboration in cloud-based software development process?

1.4 Research aims

To propose an architecture framework to enhance context-aware collaboration in the cloud-based software development process.

1.5 Research objectives

- i. Assess existing collaborative software development practices and research efforts.
- ii. Review stages in software development to identify challenges, opportunities and benefits facing a typical cloud-based software development process.
- iii. Investigate and assess the impact of cloud computing on the software development process.
- iv. Investigate cloud capabilities that can be leveraged within the software development process and review relevant challenges and issues that may arise.
- v. Propose a theoretical framework architecture for enhancing context-aware collaboration in cloud-based software development process.
- vi. Carry out a proof-of-concept implementation and evaluation of proposed framework architecture based on identified priorities from research outcomes.

1.6 Research contributions

- i. A formal process for the adoption of an appropriate theoretical basis for a research project
- ii. An activity theory framework and methodology to enhance context-aware collaboration in the cloud-based software development process.
- iii. An activity theory-based architecture to enhance context-aware collaboration in the cloud-based software development process.
- iv. A proof-of-concept prototype implementation and evaluation

1.7 Summary

In today's distributed environment, the ability to be able to capture and communicate contextual information in a timely manner, among all stakeholders involved in a software

development project is an essential requirement for fostering engagement, effective action and enhancing collaboration(Singh & Chana, 2013; Lau et al., 2017; Alvertis et al., 2016a). This can contribute towards improvement of the quality of software artefacts developed, as well as consistency and compliance. This research attempts to identify cloud characteristics and aspects that could be leveraged as collaborative mechanisms, and aligned with appropriate theoretical concepts and cloud-centric methodologies(Raj et al., 2013), to create a streamlined formal architecture to address increasing and diverse collaboration needs of cross-functional software development teams in today's global organisations.

The outline of this research thesis is as follows:

Section 1 of this thesis introduces the research area, presents research motivation, research questions, aims, objectives and contributions of this thesis.

Section 2 presents the adopted research philosophy and methodology, as well as an analysis and justification for the approach adopted in this research.

Section 3 reviews existing body of knowledge in collaborative software development in the cloud to identify gaps, challenges, and issues pertinent to the research area. This section also presents a gap analysis and impact analysis of gaps addressed.

Section 4 reviews pivotal conceptual foundations for architecture developed in this thesis and presents classifications based on thematic analysis of recurrent themes from literature review.

In Section 5, a formal process for streamlining search for adequate theoretical foundations for analysing collaborative software development in the cloud is developed. This process is applied in the selection process, along with justification. Finally, this section presents a theoretical framework for enhancing context-aware collaboration in cloud-based software development process.

Section 6 presents and describes the Activity Theory-based architecture for enhancing context-aware collaboration in cloud-based software development.

Section 7 discusses the implementation and evaluation of POC implementation of the developed architecture.

Section 8 presents the conclusion and potential future direction for work done in this thesis.

2 Research philosophy and methodology

2.1 Introduction

In industry and academic settings, the word "research" sometimes has different connotations. The most basic distinction between industrial research and academic research is that the former generally tends to be more applied in nature, while the latter generally tends to be more fundamental in nature, seeking to introduce novelty within research projects (Saunders et al., 2009). Industrial research seeks to identify or develop a solution, or a set of solutions to a specific problem. A typical academic research process has its basis on understanding of underlying philosophical set of elements, also known as scientific paradigms (Cumming, 2012). These include - pre-existing values, assumptions, beliefs, and perspectives. Underlying philosophical set of elements are often shaped by various life events, experiences and personal beliefs; and condense into research methodology (Somekh & Lewin, 2005).

Scientific paradigms constitute a starting point for reasoning about research, highlighting relationship between belief and approach towards conceiving research. The direction of a research project can be directly or indirectly influenced by the research strategy a researcher chooses to take, and is underpinned by philosophical assumptions (Coleman & O'Connor, 2007). The researcher's decision on research direction is dependent on individual understanding of a variety of influential concepts, related information, and research skills (see Figure 1). One of these influential concepts is the research philosophy and underlying theory (Pathirage et al., 2007). It is important to explore the theory behind research philosophy types, especially the ones that have received relatively more attention in literature (Bryman, 2001). Positivism and interpretivism are predominant in this regard. The aim of exploring theories behind research philosophy types is to show the relevance of research philosophies in the research process and how it can influence the direction of research.

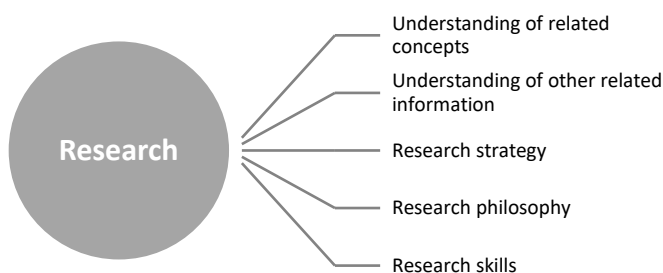


Figure 1: An illustration of typical factors affecting direction of a research project.

One of the essentials, when it comes to undertaking any research is the possession of an understanding of one’s pre-existing values, assumptions, beliefs and perspectives(Cumming, 2012). Scientific paradigms are normally condensed into the research methodology (Guba et al., 1994). It is shaped or influenced by education, life events, individual experiences, and beliefs about existence, tradition and culture (Somekh & Lewin, 2005). Altogether, these elements form the basis of correlated reasoning which informs how one views research, approaches research, ‘does’ research, organizes research activities, or engages in research. This is also known as research philosophy. The relationship between the base elements of correlated reasoning is illustrated in Figure 2. This type of reasoning influenced by the set of pre-existing elements mentioned above has some inherent shortcomings (Walliman, 2005). Shortcomings include quickly drawn conclusions, ill-tested or inadequately tested knowledgebase such as ‘common sense’.

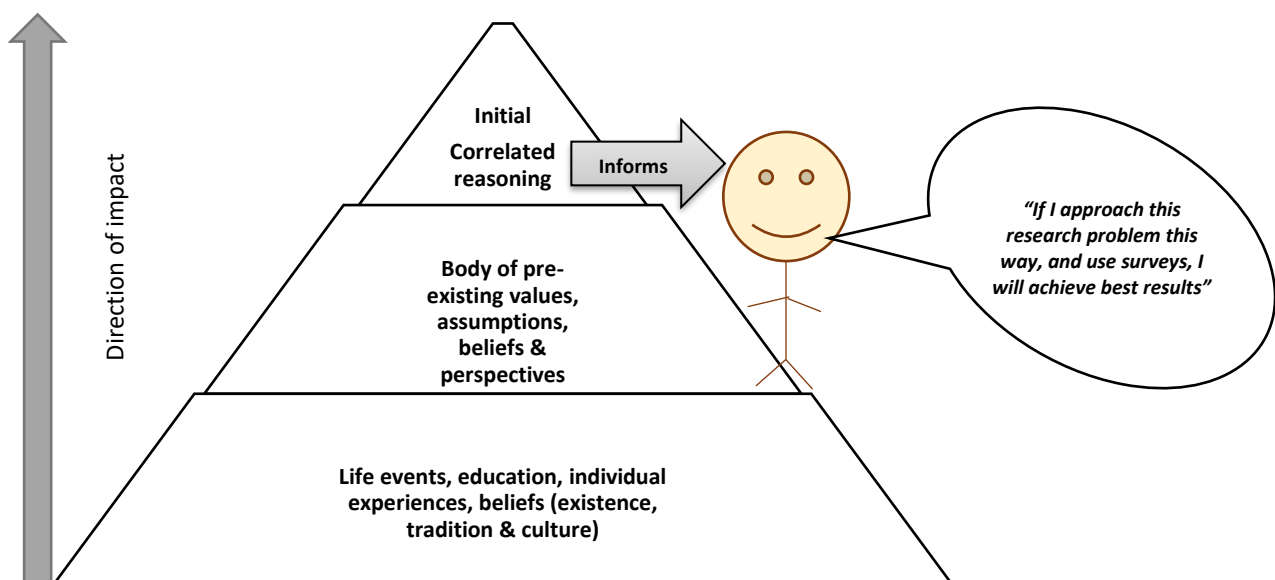


Figure 2: Highlighting “what” influences a researcher’s initial approach towards research.

Notwithstanding these shortcomings, the base element of correlated reasoning constitutes a starting point for reasoning about research. Hence the reason for viewing research sometimes, from the point of trying to study and understand something (explorative); or trying to solve a problem (formulative); or gaining familiarity with a concept or phenomenon, or generating new insights(Kothari, 2004). Isaeva et al.(2015) suggest the use of reflection on one’s beliefs, values and assumptions as a tool to better informing and developing one’s understanding of the relationship between one’s beliefs and one’s approach. This translates into a more informed approach to designing and undertaking research. This approach

improves on the view that a lack of self-reflection could lead to a problematic reduction in the likelihood of a researcher conceiving viable approaches to undertake research(Sliffe, 1998). The various ways of viewing research could result in diverse research definition, classification, and approach continuum. One of such ways is the research process. Figure 3 is a useful conceptual aid and guide when designing an appropriate research process. Figure 3 illustrates the various stages and progression involved in the research process, starting from the outer layer, and working inwards towards the inner core. Also, it provides insight into available choices within the research process. An awareness of these choices helps to provide a vantage point for better understanding of research and guidance for making appropriate choice for the research project - from choosing a research philosophy, to making a choice of research method that best encapsulate the strategy adopted to optimize the research, and the choice of an adequate method of data collection for a defined time horizon(Bryman, 2012). Making the right choices enables an easier path towards ensuring adherence to good practice in conducting the research, and ensuing review and analysis of evidence base, towards a better understanding of the phenomenon under study. The overall validity and reliability of the research work undertaken is significantly influenced by the choice of methods of data collection. Research may also be conceived from the perspective of the research approach (inductive, deductive, etc.) undertaken.

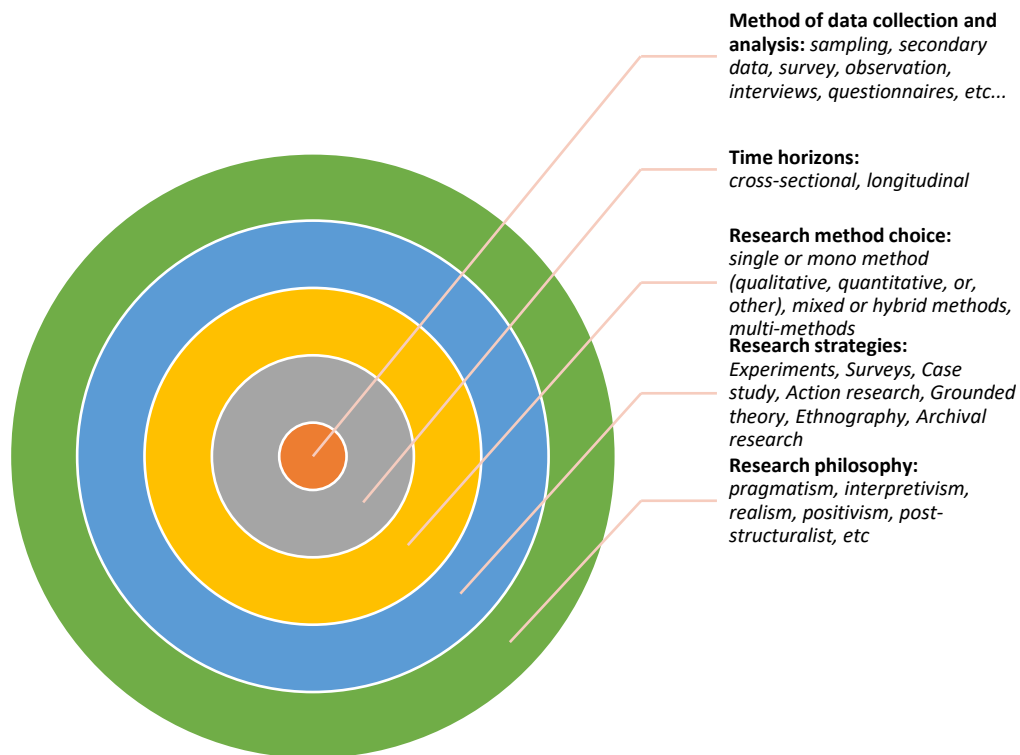


Figure 3: Adapted approach used for design of appropriate research methodology for this project (Easterby-Smith et al., 2012)

2.2 Research philosophy

The way or approach a researcher adopts to examine, study, or research a phenomenon is referred to as the research philosophy (Bajpai, 2011). It relates to the source of knowledge, nature and development of knowledge and knowledge creation as valid aspects or considerations within any research. Research philosophies include - positivism, interpretivism, pragmatism, realism, and critical theories. Preference of one approach over another may be dependent on a number of factors such as: nature of research, skill or choice of the researcher and policies of the organization sponsoring or partaking in the research (Creswell, 2002). These are some of the common factors which might influence the choice of any or some part of the philosophical approach. Potential philosophical problems could arise if mixed approaches have deep ontological and epistemological significance. This is referred to as '*method slurring*' and can undermine the credibility of a research project because it violates the assumptions and principles governing the methods used within the research (Rolfe, 2006). However, method slurring ceases to be a problem when the adoption of a mixed method approach can be justified within the context of the research and deemed necessary to achieving the aims and objectives of the research (Rolfe, 2006).

Notwithstanding, the existence of many research philosophies (Eriksson & Kovalainen, 2015) adopting and applying the right approach can add a substantial amount of value to any research project. Hence, the need to review the differences, similarities, and underlying principles of the research philosophies, along with an assessment of related merits. This would be useful for evaluating the most appropriate approach for any research project.

After initial literature review, out of existing research philosophies, two main ones were explored – positivism and interpretivism, along with characteristics, strengths, weaknesses, and advantages, as well as ontological, epistemological, and methodological perspectives associated with each. These form the basic underlying principles or key concepts of any philosophical approach. When a philosophical approach is clearly defined, it enables the creation of a viable and coherent research design based on an appropriate research strategy (Eriksson & Kovalainen, 2015). Subsequent sections expand on what these approaches are, and the ontological, epistemological, and methodological perspectives associated with each. Table 1 summarises the differences between positivism and interpretivism based on defined underlying principles that describe or characterize these philosophical approaches, thereby providing a way of analysing these approaches.

2.3 Underlying principles of a research philosophy

There are five main prominent characteristics or aspects of any approach to research design (Easterby-Smith et al., 2012). The subsequent paragraphs and Fig 4 illustrate this relationship. For a viable research design, these aspects need to be considered and applied in a consistent and coherent manner because they form the basic underlying principles or key concepts of any philosophical approach (Eriksson & Kovalainen, 2015). They also help to identify the dichotomy between underlying research philosophies and approaches.

Ontology

Refers to view of the world, and assumptions of the nature of the world and reality, or object of focus. It is regarded as the nature of reality (Easterby-Smith et al., 2012). What is this reality?

Epistemology

Refers to assumptions about how best to investigate the world or reality, or the object of focus. It is viewed as the relationship that exists between a researcher and reality, i.e. how

the researcher captures or constructs reality in his or her mind(Easterby-Smith et al., 2012).
What and how can one know this reality?

Methodology

Refers to ways of grouping research techniques in order to create a coherent and consistent picture of reality or of the object of focus(Easterby-Smith et al., 2012). A research methodology tends to be more concerned with the overarching strategy or philosophical framework used to guide the research. For example, what procedure or process can one employ to acquire knowledge about this reality?

Methods

These refer to the way investigations are carried out and how data or knowledge about reality or object of focus is collected(Easterby-Smith et al., 2012). Research methods tend to be more concerned with techniques, tools and methods used in knowledge gathering. What tools and techniques can one use to acquire knowledge about this reality?

Before embarking on any research, careful thought needs to be given to choice of methods and techniques to be employed. This is dependent to an extent on how the researcher views reality, which then influences the choice of how best to investigate and understand it.

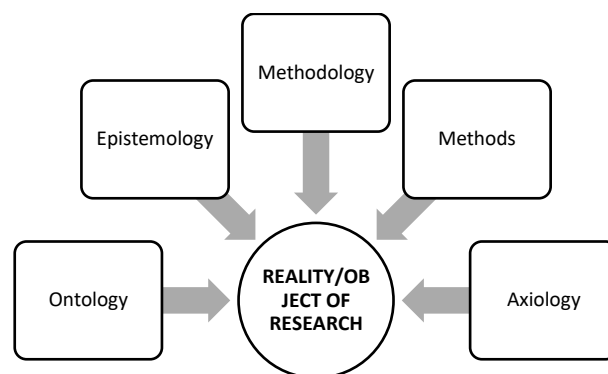


Figure 4: Characteristic principles underlying research philosophy and design.

2.4 Positivism vs Interpretivism

Positivism takes a scientific approach to studying the world or object of focus and promotes the use of formal logical reasoning methods(Thanh & Thanh, 2015). The fundamental belief in positivism is that the world or the object of focus is real and capable of independent existence. It makes use of scientific methods and tools in a bid to understand the nature or makeup of the phenomenon. It seeks to understand cause and effect in a scientific manner.

For example, this might entail developing a hypothesis, testing and evaluating the hypothesis using scientific tools and methods (Denscombe, 2010). The methods used in this approach are usually measurable and organized. An example of a common method used for obtaining information about a phenomenon using this approach is the use of observational techniques.

Interpretivism approaches the study of a phenomenon by looking at the various interpretations of that phenomenon by the world around it (Thanh & Thanh, 2015). This can incorporate a lot more subset factors than the positivist approach. In this approach, the researcher views the world or the object of focus through the lens of others' experiences and perspectives about said object (Schwartz-Shea & Yanow, 2011). From the perspectives and experiences gathered, a researcher can make his own interpretations and constructions. One of the benefits of this approach is that it allows the comparison and accommodation of multiple perspectives and contexts. This impact is felt in the degree of comprehensiveness and robustness of the data gathered. The methods used in this approach tend towards qualitative data collection techniques (Willis, 2007). They are usually less structured than those used in positivist approach. An example of such methods is the use of unstructured interviews in data collection. In this approach, there is a tendency for a researcher to influence, and be influenced by the research they are involved with. This has an impact on the degree of bias in the research.

Table 1: Summary of differences between Positivist and Interpretivist philosophical approaches (Weber, 2004)

CHARACTERISTICS	POSITIVIST APPROACH	INTERPRETIVIST APPROACH
Ontology	Reality and researcher exist as independent entities	Reality and researcher exist as interrelated entities
Epistemology	Objective reality exists beyond the human mind.	Knowledge of the object of focus is constructed through understanding of experiences
Methodology	Mostly objective	Mostly subjective
Method	Statistics, experiments, quasi-experiments, longitudinal methods	Case studies, interviews, hermeneutics, phenomenological, ethnographical, etc.
Object of research focus	Inherent qualities of object exist independent of the researcher	Researcher constructs his own understanding of object via experiences and perspectives
Theory of truth	There can be only one truth	Truth is dependent and relative to experiences and perspectives

Validity	Certainty: data truly measures reality.	Knowledge constructed are defensible.
Reliability	Reproducible	Awareness of bias implications arising from subjective interpretations

2.5 Adopted research philosophy and methodology for this research project

An 'objective interpretivist' stance was adopted towards the nature of knowledge for this research project and the investigation. Rationale for choice of research philosophy includes:

- Level of complexity of research area. This makes a positivist ontological perspective, inadequate and restrictive for this research project (Jespersen, 2011)
- The concept of software development process being an activity (Buhner, 2003) that is inherently collaborative (Mistrík et al., 2010) and involves a diverse set of people with a remix set of culture, skills, practices, environments, tools, and experiences(Zimmermann & Bird, 2012)
- The software development process investigation relies on elicitation, understanding and interpretation of experiences of software development teams and the reality of related projects. This is more in line with qualitative research methods and the interpretivist approach which gravitate towards a person's view, understanding and interpretation of experiences(Coleman & O'Connor, 2007).
- Evidence from literature advocates use of qualitative method-based research approaches for software engineering research(Coleman & O'Connor, 2007). Rationale for this is because such approaches provide opportunity to explore the complexity of the research problem thereby allowing more informative results.

The adopted ontological and epistemological stance underlies the research process, forming the main governing factor in the review of related literature, selection of an adequate theoretical perspective, design of adequate research questions, and choice of methodology, which in turn, guided and informed the research methods chosen. The principal method for data collection was iterative collection (Charmaz, 2013) and review of all related diverse perceptions, conducted experiments, opinions and views, as well as, standards in software development process. These were gathered from peer-reviewed literature and necessary for ensuring confirmability, credibility, dependability, and transferability. To strengthen the quality, validity, robustness, and convergence of the research project, the data gathered was

analysed using a triangulation method to generate a set of general constructs subject to bounding delimitations(Mertens & Hesse-Biber, 2012; Carter et al., 2014).

These constructs then form the basis for development of a solution concept. However, certain methods deemed positivist, are adopted with appropriate justification for the evaluation of the solution concept. The justification for this is to satisfy the need for empirical ways of validating research outcomes, credible confidence; to add rigor to the evaluation of the mutual dependency between the nature of collaborative activities and the science of software engineering methods employed in any collaborative software development project(Patton, 2002). Crotty's view (Crotty, 1998) of research terminologies representing very distinctive levels of decision-making within a research process has been adapted within the case study

2.6 Analysis of methods and justification

Though software engineering cannot be considered an inordinate discipline, it is not overly or unduly concerned with core or explicit theories. Hence, it may not be uncommon to see scenarios or research projects where combination of multidisciplinary theories, principles, models, and methods are used to generalize or explain phenomena, ideas, and contributions in each domain(de Souza & Redmiles, 2003). For instance, there is evidence of adoption and application of collaborative learning theories in supporting and analysing collaborative software development(Hazeyama et al., 2007) but not in cloud-based collaborative software development. Though these theories may not be software engineering-based theories, literary evidence (Hazeyama et al., 2007) highlights their usefulness and impact within the field of collaborative software development. Therefore, adopting theories outside of the principal research area intersection represent an area of synergy with potential for the research project. The question may arise - why undertake survey of different theories, including theories from other disciplines outside software engineering? Stol and Fitzgerald(2018) have demonstrated the efficacy of borrowed theories in expanding horizons within software engineering. However, this poses quite a challenge when it comes to developing or generating necessary and appropriate linkages between borrowed theories and the research or problem domain. This could be accomplished via either an inductive or deductive approach, or both(Folkestad, 2008). A combination of both approaches has been adopted to form the necessary and appropriate linkages between borrowed theories and this research.

The inductive part of the approach employed in this project involve observations via review of literary works of relevant theoretical concepts and standards of practice, and their applications within the research domain. A condensation of this review into brief outline or summary format makes it easier to identify and establish links between theories, the research objectives, and the findings from literature review. The aim of doing this is to make it easier to analyse processes or experiences present in theoretical concepts and their applications, for valid, reliable, and quality information and insights. This process is not without bias, but it has been known to yield effective results (Folkestad, 2008).

The deductive approach adopted for this quest entails the use of logical reasoning in determining what construes as pros and cons of the theories when considered in the light of preliminary review of the research domain. Information generated from this exercise then undergoes some generalization. The next step entails evaluation using a defined theoretical basis selection process for the research project and subsequently translated into constructs that are combined with literature findings to form theoretical basis for an appropriate overarching high-level framework for this research project. The mathematical selection process adopted is used partly for emphasis and assurance in the validity of the outcomes of this quest for a theoretical basis (Kenneth F. Hyde, 2000).

2.7 Summary

In research, philosophical aspects underlying methodology and choice of research methods are among foremost issues to be considered for good understanding of capabilities and limitations of methods to be adopted. The impact of understanding and choosing a philosophical stance can be felt when: defining path from research questions to conclusions, making decisions about research design, research strategy, data collection techniques and analysis. It also helps in creating awareness of possible issues that may affect research design. The choice of an approach to research should ideally, optimise the research, and aid in achieving the purpose of the research. Therefore, there should be a way of evaluating research to ensure that its purpose has been met. In cases, where the purpose of the research has not been met, justification should be provided. This research undertakes a hybrid approach, by employing both qualitative and quantitative methods of research. This includes the use of primary research tools such as surveys and the use of secondary research sources such as articles, journals, and whitepapers.

3 Literature review

3.1 Introduction

This Section reviews literature from software engineering domain to ascertain current trends in software development. This review examines related efforts in collaborative software development in the cloud, to better assess existing gaps and challenges. The review carried out reveal relatively more activity from industry in cloud-based collaboration than academia. The efforts were mainly in areas such as: content management, sharing and storage,, privacy and risks, but less in cloud-based collaborative software development(Oberhauser, 2013a). A structured systematic approach was adopted to ensure verifiable gap findings and laying of groundwork for synthesizing new approaches towards improving collaboration in cloud-based software development process (Zhang & Ali Babar, 2013).

Literary evidence reveal a variety of problems and factors which act as barriers to collaboration in cloud-based software development process (Serçe et al., 2011; Dafoulas et al., 2009; Lanubile, 2009; Noll et al., 2010; Zafar et al., 2018). These are broadly grouped into: geographical factors, sociocultural and linguistic factors, temporal factors, management, and process factors, infrastructure/technological factors, organizational factors, and trust. These barriers reveal a need for better and more cohesive collaboration within cloud-based software development process.

One popular recommendation for addressing some of the barriers to collaboration in software development is, leveraging benefits and opportunities offered by cloud computing paradigm(Derntl et al., 2015; Begel et al., 2012; Mistrík et al., 2010; Magdaleno et al., 2012). These barriers and common characteristics are shown in Fig. 7. Geographical barriers consist of factors that arise because of distribution in team, or, and activities(Bendas et al., 2017a). Socio-cultural and linguistic barriers refer to factors that exist due to differences in language and communication medium/approach(Magdaleno, 2010b). Temporal barriers refer to barriers that arise due to differences in time zones, which can also be linked to geographic barriers(Haig-Smith & Tanner, 2016). Management barriers include barriers that relate to differences and inefficiencies in the management and visibility of stakeholders, resources, and changes(Ghandehari & Stroulia, 2014). Process barriers include barriers that arise because of process planning, process implementation, and process monitoring(Noll et al.,

2010). Infrastructure barriers refer barriers that are technology-related and often centre around use, deployment, integration, and management of infrastructure, amongst others (Strode, 2016; Valilai & Houshmand, 2013).

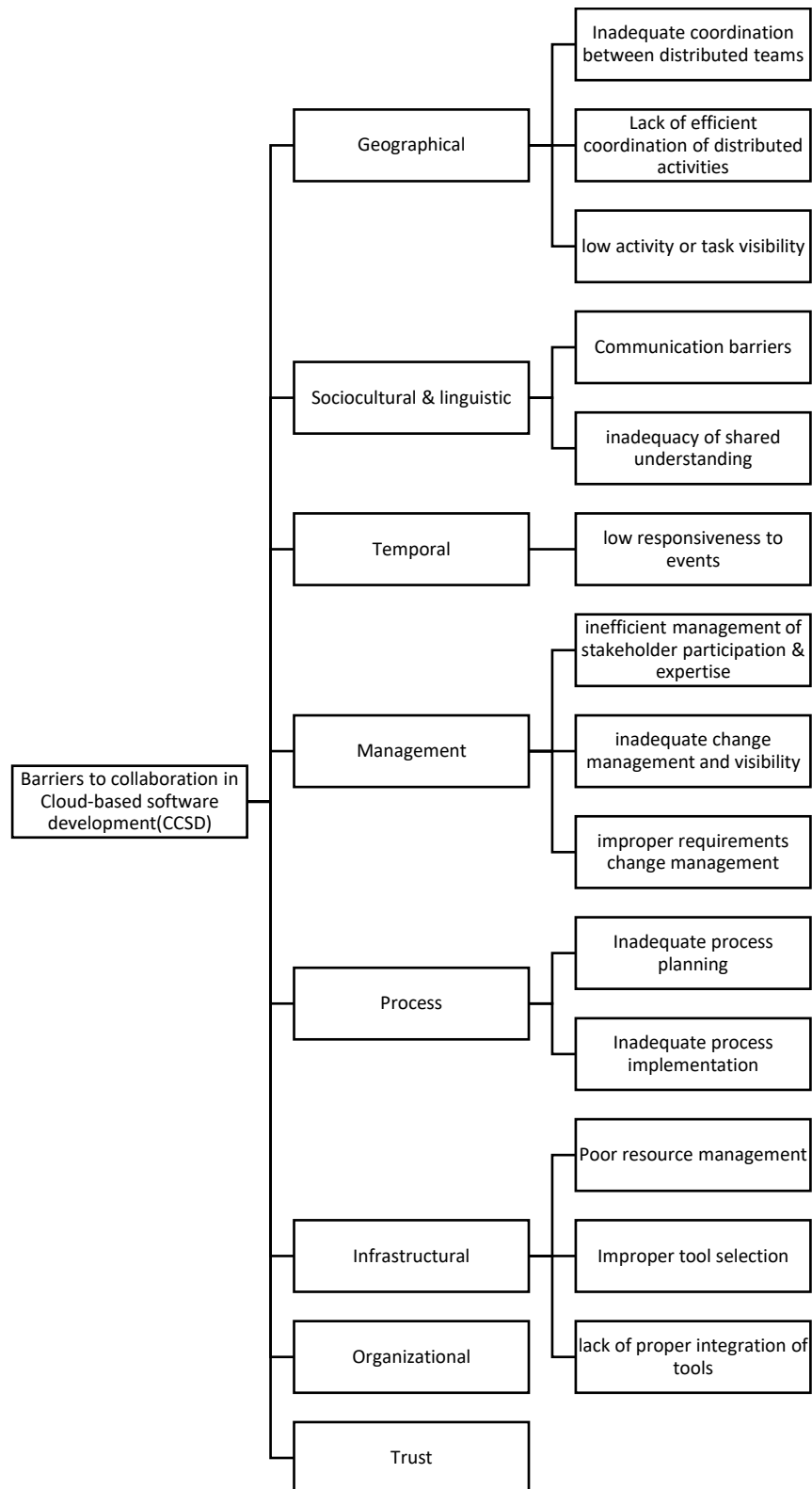


Figure 5 Barriers to collaboration in Cloud-based software development process

The prospect of leveraging cloud computing paradigm within the structured collaborative software development process presents a research area of possible synergies yet to be fully exploited (eds. Z. Mahmood & S. Saeed, 2013). The real-time collaboration and efficiency opportunities offered by the cloud, promises close-knit collaboration for cloud-based processes (Jackson, 2011; Box, 2012). Increased adoption of cloud applications and services introduce a shift in how computing resources and applications are provisioned, accessed, utilized, stored and managed (Riungu-Kalliosaari et al., 2012; Zardari & Bahsoon, 2011); and creates the need to explore and adapt collaborative software development process for the cloud (Yigitbasioglu, 2014; Ghaffari et al., 2014; Chang et al., 2013a, 2013b).

3.2 Methodology

Review of relevant literature in collaborative software development was carried out via an adapted systematic approach (Kitchenham & Charters, 2007). The review analyses existing body of knowledge in collaborative software development; related concepts that could be leveraged to enhance context-aware collaboration in software development process in the cloud. Table 2 presents the query strings used in the search and retrieval of literature for review. The search was done using Mendeley, a reference manager useful for finding, storing, managing and correlating academic research materials and libraries (Raubenheimer, 2014). Mendeley was chosen because of its reasonably fair approximation of research databases, such as Scopus. It has one of the largest databases in terms of research articles and journal coverage, and traffic (Cronin & Sugimoto, 2014).

1st tier de-duplication involved merging articles with fields where details match, or are conflicting, using the capabilities present in Mendeley (Raubenheimer, 2014). 2nd tier de-duplication involved exporting data in an xml format into Excel. In Excel, it underwent further de-duplication process by using the 'Remove Duplicates' functionality within Excel to easily identify fields that contain duplicate data. Combining these fields to form a composite set allowed further identification and removal of duplicates. This de-duped data table was then normalized, reviewed, and analysed using charts and a combination of methods involving thematic analysis (Jugder, 2016). This helped to identify gaps, challenges, issues, concepts, categories, ideas, and existing relationships and applications. This method was useful for generating themes, patterns and categories, as well as, for testing generated data against any

existing data (Grbich, 2012). Also, it is useful for a better understanding of gaps, challenges, issues, concepts, categories, ideas, and existing relationships and applications.

The chart highlights the timely relevance of this research project as can be seen from the proximity value of the coefficient of determination, R2. However, the coefficient of determination R2, does not indicate the cause of the relatively lower research effort in this research area, neither does it indicate the level of appropriateness of the chosen independent variable. This approach to literature review played an important role in definition of research themes, key dimensions, related concepts, as well as facilitating efforts towards the generation of taxonomies and ontologies (Bradley et al., 2007). The information generated was via analysis and review of data presented, in line with context, experience and understanding of authors, and this research (Basit, 2003).

Table 2: Query strings for systematic literature search

Area of literature search/topic	Query strings	Time span	Articles before de-duplication	Articles after first tier de-duplication
Extending Boehm's Software Engineering trends timeline	(title: "Software engineering trends" AND year: [2008 TO 2019]) OR (title: "trends in Software engineering" AND year: [2010 TO 2019]) OR ((title: "*Software engineering*" AND "*trends*") AND year: [2008 TO 2019])	2008 - 2019	161	97
Collaborative Software Development	((title: "distributed software development") OR (title: "collaborative software development") OR (title: "global software development")) AND (year: [2008 TO 2019])	2008 - 2019	1309	607
Collaborative Software Development in the Cloud	(((((title: "*software development*") OR (title: "*collaborative software development*") OR (title: "*software engineering*") OR (title: "*collaborative software engineering*")) AND (title: "*cloud*")) AND (year: [2008 TO 2019]))	2008 - 2019	118	76
Collaboration in Software development	(title: "*collaboration*") AND ((title: "*Software engineering*") OR (title: "*software development*") OR (title: "*cloud*")) AND (year: [2008 TO 2019])	2008 - 2019	356	277

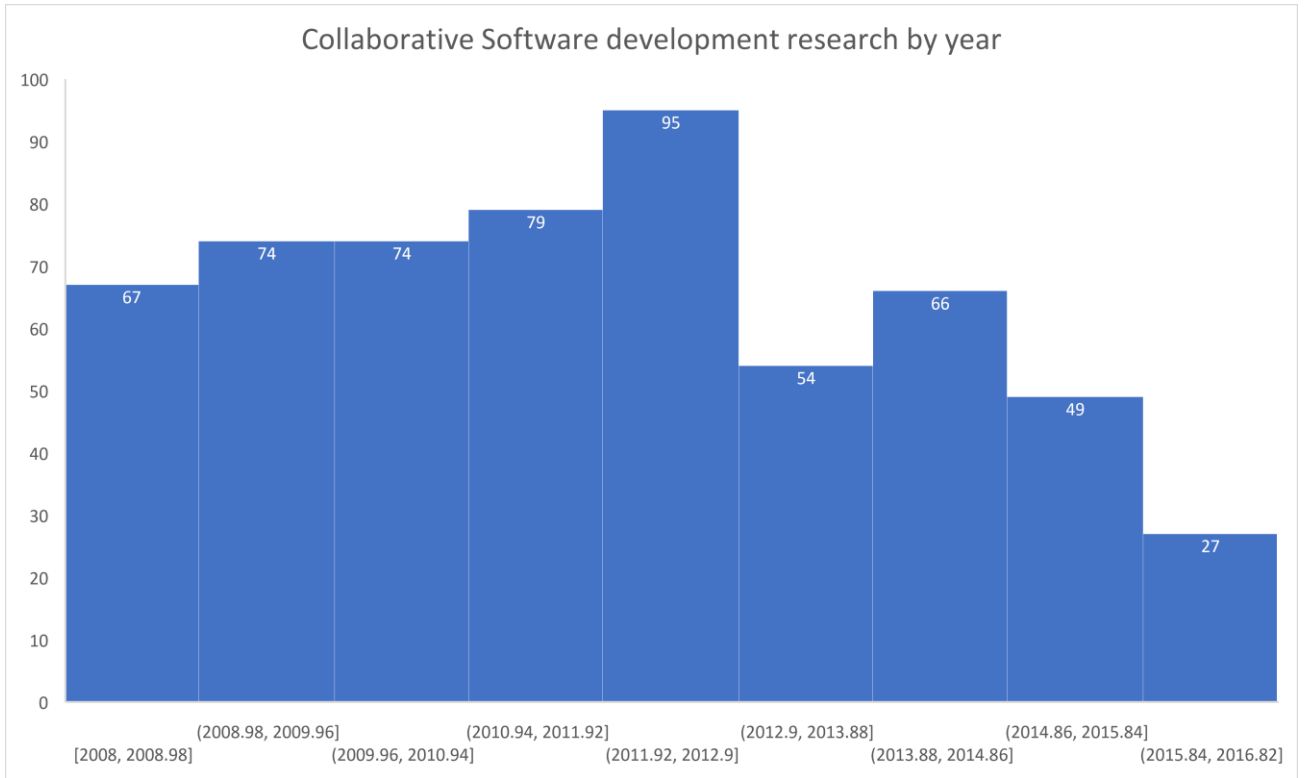


Figure 6 Decade survey of relevant collaborative software development within cloud context, grouped by year

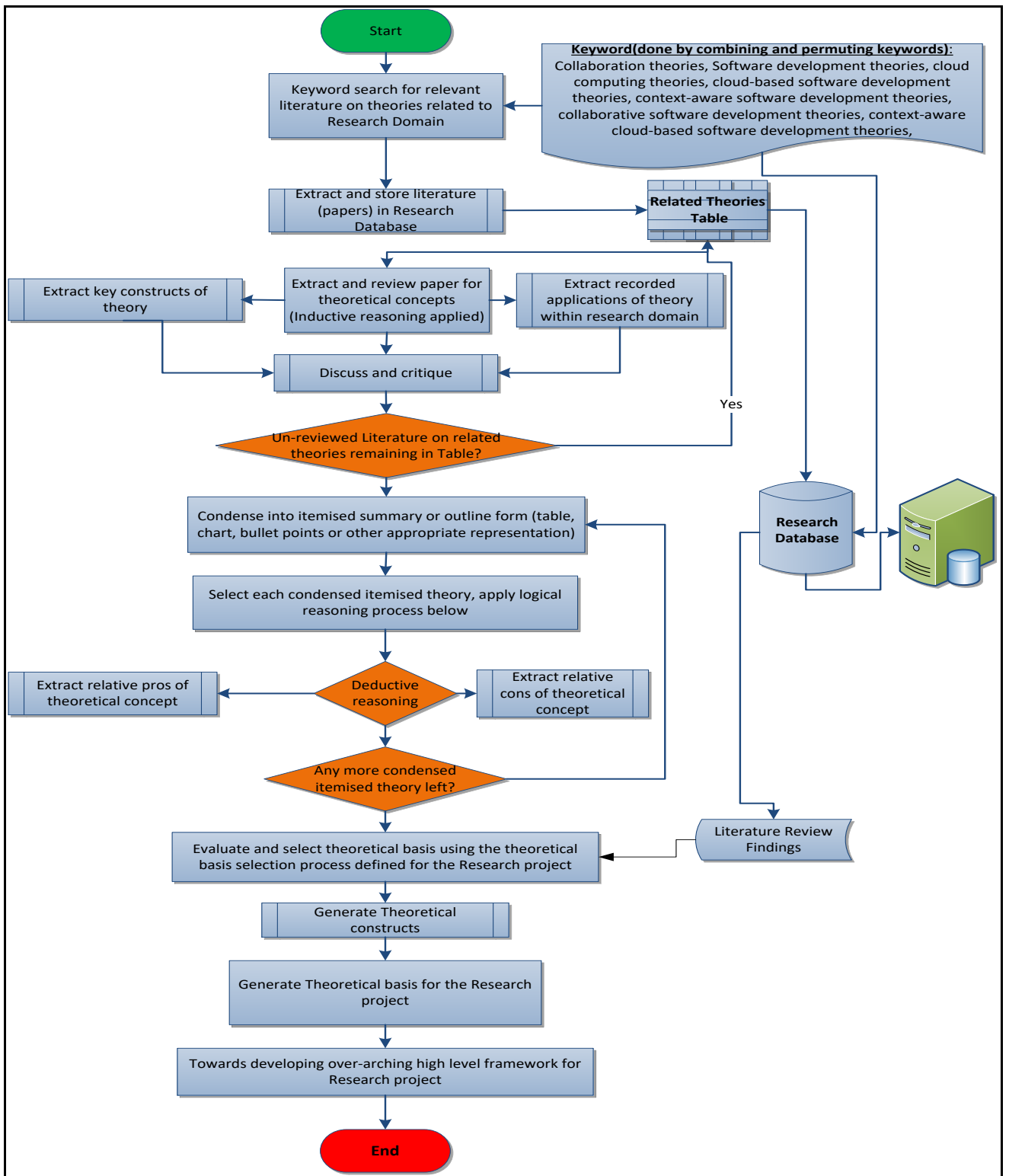


Figure 7 A flowchart representation of the adopted method

3.3 Review of software engineering trends and its relevance

Software engineering is a discipline that seeks to take away randomness in the way software is developed (Bourque et al., 2014). This is achieved by establishing and applying systematic, disciplined and procedural approaches, principles, practices, frameworks, models, and methodologies to the design, development and testing of software products and the management of the development process (Stol & Fitzgerald, 2013). A typical software engineering process involves harmonious interaction between a set of people with various skills, an environment, tangible, and intangible artefacts; towards achieving an end goal. However, factors such as constant changing needs and requirements affect interactions between different aspects of the process and ultimately the end goal. This gives rise to a constant need for adequate processes and environments that can adapt or react appropriately to changing contexts to ensure continuously meeting end goals and outcomes. The software engineering trend timeline in Figure 8 captures the current state by identifying various underlying phenomena and trends influencing evolution of software engineering practices. This timeline gives rise to predictions about future of the development process, based on observed trend pattern (Boehm, 2006a, 2010, 2006b). Verifying the veracity of these predictions and ascertaining relevance and usefulness, can be done by calibrating the prediction after reviewing the build-up to the prediction (Münch & Schmid, 2013). Calibrating predictions help in identifying current trends that were predicted and those that were not predicted. The timeline diagram reveals problems of software engineering that remain fundamentally the same. Over time, these problems have morphed into different forms identified by different labels or terminologies, and still prevail till date. These include:

- demand, growth, and diversity (issues affecting productivity, scalability, collaboration)
- software differences (issues affecting integration, interoperability, and compliance) and
- skills shortfall (technological issues)

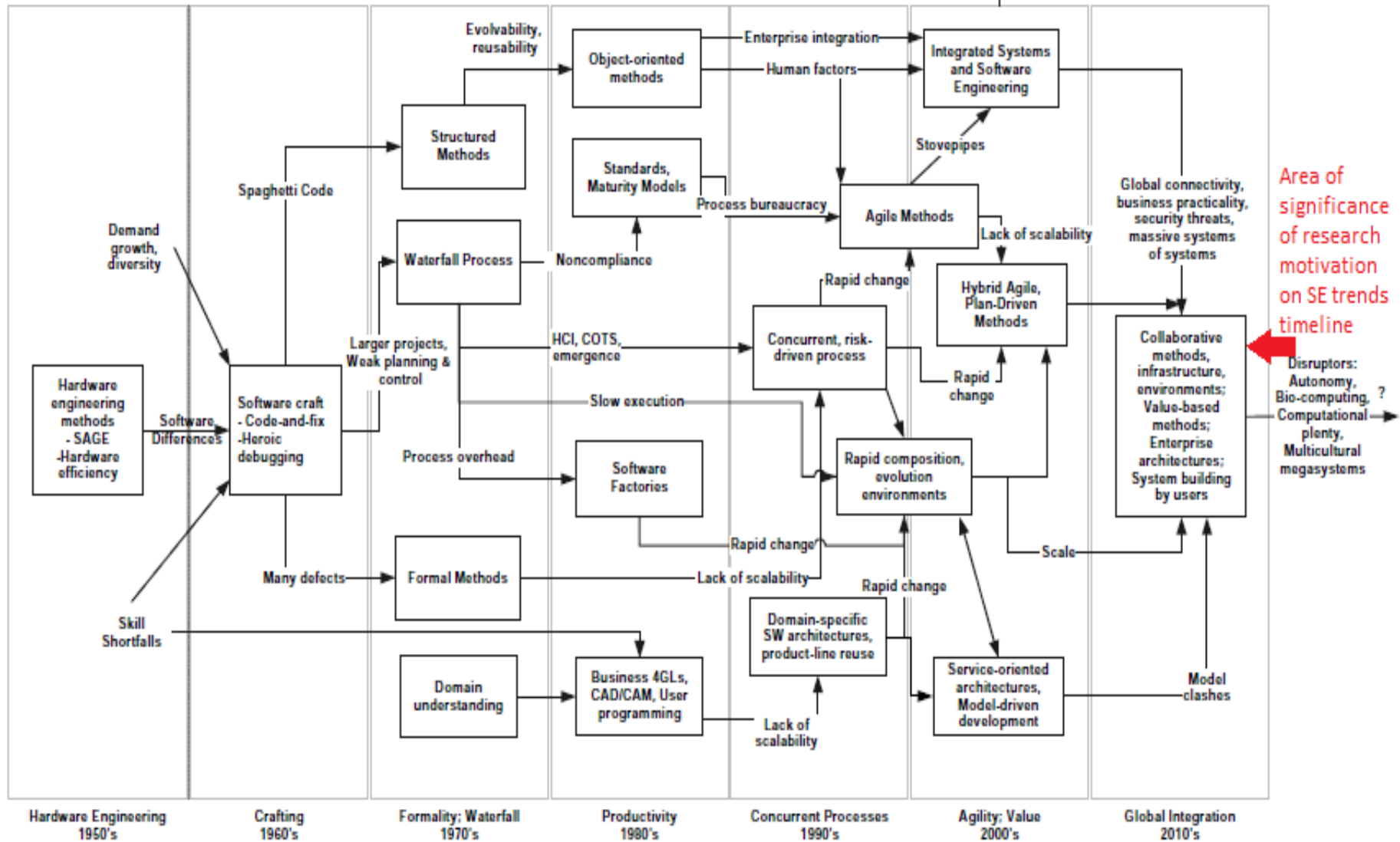


Figure 8: A timeline of software engineering trends spanning six decades (Boehm, 2006a).

However, with the advent of cloud computing, newer problems such as geographic distribution and time zone/cultural/language differences, have been added to the mix (Ulhaq et al., 2011). The former, negatively affects coordination and visibility, while the latter negatively affects communication and cooperation, inadvertently adding to complexity and barriers. The timeline also reveals trends that have contributed in ways such as continuous integration, collocation of customers, more simplistic designs, short development builds or increments, agile methods towards collaborative software development. However, the impact of these contributions has been mostly felt in small projects, but not so much in larger or distributed projects (Boehm, 2010). The trends timeline positions collaboration as a spotlight issue of this decade, because of factors such as: scale issues; clashes in models, platforms and technologies; global connectivity issues; business needs and requirements; efficiency; and security issues (Oberhauser, 2013a, 2014; Begel et al., 2012; Zimmermann & Bird, 2012; Mohtashami et al., 2009; Boehm, 2010; Jastroch, 2009; Chanda & Liu, 2015).

Investigating and developing better ways of tackling issues and challenges in collaborative software development in the cloud is not just about another trend in software engineering. It is about responding to both existing and evolving software engineering and business needs, in alignment with, available resources and technologies of the time (Nordio et al., 2011). Equally important is development and implementation of practices and context-aware mechanisms, in line with predicted future trends, likely to influence evolution and appropriate adaptation of the development process.

The timeline identifies introduction of factors such as complexity and diversity due to distribution, and differentiation at different levels. These include hardware level, the software level, cultural aspects, and software development activity phases. The identified trends impact inherent existing collaboration within the development process resulting in need to support existing collaboration and create more context-aware collaborative processes. Analysis of the trends timeline and the calibration of predicted trends highlight increasing dependence of organizations, products, services, and systems. This is indicative of the following:

- gradual trend of software-defined or software-enabled ecosystems
- need for competitive differentiation
- need for rapid adaptability to change

- need for facilitation of rapid adaptation of products to align with business and client requirements
- need for reliability and security of software-defined systems or ecosystem.
- need for cohesive collaboration between stakeholders on products/services
- need for paradigm shift from conventional to responsive to support new trends and applications(Lü et al., 2015)

Addressing these needs will entail changes to the way software is collaboratively defined, designed, developed, and deployed.

3.4 Extending Boehm's software engineering trends' diagram

Investigating and developing better ways of tackling issues and challenges in collaborative software development, is not just about another trend, but rather, about responding to both existing and evolving software engineering and business needs, and in alignment with available resources and technologies of the time (Nordio et al., 2011). Equally important is the development and implementation of practices in line with predicted future trends to allow software development processes and practices to adapt and evolve appropriately (Boehm, 2010, 2006a). An attempt is made to capture and calibrate some of Boehm's predicted related challenges of future trends, alongside current trends, and opportunities. This is to foster better understanding of considerations for planning and developing the right approach, architecture, strategy, process, and support to enhance and sustain the collaborative software development process. The trends, though a means to an end, introduce factors such as complexity and diversity because of aspects such as distribution, differentiation at different levels and aspects e.g., hardware level, the software level, cultural aspects, the software development activity phases.

The identified trends undermine and impact the inherent and existing collaboration within the collaborative software development process and result in the need for more efforts toward supporting any existing collaboration, as well as, emphasizing the need for more efforts towards enhancing and creating more context-aware collaborative processes that would be adaptive, or harder to undermine. Analysis of the trends timeline, and the calibration of the predicted trends highlighted increasing dependence of organizations, products, services, and systems. It clearly indicated:

- a gradual trend of software-defined or software-enabled ecosystem.
- need for competitive differentiation.
- need for facilitation of rapid adaptation of products to changes in business and client requirements.
- need for reliability and security of these software-defined systems or ecosystem.

Addressing these needs will entail changes to the way software is defined, designed, developed, and deployed. The trends timeline traces the path of software engineering evolution, influencing factors and trends. It culminates at the point where it highlights collaboration in software engineering as a spotlight issue of this decade. Boehm also made predictions regarding the future of software engineering in this decade. Calibrating those predictions, and building on them, with the aid of review of related literature, an attempt is made in this research to further extend the timeline.

Table 3 Software engineering trends post 2010 till date.

2010		POST 2010 PREDICTION			2010 – TILL DATE	
<i>issues & challenges</i>	<i>Trends</i>	<i>Issues & challenges</i>	<i>Predicted trends</i>	<i>Predicted trend category</i>	<i>Current trends & opportunities</i>	<i>Issues & challenges</i>
Model clashes	Enterprise Architectures	Rapid change, unpredictability, optimisation, need for enterprise integration, human factors	Increasing integration of Software Engineering & System Engineering	Surprise-free trend	More maturity models, more standards, Software-Defined systems	Compliance with document –driven requirements, requirements-deliverables mismatch, defects & bugs
Scale	System building by users	Adaptability to user, rapid change, better support, need for ambiguity-tolerance, business practicality	User, Usability & End value emphasis	Surprise-free trend	Adaptive systems, DevOps, usability enhancement techniques, enterprise support packages, data access and mining tools	Inter and intra-collaboration needs, need for value-driven metrics, context-awareness, need for value-driven metrics,

Global connectivity	Collaborative environments and infrastructure	Unforeseeable change, unanticipated software-induced catastrophes, inadequacy of current methods, processes, lack of prioritization of dependability and assurance	Software criticality, quality assurance & dependability	Surprise-free trend	Software testing, Testing-as-a-Service, CrowdTesting, Cloud-based testing	Scaling up and integration of pre-emptive development and testing, rapid change and agility, approaches increasing software vulnerabilities
Business practicality	Value-based Software Engineering and Methods	Moore's Law, increasing need for differentiation of products, global connectivity, rapid change	Rapid development and adaptability	Surprise-free trend	DevOps, hybrid mix of methods and models (Agile-driven and plan-driven), cloud development	Emerging system requirements, need for flexible integration and compatibility with legacy processes and systems, change management, rapid change
Security threats	Quality Assurance methods and security-driven development	Global connectivity, increasing need for scalability, complexity, business practicality	Globalization and interoperability	Surprise-free trend	Location-independence, mobile services, apps and devices, global distribution, pervasive and ubiquitous environments and ecosystems, outsourcing, crowdsourcing, Open-source software Development	Management visibility and control, need for shared trust and value building, efficient communication formats and semantics, need for standards application-based-infrastructure, need for effective global collaborative processes, need for bridging of cross-cultural practices, real-time change, and activity synchronization, need for sustainable collaborative processes

Massive systems of systems	Enterprise Architectures and infrastructures			Surprise-free trend		
----------------------------	--	--	--	---------------------	--	--

3.5 Software development process and models

The importance of software in business and in daily activities is evident in daily scenarios, resulting in a lot of attention and attempts been directed towards standardizing and improving the software development process. Further fuelling these attempts at improving the process are: increase in size, complexity, and distribution, involved in large and cloud-based software development projects (Kalliamvakou et al., 2015; Mistrík et al., 2010). This far exceeds what any one individual or component can handle. Hence, the need for some sort of standardized collaboration approach between diverse set of people, skills, activities, locations, tools, and environments.

Software development is a collaborative activity, involving divergent and convergent activities, carried out by people or teams, in an environment, towards achieving a set of objectives or outcome (Zimmermann & Bird, 2012). The software development process refers to the entire process of developing software, encompassing: a team, framework of activities, set of practices providing guidelines for designing, developing, testing, deploying, maintaining, and managing software. The entire process involves all the different parts working together towards an outcome. This process spans the entire development lifecycle and is usually embodied in a defined high-level abstraction commonly referred to as a software development model (Sommerville, 2010).

Software development models describe approaches for the development process that facilitate and guide activities needed to transform problem definitions and requirements into working software (Sommerville, 2010; Magdaleno et al., 2012). Various types of software development models adapted as development methodologies are essentially efforts aimed at standardizing and improving the process of developing software (Mahmood & Saeed, 2013). These process models can be broadly classified based on how linear or sequential,

iterative, incremental, responsive or collaborative the approaches are (Magdaleno et al., 2012; Mohtashami et al., 2009; Sommerville, 2010; Munassar & Govardhan, 2010).

Linear models employ an approach that steadily flows sequentially through various clearly delineated activities. Drawback to this kind of models include difficulty in response to changes after process; software is delivered at end of project making it hard to incorporate feedback on changes; lengthy durations, etc. An example is the waterfall approach.

Iterative and incremental models are a response to the drawbacks of linear models, and employ iterative activities in tandem with risk analysis or evaluation activities(Munassar & Govardhan, 2010). This is in a bid to accommodate changes. Nonetheless, these models still release software at the end of the project. An example is the spiral model(Magdaleno et al., 2012).

Agile or responsive models tend to be structured but do not focus on intensive or complete upfront planning as in the previous models(eds. J. Garbajosa et al., 2018). Agile models are more empirical in nature because changes and errors are viewed as opportunities for adaptation of software to be released. Therefore, the focus of agile models is more on individuals, interactions between individuals, changing requirements and the working condition of released software rather than on the process, tools, and documentation. Agile models tend to take relatively less development time. Examples include: Extreme programming, Scrum, test-driven development, Kanban, etc(Dybå & Dingsøy, 2008).

Some inherent similarities amongst the models include: reliance on collaborative development process and the team; accountability of team and process along lines of responsibility, roles, and functions; iteration within activities, geared towards management of change, risks and performance; design, development and testing activities geared towards achieving a common overall outcome (Lepmets & Nael, 2011). Nonetheless, when properly implemented within various activities of the software development process, any of the models has capacity to deliver quality solutions. However, some limitations still abound with existing models such as: need for mechanisms for quick assimilation, interpretation, reaction to, and application of change and review feedback from all stakeholders, across entire development lifecycle activities (Rodríguez et al., 2017).

The different models of the software development process may have distinct approaches to development of software, but ultimately, they work towards the same goal – improvement of the software development process. Depending on an organization’s software project characteristics, needs or requirements, one or more models may be selected over others. Some research efforts have been focused on tailoring of software development processes towards needs of organizations and their projects, along the lines of collaboration and discipline, while some have favoured development of hybrid models(Magdaleno, 2010b, 2010a). Table 4 summarises comparison between the most common software development models. Other considerations for choice of a model include: experience with problem domain; experience with tools and technology for implementing solutions; complexity of problem domain and solution; process maturity of team; team size & expertise, team location; regulatory and compliance requirements; need for concurrence, portability and scalability; team and organizational culture; dynamics of functional requirements and change; subjectivity of software projects to quality, cost, time, space constraints and domain knowledge; awareness; and communication (Lepmets & Nael, 2011).

Table 4 Cross-sectional comparison of software development models

CATEGORY	DIFFERENTIATING ASPECTS	SOFTWARE DEVELOPMENT MODELS		
		<i>Linear/Sequential models</i>	<i>Agile/responsive models</i>	<i>Free Open-Source Software (FOSS) models</i>
Consideration points	Underlying Philosophical objective	Seeks to establish and ensure reliability, predictability, and stability	Seeks quick ways of adding value to business, as well as adaptation to changes	Mainly seeks to ensure freedom for user
	Disciplined definition	Formal – Defined stages and activities	Formal – Agile Manifesto	Informal – works on voluntary collaboration

Development Cycles	Sequential and relatively longer	Iterative + relatively shorter + more focus on testing	Iterative + relatively shorter + more focus on testing + <i>free software</i>
Focus of development activities	Sequential processes and documentation	Customer collaboration	User participation and four freedoms – run code, study code, improve code, and distribute code
Location emphasis	Favours both co-location and geographically distributed stakeholders or team members	Emphasis on co-location	Favours both co-location and geographically distributed stakeholders or team members
Release period	Relatively less frequent	Relatively more frequent	Same as in Agile
Documentation	Relatively more documentation	Relatively less documentation	Same as in Agile
Client involvement	Relatively lower	Relatively higher	Relatively lower
Reliance on tool support for development tasks	Yes	No	Yes
Overall goal	Improvement of software development process	Improvement of software	Improvement of software

			development process	development process
	Other			
Examples		Waterfall, Unified process (e.g., as implemented in IBM's Rational)	Extreme, Scrum, Kanban, Crystal, Rapid Application Development (RAD), Lean Development methodology,	GNU, Linux, Apache, Mozilla
Typical number of activity stages	0-4	No	Yes	Yes
	5-9	Yes	No	No
Prominent challenge and issues		Less client involvement, long development times, inflexibility with management of changes in requirements, delays, and development backlog, more predictive than reactive	Less concrete planning, size of team is relatively smaller, less emphasis on documentation, can be tasking for the team in terms of time commitment, product evolution may be quite different from that envisaged, more reactive than predictive	Varies
Similarities and dissimilarities	Communication	Emphasis on formal communication	Emphasis on informal communications	Varies

	Control/Management	Approaches and implements control through structure	Approaches and implements control through flexibility	Varies
	Planning/coordination	Tends to be more upfront	Tends to be on-going as and when	Varies

Despite differences in various models, inherent similarities amongst mentioned models (Magdaleno et al., 2012; Omicini, 2013) provide rationale for considering a single iteration of the development life cycle of a software product in this research project. These similarities include: reliance on collaborative development process and the team; accountability of the team and process along the lines of responsibility, roles, and functions; iteration within the process geared towards management of change, risks and performance; design, development and testing activities geared towards achieving a common overall outcome (Lepmets & Nael, 2011).

At the beginning of any software development project, determining scope of collaboration is one of the most challenging aspects of the project. Understanding and defining what presents as core aspects and what is not is necessary and could sometimes come from stated business values, requirements, as well as from, architectures (Skerrett, 2009). This definition of core aspects underlies collaboration through stages of the development process. Stages in the development process are usually carried out via different activities grouped according to development model used, with some stages or activities overlapping (Lepmets & Nael, 2011). The stages are not always set in stone neither are the boundaries of the stages always clearly delineated or differentiated (Magdaleno et al., 2012; Munassar & Govardhan, 2010; Dybå & Dingsøyr, 2008), but the activities are usually, by consensus (Mistrík et al., 2010; Lepmets & Nael, 2011) centred around addressing questions like:

- What needs to be done – requirement gathering and analysis
- How to do what needs to be done – design
- Doing what needs to be done – coding/development
- Verifying, validating, and evaluating the solution – testing

- Deploying or handing over the solution to client or customer or user - deployment
- Ensuring that the solution remains useable and useful – maintenance

In the context of this research, collaborative software development may be defined as *a set of goal-bound actions or activities, by a group or team to satisfy the requirements of various stages across the software development life cycle within collaborative context of the specified domain*. In the case of this research, the specified domain of interest is the cloud. The need for more efficient collaboration within the process is driven by increasing distribution, complexity, and need for more efficient way of improving innovative and quality aspects of software, as well as delivery time, to meet changing needs. However, collaborative software development is yet to reach the level where the practice is routine (Chanda & Liu, 2015; Skerrett, 2009). Improving the development process necessitates standardization of collaboration between diverse set of people, skills, activities, processes, locations, tools and environments, configurations and specifications, and other relevant components or activities (Skerrett, 2009).

The collaborative software development process in the cloud comprises of divergent and convergent activities carried out by distributed team or teams. The team(s) is made up of people of diverse cultures, skillset, technical expertise, technological and non-technological viewpoints. The people either work together on different tasks or separately on complementary tasks at each stage of the process towards a common goal, all the while ensuring communication via a variety of tools or medium (Mistrík et al., 2010). This makeup, calls for efficient collaboration and management in the software development process (Zimmermann & Bird, 2012). Furthermore, the important role of software in the society and other factors such as: increase in size, complexity, and distribution involved in software development projects have generated a lot of attention, leading to efforts directed towards leveraging paradigms like cloud computing as one of the ways of improving the collaborative development process (Mistrík et al., 2010).

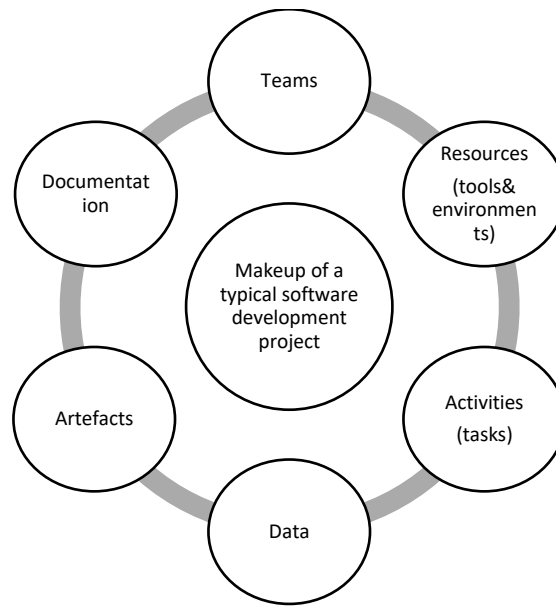


Figure 9: Typical makeup of a Software development project

3.6 Gaps and challenges facing collaborative software development in the cloud

A single person may be able to develop a piece of software, but it takes more than one person to work on large software projects. In a world where globalization is an increasingly growing trend with results such as changes in business process and operations, the need to spawn new forms of cooperation and concurrence keeps growing for domains such as software development (Chadli et al., 2016). In such distributed environments created by globalization, qualitative anecdotal evidence shows the efficacy of collaborative software development over non-collaborative or individual-based development process especially for large software projects (Weimar et al., 2017; Lindsj rn et al., 2016a; Herbsleb et al., 2005; Mistr k et al., 2010; Mahmood & Saeed, 2013). Nonetheless, this anecdotal evidence acknowledges that software development in its traditional sense, entails a degree of collaboration because it is a process made up of interdependent tasks or activities carried out by members of a team towards a pre-defined common goal.

Furthermore, experimental projects involving comparison of various software development concepts/processes such as – ‘pair programming’ versus ‘personal software processes’, emphasize the importance and advantage of a collaborative software development process (Moiz & Rizwanullah, 2012; Magdaleno et al., 2012). Related efforts in the area of collaborative software development include: open source software development movement, client-server development model, agile manifesto, global software development, integrated

systems and software engineering, and service-oriented architecture model of software development (Lepmets & Nael, 2011; Dybå & Dingsøyr, 2008; Chadli et al., 2016). These all reflect and show constant research efforts steadily directed at integrating and developing the collaborative software development process to ensure timely delivery of quality software.

The advent of cloud computing and virtualization ushered in a change in end-user environments and practices, alongside challenges and issues bordering on distribution, increased complexity, context capture/context-awareness, backward compatibility, integration, process optimisation, standardization, portability, change management, security and need for collaborative development models or processes based on sound/solid software engineering theories and principles. Amongst all the challenges identified, the most notable key challenges are: inefficient communication and coordination among distributed teams and stakeholders, and lack of effective group awareness (da Silva et al., 2010; Lanubile, 2009; Lanubile et al., 2010). Table 5 presents a summary of the gaps obtained from review of related literature in the area, publication data and trends, covering collaboration in both traditional and cloud-based collaborative software development. However, none of these efforts approached software development in the cloud from the point of enhancing collaboration that is underpinned by theoretical foundations and takes into consideration context-awareness from all participants in the process, both human and non-human.

To date, the focus of majority of R&D efforts in cloud-based software development mostly concentrate on specific aspects of the development process resulting in insufficient attention being paid to other aspects. A review of related literature reveal that related efforts towards cloud-based collaborative software development have been mainly in the areas of: trust and privacy; risks facing global software development; collaborative cloud manufacturing; asynchronous collaboration; isolated collaboration in specific aspects of the process such as coding activities; use of open-source tools for contributing, improving, and managing code; security; leveraging of social networks; collaborative cloud platform for integrating product design and contradiction analysis (Cito et al., 2015; Oberhauser, 2013a, 2014; Valilai & Houshmand, 2013; Andres et al., 2021; Liu et al., 2016; Rauch et al., 2016b). Although these efforts represent valid contributions and important enablers, they are still missing important aspects that enable a more holistic process. These aspects need to be based on solid theoretical foundation in the Cloud (Nordio et al., 2011; Oberhauser, 2014).

The concept of leveraging the cloud to create or enhance collaboration in different activities is shaping up and gaining solid ground in a lot of areas and field – manufacturing, cyber-physical systems, multimedia, learner engagement, blockchain, supply chain and contradiction analysis (Andres et al., 2021; Mourad et al., 2020; Rauch et al., 2016b; Han et al., 2013; Bendas et al., 2017a; Fisher, 2017; Barenji et al., 2021; Cancian et al., 2020; Ramis et al., 2016). Table 6 highlights some notable current reported knowledge on related work from a cross section of industry. However, it is important to state that this table is not an exhaustive list. This is attributed to reasons such as: unrecorded or unpublished work; closely guarded or undocumented industrial intellectual property (IP); experimental projects yet to be verified or validated (Cito et al., 2015; Jun & Meng, 2011; Linux Foundation, 2014). Highlighting and reviewing related work helps in highlighting gaps and emphasizing need for more research efforts (Mistrík et al., 2010; Mahmood & Saeed, 2013). Table 6 shows that leveraging the cloud for collaborative software development is a viable area that is gaining traction and being explored by industry in a bid to:

- address inefficiencies and inconsistencies of the traditional process and environment for software development
- align software development with current trends and changing business requirements
- leverage new concepts and methods for optimal development process
- address economies of scale and efficient use of resources
- enhance tighter collaboration
- facilitate more efficient management from automation and context-aware linking and sharing of information

There is growing activity from industry in cloud-based collaboration with a lot of emphasis in content management, sharing and storage, but less in the software development process. Some notable industry players, as can be seen from Table 6, have made breakthroughs in collaborative cloud-based software development. However, it is difficult to ascertain the compliance of their solutions to sound theories and principles of software engineering, even though these industry giants spare no expense at hyping the benefits and advantages of their platforms (Oberhauser, 2013a, 2014). From table 6, it can also be seen that most current solutions offered in Industry as 'cloud-based solutions' offer more support for the coding and deployment stages of the software process, and less for other stages such as the

requirements gathering stage, the testing stage, and the design stage. Some of the solutions attempt to integrate social communication by featuring some social communication tools (Begel et al., 2013; Dabbish et al., 2012; Gadea et al., 2011; Ardaiz, 2011).

Since merely developing applications compatible with the cloud does not necessarily make the applications cloud-agnostic, integrating social communication features with a cloud-based IDE does not necessarily make the development environment a collaborative cloud-based development platform. Integrating social networks in the enterprise with cloud development environments would be an approach towards enabling or enhancing collaboration in cloud development environments. But leveraging the cloud for a fully collaborative development environment in the Cloud is more than that (eds. A. Bento & A. K. Aggarwal, 2012). Table 6 present a survey of a cross-section of notable open-source tools in industry. These represent efforts towards collaborative software development process in the cloud. These have been categorized according to various emerging themes from surveyed literature. As can be seen from the table, most of the surveyed open-source development tools are cloud-based and are more collaborative in some stages of the development process than others. A good proportion of the surveyed tools are collaborative in all the stages with little or no defined metrics for specifically benchmarking collaboration in the process.

The main areas of focus for most of the tools include - continuous code quality management via inspection, analysis, and reporting on issues, bugs, or errors in code, providing interface to mash-able collection of popular development tools, and repository hosting. For example, in the case of GitHub, collaboration in the process exists in the form of team members working together via pull requests and commit actions (Kalliamvakou et al., 2014, 2015; Dabbish et al., 2012). It is sometimes difficult to figure out which projects are live, and which are abandoned. Only way of doing so is through history of commit actions, because not all pull requests are guaranteed to be accepted and merged. Another way of considering collaboration in development processes using GitHub is by considering projects in light of partial contexts such as: actions on code; who executed the actions; manual linking of related commits, comments and issues, in order to make inferences and reasoning (Arora et al., 2017). This platform used to be collaborative only in some stages of the software development lifecycle and makes provisions for using various methodologies. However, recent updates extended this collaboration across all stages. The end-to-end traceability offered by artefacts is a good

feature but there is still needed to have a full cloud-agnostic, contextualized artefact format for artefacts from all stages to allow for easy automations and implementation of automations, as well as synchronized understanding. GitHub focuses mainly on developers. Collaboration exists but mostly centred on the development, testing, and deployment of applications. Collaboration is mostly asynchronous. This is applicable to most of the surveyed tools. Collaboration is not a focal point, neither does it extend to other stages in the life cycle development process not involving code. Less focus is placed on the activity. Some of the tools seek to promote collaboration between end-users and teams via participation and incentives. They do not address underlying issues undermining collaboration such as: complexity or unified formats for output to ensure synchronized understanding. Addressing the latter could lead to developing a formal empirical way of validating that the final product meets user requirements, or the proposal of metrics for benchmarking the collaboration in the cloud-based development process.

Table 5 summary of the gaps covering collaboration in both traditional and cloud-based collaborative software development

NO.	GAP	ANALYSIS SUMMARY
1	Need for adequate theoretical foundations for cloud-based collaborative software development process	A lot of existing collaborative software development tools and platforms lack solid theoretical underpinnings and foundations. This randomizes and undermines the science behind the software development process, leaving innovative solutions to rely on results from failed implementations and glitches (Oberhauser, 2013a, 2014; Buhner, 2003; Chhabra et al., 2010; Panigrahi et al., 2017; Gill & Chana, 2012)
2	Distance or distribution-related optimisation challenges	Massive scale and distribution of teams and processes introduce challenges in: ways of communicating actions, changes and updates in a way that avoids miscommunication and decision-making delays within the cloud-based development process; methods for reducing coordination overhead while optimising development process with regards to development and testing times, timely awareness of actions, motives and changes (Bendas et al., 2017b; Gorton et al., 2016; Tell & Babar, 2012)
3	Need for efficient methods for managing complexity, synchronous regularity, and verifiable	Certain disciplines such as mechanical engineering, electrical engineering, are usually, guided, constrained and regulated by physical laws that ensure regularity and a way of keeping

	outputs/outcomes at various stages of the collaborative development process	complexity in check (Münch & Schmid, 2013). Conversely, distributed software development is not easily regulated or bound by physical laws, and so, it is not unusual to see software artefacts grow in complexity, becoming harder to understand, develop and test in the right way and correctly. This can cause reduction in productivity and delays(Münch & Schmid, 2013; Pankratus, 2010).
4	Need for efficient and adaptable techniques/methods for change management, visibility, and control	The constantly changing technology landscape and software engineering trends changes the way software is accessed, utilised, stored and maintained. It introduces consideration points such as distribution, more complexity, adaptation, and contexts. It creates a constant need to develop safe, secure, and reliable software that will continuously evolve and adapt to changing requirements, drive software engineering trends and process evolution(Boehm, 2006a, 2010). This implies the need for adequate methods and techniques for managing changes in requirements, change in the process and in the way the process adapts or learns from change (Jeffery, 2000; Zimmermann & Bird, 2012)
5	Inadequacy of conventional or traditional software development methodologies and paradigms	Recent trends have changed the way software is utilized and introduced new dimensions and levels of complexity, which have implications for the way software is currently designed, developed and deployed (Lü et al., 2015)..
6	Lack of adequate analytics mechanism for capturing actionable insights	Actionable insights could be captured from logs and feedback, from tasks, activities, interactions, executions, and transformations. These could be stored and analysed to aid improvements in management, technical, and coordinating aspects of the process. In addition, it could be used towards domain knowledge for the process, troubleshooting purposes, creation of libraries and templates, as well as improving the adaptability of the process(Gorton et al., 2016).
7	Need for standards-based environment/infrastructure	The existing standards commonly used in software development processes are quite generic in the sense that, they are mostly used for assessing and analysing how organizations follow their defined processes as well as modelling processes to monitor and control the development of software(Boehm, 2006a). It does not

		expressly cater for the analysis, assessment, and measurement of the collaborative process, not to mention the collaborative process within the cloud. The commonly used standards include: ISO 9000, CMMI, ISO 15504(OGC, 2013; Chrissis et al., 2011; Csa, 2013; Ralph, 2013b)
8	Ambiguous or missed information	Requirements, artefacts from various activities, action plans, feedback, and other important information necessary to achieve a goal are sometimes not clearly and accurately defined and agreed upon by all concerned, hence need to balance and optimise flow of information within software development teams (Mark, 2002). Furthermore, automating this flow of information can free up valuable resources such as time, reduce unnecessary noise (assumptions and discussions), and make it easier to monitor and manage - conversations, alerts, notifications, changing parameters and values, exchanges, design progress and status, and changing mission parameters, directives, and instructions(Gill & Chana, 2012). This could positively impact awareness
9	Need for culture-sensitive, collaboration-oriented, and context-aware groupware	Trends like cloud computing introduce complexity and diversity to software development which can undermine and impact the inherent and existing collaboration within software development process if not efficiently managed. The complexity and diversity are introduced via increased distribution, differentiation at various levels and aspects such as: hardware level, software level, cultural aspects, software development stages and communication/co-ordination strategies. This results in need for more efforts toward supporting existing collaboration and creating more context-aware collaborative processes that would be adaptive(Kocurova et al., 2012; Ramis et al., 2016; Valilai & Houshmand, 2013; Haig-Smith & Tanner, 2016).
10	Appropriate data capture and Knowledge management techniques to facilitate learning from historical data	Too little data is collected, or data is ignored or poorly understood. Can sometimes lead to late threat detection, identification, and resolution. It can also lead to inadequate tracking of project progress; conflicts in perspectives, understanding, interpretation and execution of activities often resulting in defective software, or software needing more

		rework(Mohtashami et al., 2011b, 2011a, 2009; Chanda & Liu, 2015; Marlowe, n.d.; Jastroch, 2009).
11	Difficulty in capturing reusable application support knowledge	This gap arises due to increase in unique complex dependencies on environment and context variables, arising from increase in distribution, software development participants, development environments and tools in cloud development environments(O’Leary, 2010).
12	Portability and interoperability of tools and other mediating artefacts	The development process could have a plethora of tools and mediating artefacts. This involves a lot of disparate technologies and data sources. Increased chances of complexity, lock-in scenarios, and compliance or interoperability issues. This can sometimes prove detrimental to collaboration within the development process(Gill & Chana, 2012; Guillén et al., 2013; Mourad et al., 2020; Karunakaran, 2013).
14	Synchronization and coordination issues	Challenges in synchronizing activities and contributions across distributed teams and sites. Short of using shared code repositories which tends to result in large and complex codebases that place builds in un-releasable states most of the time; there is a lack of readily available tools and frameworks supporting real-time synchronous collaboration in software development process. Related efforts such as GitHub, use feature branches as a way of coordinating activities and contributions amongst teams that have a lot of developers committing frequently to same code repository. However, this feature branching approach often results in unpleasant, difficult, long merging process, unsuitable for continuous development, testing, integration, and deployment (Boehm, 2010).
15	Need to bridge gap between development teams, users, and other stakeholders	Since the success of a software project depends on fulfilment of client requirements, there is need for involvement of all distributed stakeholders in ALL the activities within a cloud-based software development project (Alvertis et al., 2016b; Franken et al., 2015). However, research shows an existing collaboration gap between development teams and end users, due to inability of end users to provide continuous feedback to the development team process during the development process rather than at the end, or at intervals(Lange et al., 2016) .

		Integrating the end user community within the development process helps to improve trust and feedback. This will help in early detection, discussion, and resolution of conceptual, design, build, testing, or deployment flaws. It also allows direct interaction; ensures that everyone involved is on the same page at any point in time; misunderstandings are minimized; the development process is sped up; and unnecessary costs are avoided(Alvertis et al., 2016b; Franken et al., 2015).
16	Inadequate log and metrics management for Cloud-based collaborative software development process	Although development teams can define operational metrics and logs in the cloud, there is still a stated need to introduce tracing and metric definition as part of the development workflow. This would provide support for improved error traceability; ease of problem resolution; and benchmarking collaborative development and testing process in the Cloud(Cito et al., 2015).

The need for processes based on sound theories brought about notable attempts at finding suitable theories within software engineering and from other disciplines, in a bid to enhance the development process (Stol & Fitzgerald, 2013; Ralph, 2013b, 2013a, 2014). One of such attempts adopted the Activity theory for the analysis and evaluation of software development environments. The development environment is a space shared by the persons or teams, involved in the collaborative development activity and indeed, necessary in one form or the other, for any collaborative activity to take place (Kats et al., 2012; Soriano Camino et al., 2008). However, the analysis and evaluation carried out was restricted to development environments, which were portrayed as external to collaboration, rather than an integral part of the collaborative activity. The analysis did not take into consideration in entirety all other aspects of the software development process. In addition, it did not take into consideration, emergence and impact of pervasive environments, such as cloud-based development environments (Oberhauser, 2013a).

Cloud-based development environments arose as a result of increased adoption of cloud computing and virtualization technologies, and contributed towards increased geographic and organizational distribution in development teams (Barcus & Montibeller, 2008; Herbsleb, 2007; Gill & Chana, 2012; Fylaktopoulos et al., 2016b; Benfenatki et al., 2014; Oberhauser, 2014). However, the adoption of cloud development environments and increased distribution in teams have introduced: differences in time zones, often leading to problems in

optimisation of software development process with regards to development and testing times; poses difficulties in terms of timely awareness and communication; increase in complexity in terms of planning and coordination (Bendas et al., 2017b; Gorton et al., 2016); massive scale and additional layers of complexity in terms of abstraction levels and related components of the development process, along with their corresponding characteristics. This is in addition to existing traditional complexity in terms of measure of proportionality of activities and tasks within the development process. This further translates into an increase in contexts i.e., information that can be used to characterize the situation of entities, within the development process. These problems give rise to need for seamless and synchronized development within distributed teams. Table 6 adds perspective to this discussion context by capturing and summarising related works from a cross-section of industry practitioners and academic researchers. Furthermore, this table assists in the examination of gaps and discrepancies. It is an attempt at ‘empiricising’ state-of-the art secondary literature review.

Table 6 A Survey of cross-section of notable open-source industry tools/platforms towards Cloud-based SDLC process

Differentiation themes	GitHub	CloudTeams	Sonarqube	Atlassian Confluence. Jira	IBM jazz/CLM	CollabNet/ TeamForge	Heroku
Cloud-based/Cloud-hosted/non-Cloud	Cloud-hosted	Cloud-based	Cloud-hosted	Partially Cloud-based	Cloud-hosted	Cloud-based	Cloud-based
Explicit activity-themed, theoretical basis for architecture for Cloud-based collaborative software development process	None / Indeterminate	Indeterminate	None / Indeterminate	None / Indeterminate	None / Indeterminate	None / Indeterminate	None / Indeterminate
Implicitly associated theories	Social Network Graph	None / Indeterminate	Cognitive Complexity	None / Indeterminate	None / Indeterminate	None / Indeterminate	None / Indeterminate

Cloud-agnostic, contextualized artefact format for artefacts from all stages	partial	partial	partial	Partial	partial	None/Indeterminate	partial
Collaboration in all SDLC stages	partial	partial	partial	Yes	Yes	Yes	partial
Formal testing across all stages (Validation & verification)	partial	partial	partial	Partial	Yes	partial	partial
Metrics/bench marks	Yes	partial	partial	Yes	Yes	Yes	Yes
Metrics/bench marks for analyzing/measuring collaboration within entire development process	partial	None/Indeterminate	None/Indeterminate	None/Indeterminate	partial	None/Indeterminate	None/Indeterminate
Traceability	Yes	None / Indeterminate	partial	Yes	Yes	Yes	partial
Awareness	partial	partial	partial	partial	partial	partial	partial
Co-ordination	Yes	Yes	partial	Yes	Yes	Yes	Yes
Communication	Yes	Yes	partial	Yes	Yes	Yes	Yes
Shared Workspace	Yes	partial	Yes	Yes	Yes	Yes	Yes
Shared memory	Yes	partial	Yes	Yes	Yes	Yes	Yes
Context-awareness	partial	Partial	partial	Partial	Yes	partial	partial

Main features	Code repository, Developer profiles, dedicated project pages, code-related actions (Commits, forks, pull requests), subscription actions, version control, documentation	Customizable platform, allows 'mashable' endpoint connection of development tools, interface to allow anonymous end-user engagement with development teams in early stages	Java-code analysis engine, metrics & issue detectors, GUI Dashboard with drill-down features, Plug-in extension capabilities	Cloud Platform, UI Modules, Webhooks, Rest API, device drivers, plugin managers, network abstractions and generic services	Integrated set of tools developed on IBM Jazz platform, web-based interface, extension capabilities;	Integrated toolchain combining open-source tools for end-to-end application development & testing. Include: Eclipse, Git, Subversion, Jenkins, Visual Studio, Atlassian Jira, JFrog	Managed Containers, Heroku Pipelines, built-in monitoring tools, extension capabilities, GitHub integration, single point dashboard for managing teams & processes, API
----------------------	--	--	--	--	--	---	---

Moving on from the development environments onto the development process itself, the various aspects of the cloud-based development process i.e., environment, activities, actors, requirements, and outputs, provide a rich source of contexts. These contexts are sometimes ignored, misunderstood, or missed during the process (Mistrik et al., 2016; Mistrík et al., 2010) Some impact of this includes - lack of efficient traceability and appropriate links between activities, tasks, and artefacts from various stages or phases of the development process. This undermines the links between requirements and the developed software, weakens the analysis, reporting and tracking of change and change impact within phases, as well as making it harder to debug and fix error in time. Results of this include missed deadlines, inadequate solutions, longer time to market, a lot of bugs and defects requiring fixes or rework. Tackling these issues would help to improve awareness, thereby enhancing collaboration within the software development process.

Alongside the need to enhance collaboration, is the need to develop new metrics to measure collaboration through assessing quality of outputs, processes, exchanged information and teams, and through dynamic tracking of the processes. Current metrics have not kept up with

the evolution of software development process in line with technological advancements, therefore leading to gaps between the practical models and available data needed to validate the process, or, and output according to business requirements (Concas et al., 2011). And with cloud adoption, the need to develop adequate metrics for measuring effective collaboration in cloud-based software development becomes doubly emphasized, along with appropriate context-aware management capabilities.

Distribution has seen the rise in use of mobile devices by distributed teams working together on various projects, but still needing to stay connected whether on the move or stationary. These mobile devices have varying form factors which create the need for development of software with dynamic output that would take into consideration possible contexts of use (Baride & Dutta, 2011; Mahmood et al., 2012; Gordon, 2013; Howard et al., 2012). The challenge is not limited to form factor alone. There is a rising need for location-based services/applications, because of increased mobility and constant connectivity that comes with the use of mobile devices (Gordon, 2013). These challenges could be tackled by investigating ways to leverage the broad network access of the cloud, to ensure that developed applications are constantly accessible, and can easily update location-based characteristics as dictated by needs of mobile users (Maximilien & Campos, 2012; Begel et al., 2013, 2012).

Another pertinent issue from review of related literature in software development in the cloud is, the issue of backward compatibility and optimisation. It is not easy to convert an old or legacy application into an application that can fully exploit cloud capabilities such as scalability and auto-provisioning. Such attempts often result in increase in the number of defects and issues such as: functionality errors, interoperability, and performance issues, even security bugs. Testing legacy software, or even modern software, with the limited toolset of traditional testing practices can be quite cumbersome and does not effectively feedback into the other stages of software development in the cloud. There is need for modern methods or techniques of improving quality assurance for software and services offerings (Tilley & Parveen, 2010; Parveen & Tilley, 2010; Riungu et al., 2010; Riungu-Kalliosaari et al., 2012; Foley, 2013). Currently in the industry, there exists different development environments and platform solutions offered by various vendors, each promoting specific sets of tools. This often results in complex code and software from various

third parties that can increase chances of vendor lock-in scenarios, compliance, or interoperability issues. This also can impact collaboration in software development in the cloud (Guillén et al., 2013). Provision of context-aware functionalities to manage and audit various stages within software development in the cloud, retrieve, store, and analyse metadata for insights, could lead to improved feedback process and enable rapid defect/bug resolution(eds. Z. Mahmood & S. Saeed, 2013). Adapting software development stages, for example the testing stage, to the Cloud is an opportunity to address the shortcomings of the current/traditional software testing practices , by leveraging the strengths and opportunities of cloud computing towards delivering robust and scalable on-demand applications and services(Gao et al., 2011). Leveraging cloud computing within software development is not without risks and security issues. For example, securing the location of developed applications' data, test data depending on the degree of sensitivity, whilst also taking into considerations any legislative or privacy issues that could arise is an area of huge concern (Maximilien & Campos, 2012).

Another identified area requiring more research efforts, that could facilitate collaboration within software development in the cloud, is the development of ways for representing artefacts, data, requirements, and metadata from various stages within a software project(Hashmi, 2013). Each stage of the software development process yields some sort of output or artefact. These outputs or artefacts feed into every other stage in either: sequential, iterative, or some sort of hybrid format; and affect decision making, productivity, quality of final software and completion time. Finding a way to represent these artefacts within software development in the cloud in a common representation format that is cloud-agnostic, will play a crucial part in facilitating and enhancing collaboration in cloud-based development. This representation format could be like the use of XML for representing documents on the web, or the use of DFXML in the field of digital forensics (Garfinkel, 2011). This can ensure - proper better understanding and flow down of requirements; sharing of interoperable artefacts and metadata; effective co-ordination of development activities; facilitation of continuous and more accurate verification and validation process across stages; and reduction in the analysis/result distortion that can occur due to differences in culture, language, or disparate technologies(Hashmi, 2013). An IDG survey among more than 260 enterprises revealed that 86% of IT managers attach a high level of importance to

collaboration. Industry bodies believe cloud computing is changing collaboration by making it efficient, more effective, and faster to collaborate on projects (Box, 2012).

A classic example that further buttresses the issues discussed in the paragraphs above and emphasizes the need to evolve the development process is the report submitted by the Engineering Division within the National Defence Industrial Association (National Defense Industrial Association, 2010). This report highlighted some persistent issues which could be potentially addressed via efficient collaboration. Some of the identified issues facing existing management practices and methodologies include - little or no ability to evolve, scale and deal with changes in growth and complexities in technology, user needs and evolving paradigms, provision of adequate actionable insights commensurate with emerging development platforms. These issues restrict improvements in collaboration that could be enabled or amplified from integrating paradigms such as cloud computing with existing models.

The industrialization of the software development process emphasizes and promotes the mechanistic aspects (physical aspects, deterministic aspects – cause/effect)(Barthelme & Anderson, 2002a; Panigrahi et al., 2017), as a way of achieving standardization. Ensuing reusable components are then built from this bedrock to preserve the standardized practice, attributes, or characteristics. Since the founding process already lacks the adequate theoretical foundation, by implication, any reusable components built from this practice suffers the same lack. Hence, one of the reasons for the need for an appropriate theoretical framework that will achieve same goal of standardization, but based on sound theoretical foundations, which take into cognizance both mechanistic aspects, as well as all collaborative aspects, and changing demands or landscape(Chhabra et al., 2010). This is one of the key gaps addressed by this research project.

3.7 Impact analysis of some of the more prominent gaps and proposed recommendations

A popular belief among experts of agile software development is that typical software development teams comprise of various parts interacting, adapting, and learning within a boundary(Mistrík et al., 2010). This gives rise to complexities in the software project, including fragmentation and silo effects, amongst others. Partly to understand the various activities and complex changing parts of a software project and ecosystem, various software project teams

have either adopted, or adapted, various theories and concepts including multidisciplinary ones as well.

Theories to be considered and examined with relevance to this research project during the quest for a suitable theoretical basis, based on relevant constructs and known applications within the research area include Complexity Theory, General Systems Theory, Dynamic Systems Theory, Chaos Theory, Evolution Theory, Game Theory, Actor Network Theory, and Activity theory (Stol & Fitzgerald, 2013; Ahmedshareef et al., 2014; Fleming et al., 2013; John et al., 2013). These theories have been used and applied in a variety of areas within the software development domain. However, with respect to the questions in this research project, more emphasis and focus should be towards the context-aware and collaborative aspects of cloud-based software development activities. This necessitates an evaluation of cloud case for enhancing the collaborative software development process, to better ascertain and position necessary adjustments or adaptations for addressing the gaps and issues in the collaborative software development process (Jadeja & Modi, 2012; Armbrust et al., 2010; Lenk et al., 2009). These challenges and gaps span the entire software development process. Addressing these challenges and gaps require trade-offs which may involve the development of new architectures, approaches, and processes.

Conversely, there is growing activity from industry in Cloud-based collaboration, with a lot of emphasis in content management, sharing and storage, but relatively less in collaborative software development. Although some notable industry players like IBM, Atlassian, CollabNet, Microsoft, etc., have managed to make breakthroughs in collaborative Cloud-based software development, there is little detailed documentation available (Mistrík et al., 2010; Mahmood & Saeed, 2013). But questions exist as to the compliance of their solutions with sound theories and principles of SE, even though these industry giants spare no expense at hyping the benefits and advantages of their platforms(Oberhauser, 2014, 2013a).

3.7.1 Need for cloud-based collaborative software development architectures with explicit theoretical foundation

Emerging technologies and software engineering trends change the way software is accessed, utilized, stored, and maintained. These introduce consideration points such as: more distribution, greater complexity and increase in contexts. The result of this is a constant need to develop safe, secure, and reliable software that will continuously evolve and adapt to

changing requirements, and a constantly evolving development process. Current innovative solutions rely on results from a mix of successful and failed implementations and glitches (Oberhauser, 2013a, 2014; Buhner, 2003).

Impact

- randomness in the science of the development process
- Undermined collaboration in the software development process
- Increase in emphasis on need for better and sustainable frameworks, architectures, methods, tools, practices, and strategies, with explicit theoretical foundations to embrace and adapt to changing trends in technology, process, requirements, and related complexity, whilst still facilitating effective collaboration across the entire development process
- need for sustainable change management and self-learning methods in cloud-based collaborative software development

Proposed recommendation

Provision of explicit theoretical framework with activity underpinnings to:

- facilitate sustainable and reproducible blueprint for cloud-based context-aware, collaborative software development process
- aid understanding and conceptualisation of ways to enhance collaboration in cloud-based software development
- lay a foundation for defining processes, activities, and aligning them with goals and deliverables
- synthesize empirical knowledge to facilitate future research, development, and adaptation of collaborative models for development and testing of cloud applications
- Flag up irregularities, inconsistencies, and other factors which might impact an activity.
- reduce or eliminate randomization and reliance on results from failed implementations and glitches

3.7.2 Need for effective methods for capturing and representing contexts and other related data in a cloud-agnostic format for generation of actionable insights

Requirements, artefacts, action plans, feedback, and other important related information, necessary to achieve the defined goal are sometimes not clearly and accurately defined within the cloud-based development process. Some factors contributing to this include - poor collection methods, unsynchronized understanding, and poor application of contexts and other related metadata (Gorton et al., 2016; Chanda & Liu, 2015; Kyriakidou-Zacharoudiou, 2011; Zimmermann & Bird, 2012).

Impact

- negative impact on balancing and optimisation of information flow within development environments and teams.
- late detection and resolution of issues and bugs that could have been otherwise avoided via appropriate collection, consideration, and application of enough context data within development activities.
- inadequate tracking of project progress.
- conflicting perspectives, understanding, interpretation and execution of activities, often resulting in defective software, or software needing more rework

Proposed recommendation

Design and implementation of a common representational format for: context information, requirements, outputs from each stage of the lifecycle development process, logs, feedback, ideas, instructions, concerns, and other related data. Also recommended is the design and implementation of knowledge management mechanisms and modules for data processing, analytics, visualization, and reporting functions. This would require scalable data storage. Benefits include:

- Effective traceability, change management, better visibility and synchronized understanding and awareness
- Generation of actionable insights from: logs, feedback from tasks, activities, interactions, executions, and transformations. This would facilitate self-learning from historical data, process improvement in management, technical, and coordinating aspects

- Building up of domain knowledge for the process, troubleshooting purposes, creation of libraries and templates, as well as improving adaptability of the process
- Automation of information flow frees up valuable resources; reduces unnecessary noise (assumptions and discussions) and makes it easier to monitor and manage - conversations, alerts, notifications, changing parameters, exchanges, design progress, status, changing mission parameters, directives, and instructions.

3.7.3 Need for effective ways of managing complexity throughout cloud-based collaborative software development process

Certain disciplines such as the engineering disciplines, are usually guided, constrained, and regulated by physical laws that ensure regularity and a way of keeping complexity in check. Conversely, software engineering is not easily regulated or bound by physical laws. This makes it harder to ensure synchronous collaboration and verifiable outputs at the various stages of the process (Münch & Schmid, 2013; Pankratius, 2010; Gorton et al., 2016; Mahmood & Saeed, 2013; Mistrík et al., 2010)

Impact

- Growth in complexity of software artefacts and throughout the cloud-based development process
- Differences and difficulty in understanding, developing, and testing in the right way, and correctly.
- Increased need to challenge and validate results via some form of empirical effort

Proposed recommendation

One way to approach and reduce impact of this gap would be to limit complexity via the development of an architecture. An architecture would contribute towards managing complexity through decomposition and abstraction of main components of the cloud-based development process. Furthermore, the provision of an activity-themed and collaboration-themed theoretical foundation for the architecture would help to boost confidence in the architecture, and its sustainability. Like in the case of the engineering disciplines, this theoretical foundation can be derived from existing laws, theories, and concepts, that should be applied to guide different aspects of both the architecture and the process. Benefits include:

- Reduction of constraints impacting the ability to understand, design, develop, test, and maintain software artefacts. This helps to manage complexity and impact.
- Promotion of integrity of the process and outcomes
- Facilitation of reusability and impact analysis

3.7.4 Need for standards and adequate metrics for benchmarking cloud-based collaborative development and testing

The existing standards commonly used in software development processes are quite generic. They are mostly used for assessing and analysing how organizations follow their defined processes, as well as modelling processes to monitor and control the development of software. These standards do not expressly cater for the analysis, assessment, and measurement of the collaborative process within the cloud. Presently, the commonly used standards include ISO 9000, CMMI, ISO 15504 (Chrissis et al., 2011; Mohtashami et al., 2011b, 2011a; Bouwers, 2013).

Proposed recommendation

Introduction of suitable methods for benchmarking Cloud-based collaborative software development process to ensure monitoring and management of the process, and continuous process improvement

3.8 Summary

Research and industrial trends show a noticeable shift in the way computing resources and applications are provisioned, accessed, utilized, stored, and managed. For example, accessing applications on the desktop is giving way to accessing applications now stored in the cloud, via a web browser, API, portal, or a web application. Accessing and housing software applications in the Cloud, implies a need for change in the way these applications are engineered (Riungu et al., 2010). Furthermore, organizations are getting more distributed in terms of their distinctive groupings, processes, location, and applications (Sriram & Khajeh-Hosseini, 2010; Riungu-Kalliosaari et al., 2012). All these, call for more cohesive collaboration. Collaboration in the cloud can take varying forms, can extend across technical and social aspects of a project, and can be aimed at solving common or diverse problems encountered during the development lifecycle. Collaboration can be as simple as sharing code and designs or could take the form of real-time distributed development and testing activities across

organizational and geographic boundaries. The software development process has seen a lot of collaborative efforts over the years such as: structured processes (Agile, Scrum etc.), version control tools (e.g., Git, GitHub), client-server development, Computer-Aided Software Engineering tools, cloud-based integrated development environments, etcetera. While tools such as the above-mentioned are of immense help towards helping developers cooperate in the design and coding stage, they do little to further collaboration in other stages.

From extensive literature review, the most prevalent gaps in collaborative software development in the cloud could be surmised as follows:

- Adaptation of the software development process in the cloud based on solid theoretical foundations
- Sharing artefacts, data, metadata, support, and requirements from different stages within the development process in a timely manner.
- Scalable and flexible management of configurations and access privileges to shared data, shared artefacts, shared workspace, shared memory, and changes; across homogenous/heterogeneous teams in a distributed environment. Heterogeneity in this case could be in the makeup of team, tools, platforms, locations, or supporting components.
- enhancing awareness of group activities through designing custom mechanisms and automated workflows for features such as proactive notifications, timely updates, and intelligent predictions from data insights, artefacts, and team activities within the stages of the development process.
- facilitation of effective communication and co-ordination of activities across all stages in a distributed environment.
- Provision of a context-aware management functionality, to manage the various stages within software development by auditing the stages, retrieval, and storage of metadata for analytics and generating insights that could lead to improved feedback process and enable rapid defect/bug resolution (Mahmood & Saeed, 2013).

Overall, the literature review highlights a strong correlation between collaboration and cloud-based software development maturity. However, further analysis show collaboration within the cloud-based development to be mostly informal and unstructured and does not effectively capture militating contexts. This indicates potential opportunities for more

context-aware, structured collaborative activities within the cloud-based development process. There is also potential to significantly improve the process via context-aware, structural and process enhancements such as integration of an architecture with an activity-based process. This would provide stakeholders with a guide for adding contributions to software development projects, thereby increasing efficiency, and reducing development time.

4 Conceptual foundations

4.1 Introduction

Cloud computing is a technology trend that is changing the IT landscape and changing collaboration. One of its most notable advantage lies in its adaptability to varying contexts of use, its extensibility, as well as the numerous possibilities and opportunities it presents for all stakeholders to collaborate (Puthal et al., 2015). Stakeholders need up-to-date shared information and understanding to ensure success of a software development project. The more distributed the scenario, the greater the need for and importance of this shared understanding. Contexts have an impact on collaboration because they directly affect the quality of communication which translates into the level and quality of coordination and awareness that can be created or generated in any given scenario. Context is a continuous variable, capable of taking any infinite number of values, or pieces of information from an infinite number of variables. Currently, there are no general guidelines on how to consider or categorize contexts in software development projects (Dybå et al., 2012; Petersen & Wohlin, 2009). To get around this, this Section analyses previous related works that have attempted to provide frameworks or categorization structures that provide discrete variables offering a finite number set for providing context information (Dybå et al., 2012). This information is then synthesized and adapted for distributed teams carrying out software development process in the cloud, to ensure efficient collaboration and success of the project.

However, like most emerging paradigms, mixed feelings trail adoption of the Cloud (Ghaffari et al., 2014; Leavitt, 2009). For collaborative software development, benefits include, but are not limited to, cost savings, scalability, agility for business and development peak period needs, motivation for innovation and increased R&D (Maximilien & Campos, 2012).

On the other hand, there are fears about: security issues; vendor lock-in and interoperability issues; portability issues; automation; performance issues; availability issues; handling uncertainty about heterogeneity, content type, and location of client; bandwidth unpredictability, dynamic workload variations; varying workflow schedules; architecture and resource optimisation issues; availability and integrity of relevant information within participating teams and systems; context awareness and reproducibility within contexts; amongst others (Zhang et al., 2010; Puthal et al., 2015; Armbrust et al., 2010). Some of these

challenges and issues are partly inherited since cloud computing itself, is a paradigm that leverages a couple of other technologies (Hashemi & Bardsiri, 2009). It is necessary to analyse advantages offered by the Cloud, amongst other features and characteristics to make the case for suitability of the cloud for collaborative software development. Leveraging the cloud would require the adaptation of existing collaborative software development processes to align with the cloud capabilities. However, this is likely to raise issues for legacy applications, existing management practices and methodologies in software development projects if not done properly (Maximilien & Campos, 2012)

4.2 Cloud computing overview

One of the most adapted definitions of cloud computing is that offered by the NIST - “.... a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” (Badger et al., 2011). This definition captures the five main characteristic features of the cloud that represent the strengths from whence, most of the benefits attributed to the cloud come from. This definition also captures one key point that is sometimes overlooked - the minimal effort it entails to provision services or resources.

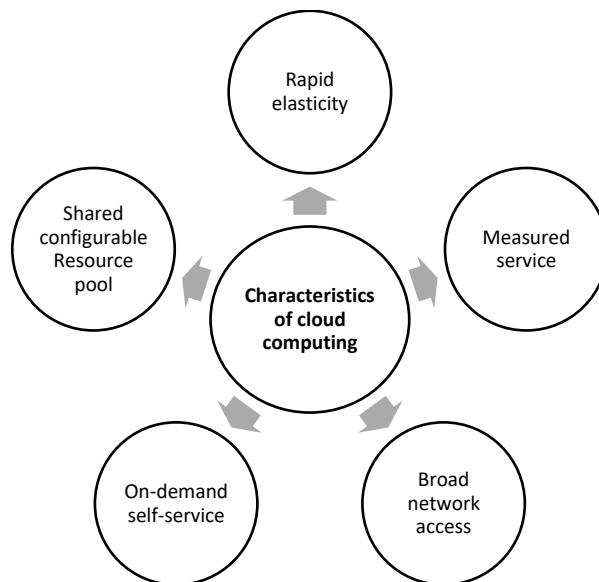


Figure 10: representation of Cloud computing characteristics based on NIST definition (Badger, Lee et al., 2012)

The advent of cloud computing has brought about an increase in the ‘servicification’ of IT resources such as: Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Platform

as a Service (PaaS); resulting in the consumption of these resources as services on a pay-per-use basis which greatly favours organizations and companies with limited resources (Armbrust et al., 2010, 2009). These services are deployed either publicly, privately, in hybrid form, or as a community model. The area of software development is not left out too. Effect of these changes can be seen in the paradigm shift from use of desktop IDEs to Cloud IDEs and Cloud APIs in building software projects. Various Cloud services providers, for example, Amazon, Google, Microsoft, IBM, and a host of others, all have their own API offerings, often built on top of their IaaS offerings (Maximilien & Campos, 2012; Doddavula et al., 2013).

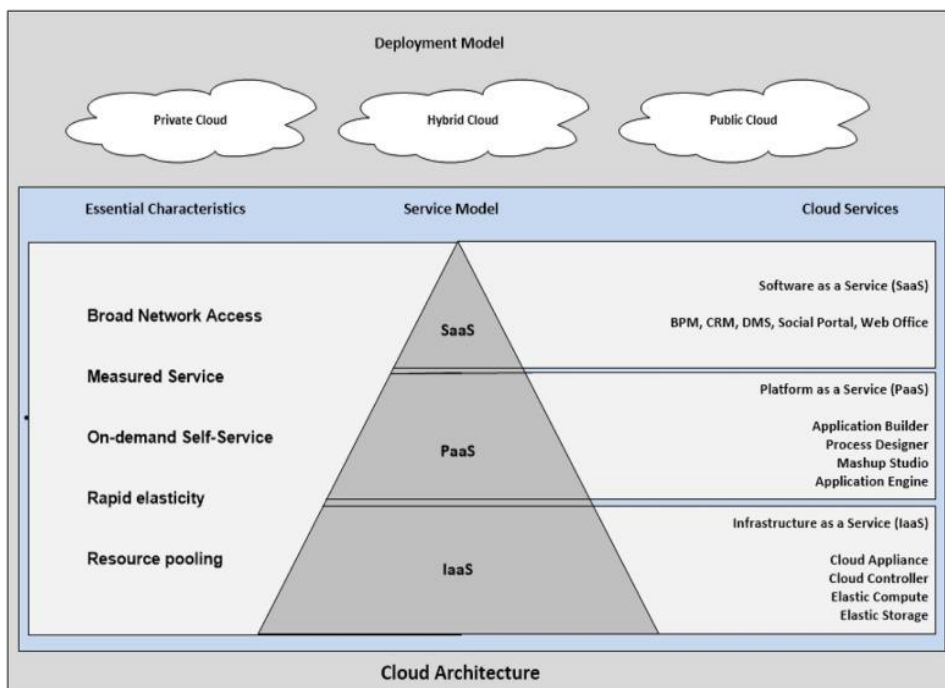


Figure 11: A view of the Cloud computing architecture (Zafar et al., 2017)

Prior to the advent of cloud computing, traditional data centres and IT setups often relied on architectures that could at best be described as like silos, making it difficult for fluid and easily scalable interactions between infrastructure, applications, and data. Situations synonymous with these include waste of resources, complex administrative and management functions, less agility, and response to changing business and user needs, high costs associated with scaling, staffing, maintenance, development, operations, maintenance, and even capital for expansion (Quest, 2012). However, the emergence of cloud computing introduces a lot of benefits, as well as open doors for countless opportunities and models of computing and business (Duraó et al., 2014). Cloud computing has become an enabler of various platforms capable of relatively higher degrees of flexibility; faster and much larger scale of computation,

processing and sharing; wider accessibility and greater availability (Warth et al., 2017). Other benefits of cloud computing include cost flexibility and efficiency; scalable resources for storage, backup, and recovery; relatively easier setting up of customized environments and quicker deployments; and a myriad of service provisioning options (Whaiduzzaman et al., 2014).

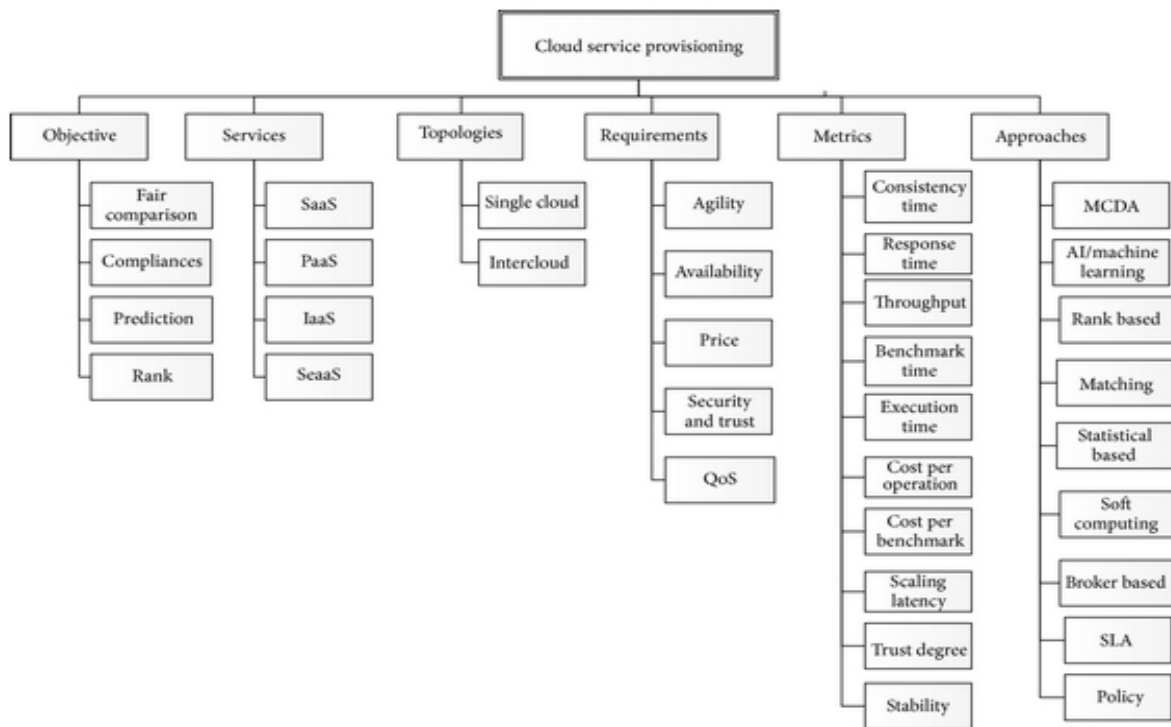


Figure 12: Service provisioning of Cloud Computing (Whaiduzzaman et al., 2014)

More efforts are directed towards exploiting and leveraging cloud computing for the range of benefits and advantages it offers, mostly as services; and this is now evident in a range of services springing up e.g., Big Data-as-a-Service, analytics-as-a-service, and a host of other service offerings in the industry (Skourletopoulos et al., 2017). However, leveraging cloud computing for more efficient collaboration in the software development process in the cloud requires a thorough understanding of what the cloud has to offer and its pitfalls.

4.3 A SWOT analysis of cloud computing

The benefits and advantages offered by the cloud makes the case for suitability of cloud for cloud-based collaborative software development. Although the cloud allows rapid provisioning of resources for rapid responsive development and provisioning of environments to enhance collaboration, adaptation of existing software development processes to align

with and take advantage of the capabilities of the Cloud is needed (Jackson, 2011; Oberhauser, 2014, 2013b). An attempt is made in this research project to summarise strengths, weaknesses, opportunities, and threats of cloud computing via SWOT analysis to highlight aspects of cloud computing that need to be critically considered and evaluated, and others that need to be further exploited for more benefits.



Figure 13: A view of Cloud challenges & Issues

Majority of R&D efforts in Collaborative software development process in the Cloud concentrates mostly on certain aspects of the process, and not paying enough attention to other factors undermining collaboration in the process such as complexity and distribution (Maximilien & Campos, 2012). In addition to providing a scalable platform for a network-based, metered utilization of elastic, shared configurable computing resources, the cloud also

provides a platform that can be leveraged for a more efficient collaborative software development process (Tsai et al., 2014; Peng et al., 2014).

Table 7: SWOT analysis of Cloud Computing

STRENGTHS (INTERNAL)	OPPORTUNITIES (EXTERNAL)
<ul style="list-style-type: none"> • Scalable and elastic infrastructures • On-demand self-service • Measured usage: pay-as-you go • Agility. Ease of resource provisioning and pooling • Broad network access • Provider assurances of over 95% availability rate • Minimal management effort • Regular and easy update 	<ul style="list-style-type: none"> • Shared resources allow for greater visibility, awareness, and collaboration • Broad network access promotes mobility and accessibility • Scalable and elastic infrastructures facilitate responsiveness • Ease of resource provisioning reduces overhead and allows redirection/optimal use of resources • Ease to setup and ease of implementation
WEAKNESSES (INTERNAL)	THREATS (EXTERNAL)
<ul style="list-style-type: none"> • Absence of universally accepted cloud interoperability standards • Requires a fast and constant internet connection for best performance • Dependency on provider, to an extent • Inability to predict peak and trough periods for resource usage • Service level agreement changes and API changes • Auditability of services/data 	<ul style="list-style-type: none"> • Legislative issues: lack of international regulatory legal precedents or framework • Security issues, privacy, and risks such as insider threat • Ownership of data and services • Scheduled and unscheduled service failures and outages

Leveraging the cloud for context-aware, collaborative software development process development is necessary for the following reasons:

- to address inefficiencies and inconsistencies of the traditional process and environment for software development.

- to align software development with current trends and changing business requirements
- to leverage new concepts and methods for optimal development process
- for economies of scale and efficient use of resources, tighter collaboration, efficient management from automation and context-aware linking and sharing of information

Anticipated benefits and impact include:

- capturing of related contexts
- better and synchronized understanding, interpretation, representation, and sharing of cloud-agnostic and interoperable artefacts and metadata
- better awareness, communication, and coordination of activities
- facilitation of continuous integration
- accurate verification and validation process across stages
- reduction in analysis or result distortion due to differences in culture, language, or disparate technologies
- enhanced collaboration and decision making
- timely resolution of bugs and issues
- better quality
- improved completion time

4.4 Collaboration overview

What is collaboration? According to the Oxford dictionary, Collaboration is “the action of working with someone to produce or create something”(Oxford Dictionaries, 2013). Collaboration is a concept spanning different context and disciplines, but is commonly used to refer to the act of working together towards a common goal (Thomson & Perry, 2006; Thomson et al., 2009a; Henneman et al., 1995). Collaboration may be in either of two forms – synchronous or asynchronous; and may be based on a variety of factors – model-based collaboration, process-based collaboration, infrastructure-based collaboration, activity-based collaboration, distance-based collaboration and inter-discipline/multi-discipline based collaboration (Lanubile et al., 2010; Lanubile, 2009; Noll et al., 2010; Whitehead, 2007). Synchronous collaboration refers to real-time collaboration, whereas asynchronous collaboration is the exact opposite. However, there have been cases of adjusted variations of

these two main forms, resulting in occurrences of semi-asynchronous or semi-synchronous forms.

Despite the numerous definitions of collaboration as a concept, it has often been misconstrued, and used quite interchangeably with other concepts or terms like: cooperation, communication, and coordination, depending on context (Camarihna-Matos & Afsarmanesh, 2008). Hence, for the purpose of this research, collaboration is used to refer to - *the set of activities involving: jointly working together to solve common problems, carrying out complementary activities to solve diverse problems, and all other activities geared towards achieving or accomplishing a common goal*(Mistrík et al., 2010). These activities could involve building and sharing knowledge; accessing shared knowledge; deriving and using insights from shared knowledge, working together in a shared space or distributed space, towards common goals. Collaboration involves more than one entity or group, working together recursively, towards common set of goals.

4.5 Key dimensions for collaboration

Though collaboration can take many forms, and may be implemented in various ways, not every form of collaboration is effective or necessary. Some forms can be quite detrimental, for example - creating more administrative and management overhead; or generating more information than useful resulting in important and necessary information being obscured or overlooked(Mistrík et al., 2010). Literature review reveals a need for clear and effective collaboration models, especially for software development in the cloud (Erickson et al., 2009). There is also emphasis on the importance of measuring collaboration to be able to “*inform practice*” for more success outcomes (Thomson et al., 2009b).

One of the most empirically comprehensive model for defining and measuring collaboration in software teams, is the Teamwork Quality model - TWQ(Lindsjørn et al., 2016b, 2018). The TWQ model shows the relationship between collaboration and the success of a software development project, as well as product quality. The earliest TWQ model was extensively tested using structural models comprising of data extracted from an experiment involving ratings from 145 software teams comprised of 575 team members, managers, and team leaders. These ratings are based on interactions on common tasks(Lindsjørn et al., 2016b, 2018). The TWQ model measures collaboration via six facets: coordination; communication; balance of member contributions; cohesion; mutual support and effort. The one drawback or

oversight from this concept is that it does not factor in data from users and software owners who also constitute part of the stakeholders in a software project. Weimar et al(2017) extended these TWQ experiments using data from 29 teams comprising of 252 team members and stakeholders with similar results, and addition of three new measurement facets for the TWQ model – value sharing, trust and coordination of expertise. These findings have been reiterated and confirmed by reports from various further experiments involving a cross-section of teams, a total of 64 agile teams consisting of 320 team members and team leaders – 33 teams in large projects, 31 teams in small projects(eds. J. Garbajosa et al., 2018).

The measurement facets from the TWQ experiments have been grouped into two categories – interaction and motivation. The interaction aspect allows benchmarking of collaboration via assessment of communication, coordination and mutual support within development teams, while the motivation aspect allows benchmarking of collaboration through assessment of effort, balance of member contributions and cohesion(eds. J. Garbajosa et al., 2018). The relative importance of the interaction aspect over the motivation aspect is directly proportional to the size of the project due to factors such as increased complexity and task uncertainty within larger team settings than in smaller team settings(Hoegl et al., 2004). Therefore, the larger the development project, the more important the role of the interaction aspect in collaboration.

Due to the validity offered by their empirical nature, results from these experiments are combined and adapted in this research, to form key dimensions. These key dimensions, based on TWQ constructs are then used as base reference points for examining, assessing, reasoning, and measuring collaboration in cloud-based software development process. These key dimensions are coordination, communication, awareness and balance of member contributions, cohesion of tasks and activities, value sharing and trust and are illustrated in figure 14 below.

4.5.1 Coordination

This refers to the management of dependencies between activities within the project. It is an approach to ensuring common understanding and agreement on tasks, schedules, and deliverables. Literature posits that coordination models could help implementation of features such as autonomy and self-organization at both component and system level,

respectively. Adapting the four goals of nature-inspired computing from the works of Omicini (2013), co-ordination could be enhanced through the following approach:

- devising what constitute sources of complexity in software development process
- understanding the mechanisms, as well as patterns which need to be exploited to enable tackling issues and unpredictability that comes with distributed or large-scale software development
- mapping or substituting these patterns and mechanisms with suitable and relevant equivalents
- Ensuring that these equivalents work with selected desirable features of the cloud, and in accordance with relevant underlying software engineering theories and principles. Including the notion of agents at this point would be a value-add consideration at this point. The agents, should ideally be autonomous software components capable of achieving tasks via interaction with the immediate environment (Ciancarini et al., 2000).

The above approach aimed at enhancing coordination will only work, if suitable models, methodologies, or techniques/technologies are used to express or re-engineer existing activities within the software development process. The result of a successful implementation of the above approach could be an intelligent, context-aware, and automated or self-organizing software development process.

4.5.2 Communication

This refers to the various ways in which stakeholders communicate about activities within a software project. Communication can be classed in a variety of ways. Current modes of communication in software development in the cloud and other web environments are considered to be ill-suited for software development in the cloud (Omicini et al., 2004). This is attributed to the direct nature of such communication forms. They sometimes lack capability of full expressiveness required to deal with issues such as dynamicity, heterogeneity, and security within the software development process. This often leads to disparaging meaning being attributed to otherwise useful information, often resulting in oversight. This accounts for some cases of failed software projects, where requirements had

been overlooked at various stages of the software development process. These key dimensions can be adapted to the collaborative software development process as highlighted in the illustration below, to improve representation and adaptation within the development process.

4.5.3 Balance of member contributions

This refers to how member stakeholders' skills and expertise can be used to achieve the activity goal.

Mutual support: refers to the ability of stakeholders to assist each other when and how needed.

Effort: refers to the relative proportion of workload undertaken or carried out by stakeholders on tasks

Cohesion: refers to the ability and tendency of stakeholders to stick together to ensure that the goal and objectives of activities are met.

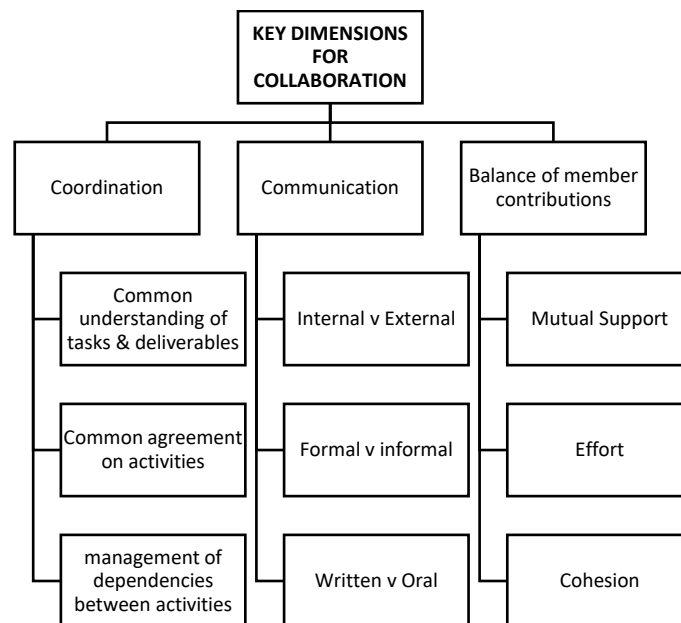


Figure 14: Key dimensions for examining and assessing collaboration needs in collaborative cloud-based software development processes

4.6 Classification of approaches for enhancing collaboration

The very nature of the software development process as a group activity requiring joint efforts geared towards a common goal implies a need for collaboration. A review and classification of collaborative approaches is necessary to foster better understanding, analysis, and

evaluation of ways to align and streamline collaborative activities. Sequel to this would be the investigation of paradigms and technologies to leverage towards addressing challenges in the collaborative software development process. To be able to identify an adequate collaborative approach for the cloud-based software development process, it is necessary to identify the components of a typical development process, related aspects and contexts that would be present. In addition, it is also necessary to identify the activities and practices involved, as well as other points of consideration. Below are different ways of analysing collaboration or collaborative approaches and activities within the software development process. Different schools of thought exist with regards to classification of collaborative work within the software development process ((Hajjdiab & Al Shaima Taleb, 2011; Nordio et al., 2011; Dabbish et al., 2012)).

4.6.1 Classification based on empirically measured activities within software development process

This is broken down into smaller categories for easier understanding of interactions and how best to support collaboration within. This classification focuses more on main actors, rather than all actors, process, and related contexts. The four classes are: mandatory collaborative activities, called collaborative activities, ad-hoc collaborative activities and individual collaborative activities (Robillard & Robillard, 2000; Clear, 2009; Mistrík et al., 2010). Mandatory collaborative activities refer to formally scheduled activities. Can be either technical or non-technical. Called collaborative activities refer to activities initiated primarily to solve a problem - mostly technical in nature. Ad-hoc collaborative activities refer to activities requiring more than one team member/process working on same task simultaneously.

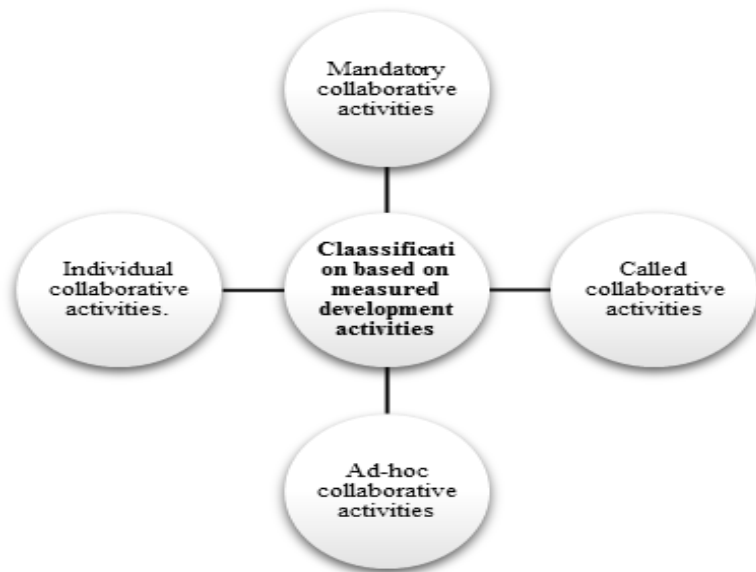


Figure 15: Classification based on empirically measured development activities

4.6.2 Classification derived from objectives of activities

The interactions between components of the software development process provides another perspective for analysing and classifying collaborative work, as well as motivation for enhancing the process (Strode, 2012; Strode et al., 2012; Magdaleno, 2010b). This classification stems from the need for effective and efficient interactions between all aspects and components of the process or activity, to ensure meeting the desired goal. As such, classification is done based on interactions according to objectives of the activity (Munassar & Govardhan, 2010; Souza, 2010). This is depicted in figure 16 below, showing a generalized view of stages in a typical software development project.

Within each stage or parent activity, smaller or sub-activities are carried out, to ensure that the objectives of the parent activity are met. If the need arises, these sub-activities are further broken down into sub-sub-activities, which are further broken down till it gets to the nth activity. This decomposition goes on and on, depending on need. Within each activity, interactions take place to achieve the desired transformation or objective. These interactions may be sequential or concurrent, subject to dependencies, and may be in any, or all the following forms: human to human; human to non-human; non-human to non-human. These interactions may involve the sharing of artefacts such as code, design specifications, requirement documentation and use cases, test scripts, test specifications, etcetera. Suffice to say, the larger the project, the more the components, the more the number of interactions,

and the more the artefacts. Hence the increase in complexity, that would need to be kept track of, and managed properly. Situations like this in any of the stages, say for example, the requirements' stage, could quickly lead to backlogs of inconsistent and ambiguous user stories or use cases. Arising results from this include inadequate or very poor-quality output, oversights, and late schedules (Hajjdiab & Al Shaima Taleb, 2011).

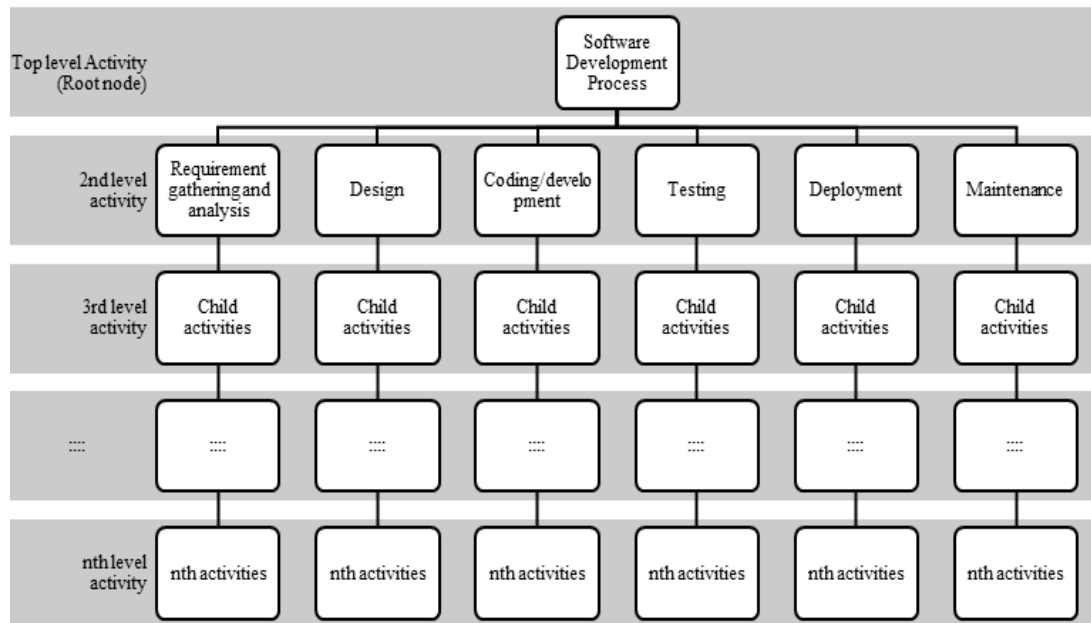


Figure 16: Classification based on objectives of development activities

4.6.3 Classification based on software development process characteristics

This classification focuses on characteristics of the process, rather than context or activity levels (Hildenbrand et al., 2008; Lanubile et al., 2010; Dafoulas et al., 2009; Serçe et al., 2011). These characteristics are grouped into distribution-based characteristics and process-based characteristics. Distribution-based characteristics include organizational, spatial, and temporal distributions.

Organizational distribution refers to distribution of the development process based on organizational units – these could be exhibiting inter-organizational or intra-organizational characteristics; or project-related characteristics – these could be inter-individual or inter-team; or business-related - these could be either company-wide or on the scale of the business unit.

Spatial distribution makes the distinction between spatial distribution and spatial collocation which can occur either within or across organizations. These characteristics can include tacit

knowledge transfer considerations, personal contact considerations, and coordination considerations, differences in time zones, and development culture and practice, along with potential impact on collaboration contexts.

Temporal distribution refers to making the distinction between synchronous and asynchronous characteristics of software development processes or activities. This includes processing of requirements, artefacts, or information. Process-based characteristics include process disciplines, process directions and process intensity. Process disciplines refer to the phases, also known as disciplines, of the software development process e.g., requirements gathering/analysis, design, development etc. Process direction encompasses collaboration which occurs either within value-creation phases of the software development process (horizontal), or the collaboration which occurs between value-creation phases (vertical). Process intensity distinguishes between higher and lower flow of information and knowledge between the actors of the process. These are referred to as higher intensity and lower intensity, respectively. The process intensity is dependent on either work done collectively, or, on collaborative exchange of information or knowledge between disjoint complementary activities.

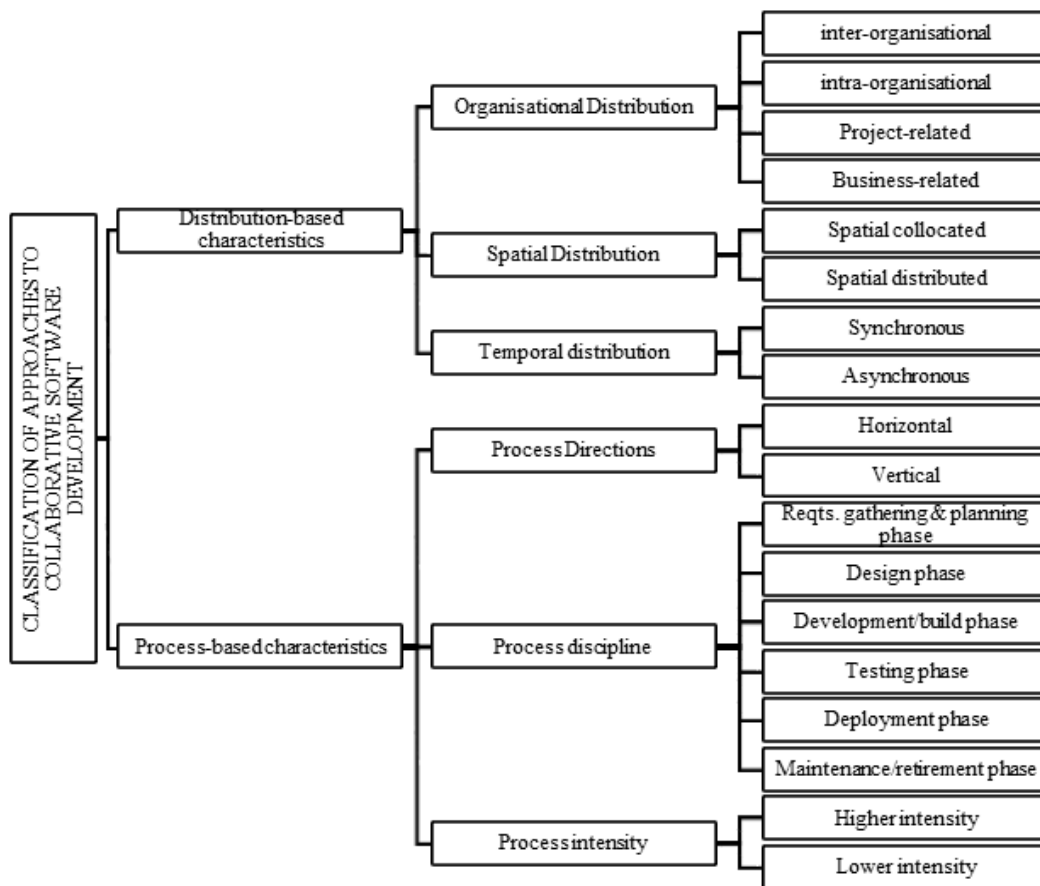


Figure 17: Classification based on both internal and external characteristics of the development process

4.6.4 Classification based on analysis of interactions between all aspects of the process

The need for more efficient collaboration within the process is driven by increasing distribution within teams, complexity within the process, and need for more efficient ways of improving quality aspects of software, as well as delivery time, to meet changing needs. However, Cloud-based collaborative software development is yet to reach the level where the practice and interactions amongst all the components of the process is routine (Skerrett, 2009; Chanda & Liu, 2015). Improving the development process necessitates standardization of collaborative interactions between diverse set of people, skills, activities, processes, tools, configurations, specifications, and other relevant components, across factors such as location, distance, characteristics, objectives, and nature of being. Analysis of interactions between all components of the process that contribute to bringing about a successful outcome, yields another basis for classification (Mistrík et al., 2010; Skerrett, 2009; Strode, 2016; Strode et al., 2012; Strode, 2012).

Enhancing collaboration in Cloud-based software development requires the ability to recognize and identify various possible collaboration contexts, as well as approaches aimed at these. When designing efficient architectures, frameworks, and methodologies for cloud-based software development process, it is important to consider both implicit and explicit differences present, as a result of varying contextual factors and characteristics. This creates a more inclusive awareness of more subtle aspects of collaboration and development contexts with potential to impact effectiveness. Furthermore, this provides a useful means for making trade-offs and selecting most apposite contingencies when seeking to leverage cloud capabilities and design solutions to improve collaboration and efficiency in software development process in the cloud.

4.7 Context awareness overview

Context-awareness refers to the ability to perceive, identify and understand contexts pertaining to a subject, or object of focus; as well as, respond or adapt accordingly, even if the contexts changes (Brézillon & Gonzalez, 2014). Any process or system with the ability to acquire contexts, process contexts, react to contexts, and utilize contextual information in adapting, customizing, or meeting needs or requirements is referred to as context-aware (Cassens & Kofod-Petersen, 2006; Vilela et al., 2016).



Figure 18: Classification based on analysis of interactions between all development process aspects

Context refers to any piece of information, or collection of pieces of information, that can be used to characterize an object or entity, or situation of an object or entity (Ntanos et al., 2014). It is an essential element that provides an entity with the ability to understand and interpret impact of surrounding occurrences, as well as “infer possible actions and information needs” (eds. T. R. Roth-Berghofer et al., 2006). Annotating an object with information that explains or comments on the object, or any aspect related to the object, provides context for the object. It can also help to track the structure or associated changes of the object, and any related relationships, thereby enabling cross-function collaboration (Goede et al., 2004).

An object or entity about which context can be collected, could be a person, tool, place, or even an abstract concept considered relevant to an interaction between the object itself, and another object (Dey, 2001). Context could vary depending on the bounding system or environment containing, or interacting with the object (Ntanos et al., 2014). Research identifies different categories of contexts, with Dewey(2009) giving the most comprehensive and encompassing categorization to aid in more explicit identification of contexts.

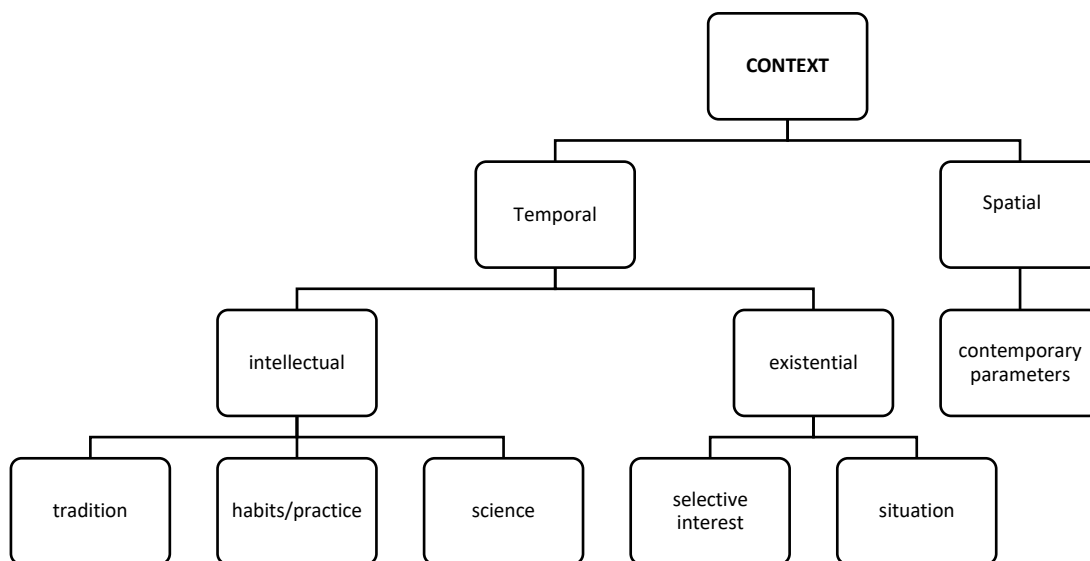


Figure 19: Classification of context

Relatively more research efforts in the area of context have been focused on technical and syntactic perspectives, than from a knowledge perspective or socio-technical perspective(Cassens & Kofod-Petersen, 2006). This research focuses on approaching context from a knowledge perspective to ensure that the user is aware of relevant knowledge capable of describing situations and impacting actions and resulting artefacts. This is geared towards facilitating and supporting effective collaboration among all stakeholders.

4.8 Relevance of context-awareness and proposed process for application of contextual information to collaborative software development process in the cloud

In the same way humans interact, share and convey ideas within a context, or create a context when doing so, same applies to teams or individuals working together on a software development project(Petersen & Wohlin, 2009). Empirical studies show that contexts help in

improving validity of software and contribute towards ensuring that developed software meets both functional and non-functional requirements via (Oh et al., 2010; Ntanos et al., 2014):

- Provision of more information about artefacts, objects, subjects, activities, and virtually anything, for more understanding, exploitation, and innovation.
- Enhancement of the validity of drawn conclusions or decisions made at each stage of the software development process, or before, or during a task, by providing intelligent insights and services (Oh et al., 2010).
- Provision of assistance in troubleshooting scenarios

Context awareness within collaborative software development process, concerns itself with the development, provisioning, and maintenance of readily available shared understanding of overall state of the development object within the process, in relation to:

- project
- team
- related activities
- tasks
- artefacts and
- resources

Some related works have implicitly considered contexts (Runeson & Höst, 2009), while others have used some elements within a project, such as study objectives, baseline, and constraints, to consider contexts for the project (Kitchenham et al., 2004; Babar & Kitchenham, 2007). Figure 20 shows one of the most apt categorizations of contexts covering six main facets – product; processes; practices, techniques, tools; people, organization, and market; with the object of focus or study at the centre, and sources of contexts round it. Each facet in turn, comprise of various context elements. It is an attempt to propose a complete checklist to cover all possible context facets, gathered from review of industrial studies of non-cloud-based development projects (Petersen & Wohlin, 2009). The limitation of this context categorization is that it excludes reviews of reports, surveys, whitepapers, and open-source experimentations, thereby creating room for extending and simplifying the structure to align with cloud-based software development. Literature emphasizes the impact of contexts on

awareness during the process, validity at each delineation, and the interdependence of component context elements, as well as the extensibility of these facets and elements (Petersen & Wohlin, 2009).

An understanding of the different contexts within existing software development process is necessary when moving existing development processes into the cloud. This is because such a move is bound to introduce more contexts into the process due to the presence of more entities, as well as any other external factors – see Fig 20 below (Dybå et al., 2012). This understanding helps in the categorization of contexts within the process, thereby raising awareness. Rationale for this line of thought comes from the fact that every context within the development process, introduces an opportunity and a possibility, to create, or enhance the effectiveness of the process (Kim et al., 2004).

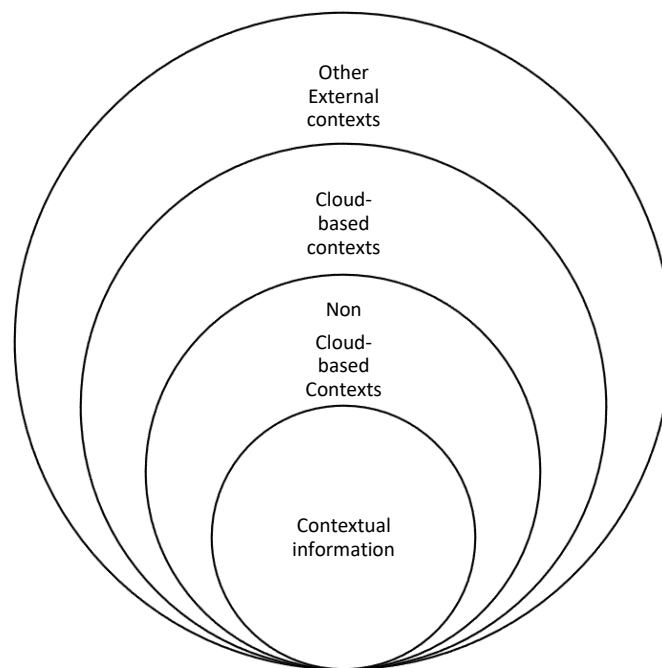


Figure 20: Understanding context-awareness requirements for cloud-based collaborative software development

The journey towards making the cloud-based software development process to be context-aware, begins with management of contextual information. This entails the following considerations (Ntanos et al., 2014):

- how to gather contextual information
- representation of contextual information
- application of contextual information to the process

- storing and retrieval of the contextual information

Challenges facing the management of contextual information includes (Ntanos et al., 2014; Dybå et al., 2012):

- a uniform representation of types of context (taxonomy)
- suitable framework that provides a means of designing, implementing, adapting, and improving context-awareness within the development process
- theoretical structure for reporting contextual information to ensure that common or recurrent context factors can be identified and collected.
- use of sensitive analytical and implementation strategies

Context management is the function within the software development process that is responsible for management and publication of context data. The impact of context data in planning and coordinating activities within the process include:

- Tracking and management of changes to artefact-related data
- less time spent in organizing and tracking artefact-related data
- Improvement in productivity through reuse of artefact-related data
- Enhancement of collaboration
- Provision of collaboration services on artefact structure management, activity management, alignment of activities with stated goal

The existing process manages data from activities via SVM systems like Git and subversion during development. However, it does not take multi-disciplinary integration into consideration to reduce the number of possible contradictions to resolve during the development process. Contradictions refer to conflicts or tensions between two or more parameters or entities (Liu et al., 2016). Contradictions may present as obstacles, but also represent potential opportunities for improvement and innovation, through the elimination of compromise (de Souza & Redmiles, 2003).

Some ways of resolving contradictions include the use of contradiction matrix for solving technical contradictions(Engeström, 2001); use of separation principles for solving physical contradictions; inventive principles; application of DFX approach in generation and application of knowledge to improve, control and invent traits for an artefact(Liu et al., 2016).

In proposing a method for context data management for cloud-based collaborative software development process, general characteristics and attributes of contexts need to be considered (Varaee et al., 2015; Fazil et al., 2010). This consideration should be with respect to artefacts from various activities within the process. This lays the groundwork for adapting the use of contradiction matrix (Engeström, 2001) for contradiction analysis and for solving technical contradictions afterwards.

Table 8: Adaptation of Zachman's framework for definition of context data and levels

Context levels or scope/Context types	What	How	Who	When	Why	Where	Other
Pre-existing	Objects important/related to the activity (Entity = class of thing). E.g., data, relationships	Actions/Tasks to be performed (function = class of action) E.g., function or operation, views	Subjects related to thing or action. E.g., people, organizational unit, roles	Remarkable events or frequency. E.g., time of event, event cycle	Goals/strategy. E.g., motivation, objective	Location of task/action. E.g., cloud node	Anything else relevant to activity/task
Emerging	Objects important/related to the activity (Entity = class of thing). E.g., data, relationships	Actions/Tasks to be performed (function = class of action) E.g., function or operation, views	Subjects related to thing or action. E.g., people, organizational unit, roles	Remarkable events or frequency. E.g., time of event, event cycle	Goals/strategy. E.g., motivation, objective	Location of task/action. E.g., cloud node	Anything else relevant to activity/task
Proposed	Objects important/related to the activity	Actions/Tasks to be performed (function =	Subjects related to thing or action.	Remarkable events or frequency. E.g., time	Goals/strategy. E.g., motivation, objective	Location of task/action.	Anything else relevant to

	(Entity = class of thing). E.g., data, relationships	class of action) E.g., function or operation, views	E.g., people, organizational unit, roles	of event, event cycle		E.g., cloud node	activity/task
--	---	--	--	-----------------------	--	------------------	---------------

Some key questions, along with an adequate representational format structure, have been adapted to provide simplified, but expandable key dimensions for collecting, structuring, and analysing contextual information for the collaborative software development process (Kitchenham et al., 2002; Broens et al., 2006; Dybå et al., 2012). The objective of the adapted questions is to collect as much contextual information as possible for the project from defined sources built around: measures, entities and attributes that carefully consider and answer the related questions in alignment with the object of focus.

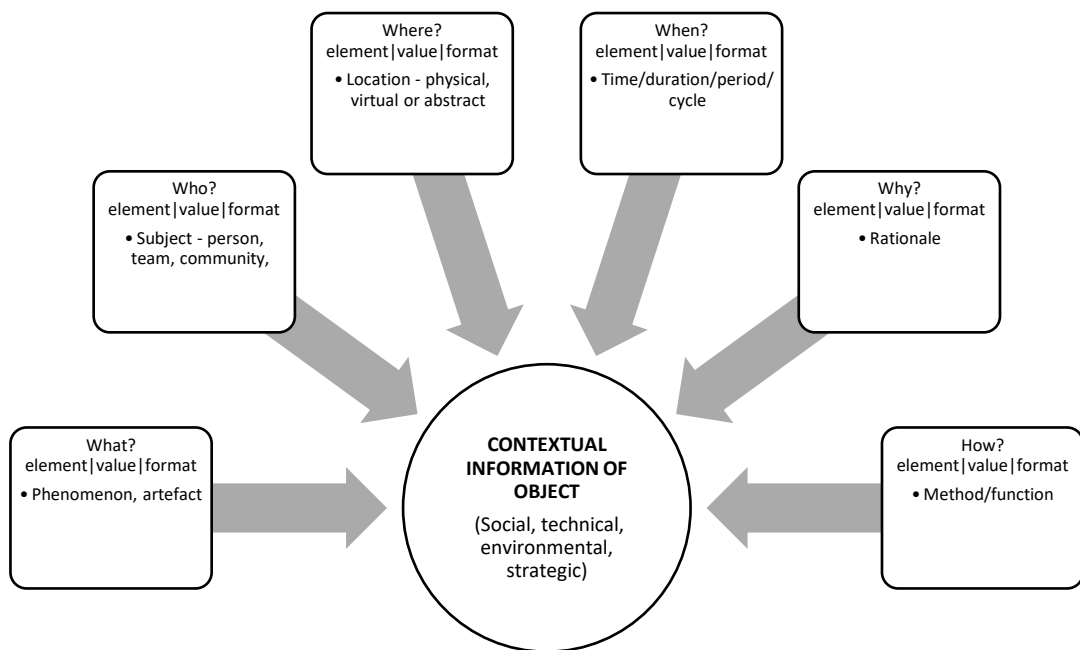


Figure 21: Adapted key dimensions for collecting, categorizing, analysing, and applying contextual information

The contextual information collated using the process shown in Figure 21, is applied to the different stages in the development process using an appropriate algorithm or process - see Figure 22.

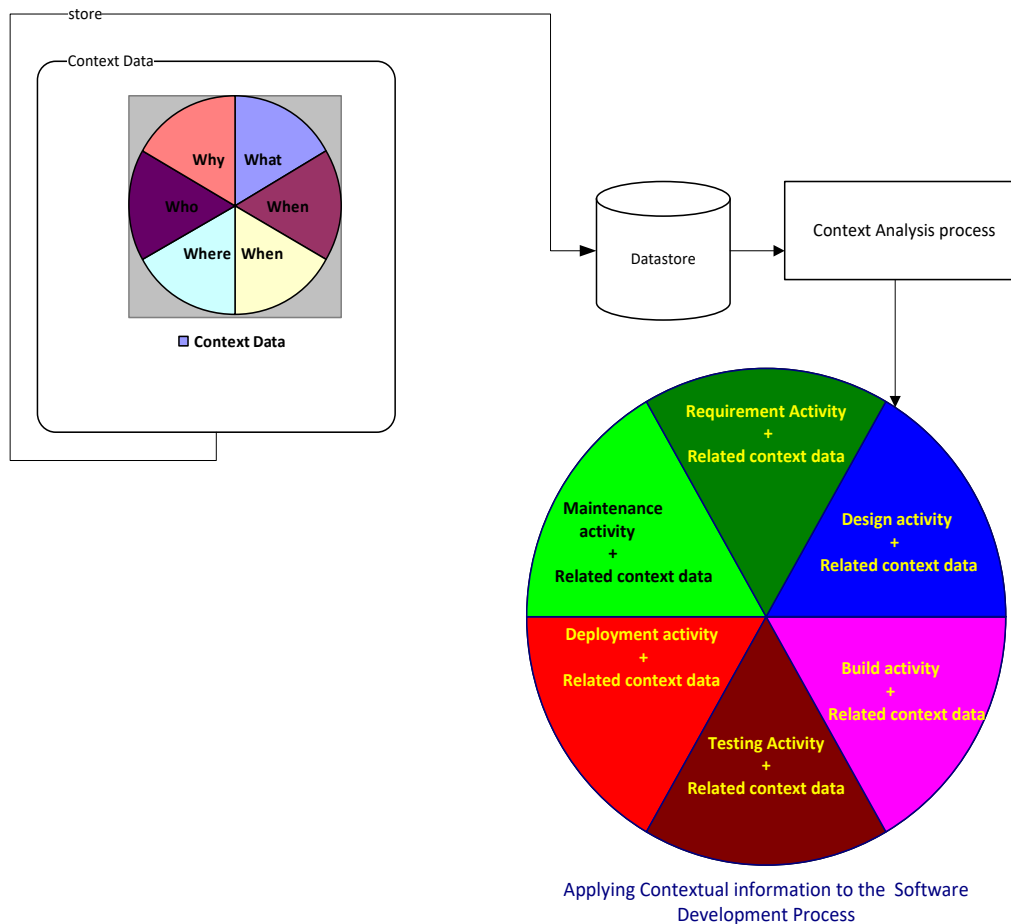


Figure 22: Applying contextual information to the Collaborative Software Development process

In this research project, matrix multiplication is employed in adapting the contradiction matrix (Engeström, 2001; Childs, 2019) for applying contextual information to different activities and tasks within the different stages in the collaborative software development process to aid analysis/technical resolution of contradictions that arise during context management. This matrix multiplication method is used as a way of further discretizing context variables and contextual information (Bini, 2013), thereby reducing complexity, and optimising analysis for improved accuracy (see Table 9, 10, and 11 below). This result can then be converted into an appropriate format as the artefact from each activity, task, or stage for use in the collaborative process. This represents all “knowable” or holistic information about each activity, task, or stage, to ensure an optimal product that meets both functional and non-functional requirements (Geszten et al., 2018). Given that number of activities or tasks or

resources for any software development project could scale up and down in line with project requirements, this translates into direct proportionality for contexts that could arise.

Given above discourse, contextual information for object of focus is represented as matrix C:

Table 9 Adapted contradiction matrix for analysis of contextual information for an object in cloud-based software development.

C =

What	Where	Who	When	Why	How
C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅	C ₁₆
C ₂₁	C ₂₂	C ₂₃	C ₂₄	C ₂₅	C ₂₆
⋮	⋮	⋮	⋮	⋮	⋮
C _{m1}	C _{m2}	C _{m3}	C _{m4}	C _{m5}	C _{m6}

Similarly, information, activities, tasks, and processes relating to the various stages within a software development process, SD is represented as the matrix, S below. This scenario takes *n* iterations of a standard waterfall process. For an agile process, or other methodology, stages can be removed, or added, as needed.

Table 10 Adapted contradiction matrix for representation of activities and tasks in cloud-based software development.

S =

Requirements stage	SD ₁₁	SD ₁₂	⋮	⋮	SD _{1n}
Design stage	SD ₂₁	SD ₂₂	⋮	⋮	SD _{2n}
Build stage	SD ₃₁	SD ₃₂	⋮	⋮	SD _{3n}
Testing stage	SD ₄₁	SD ₄₂	⋮	⋮	SD _{4n}
Deployment stage	SD ₅₁	SD ₅₂	⋮	⋮	SD _{5n}
Maintenance stage	SD ₆₁	SD ₆₂	⋮	⋮	SD _{6n}

The total information pertinent to each stage, activity, task of the collaborative software development process is represented as a product of matrices C and S.

Table 11 Employing matrix multiplication in adapting contradiction matrix for applying contextual information to activities and tasks in cloud-based software development.

C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	X	SD_{11}	SD_{12}	⋮	⋮	SD_{1n}	SD_{11}
C_{21}	C_{22}	C_{23}	C_{24}	C_{25}	C_{26}		SD_{21}	SD_{22}	⋮	⋮	SD_{2n}	SD_{21}
⋮	⋮	⋮	⋮	⋮	⋮		SD_{31}	SD_{32}	⋮	⋮	SD_{3n}	SD_{31}
C_{m1}	C_{m2}	C_{m3}	C_{m4}	C_{m5}	C_{m6}		SD_{41}	SD_{42}	⋮	⋮	SD_{4n}	SD_{41}
							SD_{51}	SD_{52}	⋮	⋮	SD_{5n}	SD_{51}
							SD_{61}	SD_{62}	⋮	⋮	SD_{6n}	SD_{61}

The adapted key dimension for collecting contextual information helps in providing bounding limits, which could be applied to the various stages. This method is flexible enough to scale up and down as either contexts, or stage activities increase or decrease, and helps to provide combinatorial logic (Cohn et al., 2005) for matching contextual information to stages. Matrix multiplication have been applied in other similar areas to optimise the analysis and matching of bounded large data (Akutsu et al., 2000). The arithmetical complexity of the scenario could be evaluated and addressed using a matrix dot product operation (Bini, 2013), incorporating the minimum set of contexts and stages for any software development project in the cloud.

4.9 Summary

The importance of context in any system, scenario, or process, lies in the provision of characteristic information pertinent to that system, scenario or process(Hong et al., 2009). Contextual information could pertain to people, places, artefacts, processes, practices, tools, techniques, products, or literally, any facet of the whole(Petersen & Wohlin, 2009). Contextual information could be of different types or categories - implicit or explicit - i.e., either stated clearly, or indirectly expressed or suggested, but still related to the situation. Contextual information can also exist in different forms - naturally occurring, or as a by-product of something, someone, or some action. Nonetheless, no matter the type or form, contexts are very useful for characterization, for better, holistic and truthful representation of form at any given time or place (Petersen & Wohlin, 2009). The cloud possesses functionalities such as cloud repositories, that could be leveraged as a scalable distributed

mechanism for storing contextual information, project artefacts, requirements, user reviews and feedback. These scalable repositories can be accessed by all stakeholders in the team irrespective of distribution and decentralisation settings, via suitable synchronous and asynchronous means of communication

5 Theoretical framework

5.1 Introduction

Generally, theories are often adapted to better understand, generalize, abstractify, explain, predict, represent or back up phenomena, ideas, and contributions in any field (Gregor, 2006; Gregor & Jones, 2007). Theories can be multidisciplinary body of knowledge, providing a scientific basis for any phenomena (de Souza & Redmiles, 2003; Stol & Fitzgerald, 2013; Iyer & Power, 2014). Collaborative software development is not a new concept (Mistrík et al., 2010) but cloud-based collaborative software development is relatively new. Evidence shows that in large scale software development projects, collaboration takes place between people with a diverse remix set of different skills and experiences to ensure successful attainment of desired goal (Soriano Camino et al., 2008; DeFranco-Tommarello & Deek, 2002; Mockus & Herbsleb, 2001). Software engineering trends and emerging paradigms like cloud computing have impacted the collaborative nature of the development process in terms of distribution, complexity, adaptability and so on (Jeffery, n.d.; Zimmermann & Bird, 2012). Adequate theoretical basis is necessary for synthesis of knowledge and conceptualisation efforts towards enhancing collaboration in cloud-based collaborative software development (Jeffery, 2000; Ralph, 2013).

Cloud-based collaborative software development process refers to how all stakeholders within a software development project work together throughout the software development lifecycle, to achieve a common goal or outcome (Nordio et al., 2011; Skerrett, 2009). The collaborative software development process is one giant activity, made up of sub-activities. These involve requirements that undergo transformations via interactions, to yield artefacts. Artefacts from preceding activities, along with development tools, mediate and influence succeeding or subsequent activities. These artefacts also form the basis for verifying and validating each stage of the process, until the end goal is achieved. This process can be very resource-intensive and sometimes resource-specific.

Companies who have transitioned their development environments to the cloud, have started realizing benefits such as: cost reduction in hardware; relatively accelerated software development process via reduction of time and effort needed to set up development and testing environments; unified management; service and functionality expansion; on-demand

provisioning; and access to resources and development environments (Oberhauser, 2014, 2013; Jackson, 2011; Mahmood & Saeed, 2013). Collaborative software development process in the cloud presents complexities and contexts, amidst other factors, that need to be considered during the process (Mahmood & Saeed, 2013; Boehm, 2010). These are sometimes underestimated, ignored, or just not given enough consideration and planning. Result includes undermined collaboration in the process; negative impact on ability to facilitate a reproducible, sustainable, context-aware collaborative software development process in the cloud (Oberhauser, 2014, 2013). The factors and impact outlined in the paragraph above constitute principal motivation for need for adequate theoretical basis. Consistent reproduction of the software development process in the cloud requires standardization.

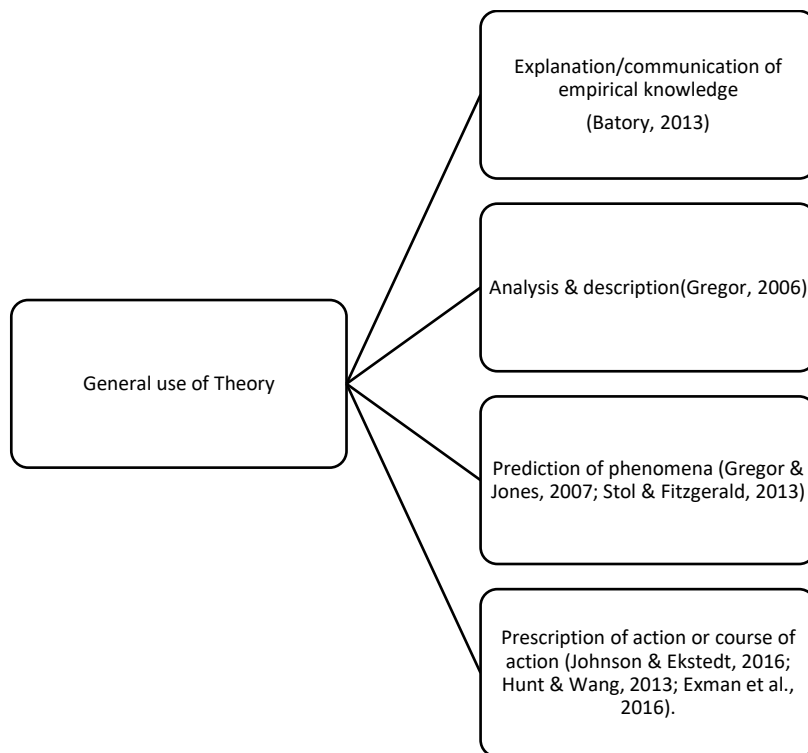


Figure 23: Classification of theory use

A suitable theoretical basis would ensure solid underpinnings for frameworks and architectures used in cloud-based development processes. It would contribute greatly towards facilitating reproducibility within the development process in the cloud. Currently, there are no existing theoretical foundations explicitly for architectures and frameworks for collaborative software development process in the cloud (Hunt & Wang, 2013; Ralph et al., 2013; Ralph, 2015). Software architectures provide foundational basis for large scale software

development and evolution. Because a software architecture serves as the intellectual centrepiece for developing software, it is necessary that all facets of the architecture be based on solid theoretical constructs and principles that support collaborative creation of software (Taylor et al., 2015). A review of current industry offerings highlights the need for architectures and frameworks with explicit theoretical foundations (Oberhauser, 2014; Richards, 2015; Chanda & Liu, 2015; Franken et al., 2015). This need is further reemphasized by reports of failed software projects (National Defense Industrial Association, 2010).

Other motivations for this research include: a need for identification of reliable ways of managing and measuring collaboration and success factors within the development process in the cloud; a need for new methodologies for enhancing effective collaboration across the entire development process; need for effective ways of managing complexity and ensuring synchronous regularity of process and understanding (Mohtashami et al., 2011; Bouwers, 2013; Münch & Schmid, 2013; Gorton et al., 2016). Software development concepts and methodologies show evidence of inherent collaboration within the software development process (Mistrík et al., 2010). The process typically involves different people with different background, skill sets and cultural practices of development, working together to-wards a common goal (Zimmermann & Bird, 2012).

However, the inherent collaborative nature of the software development process has been impacted in various ways by trends and emerging paradigms (Boehm, 2010, 2006). Some of these trends, alongside new paradigms, have also contributed towards undermining collaboration in some aspects of the software development process. These salient points call for a shift in research efforts, and more focus on how to enhance and improve collaboration in the software development process (Zimmermann & Bird, 2012). To do this, research efforts have sometimes veered away from traditional software engineering theories, and forayed into external disciplines for possible theories to leverage as basis, in the formulation of hypotheses, or design of systems (Jeffery, 2000).

In this research, a formal process is proposed to aid the selection of an appropriate theoretical basis for cloud-based collaborative software development. Prior to the proposal, related approaches that have previously been used in selecting theoretical foundations in software development are reviewed. The proposed formal process aids in the adoption of an appropriate theoretical basis for cloud-based collaborative software development process.

This process helps in the identification and review of a cross-section of relevant theories and concepts in relation to cloud-based collaborative software development. From this review, an adequate theoretical basis is selected with justification provided.

5.2 Related work

Recent research studies have shown increasing interest in the role of theories for software engineering, and emphasis on need for stronger, explicit theoretical foundations (Ralph, 2015; Stol & Fitzgerald, 2013; Ralph, 2016). Theories provide a solid body of knowledge that provide suitable framework for: communicating empirical knowledge, focusing on fundamental aspects and conceptual abstraction; rigorous hypothesis design, implementation, and evaluation (Batory, 2013; Johnson & Ekstedt, 2016). This has further fuelled existing efforts toward development of guidelines and approaches for selecting appropriate theoretical foundations for the software development process. Proponents agree that efforts should not only be restricted to synthesis of only existing theories within software engineering discipline but should extend to other reference disciplines too (Hunt & Wang, 2013; Exman et al., 2016). However, there is a lack of consensus regarding the process of selecting a theoretical foundation, with related ontological and epistemological aspects involved (Hunt & Wang, 2013). A consensus approach to selecting a theoretical foundation would facilitate choosing the right quality theoretical basis that would broadly explain, unify, and support collaborative cloud-based software development.

Software development entails a variety of phenomena (Hunt & Wang, 2013) which has multiplied with the advent of cloud environments. According to Gregor's taxonomy (Gregor & Jones, 2007; Gregor, 2006), appropriate theories and concepts can be adapted for use in: analysis and description of phenomena; explanation of workings or processes; predictions; and prescription of best course of action, based on predictions made. Theories used can be multidisciplinary, or transcend various disciplines (de Souza & Redmiles, 2003; Iyer & Power, 2014). The theories present a body of knowledge that provide a scientific basis for phenomena under investigation; and may derive inspiration from other directions. These directions may be technical, demographic, ethnographic, biological, economic, academic, or sociological. However, it is not always the case, for a selection of reviewed theories to have a homogenous mixture of best-fit concepts or features (Stol & Fitzgerald, 2013). There is need for proper consideration before adopting a theoretical basis.

A methodical protocol was developed and used in conjunction with concise query strings to guide and streamline the search and retrieval of relevant literature for review. This was done using Mendeley, a reference manager with a massive interconnected academic database, useful for finding, storing, managing, and correlating academic research materials and libraries (Raubenheimer, 2014). Mendeley was chosen because of its reasonably fair aggregation of research databases and has one of the largest databases in terms of research articles and journal coverage, and traffic (Cronin & Sugimoto, 2014).

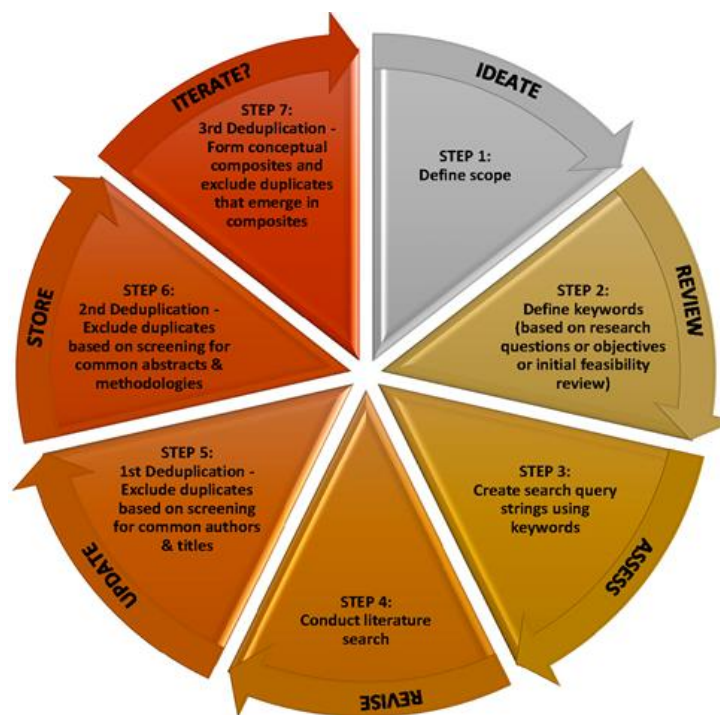


Figure 24: Systematic protocol for identifying relevant literature

Table 12 Overview of steps for searching, selecting, and deduping relevant articles based on use of relevant keywords.

S/N	Keyword-based Query Strings	# Articles before de-dupe	# Articles after first de-dupe	# Articles after second de-dupe
1	(title: "theoretical basis" OR "theoretical foundation" AND "*software development*")	100	52	20
2	(title: "theoretical basis" OR "theoretical foundation" AND "*software engineering*")	104	86	53

3	(title: "theoretical basis" OR "theoretical foundation" AND "*collaborative software engineering*")	0	0	0
4	title: "*theory*" AND "*collaborative software development*"	1	1	1
5	title: "*foundation*" AND "*collaborative software engineering*"	1	1	1
6	title: "*foundation*" AND "*collaborative software development*"	0	0	0
7	title: "*basis*" AND "*collaborative software development*"	0	0	0
8	title: "*basis*" AND "*collaborative software engineering*"	0	0	0
Total Number of Articles		204	138	73

Table 13 shows a cross-sectional summary of approach categories that have been used for selecting and adopting theoretical foundation in related works in software development projects. The observations made from a review of these approaches formed part of the considerations in the development of the proposed formal process. The proposed formal process described in this Section, is an approach that is reproducible, verifiable, and generalizable for conventional purposes. Although with formal methods, possibility of over-simplification and over-rationalization of reality exists, these could be addressed via trade-offs between possibility and empiricism (Ralph, 2014, 2013). The formal process developed in this research is similar in some respects to the “inductive grounded” theory approach in Table 13. This similarity is in terms of analysis and evaluation of information gathered, for “best potential” for research objectives according to pre-established set of guidelines. Information gathered is not based on subjective accounts, but rather on verifiable peer-reviewed literature.

Table 13 Cross-sectional summary of categories of common approaches for selecting theoretical foundations in the field of software development.

	Approach	Description	Observation
1	Randomized control trials (Concato et al., 2000; Webb et al., 2010; Ralph, 2014)	<ul style="list-style-type: none"> - Focus mostly on causal relationships between constructs. - For evaluating theories with dependent/independent variables. 	- Inadequate for focus on how a phenomenon changes, develops; is applied, described, predicted, or analysed
2	Observation approach (Noor, 2008; Ralph, n.d., 2014)	<ul style="list-style-type: none"> - Used in qualitative research. - Involves making observations compatible with constructs of proposed theoretical foundations. - Observations can also be incongruous, and mostly about responses. 	- Easily open to unconscious observer and response bias exploitation, to favour positive results
3	Quantitative questionnaire study (Ralph, 2013a; Ralph et al., 2013; Ralph, 2014)	<ul style="list-style-type: none"> - Involves making propositions for testing rival theories. - Propositions serve as basis for distinguishing between rival theories. - Survey questions, designed based on the propositions, are used to provide response distributions 	<ul style="list-style-type: none"> - Validity and objectivity of devised questions are not always reliable indicators of underlying theory constructs. - Need additional validation from experts or focus groups via detailed interpretations.
4	Qualitative field study (Given, 2008; Taylor et al., 2015; Ralph, 2014)	<ul style="list-style-type: none"> - Involves development of schemes for coding and listing constructs and relationships of a theory. - Coding scheme used as basis for building evidence for or against theory. - Evidence can be weighed and used to arrive at a conclusion. 	<ul style="list-style-type: none"> - Valid approach, but subject to mono-method bias. - Bias can be mitigated by facilitating data triangulation, for example, through introduction of empirical aspects
5	“Inductive grounded” theory approach (Hansen & Kautz, 2005; Urquhart, 2012;	<ul style="list-style-type: none"> - Allow generation of theory based on experiential accounts of practitioners. - Perspectives from experiential 	Investigations focus on experiential accounts of actors and their interrelations. Accounts are evaluated and used to generate mid-level

	Coleman & O'Connor, 2007; Adolph & Kruchten, 2013)	accounts are reconciled and evaluated for “best potential” according to pre-established set of guidelines.	theories to aid in design of software methodologies and architectures. These accounts are subjective; hence veracity is not assured. Accounts may lack the view of the big picture.
6	Separability Principles approach(Exman et al., 2016)	<ul style="list-style-type: none"> - based on separability principle in software engineering. - involve use of software design techniques to identify and understand relations between gathered candidate theories. - Relations are used to guide assembly of general theoretical framework. 	Originally meant to address problem of how to assemble appropriate theoretical concepts from a heterogeneous mix of theories. However, it does not cater to process of how to determine relevancy when gathering theories to be considered

5.3 A formal process for adoption of an appropriate theoretical basis

Due to the multi-faceted nature of the research area of cloud-based collaborative software development, the exploration of different theories from other disciplines external to the area of software engineering is necessary. This is sometimes the case for some research projects having interrelated themes or requiring multidisciplinary approaches or alternative approach to solution (Berg, 2017). The exploration of multidisciplinary theories could also be because of limitations or inadequacy of general theories in each domain or project, sometimes resulting in similar constructs showing up when exploring suitability of theories for adoption (de Souza & Redmiles, 2003; Stol & Fitzgerald, 2013; Ralph, 2013). There exists the likelihood that selected theories for various projects within same domain could differ in how and where they have been applied. Hence, there is need for a process that takes this into consideration.

Keyword search was carried out for literature on relevant theories relating to the following main interest areas - collaborative software development in the cloud, context-awareness, and collaboration, in line with Gregor’s proposed theory taxonomy (Gregor & Jones, 2007; Gregor, 2006). The search yielded scenarios or research projects where combination of multidisciplinary theories, principles, models, and methods were used to generalize or explain phenomena, ideas, and contributions in various domains. Reviewed literature also showed evidence that theoretical concepts of multidisciplinary theories had also been adopted in proposals and applications of ways for supporting and analysing software development.

However, very few showed adoptions for the entire cloud-based collaborative software development process lifecycle. The multidisciplinary nature of adopted theories creates challenges when it comes to creating common basis for comparison; and, developing necessary and appropriate linkages between borrowed theories, and the specified project or domain. One way to approach this challenge could be via an inductive or deductive approach, or both (Fereday & Muir-Cochrane, 2006). To form the necessary and appropriate linkages between borrowed theories and this research problem domain, a combination of both inductive and deductive approach was adopted.

The inductive part of the approach entailed obtaining observations via review of literature on relevant theoretical concepts and the application of these concepts within cloud-based collaborative software development. This review was then condensed into a summary, to make it easier to identify and establish links between theories and defined objectives. Applying this inductive approach makes it easier to analyse processes present in theoretical concepts and applications, for valid, reliable, and quality information and insights. This process is not without bias, but it has been known to yield effective results (Fereday & Muir-Cochrane, 2006; Thomas, 2006). The deductive part of the approach entails the use of logical reasoning in determining pros and cons of reviewed theories with respect to cloud-based collaborative software development. This information can be evaluated for constructs towards forming the theoretical basis for an appropriate over-arching high-level framework for cloud-based collaborative software development.

5.3.1 Overview of the proposed formal process

The proposed process provides a means of capturing essential features of a defined problem scenario within a given project or domain. These captured essential features form a set of predefined themes that represent domain-based or project-based dimensions and can be used as frame of reference (Fereday & Muir-Cochrane, 2006). The set of predefined themes could be changed according to the chosen domain or project and used by the process as frames of reference for analysing, measuring, and evaluating various theories with the aim of selecting an appropriate theoretical basis (Chorin & Hald, 2014). The chosen domain case study in this paper is cloud-based collaborative software development. The values for the predefined themes for the proposed process can be numerical values or weighted vectors. These values are used to measure the increase or decrease in relevance or suitability of a

theory to cloud-based collaborative software development. To measure the relevance or suitability of theories, it is important to understand main constructs, organization, application, and limitations.

Generally, it is easier to measure and analyse quantitative value data than qualitative value data, using analytical techniques (Jr & Boone, 2012). Qualitative value data possess non-numeric attributes such as descriptions and so on. Therefore, they are mostly nominal in nature, but still capable of generating insights that enable conclusions to be drawn. An approach to enabling empirical conclusions from qualitative value data is via a reliable quantification of qualitative data. One way of doing this is by identifying patterns that can be quantified in terms of relationships or frequencies (Jr & Boone, 2012). To do this, qualitative value data must be organized into groups; systematically categorized according to carefully chosen labels or themes relevant to the domain or project. These labels or themes can either be assigned numerical values or weights. The resulting frequencies of emerging patterns are identified and counted; allowing for estimations based on some sort of content intervals (Johnson et al., 2010). In the absence of naturally occurring patterns in each set of qualitative value data, it is possible to adapt a Likert scale-type method of measurement to generate patterns that can be quantified and measured (Chorin & Hald, 2014). The benefits of this approach to quantifying qualitative data cannot be overemphasized as highlighted by Johnson et al (Johnson et al., 2010).

One of the rationales for developing this approach is to attempt to standardize the selection of theoretical basis in software engineering and cloud computing research projects, as well as to having a means to deal with any multiplicities that may arise. The process proposed in this paper, provides a means of measuring similar constructs via selecting and combining appropriately correlated variables to form composites for general measurements (Marusteri & Bacarea, 2010; Stuckey et al., 2014). The adopted process for selecting theoretical basis, with the mathematical evaluation method of the process, is used partly for emphasis and assurance in the validity of the outcomes of this quest for a theoretical basis (Kenneth F. Hyde, 2000). This formal process is depicted using a flowchart in Fig. 26 below.

5.3.2 The problem scenario

In selecting an adequate theoretical foundation for the domain - cloud-based collaborative software development, a universal set or group of theories characterized by a set of

observable and measurable variables is considered. Can any theory or theories from this universal set be shown to be related to collaborative activities and interactions between members of a team geared towards achievement of a stated goal or outcome? Relationship between the theories and the collaborative activities could be with respect to theory constructs; application areas; identified patterns, themes, or dimensions; etcetera. The goal is to identify and select from the universal set of theories, a theory or set of theories that best meet the criteria for an adequate theoretical foundation, given the specified area of interest and initial conditions.

5.3.3 Criteria for theoretical foundation

An appropriate theoretical foundation for a domain or project, must be able to meet all, or some of the importance/uses of theory, in line with Gregor's taxonomy (Gregor, 2006; Gregor & Jones, 2007). These uses are:

- analysis and description
- explanation of workings or processes
- prediction
- prescription of best course of action based on predictions

5.3.4 Question

Which theory or set of theories best satisfy, or meet the criteria for theoretical foundation?

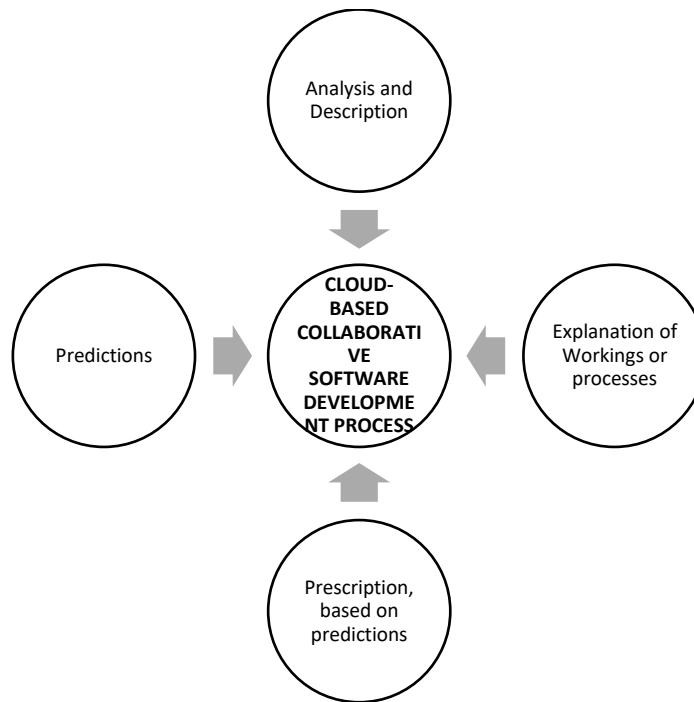


Figure 25: Relevance of an adequate theoretical foundation for cloud-based collaborative software development

5.3.5 Parameters

The following variables are defined for this process:

Table 14 Proposed process' variables and description

VARIABLES	DESCRIPTION
P	Specified domain or project
RIA_p	Set of main related interest areas of domain or project.
X_p	Set of n observable variables (theories) related to the domain or project.
TI_p	Importance/use categories for theory. Equal weights of 1 Analyse/Describe = TI_1 ; Explain = TI_2 ; Predict = TI_3 ; Prescribe = TI_4
W_p	Likert-style weighting of theory relevance based on sum of weights: W_1 = Neutral; W_2 = Somewhat relevant; W_3 = Relevant; W_4 = Very relevant 4
Y_p	set of parameters characterizing the project or domain
TF	Result of the estimate process

5.3.6 Assumptions

Given any collection of multidisciplinary or same-discipline theories, any subset has the potential to pre-send as an adequate theoretical foundation, depending on constructs, and trade-off between application of constructs and limitations, relative to importance or use of theory for a project or domain in question.

5.3.7 Initial conditions

At start of process,

$$RIAp = \{\}; Xp = \{\}; TIp = 0; Wp = 0$$

5.3.8 Modelling the process

The selection process for adequate theoretical basis for a given domain or project depicted in Fig.26 below, follow the steps depicted below:

- i. Carefully identify and define the main related interest areas, (RIA) of the domain or project.

$$RIA_p = \{RIA_1, RIA_2, RIA_3, \dots, RIA_n\}$$

- ii. Identify/define set of theories and concepts to be evaluated for suitability as theoretical basis

$$X_p = \{X_1, X_2, X_3, \dots, X_n\}$$

- iii. Analyse and categorize data from literature pertinent to each X_i above, into the themes or labels defined below. These are assumed defaults to give a sense of direction. These defaults can be changed for more suitable ones depending on project considerations. The default analysis themes, or labels are as below:

- Constructs – refers to main or general concepts of the identified theory
- Known applications – refers to known applications of the identified theory within any or all the main related interest areas
- Pros – refers to known positives/benefits or advantages of identified theory with respect to any or all main related interest areas
- Cons - refers to known negatives or disadvantages of identified theory with respect to any or all the main related interest areas

iv. From (III), match theory X_i to most appropriate importance/use category TI , for domain or project, in relation to defined related interest areas (RIA_p) and based on the themes in (III) above. A value of 1 is assigned for every match, while a value of 0 is assigned if there is no match. Assignment of weights of 1 to identified matches, ensures that stability is observed in the values

For any given member of X_p , the degree of relevance is denoted as:

$$TI_p = \sum_{i=1}^n TI$$

v. Since theory can be used to guide alignment and integration of sub-components of a domain (Taxén, 2007; Gregor, 2006; Batory, 2013; Gregor & Jones, 2007), the alignment of the candidates for theoretical foundation can be altitudinally measured by mapping the degree of relevance TI_p , to the defined 4-point Likert scale.

Table 15 4-point Likert scale for the formal process for selecting theoretical foundation.

LIKERT SCALE	DESCRIPTION	WEIGHT	CONDITION
W₁	Not relevant	0	$TI_p = 0$
W₂	Neutral	1	$TI_p = 1$
W₃	Somewhat relevant	2	$TI_p = 2$
W₄	Relevant	3	$TI_p = 3$
W₅	Very relevant	4	$TI_p = 4$

vi. Make selection based on element of set XP evaluating to highest value on defined Likert scale.

$$\underline{\text{Theoretical foundation}} = \text{Max } \underline{TI_p} = \text{Max } \left(\sum_{i=1}^n TI \right)$$

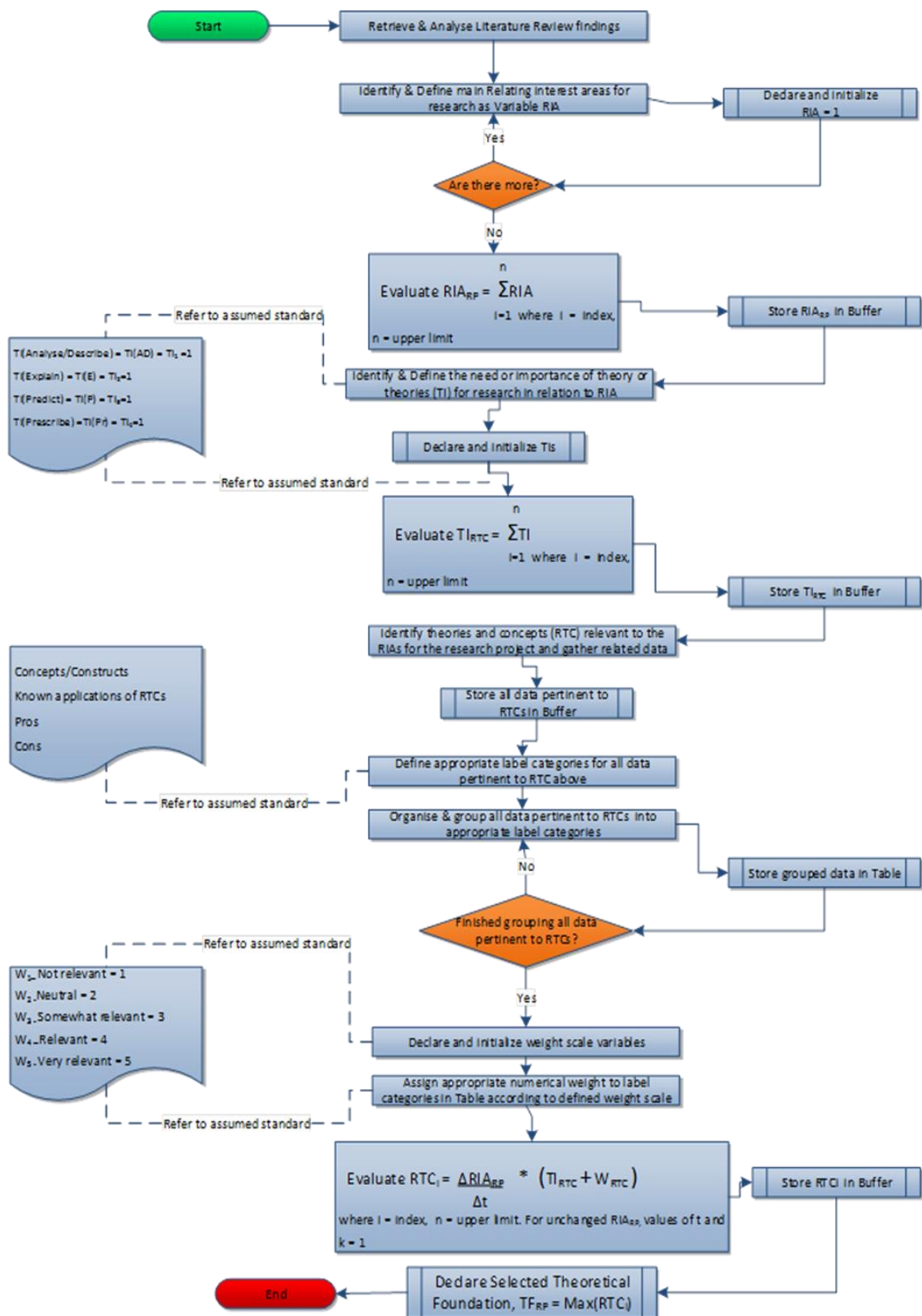


Figure 266: Flowchart for a formal process for adoption of an appropriate theoretical basis

5.4 Application of the proposed formal process

- i. Where P represents cloud-based collaborative software development domain, the main related interest areas, (RIA) are:

$$RIA_p = \{software\ development, cloud\ computing, collaboration, context-awareness\}$$

- ii. The set of theories to be evaluated for suitability as theoretical foundation for cloud-based collaborative software development is as shown below:

$$X_p = \{Information\ Foraging\ Theory, Complexity\ Theory, Game\ Theory, Actor-Network\ Theory, Activity\ Theory\}$$

Note: These theories are selected based on applicability of theoretical concepts to the main related interest areas for cloud-based collaborative software development. Evidence is supplied from literature analysis

- iii. Section 4.1 shows the analysis of data from literature pertinent to each X_p above, based on defined themes – constructs, known applications, pros, and cons
- iv. Each member of X_p is matched to the most appropriate importance or use category TI , in relation to defined related interest areas for cloud-based collaborative software development (RIA_p) and based on the themes in (III). A value of 1 is assigned for matches, while a value of 0 is assigned for non-matches. This is shown in Table 15.
- v. For any given member of X_p , the degree of relevance is denoted as:

$$TI_p = \sum_{i=1}^n TI$$

From Table 15 above,

$$TI_{Information\ Foraging\ Theory} = 2$$

$$TI_{Complexity\ Theory} = 3$$

$$TI_{Game\ Theory} = 3$$

$$TI_{Actor-Network\ Theory} = 3$$

$$TI_{Activity\ Theory} = 4$$

Since, Selected theoretical foundation = $\text{Max}(W_p)$

Hence, Selected theoretical foundation = Activity Theory

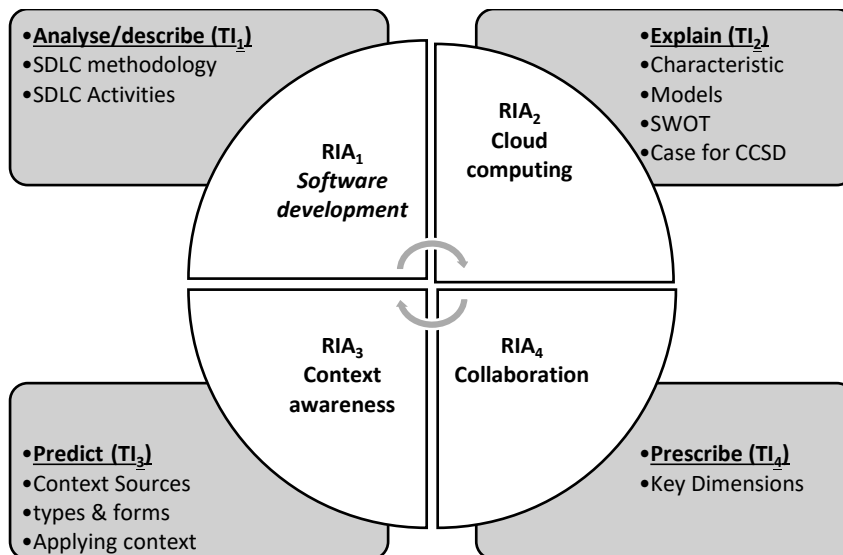


Figure 27: key focus points and impact areas for cloud-based collaborative software development based on output of formal theoretical process applied in Sections 5.1 – 5.4, in line with Gregor’s taxonomy (Gregor, 2006; Gregor & Jones, 2007)

5.5 Cross-sectional review of relevant theories

This section presents a cross-sectional review of relevant theories and evaluation using the theoretical basis selection process (see Table 16). The results are presented in Table 16 and shows how evaluated theories match up to appropriate importance or use category for cloud-based collaborative software development. To reduce the likelihood of false negatives, expanded keyword-based query strings to ensure wider range of relevant literature was gathered for observation and review. To reduce over-simplification and over-rationalization during review and evaluation, trade-offs had to be made between possibility and empiricism by ensuring that analysis and evaluation was carried out for “best potential”, rather than “exactness”, and based on peer-reviewed accounts rather than subjective accounts. Adopting this approach comes with a negligible risk of introducing false positives because of the reduction in threshold for significance (Ralph, 2014, 2013). However, this was managed by keeping the investigation focus experiential accounts of actors and their interrelations. In addition, alternative data collection and analysis methods were employed to help to confirm the findings.

5.5.1 Information Foraging Theory (IFT)

Concepts/Constructs:

The three major components of IFT are – information diet, information patches, and information scents, all used towards achieving a goal (Kwan et al., 2012; Lawrance et al., 2007). IFT postulates that useful information features (diet), are usually found in patches connected by scents or links, with associated costs. These information features have values. Possible gains that could accrue from following a link are suggested by cues (Pirulli, 2005). This helps in predicting navigations based on the supposition that the forager would ideally want to find or navigate to patches which provide or contain rich and valuable information at very low cost.

Known Applications:

IFT has been applied in the design of program maintenance support tools and environments to aid developers in seeking, relating, collecting, and applying information to tasks and activities in a timely fashion (Ko et al., 2011). The rationale for this was based on the following: when programmers, testers, or designers are faced with tasks or issues, they tend to spend a sizeable amount of time to navigate or sift through various hypotheses, or methods of either approaching a task or solving an issue. Not knowing where to look, or taking too long doing so, may sometimes result in delays in the development, or resolution of issues, or an endless loop of interleaving between courses of action.

Information Foraging Theory has been adopted in some areas to address how to approach a task or problem, by optimising how developers and testers can navigate source codes and designs, as well as, in predicting maintenance and coding behaviours. It also proved useful in helping developers to find and apply relevant information during the development process, for desired outcomes. This is particularly useful for testing and maintenance activities within the software development process, and in the design of notification, or awareness systems, to support the establishment and maintenance of awareness of team members' activities, intentions, tasks, and results (Carroll et al., 2003). Other applications of IFT include: CogTool-Explorer (John et al., 2013), Hipikat and PFIS (Cubranic et al., 2005; Lawrance et al., 2010).

Observation:

Though conducted experiments showed relationships between predictions of developers' behaviours with their actual behaviours, the missing link was the explanation as to the cause of the behaviour (Fleming et al., 2013; Kwan et al., 2012). Knowing the root cause of a behaviour could be useful in increasing the accuracy of pinpointing and identifying relationships, which would of course, translate into greater accuracy in predictions, as well as the generation of other useful insights. Furthermore, in cases where a disparity exists between predicted and actual behaviour, it can be quite difficult to determine the cause or fault.

Secondly, IFT does not fully take into consideration the complexity of systems, or in this case, the potential complexity in a large scale or distributed development process, such as software development in the Cloud. It remains to be seen how adequate it would be in supporting more complex and persistent activities across heterogeneous environments, or team makeup (Cubranic et al., 2005; Lawrance et al., 2010). However, the application of IFT could be effective in other activities involving planning, coordination and collaborative foraging or exploration of information.

5.5.2 Game Theory

Concepts/Constructs:

Game theory is a theoretical model which is mainly used for interactive analysis and decision making, through evaluation and weighing of choices, and interactions between participants in a system or model (Camerer, 2003). The outcome for each participant is dependent on, or affected by, the collective actions of all the participants. It can also provide insights in situations where the participating individuals, or individual groups, have individual preferences or goals (Savani & Stengel, 2014). The application of game theory comes in very handy, in scenarios where strategic thinking is necessary for striking a balance between competition and cooperation.

Known Applications:

Commonly used to model agent-based decision systems, games, game engines, simulation applications and scenarios e.g., Game Theory Explorer, involving strategic thinking, strategic interaction, conflicts, and cooperation interaction (Parsons et al., 2012; Sazawal & Sudan, 2009).

Observation:

Although game theory provides a very resourceful framework for analysing and evaluating problems to work out best strategy or decision for optimal gain or utility; with respect to collaborative software development, its adequateness is not proportional to all the identified aspects of cloud-based collaborative soft-ware development (Sazawal & Sudan, 2009). Game theory tilts more towards facilitating coordination, and strategically alternates between optimising cooperation or competition, depending on identified stakes. Guarantees for any evaluated decisions and alternatives are only available in hindsight (Highsmith, 2013).

One school of thought suggests that, because collaboration does not always comprise of mutually aligned goals or values, adopting game theory would be appropriate (Saoud & Mark, 2006). This opinion is hinged on game theory's strategic significance when it comes to overcoming goals or values that are negative, or, not mutually aligned, and optimising outcomes, via optimal cooperation or competition. This stance is depicted in the classic Prisoner's Dilemma scenario, where game theory is sought to evaluate and address the interactions between the two parties, towards revealing the importance of knowing when to compete, versus when to cooperate (Saoud & Mark, 2006). Enhancing the collaboration within the software development process in the cloud is not about evaluating competition versus cooperation scenarios in the face of mutually aligned and mutually exclusive scenarios within the process (Oberhauser, 2013, 2014). Rather, it is about facilitating, enhancing, and optimising consistent and continuous collaboration and integration within the software development process, irrespective of mutuality clauses present.

With respect to software development, applying game theory to make strategic decisions, one may need to be aware of how all composite components within a software development ecosystem function, or work together within the ecosystem. This does not explicitly mean that it facilitates or implements the functionalities of the individual composite components of a soft-ware development ecosystem. Although, it may appear to take into consideration various aspects of interaction, it does not necessarily proactively facilitate awareness mechanisms or techniques. The known applications of game theory emphasize its usefulness when evaluating decisions involving cooperation versus competition, ultimately geared towards an optimal outcome, rather than, explicit collaboration towards an identified common goal. This is not to say that it can-not be used in collaborative endeavours.

5.5.3 Complexity Theory

Concepts/Constructs:

This comprises of a set of procedures, practices, and techniques for studying and investigating complex systems, and the interactions between the actors and components of complex systems (Axelrod, 2015). Complex systems are usually referred to as systems, made up of composite parts and their interactions (Manson, 2001). Collaborative software development can be viewed in the light of the above definition, as a complex system or process, since it does conform to the general characteristics of a complex system, even if marginally so. Introducing Cloud computing contexts to this, arguably adds to the complexity. Complexity theory usually concerns itself with three categories of complexity - algorithmic, deterministic, and aggregate complexity (Barthelmess & Anderson, 2002b). All three forms of complexity share similar historical antecedents, as well as, are concerned with the characterization of a complex system with reference to constituent parts in a manner that is classed as non-reductionist manner. Because it is one of the antecedents of complexity theory, a review of General Systems Theory is not done (Ponti, 2011).

Known Applications:

One of the significant applications of the complexity theory is the construction of agent-based simulators to aid collaboration for design teams (Axelrod, 2015). These simulators are then used to simulate various scenarios based on parameters and values and used to identify best-case scenarios and worst-case scenarios for collaborative engineering design workloads. Agent-based simulators can be useful in planning and modelling processes. Other areas of application of complexity theory include design and tuning of networks to collaborative structures (Saoud & Mark, 2006).

Observation:

From the definition and applications highlighted above, it can be deduced that the use of complexity theory comes across as a passive approach. Passivity is used here to represent being more pro-active, than reactive. For instance, adopting complexity theory would be useful in understanding and modelling complex systems, identifying best scenarios, worst scenarios, and good strategies. This is good because it contributes towards answering the 'what', and a bit of the 'when' when modelling scenarios and solutions. But it does not answer

the 'how', nor the 'when' aspect relating to the 'how'. It also does not clearly offer much with regards to introducing flexibility for non-existent components or interactions, at the time of modelling or building. This presents the potential for introducing integration or compatibility issues, amongst others. Complexity theory investigates the known aspects of a complex system, as well as the known interactions. However, collaborative software development in the Cloud is not so close-ended as to be oblivious to the possibility of the introduction of unknowns during a software development project. Besides, the parameters and values garnered from studying or investigating one instance of Cloud-based collaborative development does not represent a conclusive representation of all other instances. Hence, adopting complexity theory as a suitable theoretical basis does not present as the best alternative.

5.5.4 Actor-Network Theory (ANT)

Concepts/Constructs:

ANT is a conceptual approach (Ahmedshareef et al., 2014), a theory geared towards modelling and understanding complexities, or organizations, as well as the contextual role and impact of technology (Cresswell et al., 2010). ANT is acknowledged to be useful in understanding and describing human inter-actions with objects – both animate and inanimate. ANT is mostly used in practice to examine and de-scribe the relationships and interactions between people, objects, things, organizations, and ideas, as well as the creation, maintenance, and modification of these over time, in their journey towards achieving a goal. In actor-network theory, the emphasis lies more with the actors (nodes) existing within a network and their interactions (links) needed to achieve a goal. Actors in this context refer to either human or non-human counterparts with different makeup or value.

Known Applications:

Some of the applications of ANT theory include understanding and implementation of information systems in healthcare; application of quantitative project metrics analysis and ANT-based qualitative analysis in software project management research (Cresswell et al., 2010; Heeks & Stanforth, 2015).

Observation:

There are some criticisms and scepticisms around the application of ANT in software development research and other projects from other disciplines. Firstly, although ANT can prove helpful in understanding interactions or how things happen within a system or network or group, it falls a little short when it comes to providing satisfactory reasons why they happen (Rivers et al., 2009). This could translate into ambiguity or difficulty when it comes to employing ANT as an approach for developing practical guidelines for implementations. This is not an oversight from the development or formulation of the theory. Rather, it is an inherent approach embedded in the ANT methodology. Arguably, though there is an affirmation of the in-separability of description from explanation, the methodology promoted by ANT sidesteps this as a way of avoiding any explanations with pointers to principles, or aspects unconnected with the actions of the actors. The ANT methodology does not acknowledge the existence of contexts outside those generated by the actions of the actor (Ahmedshareef et al., 2014). This aspect of ANT does not align with the objectives of cloud-based collaborative software development, which seeks to acknowledge the possibility of collaboration being impacted by all other contexts, implicitly or explicitly associated within a software development project. However, ANT's methodology can be useful in explaining relationships between actors, or process or trajectory of interactions, as well as the role played by any underlying technology. It can also help in the analysis of outcomes dependent on either the actors or the interactions.

5.5.5 Activity Theory (AT)

Concepts/Constructs:

AT is a descriptive and analytical framework whose earliest form was represented by the external and internal relationships and interactions between three aspects or components of an activity – subject, object, and mediating tools (Rivers et al., 2009; C. Ghaouied, 2006). This theory can be summarised by three questions: Who is doing what? Why? How is it being done? AT was originally created to aid in better understanding the structure, context, and development of activities (Engeström, 2001; Engestrom, 2000).

Known Applications:

Some of the applications of AT include analysis and evaluation of software development environments and their collaborative capabilities; in the design of human computer interfaces, collaborative educational learning systems and other interactive systems (C. Ghaouied, 2006).

Observation:

The subject represents the human elements and activities that strive to satisfy the objectives; the object can represent either a concrete entity, or an abstract notion (e.g., an idea); the mediating tools represent the supporting tools (e.g., models, physical like IDEs, etc.) that mediate the relationship between the subject and object and used in the transformation of the object into a desired outcome. This representation connotes or implies an alignment with the principles of software engineering – abstraction, separation of concerns, modularity, change anticipation and management, generality, incremental development, as well as concepts such as object-orientation (Desai, 2007). AT in its original form as postulated by Vygotsky was quite restrictive in the sense that it was focused mainly on human psychology, and quite difficult to extend or even apply to other domains outside the social sciences. It required more research efforts to be extended to a wider domain.

Additional efforts by researchers, further expanded AT into the second and third generation forms by introducing cultural and social contexts (Il'enkov, 2009; Leont'ev, 2009). These expanded forms broke the tools down and incorporated the following: community, to mediate the relationship between the subject and the object; rules, to mediate the relationship between the subject and the community; division of labour, to mediate the relationship between the community and the object. The community represents grouping or alignment of common interest; the division of labour represents the separation of concerns or complexities and allocation of responsibilities; the rules represent the regulations, boundaries, syntax, semantics, constraints, and guidelines that arise from the existence of division of labour.

In AT, the unit of analysis is the Activity, which comprises of focused, goal-oriented actions geared towards the transformation of the object. The main characteristics or fundamental generalizations of an activity espoused by the AT framework include: an activity is directed

towards an object, distinguishable by that object, and realized through a set of actions or operations; an activity is mediated by tools; and exist within contexts which could be either social, cultural, or technical. All three characteristics are underlined or guided by anticipation, which is synthesized from an afferent synthesis of the mediating tools, the subject, and the object. This anticipation is enabled by the recurrent nature of these three components, and dependent on time.

The latter generation of AT facilitates the implementation of a framework which allows for the coordination of resources via rules and division of labour, whilst taking into consideration contextual conditions. While the rules play a fundamentally formal role in the organization of activities necessary for the trans-formation of the object, the contexts help to ensure that adequate considerations and conditions are considered to ensure alignment with the objectives, to ensure the right outcome. The conceptualisation afforded by this analytical framework, makes it possible to implement a loosely coupled, but all-inclusive Cloud framework to facilitate and enhance contextualized collaborative activities geared towards the successful achievement of a desired goal.

Table 16 Summary of evaluated theories and their matches to appropriate importance or use category for cloud-based collaborative software development:

THEORIES	GENERAL USE CASE	EXAMPLE APPLICATION	PROS	CONS	Theory Importance aspects		Weight Scale (W1, W2, W3, W4, W5)						
					I(AD)	TI(E)	TI(P)	TI(Pr)	W1	W2	W3	W4	W5
Information Foraging Theory	➤ Information search, gathering, filtering, use & tracking	<ul style="list-style-type: none"> ➤ Design of collaborative tools and support tools for timely and collaborative seeking, relating, collecting, & applying of information and resources relevant to tasks & activities in global software teams e.g., CogTool Explorer, HipiKat, PFIS ➤ Design of tools for information-intensive, recurring activities 	<ul style="list-style-type: none"> ➤ promote awareness of relevant information ➤ optimise navigation of relevant information to support development process ➤ positive impact on activity coordination 	<ul style="list-style-type: none"> ➤ Knowledge-centric ➤ tendency to starve other aspects of attention ➤ time consuming 	0	1	1	0	-	-	X	-	-
Complexity Theory	<ul style="list-style-type: none"> ➤ Investigation of complex systems & concerns – algorithmic, deterministic & aggregate ➤ Analysis of interaction between 	<ul style="list-style-type: none"> ➤ Construction of agent-based simulators to aid collaboration for design teams ➤ Design and tuning of networks to collaborative structures 	<ul style="list-style-type: none"> ➤ Proactive approach for planning, designing, and modelling scenarios and strategies in existing systems ➤ Useful for investigating known aspects of a system 	<ul style="list-style-type: none"> ➤ Not ideal for processes requiring flexibility for future components or interactions ➤ Not ideal for open-ended collaborative systems, projects, and platforms 	1	1	1	0	-	-	-	X	-

	components & actors													
Game Theory	<ul style="list-style-type: none"> ➤ Analysis/evaluation of interactions and collective actions of system participants 	<ul style="list-style-type: none"> ➤ Games, game engines, simulation applications, strategy applications and scenarios e.g., Game Theory Explorer ➤ Modelling of agent-based decision systems and practical applications in fields e.g., Economics, Biology, etc. 	<ul style="list-style-type: none"> ➤ Resourceful framework for analysing and evaluating best strategy out of a given set of strategies ➤ Facilitates awareness of possible and best outcomes in a situation ➤ Facilitates and promotes coordination when applied towards optimising cooperation or competition ➤ Useful in facilitating implicit & partial collaboration 	<ul style="list-style-type: none"> ➤ Not ideal for enhancing, explicit, continuous, holistic collaboration and integration ➤ Enhancing communication is not one of the strongest suits 	1	1	1	0	-	-	-	X	-	
Actor-Network Theory	<ul style="list-style-type: none"> ➤ Description/modelling of nodes, links & interactions within a network 	<ul style="list-style-type: none"> ➤ Modelling and implementation of information systems in healthcare and other fields ➤ Application of quantitative project metrics analysis in Software Project management 	<ul style="list-style-type: none"> ➤ Useful for analysing and describing actors' interactions from creation, through modification and maintenance ➤ Useful for modelling actor-related contexts surrounding actors and interactions. 	<ul style="list-style-type: none"> ➤ Ambiguity in analysis and modelling of interactions unconnected to actors ➤ inability to provide satisfactory explanations for lack of connection between interactions & existing actors ➤ little or no acknowledgement of other contexts un-related with actors 	1	1	1	0	-	-	-	X	-	

Activity Theory	➤ Contextual definition and modelling of activity, components, interactions & transformations	➤ Analysis/evaluation of Process-centred software development environments (PCSDE) with respect to the impact of their capabilities on the collaborative nature of Software development projects	<ul style="list-style-type: none"> ➤ Provide different perspective from production-oriented one ➤ Descriptive tool useful in analysis of collaborative work ➤ Centres on broader classification of collaborative work/activities 	➤ Acknowledges existence of interactions necessary for object transformations, but no clearly defined major or minor steps or interactions between subject and object	1	1	1	1	-	-	-	-	X
-----------------	---	--	---	---	---	---	---	---	---	---	---	---	---

5.6 Analysis and justification for Activity theory as theoretical basis for cloud-based collaborative software development

The use of AT in this research project focuses on better understanding of collaborative software development process as an activity and the interactions within; as well as, gaining insight into the value and adequateness of AT for modelling an efficient framework for context-aware, collaborative software development in the cloud. There is evidence of inherent collaboration in the software development process, but research also highlights gaps present, alongside effects of emerging technology trends (Mahmood & Saeed, 2013; Boehm, 2010). Rapidly changing distributive trends like cloud computing, create urgent need for creation of better frameworks with sound theoretical underpinnings to withstand the test of time and to embrace such trends and the complexity they introduce, whilst facilitating greater productivity and experience (Kyriakidou-Zacharoudiou, 2011).

Activity theory stems from early work in the field of psychology towards the development of a psychological theory, with relation to human action and thinking (Kozulin, 1986). These early efforts differentiate between people and things based on motive and consciousness. Activity theory is a conceptual framework, with 'Activity' as unit of analysis, making it ideal for: studying activities or activity systems and related practices; identifying congruencies and contradictions emerging from interactions (Dennehy & Conboy, 2016, 2017; Spinuzzi, 2015). An activity in the first generation of activity theory refers to interactions between subject or subjects ('actors'), and the object ('world'), mediated by tools, or artefacts (Soegaard & Friis Dam, 2013). The interactions refer to the process that relates the subject and the object, implying that an activity is dependent, or can be influenced by attributes of both object and subjects.

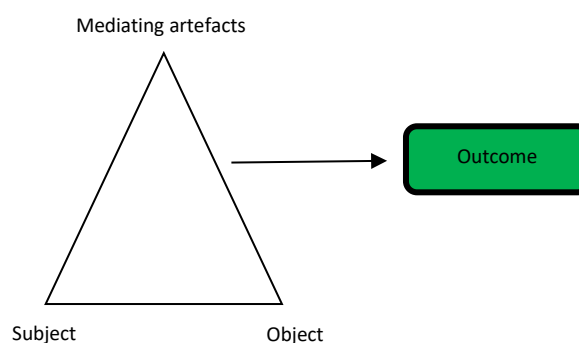


Figure 28: First generation of Activity Theory

The first generation activity theory used the concept of activity to investigate a wider scope of problems related to societal phenomena, by taking into account the dynamics of all social interactions in relation to the societal phenomena, in a bid to achieve a better outcome (Bedny & Harris, 2005). One of the strengths of first generation of activity theory is its basis - not on any abstract concept, but rather, on humans, activities, associated conditions and artefacts (Roth & Lee, 2007; Kozulin, 1986). Secondly, it focuses on social-cultural perspectives of human activity which underlines relationship between the object and subject, and its inseparability from the human mind, existing side-by-side, in various social and cultural contexts (Bedny & Harris, 2005; Roth & Lee, 2007). This represents an introduction of context as a factor capable of affecting actions within the activity.

However, one of the lapses of first generation of activity theory, is lack of a clear distinction between action and activity (Engestrom, 2000; Engeström et al., 1999). It takes for granted, or rather, gives the subject(s) free rein to infer the aims of actions based on the artefact or the preceding action, without taking into consideration the potential impact of all contexts on actions. This introduces a lack of uniform or systematic approach towards a means of synchronized understanding of the object and create room for potential misunderstandings, misinterpretations, misrepresentations, and by-passing of relevant aspects necessary to better understand the object and enhance the desired outcome (Engestrom, 2000). This in turn affects the next action and any subsequent generated object, since objects motivate and direct activities, just as much as activities are directed towards objects and are differentiated or distinguished from one another by each activity's respective object (Soegaard & Friis Dam, 2013). This object-oriented transitive relationship is represented in Figure 29. As a result of this relationship, analysis of the object and consideration of all related contexts is a necessary requirement for individual and collective understanding of activities. This immediately highlights the next limitation of the first generation of activity theory – the focus of the unit of analysis from an individual perspective (Uden et al., 2008).

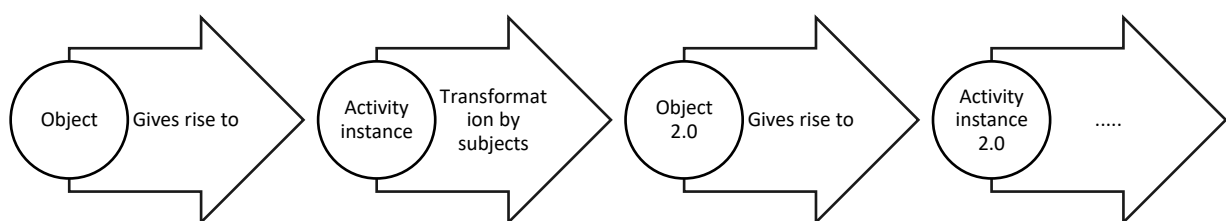


Figure 29: object-oriented transitive relationship between object and activity instance

The second and third generations of activity theory extend this individual focus to reflect joint activity, interactions between subjects and their environments in activity analysis, as well as, other context factors such as the community, responsibilities within activity systems and rules to govern interactions and transformations (Engestrom, 2000; Uden et al., 2008). This expanded perspective result in a structure extended to include the afore-mentioned context factors as three key components used to establish more collective context – rules, community, and division of labour. These components introduce focus on the collectiveness of an activity through actions contributed by individuals, groups, and organizations (community); governed by rules; supported by tools and signs; and carried out in defined structures or patterns (division of labour).

The rules, tools, and division of labour aspects, mediate interactions between the community and the subject, as well as interactions between the community and the object. They can also be extended for coordination and management purposes. This is because the mediating rules are binding on subjects and community within the activity, in a pattern aligned with their roles and responsibilities. These rules ensure that activity transformations (actions of the subject and community) remain focused on the goal and are object-oriented. The second and third generation of AT ensure that activity acknowledges the *“influential nature and interrelatedness of the larger social context”* within which an individual or group of individual carries out goal-oriented actions on an object subject to constraints or bounding logic and conventions(Dennehy & Conboy, 2016).

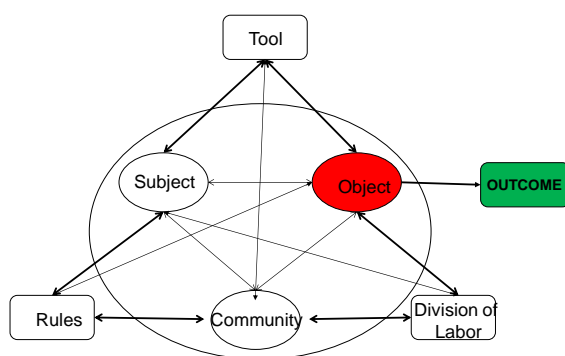


Figure 30 Engestrom’s model (2001) of Activity Theory

One lapse of the second and third generation of activity theory, which is crucial for collaborative software development process is the non-explicit representation of the time

context of the activity, along with related historical elements, as well as future elements (Tell & Babar, 2012). The value of this lies in providing the ability to analyse historical activity instances (historical transformations of objects by subjects/members of the activity instances and mediating tools/rules) and predict future transformations based on current context.

Another lapse of activity theory that needs addressing to make it suitable for collaborative software development process is, the issue of lack of clear distinction between activity and action. This can be done by decomposing the activity into hierarchical levels, such that each level comprise of subset(s) of the activity that lend towards achieving the goal of the activity(Uden et al., 2008). Any change in the activity components that does not correspond or contribute to the goal of the activity, represents an undesirable change in the activity.

Even though Activity theory have seen more applications in the social realm, than in the technological realm, it provides a transferrable approach to analysing and conceptualising cloud-based software development. (Núñez, 2009; Bedny & Harris, 2005; O'Leary, 2010; Elizabeth, 2013; Frans Prektert, 2006). There have been quite a few successful adoptions or applications of this theory in the domain of software development(Dennehy & Conboy, 2016; Barthelmess & Anderson, 2002b; de Souza & Redmiles, 2003; Georg, 2011; De Souza, 2003; de Souza & Redmiles, 2003). Reasons for making a choice of a theoretical basis is not solely based on the objectivity of the selected theory or model when representing a phenomena, but rather on theory suitability when it comes to shaping or analysing a phenomena for the identification of issues or gaps (Barthelmess & Anderson, 2002b).

To aid effective adoption of Activity theory as the most appropriate theoretical basis of choice for software development in the cloud, the process is viewed as an activity. Activity theory is known for its suitability for structuring and conceptualising activities and human practices(Said et al., 2014). Prior related work(Dennehy & Conboy, 2016; de Souza & Redmiles, 2003; Georg, 2011) have also taken and justified this view, although not exactly in the same direction as this project. Programming is often viewed as a personal activity, but, the development of large scale software is considered a collaborative activity (Barthelmess & Anderson, 2002b). This is understandably so, because, the level of complexity involved in developing such software requires a team of people with different individual skillsets and skill levels, necessary for performing various tasks geared towards achieving a common goal (Ghezzi et al., 2002).

With the advent of cloud computing, teams can be distributed, collaborating on different development activities within a single project. The use of Activity theory as the underlying basis helps in identification and classification of a broader range of influential factors capable of affecting cloud-based collaborative software development. This in turn ensures that all the right parameters can be factored in at the different stages of the collaborative development process where they occur. The downside is that the wider or broader the classification, the more the chances of complexity within the process.

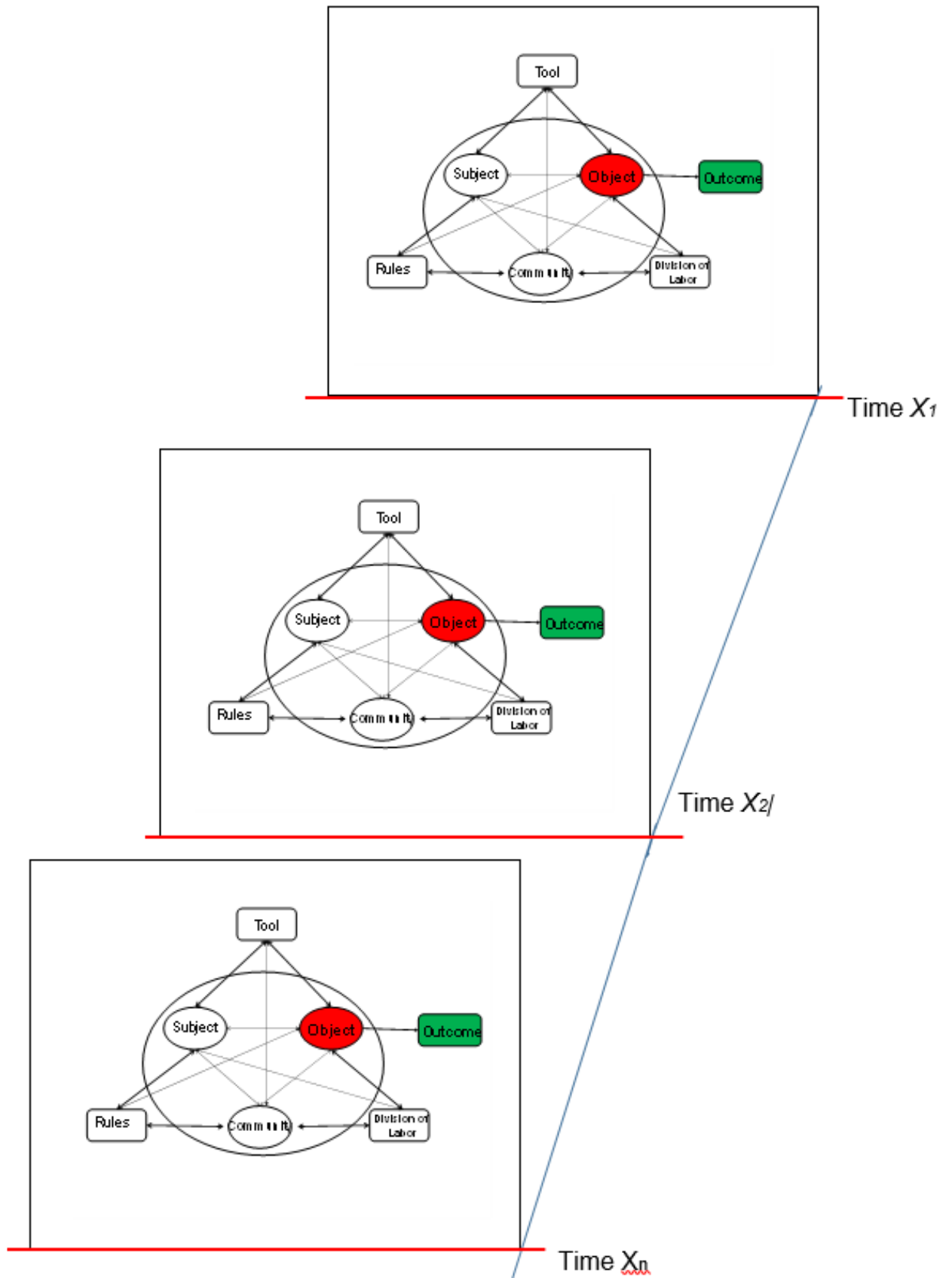


Figure 30 A representation of time context between activity instances

5.7 Developing an AT-based framework for enhancing context-aware collaboration in cloud-based software development

Adapting Activity theory to model collaborative software development process, helps in context-specific analysis of activity systems within the process, thereby providing basis for creating a viable blueprint that supports context-awareness and collaboration (Méndez Fernández & Passoth, 2019). Though the introduction of context-specific analysis by Activity theory extends the problem scope, it helps to flag up irregularities, inconsistencies, and other factors which might impact an activity. For example, with the second and third Activity theory generations, came realization of need for tools to: aid understanding and communication of multiple perspectives; and formation of effective interacting activity systems networks (Engeström, 2001).

Most existing collaborative software development tools and platforms are mostly paradigmatic and lack solid explicit theoretical underpinnings and foundations (Méndez Fernández & Passoth, 2019). The cloud is one of such platforms that possess capability to enable multiple activity instances of a development process concurrently or sequentially, for collaboration across distributed networks. One way of leveraging for better collaboration within cloud-based software development process is, the use of Activity theory for *a priori* structuring and modelling of activity instances to support and improve coordination of the process, and ultimately enhance collaboration (Tell & Babar, 2012; Said et al., 2014).

The approach taken is guided by an interpretive qualitative methodology (Cohen et al., 2009) that attempts explain reality through understanding of software development process activities, and the interactions amongst component parts, within collective and individual contexts.

5.7.1 Step 1: Define use case scenario

The importance of defining the use case scenario lies in the role it plays in the clarification of the goals of the activity system. A well-defined use case scenario helps in the understanding of:

- how the activity system or process is currently envisaged
- what takes place within the activity system or process
- motivations and components of the current activity system or process

Collaborative software development process encompasses the set of joint or complementary activities, engaged upon by various stakeholders (may be distributed or co-located) using support tools (may be decentralized or centralized), throughout the development lifecycle of software to ensure the final goal of developing usable software that meets stated or defined requirements (Mistrík et al., 2010). In order to promote consistent characterization and standardization, the activity scope for software development process is defined by Bourque et al.(2014) as comprising of 7 main phases or classes of sub-activities. These are: software requirements activities, software design activities, software construction or build activities, software testing activities, software maintenance activities, software configuration management activities and software process management activities. To collaborate within these activities, stakeholders must adopt the use of enabling tools, models, and methods for tasks for each activity. Below is a quick summary of each activity within the process.

Software requirements activities refers to activities concerned with requirement elicitation, analysis, specification, validation, and management throughout the software life cycle. *Software design activities* refers to activities concerned with the production of descriptions of internal structures, components, and characteristics of software to be designed, from the analysis of requirements specification. These descriptions provide the basis for software construction activities. *Software construction or build activities* refers to activities concerned with the creation or building of working or usable software. *Software testing activities* refers to activities concerned with verification and validation of software in line with identified requirements.

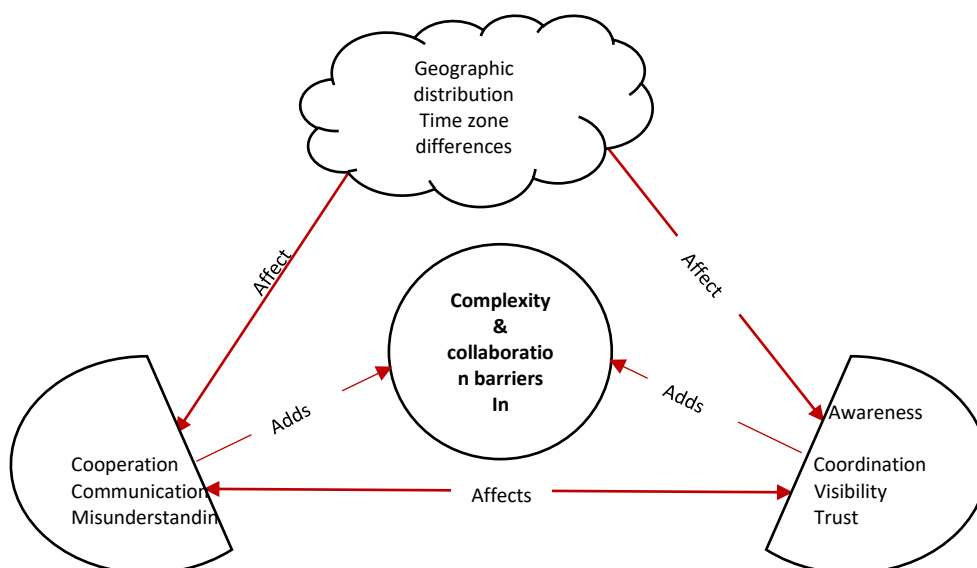


Figure 31 Conceptualizing the problem scenario

5.7.2 Step 2: Define requirements

Activity theory is a conceptual framework with 'Activity' as unit of analysis, making it ideal for: studying activities or activity systems and dynamics of interactions in relation to the objectives; identifying possible congruencies and contradictions emerging from interactions, towards a better outcome (Núñez, 2009; Antoniadou, 2011). The AT-based consideration of the entire cloud-based collaborative software development process is with a primary focus on the following:

- enhancing collaboration in the face of increasing distribution of software development teams and other stakeholders
- effective coordination and management of development teams, all other stakeholders, development activities; team efforts and interactions; large number of resources, artefacts, information, and contexts to ensure delivery of an outcome that meets stated objectives of the software project.
- timely awareness of stakeholder contributions and resulting artefacts
- reinforcing emphasis on the importance of the outcome

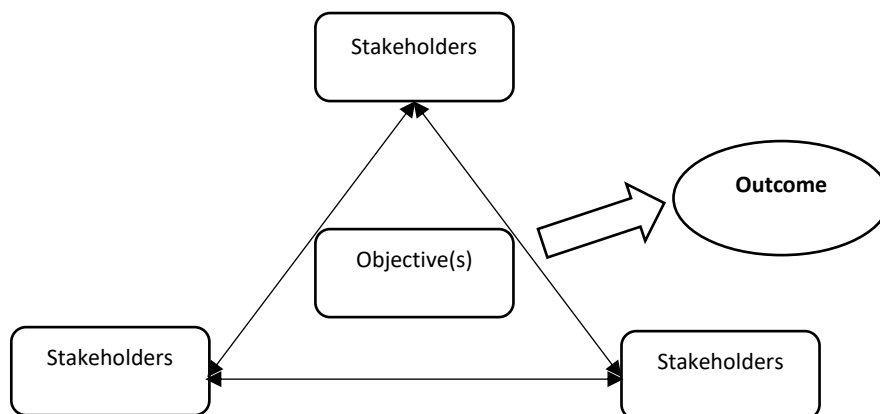


Figure 32 Initial working requirements model for stakeholder collaboration

5.7.3 Step 3: Identify Activity theory concepts/components to leverage

Abstracting and decomposing Activity theory concepts into hierarchical components present one way of approaching complexity; and facilitating generalizations based on analysis of common behaviours and structures. This relies on the assumption that components of related abstraction would logically, mostly exhibit similar properties, structure or behaviour (ed. C. Ghaoui, 2006). However, the perspective provided by Activity theory is very broad and

functions at a macro level (Elizabeth, 2013). One of its focuses is – analysis of how objects are transformed through interactions of components of the activity system and how this transformation is mediated. This focus extends from any one component, across all other components of the activity system and proffers the premise that - the type of transformation that can be directed to an object by any subject or group of subjects, is influenced by the sum of all components and inherent tensions existing within. Therefore, an examination of all components of the activity system becomes necessary for a unified or collaborative activity system.

The components of Activity theory make up a classification scheme or taxonomy that can be used for an activity system, providing class concepts and a template approach for capturing activity contexts within the system, whether team or individual (O’Leary, 2010). This is useful as a foundational basis for mappings between the activity system and Activity theory. However, the drawback to this approach is that Activity theory acknowledges existence of interactions as a necessity for object transformation but lacks clearly defined interaction steps between the subject/community and the object. Due to this reason, and partially due to the abstract nature of Activity theory, another suitable method e.g., object-oriented decomposition method, could be used as an appropriate approach for breaking down Activity theory concepts into abstract components. These components can be viewed as separate entities coming together to enable activity at various levels to achieve transformation of the object into a desired outcome. The activity itself, can be considered an object for other activities, and therefore, included in the decomposition (C. Floyd et al.ed.s., 1992, Mota et al., 1994).

The components of an activity system are dynamic and interact with each other continuously to define the system and the outcome. Therefore, to fully examine or redefine the activity system, consideration of all the components and their interactions is necessary, including any characteristic tensions therein. Tensions refer to the dualities that exist along the value chain. An example of such dualities is as follows: an action is performed, to yield an outcome, or to achieve a goal (Barab et al., 2002). Examining these tensions help to better understand interplay involved within dualities of the activity system. This translates into an ability to better leverage dynamics of the dualities within the activity system, as well as better understanding of ways to support the activity system.

Activity

Human activity is the focus of AT. Activity is defined as any motivated form of action directed towards transforming some object into an outcome or set of outcomes (Barthelme & Anderson, 2002). The activity is defined by the object and its existence is motivated by transformations of the object. Due to this self-regulating integration of motivation and behaviour towards a defined or stated goal, the activity is considered goal directed (Bedny & Harris, 2005). An activity may comprise of one or more related or unrelated child activities which can be realized via actions/tasks; and may be dynamically distributed and re-distributed along internal and external dimensions and components(ed. C. Ghaoui, 2006). Actions can be further decomposed into operations (automated actions). Each level of activity breakdown is driven by a different need and provides a clear separation which helps to fundamentally improve understanding of activities at a more granular level(Tell & Babar, 2012). This, in turn allow the generation of more detail on the activity based on the analysis of the workings of the lower levels of an activity (O’Leary, 2010). Since activities may vary, so also activity breakdown. An activity may comprise of interactions which are either transformation actions or development actions, and can happen inter (between subjects), or intra (within a subject).

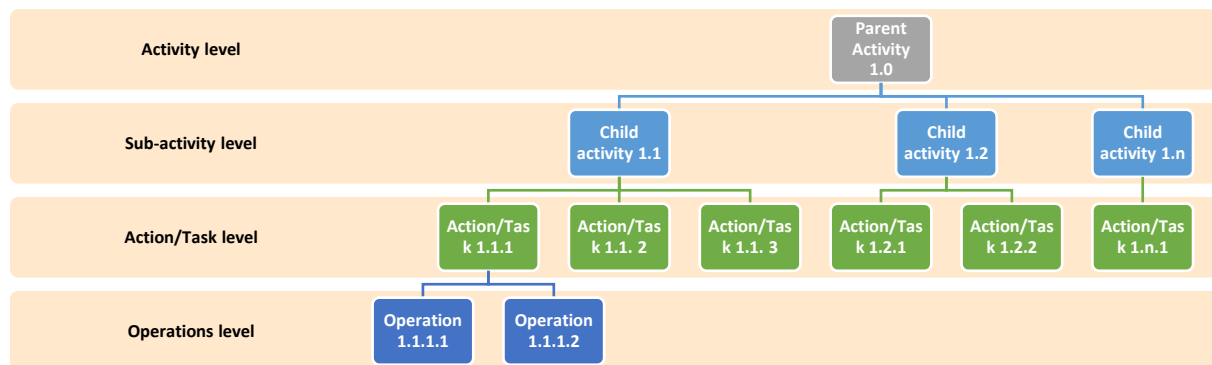


Figure 33 Hierarchical breakdown of the Activity

Subject

This refers to any person or system that performs an action or undertakes an activity (Bedny & Harris, 2005). A subject is usually a part of a collective effort, acting in a role to transform an object using information, tools, or some other artefact. The actions and interactions performed by the participants in the activity system are directed towards the object, resulting in transformations that finally yield the desired outcome. These transformations are mediated by other aspects or components of the activity system. The subject or subjects provide the lines along which activity internalization and externalization are defined.

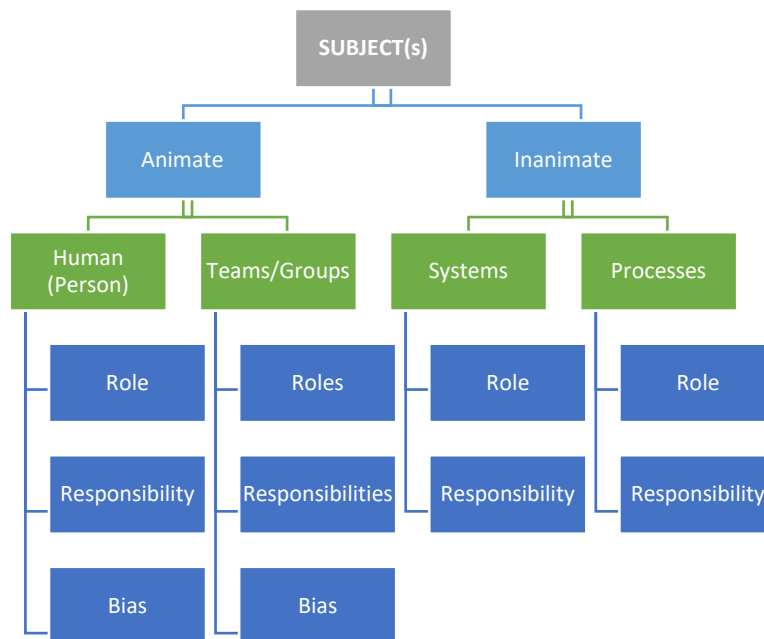


Figure 34 Hierarchical breakdown of the Subject component of Activity theory

Object

This refers to the Activity theory component that is modified or transformed by the subject, in line with the goal of the activity. This reference includes problem spaces, specifications, stated problems or identified needs, conceptual understandings, or any form of raw material that is the principal focus or point of direction of an activity or activities. An object is a structured, discretely existing entity, with objective meanings capable of being wholly or partly determined by relationships with the subject, or other objects. This relationship forms the basis of an activity and provides the lines along which an object should be analysed for a better understanding of the activity. The object can be material, immaterial, or useable

knowledge, defining and motivating actions and goals of the activity system (Engeström, 2005; Bedny & Harris, 2005).

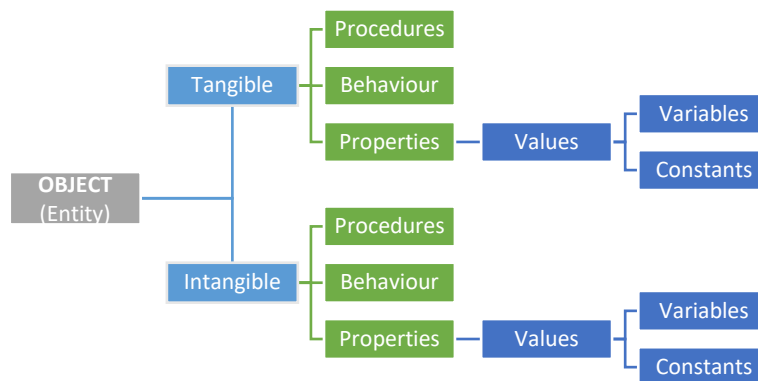


Figure 35 Hierarchical breakdown of the Object component of Activity theory

Tools

Refers to physical or non-physical artefacts that shape interactions and transformations through either capture or use of information, or influence on behaviour. Tools can also be shaped by experience, practice, or culture. Tools provide enhanced capabilities and interactions, or limit capabilities and interactions. Tools can be external, internal, physical, and symbolic. Together with object, tools provide the cultural and historical contexts of the activity.

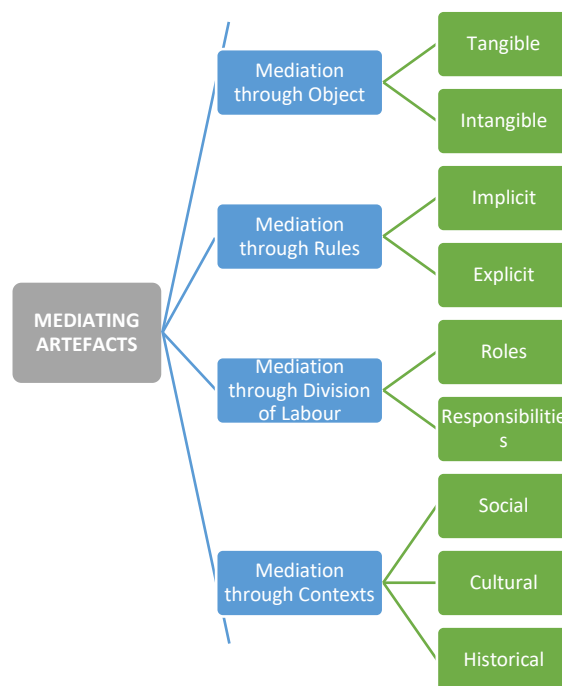


Figure 36 Hierarchical breakdown of the Tool component of Activity theory

Community

This refers to all the stakeholders of an activity i.e., all the people involved in the activity for the entire lifecycle of that activity. These usually share general or common objects, and together with the subject, provide the social contexts of the activity

Rules

This refers to all conventions, constraints, guidelines, policies, regulations, processes, methodologies, specifications, configurations, logic, and standards that guide the activity and every aspect of the activity.

Division of labour

This refers to the structuring or organization of activities and component parts to ensure balance. Division of labour refer to how tasks are distributed and run, both vertically and horizontally along the lines of equality status and hierarchy. This could also be non-hierarchical.

Goal

One school of thought proposes the existence of the goal concept as a way of representing the expected result or expected outcome of an activity (Bedny & Harris, 2005). The goal drives the activity and is embodied by the outcome or result of the activity. This can be material or immaterial. Goals represent a future state and can be modified at any point in time during the lifecycle of the activity.

Outcome

The outcome constitutes the result of the activity and represents the embodiment of the goal of the activity (Bedny & Harris, 2005).

5.7.4 Step: Mapping of Activity theory concepts to cloud-based software development aspects

The adoption of activity theory as a theoretical basis is based on the following premises of the collaborative software development process (Barthelmess & Anderson, 2002b):

- software development process is inherently, a collaborative activity geared towards a defined goal or set of goals, and normally involving one or more persons, supported by resources and techniques e.g., techniques, tools, etc.
- The various resources involved in a typical software development process can influence the process, or/and the outcome or goal. Hence the need to analyse and evaluate them in terms of capabilities and impact on collaboration.
- The development process relies on artefacts to mediate the entire process and its interactions

Applicable postulates of AT to the collaborative software development process are as follows:

- Activity is the unit of analysis.
- Activity comprises of subject and community, interacting with an object through goal-directed actions, towards achieving an outcome.
- Rules govern the collective and individual actions of the subject and community.
- Tools support actions & interactions of the subject and community.
- Actions & interactions of the subject and community execute in defined structures or patterns – division of labour.
- Inseparability of the mind from activity
- Rules, tools, and division of labour components mediate interactions between the community and subjects, as well as interactions between community and object.

This research adopts and focuses on the above postulates of Activity theory, specifically chosen to aid theoretical analysis and understanding of cloud-based collaborative software development, and gain insight into the process. These extracted postulations encapsulate a set of principles that provide a conceptual system for explaining phenomena such as collective work and for accounting or justifying actions and interactions. Furthermore, Activity theory provides a framework for holistic analysis and consideration of the entire cloud-based collaborative software development process. The primary focus of this collaboration include:

- increasing distribution of software development teams.
- large number of resources, artefacts, information, and contexts at play in any sizeable software project.
- importance of the end goal

The Activity framework enables examination at a more macro level of the collaborative software development process and its entire makeup: teams/groups; communications, interactions in socio-technical and cultural contexts, relationships, historical factors, information, motivations; and artefacts, and goals/outcomes. All these can be studied, patterned, and used for inferential purposes, as well as expanding knowledge.

Table 17 Mapping AT to collaborative software development components

ID	AT COMPONENT	DESCRIPTION	EQUIVALENT SOFTWARE DEVELOPMENT COMPONENTS
1	Activity	Refers to the instance unit of analysis. It comprises of all the other components below	A project instance of the software development process
2	Transformations	“relates to a defined tasks/actions towards a defined goal”(Bedny & Harris, 2005)	Software development tasks within the project including requirements analysis tasks, designing tasks, build/coding tasks, testing tasks, deployment tasks etcetera.
3	Subject	refers to individual, group or subgroup chosen as point of view of activity analysis(Dennehy & Conboy, 2016)	Include development or cross functional team members: analysts, designers, developers, testers, manager
4	Object	Refers to problem space/definition or material which is shared and transformed into outcome(s) (Dennehy & Conboy, 2016)	Includes: requirements, designs, specifications, code, and artefacts within various phases of the process
5	Tools	Refers to different tools, environments, and signs used for transforming the object(Dennehy & Conboy, 2016)	Includes the cloud platform or development environments, other cloud resources or third-party tools needed or used in the development process, as well as development environment
6	Rules	refers to requirements for membership of community, and conventions/logic constraining interactions and actions within activity system(Dennehy & Conboy, 2016)	guidelines, standards, logic, and conventions used for instantiating and managing interactions/practices within the process

7	Community	Refers to group or sub-group of multiple individuals sharing same general object or goals(Dennehy & Conboy, 2016)	Grouping of users, software owners, focus groups, analysts, developers, testers, designers, project manager etcetera.
8	Division of labour	Refers to horizontal or vertical division and distribution of tasks(Dennehy & Conboy, 2016)	One-to-one task assignment or one-to-many task assignment to members of the software development team depending on complexity of task and availability of resources.
9	Outcome	Result of the activity. This can be material or immaterial. (Dennehy & Conboy, 2016)	Refers to the final outcome of the process e.g., feature, functionality, software, or knowledge artefact

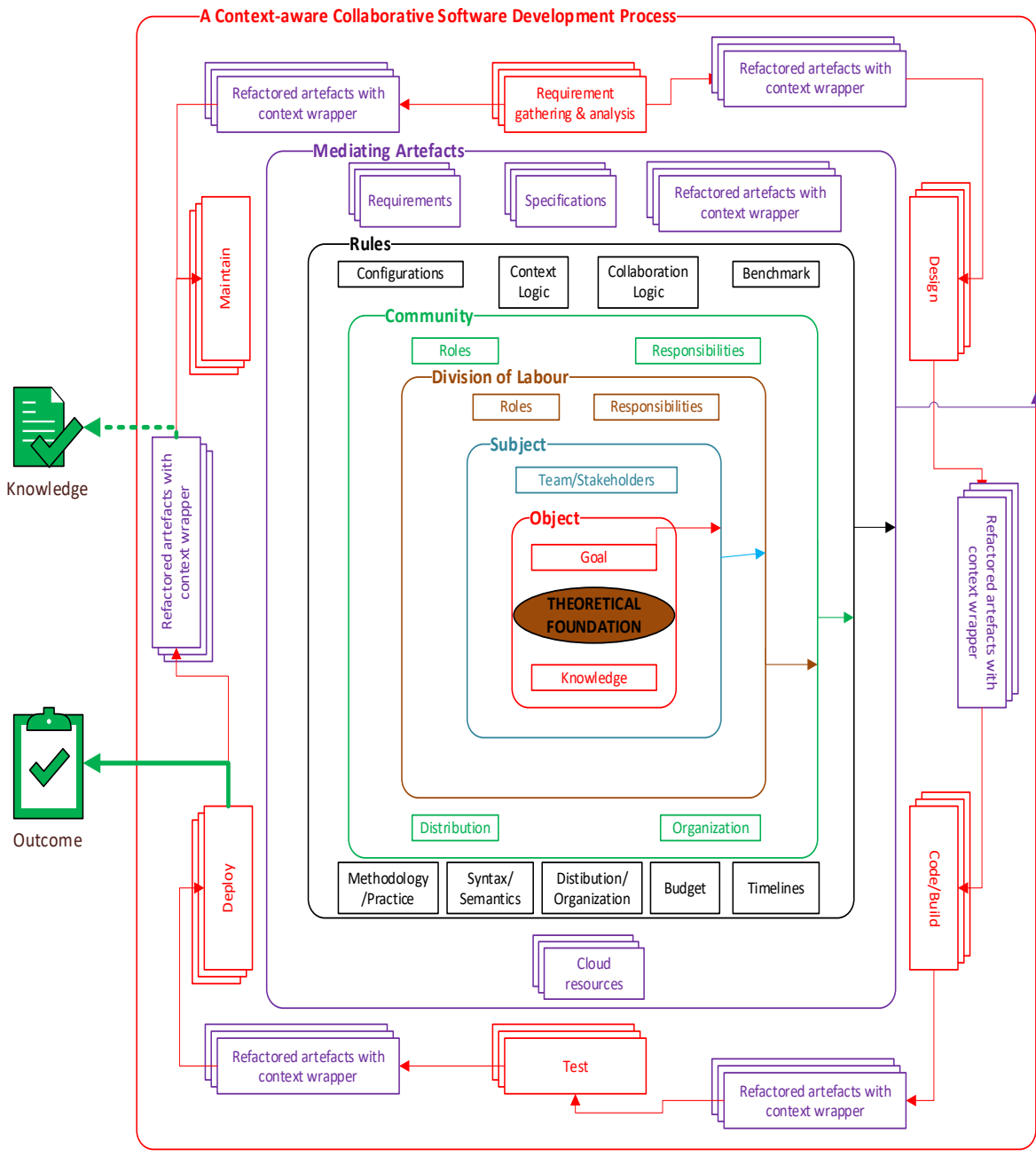


Figure 37 Visualizing future context-aware collaborative software development process enabled at the core by solid theoretical foundation.

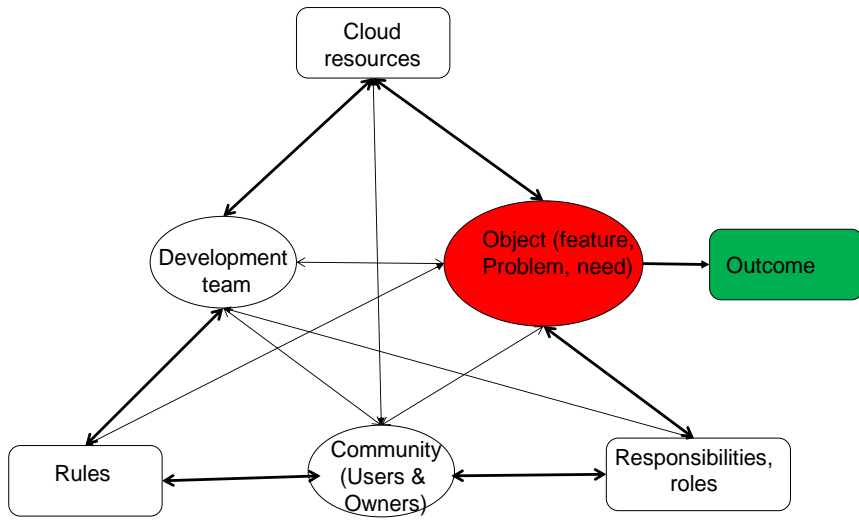


Figure 38 Adopted mapping of main software development process components to activity theory components.

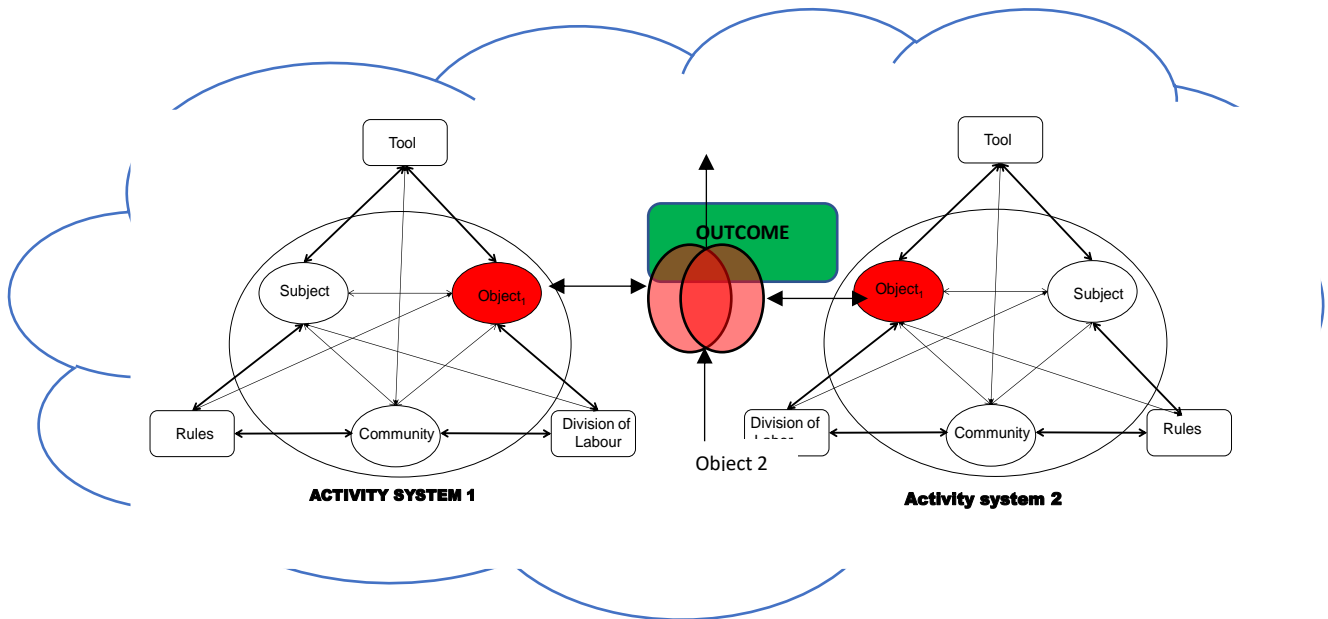


Figure 39 AT-based conceptualization of distributed interacting activity systems in the cloud

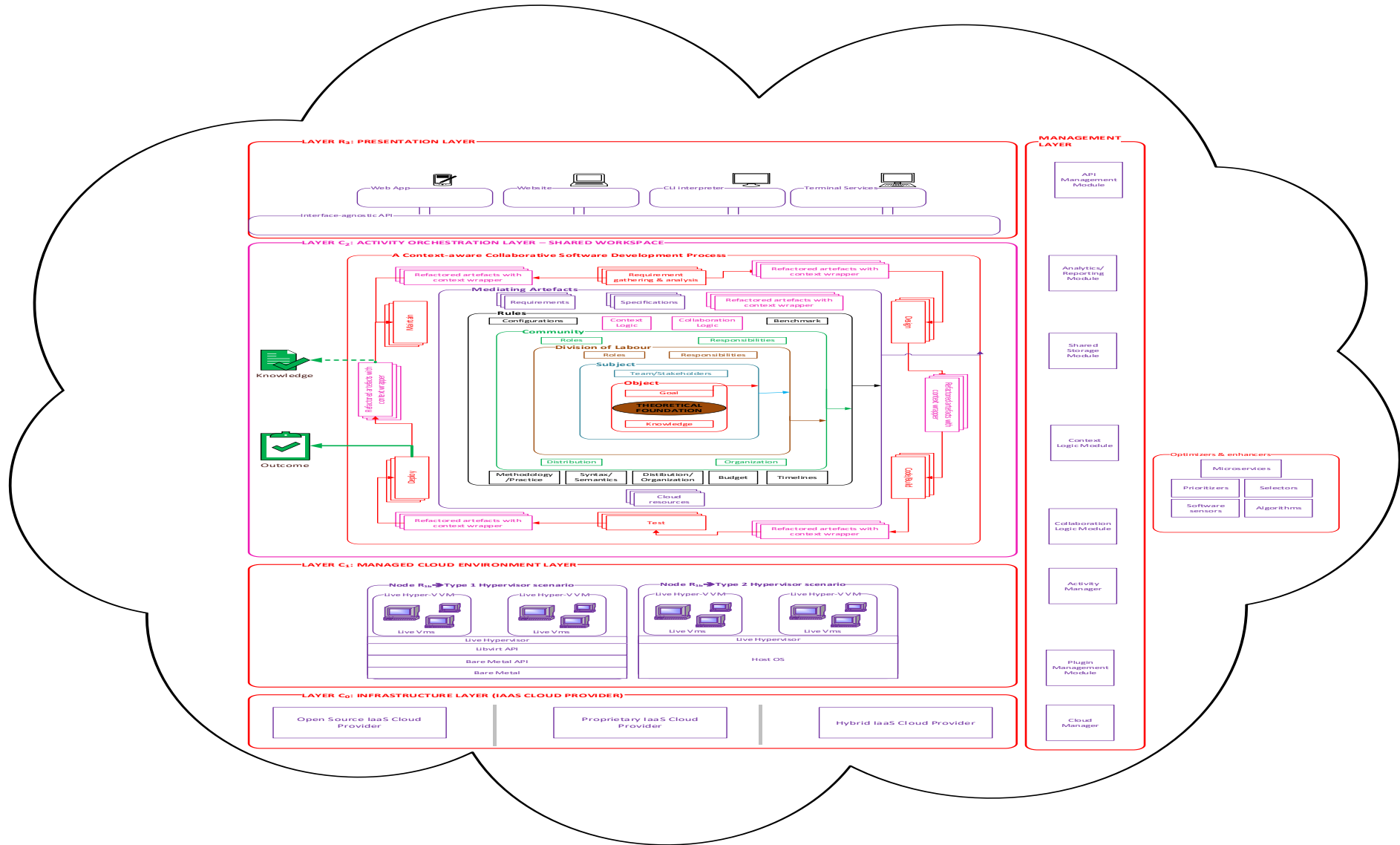


Figure 40 Mapping proposed theoretical framework - Initial architecture block diagram for existing system.

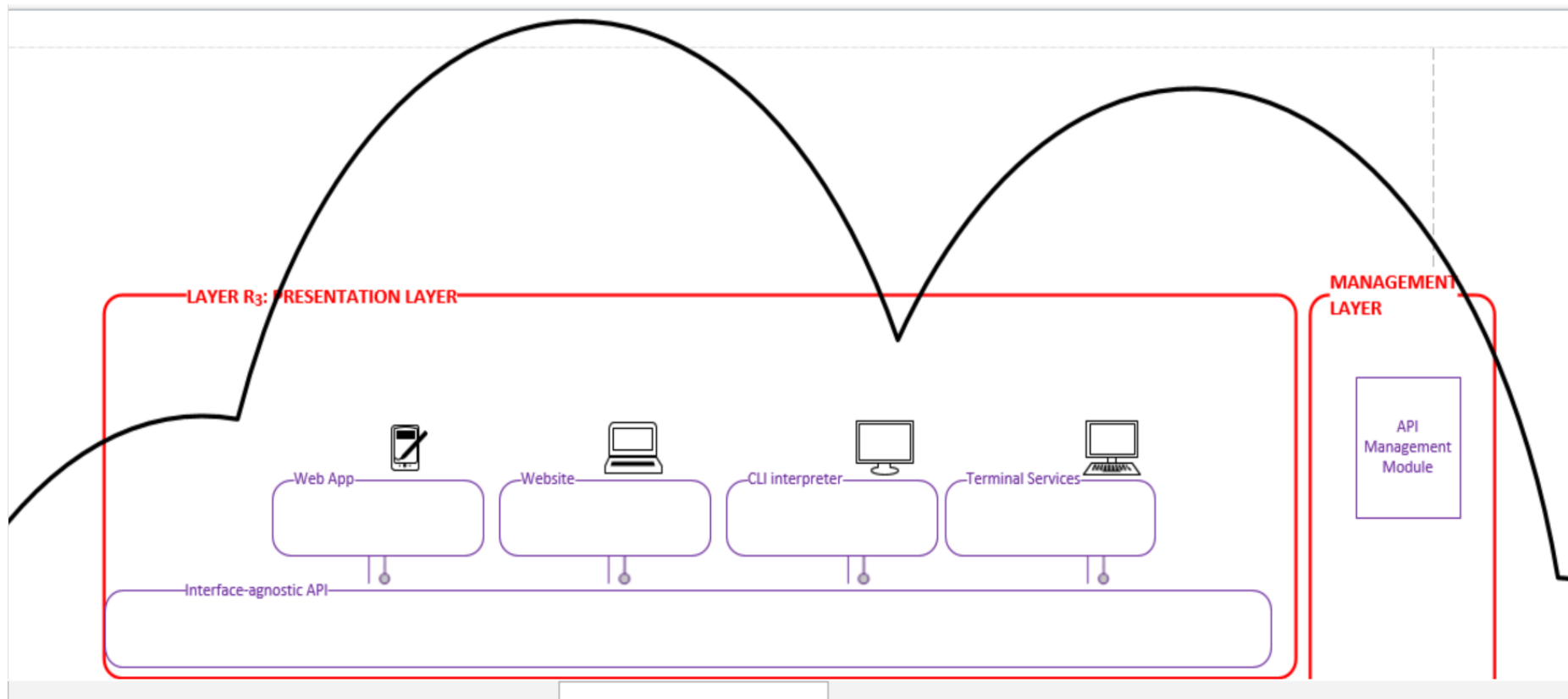


Figure 41 Zooming in on Mapping proposed theoretical framework - Initial architecture block diagram for existing system: Presentation layer.

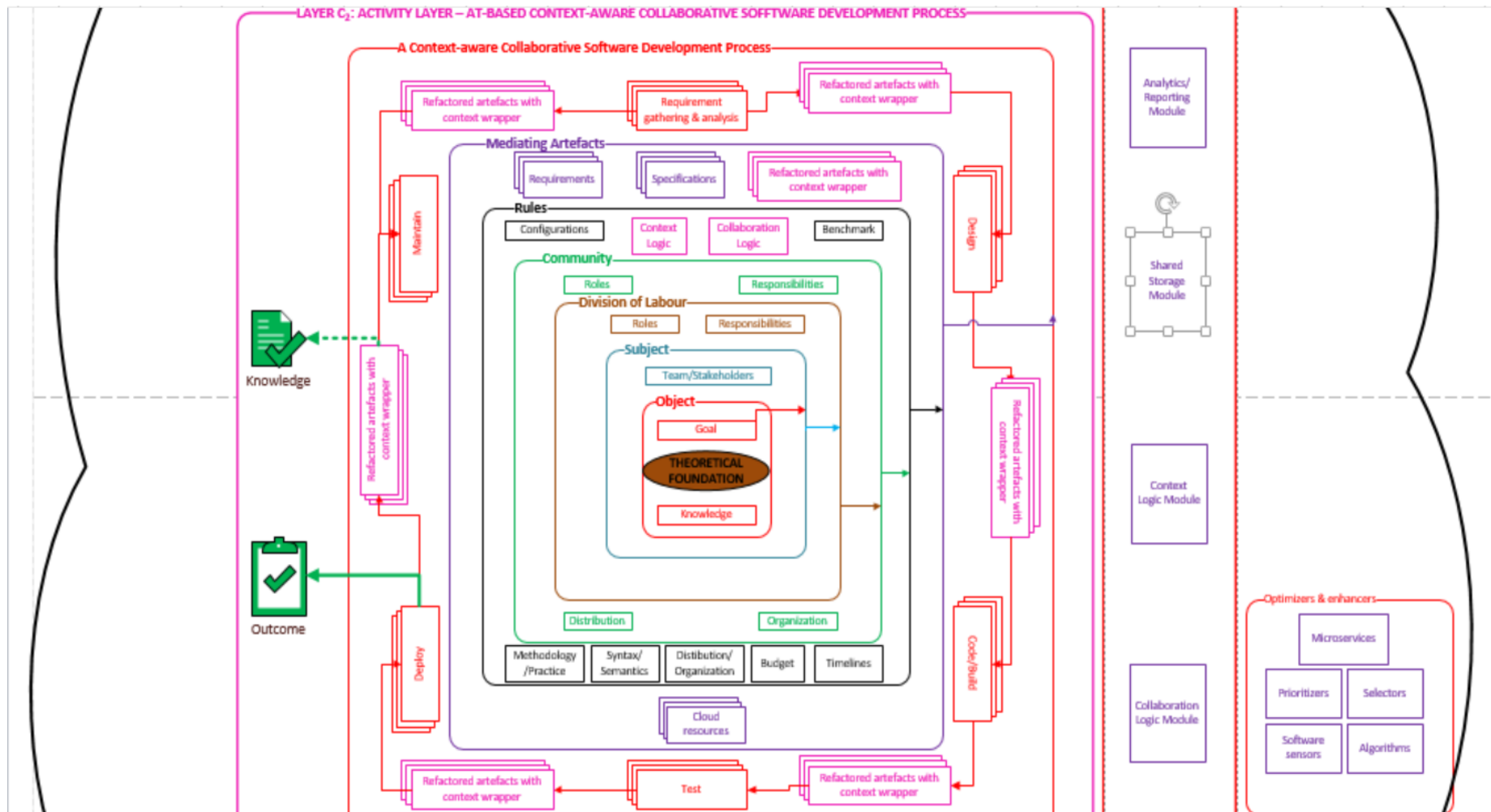


Figure 42 Zooming in on Mapping proposed theoretical framework - Initial architecture block diagram for existing system: Activity layer.

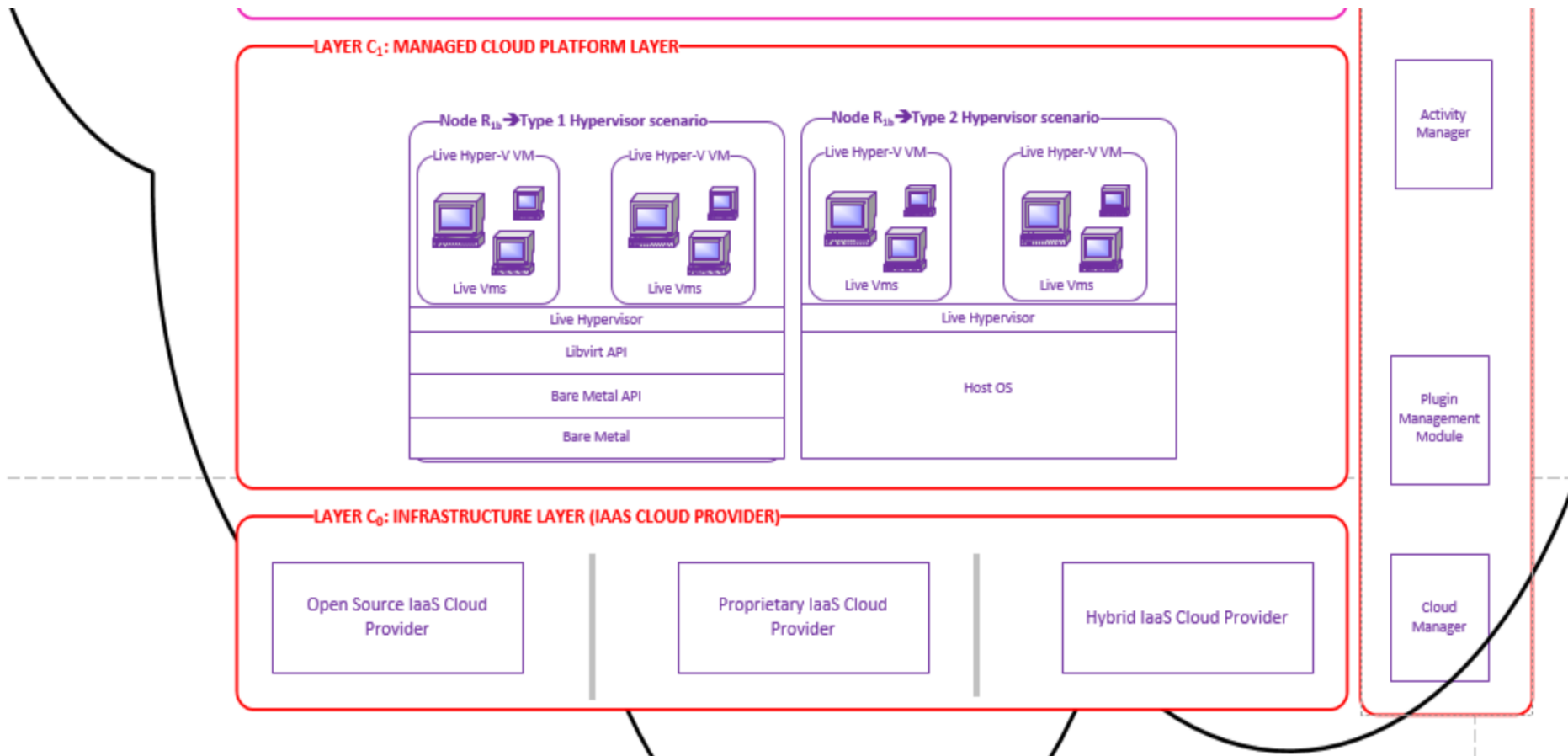


Figure 43 Zooming in on Mapping proposed theoretical framework - Initial architecture block diagram for existing system: Managed cloud platform layer.

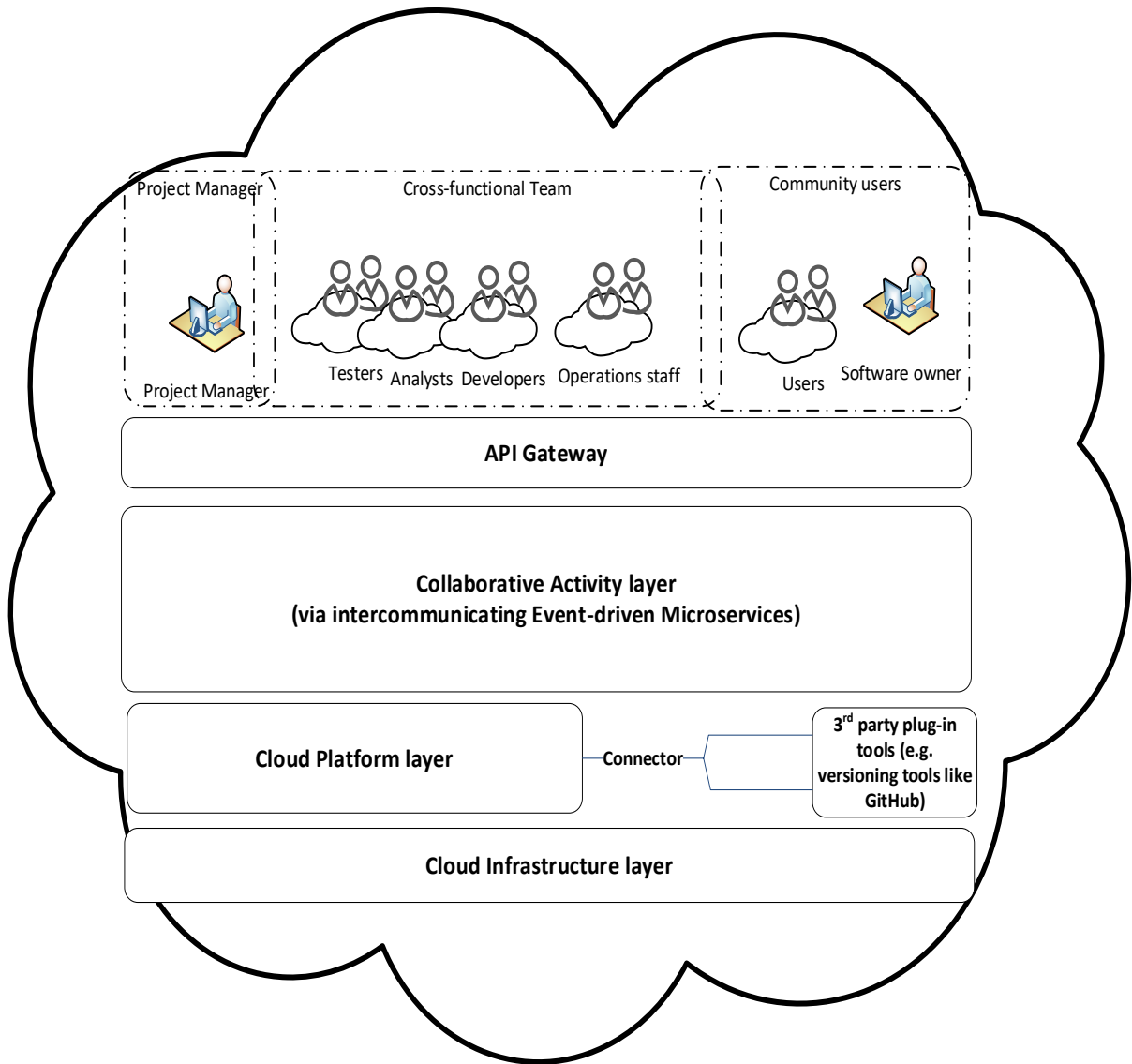


Figure 44 Mapping proposed theoretical framework to existing system – architecture block diagram.

5.7.5 Step: Define baseline activity structure for collaborative software development process

The collaborative software development process is directed by and based on objectives/object of the activity and aided by tools. The rules are binding on subjects within the community in which the activity takes place, in a pattern aligned with the roles and responsibilities of the subject, or subjects within the community. This ensures that the activity is analysed and carried out within all related contexts. A key requirement of activity theory is analysis of activities within contexts to provide a deeper understanding of the object, which is key to showing object transformations(Uden et al., 2008). However, because interactions within the activity uniquely differ from object to object, based on the objective, type, or level of transformation or action, the implication is lack of a prescribed method for this kind of analysis. This implies that similar objects with similar interactions will have to an extent, similar contextual activity analysis. Therefore, enabling analysis of complete object transformation trajectory improves understanding of the object, and could act as predictive input with regards to any further or future transformations. To do this would require definition of a baseline activity structure; as well as a way of capturing contextual information relating to the object, and other components interacting with the object.

The goal of an activity can be met via a variety of actions, or/and tasks, which in turn, can contribute to other activities. In the same vein, actions may require operations that meet certain conditions that help to meet the goals, and in turn, operations can contribute to actions. However, a distinction is made between tasks and actions. Tasks can be regarded as conscious and goal-driven actions, whereas, actions are dependent on operational conditions of tasks, and hence, can become routine or become automatic through defined constant or routine use(Uden et al., 2008). This distinction between tasks and actions reveals that a task is not dependent on an operational layer but can specify operational conditions for actions. The distinction between motives, goal and conditions, as well as distinction between activity, tasks and actions, provides a means of defining activity levels and relationships, which can in turn, be collectively analysed, by using and integrating the viewpoints of the contexts introduced by the second generation of activity theory(Uden et al., 2008).

However, though operational elements e.g. environments, are not directly related to the goal of an activity, it is necessary and recommended that the form of the activity be adapted to

them(Wolff-Piggott & Rivett, 2016; Bærentsen & Trettvik, 2002). An activity orientates towards a goal. The objectives of the goal correspond to what subjects of the activity need to attain. Goal-directed actions constitute how subjects attain the goal, and these actions are implemented via automatized or conscious improvised steps

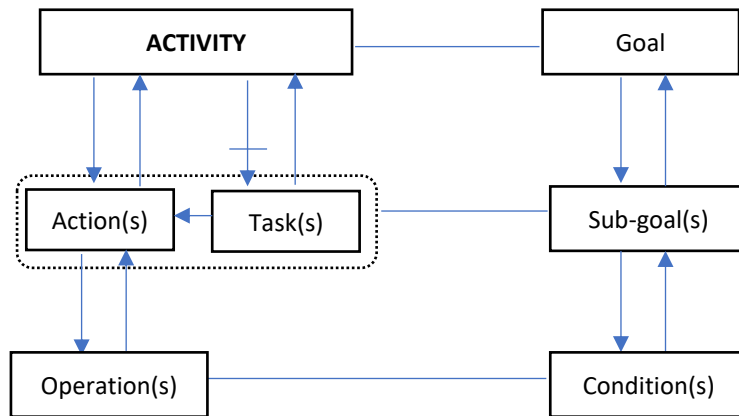


Figure 45 Distinction between Activity levels

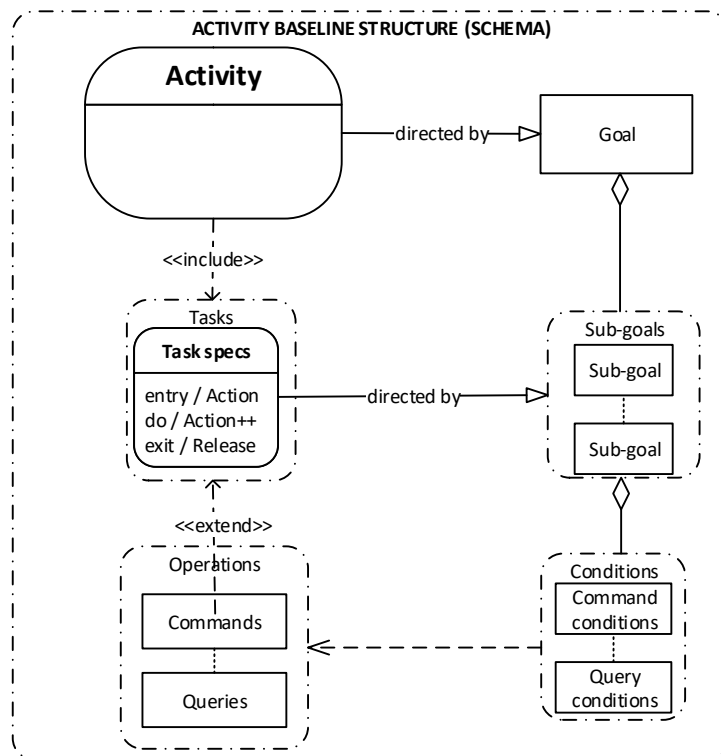


Figure 46 Proposed Activity baseline structure (schema) for designing/creating a collaborative software development activity.

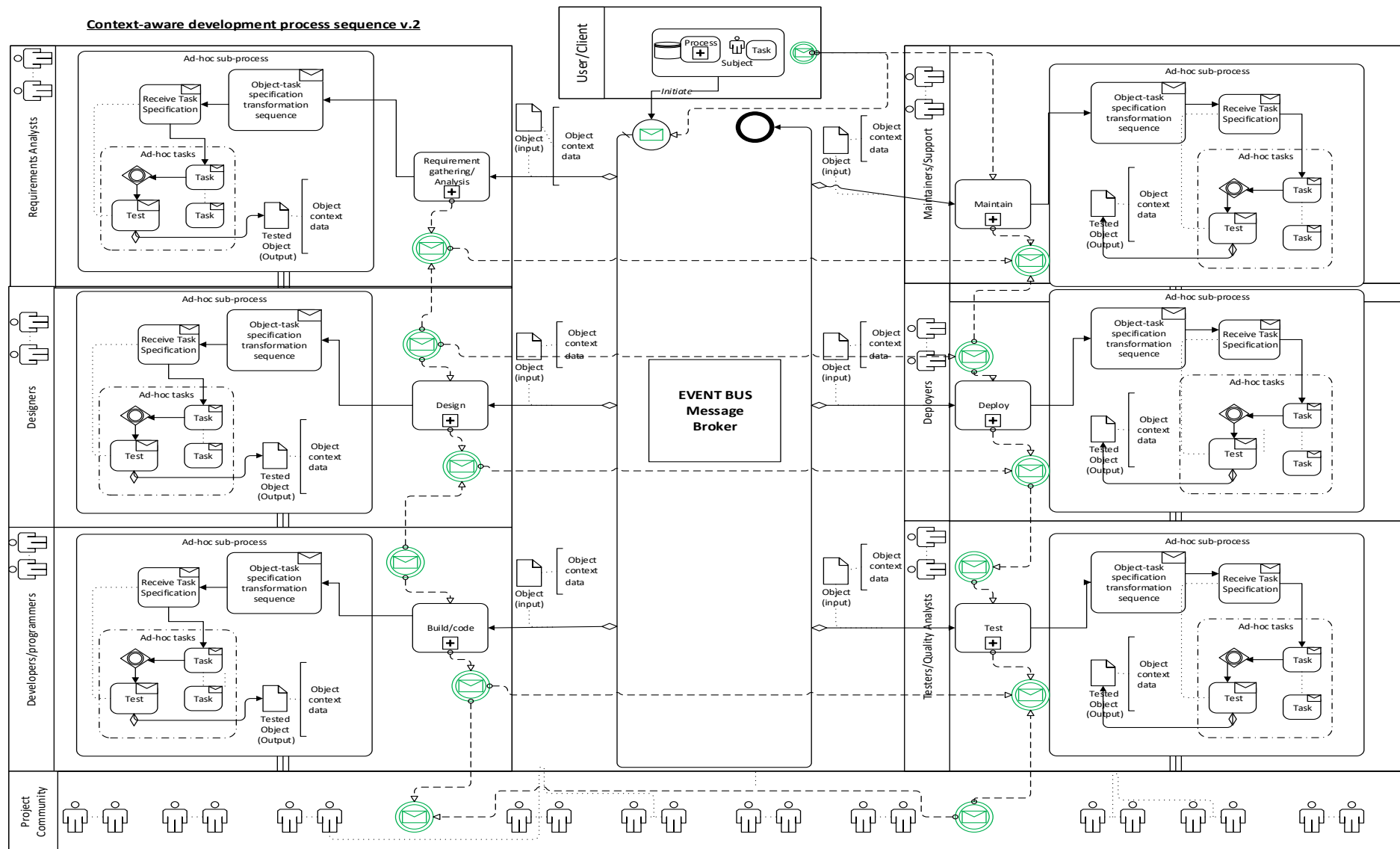


Figure 47 Modelling context-aware development process sequence

5.8 Summary

The proposed process for selecting appropriate theoretical basis as outlined above serves to provide a formal and systematic approach. The target is to deliver an optimal choice of theoretical foundation (i.e., a choice which competes favourably with any other available option in a consistent manner) for architectures for cloud-based software development. This process can be adapted for application in other domains or projects. The proposed process may prescribe more than one suitable outcome, depending on the given parametric model, input and method of analysis applied. At first glance, this process may appear to be more complicated than the existing ad-hoc methods of adopting theoretical foundations, but it presents a more empirical and reproducible method that is applicable in any domain. A case study example is used to demonstrate how the process may be applied. This application demonstrates the usefulness of this proposed process as a detailed empirical prescription of theoretical behaviour. Application of selection process resulted in the identification of activity theory as a more appropriate theoretical framework for collaborative cloud-based software development process. The tenets of AT discussed in this Section can be reified using any arbitrary number of models or frameworks. These can be designed and modelled to be representations of the desired principles and characteristics for collaborative cloud-based software development process.

6 The Architecture

6.1 Introduction

In the previous Sections, a systematic review of literature was carried out to identify and extract gaps, themes and other relevant building blocks including theoretical foundations, necessary to form underlying constructs for proposal of a suitable Cloud-based architecture. Main identified gaps and addressed in this research include: underestimated or ignored complexities and contexts that end up undermining collaboration in the process; motivations include: need for identification of reliable ways of managing collaborative activities and managing complexity within the process; ways to ensure synchronous regularity; and need for sound theoretical methodologies for enhancing effective collaboration to ensure verifiable and quality outputs and outcomes at all stages of the development process (Ewenike et al., 2017a, 2010). Being able to consistently reproduce the enhanced development process would require standardization in the form of frameworks, architectures and standards (Benedek & Lajos, 2012). Summary of main concerns from literature, along with recommendations and anticipated benefits are discussed in the literature review Section. The main gaps to be addressed by architecture are summarised in table 18 below and published in conference transactions captured below.

Table 18 Summary of gaps in cloud-based collaborative software development (Ewenike et al., 2010, 2017b)

Main identified gaps addressed	Comments	Observed impact include:
<p>➤ Need for cloud-based, context-aware collaborative software development architectures, with explicit theoretical foundation</p>	<p>➤ Emerging technologies change the way software is accessed, utilized, stored, and maintained. They introduce or emphasize new considerations such as: distribution, more complexity, and more contexts.</p> <p>➤ There is need to develop reliable software for continuous adaptation to changing requirements.</p>	<p>i. Randomness in the science of Software Engineering process</p> <p>ii. Undermined collaboration in collaborative software development process</p> <p>ii. Emphasis on need for better and sustainable frameworks, architectures, tools, and strategies, with explicit theoretical foundations for more</p>

	<ul style="list-style-type: none"> ➤ Current innovative solutions rely on results from mix of successful and failed implementations, as well as glitches. 	<p>structured adaptation and sustainable collaboration</p> <ul style="list-style-type: none"> v. need for adequate methods for managing change in the cloud-based development process in the cloud, and knowledge creation
<ul style="list-style-type: none"> ➤ Need for effective capture and representation of context data and all related data across entire development lifecycle in a cloud-agnostic format for generation of actionable insights 	<ul style="list-style-type: none"> ➤ Insufficient context data and other related data are sometimes poorly collected, completely missed, ignored, misunderstood, or poorly applied ➤ Requirements, artefacts from various activities, action plans, feedback, and other important related information necessary to achieve a goal are sometimes not clearly and accurately defined, and agreed upon by all concerned 	<ul style="list-style-type: none"> i. negative impact on balancing and optimising of flow of information within the development environments and teams. ii. late detection and resolution of issues and bugs that could have been otherwise avoided if enough context data are collected, and taken into consideration and applied within activities. ii. inadequate tracking of project progress. v. conflicts in perspectives, understanding, interpretation and execution of activities. This often results in defective software, or software needing more rework
<ul style="list-style-type: none"> ➤ Need for effective ways for managing complexity across stages in the lifecycle of cloud-based development process to ensure synchronous collaboration and verifiable outputs/outcomes at various stages of the process 	<ul style="list-style-type: none"> ➤ Certain disciplines such as the engineering disciplines, are usually guided, constrained, and regulated by physical laws that ensure regularity and a way of keeping complexity in check. Conversely, Software Engineering is not easily regulated by physical laws. 	<ul style="list-style-type: none"> i. Growth in complexity of software artefacts and the lifecycle process ii. Differences and difficulty in understanding, developing, and testing in the right way and correctly.

		ii. Increased need to challenge and validate results via some form of empirical effort
--	--	--

These gaps identified above, comprise the requirements which form the basis for the top-level use cases for the proposed architecture. The reasons for translating the requirements from the gaps into use cases include - consolidation of requirements into simplified use case; to aide development and mapping of functionality to address requirements; elimination of redundancies; to aid prioritization of phased implementation of functionalities; and to aid and simplify explorative study.

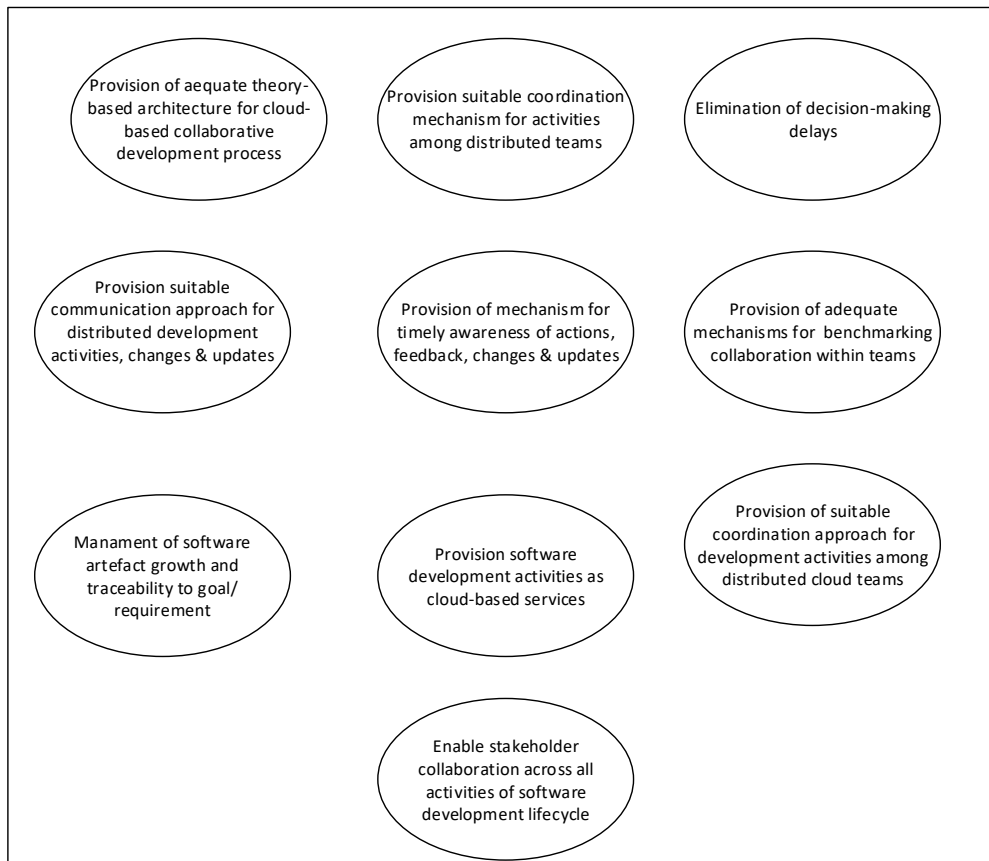


Figure 49 Prominent identified use cases for the proposed platform

6.2 Overview of an Architecture

An architecture refers to the abstraction structure for mapping process functionality to systems or system components or modules - helping to realize requirements specifications (Gerber, et al., 2006). It represents the foundational structure of a system - an abstract depiction of a system (Clements et al., 2001). An architecture is made up of components or

elements, with relationships and constraints that guide the functions of the components. Pressman (2004) defines an architecture as a “comprehensive framework that describes the form and organization of its constituent components and their organizations”. It can be viewed as a model, an abstraction that represent views of significant aspects of a real-world process or system (Bass, et al., 2003; Pressman, 2004).

The absence of a “clear and well-defined” formal architecture for the cloud-based software development process results in the process resorting to a de-facto standard type architecture – the n-tier architecture. In this type of unplanned architecture, source code modules are split into packages. These packages are implicit layers that are not well organized and planned with considerations for clear roles and relationships to one another (Richards, 2015). The result of this includes tight or close coupling of components, lack of clear direction or vision, difficulty in managing change, etcetera. Considerations when building a formal architecture should include architecture pattern or style, components along with features or properties, and relationships between the components (Obrenovic & Starcevic, 2006; Fowler, 2003).

Another key consideration when building an architecture is to allow logical reasoning of constraints, key requirements and any subsequent modifications to ensure sustainability (Gonzalez-Huerta et al., 2015). Clearly defining key consideration aspects is central to ensuring proper development and provision of adequate runtime support for the system or process for which the architecture is developed (Georgantas, et al., 2011). These aspects help to specify the right abstractions and abstraction level needed to model the proposed architecture.

Without a formal architecture design prior to implementation, there is a risk of developing software applications or platforms that are just collections of modules packaged together but lacking clarity in terms of roles, responsibilities and relationships between modules (Newman, 2015). This is known as an architecture anti-pattern. Other impact of lack of a formal architecture pattern include brittle and tightly coupled components; difficulty in effecting change; consideration of factors such as dependencies, change responsiveness, deployment, and performance characteristics, etc. Formal architecture patterns help to provide: justification for architectural decisions and choices; performance assurance through identification and definition of basic characteristics, behaviour, strengths and the weaknesses

of the software application or platform to be developed, that best meets identified business requirement needs and goals(Mistrik et al., 2016; Richards, n.d.).

6.3 Review of architecture patterns

A pattern is a description of repetitive problem in a specific setting, as well as, its reusable solution core(Richards, 2015b). Architectural patterns refer to architectural recurrences within architectural models that provide an approach for defining behaviour, basic characteristics and functions of an application or system(Richards, 2015a). These patterns provide a best-practice solution to recurring problems in a specified environment.

To choose an appropriate architecture pattern for designing an adequate architecture for a process, it is necessary to understand the weaknesses, strengths, and characteristics of different patterns via a review of common architecture design patterns (Richards, 2015; Gonzalez-Huerta et al., 2015). Richards(2015b) discusses and compares five common software architecture patterns based on the following: pattern description, key concepts of the patterns, examples illustrating the application of these patterns, and key considerations to bear in mind; as well as a pattern analysis. The table below provides a comparative summary of common software architecture patterns and provides an analysis scoring based on pattern considerations such as agility, scalability, development, testability, deployment, and performance. The analysis scoring helps in determining patterns best suited for a formal architecture development; and provides guidance towards making and justifying architectural decisions for collaborative software development process in the cloud. Reasons for a formal architecture include:

- Clear definition of basic characteristics of a system or process
- Clear definition of behaviour of the system or process
- Management of complexity
- To answer questions about deployment, scale, performance characteristics
- Change management
- Understanding, description and further development of multifaceted systems

6.3.1 Layered architecture pattern

Architectural components are arranged into n layers with specific roles and responsibilities. Examples of common layers include presentation layer to handle browser and user interface

logic and communication; business layer for handling user request-associated business rules, etc. Benefits of this kind of organization include separation of concerns between components – each layer has a specific role/responsibility allowing ease of introducing new roles and responsibilities in the future. The figure below illustrates a layered architecture pattern.

6.3.2 Service-oriented architecture (SOA) pattern

This pattern lends itself to the organization and utilization of distributed capabilities under the control of different domain ownership (Kreger & Estefan, 2009). SOA patterns provide a framework that matches needs to capabilities and combines matched capabilities to address needs. In SOA, bringing together needs and capabilities is done via a mechanism referred to as a *service*. *Interaction*, *visibility* and *effect* are among the key concepts of this pattern (Hodges, 2002). These concepts provide entities within an SOA pattern, access to capacity that enables them to **see** and **match** needs to capabilities (and vice versa). This is done via accessible and understandable description of requirements, functions, related constraints, policies, and access and response mechanisms. *Interactions* are mediated by message exchange and invoked actions while service descriptions promote *visibility*. The result of these two concepts yields an *effect*. SOA is task-focused and promotes reuse of capabilities or solutions, service visibility, interaction support, interoperability, ease of growth and modification. An SOA architectural pattern is commonly implemented with the aid of web services.

6.3.3 Microservices pattern

This is an architectural pattern comprised of loosely coupled service elements having bounded contexts. This architectural approach decomposes a process or application into service components that form the unit of modularity for the architecture, and each service component focuses on a cohesive set of responsibilities (Richardson, 2019). This cloud-native architecture pattern facilitates development of processes and applications as suites of small, self-contained, task-oriented services that are easy to understand, independently scale and deploy, and communicating via lightweight mechanisms (Cerone & Roveri, 2018). In the microservices architecture pattern, an application or process can be scaled or decomposed using the three-dimensional scale cube model (Ref – Martin Abbott and Michael Fisher, 2015: The art of scalability). This model defines the following approach for scaling: scaling along the X-axis balances process or application requests across multiple instances of the process or

application; scaling along the Y-axis functionally decomposes the service or application into functional service components; and scaling along the Z-axis routes process or application requests based on the attribute of the requests (data partitioning).

The microservices architecture pattern provides a useful reference frame for building reusable self-contained services around functions or capabilities, accessible via a prescribed interface in line with specified constraints and policies (Mirri et al., 2016). Modelling various functions as services allows the standardization of collection of data from different services and accessibility via standardized interfaces. Due to its distributed nature, service components are remotely accessed via remote access protocols e.g., representational state transfer (REST), Microsoft Messaging Queuing (MSMQ), etc.

Advantages of microservices pattern include: scalability, decoupling due to loosely coupled nature of components; better management of development, testing, deployment and maintenance due to self-contained and modular nature of components (Wolff, 2016).

Considerations to tackle when adopting this architecture pattern include:

- Service contract: agreement between service (remote) and service consumer or client, specifying both inbound and outbound data, format (XML, JSON, etc), versioning (homogenous or heterogeneous) and maintenance. Can be service based or consumer-driven contract
- service access: choice of remote access protocol
- Service availability and response: ability to establish connection to service and ability to get a response from service
- Service security: securing access to remote services and levels of such access to functionality within service i.e., authentication, authorization
- Service composition: Level of granularity of service component during design to avoid increase in complexity. This considers scope and functionalities
- Distributed transaction management –managing atomicity, consistency, isolation & durability of transactions at transaction level, as well as transaction state







6.3.4 SOA architectural patterns vs. Microservices architectural patterns













Both architectural patterns are distributed patterns that involve service components providing functionality or capability to other components as services remotely, via standard







communication protocols over a network(Cerny et al., 2018). In both patterns, each service component has a defined responsibility and is commonly implemented using web services e.g. RESTful Web services(Zimmermann, 2017). In SOA pattern, the service components are integrated via an Enterprise Service Bus (ESB) which allows communication via a common communication bus – the ESB. This bus can comprise of a variety of point-to-point connections between service components and other components of the architecture accessing the services. In Microservices pattern, service components communicate with each other via well-defined REST APIs that are language agnostic. The implication of this is that - in SOA, each service needs to be aware of the common communication mechanism for communicating with each other. This communication dependence on the ESB as the common communication mechanism in SOA, makes it a single point of failure or performance degradation(Richards, 2015a). However, in Microservices pattern, each service is independent of the other, and can be developed or deployed as such, allowing for greater fault tolerance and independent scaling or deployment. Inter-service communication mechanism in microservices architectural pattern is implemented via well-defined APIs and advanced message queues to shared repositories(Richards, 2015b).

Microservices architecture patterns tend to be smaller in size and scope, often consisting of relatively smaller, finely grained, self-contained services capable of being developed, tested, and deployed independently. SOA patterns tend to be relatively larger in size and scope and can comprise of multiple microservices. Although the goal of both patterns is to break applications into loosely coupled, more manageable service components, the finely-grained nature of services with less dependency in microservices patterns, offers greater flexibility, scalability and performance(Rademacher et al., 2017).

Table 19 comparative summary of common software architecture patterns (Richards, 2015b)

Architecture pattern	Key concepts	Description	Pros	Cons	Pattern Analysis scoring					
					Agility	Deployment	Testability	Performance	Scalability	Development
Layered	<ul style="list-style-type: none"> • Distinct layers • 4 standard layers. Can create more • Open-close concept • “Layers of isolation” concept for changes 	<ul style="list-style-type: none"> • Organizes components into n distinct layers • Each layer has specific role • Layer stacking determined by relationship or organization of components. • Uses “open” & “close” concepts to define layer relationships • Higher layers use services defined by immediate lower layer (in closed layering), or all lower layers (in open layering) 	<ul style="list-style-type: none"> • Widely known • Separation of concerns makes it easy to build roles • Limited component scope & well-defined interfaces make it easy to develop & test • Breaks apart complex systems • each layer can function as a coherent whole • Layer dependencies are minimized • Supports standardization through definition of layers & interfaces 	<ul style="list-style-type: none"> • Can introduce overhead e.g., storage or speed • Risk of architecture sinkhole anti-pattern • Can become monolithic (tightly coupled) & difficult to test/maintain without open-close concept 						

Event-driven	<ul style="list-style-type: none"> • 2 Topologies – broker & mediator • Decoupled single purpose component • Asynchronous process events • • 	<ul style="list-style-type: none"> • Mediator topology for orchestrating multiple steps of an event using a central mediator. • Broker topology for chaining together multiple events without using a central mediator • Mediator topology event components (event queues, event mediator, event channels, event processors) 	<ul style="list-style-type: none"> • Distributed • Asynchronous • Highly adaptable • Highly scalable 							
Space-based	<ul style="list-style-type: none"> • Processing unit • Virtualized middleware 	<ul style="list-style-type: none"> • Processing unit comprising of application modules & in-memory data grid for handling functionality 	<ul style="list-style-type: none"> • solves concurrency & scalability issues • Replicated application in-memory (distributed shared memory) • Components can be implemented through 3rd party services 	<ul style="list-style-type: none"> • Complex to implement • Expensive to implement • Not well-suited for large-scale database application with large data 						

		<ul style="list-style-type: none"> Virtualized middleware component to handle requests, data, sessions, communications, and other requirements. Comprise of four grids – messaging, data, processing, and deployment manager 	<ul style="list-style-type: none"> Good for small web-based applications Quick scaling of processing units. Responds well to changes 	<ul style="list-style-type: none"> Generally, not decoupled 						
Service-based patterns	<ul style="list-style-type: none"> Separate deployable units service components distributed architecture 	<ul style="list-style-type: none"> Topologies: API REST-based, Application REST-based and centralized messaging components designed as services with varying degrees of granularity Each service component can 	<ul style="list-style-type: none"> Distributed service architecture Offers considerable levels of abstraction Heterogeneous connectivity Easier deployment Streamlined delivery Service orchestration Increased scalability due to distributed nature 	<ul style="list-style-type: none"> Challenging to design right level of granularity Too many fine-grained service components can turn architecture into complex pattern. Database bottleneck due to use of centralized (shared) database 						

		<p>contain one or more modules/functions</p> <ul style="list-style-type: none"> • components are accessed through remote access protocol 	<ul style="list-style-type: none"> • Evolved to address monolithic issue of layered architecture • Manages growth of large systems • Facilitates service provisioning & usage • Service orchestration • Alignment of business goals with capabilities 	<ul style="list-style-type: none"> • Ubiquitous • Relative difficulty in implementation • Difficult to understand – complex. • Can be expensive 						
--	--	---	--	---	--	--	--	--	--	--

6.4 Modelling the architecture

Key assumptions

- The development process scenario is distributed and multi-phased
- The development process scenario is made up of at least one activity
- The development process scenario involves at least two people
- The development process scenario has at least one goal
- Resources involved within the development process are distributed
- People and systems interact within development process through exchange of messages and artefacts
- A service can be represented as implementation of a clearly defined activity or task
- Services can be provided by an individual or a component of the process

It is proposed that the cloud-based software development process adapt 6 key AT-based categories, as well as guiding principles, for all aspects of the collaborative development process(Engeström, 2001). These categories are:

- subject (stakeholders i.e., project managers, cross functional team members, community users),
- object (requirements which eventually influence task creation, modification, evaluation and sign off),
- cloud resources (tools – can be plugins),
- division of labour (team roles/responsibilities),
- Community, rules, and outcome.

These categories make up a unit of the collaborative development process and provide contexts to the process. The key assumptions and AT-based categories above are illustrated in the context model diagram in the Figure 43 below. Guiding principles are as follows:

- Collectiveness: The collective activity system (with all objects & artefacts) viewed in relation to other activity systems, is represented as a prime unit of analysis. This prime unit of analysis must be:
 - Goal-directed (tasks & actions are directed towards an identified goal)
 - Object-oriented (tasks & actions are focused on object transformation)
 - Reproducible through generation of tasks/actions/operations

- **Multi-voicedness:** activity system must be able to collect, translate and negotiate the views of the different participants in the various roles & responsibilities
- **Historicity:** The activity system needs to analyse object transformations against object's own historical transformations and actions that have shaped the object.
- **Mediation:** The activity system's interactions and transformations are mediated by tools and signs

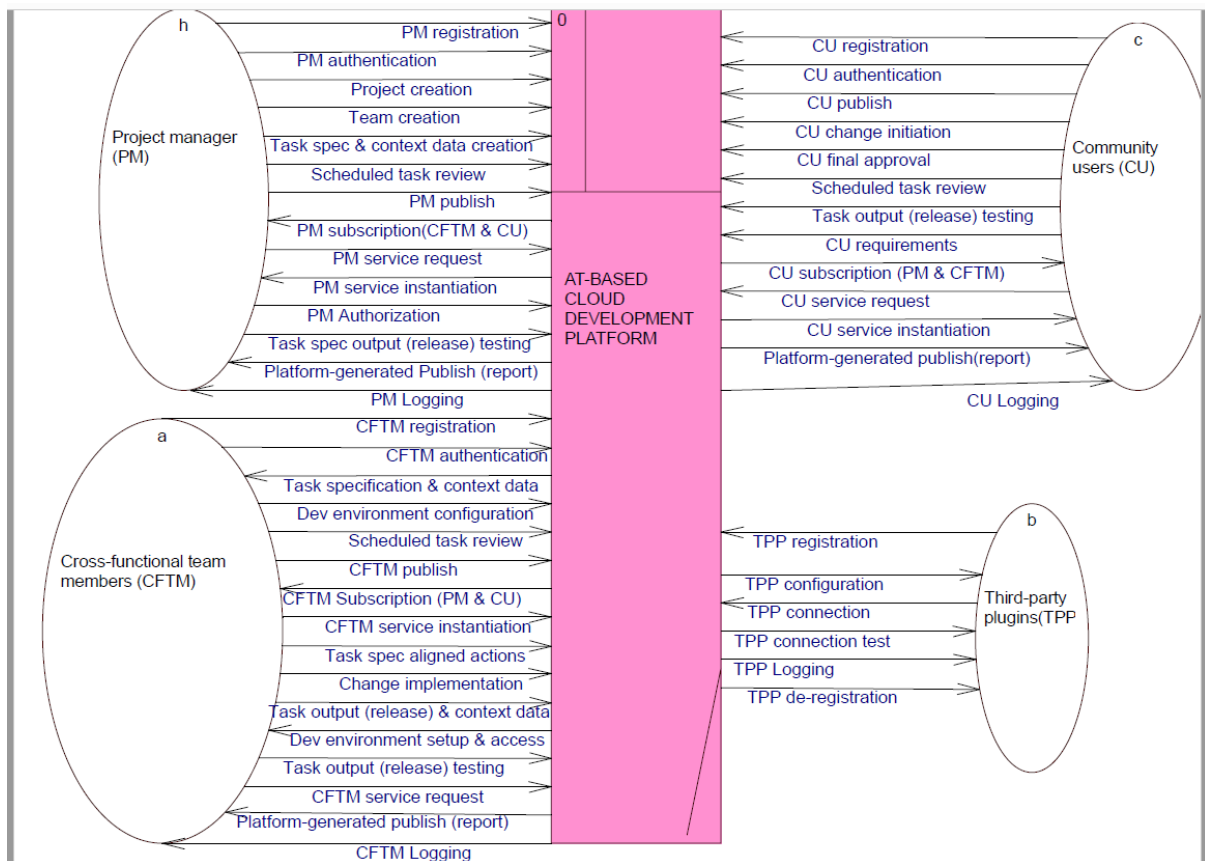


Figure 50 Context model for proposed AT-based architecture for enhancing context-aware collaboration cloud-based software development process.

Table 20 Components of context model for proposed AT-based architecture for enhancing context-aware collaboration cloud-based software development process.

Name	Classification	Data Flows
AT-BASED CLOUD SOFTWARE DEVELOPMENT PLATFORM (POC)	Process	
Project manager (PM)	External entities	PM registration PM authentication PM authorization PM service request PM service instantiation Project creation Team creation Task spec & context data creation Scheduled task review PM publish PM subscription (CFTM & CU) Task spec output (release) testing Platform-generated publish (report) PM Logging
Cross-functional team members (CFTM)	External entities	CFTM registration CFTM authentication CFTM service request CFTM service instantiation Task specification & context data Scheduled task review Dev environment configuration Dev environment setup & access Task spec aligned actions

		<p>Task output (release) & context data</p> <p>Task output (release) testing</p> <p>Change implementation</p> <p>CFTM publish</p> <p>CFTM Subscription (PM & CU)</p> <p>Platform-generated publish(report)</p> <p>CFTM Logging</p>
Community users (CU)	External entities	<p>CU registration</p> <p>CU authentication</p> <p>CU service request</p> <p>CU service instantiation</p> <p>CU requirements</p> <p>Scheduled task review</p> <p>Task output (release) testing</p> <p>CU change initiation</p> <p>CU final approval</p> <p>CU publish</p> <p>CU subscription (PM & CFTM)</p> <p>Platform-generated publish (report)</p> <p>CU Logging</p>
Third-party plugins (TPP)	External entities	<p>TPP registration</p> <p>TPP configuration</p> <p>TPP connection</p> <p>TPP connection test</p> <p>TPP de-registration</p> <p>TPP Logging</p>

6.5 Architecture description of activity scenario for collaborative software development process in the cloud

The software development process has the capacity to produce software-based and knowledge-based artefacts (the underlying capability). A context-aware AT-based architectural framework (the platform/service) will provide means to support collaboration within the software development process in the cloud (service functionality). A stakeholder (client or team member) will access activities, resources, and artefacts within the architectural framework (the output of invoking the platform service) via a standard interface (service interface). To access activities, resources and artefacts, a stakeholder will need to understand what type of resources to use, and other aspects of the cloud-based process, including possible limitations. There is a presumption that stakeholders will only be able to securely access activities, resources, and artefacts they participate in or collaborate on (service technical assumptions).

A stakeholder (client or team member) will need to create an account to use the service (service constraint) and the cloud-based architectural framework will meter usage and expect the stakeholder to use only what they are authorized to use, at the rate prescribed (service policy). When the stakeholder is authenticated on the platform, thereby agreeing on constraints and polices (service contract), the stakeholder receives authorization to access activities, resources and artefacts using the service as long as team membership and connection to service remain intact (e.g. leaving the team or lack of strong network connection would disrupt service distribution) and the stakeholder can have reports, alerts and payment sent (e.g. by messages, notifications or logs) to the registered account (reachability).

Another stakeholder (for example, a guest user or guest reviewer or guest developer) may use a contracted (limited) service without any registered account or any requirement to also satisfy the initial service constraint (i.e., reachability only requires intact network connection) but would nonetheless be expected to be compatible with the service interface. In certain situations (for example, excessive demand), an account may limit service usage or allow offline saves (service policy). A stakeholder might lodge a formal complaint if this occurs frequently (stakeholder's implied policy). If the account requires dedicated connection to

every activity, resource and artefact, the underlying capability would still be there, but this would be a very different service and have a very different service interface.

6.6 High-level architecture components

Architectural components are useful for the specifying the principal elements and behaviour of the system or architecture (Portocarrero et al., 2017). This diagram features the main components of the Architecture in tiers, for ease of modification or extension

Cloud Service Manager

Responsible for:

- service management
- Installing service on node
- Configuring service
- Rebalancing service across nodes
- Service failure detection
- Starting service
- Stopping service
- Pausing service
- Monitoring service
- Service logging

Service registry

Responsible for:

- maintaining list of services and service definitions
- maintaining list of service nodes
- enabling service lookup functionality

API Gateway

Responsible for:

- providing entry point for clients' requests and access to services
- forwarding client requests to appropriate service at the backend (Activity manager)
- aggregating and returning service responses to clients

- allowing decoupling of service from clients to facilitate ease of service refactoring
- providing client authentication
- providing logging functionality
- aiding load balancing
- Error detection

Activity Manager (Orchestrator)

- Create, define & sequence activities
- Create & assign roles
- Define environment variables
- Set up environments
- Suspend, resume & end activities/environments
- Synchronize activities/environments
- Environment/activity/role logging
- Manage activity workflow/environment
- Logic controller
- Manage artefacts
- Manage contexts
- Handle collaboration logic
- Handle context logic

Database service

This storage component of the architecture implements an efficient and structured storage mechanism for storing, organizing, and protecting data related to software projects before, during and after the process. This component of the framework implements an efficient and structured storage mechanism for storing, organizing, and protecting data related to software projects before, during and after development.

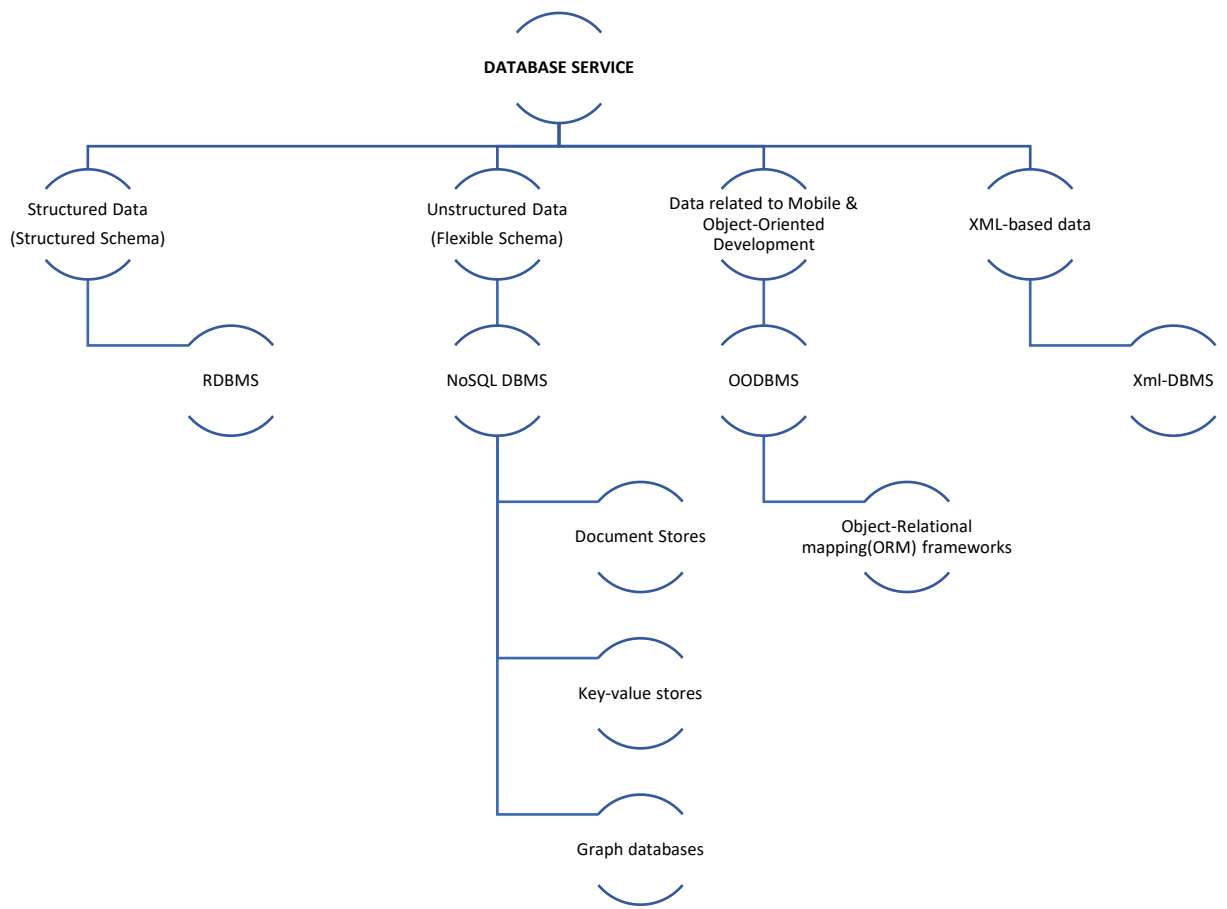


Figure 51 Database service categories

6.7 Activity and activity sequence decomposition

1. Stakeholder management services (*Based on roles*)
 - a) Cross-functional team member (CFTM) stakeholder service
 - b) Community user (CU) stakeholder service
 - c) Project manager (PM) stakeholder service
2. Core activity services - have been adapted from SWEBOK v3.0 (2014)
 - a) Project management service (*Accessible to PM role only*)

- Create project
- Register goal
- Create team
- Create task
- Manage/monitor goal
- Manage/monitor Project
- Manage/monitor team
- Manage/monitor task

- b) Requirements service (*accessible to CFTM roles*)

Modules

- Retrieve task specification
- Setup tasks
- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release

- c) Design service – accessible to CFTM roles

Modules

- Retrieve task specification
- Setup tasks

- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release

d) Build service – accessible to CFTM roles

Modules

- Retrieve task specification
- Setup tasks
- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release
- Release

e) Continuous Test service - – accessible to CFTM roles

Modules

- Retrieve task specification
- Setup tasks
- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release

f) Deployment service – accessible to CFTM roles

Modules

- Retrieve task specification
- Setup tasks
- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release

g) Maintenance service – accessible to CFTM roles

Modules

- Retrieve task specification
- Setup tasks
- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release

3. *Value-add services*

- Cloud Activity Review service
- Message broker service (Messaging Middleware)
- Reporting service – Cloud Activity Data Aggregation Service
- Resource Management service
- Cloud Activity Security service
- Database service

6.8 Operation-condition sequences for activity

1. Operation: Suppose a client or software owner has a need or goal e.g., needs development of an online pizza ordering system. The client enters this need using the CU service component. This action becomes the initial event and might be given a

topic like – ‘need or goal event’; with a message like – ‘require development of online pizza online system’.

2. **Operation:** This event is published to event channels (message broker).
3. **Condition:** Every action within any service component is treated as an event and automatically published to the event channels
4. **Condition:** Every published event is received and stored by every service component.
5. **Condition:** All service components are subscribed to event channels.
6. **Operation:** All service components receive, and store published event; and react to only those important to their function, or those with possible impact on their function.
7. **Operation:** The review service component can be invoked from within any other service component, and by any stakeholder.
8. **Operation:** Once the review service component is invoked, every other service is automatically paused, and will have to be manually resumed.
9. **Operation:** The entire process continues until the project management service component publishes a ‘project complete’ event.
10. **Operation:** Service components other than the reporting service component, can then discard unrelated published events that are stored in their data store periodically. The reporting service component stores every published event.
11. **Operation:** The reporting service component stores every published event and can be queried to generate reports and insights for the entire activity process, or filtered by service components, or some other criteria.

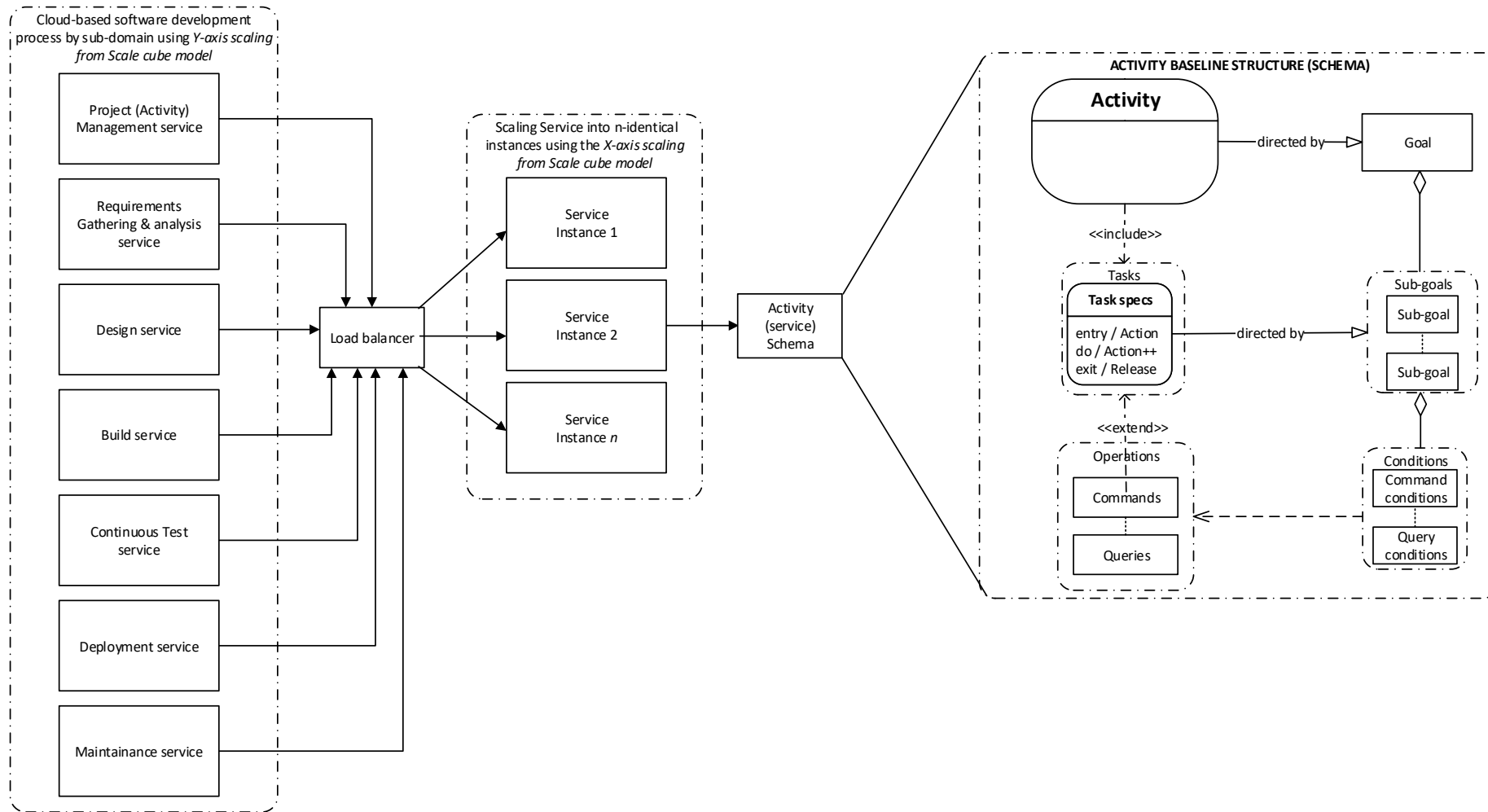


Figure 52 Service decomposition & structure

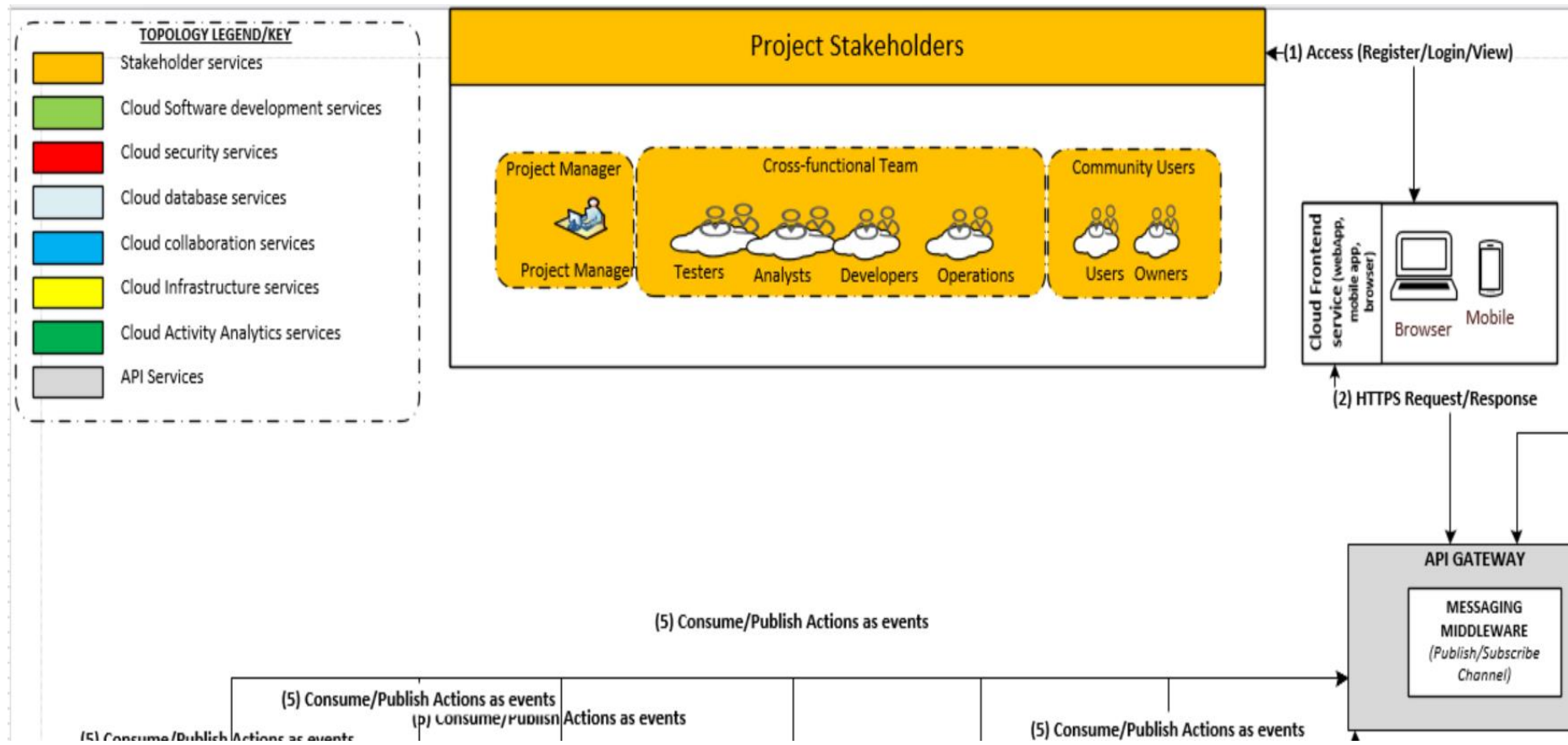


Figure 53 Project stakeholders, Cloud frontend and API Gateway

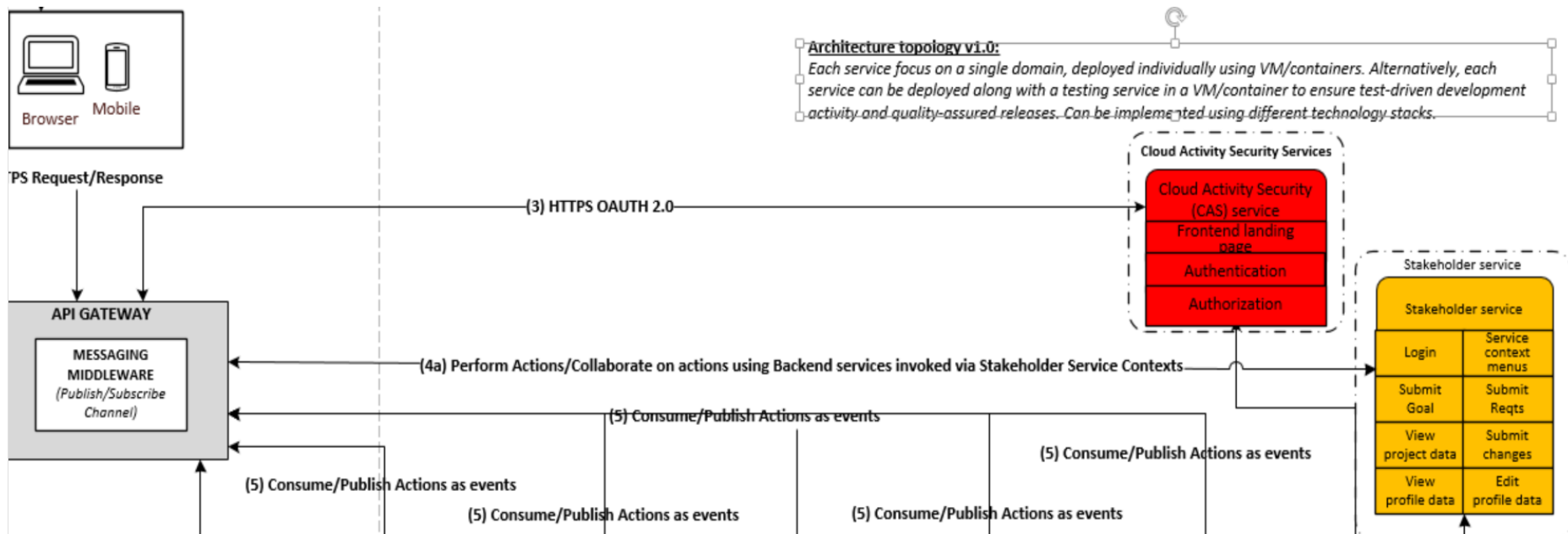


Figure 54 Cloud frontend, API Gateway, Cloud Activity service, and Stakeholder service

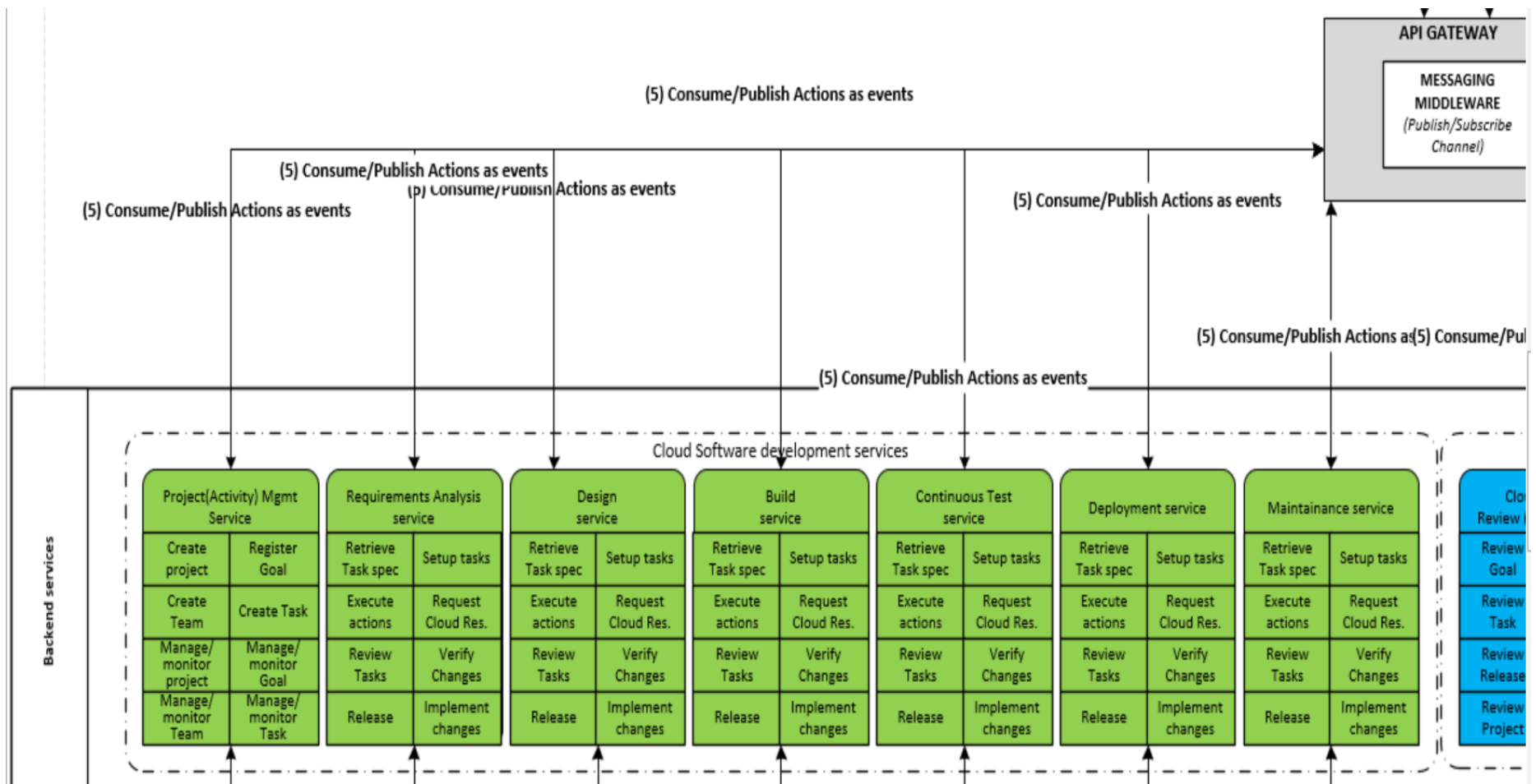


Figure 55 API Gateway and cloud software development services

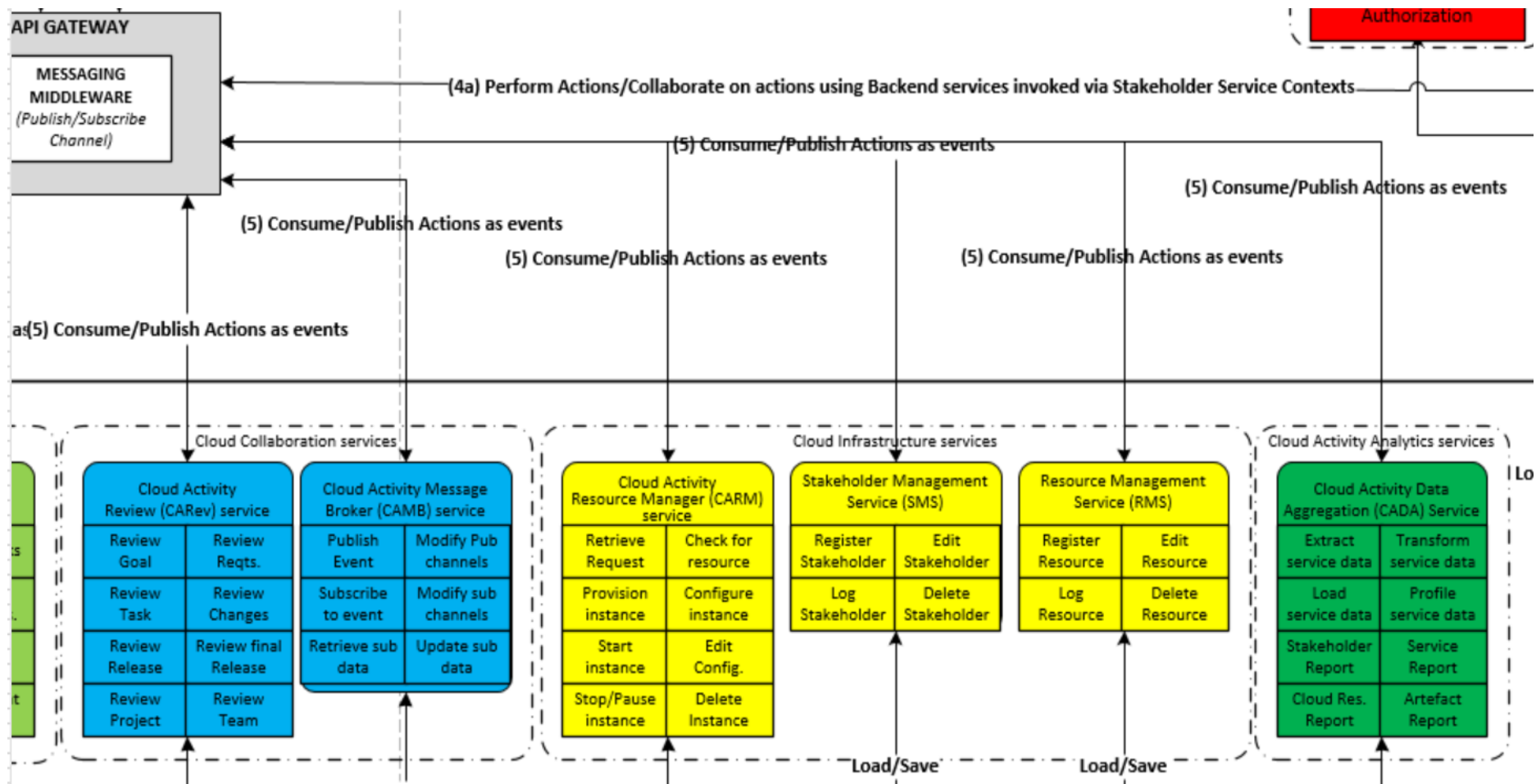


Figure 56 API gateway Cloud collaboration services, Cloud infrastructure services and Cloud activity analysis services

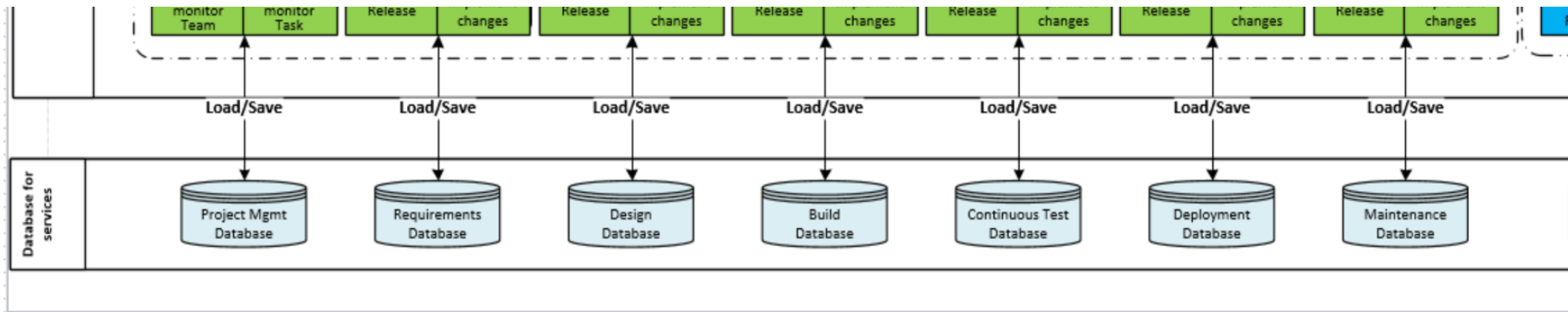


Figure 57 Datastores for Cloud software development services

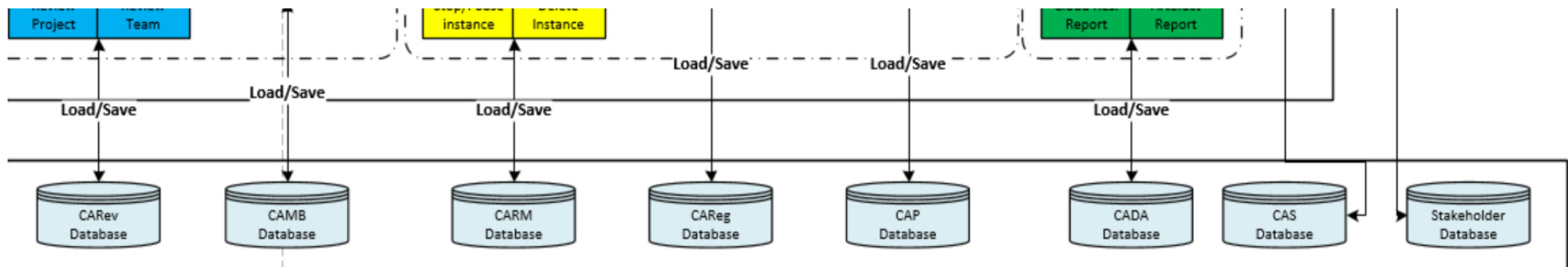


Figure 58 Datastores for Cloud collaboration services, Cloud infrastructure services and Cloud activity analysis services

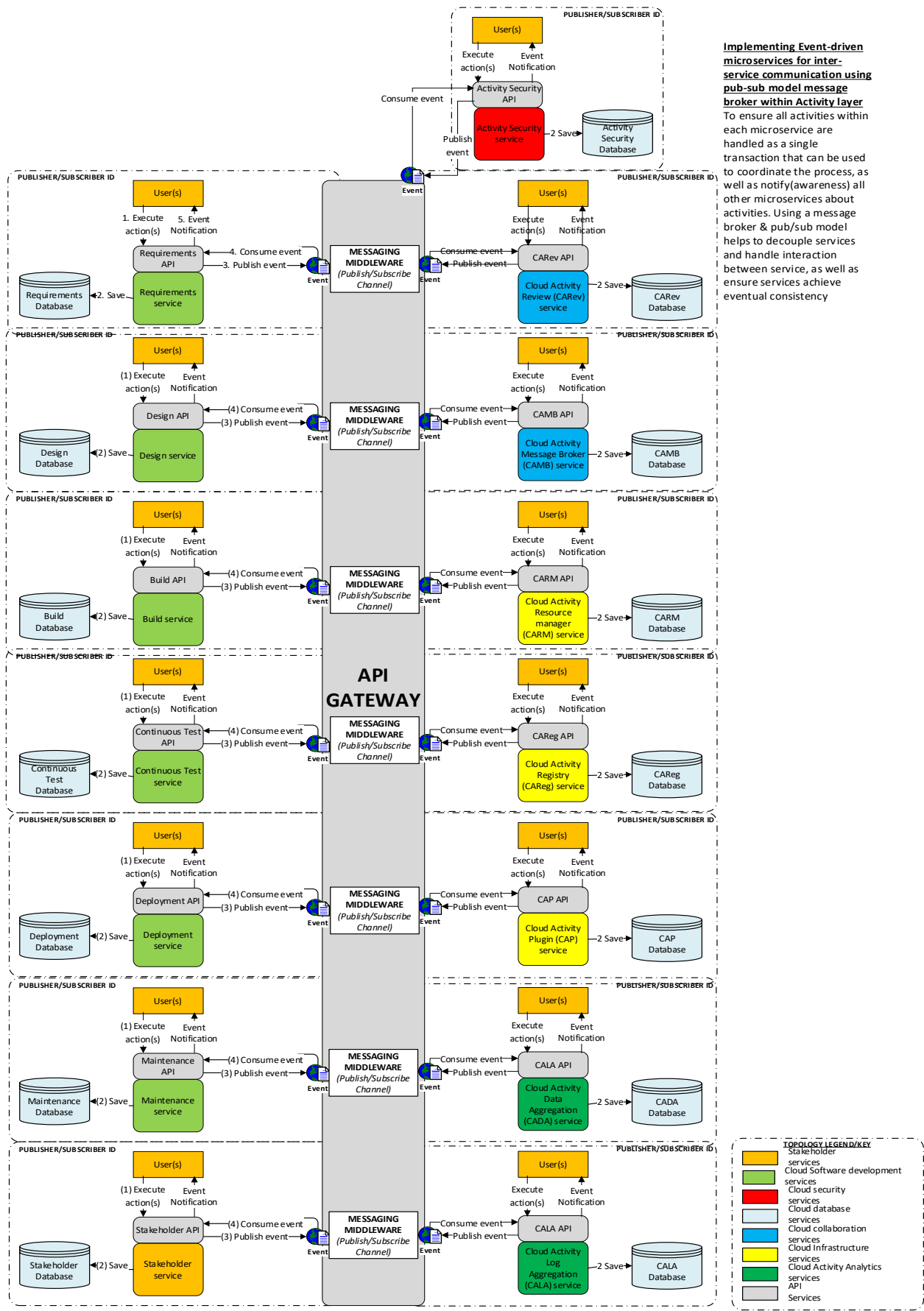


Figure 59 Event-driven interservice communication for Activity layer

Messaging Middleware pattern (Publish/Subscribe)

Coordination: Access to services and related data are **coordinated** through APIs

Trade-off - Using request-response model.

Pros: inter-service communication is synchronous; Abstraction to hide implementation details and flexibility to use/change implementation technologies with minimum impact on users.

Cons: services have to wait for data from API in order to perform required action

Using publish-subscribe model.

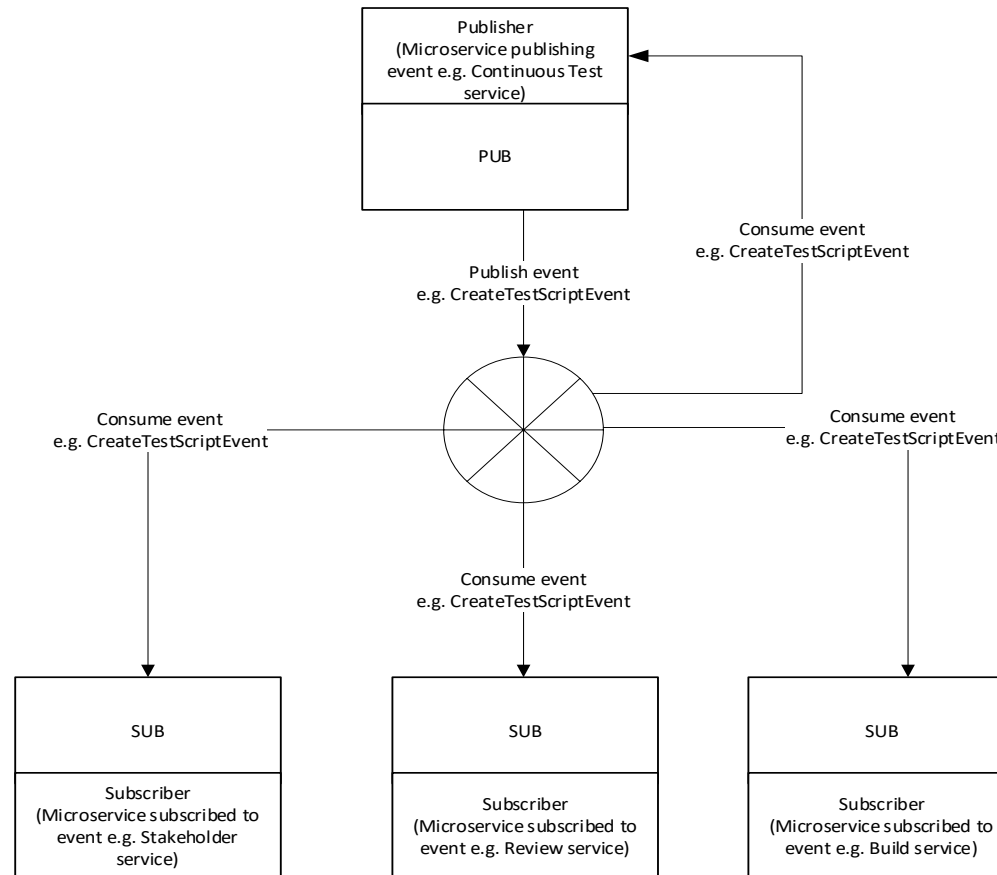


Figure 60 Messaging Middleware pattern (Publish/Subscribe)

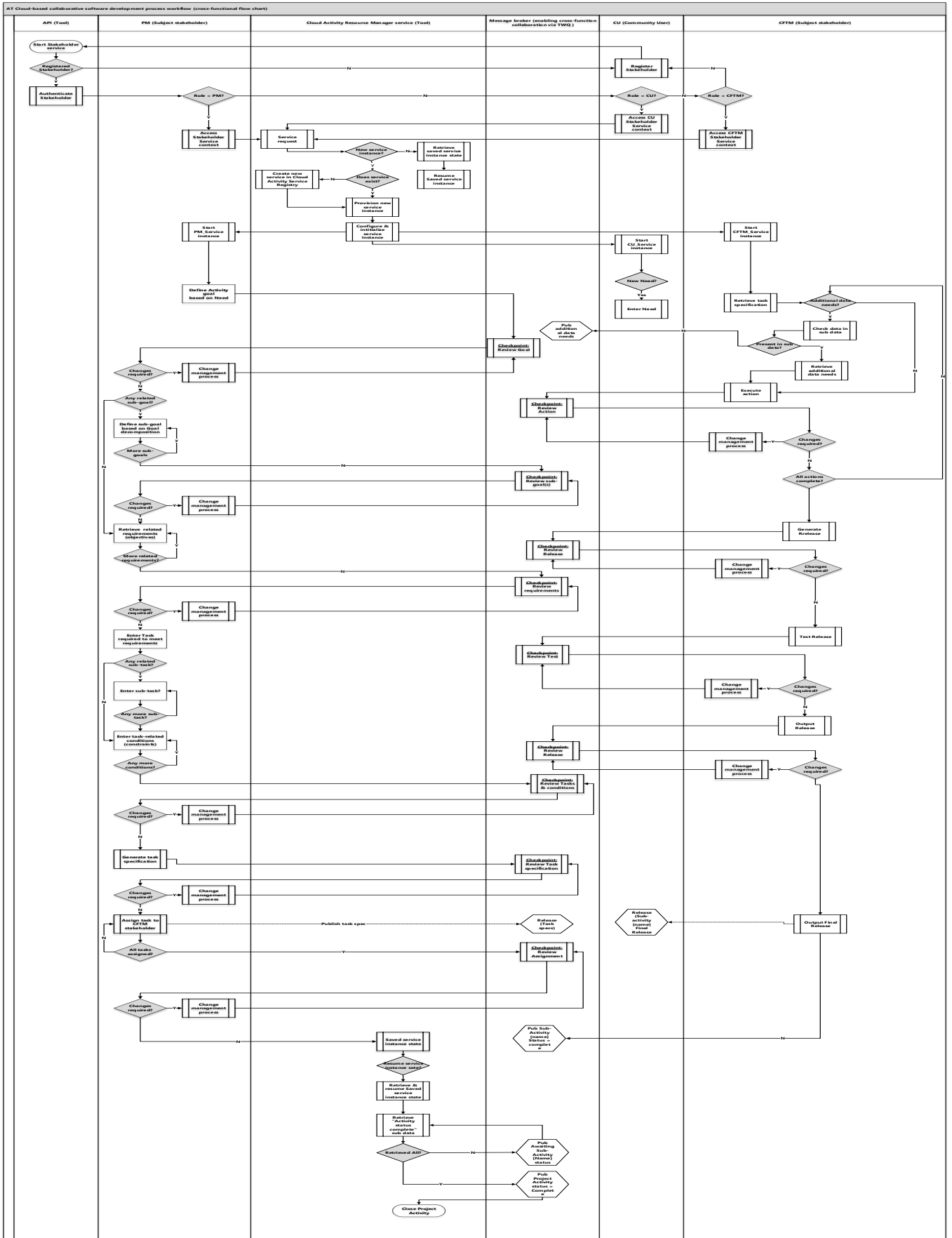


Figure 61 AT Cloud-based collaborative software development process workflow (cross-functional flow chart)

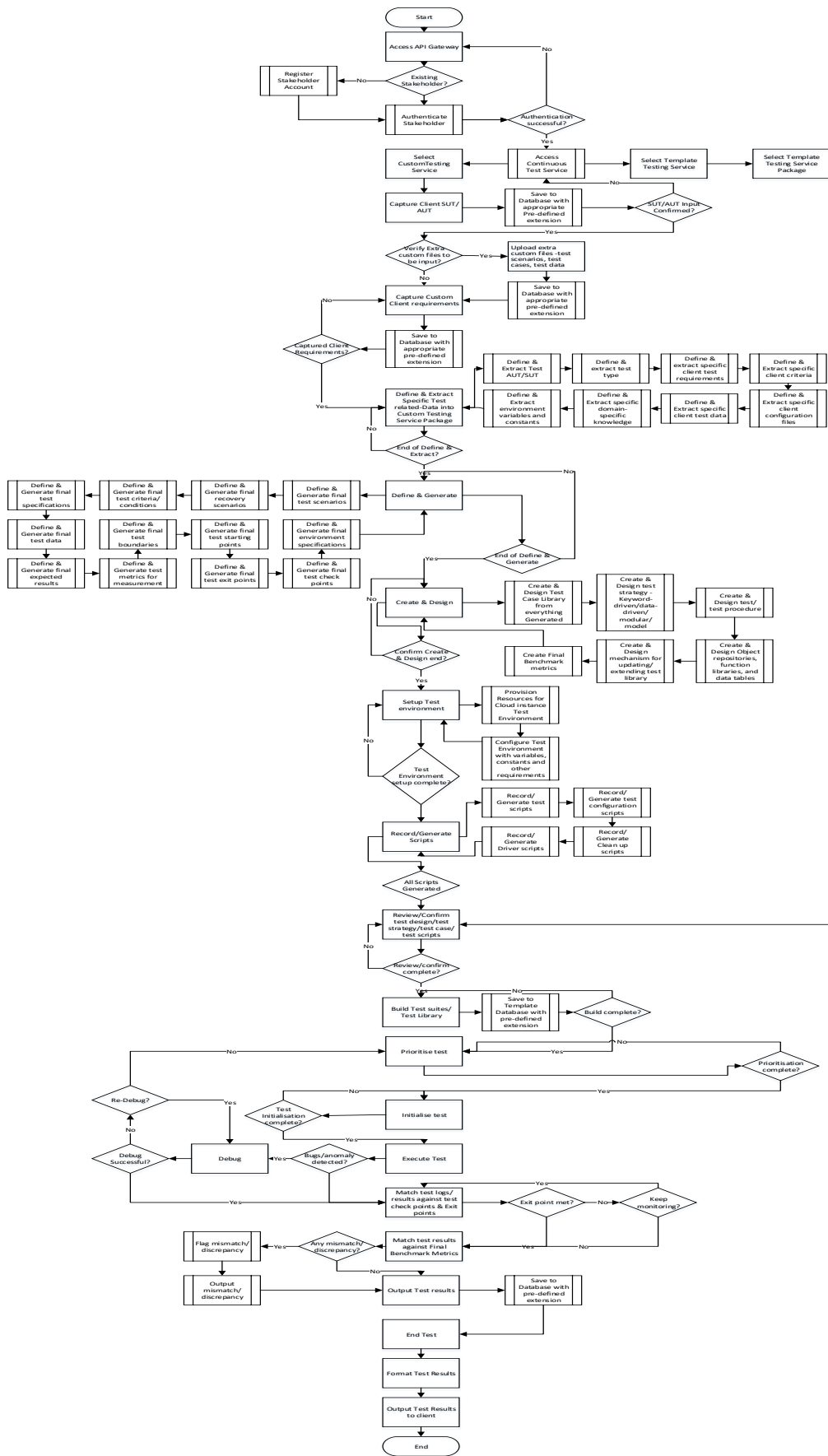


Figure 62 CFTM_Test workflow

NOTE: NEW DEVELOPERS, who have not yet completed training, are not allowed to participate in this process with the following exceptions:

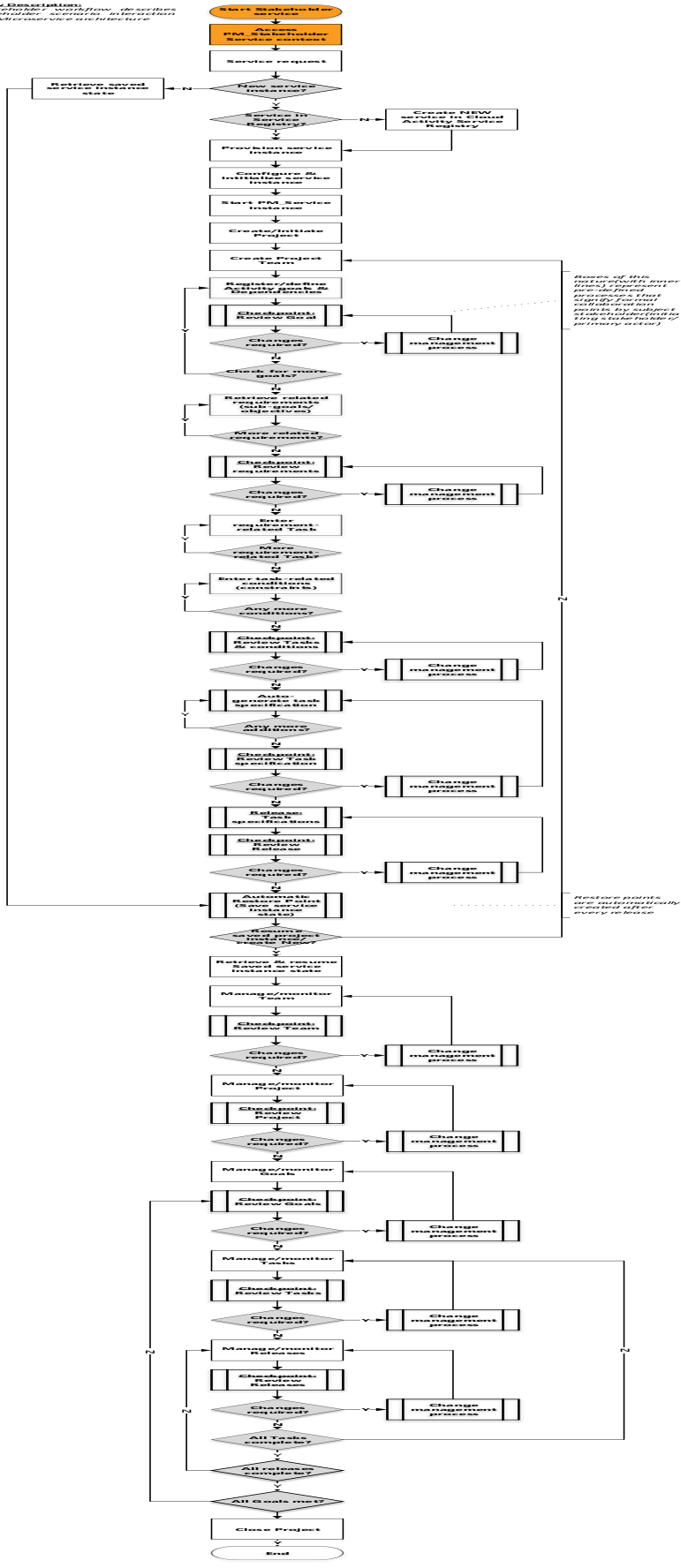


Figure 63 PM_Stakeholder Workflow

7 Architecture implementation and evaluation

7.1 Introduction

This section seeks to establish the trustworthiness of the research outcomes in this work. This is done through validation of literature review by means of published work in peer-reviewed conferences (see List of publications on page) and evaluation of the proposed architecture via a proof-of-concept implementation (POC). The POC implementation is based on the proposed microservice architecture and leverages the Cloud (Amazon EC2 instances). This implementation is then evaluated in terms of accessibility, functionality and performance through test scenarios, test cases and simulated mock services. The evaluation criteria of Usability, Functionality, Modifiability, Subset-ability, Reliability, and Performance have been discussed in Section 4.7.4 (of Section 4). First, Section 7.2 summarises the design of the evaluation exercises performed, the case study scenario and use cases involved. Then, Sections 7.3 and 7.4 report the methods (test scenarios, test cases) and test results. Section 5 already discussed the theoretical framework and methodology underpinning the architecture, while Section 6 already discussed and presented the architecture in terms of topology, description, and workflow, designed to accommodate the proposed recommendations in Section 3 using the conceptual foundations and theoretical framework discussed in Section 4 & 5. Finally, Section 7.6 summarises the main findings from the evaluation.

7.2 Evaluating and validating an architecture

The best way to assess and validate the ability of an architecture to meet stated requirements, as well as determine sustainability of the architecture, is through evaluation (Gonzalez-Huerta et al., 2015). Various ways or methods exist for evaluating an architecture: simulation or controlled experiments (Golden et al., 2005; Babar & Kitchenham, 2007); experience-based evaluations through expert knowledge; simulation or proof-of-concept based evaluations; use of mathematical models, methods and proofs; or scenario-based evaluation methods (e.g. Architecture Trade-off Analysis Method (ATAM), Architecture Level Modifiability Analysis method (ALMA), and Software Architecture Analysis Method (SAAM)); evaluation using ontologies (Erfanian & Shams Aliee, 2008; Mattsson et al., n.d.; Szwed et al., 2013; Omidvar & Vaziri, 2013). Maurya & Hora (1970) provide a comparison to allow for the selection of an evaluation method that could evaluate an architecture for more than one

quality attribute. Other methods evaluate software architecture for quality attributes by adopting aspect-orient programming methods, or use of metrics or use cases, or static evaluation techniques (Zayaraz et al., 2009; Barros et al., 2009; Bouwers, 2013; felienne, 2013; Knodel et al., 2006). Quality attributes of an architecture that can be evaluated include:

- Usability – refers to a measure of how well a user can effectively use a system
- Functionality – refers to a measure of the system’s ability to carry out functions that meet stated requirements
- Modifiability – refers to a measure of how easy & cost-effective it is to make changes to the system
- Subset-ability – refers to a measure of how easy it is to decompose a system into its component parts and keep it functional
- Reliability – refers to a measure of the operational accuracy/correctness of the system over time

There is lack of consensus in literature with regards to design and evaluation criteria for architectures (Gerber, et al., 2006). Therefore, to ensure availability of valuable information that provide insight into requirements of the architecture; guide architecture design; as well as comparatively evaluate against existing architectures, the following best practice criteria list have been compiled from literature.

Table 21 Summary best practice design and evaluation criteria (Bruegge & Dutoit, 2004)

CRITERIA	DESCRIPTION	EVALUATION QUESTION	REFERENCE
Clear context	Relates to determination of important aspects of the architecture model, components of system & properties, relationship between components	<ul style="list-style-type: none"> • Does the architecture description highlight or identify context? 	(Bass, et al., 2003) (Fowler, 2003) (Bruegge & Dutoit, 2004) (Gerber, et al., 2006) (Parnas, 2001)
Appropriate abstraction level & hiding of	Relates to ensuring sufficiently high-level holistic visibility of only relevant aspects of an architecture model at a system or subsystem level. Implementation details are hidden	<ul style="list-style-type: none"> • Can level view of the architecture model be considered holistic, within context? • Can any component, property or relationship be removed without losing 	(Bass, et al., 2003) (Fowler, 2003) (Bruegge & Dutoit, 2004)

implementation details		<p>wholeness of architecture model at the specified abstraction level?</p> <ul style="list-style-type: none"> • Are there any visible implementation details in the description of architecture, components, or relationships? 	(Gerber, et al., 2006)
Clear definition of component functionality	Refers to determination of architecture model components and/or component grouping	<ul style="list-style-type: none"> • Is the function of the component within the architecture model, specified by the component description? • Is the position of the component within the architecture model specified by the component description? • Can the component be removed without compromising architecture integrity? 	(Bass, et al., 2003) (Fowler, 2003) (Bruegge & Dutoit, 2004) (Gerber, et al., 2006)
Modularity	relates to modular organization or structuring of components or component grouping, to allow flexibility to make implementation changes provided functionality and interface remain same	<ul style="list-style-type: none"> • Can component implementation be replaced with another of same functionality and interface, without compromising architecture integrity? 	(Bass, et al., 2003) (Fowler, 2003) (Bruegge & Dutoit, 2004) (Gerber, et al., 2006)
Appropriate organization/structure	Relates to how components are organized/structured within the architecture model. This criterion includes specification of relationships and dependencies	<ul style="list-style-type: none"> • Are the components or component grouping clearly structured? • Are there components or component grouping requiring or depending on functionality defined or provided by another? • Are the dependencies clearly specified? 	(Bass, et al., 2003) (Fowler, 2003) (Bruegge & Dutoit, 2004) (Gerber, et al., 2006)

7.3 Architecture Implementation approach

The architecture adopts a hybrid of the service-based architecture pattern and layered architecture pattern. This translates into components exposed or modelled as services and running in an instance on top of a cloud middleware layer. Each service component is modelled as a single, self-contained service with functionalities. Each service is responsible for managing its data and state.

Table 22 Architecture implementation approach

PATTERN	COMPONENT	ROLE/RESPONSIBILITY	SUB-COMPONENTS	RESPONSIBILITY
Implementing Microservices Architecture				
Microservices	Service Components	<ul style="list-style-type: none"> perform specific functions of process 		
	Service access component	<ul style="list-style-type: none"> Provide access to remote service 	<ul style="list-style-type: none"> Can use REST-based User interface to access service OR lightweight centralized message broker as lightweight transport to access service Both above 	<p>REST-based API</p> <ul style="list-style-type: none"> Expose Services to stakeholders through API Receive stakeholder requests and messages Provide remote access to services <p>Lightweight message broker</p> <ul style="list-style-type: none"> Lightweight transport to access remote services Provide advanced queueing mechanisms and asynchronous messaging Monitor service Provide error handling Provide load balancing and scalability Provide broker clustering & broker federation
	Utility components	<ul style="list-style-type: none"> Handle shared functionality within service components 		
	Shared database	<ul style="list-style-type: none"> Handles inter-service communication Handle information needs 	<ul style="list-style-type: none"> Define Shared repositories as needed 	<ul style="list-style-type: none"> Define repository responsibilities as needed
Implementing Layered Architecture				

Layered	Presentation layer	<ul style="list-style-type: none"> • Handle browser communication logic • user interface • Handle presentation logic - format data for display 	<ul style="list-style-type: none"> • User screen • User delegate module (UDM) 	<p><u>Incoming</u></p> <ul style="list-style-type: none"> • User request is received through user screen • UDM locates appropriate UO • UDM determines what request-related data needs to be sent to responsible UO • UDM determines how to get request-related data to responsible UO • UDM assigns user request or process to responsible UO <p><u>Outgoing</u></p> <ul style="list-style-type: none"> • UDM formats result data • Passes formatted result data to User screen for onward display
	Activity orchestration layer	<ul style="list-style-type: none"> • Execute collaboration/context rules associated with user requests/processes • Perform collaboration/context logic on data 	<ul style="list-style-type: none"> • User object (UO) 	<p><u>Incoming</u></p> <ul style="list-style-type: none"> • Receives request & related data from UDM • Aggregates all info needed to execute request • Passes this info to appropriate UDAO (calls out) <p><u>Outgoing</u></p> <ul style="list-style-type: none"> • Checks result data to ensure it meets request requirements • Passes verified result data back to UDM
	n-layer e.g., shared services layer	<ul style="list-style-type: none"> • To expand functionality 	<ul style="list-style-type: none"> • As needed 	<ul style="list-style-type: none"> • Define as needed
	Persistence layer		<ul style="list-style-type: none"> • User data access object (UDAO) 	<p><u>Incoming</u></p> <ul style="list-style-type: none"> • Receives aggregated request-related info from UO • Gets requested data related to user request from database <p><u>Outgoing</u></p> <ul style="list-style-type: none"> • Aggregates result data from database • Passes result data back to UO

To evaluate the architecture developed in this research thesis, a proof-of-concept implementation was carried out, along with a simulation of a cloud-based activity scenario to evaluate performance of implemented functionality. Simulation was the approach chosen in this research thesis as a means of evaluating important approaches that can be employed to explore activity designs and gain confidence in the ability of the architecture to perform as expected (Taušan et al., 2017). The simulation conceptual model was used to describe and evaluate the architecture, what it represents, assumptions, and capabilities for satisfying specified requirements. The simulation was instrumental to troubleshooting and informing design decisions by allowing quantification of performance aspects, generation of test data and qualification of the system. Other pros to the use of simulation in this research thesis include reduction in costs (expense, resources, and time) of validation and verification that came from prototype implementation and testing using AWS services and Blazemeter tool for more horizontal test coverage. Usually, simulation can either be carried out in one phase, or broken down into some or all the following phases: verification, validation, and accreditation and simulation conceptual model (Peltz, 2003; Dijkman & Dumas, 2004). Verification: refers to the process undertaken to determine that an architecture or framework model, and related data, are an accurate representation of a conceptual description and specification. Validation: refers to the process taken to determine degree of accuracy to which an architecture or framework model and related data, accurately represents the real world, from the perspective of the model's intended use. Accreditation: refers to the official certification of an architecture or framework model, and related data, as acceptable for specific use. In this research thesis, all phases were bundled up into one phase.

7.4 Software requirements specifications (SRS) for POC implementation of architecture

The following requirements outlined below cover the core requirements for the proof-of-concept and provide additional bounded contexts for the implementation of the proposed architecture. This SRS summarises the service requirements implemented in this POC. The system should comprise of the following:

- R1. The system should provide access to team members irrespective of location or device
- R2. The system should be able to cope with large number of users and resources
- R3. The system should provide a user management service for registering users

- R4. The system should allow user registration via a standard interface.
- R5. The system should allow user registration *according to roles* (Roles are Admin, Project manager, Cross-functional team member & Community user)
- R6. The system should be able to add new users or remove users
- R7. The system should allow users to authenticate and log in via a standard interface
- R8. The system should allow authorization of user access to resources/functionalities with access privileges based on role
- R9. The system should provide *a requirement service*
- R10. The system should allow user roles to submit requirements for project(s).
- R11. The system should allow user roles to update requirements for a project.
- R12. The system should allow project manager roles to receive user requirements
- R13. The system should allow project manager roles to approve/reject user requirements.
- R14. The system should allow promotion of requirements to become projects.
- R15. The system should not allow un-promoted requirements to become projects
- R16. The system should allow requirements to be deleted.
- R17. The system should provide a resource management service.
- R18. The system should be able to add/remove resources (files attached to projects e.g., text, image, video, links etc)
- R19. The system should allow addition and utilization of external or internal cloud tool/resource of choice for testing. This could be via API/URL access
- R20. The system should provide flexibility to add to a project at any time.
- R21. The system should provide *a project service*
- R22. The system should allow tests to be carried out on projects
- R22. The system should only allow users to collaborate on projects they are registered on
- R23. The system should be able to log every service, user actions and outputs
- R24. The system should allow user roles to receive feedback for a project.

- R25. The system should allow provision of continuous system-wide notifications to users on actions, changes, feedback, tasks, updates, task status, and instructions/assignments
- R26. The system should allow members on a project to view project status from different devices and locations
- R27. The system should allow members on a project to make comments which others can respond to right away.
- R28. The system should allow test results generated from 3rd party apps to be uploaded to a project if the external app provides such resources.
- R29. The system should allow where possible, actions to be predefined, and users to select predefined actions

7.5 Development and deployment

Known for providing compute capacity that is both scalable and resizable as per demands, Amazon web services (AWS) was leveraged to launch and configure the required cloud resource via AWS EC2 instance(Kokkinos et al., 2015; Fusaro et al., 2011). Part of the considerations for this include the strengths of the cloud, as already covered in the Section 4 of this thesis – on-demand nature, ubiquity, elasticity, availability, scalability, accessibility if there is network connection, etcetera. Previous case studies have explored and evaluated the feasibility of use of AWS EC2 as development and deployment approach for SaaS(Balasubramanian Sekar et al., 2017; Kokkinos et al., 2015; Ostermann et al., 2010; Ellman et al., 2018). Also, Amazon has a free tier offering available to everyone(including students) and the cloud services offered covers 9 regions spanning Asia, Europe, South America, and the USA(Kamiński & Szufel, 2015). This greatly extends the coverage and reach, which is of considerable impact on collaboration and availability within a distributed development project with non-located stakeholders/members of the team. Due to these characteristics, setting up the POC implementation did not take very long. Also, when new functionalities had to be introduced in line with the requirements necessary to demonstrate the recommendations of this research, the simplicity of the AWS management console (see Figure below) enabled easy addition/scaling of the cloud resources needed i.e., computing power, optimised memory and storage needed. The low upfront cost implications in terms of setting up, running, and maintaining the EC2 instance was also a bonus. Amazon EC2

instances are instances that are run on physical resources by means open source virtualization middleware(Ostermann et al., 2010).

An Ubuntu Amazon machine image (AMI) and a t2.micro instance type was chosen because they are part of Amazon’s free tier offering. An AMI is an Amazon machine image pre-installed with an operating system(Balduzzi et al., 2012). The free tier offering includes the following monthly: 750 hours of resizable cloud compute capacity, 5GB of Amazon S3 scalable storage for databases, application and user files and 750 hours of db.t2.micro managed relational database service. 2 security groups were set up to configure the virtual firewall with a key-pair file and inbound/outbound rules created to facilitate secure log in to the EC2 instance via an SSH client (Fusaro et al., 2011).

The LAMP stack - Apache, PHP and MySQL support(Karanjit, 2016) was installed and set up to aid in the deployment of the POC application. Within the LAMP stack, Apache is used to run the web application manage HTTP/HTTPS requests and responses for clients/users accessing the application. MySQL was used as the relational database management system for storing data due because it is opensource and utilizes SQL queries for transactions, hence addresses vendor lock-in issues. PHP, also an open source server-side scripting language, was used to implement modular server-side data processing and management for the application and deploy the POC application via a model-view-controller design (Lotfy & Pyatt, 2018).

Amazon CloudWatch was used to set up easy application and resource monitoring for actionable insights in the form of event logging and metrics. This helps to provide application wide visibility and access to operational data for actions, issue resolution or improvement. Custom detailed metrics collection was set up to allow deep dives for additional contexts and service instrumentation - see figures below. CloudWatch log agents are responsible for real-time logging of events and services.

Remote Dictionary Server – REDIS(Klaesson, 2013; Sanchez et al., 2014; Gorlick & Taylor, 2014; Ghandehari & Stroulia, 2014), was used to store data in “key-value” pairs in memory as message queues that could be ‘popped’ and ‘pushed’ to facilitate the architecture’s events coordination and inter-process communication model between the services. Data such as user sessions, cookies, authentication tokens, messages, etcetera, are stored and transmitted amongst the development project stakeholders via Redis’ ‘pop/ and ‘push’ commands. This approach is shown in Figure 59: Messaging Middleware pattern (Publish/Subscribe) in Section

6. JavaScript Object-Notation (JSON) was adapted as a lightweight and portable file format to store and transfer data as objects within the application in a bid to avoid vendor lock-in issues highlighted in Section 3.

7.6 Evaluation of POC implementation and functionality performance

For the implementation of the POC for the proposed architecture a scaled-down version of the proposed architecture topology was used see (figure below). This is because building a simulation model at an appropriate level of abstraction makes it possible and more convenient to build a model that sufficiently describes the different parts of the real system. This approach allows capturing a mix of broad views and fine details of different aspects of the problem. The POC implementation serves as an experiment that provides an opportunity for empirical investigation and assessment of fundamental processes, resources, components, and relations encapsulated in the proposed architecture. The results of this exercise can then be used to provide basis to refute or backup claims(Menzies et al., 2016).

The POC featured a web interface for both stakeholder login and registration. This interface was designed to be accessible from any location or device. The main requirement for accessibility is the availability of a working network connection. Once the stakeholder has been registered and authenticated, depending on the role type registered, the relevant services available to the role is exposed via the API. Figure below shows the workflow employed for this activity. For this implementation, the active roles were restricted to 'Project manager', 'Team member' and 'User' and the functionality available to each role (see Architecture context model and data flows in Figure 48, 49, 51 & 52 & Table 16) is made available to the stakeholder. There is an additional implicit role – the Administrator (Admin) who retains full access privileges for administration purposes. The user can create requirements, make changes requests, provide feedback, and have visibility of activities within the development project(s) that he or she is registered on, at any point during the development project's lifecycle. More from lack of expertise than from application design, the user is not able to carry out development activities requiring specific expertise, such as coding, deployment, etcetera. Nonetheless, the user has visibility of all 'activities requiring specific expertise' carried out by other roles. Also, the user can at any point in time, access detailed logs, or upload additional requirements or change requests. However, all these are subject to a collective review by all members of the team, irrespective of roles, and

moderated by the Project manager. For a full view of all functionalities available to the user, as well as functionalities available to other roles, refer to the Architecture context model and data flows. For more evaluation outputs, please refer to Appendix 1. For further assessment of the behaviour of the POC implementation, a sample case study ‘project’ was devised, mirroring as near as possible, a real-life scenario.

Workflow Description:

This stakeholder workflow describes PM_stakeholder scenario interaction with the Microservice architecture

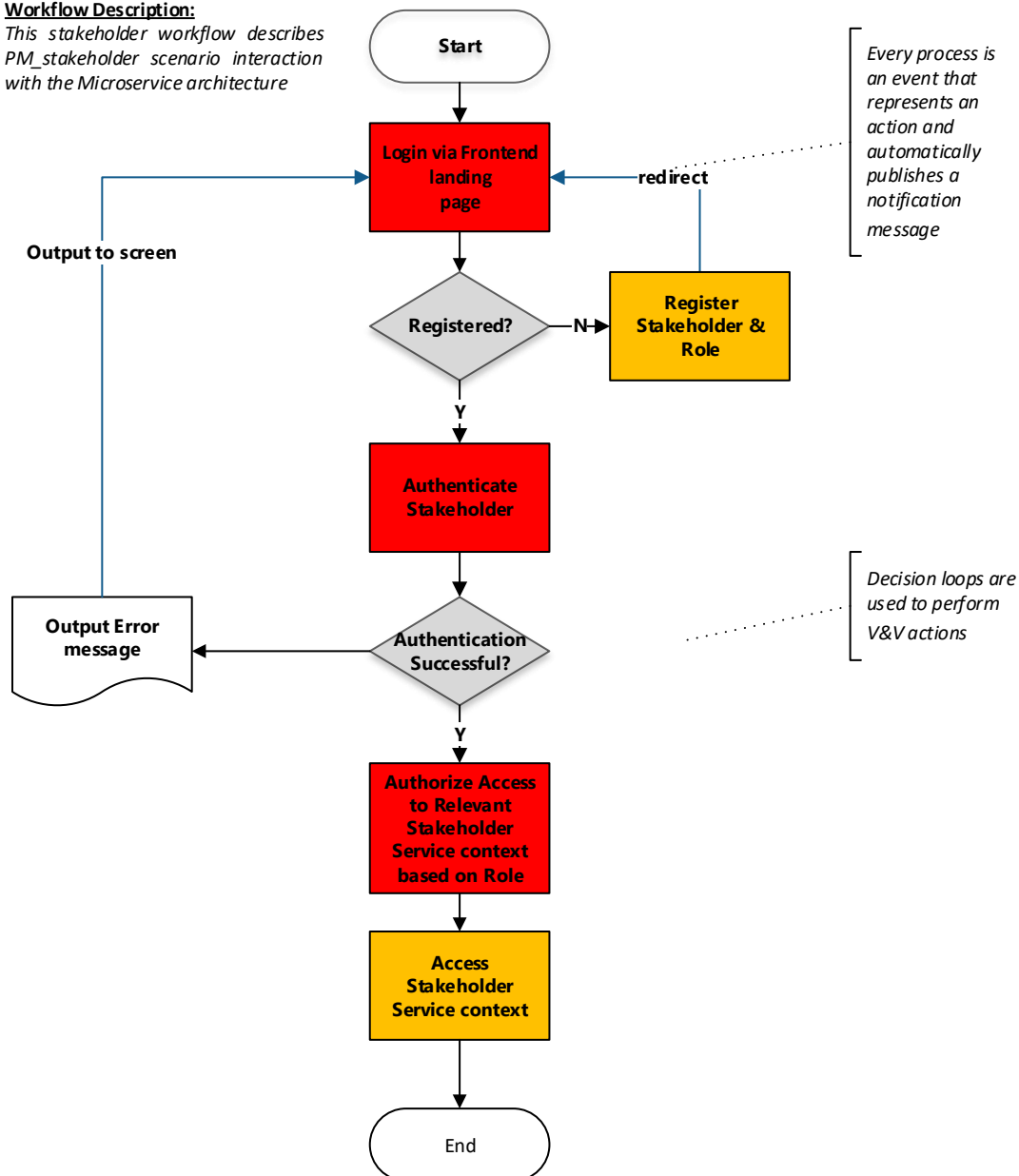


Figure 645: Cloud activity workflow

7.6.1 Case study Test Scenario:

A client (software owner) has a product (this POC application) with a variety of features that can be used by millions of users at any time (24hrs). Features of this product can be used by

the client’s customers who are in different parts of the world, via different browsers (Chrome, Safari, and Edge) and different devices (Android/iOS/Desktop/Laptop). Hence, the need to identify any cross browser/platform issue that may arise.

The client would also like to make feature updates of the product weekly. However, these features need to be tested by members of the Alpha testing team (me and my supervision team) prior to deployment, so that existing features on the live product are not affected. However, the members of Alpha testing team are in a variety of locations, different to the location of the client, but still need to work together to meet client’s needs.

The members of Alpha testing team may need to import testing tools to use in testing various features. The tool to be imported is based on either: suitability of the tool, or expertise of the team member. A test report needs to be generated to show test results for the different features tested, and any other information that may be of use.

Below is a sample of feature test scenario written using Gherkin keyword structure approach (dos Santos & Vilain, 2018). Keywords are feature; Scenario; Given, When, Then, And, But

Sample Feature to Test: Check Login Functionality

Scenario: The POC software owner would like to deploy Login functionality for the POC users. The POC software owner would like to ensure that only valid users can be authenticated and authorised to access services. Given that only registered users can access services, when non-registered users attempt to log in, access to services to submit requirements or create change requests should be denied. When registered users enter invalid details, access should be denied. But if valid details are entered then access to services should be granted.

Table 23: Test cases for sample feature to test.

Test Scenario #	Software Requirement Specification ID	Test Scenario Description	Test Cases
1	R7	Check Login Functionality	<ol style="list-style-type: none"> 1. Check system behavior when valid email id and password is entered. 2. Check system behavior when invalid email id and valid password is entered. 3. Check system behavior when valid email id and invalid password is entered. 4. Check system behavior when invalid email id and invalid password is entered. 5. Check system behavior when email id and password are left blank and Sign in entered. 6. Check Forgot your password is working as expected

7.6.2 Implemented service, roles, and activity sequence for POC

Services

- Stakeholder management services (*based on roles*)
- Project management service (*Accessible to PM role only*)
- Requirements service (*accessible to CFTM roles*)
- Continuous Test service - – accessible to CFTM roles
- *Value-add services*
 - Cloud Activity Review service
 - Message broker service (Messaging Middleware)
 - Reporting service – Cloud Activity Data Aggregation Service
 - Resource Management service
 - Cloud Activity Security service
 - Database service

Roles

- Cross-functional team member (CFTM) stakeholder service
- Community user (CU) stakeholder service
- Project manager (PM) stakeholder service

Activity sequences

- a) Project management service (*Accessible to PM role only*)

Modules

- Create project
- Register goal
- Create team
- Create task
- Manage/monitor goal
- Manage/monitor Project
- Manage/monitor team
- Manage/monitor task

- b) Requirements service (*accessible to CFTM roles*)

Modules

- Retrieve task specification
- Setup tasks
- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release

c) Continuous Test service - – accessible to CFTM roles

Modules

- Retrieve task specification
- Setup tasks
- Request cloud resources
- Execute tasks (actions)
- Review tasks – validate builds (with all stakeholders)
- Verify changes
- Implement changes
- Release

7.7 Operation-condition sequences for case study scenario

- **Operation:** Software owner had a need or goal and enters this need using the CU service component. This action becomes the initial event
- **Operation:** This event is published to event channels (via REDIS)
- **Condition:** Every action within any service component is treated as an event and automatically published to the event channels. E.g., entering the need or requirement is an action, approval of the requirement by PM or Admin is an action, assigning task to team member is an action, etc. These events are treated as actions by the POC application and published to all
- **Condition:** Every published event is received and stored by every service component.

- **Condition:** All service components e.g., team members/stakeholders are subscribed to event channels.
- **Operation:** All service components receive, and store published event; and react to only those important to their function, or those with possible impact on their function.
- **Operation:** The review service component can be invoked from within any other service component, and by any stakeholder.
- **Operation:** Once the review service component is invoked, every other service is automatically paused, and will have to be manually resumed.
- **Operation:** The entire process continues until the project management service component publishes a 'project complete' event.
- **Operation:** Service components other than the reporting service component, can then discard unrelated published events that are stored in their data store periodically. The reporting service component stores every published event.
- **Operation:** The reporting service component stores every published event and can be queried to generate reports and insights for the entire activity process, or filtered by service components, or some other criteria.

8 Conclusion and future work

This research thesis started by introducing software development in the cloud and the motivations that make the case for the need for enhancing collaboration within the process in the cloud.

Section 1 commenced this research project by introducing the research area, the motivations from preliminary research as well as from experience of working in the industry. The research objectives and contributions were also stated. The research aim of this project is to develop an architecture with sound theoretical foundation to ensure a sustainable approach to enhancing context-aware collaboration in cloud-based software development process. Section 2 presents a discourse on the underlying research philosophy and choice of research and a justification of methods adopted in this project. Section 3 reviews the existing body of knowledge in collaborative software development in the cloud. The goal of this Section is two-fold. It is a systematic state-of-the-art description, as well as an analysis of the research area. It is geared towards fostering an in-depth understanding of the research domain, as well as identification of research motivations, gaps, challenges, and issues pertinent to the research area and questions. It builds a case for modifying the scene. The research process described in Section 2 and carried out in Section 3 was based on the research questions defined in Section 1, the research philosophy described in Section 2, the search keywords and hybrid methodology described in Section 3. Most of the research were from Conference papers and journals, which is indicative of the maturity level of the research area. The gaps prioritised provided direction for the rest of this research thesis and the synthesized knowledge from the systematic literature review provided the substrate or springboard for review of conceptual building blocks.

Section 4 reviews and discusses conceptual foundations that are pivotal in. This Section attempts to develop classifications based on thematic analysis of recurrent themes from literature review towards the development of a more robust and holistic framework. Section 5 develops a formal process for streamlining the search for adequate theoretical basis, applies the developed process and provides justification for selecting activity theory as theoretical basis. This chapter then proceeds to assemble a theoretical framework and methodology for enhancing context-aware, collaboration in the cloud-based software development process. The lack of a de-facto architecture method for cloud-based software development meant that

Section 6 had to synthesize the methodology provided by the theoretical framework and software architectural patterns to develop an AT-based architecture to enhance a context-aware, collaboration in the cloud-based software development process. Section 7 presents a Proof-of-Concept implementation and evaluation of the architecture for enhancing context-aware collaboration in the cloud-based software development process. Section 8 concludes the research project. The diagram below in Fig 65 summarises this research journey.

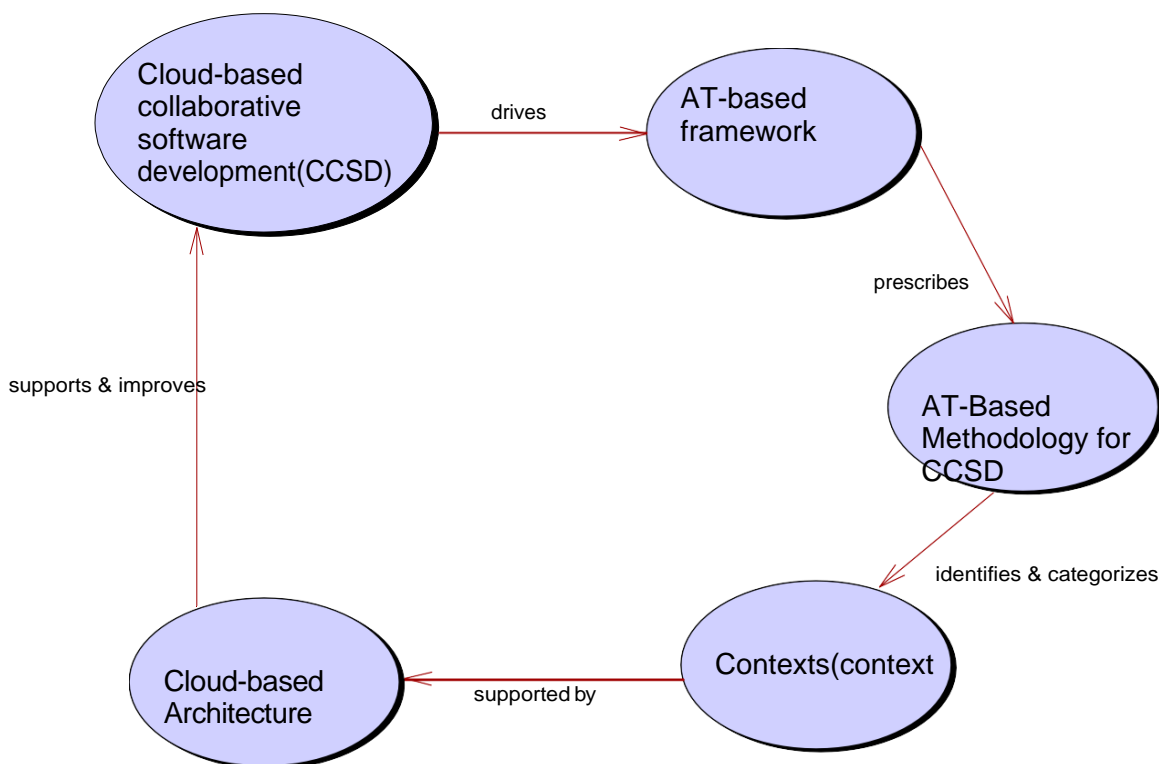


Figure 65 Summary of the research journey

The architecture developed in this thesis focuses on enhancing context-aware between all stakeholders in a cloud-based software development process in a cloud-based software development process. These stakeholders were defined in the theoretical framework in section 4 to include users, project managers, developers, testers, operations team members and all other members of the distributed development team. This definition also includes cloud and service providers who are involved in the management of hosted services and security. The architecture was built on top of AT-based concepts and implemented using the cloud towards enhancing collaboration in the software development process in the cloud and a proof of concept was developed, deployed it on AWS cloud platform and evaluated for

functionality performance. The biggest challenge that exists herein lies in the lack of ability to guarantee the absence of outages on the AWS platform. The implementation and evaluation of the POC for the architecture designed, developed, and described in this research thesis has highlighted quite a few directions which are hereby recommended as future work in this area with potential benefits.

Firstly, management functionality for the POC implementation was provided via leveraging AWS services. This approach is external to the POC's deployment. The risk posed by this approach is the possibility of the presence of intrinsic limits (Toffetti et al., 2015) that may inhibit or restrict natural scaling based on collaboration needs of the stakeholders in the cloud-based software development process. Hence, scenarios may arise where additional code or AWS intervention may be required. Furthermore, leveraging AWS provisioned-management functionality creates the possibility for vendor lock-in in scenarios where the platform may not necessarily be the most effective for a given collaborative software development project. Even though, provisions have been made for the registration and use of third-party plugins (i.e., external clouds, tools etc.), compatibility and interoperability may not be guaranteed.

Secondly, from literary evidence (Richardson, 2019), there is an expectation that there will be technology-related challenges to tackle when implementing a Microservice architecture. However, results from implementation and evaluation of the POC for the architecture developed in this research thesis, highlights the presence of challenges that may be more related to organizational structure, team setup, process, and strategy. To further understand the nature and characteristics of these challenges, more large-scale experimentation on a multi-organizational/multi-team level is required to provide more varied use case scenarios. This is of a wider scope than this research thesis can handle.

Thirdly, due to the novelty of Microservices architecture and lack of in-depth research on its anti-patterns to microservices architecting (Taibi et al., 2020), there are not much existing migration catalogues that can be useful to cloud-based software development teams looking to avoid pitfalls during adoption or migration to a microservices architecture (Newman, 2019; Balalaie et al., 2018). Further research into microservice antipatterns for collaborative software development in the cloud would help in the identification and classification of migration pitfalls. This research would also aid development of a Microservices antipattern

taxonomy for cloud-based collaborative software development. Expected impact will include development of more efficient migration plans (Balalaie et al., 2018), and additional tools to aid more detailed evaluation of microservices architecture for fine -grained collaboration in software development process in the cloud.

Lastly, challenges such as: lack of consensus on what should constitute the right level of granularity or modularity for a microservice; lack of consensus what should constitute the right level of responsibility assignment per microservice; lack of consensus as to the best-practice implementation of a microservice architecture as opposed to using methods such as architectural trade-offs. There is need for more research effort into the development of best practice patterns for design decisions involved in creating, resizing, and refactoring software development activities as services. Improper designation and delineation of boundaries could lead to increased network communication and bandwidth bottlenecks (Jamshidi et al., 2018).

Finally, the threats to validity for this research thesis include obtaining of articles using keyword search. Exclusions were subjective to a degree based on own interpretation of knowledge from preliminary research, and experience of the area. There exists the possibility of missing out on crucial articles due to this exclusion approach.

9 References

- Adolph, S. & Kruchten, P. (2013). Generating a useful theory of software engineering. In: *2013 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE)*. May 2013, pp. 47–50.
- Ahmedshareef, Z., Hughes, R. & Petridis, M. (2014). Applying Actor-Network Theory to Software Project Management Research. In: *European Conference on Research Methodology for Business and Management Studies*. [Online]. June 2014, Kidmore End, United Kingdom: Academic Conferences International Limited, pp. 1–10. Available from: <http://search.proquest.com/docview/1546004942/abstract/C2563C83CD33470DPQ/1>. [Accessed: 14 June 2016].
- Alvertis, I., Koussouris, S., Papaspyros, D., Arvanitakis, E., Mouzakitidis, S., Franken, S., Kolvenbach, S. & Prinz, W. (2016a). User Involvement in Software Development Processes. *Procedia Computer Science*. 97. p.pp. 73–83.
- Alvertis, I., Koussouris, S., Papaspyros, D., Arvanitakis, E., Mouzakitidis, S., Franken, S., Kolvenbach, S. & Prinz, W. (2016b). User Involvement in Software Development Processes. *Procedia Computer Science*. 97. p.pp. 73–83.
- Andres, B., Poler, R. & Sanchis, R. (2021). A data model for collaborative manufacturing environments. *Computers in Industry*. 126. p.p. 103398.
- Antoniadou, V. (2011). Using Activity Theory to understand the contradictions in an online transatlantic collaboration between student-teachers of English as a Foreign Language. *ReCALL*. 23 (3). p.pp. 233–251.
- Ardaiz, S. (2011). Collaborative Communication: Why Methods Matter. *Triple Pundit: People, Planet, Profit*. [Online]. Available from: <http://www.triplepundit.com/2011/12/collaborative-communication-methods-matter/>. [Accessed: 10 February 2015].
- Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & others (2010). A view of cloud computing. *Communications of the ACM*. 53 (4). p.pp. 50–58.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., & others (2009). *Above the clouds: A berkeley view of cloud computing*. [Online]. Available from: http://home.cse.ust.hk/~weiwa/teaching/Fall15-COMP6611B/reading_list/AboveTheClouds.pdf. [Accessed: 5 October 2016].
- Arora, R., Goel, S. & Mittal, R.K. (2017). Supporting collaborative software development over GitHub. *Software: Practice and Experience*. 47 (10). p.pp. 1393–1416.
- Babar, M.A. & Kitchenham, B. (2007). The Impact of Group Size on Software Architecture Evaluation: A Controlled Experiment. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. September 2007, pp. 420–429.

Bærentsen, K.B. & Trettvik, J. (2002). An Activity Theory Approach to Affordance. *Nordichi 2002. Proceedings of the Second Nordic Conference on Human-Computer Interaction*. p.pp. 51–60.

Bajpai, N. (2011). *Business research methods*. [Online]. Pearson Education India. Available from:
[https://books.google.co.uk/books?hl=en&lr=&id=wY2bSaEm8l8C&oi=fnd&pg=PR22&dq=Bajpai,+N.+\(2011\)+%E2%80%9CBusiness+Research+Methods%E2%80%9D+Pearson+Education+India&ots=p8DNITGygm&sig=SaptARbBkxuc_I0YbbmQCFycqHs](https://books.google.co.uk/books?hl=en&lr=&id=wY2bSaEm8l8C&oi=fnd&pg=PR22&dq=Bajpai,+N.+(2011)+%E2%80%9CBusiness+Research+Methods%E2%80%9D+Pearson+Education+India&ots=p8DNITGygm&sig=SaptARbBkxuc_I0YbbmQCFycqHs). [Accessed: 18 October 2016].

Balasubramanian Sekar, V., Patil, V., Giusti, M., Bhide, A. & Gupta, A. (2017). AWS EC2 vs. Joyent's Triton: A Comparison of Docker Container-hosting Platforms. In: *Proceedings of the 8th Workshop on Scientific Cloud Computing*. ScienceCloud '17. [Online]. 27 June 2017, Washington, DC, USA: Association for Computing Machinery, pp. 33–36. Available from: <https://doi.org/10.1145/3086567.3086572>. [Accessed: 31 March 2020].

Balduzzi, M., Zaddach, J., Balzarotti, D., Kirda, E. & Loureiro, S. (2012). A security analysis of amazon's elastic compute cloud service. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC '12. [Online]. 26 March 2012, Trento, Italy: Association for Computing Machinery, pp. 1427–1434. Available from: <https://doi.org/10.1145/2245276.2232005>. [Accessed: 31 March 2020].

Barcus, A. & Montibeller, G. (2008). Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis. *Omega*. 36 (3). p.pp. 464–475.

Barenji, A.V., Guo, H., Wang, Y., Li, Z. & Rong, Y. (2021). Toward blockchain and fog computing collaborative design and manufacturing platform: Support customer view. *Robotics and Computer-Integrated Manufacturing*. 67. p.p. 102043.

Baride, S. & Dutta, K. (2011). A cloud based software testing paradigm for mobile applications. *SIGSOFT Softw. Eng. Notes*. 36 (3). p.pp. 1–4.

Barnett, L. & Schwaber, C.E. (2004). Applying open source processes in corporate development organizations. *Forrester Research*. p.pp. 1–15.

Barthelmess, P. & Anderson, K.M. (2002a). A View of Software Development Environments Based on Activity Theory. *Computer Supported Cooperative Work (CSCW)*. 11 (1–2). p.pp. 13–37.

Barthelmess, P. & Anderson, K.M. (2002b). A View of Software Development Environments Based on Activity Theory. *Comput. Supported Coop. Work*. 11 (1–2). p.pp. 13–37.

Basit, T. (2003). Manual or electronic? The role of coding in qualitative data analysis. *Educational Research*. 45 (2). p.pp. 143–154.

Bedny, G.Z. & Harris, S.R. (2005). The Systemic-Structural Theory of Activity: Applications to the Study of Human Work. *Mind, Culture, and Activity*. 12 (2). p.pp. 128–147.

Begel, A., Bosch, J. & Storey, M.-A. (2013). Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. *IEEE Software*. 30 (1). p.pp. 52–66.

Begel, A., Herbsleb, J.D. & Storey, M.-A. (2012). The future of collaborative software development. In: *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion*. CSCW '12. [Online]. 2012, New York, NY, USA: ACM, pp. 17–18. Available from: <http://doi.acm.org/10.1145/2141512.2141522>. [Accessed: 3 July 2013].

Bendas, D., Saari, L., Coutinho, C., de Juan-Marín, R. & Bernabé-Gisber, J. (2017a). *Distributed Software Development of a Cloud Solution for Collaborative Manufacturing Networks*. In: 29 June 2017.

Bendas, D., Saari, L., Coutinho, C., Marín, R. de J., Gisbert, J.B. & Lopes, L. (2017b). Distributed software development of a cloud solution for collaborative manufacturing networks. In: *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. June 2017, pp. 741–749.

Benedek, A. & Lajos, G. (2012). BUILDING AUGMENTED KNOWLEDGE ARCHITECTURES: REQUIREMENTS FOR COLLABORATION PLATFORMS OF NEXT-GEN CONCEPT ORGANIZATION TOOLS. *ICERI2012 Proceedings*. p.pp. 1492–1506.

Benfenatki, H., Ferreira Da Silva, C., Benharkat, A.-N. & Ghodous, P. (2014). Cloud-Based Business Applications Development Methodology. In: *2014 IEEE 23rd International WETICE Conference*. June 2014, pp. 275–280.

A. Bento & A. K. Aggarwal (eds.) (2012). 00002. *Cloud Computing Service and Deployment Models: Layers and Management*. [Online]. IGI Global. Available from: <http://www.igi-global.com/chapter/requirements-engineering-cloud-application-development/70138>. [Accessed: 19 March 2014].

Boehm, B. (2006a). A View of 20th and 21st Century Software Engineering. In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE '06. [Online]. 2006, New York, NY, USA: ACM, pp. 12–29. Available from: <http://doi.acm.org/10.1145/1134285.1134288>. [Accessed: 21 July 2014].

Boehm, B. (2006b). Some future trends and implications for systems and software engineering processes. *Systems Engineering*. 9 (1). p.pp. 1–19.

Boehm, B.W. (2010). Some Future Software Engineering Opportunities and Challenges. In: *ResearchGate*. [Online]. 1 January 2010, pp. 1–32. Available from: https://www.researchgate.net/publication/221350488_Some_Future_Software_Engineering_Opportunities_and_Challenges. [Accessed: 18 November 2016].

Bojanova, I., Zhang, J. & Voas, J. (2013). Cloud Computing. *IT Professional*. 15 (2). p.pp. 12–14.

Bourque, P., Fairley, R.E., & IEEE Computer Society (2014). *SWEBOK: guide to the software engineering body of knowledge*.

- Bouwers, E.M. (2013). *Metric-based Evaluation of Implemented Software Architectures*. [Online]. Available from: <http://repository.tudelft.nl/islandora/object/uuid:6b65c5f5-398c-4a41-8806-31c638b1891c/?collection=research>. [Accessed: 1 March 2017].
- Box (2012). The Cloud: Reinventing Enterprise Collaboration. *FierceCIO*. [Online]. Available from: <http://whitepapers.fiercecio.com/content19723>. [Accessed: 20 March 2014].
- Bradley, E.H., Curry, L.A. & Devers, K.J. (2007). Qualitative Data Analysis for Health Services Research: Developing Taxonomy, Themes, and Theory. *Health Services Research*. 42 (4). p.pp. 1758–1772.
- Brézillon, P. & Gonzalez, A.J. (2014). Google-Books-ID: 87DLBQAAQBAJ. *Context in Computing: A Cross-Disciplinary Approach for Modeling the Real World*. Springer.
- Bryman, A. (2001). Google-Books-ID: 3ulxQgAACAAJ. *Social Research Methods*. Oxford University Press.
- Bryman, A. (2012). Google-Books-ID: vCq5m2hPkOMC. *Social Research Methods*. OUP Oxford.
- Buhrer, H.K. (2003). Software Development: What It is, What It Should Be, and How to Get There. *SIGSOFT Softw. Eng. Notes*. 28 (2). p.pp. 5-.
- Camarihna-Matos, L.M. & Afsarmanesh, H. (2008). *Concept of Collaboration*. [Online]. 2008. Academia.edu. Available from: http://www.academia.edu/248756/Concept_of_Collaboration. [Accessed: 24 June 2013].
- Cancian, M.H., Rabelo, R.J. & Hauck, J.C.R. (2020). Towards a capability and maturity model for Collaborative Software-as-a-Service. *Innovations in Systems and Software Engineering*. [Online]. Available from: <https://doi.org/10.1007/s11334-020-00360-9>. [Accessed: 30 March 2020].
- Carter, N., Bryant-Lukosius, D., DiCenso, A., Blythe, J. & Neville, A.J. (2014). The use of triangulation in qualitative research. *Oncology Nursing Forum*. 41 (5). p.pp. 545–547.
- Cassens, J. & Kofod-Petersen, A. (2006). Using Activity Theory to Model Context Awareness: A Qualitative Case Study. In: *FLAIRS Conference*. 2006, pp. 619–624.
- Cerny, T., Donahoo, M.J. & Trnka, M. (2018). Contextual Understanding of Microservice Architecture: Current and Future Directions. *SIGAPP Appl. Comput. Rev.* 17 (4). p.pp. 29–45.
- Cerone, A. & Roveri, M. (2018). Google-Books-ID: fB5KDwAAQBAJ. *Software Engineering and Formal Methods: SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers*. Springer.
- Chadli, S.Y., Idri, A., Ros, J.N., Fernández-Alemán, J.L., Gea, J.M.C. de & Toval, A. (2016). Software project management tools in global software development: a systematic mapping study. *SpringerPlus*. [Online]. 5 (1). Available from: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5121116/>. [Accessed: 8 May 2019].

- Chanda, N. & Liu, X.F. (2015). Intelligent analysis of software architecture rationale for collaborative software design. In: *2015 International Conference on Collaboration Technologies and Systems (CTS)*. June 2015, pp. 287–294.
- Chang, B.-Y., Hai, P.H., Seo, D.-W., Lee, J.-H. & Yoon, S.H. (2013a). The determinant of adoption in cloud computing in Vietnam. In: *2013 International Conference on Computing, Management and Telecommunications (ComManTel)*. 2013, pp. 407–409.
- Chang, V., Walters, R.J. & Wills, G. (2013b). The development that leads to the Cloud Computing Business Framework. *International Journal of Information Management*. 33 (3). p.p. 524–538.
- Charmaz, K. (2013). *Constructing Grounded Theory*. 2 edition. London ; Thousand Oaks, Calif: Sage Publications Ltd.
- Chhabra, B., Verma, D. & Taneja, B. (2010). *Software Engineering Issues from the Cloud Application Perspective*. p.p. 5.
- Childs, P.R.N. (2019). 3 - Ideation. In: P. R. N. Childs (ed.). *Mechanical Design Engineering Handbook (Second Edition)*. [Online]. Butterworth-Heinemann, pp. 75–144. Available from: <https://www.sciencedirect.com/science/article/pii/B9780081023679000032>. [Accessed: 3 July 2022].
- Chorin, A.J. & Hald, O.H. (2014). *Stochastic Tools in Mathematics and Science*. Springer Science & Business Media.
- Chrissis, M.B., Konrad, M. & Shrum, S. (2011). *CMMI for Development: Guidelines for Process Integration and Product Improvement*. 3 edition. Upper Saddle River, NJ: Addison Wesley.
- Ciancarini, P., Omicini, A. & Zambonelli, F. (2000). Multiagent System Engineering: The Coordination Viewpoint. In: N. R. Jennings & Y. Lespérance (eds.). *Intelligent Agents VI. Agent Theories, Architectures, and Languages*. Lecture Notes in Computer Science. [Online]. Springer Berlin Heidelberg, pp. 250–259. Available from: http://link.springer.com/chapter/10.1007/10719619_19. [Accessed: 4 February 2015].
- Cico, O. & Cico, B. (2019). Reliable Cloud Software Development Architectures and Business Models Case Study: RIDEaaS and GAE Launcher. In: *Proceedings of the 9th Balkan Conference on Informatics*. BCI'19. [Online]. 26 September 2019, New York, NY, USA: Association for Computing Machinery, pp. 1–8. Available from: <https://doi.org/10.1145/3351556.3351586>. [Accessed: 13 June 2021].
- Cito, J., Leitner, P., Fritz, T. & Gall, H.C. (2015). The making of cloud applications: An empirical study on software development for the cloud. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. [Online]. 2015, ACM, pp. 393–403. Available from: <http://dl.acm.org/citation.cfm?id=2786826>. [Accessed: 5 July 2017].
- Clear, T. (2009). Dimensions of Collaboration in Global Software Engineering Teams: Explorations of 'Collaborative Technology Fit'. In: *Fourth IEEE International Conference on Global Software Engineering, 2009. ICGSE 2009*. 2009, pp. 297–298.

- Cohen, L., Manion, L. & Morrison, K. (2009). *Research methods in education*. London: Routledge.
- Coleman, G. & O'Connor, R. (2007). Using grounded theory to understand software process improvement: A study of Irish software product companies. *Information and Software Technology*. 49 (6). p.pp. 654–667.
- Concas, G., Di Penta, M., Tempero, E. & Zhang, H. (2011). Workshop on Emerging Trends in Software Metrics (WETSoM 2011). In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. [Online]. 2011, New York, NY, USA: ACM, pp. 1224–1225. Available from: <http://doi.acm.org/10.1145/1985793.1986057>. [Accessed: 29 September 2014].
- Concato, J., Shah, N. & Horwitz, R.I. (2000). Randomized, Controlled Trials, Observational Studies, and the Hierarchy of Research Designs. *New England Journal of Medicine*. 342 (25). p.pp. 1887–1892.
- Creswell, J.W. (2002). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 2 edition. Thousand Oaks, Calif: SAGE Publications, Inc.
- Cronin, B. & Sugimoto, C.R. (2014). Google-Books-ID: BVuaAWAAQBAJ. *Beyond Bibliometrics: Harnessing Multidimensional Indicators of Scholarly Impact*. MIT Press.
- Crotty, M. (1998). Google-Books-ID: j4hXocGn1yIC. *The Foundations of Social Research: Meaning and Perspective in the Research Process*. SAGE.
- Csa (2013). Mapping the Forensic Standard ISO/ IEC27037 to Cloud Computing.pdf. *Cloud Security Alliance*. (June). p.pp. 1–31.
- Cumming, B. (2012). Revisiting Philosophical and Theoretical Debates in Contemporary Educational Research and Major Epistemological and Ontological Underpinnings. *Online Submission*. [Online]. Available from: <http://eric.ed.gov/?id=ED537463>. [Accessed: 28 November 2016].
- Dabbish, L., Stuart, C., Tsay, J. & Herbsleb, J. (2012). Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*. CSCW '12. [Online]. 2012, New York, NY, USA: ACM, pp. 1277–1286. Available from: <http://doi.acm.org/10.1145/2145204.2145396>. [Accessed: 19 March 2014].
- Dafoulas, G.A., Swigger, K., Brazile, R., Alpaslan, F.N., Cabrera, V.L. & Serce, F.C. (2009). Global Teams: Futuristic Models of Collaborative Work for Today's Software Development Industry. In: *2009 42nd Hawaii International Conference on System Sciences*. January 2009, pp. 1–10.
- De Souza, C. (2003). Interpreting activity theory as a software engineering methodology. In: *présenté dans l'atelier: Applying Activity Theory to CSCW research and practice du 8th European Conference of Computer-Supported Cooperative Work, Helsinki, Finland*. 2003.

- Dennehy, D. & Conboy, K. (2016). Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software*.
- Dennehy, D. & Conboy, K. (2017). Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software*. 133. p.pp. 160–173.
- Denscombe, M. (2010). *The Good Research Guide: For Small-Scale Social Research Projects: for small-scale social research projects*. 4 edition. Open University Press.
- Derntl, M., Renzel, D., Nicolaescu, P., Koren, I. & Klamma, R. (2015). Distributed Software Engineering in Collaborative Research Projects. In: *2015 IEEE 10th International Conference on Global Software Engineering*. July 2015, pp. 105–109.
- Dey, A.K. (2001). Understanding and Using Context. *Personal Ubiquitous Comput.* 5 (1). p.pp. 4–7.
- Dijkman, R. & Dumas, M. (2004). Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*. 13 (4). p.pp. 337–368.
- Dillon, T., Wu, C. & Chang, E. (2010). Cloud Computing: Issues and Challenges. In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*. 2010, pp. 27–33.
- Doddavula, S.K., Agrawal, I. & Saxena, V. (2013). Cloud Computing Solution Patterns: Infrastructural Solutions. In: Z. Mahmood (ed.). *Cloud Computing*. Computer Communications and Networks. [Online]. Springer London, pp. 197–219. Available from: http://link.springer.com/chapter/10.1007/978-1-4471-5107-4_10. [Accessed: 17 July 2013].
- Durao, F., Carvalho, J.F.S., Fonseca, A. & Garcia, V.C. (2014). A systematic review on cloud computing. *The Journal of Supercomputing*. 68 (3). p.pp. 1321–1346.
- Dybå, T. & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*. 50 (9–10). p.pp. 833–859.
- Dybå, T., Sjöberg, D.I.K. & Cruzes, D.S. (2012). What works for whom, where, when, and why? On the role of context in empirical software engineering. In: *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. September 2012, pp. 19–28.
- Easterby-Smith, M., Thorpe, R. & Jackson, P.R. (2012). *Management Research*. SAGE.
- Elizabeth, M. (2013). Google-Books-ID: iOSWBQAAQBAJ. *Activity Theory Perspectives on Technology in Higher Education*. IGI Global.
- Ellman, J., Lee, N. & Jin, N. (2018). Cloud computing deployment: a cost-modelling case-study. *Wireless Networks*. [Online]. Available from: <https://doi.org/10.1007/s11276-018-1881-2>. [Accessed: 31 March 2020].
- van Engelenburg, S., Janssen, M. & Klievink, B. (2019). Designing context-aware systems: A method for understanding and analysing context in practice. *Journal of Logical and Algebraic Methods in Programming*. 103. p.pp. 79–104.

- Engestrom, Y. (2000). Activity theory as a framework for analyzing and redesigning work. *Ergonomics*. 43 (7). p.pp. 960–974.
- Engeström, Y. (2001). Expansive learning at work: Toward an activity theoretical reconceptualization. *Journal of education and work*. 14 (1). p.pp. 133–156.
- Engeström, Y., Miettinen, R. & Punamäki, R.-L. (1999). *Perspectives on Activity Theory*. Cambridge University Press.
- Erickson, J.S., Spence, S., Rhodes, M., Banks, D., Rutherford, J., Simpson, E., Belrose, G. & Perry, R. (2009). Content-Centered Collaboration Spaces in the Cloud. *IEEE Internet Computing*. 13 (5). p.pp. 34–42.
- Eriksson, P. & Kovalainen, A. (2015). *Qualitative Methods in Business Research: A Practical Guide to Social Research*. SAGE.
- Evans, L. (2009). 00001. *Reflective Assessment and Student Achievement in High School English*. ProQuest.
- Ewenike, S., Benkhelifa, E. & Chibelushi, C. (2017a). Cloud based collaborative software development: A review, gap analysis and future directions. In: *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. 2017, IEEE, pp. 901–909.
- Ewenike, S., Benkhelifa, E. & Chibelushi, C. (2017b). *Cloud Based Collaborative Software Development: A Review, Gap Analysis and Future Directions*. In: 1 October 2017, pp. 901–909.
- Ewenike, S., Benkhelifa, E. & Chibelushi, C. (2010). *Systematic Review of Trends and Gaps in Collaborative Software Engineering in the Cloud*.
- Exman, I., Perry, D.E., Barn, B. & Ralph, P. (2016). Separability Principles for a General Theory of Software Engineering: Report on the GTSE 2015 Workshop. *SIGSOFT Softw. Eng. Notes*. 41 (1). p.pp. 25–27.
- Fazil, Q.A.A., Abdullah, Z. & Noah, S.A.M. (2010). Applying Zachman Framework to determine the content of semantic theses digital library. In: *2010 International Symposium on Information Technology*. June 2010, pp. 1596–1600.
- Fereday, J. & Muir-Cochrane, E. (2006). Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development. *International Journal of Qualitative Methods*. 5 (1). p.pp. 80–92.
- Fisher, C.D. (2017). Padlet: An Online Tool for Learner Engagement and Collaboration, Available at <https://Padlet.com>. *Academy of Management Learning & Education*. 16 (1). p.pp. 163–165.
- Fleming, S.D., Scaffidi, C., Piorkowski, D., Burnett, M., Bellamy, R., Lawrance, J. & Kwan, I. (2013). An Information Foraging Theory Perspective on Tools for Debugging, Refactoring, and Reuse Tasks. *ACM Trans. Softw. Eng. Methodol.* 22 (2). p.p. 14:1-14:41.

Foley, M.J. (2013). *Microsoft is pushing to move its internal software development to the cloud*. [Online]. 8 August 2013. ZDNet. Available from: <http://www.zdnet.com/microsoft-is-pushing-to-move-its-internal-software-development-to-the-cloud-7000019159/>. [Accessed: 20 July 2014].

Folkestad, B. (2008). Analysing Interview Data Possibilities and challenges. *undefined*. [Online]. Available from: </paper/Analysing-Interview-Data-Possibilities-and-Folkestad/0a5f03bcf2b7cbaaab910721705db9e3000ed876>. [Accessed: 15 June 2021].

Franken, S., Kolvenbach, S., Prinz, W., Alvertis, I. & Koussouris, S. (2015). CloudTeams: Bridging the Gap Between Developers and Customers During Software Development Processes. *Procedia Computer Science*. 68. p.pp. 188–195.

Frans Prenekert (2006). A theory of organizing informed by activity theory: The locus of paradox, sources of change, and challenge to management. *Journal of Organizational Change Management*. 19 (4). p.pp. 471–490.

Fusaro, V.A., Patil, P., Gafni, E., Wall, D.P. & Tonellato, P.J. (2011). Biomedical Cloud Computing With Amazon Web Services. *PLOS Computational Biology*. 7 (8). p.p. e1002147.

Fylaktopoulos, G., Goumas, G., Skolarikis, M., Sotiropoulos, A. & Maglogiannis, I. (2016a). An overview of platforms for cloud based development. *SpringerPlus*. [Online]. 5. Available from: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4715041/>. [Accessed: 30 October 2018].

Fylaktopoulos, G., Goumas, G., Skolarikis, M., Sotiropoulos, A. & Maglogiannis, I. (2016b). An overview of platforms for cloud based development. *SpringerPlus*. 5 (1). p.p. 38.

Gadea, C., Solomon, B., Ionescu, B. & Ionescu, D. (2011). A Collaborative Cloud-Based Multimedia Sharing Platform for Social Networking Environments. In: *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. July 2011, pp. 1–6.

Gai, K. & Li, S. (2012). Towards Cloud Computing: A Literature Review on Cloud Computing and Its Development Trends. In: *2012 Fourth International Conference on Multimedia Information Networking and Security*. November 2012, pp. 142–146.

Gao, J., Bai, X. & Tsai, W.-T. (2011). Cloud Testing-Issues, Challenges, Needs and Practice. *Software Engineering: An International Journal*. 1 (1). p.pp. 9–23.

J. Garbajosa, X. Wang, & A. Aguiar (eds.) (2018). *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. [Online]. Cham: Springer International Publishing. Available from: <http://link.springer.com/10.1007/978-3-319-91602-6>. [Accessed: 22 February 2019].

Garfinkel, S. (2011). Cloud Computing Defined. *MIT Technology Review*. [Online]. Available from: <http://www.technologyreview.com/news/425618/cloud-computing-defined/>. [Accessed: 29 May 2014].

- Georg, G. (2011). Activity theory and its applications in software engineering and technology. *Colorado State University Technical Report CS-11-101*. 1025.
- Geszten, D., Hámornik, B.P. & Hercegi, K. (2018). Exploring awareness related usability problems of collaborative software with a team usability testing approach. In: *2018 9th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. August 2018, pp. 000045–000050.
- Ghaffari, K., Delgosha, M.S. & Abdolvand, N. (2014). Towards Cloud Computing: A SWOT Analysis on its Adoption in SMEs. *International Journal of Information Technology Convergence and Services*. 4 (2). p.pp. 13–20.
- Ghandehari, M. & Stroulia, E. (2014). A Lightweight Coordination Approach For Resource-Centric Collaborations. In: C. Pautasso, E. Wilde, & R. Alarcon (eds.). *REST: Advanced Research Topics and Practical Applications*. [Online]. New York, NY: Springer, pp. 147–165. Available from: https://doi.org/10.1007/978-1-4614-9299-3_9. [Accessed: 31 March 2020].
- C. Ghaoui (ed.) (2006). *Encyclopedia of Human Computer Interaction*. Pck edition. Hershey PA: IGI Global.
- Ghezzi, C., Jazayeri, M. & Mandrioli, D. (2002). 01395. *Fundamentals of Software Engineering*. 2nd Ed. Upper Saddle River, NJ, USA: Prentice Hall PTR.
- Gill, S.S. & Chana, I. (2012). Cloud Based Development Issues: A Methodical Analysis. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*. 2.
- Given, L.M. (2008). *The Sage encyclopedia of qualitative research methods*. [Online]. Sage Publications. Available from: [http://books.google.co.uk/books?hl=en&lr=&id=byh1AwAAQBAJ&oi=fnd&pg=PP1&dq=+Rot+hbauer,+Paulette+\(2008\)+%22Triangulation.%22+In+Given,+Lisa+\(Ed.\),+%22The+SAGE+Encyclopedia+of+Qualitative+Research+Methods.%22+Sage+Publications.+pp.+892-894.+Jump+up+%5E&ots=LNYOKN6N8n&sig=RuiHQd2nUGqNUOwfho8I2yc4ZZ8](http://books.google.co.uk/books?hl=en&lr=&id=byh1AwAAQBAJ&oi=fnd&pg=PP1&dq=+Rot+hbauer,+Paulette+(2008)+%22Triangulation.%22+In+Given,+Lisa+(Ed.),+%22The+SAGE+Encyclopedia+of+Qualitative+Research+Methods.%22+Sage+Publications.+pp.+892-894.+Jump+up+%5E&ots=LNYOKN6N8n&sig=RuiHQd2nUGqNUOwfho8I2yc4ZZ8). [Accessed: 21 June 2017].
- Goede, P.A., Lauman, J.R., Cochella, C., Katzman, G.L., Morton, D.A. & Albertine, K.H. (2004). A Methodology and Implementation for Annotating Digital Images for Context-appropriate Use in an Academic Health Care Environment. *Journal of the American Medical Informatics Association : JAMIA*. 11 (1). p.pp. 29–41.
- Golightly, D., Sharples, S., Patel, H. & Ratchev, S. (2016). Manufacturing in the cloud: A human factors perspective. *International Journal of Industrial Ergonomics*. 55. p.pp. 12–21.
- Gordon, A.J. (2013). Concepts for mobile programming. In: *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. ITiCSE '13. [Online]. 2013, New York, NY, USA: ACM, pp. 58–63. Available from: <http://doi.acm.org/10.1145/2462476.2462483>. [Accessed: 17 July 2013].
- Gorlick, M.M. & Taylor, R.N. (2014). Communication and Capability URLs in COAST-based Decentralized Services. In: C. Pautasso, E. Wilde, & R. Alarcon (eds.). *REST: Advanced*

- Research Topics and Practical Applications*. [Online]. New York, NY: Springer, pp. 9–25. Available from: https://doi.org/10.1007/978-1-4614-9299-3_2. [Accessed: 31 March 2020].
- Gorton, I., Bener, A.B. & Mockus, A. (2016). Software Engineering for Big Data Systems. *IEEE Software*. 33 (2). p.pp. 32–35.
- Grbich, C. (2012). Google-Books-ID: B4dkAEIuDz4C. *Qualitative Data Analysis: An Introduction*. SAGE.
- Gregor, S. (2006). The nature of theory in information systems. *Mis Quarterly*. 30 (3). p.pp. 611–642.
- Gregor, S. & Jones, D. (2007). The anatomy of a design theory. *Journal of the Association for Information Systems*. 8 (5). p.pp. 312–335.
- Guba, E.G., Lincoln, Y.S., & others (1994). Competing paradigms in qualitative research. *Handbook of qualitative research*. 2 (163–194). p.p. 105.
- Guha, R. & Al-Dabass, D. (2010). Impact of Web 2.0 and Cloud Computing Platform on Software Engineering. In: *2010 International Symposium on Electronic System Design*. December 2010, pp. 213–218.
- Guillén, J., Miranda, J., Murillo, J.M. & Canal, C. (2013). A service-oriented framework for developing cross cloud migratable software. *Journal of Systems and Software*. [Online]. Available from: <http://www.sciencedirect.com/science/article/pii/S0164121212003421>. [Accessed: 3 June 2013].
- Haig-Smith, T. & Tanner, M. (2016). *Cloud Computing as an Enabler of Agile Global Software Development*. 13. p.p. 24.
- Hajjdiab, H. & Al Shaima Taleb (2011). Adopting Agile Software Development: Issues and Challenges. *International Journal of Managing Value and Supply Chains*. 2 (3). p.pp. 1–10.
- Han, B.J., Jung, I.-Y., Kim, K.-H., Lee, D., Rho, S. & Jeong, C. (2013). Cloud-based active content collaboration platform using multimedia processing. *EURASIP Journal on Wireless Communications and Networking*. 2013 (1). p.pp. 1–13.
- Hansen, B.H. & Kautz, K. (2005). Grounded theory applied-studying information systems development methodologies in practice. In: *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*. 2005, IEEE, pp. 264b–264b.
- Hashmi, S.I. (2013). *Global requirements engineering on the cloud: PhD research proposal*. [Online]. Available from: <http://ulir.ul.ie/handle/10344/3482>. [Accessed: 19 March 2014].
- Hazeyama, A., Aoki, H., Ishikawa, Y., Ito, H., Ogane, K., Okazaki, J., Kobayashi, Y., Miura, M. & Yamashita, I. (2007). *A Survey on Software Development Support Based on Collaborative Learning Theories*. In: 2007.
- HAZEYAMA, A., AOKI, H., ISHIKAWA, Y., ITOa, H., OGANE, K., OKAZAKI, J., KOBAYASHI, Y., MIURA, M. & YAMASHITA, I. (n.d.). *A Survey on Software Development Support Based on Collaborative Learning Theories*. [Online]. Available from: <http://www.u->

gakugei.ac.jp/~hazeyama/papers/ICCE2007-poster-Hazeyama.pdf. [Accessed: 5 December 2015].

Henneman, E.A., Lee, J.L. & Cohen, J.I. (1995). Collaboration: a concept analysis. *Journal of Advanced Nursing*. 21 (1). p.pp. 103–109.

Herbsleb, J.D. (2007). Global Software Engineering: The Future of Socio-technical Coordination. In: *2007 Future of Software Engineering*. FOSE '07. [Online]. 2007, Washington, DC, USA: IEEE Computer Society, pp. 188–198. Available from: <http://dx.doi.org/10.1109/FOSE.2007.11>. [Accessed: 19 March 2014].

Herbsleb, J.D., Paulish, D.J. & Bass, M. (2005). Global Software Development at Siemens: Experience from Nine Projects. In: *Proceedings of the 27th International Conference on Software Engineering*. ICSE '05. [Online]. 2005, New York, NY, USA: ACM, pp. 524–533. Available from: <http://doi.acm.org/10.1145/1062455.1062550>. [Accessed: 2 August 2017].

Hildenbrand, T., Rothlauf, F., Geisser, M., Heinzl, A. & Kude, T. (2008). Approaches to Collaborative Software Development. In: *International Conference on Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008*. March 2008, pp. 523–528.

Hiremath, M.M. & Patil, A.P. (2015). Collaboration in multi-cloud computing environments: Framework and security issues. *IJCSIT*. 6 (3). p.pp. 2859–62.

Hodges, J. (2002). *Lightweight directory access protocol (v3): Technical specification*.

Hoegl, M. & Gemuenden, H.G. (2001). Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence. *Organization Science*. 12 (4). p.pp. 435–449.

Hoegl, M., Weinkauff, K. & Gemuenden, H.G. (2004). Interteam Coordination, Project Commitment, and Teamwork in Multiteam R&D Projects: A Longitudinal Study. *Organization Science*. 15 (1). p.pp. 38–55.

Hong, J., Suh, E. & Kim, S.-J. (2009). Context-aware systems: A literature review and classification. *Expert Systems with Applications*. 36 (4). p.pp. 8509–8522.

Howard, C., Plummer, D.C., Genovese, Y., Mann, J., Willis, D.A. & Smith, D.M. (2012). The nexus of forces: social, mobile, cloud and information. *On-line at <http://www.gartner.com/technology/research/nexus-of-forces>*.

Isaeva, N., Bachmann, R., Bristow, A. & Saunders, M.N.K. (2015). Why the epistemologies of trust researchers matter. *Journal of Trust Research*. 5 (2). p.pp. 153–169.

Jackson, B. (2011). Cloud Collaboration. *Mix*. 35 (5). p.pp. 16–18.

Jadeja, Y. & Modi, K. (2012). Cloud computing - concepts, architecture and challenges. In: *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*. 2012, pp. 877–880.

- Jastroch, N. (2009). *Advancing Adaptivity in Enterprise Collaboration*. [Online]. Rochester, NY: Social Science Research Network. Available from: <https://papers.ssrn.com/abstract=1907348>. [Accessed: 15 February 2017].
- Jeffery, R. (2000). Theory, models and methods in software engineering research. In: *ICSE'2000 Workshop on " Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research"(2000)*. [Online]. 2000, pp. 2–7. Available from: https://www.researchgate.net/profile/R_Jeffery2/publication/2909335_Theory_Models_and_Methods_in_Software_Engineering_Research/links/543347d00cf22395f29e0a37.pdf. [Accessed: 8 February 2017].
- Jespersen, J. (2011). Google-Books-ID: y12RtMXthCOC. *Macroeconomic Methodology: A Post-Keynesian Perspective*. Edward Elgar Publishing.
- John, B.E., Swart, C., Bellamy, R.K.E., Blackmon, M.H. & Brown, R. (2013). An Open Source Approach to Information Scint. In: *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '13. [Online]. 2013, New York, NY, USA: ACM, pp. 355–360. Available from: <http://doi.acm.org/10.1145/2468356.2468419>. [Accessed: 4 May 2016].
- Johnson, J.R., Burnell-Nugent, M., Lossignol, D., Ganae-Motan, E.D., Potts, R. & Fallon, M.T. (2010). Multicenter, Double-Blind, Randomized, Placebo-Controlled, Parallel-Group Study of the Efficacy, Safety, and Tolerability of THC:CBD Extract and THC Extract in Patients with Intractable Cancer-Related Pain. *Journal of Pain and Symptom Management*. 39 (2). p.pp. 167–179.
- Johnson, P. & Ekstedt, M. (2016). The Tarpit – A general theory of software engineering. *Information and Software Technology*. 70. p.pp. 181–203.
- Jr, H.N.B. & Boone, D.A. (2012). Analyzing Likert Data. *Journal of Extension*. [Online]. 50 (2). Available from: <https://www.joe.org/joe/2012april/tt2.php>. [Accessed: 7 December 2016].
- Jugder, N. (2016). *The thematic analysis of interview data: an approach used to examine the influence of the market on curricular provision in Mongolian higher education institutions*.
- Jun, W. & Meng, F. (2011). Software Testing Based on Cloud Computing. In: *2011 International Conference on Internet Computing Information Services (ICICIS)*. 2011, pp. 176–178.
- Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L. & German, D.M. (2015). Open source-style collaborative development practices in commercial projects using github. In: *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. [Online]. 2015, IEEE Press, pp. 574–585. Available from: <http://dl.acm.org/citation.cfm?id=2818825>. [Accessed: 26 February 2017].
- Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M. & Damian, D. (2014). The Promises and Perils of Mining GitHub. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. [Online]. 2014, New York, NY, USA: ACM, pp. 92–101. Available from: <http://doi.acm.org/10.1145/2597073.2597074>. [Accessed: 9 March 2017].

- Kamiński, B. & Szufel, P. (2015). On optimization of simulation execution on Amazon EC2 spot market. *Simulation Modelling Practice and Theory*. 58. p.pp. 172–187.
- Kannan, N. (2012). 6 Ways the Cloud Enhances Agile Software Development. *CIO*. [Online]. Available from: <http://www.cio.com/article/2393022/enterprise-architecture/6-ways-the-cloud-enhances-agile-software-development.html>. [Accessed: 20 July 2014].
- Karanjit, A. (2016). MEAN vs. LAMP Stack. *Culminating Projects in Computer Science and Information Technology*. [Online]. Available from: https://repository.stcloudstate.edu/csit_etds/11.
- Karunakaran, S. (2013). Impact of Cloud Adoption on Agile Software Development. In: Z. Mahmood & S. Saeed (eds.). *Software Engineering Frameworks for the Cloud Computing Paradigm*. Computer Communications and Networks. [Online]. London: Springer, pp. 213–234. Available from: https://doi.org/10.1007/978-1-4471-5031-2_10. [Accessed: 14 June 2021].
- Kats, L.C., Vogelij, R.G., Kalleberg, K.T. & Visser, E. (2012). Software development environments on the web: a research agenda. In: *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. [Online]. 2012, pp. 99–116. Available from: <http://dl.acm.org/citation.cfm?id=2384603>. [Accessed: 25 June 2013].
- Kenneth F. Hyde (2000). Recognising deductive processes in qualitative research. *Qualitative Market Research: An International Journal*. 3 (2). p.pp. 82–90.
- Kim, S.W., Park, S.H., Lee, J., Jin, Y.K., Park, H.-M., Chung, A., Choi, S. & Choi, W.S. (2004). Sensible Appliances: Applying Context-awareness to Appliance Design. *Personal Ubiquitous Comput.* 8 (3–4). p.pp. 184–191.
- Kitchenham, B. & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report.
- Kitchenham, B.A., Dyba, T. & Jorgensen, M. (2004). Evidence-based software engineering. In: *Proceedings of the 26th international conference on software engineering*. [Online]. 2004, IEEE Computer Society, pp. 273–281. Available from: <http://dl.acm.org/citation.cfm?id=999432>. [Accessed: 29 January 2017].
- Klaesson, P. (2013). *Building a scalable social game server*. In: 2013.
- Kocurova, A., Oussena, S., Komisarczuk, P. & Clark, T. (2012). Context-aware content-centric collaborative workflow management for mobile devices. In: *Proceedings of the 2nd International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA'12)*. [Online]. 2012, pp. 54–57. Available from: http://www.eis.mdx.ac.uk/staffpages/tonyclark/Papers/Kocurova_Colla2012-2.pdf. [Accessed: 30 January 2017].

Kokkinos, P., Varvarigou, T.A., Kretsis, A., Soumplis, P. & Varvarigos, E.A. (2015). SuMo: Analysis and Optimization of Amazon EC2 Instances. *Journal of Grid Computing*. 13 (2). p.pp. 255–274.

Kothari, C.R. (2004). *Research methodology: Methods and techniques*. [Online]. New Age International. Available from: https://books.google.com/books?hl=en&lr=&id=hZ9wSHysQDYC&oi=fnd&pg=PA2&dq=%22with+this+object+in+view+are+known+as+descriptive+research%22+%22Desire+to+get+int+ellectual+joy+of+doing+some+creative%22+%22Descriptive+vs.+Analytical:+Descriptive+research+includes+surveys+and+fact-finding%22+%22&ots=1r_cnEf1C7&sig=XT1cB1ZWhtg0_Ux-q44UIRhmRw. [Accessed: 28 September 2016].

Kozulin, A. (1986). The concept of activity in Soviet psychology: Vygotsky, his disciples and critics. *American Psychologist*. 41 (3). p.pp. 264–274.

Kreger, H. & Estefan, J. (2009). Navigating the soa open standards landscape around architecture. *Joint Paper, The Open Group, OASIS, and OMG*.

Krishna, R. & Jayakrishnan, R. (2013). Impact of Cloud Services on Software Development Life Cycle. In: Z. Mahmood & S. Saeed (eds.). *Software Engineering Frameworks for the Cloud Computing Paradigm*. Computer Communications and Networks. [Online]. London: Springer, pp. 79–99. Available from: https://doi.org/10.1007/978-1-4471-5031-2_4. [Accessed: 13 June 2021].

Kyriakidou-Zacharoudiou, A. (2011). *Distributed development of large-scale distributed systems: the case of the particle physics grid*. phd. [Online]. The London School of Economics and Political Science (LSE). Available from: <http://etheses.lse.ac.uk/212/>. [Accessed: 11 July 2016].

Lange, P.D., Nicolaescu, P., Derntl, M., Jarke, M. & Klamma, R. (2016). *Community application editor: collaborative near real-time modeling and composition of microservice-based web applications*. [Online]. Gesellschaft für Informatik e.V. Available from: <http://dl.gi.de/handle/20.500.12116/844>. [Accessed: 18 May 2019].

Lanubile, F. (2009). Collaboration in distributed software development. *Software Engineering*. p.pp. 174–193.

Lanubile, F., Ebert, C., Prikladnicki, R. & Vizcaino, A. (2010). Collaboration Tools for Global Software Engineering. *IEEE Software*. 27 (2). p.pp. 52–55.

Lau, J.W., Lehnert, E., Sethi, A., Malhotra, R., Kaushik, G., Onder, Z., Groves-Kirkby, N., Mihajlovic, A., DiGiovanna, J., Srdic, M., Bajcic, D., Radenkovic, J., Mladenovic, V., Krstanovic, D., Arsenijevic, V., Klisic, D., Mitrovic, M., Bogicevic, I., Kural, D. & Davis-Dusenbery, B. (2017). The Cancer Genomics Cloud: Collaborative, Reproducible, and Democratized—A New Paradigm in Large-Scale Computational Research. *Cancer Research*. 77 (21). p.pp. e3–e6.

Leavitt, N. (2009). Is Cloud Computing Really Ready for Prime Time? *Computer*. 42 (1). p.pp. 15–20.

- Lenk, A., Klems, M., Nimis, J., Tai, S. & Sandholm, T. (2009). What's inside the Cloud? An architectural map of the Cloud landscape. In: *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. CLOUD '09. [Online]. 2009, Washington, DC, USA: IEEE Computer Society, pp. 23–31. Available from: <http://dx.doi.org/10.1109/CLOUD.2009.5071529>. [Accessed: 17 April 2013].
- Lepmets, M. & Nael, M. (2011). Comparison of Plan-driven and Agile Project Management Approaches: Theoretical Bases for a Case Study in Estonian Software Industry. In: *Proceedings of the 2011 Conference on Databases and Information Systems VI: Selected Papers from the Ninth International Baltic Conference, DB&IS 2010*. [Online]. 2011, Amsterdam, The Netherlands, The Netherlands: IOS Press, pp. 296–308. Available from: <http://dl.acm.org/citation.cfm?id=1940590.1940618>. [Accessed: 6 December 2016].
- Lindsjørn, Y., Bergersen, G.R., Dingsøy, T. & Sjøberg, D.I.K. (2018). Teamwork Quality and Team Performance: Exploring Differences Between Small and Large Agile Projects. In: J. Garbajosa, X. Wang, & A. Aguiar (eds.). *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. 2018, Cham: Springer International Publishing, pp. 267–274.
- Lindsjørn, Y., Sjøberg, D., Dingsøy, T., Bergersen, G.R. & Dybå, T. (2016a). *Teamwork Quality and Project Success in Software Development: A Survey of Agile Development Teams*. [Online]. Available from: <https://brage.bibsys.no/xmlui/handle/11250/2420977>. [Accessed: 22 February 2019].
- Lindsjørn, Y., Sjøberg, D.I.K., Dingsøy, T., Bergersen, G.R. & Dybå, T. (2016b). Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*. 122. p.pp. 274–286.
- Linux Foundation (2014). *Collaborative Development Trends Report, 2014*. [Online]. Available from: http://pix.cs.olemiss.edu/csci323/lfw_p_collabdevtrends_v3.pdf. [Accessed: 29 January 2015].
- Liu, T.L., Li, Y.C. & Li, M.L. (2016). A framework of cloud-based collaborative platform to integrate product design requests and contradiction analysis. In: *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*. November 2016, pp. 475–478.
- Lotfy, M. & Pyatt, K. (2018). *A two-course web application development sequence covering the lamp and mean stacks*.
- Lü, J., Rosenblum, D.S., Bultan, T., Issarny, V., Dustdar, S., Storey, M.A. & Zhang, D. (2015). Roundtable: The Future of Software Engineering for Internet Computing. *IEEE Software*. 32 (1). p.pp. 91–97.
- Magdaleno, A.M. (2010a). An Optimization-based Approach to Software Development Process Tailoring. In: *Proceedings of the 2nd International Symposium on Search Based Software Engineering*. SSBSE '10. [Online]. 2010, Washington, DC, USA: IEEE Computer

Society, pp. 40–43. Available from: <http://dx.doi.org/10.1109/SSBSE.2010.15>. [Accessed: 20 June 2013].

Magdaleno, A.M. (2010b). Balancing collaboration and discipline in software development processes. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2. ICSE '10*. [Online]. 2010, New York, NY, USA: ACM, pp. 331–332. Available from: <http://doi.acm.org/10.1145/1810295.1810378>. [Accessed: 7 May 2013].

Magdaleno, A.M., Werner, C.M.L. & Araujo, R.M. de (2012). Reconciling software development models: A quasi-systematic review. *J. Syst. Softw.* 85 (2). p.pp. 351–369.

Mahmood, R., Esfahani, N., Kacem, T., Mirzaei, N., Malek, S. & Stavrou, A. (2012). A whitebox approach for automated security testing of Android applications on the cloud. In: *2012 7th International Workshop on Automation of Software Test (AST)*. 2012, pp. 22–28.

Mahmood, Z. & Saeed, S. (2013). 00001 Cited by 0000. *Software Engineering Frameworks for the Cloud Computing Paradigm*. Springer Publishing Company, Incorporated.

Z. Mahmood & S. Saeed (eds.) (2013). *Software Engineering Frameworks for the Cloud Computing Paradigm*. Computer Communications and Networks. [Online]. London: Springer London. Available from: <http://link.springer.com/10.1007/978-1-4471-5031-2>. [Accessed: 29 March 2020].

Mark, G. (2002). Extreme Collaboration. *Commun. ACM*. 45 (6). p.pp. 89–93.

Marlowe, T. (n.d.). *Addressing Change in Collaborative Software Development: Process and Product Agility and Automated Traceability*. [Online]. Available from: https://www.academia.edu/18729560/Addressing_Change_in_Collaborative_Software_Development_Process_and_Product_Agility_and_Automated_Traceability. [Accessed: 15 February 2017].

Maximilien, E.M. & Campos, P. (2012). Facts, trends and challenges in modern software development. *International Journal of Agile and Extreme Software Development*. 1 (1). p.pp. 1–5.

Méndez Fernández, D. & Passoth, J.-H. (2019). Empirical software engineering: From discipline to interdiscipline. *Journal of Systems and Software*. 148. p.pp. 170–179.

Menzies, T., Williams, L. & Zimmermann, T. (2016). *Perspectives on Data Science for Software Engineering*. 1 edition. Amsterdam Boston Heidelberg: Morgan Kaufmann.

Mertens, D.M. & Hesse-Biber, S. (2012). Triangulation and Mixed Methods Research Provocative Positions. *Journal of Mixed Methods Research*. 6 (2). p.pp. 75–79.

Mirri, S., Prandi, C., Salomoni, P., Callegati, F., Melis, A. & Prandini, M. (2016). *A Service-Oriented Approach to Crowdsensing for Accessible Smart Mobility Scenarios*. [Online]. 2016. Mobile Information Systems. Available from: <https://www.hindawi.com/journals/misy/2016/2821680/>. [Accessed: 15 November 2018].

- Mistrík, I., Ali, N., Kazman, R., Grundy, J. & Schmerl, B. (2016). Google-Books-ID: F803CgAAQBAJ. *Managing Trade-offs in Adaptable Software Architectures*. Morgan Kaufmann.
- Mistrík, I., Grundy, J., Hoek, A. & Whitehead, J. (2010). *Collaborative Software Engineering*. Springer Science & Business Media.
- Mohtashami, M., Kirova, V., Marlowe, T. & Deek, F. (2009). A Comparison of Three Modes of Collaboration for Software Development. *AMCIS 2009 Proceedings*. [Online]. Available from: <http://aisel.aisnet.org/amcis2009/19>.
- Mohtashami, M., Marlowe, T.J., Kirova, V.D. & Deek, F.P. (2011a). Risk-driven Management Contingency Policies in Collaborative Software Development. *International Journal of Information Technology and Management*. 10 (2–4). p.pp. 247–271.
- Mohtashami, M., Marlowe, T.J. & Ku, C.S. (2011b). Metrics Are Needed for Collaborative Software Development. *Journal of Systemics, Cybernetics, and Informatics*. 9 (5). p.pp. 41–47.
- Moiz, S.A. & Rizwanullah, M. (2012). Model based Software Deveelopment: Issues & Challenges. *arXiv preprint arXiv:1203.1314*. [Online]. Available from: <http://arxiv.org/abs/1203.1314>. [Accessed: 25 June 2013].
- Mourad, M.H., Nassehi, A., Schaefer, D. & Newman, S.T. (2020). Assessment of interoperability in cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*. 61. p.p. 101832.
- Munassar, N.M.A. & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. *IJCSI International Journal of Computer Science Issues*. 7 (5). p.pp. 94–101.
- Münch, J. & Schmid, K. (2013). Google-Books-ID: NwREAAAQBAJ. *Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach*. Springer Science & Business Media.
- Murthy, M.S.N. & Suma, V. (2017). Software testing and its scope in CLOUD: A detailed survey. In: *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. February 2017, pp. 269–273.
- National Defense Industrial Association (2010). *NDIA Top SW Issues 2010 Report v5a*. [Online]. September 2010. Available from: <http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/NDIA%20Top%20SW%20Issues%202010%20Report%20v5a%20final.pdf>. [Accessed: 20 March 2014].
- Newman, S. (2015). *Building Microservices*. 1 edition. Beijing Sebastopol, CA: O’Reilly Media.
- Nogueira, E., Moreira, A., Lucrédio, D., Garcia, V. & Fortes, R. (2016). Issues on developing interoperable cloud applications: definitions, concepts, approaches, requirements, characteristics and evaluation models. *Journal of Software Engineering Research and Development*. 4 (1). p.p. 7.

- Noll, J., Beecham, S. & Richardson, I. (2010). Global software development and collaboration: barriers and solutions. *ACM inroads*. 1 (3). p.pp. 66–78.
- Noor, K.B.M. (2008). Case study: A strategic research methodology. *American journal of applied sciences*. 5 (11). p.pp. 1602–1604.
- Nordio, M., Estler, H.-C., Furia, C.A. & Meyer, B. (2011). Collaborative Software Development on the Web. *arXiv:1105.0768 [cs]*. [Online]. Available from: <http://arxiv.org/abs/1105.0768>. [Accessed: 7 November 2016].
- Ntanos, C., Botsikas, C., Rovis, G., Kakavas, P. & Askounis, D. (2014). A context awareness framework for cross-platform distributed applications. *Journal of Systems and Software*. 88. p.pp. 138–146.
- Núñez, I. (2009). Contradictions as Sources of Change: A literature review on Activity Theory and the Utilisation of the Activity System in Mathematics Education. *Educate* ~. 9 (3). p.pp. 7–20.
- Oberhauser, R. (2014). Cloud-based Collaborative Software Development: A Concept for Managing Transparency and Privacy based on Datasteads. *International Journal On Advances in Software*. 7 (3 and 4). p.pp. 435–445.
- Oberhauser, R. (2013a). Towards Cloud-based Collaborative Software Development: A Developer-Centric Concept for Managing Privacy, Security, and Trust. In: *ICSEA 2013, The Eighth International Conference on Software Engineering Advances*. [Online]. 2013, pp. 533–538. Available from: http://www.thinkmind.org/index.php?view=article&articleid=icsea_2013_19_20_10121. [Accessed: 1 September 2014].
- Oberhauser, R. (2013b). *Towards Cloud-based Collaborative Software Development: A Developer-Centric Concept for Managing Privacy, Security, and Trust*. In: [Online]. 27 October 2013, pp. 533–538. Available from: http://www.thinkmind.org/index.php?view=article&articleid=icsea_2013_19_20_10121. [Accessed: 5 February 2015].
- OGC (2013). *OGC Standards | OGC(R)*. [Online]. 2013. OGC. Available from: <http://www.opengeospatial.org/standards/is>. [Accessed: 22 October 2013].
- Oh, Y., Han, J. & Woo, W. (2010). A context management architecture for large-scale smart environments. *IEEE Communications Magazine*. 48 (3). p.pp. 118–126.
- O’Leary, D. (2010). An Activity Theory Framework for DSS for Extreme Events. In: *Frontiers in Artificial Intelligence and Applications*. 1 January 2010, pp. 487–497.
- Omicini, A. (2013). Nature-Inspired Coordination Models: Current Status and Future Trends. *ISRN Software Engineering*. 2013. p.pp. 1–13.
- Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C. & Tummolini, L. (2004). Coordination Artifacts: Environment-Based Coordination for Intelligent Agents. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* -

Volume 1. AAMAS '04. [Online]. 2004, Washington, DC, USA: IEEE Computer Society, pp. 286–293. Available from: <http://dx.doi.org/10.1109/AAMAS.2004.95>. [Accessed: 4 February 2015].

Omoronya, I., Ferguson, J., Roper, M. & Wood, M. (2010). A Review of Awareness in Distributed Collaborative Software Engineering. *Softw. Pract. Exper.* 40 (12). p.pp. 1107–1133.

Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T. & Epema, D. (2010). A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In: D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, & E. Dekel (eds.). *Cloud Computing*. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. 2010, Berlin, Heidelberg: Springer, pp. 115–131.

Oxford Dictionaries (2013). *collaboration: definition of collaboration in Oxford dictionary (American English)*. [Online]. July 2013. Available from: http://oxforddictionaries.com/definition/american_english/collaboration. [Accessed: 4 July 2013].

Panigrahi, C.R., Mall, R. & Pati, B. (2017). *Software Development Methodology for Cloud Computing and Its Impact*. In: 2017.

Pankratius, V. (2010). Google-Books-ID: 74CryOzvO3gC. *Emerging Research Directions in Computer Science: Contributions from the Young Informatics Faculty in Karlsruhe*. KIT Scientific Publishing.

Parveen, T. & Tilley, S. (2010). When to Migrate Software Testing to the Cloud? In: *2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*. 2010, pp. 424–427.

Pathirage, C.P., Amaratunga, R.D.G. & Haigh, R.P. (2007). The role of philosophical context in the development of theory: Towards methodological pluralism. *The Built and Human Environment Review*. [Online]. 1 (1). Available from: <http://www.tbher.org/index.php/bher/issue/view/2>. [Accessed: 28 November 2016].

Patidar, S., Rane, D. & Jain, P. (2012). A Survey Paper on Cloud Computing. In: *2012 Second International Conference on Advanced Computing Communication Technologies*. January 2012, pp. 394–398.

Patton, M.Q. (2002). Google-Books-ID: FjBw2oi8El4C. *Qualitative Research & Evaluation Methods*. SAGE.

Peltz, C. (2003). Web services orchestration and choreography. *Computer*. 36 (10). p.pp. 46–52.

Peng, X., Babar, M.A. & Ebert, C. (2014). Collaborative Software Development Platforms for Crowdsourcing. *IEEE software*. 31 (2). p.pp. 30–36.

Petersen, K. & Wohlin, C. (2009). Context in Industrial Software Engineering Research. In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering*

and Measurement. ESEM '09. [Online]. 2009, Washington, DC, USA: IEEE Computer Society, pp. 401–404. Available from: <http://dx.doi.org/10.1109/ESEM.2009.5316010>. [Accessed: 8 December 2016].

Portocarrero, J.M.T., Delicato, F.C., Pires, P.F., Costa, B., Li, W., Si, W. & Zomaya, A.Y. (2017). RAMSES: A new reference architecture for self-adaptive middleware in Wireless Sensor Networks. *Ad Hoc Networks*. 55. p.pp. 3–27.

Puthal, D., Sahoo, B.P.S., Mishra, S. & Swain, S. (2015). *Cloud Computing Features, Issues, and Challenges: A Big Picture*. In: [Online]. January 2015, IEEE, pp. 116–123. Available from: <http://ieeexplore.ieee.org/document/7053814/>. [Accessed: 4 July 2017].

Quest (2012). *Challenges-Benefits-Cloud-Computing.pdf*. p.pp. 1–10.

Rademacher, F., Sachweh, S. & Zündorf, A. (2017). Differences between model-driven development of service-oriented and microservice architecture. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 2017, IEEE, pp. 38–45.

Raj, P., Venkatesh, V. & Amirtharajan, R. (2013). Envisioning the Cloud-Induced Transformations in the Software Engineering Discipline. In: Z. Mahmood & S. Saeed (eds.). *Software Engineering Frameworks for the Cloud Computing Paradigm*. Computer Communications and Networks. [Online]. London: Springer, pp. 25–53. Available from: https://doi.org/10.1007/978-1-4471-5031-2_2. [Accessed: 14 June 2021].

Ralph, P. (2014). Evaluating process theories in software engineering. In: *Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering*. [Online]. 2014, ACM, pp. 5–8. Available from: <http://dl.acm.org/citation.cfm?id=2593754>. [Accessed: 20 June 2017].

Ralph, P. (2013a). Possible core theories for software engineering. In: *2013 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE)*. May 2013, pp. 35–38.

Ralph, P. (n.d.). *Possible Core Theories for Software Engineering*. [Online]. Available from: <http://paulralph.name/wp-content/uploads/2012/06/Ralph-2013-Possible-Core-Theory-for-Software-Engineering.pdf>. [Accessed: 20 June 2013].

Ralph, P. (2013b). Software Engineering Process Theory: A Multi-Method Comparison of Sensemaking-Coevolution-Implementation Theory and Function-Behavior-Structure Theory. *arXiv:1307.1019 [cs]*. [Online]. Available from: <http://arxiv.org/abs/1307.1019>. [Accessed: 8 February 2017].

Ralph, P., Johnson, P. & Jordan, H. (2013). Report on the First SEMAT Workshop on General Theory of Software Engineering (GTSE 2012). *SIGSOFT Softw. Eng. Notes*. 38 (2). p.pp. 26–28.

Ramis, B., de Juan-Marín, R., Miedes, E., Nieto, N., Martínez Lastra, J.L. & Peña-Ortiz, R. (2016). *Towards a Cloud-based Platform for Enabling Supply Chain Collaboration*. In: 29 March 2016.

Raubenheimer, J. (2014). Google-Books-ID: hmPBoAEACAAJ. *Mendeley: Crowd-sourced Reference and Citation Management in the Infomation Era*. True Insight Publishing.

- Rauch, E., Seidenstricker, S., Dallasega, P. & Hämmerl, R. (2016a). *Collaborative Cloud Manufacturing: Design of Business Model Innovations Enabled by Cyberphysical Systems in Distributed Manufacturing Systems*. [Online]. 2016. Journal of Engineering. Available from: <https://www.hindawi.com/journals/je/2016/1308639/>. [Accessed: 22 November 2018].
- Rauch, E., Seidenstricker, S., Dallasega, P. & Hämmerl, R. (2016b). Collaborative Cloud Manufacturing: Design of Business Model Innovations Enabled by Cyberphysical Systems in Distributed Manufacturing Systems. *Journal of Engineering*. 2016. p.pp. 1–12.
- Rellermeyer, J.S., Lee, S.-W. & Kistler, M. (2013). Cloud platforms and embedded computing: the operating systems of the future. In: *Proceedings of the 50th Annual Design Automation Conference*. DAC '13. [Online]. 2013, New York, NY, USA: ACM, p. 75:1-75:6. Available from: <http://doi.acm.org/10.1145/2463209.2488826>. [Accessed: 17 July 2013].
- Richards, M. (2015a). *Microservices vs. service-oriented architecture*. O'Reilly Media.
- Richards, M. (2015b). *Software architecture patterns*. 1st Ed. [Online]. O'Reilly Media, Inc. Available from: <http://www.oreilly.com/programming/free/files/software-architecture-patterns.pdf>. [Accessed: 28 February 2017].
- Richards, N.F., Mark (n.d.). *Software Architecture Fundamentals Understanding the Basics*. [Online]. Available from: <http://shop.oreilly.com/product/110000195.do>. [Accessed: 11 July 2018].
- Richardson, C. (2019). *Microservice Patterns: With examples in Java*.
- Riungu, L.M., Taipale, O. & Smolander, K. (2010). Research Issues for Software Testing in the Cloud. In: *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*. 2010, pp. 557–564.
- Riungu-Kalliosaari, L., Taipale, O. & Smolander, K. (2012). Testing in the Cloud: Exploring the Practice. *IEEE Software*. 29 (2). p.pp. 46–51.
- Robillard, P.N. & Robillard, M.P. (2000). Types of collaborative work in software engineering. *Journal of Systems and Software*. 53 (3). p.pp. 219–224.
- Rolfe, G. (2006). Validity, trustworthiness and rigour: quality and the idea of qualitative research. *Journal of Advanced Nursing*. 53 (3). p.pp. 304–310.
- Roth, W.-M. & Lee, Y.-J. (2007). “Vygotsky’s Neglected Legacy”: Cultural-Historical Activity Theory. *Review of Educational Research*. 77 (2). p.pp. 186–232.
- T. R. Roth-Berghofer, S. Schulz, & D. B. Leake (eds.) (2006). *Modeling and Retrieval of Context: Second International Workshop, MRC 2005, Edinburgh, UK, July 31-August 1, 2005, Revised Selected Papers*. Lecture Notes in Artificial Intelligence. [Online]. Berlin Heidelberg: Springer-Verlag. Available from: <https://www.springer.com/gp/book/9783540335870>. [Accessed: 15 June 2021].
- Runeson, P. & Höst, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Softw. Engg.* 14 (2). p.pp. 131–164.

- Said, M.N.H.M., Tahir, L.M., Ali, M.F., Noor, N.M., Atan, N.A. & Abdullah, Z. (2014). Using Activity Theory as Analytical Framework for Evaluating Contextual Online Collaborative Learning. *International Journal of Emerging Technologies in Learning (IJET)*. 9 (5). p.pp. 54–59.
- Sanchez, R.V.V., Oliveira, R.R. de & Fortes, R.P. de M. (2014). RestML: Modeling RESTful Web Services. In: C. Pautasso, E. Wilde, & R. Alarcon (eds.). *REST: Advanced Research Topics and Practical Applications*. [Online]. New York, NY: Springer, pp. 125–143. Available from: https://doi.org/10.1007/978-1-4614-9299-3_8. [Accessed: 31 March 2020].
- Sangwan, R.S., Jablokow, K.W. & DeFranco, J.F. (2020). Asynchronous Collaboration: Bridging the Cognitive Distance in Global Software Development Projects. *IEEE Transactions on Professional Communication*. 63 (4). p.pp. 361–371.
- dos Santos, E.C. & Vilain, P. (2018). Automated Acceptance Tests as Software Requirements: An Experiment to Compare the Applicability of Fit Tables and Gherkin Language. In: J. Garbajosa, X. Wang, & A. Aguiar (eds.). *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. 2018, Cham: Springer International Publishing, pp. 104–119.
- Saunders, M.N.K., Lewis, P. & Thornhill, A. (2009). *Research methods for business students*. 5th ed. New York: Prentice Hall.
- Schwartz-Shea, P. & Yanow, D. (2011). *Interpretive Research Design: Concepts and Processes*. 1 edition. New York, NY: Routledge.
- Serçe, F.C., Swigger, K., Alpaslan, F.N., Brazile, R., Dafoulas, G. & Lopez, V. (2011). Online collaboration: Collaborative behavior patterns and factors affecting globally distributed team performance. *Computers in Human Behavior*. 27 (1). p.pp. 490–503.
- da Silva, F.Q., Costa, C., Franca, A.C.C. & Prikladinicki, R. (2010). Challenges and solutions in distributed software development project management: A systematic literature review. In: *2010 5th IEEE International Conference on Global Software Engineering*. 2010, IEEE, pp. 87–96.
- Singh, S. & Chana, I. (2013). *Introducing Agility in Cloud Based Software Development through ASD*.
- Sitaram, D. & Manjunath, G. (2011). *Moving To The Cloud: Developing Apps in the New World of Cloud Computing*. Elsevier.
- Skerrett, I. (2009). Collaborative Software Development in the Enterprise. *Open Source Business Resource*. (January 2009).
- Skourletopoulos, G., Mavromoustakis, C.X., Mastorakis, G., Batalla, J.M., Dobre, C., Panagiotakis, S. & Pallis, E. (2017). Big Data and Cloud Computing: A Survey of the State-of-the-Art and Research Challenges. In: C. X. Mavromoustakis, G. Mastorakis, & C. Dobre (eds.). *Advances in Mobile Cloud Computing and Big Data in the 5G Era*. Studies in Big Data. [Online]. Springer International Publishing, pp. 23–41. Available from:

http://link.springer.com/chapter/10.1007/978-3-319-45145-9_2. [Accessed: 28 February 2017].

Slife, B.D. (1998). Raising the consciousness of researchers: Hidden assumptions in the behavioral sciences. *Adapted Physical Activity Quarterly*. 15 (3). p.p. 208.

Soegaard, M. & Friis Dam, R. (2013). *The Encyclopedia of Human-Computer Interaction, 2nd Ed.*

Somekh, B. & Lewin, C. (2005). Google-Books-ID: qNOJj3avR0wC. *Research Methods in the Social Sciences*. SAGE Publications.

Sommerville, I. (2010). 00000. *Software Engineering*. 9 edition. Boston: Addison Wesley.

Soriano Camino, F.J., López Gómez, G. & Fernández Gallego, R. (2008). Collaborative Development Environments. In: *Encyclopedia of Networked and Virtual Organizations*.

[Online]. EEUU: Facultad de Informática (UPM), pp. 225–231. Available from:

[\[global.com/Bookstore/TitleDetails.aspx?TitleId=369&DetailsType=Description\]\(http://www.igi-global.com/Bookstore/TitleDetails.aspx?TitleId=369&DetailsType=Description\). \[Accessed: 30 April 2013\].](http://www.igi-</p></div><div data-bbox=)

de Souza, C.R. & Redmiles, D.F. (2003). Using Activity Theory to Understand Contradictions in Collaborative Software Development. *Automated Software Engineering, Montreal, CA, IEEE Press*.

Souza, R. (2010). *Interaction Design and Activity Theory: designing for social code review*.

[Online]. Available from: <http://mdsoar.org/handle/11603/3712>. [Accessed: 23 April 2018].

Spinuzzi, C. (2015). Toward a Typology of Activities: Understanding Internal Contradictions in Multiperspectival Activities. *Journal of Business and Technical Communication*. 29 (1).

p.pp. 3–35.

Sriram, I. & Khajeh-Hosseini, A. (2010). Research agenda in cloud technologies. *arXiv preprint arXiv:1001.3259*.

[Online]. Available from: <http://arxiv.org/abs/1001.3259>.

[Accessed: 10 September 2013].

Stol, K.-J. & Fitzgerald, B. (2018). The ABC of Software Engineering Research. *ACM Transactions on Software Engineering and Methodology*. 27 (3). p.p. 11:1-11:51.

Stol, K.-J. & Fitzgerald, B. (2013). *Uncovering Theories in Software Engineering*. [Online].

Available from: http://staff.lero.ie/stol/files/2013/03/uncovering_theories_se.pdf.

[Accessed: 20 June 2013].

Strode, D.E. (2016). A dependency taxonomy for agile software development projects. *Information Systems Frontiers*. 18 (1). p.pp. 23–46.

Strode, D.E. (2012). *A theory of coordination in agile software development projects*.

Strode, D.E. (2012). *A theory of coordination in agile software development projects*.

Strode, D.E., Huff, S.L., Hope, B. & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*. 85 (6). p.pp. 1222–1238.

- Taušan, N., Markkula, J., Kuvaja, P. & Oivo, M. (2017). Choreography in the embedded systems domain: A systematic literature review. *Information and Software Technology*. 91. p.pp. 82–101.
- Taylor, S.J., Bogdan, R. & DeVault, M. (2015). *Introduction to qualitative research methods: A guidebook and resource*. [Online]. John Wiley & Sons. Available from: <http://books.google.co.uk/books?hl=en&lr=&id=pONoCgAAQBAJ&oi=fnd&pg=PR11&dq=Qualitative+research+in+education:+An+introduction+to+theory+and+methods.+&ots=QgBlay6D0N&sig=Fx-F9CJQvmh18BTskdcMDPV4hyc>. [Accessed: 21 June 2017].
- Tell, P. & Babar, M.A. (2012). Activity Theory Applied to Global Software Engineering: Theoretical Foundations and Implications for Tool Builders. In: *2012 IEEE Seventh International Conference on Global Software Engineering*. August 2012, pp. 21–30.
- Thanh, N.C. & Thanh, T.T. (2015). The interconnection between interpretivist paradigm and qualitative methods in Education. *American Journal of Educational Science*. 1 (2). p.pp. 24–27.
- Thomson, A.M. & Perry, J.L. (2006). Collaboration Processes: Inside the Black Box. *Public Administration Review*. 66. p.pp. 20–32.
- Thomson, A.M., Perry, J.L. & Miller, T.K. (2009a). Conceptualizing and Measuring Collaboration. *Journal of Public Administration Research and Theory*. 19 (1). p.pp. 23–56.
- Thomson, A.M., Perry, J.L. & Miller, T.K. (2009b). Conceptualizing and measuring collaboration. *Journal of Public Administration Research and Theory*. 19. p.pp. 23–56.
- Tilley, S. & Parveen, T. (2010). Migrating software testing to the cloud. In: *2010 IEEE International Conference on Software Maintenance (ICSM)*. 2010, pp. 1–1.
- Tsai, W.T., Wu, W. & Huhns, M.N. (2014). Cloud-Based Software Crowdsourcing. *IEEE Internet Computing*. 18 (3). p.pp. 78–83.
- Uden, L., Aranda, P.J.V. & López, O.P. (2008). An activity-theory-based model to analyse Web application requirements. *Information Research*. 13 (2). p.p. 1.
- Ulhaq, S., Raza, M., Zia, A. & Naeem Ahmed Khan, M. (2011). Issues in Global Software Development: A Critical Review. *JSEA*. 4. p.pp. 590–595.
- Urquhart, C. (2012). Google-Books-ID: TivDdBx80YIC. *Grounded Theory for Qualitative Research: A Practical Guide*. SAGE.
- Valilai, O.F. & Houshmand, M. (2013). A collaborative and integrated platform to support distributed manufacturing system using a service-oriented approach based on cloud computing paradigm. *Robotics and Computer-Integrated Manufacturing*. 29 (1). p.pp. 110–127.
- Varaee, T., Habibi, J. & Mohaghar, A. (2015). Presenting an Approach for Conducting Knowledge Architecture within Large-Scale Organizations. *PLOS ONE*. 10 (5). p.p. e0127005.

Vilela, J., Castro, J. & Pimentel, J. (2016). A systematic process for obtaining the behavior of context-sensitive systems. *Journal of Software Engineering Research and Development*. 4 (1). p.p. 2.

Walliman, D.N. (2005). *Your Research Project: A Step-by-Step Guide for the First-Time Researcher*. Second Edition edition. Sage Publications Ltd.

Warth, B., Levin, N., Rinehart, D., Teijaro, J., Benton, H.P. & Siuzdak, G. (2017). Metabolizing Data in the Cloud. *Trends in Biotechnology*. [Online]. Available from: <http://www.sciencedirect.com/science/article/pii/S0167779916302335>. [Accessed: 28 February 2017].

Webb, T.L., Joseph, J., Yardley, L. & Michie, S. (2010). Using the Internet to Promote Health Behavior Change: A Systematic Review and Meta-analysis of the Impact of Theoretical Basis, Use of Behavior Change Techniques, and Mode of Delivery on Efficacy. *Journal of Medical Internet Research*. [Online]. 12 (1). Available from: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2836773/>. [Accessed: 21 June 2017].

Weimar, E., Nugroho, A., Visser, J., Plaat, A., Goudbeek, M. & Schouten, A.P. (2017). The influence of teamwork quality on software team performance. *arXiv preprint arXiv:1701.06146*.

Whaiduzzaman, M., Sookhak, M., Gani, A. & Buyya, R. (2014). A survey on vehicular cloud computing. *Journal of Network and Computer Applications*. 40. p.pp. 325–344.

Whitehead, J. (2007). Collaboration in Software Engineering: A Roadmap. In: *Future of Software Engineering, 2007. FOSE '07*. May 2007, pp. 214–225.

Whitehead, J., Mistrík, I., Grundy, J. & van der Hoek, A. (2010). Collaborative Software Engineering: Concepts and Techniques. In: I. Mistrík, J. Grundy, A. Hoek, & J. Whitehead (eds.). *Collaborative Software Engineering*. [Online]. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–30. Available from: https://doi.org/10.1007/978-3-642-10294-3_1. [Accessed: 22 November 2018].

Willis, J.W. (2007). *Foundations of Qualitative Research: Interpretive and Critical Approaches*. Thousand Oaks: SAGE Publications, Inc.

Wolff, E. (2016). Google-Books-ID: zucwDQAAQBAJ. *Microservices: Flexible Software Architecture*. Addison-Wesley Professional.

Wolff-Piggott, B. & Rivett, U. (2016). *An activity theory approach to affordance actualisation in mHealth: The case of MomConnect*.

Yigitbasioglu, O.M. (2014). *Cloud Computing Adoption in Australia: Evidence from the Forensic Accounting Industry - DO NOT CITE*.

Zafar, A.A., Saif, S., Khan, M., Iqbal, J., Akhunzada, A., Wadood, A., Al-Mogren, A. & Alamri, A. (2018). Taxonomy of Factors Causing Integration Failure during Global Software Development. *IEEE Access*. 6. p.pp. 22228–22239.

Zahedi, M., Tessier, V. & Hawey, D. (2017). Understanding Collaborative Design Through Activity Theory. *The Design Journal*. 20. p.pp. S4611–S4620.

Zardari, S. & Bahsoon, R. (2011). Cloud Adoption: A Goal-oriented Requirements Engineering Approach. In: *Proceedings of the 2Nd International Workshop on Software Engineering for Cloud Computing*. SE-CLOUD '11. [Online]. 2011, New York, NY, USA: ACM, pp. 29–35. Available from: <http://doi.acm.org/10.1145/1985500.1985506>. [Accessed: 19 March 2014].

Zhang, H. & Ali Babar, M. (2013). Systematic reviews in software engineering: An empirical investigation. *Information and Software Technology*. 55 (7). p.pp. 1341–1354.

Zimmermann, O. (2017). Microservices tenets. *Computer Science - Research and Development*. 32 (3). p.pp. 301–310.

Zimmermann, T. & Bird, C. (2012). Collaborative Software Development in Ten Years: Diversity, Tools, and Remix Culture. In: *Proceedings of the Workshop on The Future of Collaborative Software Development*. [Online]. 2012. Available from: <http://thomas-zimmermann.com/publications/files/zimmermann-futurecsd-2012.pdf>. [Accessed: 1 August 2013].

10 Appendices

A. List of publications

Some parts of this PhD thesis have been published or submitted for publication in peer-reviewed conferences. Some other parts of the thesis have been presented at academic seminars. Please see list below:

Publications

1. Ewenike, S., Benkhelifa, E., & Chibelushi C, 2017. Cloud Based Collaborative Software Development: A Review, Gap Analysis and Future Directions. Submitted & presented at the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA) Tunisia. IEEE. Accepted & published.
2. Ewenike, S., Benkhelifa, E., & Chibelushi C, 2017. Classifying collaborative approaches for Cloud Based Collaborative Software Development. Submitted & presented at the 2017 International Conference on the Frontiers and Advances in Data Science (FADS). IEEE. Accepted & published.
3. Ewenike, S., Benkhelifa, E., & Chibelushi C, 2017. Systematic Review of Trends and Gaps in Collaborative Software Engineering for the Cloud. Submitted to Future Technologies Conference. Science and Information (SAI) Organization, Indexed in IEEE Proceedings. Accepted & published.
4. Benkhelifa, E., Abdel-Maguid, M., Ewenike, S. & Heatley, D., 2014. The Internet of Things: The eco-system for sustainable growth. In Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference. IEEE. Accepted & published.

Seminar presentation

1. Ewenike, S., 2019. A systematic approach to select an underpinning theory. Presented at the School of Computing and Digital Technologies Research Seminar series. Staffordshire University. 29th May 2019
2. Ewenike, S., 2017. Systematic Review of Trends and Gaps in Collaborative Software Engineering in the Cloud. Presented at the School of Computing and Digital Technologies Research Seminar series. Staffordshire University. 29th Nov 2017

- Ewenike, S., 2016. Research philosophy and methodology. Presented at the School of Computing and Digital Technologies Research Seminar series. Staffordshire University. 19th Oct 2016

B. POC IMPLEMENTATION – SOURCE CODE

Clients

ProjectMicroServiceClient

```
<?php
namespace App\Clients;
use Config;
use Illuminate\Http\UploadedFile;
use StanleyMSACCommon\MSACCommon\Clients\MicroServiceClient;
class ProjectMicroServiceClient extends MicroServiceClient
{
    public function __construct()
    {
        parent::__construct(Config::get('microservices-end-points.project'));
    }
    public function getPendingProjectsList()
    {
        return $this->request('GET', 'requirement');
    }
    public function createRequirements(array $data, array $files = null)
    {
        $multipart = [];
        foreach ($data as $key => $datum) {
            $multipart[] = [
                'name' => $key,
                'contents' => $datum,
            ];
        }
        if (is_array($files)) {
            foreach ($files as $key => $file) {
                $multipart[] = [
                    'name' => 'files[]',
                    'contents' => fopen($file, 'r'),
                    'filename' => $file->getClientOriginalName()
                ];
            }
        }
        return $this->request('POST', 'requirement', [
            'multipart' => $multipart
        ]);
    }
    public function updateProject($projectId, array $data, array $files = null)
    {
        $multipart = [];
        foreach ($data as $key => $datum) {
            $multipart[] = [
                'name' => $key,
                'contents' => $datum,
            ];
        }
        if (is_array($files)) {
```



```

        foreach ($files as $key => $file) {
            $multipart[] = [
                'name' => 'files[]',
                'contents' => fopen($file, 'r'),
                'filename' => $file->getClientOriginalName()
            ];
        }
    }
    return $this->request('POST', $projectId . '/update', [
        'multipart' => $multipart
    ]);
}
public function getRequirementDetail($projectId)
{
    return $this->request('GET', 'requirement/' . $projectId);
}
public function approveRequirement($projectId, $managerUserId)
{
    return $this->request('POST', 'admin/requirement/' . $projectId .
'/approve', [
        'form_params' => [
            'manager_user_id' => $managerUserId
        ]
    ]);
}
public function deleteRequirement($projectId)
{
    return $this->request('GET', 'requirement/' . $projectId .
'/delete');
}
public function getProjectsListing()
{
    return $this->request('GET', '');
}
public function getProjectDetails($projectId)
{
    return $this->request('GET', $projectId);
}
public function getProjectLogs($projectId)
{
    return $this->request('GET', $projectId . '/logs');
}
public function getProjectTasks($projectId)
{
    return $this->request('GET', $projectId . '/tasks');
}
public function getProjectTaskDetails($projectId, $projectTaskId)
{
    return $this->request('GET', $projectId . '/tasks/' .
$projectTaskId);
}
public function completeProjectTask($projectId, $projectTaskId)
{
    return $this->request('GET', $projectId . '/tasks/' .
$projectTaskId . '/complete');
}
public function createProjectTask($projectId, array $data)
{
    return $this->request('POST', $projectId . '/tasks', [
        'form_params' => $data
    ]);
}

```

```

}
public function exportProjectLogs ($projectId)
{
    return dd($this->request('GET', $projectId . '/logs/export'));
}
public function getNotifications ()
{
    return $this->request('GET', 'notification');
}
public function markNotificationAsRead($notificationId)
{
    return $this->request('GET', 'notification/' . $notificationId .
'/read');
}
public function getProjectQAReports ($projectId)
{
    return $this->request('GET', $projectId.'/qa');
}
public function submitProjectQAReport ($projectId, array $data, array
$files = null)
{
    $multipart = [];
    foreach ($data as $key => $datum) {
        $multipart[] = [
            'name' => $key,
            'contents' => $datum,
        ];
    }
    if (is_array($files)) {
        foreach ($files as $key => $file) {
            $multipart[] = [
                'name' => 'files[]',
                'contents' => fopen($file, 'r'),
                'filename' => $file->getClientOriginalName()
            ];
        }
    }
    return $this->request('POST', $projectId . '/qa', [
        'multipart' => $multipart
    ]);
}
public function getProjectQAReportById($projectId,$qaId)
{
    return $this->request('GET', $projectId.'/qa/'.$qaId);
}
public function closeProject($projectId){
    return $this->request('GET', $projectId.'/complete');
}
public function getProjectsCount(){
    return $this->request('GET', 'admin/project/get-projects-count');
}
public function getRequirementsCount(){
    return $this->request('GET', 'admin/requirement/get-requirements-
count');
}
public function exportBackup(){
    $this->request('GET', 'admin/backup/create-database-dump-file');
    try{
        $path = public_path('backup').'/project.sql';
        $filePath = fopen($path,'w');
        $this->request('GET', 'admin/backup/download-backup-file',

```

```

['sink' => $filePath]);
    }catch (\Exception $exception){
    }
}
}

```

TestingToolMicroServiceClient

```

<?php
namespace App\Clients\Project;
use Config;
use StanleyMSACommon\MSACommon\Clients\MicroServiceClient;
class TestingToolMicroServiceClient extends MicroServiceClient
{
    public function __construct()
    {
        parent::__construct(Config::get('microservices-end-
points.project'));
    }
    public function getList(){
        return $this->request('GET', 'testing-tool');
    }
    public function create(array $data){
        return $this->request('POST', 'admin/testing-tool', [
            'form_params' => $data
        ]);
    }
    public function getTestingToolsCount(){
        return $this->request('GET', 'admin/testing-tool/testing-tools-
count');
    }
}

```

AuthMicroServiceClient - User

```

<?php
namespace App\Clients;

use Config;
use StanleyMSACommon\MSACommon\Clients\MicroServiceClient;

class AuthMicroServiceClient extends MicroServiceClient
{
    public function __construct()
    {
        parent::__construct(Config::get('microservices-end-points.auth'));
    }
    public function getUserById($userId){
        return $this->request('GET', 'user/'.$userId);
    }
    public function getAdminUsers(){
        return $this->request('GET', 'user/admin-users-list');
    }
}

```

AuthMicroServiceClient – App Front

```

<?php
namespace App\Clients;
use Config;
use StanleyMSACommon\MSACommon\Clients\MicroServiceClient;

```

```

class AuthMicroServiceClient extends MicroServiceClient
{
    public function __construct()
    {
        parent::__construct(Config::get('microservices-end-points.auth'));
    }
    public function login($email, $password)
    {
        return $this->request('POST', 'auth/login', [
            'form_params' => [
                'email' => $email,
                'password' => $password
            ]
        ]);
    }
    public function register(array $data)
    {
        $this->request('POST', 'auth/register', [
            'form_params' => $data
        ]);
    }
    public function sendResetPasswordEmail($email)
    {
        $this->request('POST', 'auth/send-forgot-password-email', [
            'form_params' => [
                'email' => $email
            ]
        ]);
    }
    public function resetPassword($password, $confirmedPassword, $token)
    {
        $this->request('POST', 'auth/reset-password', [
            'form_params' => [
                'password' => $password,
                'password_confirmation' => $confirmedPassword,
                'token' => $token
            ]
        ]);
    }
    public function logout() {
        return $this->request('GET', 'auth/logout');
    }
    public function getActiveUsersList() {
        return $this->request('GET', 'admin/user/get-active-users');
    }
    public function getPendingUsersList() {
        return $this->request('GET', 'admin/user/get-pending-users');
    }
    public function activateUser($userId, bool $activate) {
        return $this->request('POST', 'admin/user/'.$userId.'/activate', [
            'form_params' => [
                'activate' => $activate
            ]
        ]);
    }
    public function getUserById($userId) {
        return $this->request('GET', 'user/'.$userId);
    }
    public function getManagersList() {
        return $this->request('GET', 'manager');
    }
}

```

```

public function getEmployeesList () {
    return $this->request('GET', 'employee');
}
public function getDashboardText () {
    return $this->request('GET', 'dashboard-text');
}
public function getUsersCount () {
    return $this->request('GET', 'admin/user/get-users-count');
}
public function updateDashboardText ($dashboardText) {
    return $this->request('POST', 'admin/dashboard-text', [
        'form_params' => [
            'dashboard_text' => $dashboardText
        ]
    ]);
}
public function exportBackup () {
    $this->request('GET', 'admin/backup/create-database-dump-file');
    try{
        $path = public_path('backup').'/auth.sql';
        $filePath = fopen($path, 'w');
        $this->request('GET', 'admin/backup/download-backup-file',
['sink' => $filePath]);
    }catch (\Exception $exception){
    }
}
}
}

```

Providers

AppServiceProvider

```

<?php
namespace App\Providers;
use Illuminate\Support\ServiceProvider;
class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     * @return void
     */
    public function boot()
    {
        //
    }
    /**
     * Register any application services.
     * @return void
     */
    public function register()
    {
        if ($this->app->environment() !== 'production') {
            $this->app-
>register(\Barryvdh\LaravelIdeHelper\IdeHelperServiceProvider::class);
        }
    }
}

```

AuthServiceProvider

```

<?php
namespace App\Providers;
use Illuminate\Support\Facades\Gate;

```

```

use Illuminate\Foundation\Support\Providers\AuthServiceProvider as
ServiceProvider;
class AuthServiceProvider extends ServiceProvider
{
    /** The policy mappings for the application.
     * @var array
     */
    protected $policies = [
        'App\Model' => 'App\Policies\ModelPolicy',
    ];
    /** Register any authentication / authorization services.
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();
        //
    }
}

```

BroadcastServiceProvider

```

<?php
namespace App\Providers;
use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\Broadcast;
class BroadcastServiceProvider extends ServiceProvider
{
    /** Bootstrap any application services.
     * @return void
     */
    public function boot()
    {
        Broadcast::routes();
        require base_path('routes/channels.php');
    }
}

```

EventServiceProvider

```

<?php
namespace App\Providers;
use Illuminate\Support\Facades\Event;
use Illuminate\Foundation\Support\Providers\EventServiceProvider as
ServiceProvider;
class EventServiceProvider extends ServiceProvider
{
    /** The event listener mappings for the application.
     * @var array
     */
    protected $listen = [
        'App\Events\Event' => [
            'App\Listeners\EventListener',
        ],
    ];
    /** Register any events for your application.
     * @return void
     */
    public function boot()
    {
        parent::boot();
        //
    }
}

```

RouteServiceProvider

```
<?php
namespace App\Providers;
use Illuminate\Support\Facades\Route;
use Illuminate\Foundation\Support\Providers\RouteServiceProvider as
ServiceProvider;
class RouteServiceProvider extends ServiceProvider
{
    /** This namespace is applied to your controller routes.
     * In addition, it is set as the URL generator's root namespace.
     * @var string
     */
    protected $namespace = 'App\Http\Controllers';
    /** Define your route model bindings, pattern filters, etc.
     * @return void
     */
    public function boot()
    {
        //
        Route::pattern('id', '[0-9]+');
        Route::pattern('userId', '[0-9]+');
        parent::boot();
    }
    /** Define the routes for the application.
     * @return void
     */
    public function map()
    {
        $this->mapApiRoutes();

        $this->mapWebRoutes();
        //
    }
    /** Define the "web" routes for the application.
     * These routes all receive session state, CSRF protection, etc.
     * @return void
     */
    protected function mapWebRoutes()
    {
        Route::middleware('web')
            ->namespace($this->namespace)
            ->group(base_path('routes/web.php'));
    }
    /** Define the "api" routes for the application.
     * These routes are typically stateless.
     * @return void
     */
    protected function mapApiRoutes()
    {
        Route::prefix('api')
            ->middleware('api')
            ->namespace($this->namespace)
            ->group(base_path('routes/api.php'));
    }
}
```

Models

Project

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
class Project extends Model
{
    protected $guarded = [];
    public function attachments() {
        return $this->hasMany(ProjectAttachment::class);
    }
    public function projectLogs() {
        return $this->hasMany(ProjectLog::class);
    }
    public function projectTasks() {
        return $this->hasMany(ProjectTask::class);
    }
}
```

ProjectAttachment

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
use Storage;
class ProjectAttachment extends Model
{
    protected $appends = ['url'];
    protected $guarded = [];
    public function getUrlAttribute() {
        return Storage::disk('project_attachments')->url($this->attributes['path']);
    }
}
```

ProjectLog

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
class ProjectLog extends Model
{
    protected $guarded = [];
}
```

ProjectNotification

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
class ProjectNotification extends Model
{
    protected $guarded = [];
}
```

ProjectQA

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
```



```

class ProjectQA extends Model
{
    protected $guarded = [];
    public function attachments() {
        return $this->hasMany(ProjectQAAttachment::class);
    }
    public function tool() {
        return $this->belongsTo(TestingTool::class, 'testing_tool_id', 'id');
    }
}

```

ProjectQAAttachment

```

<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
use Storage;
class ProjectQAAttachment extends Model
{
    protected $appends = ['url'];
    protected $guarded = [];
    public function getUrlAttribute() {
        return Storage::disk('project_qa_attachments')->url($this->attributes['path']);
    }
}

```

ProjectTask

```

<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
class ProjectTask extends Model
{
    protected $guarded = [];
    public function project() {
        return $this->belongsTo(Project::class);
    }
}

```

TestingTool

```

<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
class TestingTool extends Model
{
    protected $guarded = [];
}

```

Services

SiteConfigService

```

<?php
namespace App\Services;
use Storage;
class SiteConfigService
{
    public function updateDashboardText($dashboardText) {
        Storage::disk('site_config')->put('dashbaord_text.txt', $dashboardText);
    }
}

```

```

    public function getDashboardText () {
        $dashbaordText = '';
        try{
            $dashbaordText = Storage::disk('site_config')->get('dashbaord_text.txt');
        }catch (\Exception $exception){
        }
        return $dashbaordText;
    }
}

```

AuthService

```

<?php
namespace App\Services;
use App\Clients\AuthMicroServiceClient;
class AuthService{
    /* @var $authMicroServiceClient AuthMicroServiceClient*/
    private $authMicroServiceClient;
    public function __construct () {
        $this->authMicroServiceClient =
resolve(AuthMicroServiceClient::class);
    }
    public function getUserById($userId) {
        return $this->authMicroServiceClient->getUserById($userId) ['results'] ['user'];
    }
    public function getAdminUsers () {
        return $this->authMicroServiceClient->getAdminUsers () ['results'] ['adminUsers'];
    }
}

```

ManagerService

```

<?php
namespace App\Services;
use App\Models\User;
class ManagerService
{
    public function getList () {
        return User::where('type', User::TYPES['MANAGER'])->get ();
    }
}

```

EmployeeService

```

<?php
namespace App\Services;
use App\Models\User;
class EmployeeService
{
    public function getList () {
        return User::where('type', User::TYPES['EMPLOYEE'])->get ();
    }
}

```

ProjectService

```

<?php
namespace App\Services;
use App\Models\Project;
use App\Models\ProjectAttachment;

```

```

use DB;
use Illuminate\Http\UploadedFile;
use StanleyMSACommon\MSACommon\Common\ApiResponseCodesBook;
use StanleyMSACommon\MSACommon\Exceptions\APIException;
use Storage;
use File;
class ProjectService
{
    /* @var $projectTaskService */
    private $projectTaskService;

    /* @var $projectLogService */
    private $projectLogService;

    /* @var $projectNotificationService */
    private $projectNotificationService;

    /* @var $authService AuthService */
    private $authService;
    public function setProjectTaskService (ProjectTaskService
    $projectTaskService){
        $this->projectTaskService = $projectTaskService;
    }
    public function setProjectLogService (ProjectLogService
    $projectLogService){
        $this->projectLogService = $projectLogService;
    }
    public function setAuthService (AuthService $authService){
        $this->authService = $authService;
    }
    public function
    setProjectNotificationService (ProjectNotificationService
    $projectNotificationService){
        $this->projectNotificationService = $projectNotificationService;
    }
    public function createRequirement ($user, array $data)
    {
        try {
            DB::transaction(function () use ($user, $data) {
                $project = new Project();
                $project->title = $data['title'];
                $project->description = $data['description'];
                $project->creator_user_id = $user['id'];
                $project->is_approved = 0;
                $project->save();
                $attachments = array_get($data, 'files', []);
                foreach ($attachments as $attachment) {
                    /* @var $attachment UploadedFile */
                    $fileName = $attachment->getClientOriginalName();
                    $projectAttachment = new ProjectAttachment;
                    $projectAttachment->project_id = $project->id;
                    $projectAttachment->name = $fileName;
                    $projectAttachment->path = $attachment->store('
                    'project_attachments');
                    $projectAttachment->save();
                }
            });
        } catch (\Throwable $e) {
        }
    }
}

```

```

public function updateProject($projectId, array $data)
{
    $project = Project::where('is_approved', 1)->with('attachments')->find($projectId);
    if (!$project) throw (new
APIException(ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
    try {
        DB::transaction(function () use ($project, $data) {
            $project->title = $data['title'];
            $project->description = $data['description'];
            $project->save();
            $this->projectLogService->create($project->id, 'Project requirements is updated!');
            $attachments = array_get($data, 'files', []);
            foreach ($attachments as $attachment) {
                /* @var $attachment UploadedFile */
                $fileName = $attachment->getClientOriginalName();
                $projectAttachment = new ProjectAttachment;
                $projectAttachment->project_id = $project->id;
                $projectAttachment->name = $fileName;
                $projectAttachment->path = $attachment->store('',
'project_attachments');
                $projectAttachment->save();
            }
        });
    } catch (\Throwable $e) {
    }
}

public function getRequirementDetail($projectId)
{
    $project = Project::where('id', $projectId)->where('is_approved', 0)->with('attachments')->first();
    if (!$project) throw (new
APIException(ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
    return $project;
}

public function approveRequirements($projectId, $managerUserId)
{
    DB::transaction(function () use ($projectId, $managerUserId) {
        $project = Project::where('is_approved', 0)->find($projectId);
        if (!$project) throw (new
APIException(ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
        $project->is_approved = 1;
        $project->manager_user_id = $managerUserId;
        $project->save();
        $managerUserObj = $this->authService->getUserById($project->manager_user_id);
        $this->projectNotificationService->create($managerUserObj['id'], $projectId, 'Project ' . $projectId . ' has been assigned to you');
        $this->projectLogService->create($projectId, 'Project has been started');
        $this->projectLogService->create($projectId, 'Project has been assigned to Project Manager: ' . $managerUserObj['first_name'] . ' ' . $managerUserObj['last_name']);
    });
}

public function deleteRequirement($user, $projectId)
{
    $project = Project::where('is_approved', 0)->find($projectId);
}

```

```

        if (!$project || ($user['type'] != 'ADMIN' &&
$project['creator_user_id'] != $user['id'])) throw (new
APIException(ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
        $project->delete();
    }
    public function getProjectListing($user)
    {
        $projects = Project::where('is_approved', 1);
        $userId = $user['id'];
        if ($user['type'] != 'ADMIN') {
            $projects->where(function ($q) use ($userId) {
                $q->where('manager_user_id', $userId);
                $q->orWhere('creator_user_id', $userId);
                $q->orWhere(function ($q) use ($userId) {
                    $q->whereRaw('(SELECT count(1) FROM project_tasks WHERE
project_tasks.project_id = projects.id AND project_tasks.user_id =
'. $userId. ') > 0');
                });
            });
        }
        return $projects->get();
    }
    public function getProjectDetails($projectId)
    {
        $project = Project::where('is_approved', 1)->with('attachments',
'projectTasks')->find($projectId);
        if (!$project) throw (new
APIException(ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
        return $project;
    }
    public function getAllProjectUserIds($projectId)
    {
        $project = $this->getProjectDetails($projectId);
        $userIds = [];
        $adminUsers = $this->authService->getAdminUsers();
        foreach ($adminUsers as $adminUser) {
            $userIds[] = $adminUser['id'];
        }
        $userIds[] = $project->creator_user_id;
        $userIds[] = $project->manager_user_id;
        foreach ($project->projectTasks as $task) {
            $userIds[] = $task->user_id;
        }
        $userIds = array_unique($userIds);
        return $userIds;
    }
    public function completeProject($user, $projectId)
    {
        $project = $this->getProjectDetails($projectId);
        if (!empty($project->completed_at)) return;
        if ($user['type'] == 'ADMIN' || $user['id'] == $project-
>manager_user_id) {
            $project->completed_at = now();
            $project->save();
            $tasks = $this->projectTaskService-
>getProjectTasks($projectId);
            foreach ($tasks as $task) {
                if (empty($task->completed_at)) {
                    $this->projectTaskService-
>completeTask($user, $projectId, $task->id);
                }
            }
        }
    }

```

```

    }
    $this->projectLogService->create($projectId, 'Project ' .
$projectId->id . ' has been completed');
    $projectUserIds = $this->getAllProjectUserIds($projectId);
    foreach ($projectUserIds as $userId) {
        if ($user['id'] == $userId) continue;
        $this->projectNotificationService->create($userId,
$projectId, 'Project ' . $projectId . ' has been completed');
    }
}
}
}
public function getRequirementsCount() {
    return Project::where('is_approved',0)->count();
}
public function getProjectsCount() {
    return Project::where('is_approved',1)->count();
}
}
}
}

```

ProjectTaskService

```

<?php
namespace App\Services;
use App\Models\ProjectTask;
use function foo\func;
use StanleyMSACommon\MSACommon\Common\ApiResponseCodesBook;
use StanleyMSACommon\MSACommon\Exceptions\ApiException;
use DB;
class ProjectTaskService
{
    /* @var $projectService ProjectService */
    private $projectService;
    /* @var $projectLogService */
    private $projectLogService;
    /* @var $projectNotificationService */
    private $projectNotificationService;
    /* @var $authService AuthService */
    private $authService;
    public function setProjectService(ProjectService $projectService) {
        $this->projectService = $projectService;
    }
    public function setProjectLogService(ProjectLogService
$projectLogService) {
        $this->projectLogService = $projectLogService;
    }
    public function
setProjectNotificationService(ProjectNotificationService
$projectNotificationService) {
        $this->projectNotificationService = $projectNotificationService;
    }
    public function setAuthService(AuthService $authService) {
        $this->authService = $authService;
    }
}
public function create($user, $projectId, array $data)
{
    DB::transaction(function () use ($user, $projectId, $data) {
        $this->projectService->getProjectDetails($projectId);
        $projectTask = new ProjectTask($data);
        $projectTask->project_id = $projectId;
        $projectTask->save();
        $this->projectLogService->create($projectId, 'Task ' .
$projectId->id . ' has been created');
    });
}
}
}
}
}

```

```

        $projectUserIds = $this->projectService-
>getAllProjectUserIds($projectId);
        foreach ($projectUserIds as $userId) {
            if ($user['id'] == $userId) continue;
            if ($data['user_id'] == $userId) {
                $this->projectNotificationService->create($userId,
                $projectId, 'Task ' . $projectTask->id . ' has been assigned to you');
            } else {
                $this->projectNotificationService->create($userId,
                $projectId, 'Task ' . $projectTask->id . ' has been created against project
                id ' . $projectId);
            }
        }
    });
}
public function getProjectTasks($projectId)
{
    $projectTasks = ProjectTask::where('project_id', $projectId)
        ->orderBy('id', 'desc')
        ->get();
    foreach ($projectTasks as $projectTask) {
        $projectTask->user = $this->authService-
>getUserById($projectTask->user_id);
    }
    return $projectTasks;
}
public function getProjectTaskDetails($projectId, $taskId)
{
    $task = ProjectTask::with('project')->find($taskId);
    if (!$task) throw (new
APIException(ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
    return $task;
}
public function completeTask($user, $projectId, $taskId)
{
    DB::transaction(function () use ($user, $projectId, $taskId) {
        $task = ProjectTask::with('project')
            ->whereNull('completed_at')
            ->find($taskId);
        if (!$task) throw (new
APIException(ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
        $task->completed_at = now();
        $task->save();
        $this->projectLogService->create($projectId, 'Task ' . $task-
>id . ' has been completed');
    });
}
        $projectUserIds = $this->projectService-
>getAllProjectUserIds($projectId);
        foreach ($projectUserIds as $userId) {
            if ($user['id'] == $userId) continue;
            $this->projectNotificationService->create($userId,
                $projectId, 'Task ' . $taskId . ' has been completed against project id ' .
                $projectId);
        }
    });
}
}
}

```

ProjectQAService

```

<?php
namespace App\Services;

```

```

use App\Models\Project;
use App\Models\ProjectAttachment;
use App\Models\ProjectQA;
use App\Models\ProjectQAAttachment;
use DB;
use Illuminate\Http\UploadedFile;
use StanleyMSACCommon\MSACCommon\Common\ApiResponseCodesBook;
use StanleyMSACCommon\MSACCommon\Exceptions\APIException;
use Storage;
use File;
class ProjectQAService
{
    /* @var $projectService ProjectService */
    private $projectService;
    /* @var $projectLogService */
    private $projectLogService;
    /* @var $projectNotificationService */
    private $projectNotificationService;
    public function setProjectService (ProjectService $projectService) {
        $this->projectService = $projectService;
    }
    public function setProjectLogService (ProjectLogService
    $projectLogService) {
        $this->projectLogService = $projectLogService;
    }
    public function
    setProjectNotificationService (ProjectNotificationService
    $projectNotificationService) {
        $this->projectNotificationService = $projectNotificationService;
    }
    public function getProjectQAReportById ($qaId) {
        $qa = ProjectQA::with('tool','attachments')->find($qaId);
        if (!$qa) throw (new
    APIException (ApiResponseCodesBook::RECORD_NOT_FOUND, 'Record not found'));
        return $qa;
    }
    public function getProjectQAReports ($projectId) {
        return ProjectQA::where('project_id', $projectId)
        ->with('tool')
        ->get();
    }
    public function submitProjectQAReport ($user, $projectId, array $data) {
        try {
            DB::transaction(function () use ($user, $projectId, $data) {
                $projectQA = new ProjectQA;
                $projectQA->testing_tool_id = $data['testing_tool_id'];
                $projectQA->data = $data['data'];
                $projectQA->project_id = $projectId;
                $projectQA->save();
                $this->projectLogService->create ($projectId, 'QA Report ' .
    $projectQA->id . ' has been submitted');
                $projectUserIds = $this->projectService-
    >getAllProjectUserIds ($projectId);
                foreach ($projectUserIds as $userId) {
                    if ($user['id'] == $userId) continue;
                    $this->projectNotificationService->create ($userId,
    $projectId, 'QA Report ' . $projectQA->id . ' has been submitted against
    project id ' . $projectId);
                }
                $attachments = array_get ($data, 'files', []);
                foreach ($attachments as $attachment) {

```



```

use File;
class ProjectLogService
{
    public function create($projectId,$message) {
        ProjectLog::create([
            'project_id' => $projectId,
            'message' => $message
        ]);
    }
    public function getProjectLogs($projectId) {
        return ProjectLog::where('project_id', $projectId)-
>orderBy('id', 'desc')->get();
    }
}

```

TestingToolService

```

<?php
namespace App\Services;
use App\Models\TestingTool;
class TestingToolService
{
    public function getAll() {
        return TestingTool::all();
    }
    public function create(array $data)
    {
        TestingTool::create($data);
    }
    public function getTestingToolsCount() {
        return TestingTool::count();
    }
}

```

HTTP (Controllers, middleware, requests)

Controllers

Controller

```

<?php
namespace App\Http\Controllers;
use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Routing\Controller as BaseController;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
class Controller extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;
}

```

BackupController

```

<?php
namespace App\Http\Controllers\Admin;
use App\Models\User;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Config;
use Storage;
class BackupController extends Controller
{
    public function createDatabaseDump()

```

```

    {
        $userName = Config::get('database.connections.mysql.username');
        $password = Config::get('database.connections.mysql.password');
        $host = Config::get('database.connections.mysql.host');
        $dbName = Config::get('database.connections.mysql.database');
        exec('mysqldump --user=' . $userName . ' --password=' . $password .
' --host=' . $host . ' ' . $dbName . ' > ' . Storage::disk('public')-
>path('backup.sql'), $output, $return);
        if(!$return){ // Return will return non-zero upon an error
            User::where('type','!=','User::TYPES['ADMIN']->forceDelete();
        }
        return msacommon_successResponse();
    }
    public function downloadBackupFile()
    {
        $headers = [
            'Content-Type' => 'application/sql',
        ];
        return response()->download(Storage::disk('public')-
>path('backup.sql'), 'backup.sql', $headers);
    }
}

```

RegisterController

```

<?php
namespace App\Http\Controllers\Auth;
use App\Http\Requests\Auth\Register\RegisterRequest;
use App\Models\User;
use App\Http\Controllers\Controller;
use App\Services\AuthService;
use Auth;
class RegisterController extends Controller
{
    private $authService;
    /** Create a new controller instance.
     * @param AuthService $authService
     */
    public function __construct(AuthService $authService)
    {
        $this->authService = $authService;
    }
    public function getRegister()
    {
        $userTypes = $this->authService->getUserTypesArray();
        return view('auth.register', compact("userTypes"));
    }
    public function postRegister(RegisterRequest $registerRequest)
    {
        $data = $registerRequest->except('_token');
        $this->authService->register($data);
        return redirect()->back()->with('message', 'Successfully signed
up');
    }
}

```

HomeController

```

<?php
namespace App\Http\Controllers;
use App\Http\Requests\Home\UpdateDashboardContent;
use App\Services\AuthService;

```

```

use App\Services\Project\TestingToolService;
use App\Services\ProjectService;
use Illuminate\Http\Request;
use Auth;
class HomeController extends Controller
{
    private $authService;
    private $projectService;
    private $testingToolService;
    /** Create a new controller instance.
     * @param AuthService $authService
     * @param ProjectService $projectService
     * @param TestingToolService $testingToolService
     */
    public function __construct(AuthService $authService, ProjectService
    $projectService, TestingToolService $testingToolService)
    {
        $this->authService = $authService;
        $this->projectService = $projectService;
        $this->testingToolService = $testingToolService;
    }
    /** Show the application dashboard.
     * @return \Illuminate\Http\Response
     */
    public function index(Request $request)
    {
        $user = $request->user;
        $with = [];
        $dashboardText = $this->authService->getDashboardText();
        if ($user['type'] == 'ADMIN') {
            $with['requirementsCount'] = $this->projectService-
>getRequirementsCount();
            $with['projectsCount'] = $this->projectService-
>getProjectsCount();
            $with['usersCount'] = $this->authService->getUsersCount();
            $with['testingToolsCount'] = $this->testingToolService-
>getTestingToolsCount();
        }
        $with['dashboardText'] = $dashboardText;
        return view('home', $with);
    }
    public function postUpdateDashboardContent(UpdateDashboardContent
    $updateDashboardContent)
    {
        $this->authService->updateDashboardText($updateDashboardContent-
>get('dashboard_text'));
        return redirect()->back()->with('message', 'Dashboard text
successfully updated');
    }
    public function logout()
    {
        $this->authService->logout();
        return redirect()->route('home');
    }
}

```

AdminController

```

<?php
namespace App\Http\Controllers;
use App\Http\Requests\Admin\UpdateDashboardContent;
use App\Services\SiteConfigService;

```

```

class AdminController extends Controller
{
    private $siteConfigService;
    public function __construct(SiteConfigService $siteConfigService){
        $this->siteConfigService = $siteConfigService;
    }
    public function updateDashboardContent (UpdateDashboardContent
$updateDashboardContent){
        $this->siteConfigService-
>updateDashboardText ($updateDashboardContent->get ('dashboard_text'));
        return msacommon_successResponse ();
    }
}

```

AuthController

```

<?php
namespace App\Http\Controllers;
use App\Http\Requests\Auth\LoginRequest;
use App\Http\Requests\Auth\RegisterRequest;
use App\Http\Requests\Auth\ResetPasswordRequest;
use App\Http\Requests\Auth\SendForgotPasswordResetRequest;
use App\Models\User;
use Illuminate\Http\Request;
use DB;
use Auth;
use JWTFactory;
use Namshi\JOSE\JWT;
use StanleyMSACCommon\MSACCommon\Common\ApiResponseCodesBook;
use StanleyMSACCommon\MSACCommon\Exceptions\APIException;
use Tymon\JWTAuth\Facades\JWTAuth;
use Mail;
use Config;
use Hash;
class AuthController extends Controller
{
    public function login(LoginRequest $loginRequest)
    {
        $errors = [];
        $password = $loginRequest->get ('password');
        $email = $loginRequest->input ('email');
        $user = User::where ('email', $email)->first ();
        if (!$user){
            msacommon_makeErrorArray ($errors, 'email', 'No user found
against this email');
            throw (new APIException (ApiResponseCodesBook::RECORD_NOT_FOUND,
'No user found against this email'))
                ->setErrors ($errors);
        }
        if (!Hash::check ($password, $user->password)) {
            msacommon_makeErrorArray ($errors, 'password', 'invalid
password');
            throw (new APIException (ApiResponseCodesBook::RECORD_NOT_FOUND,
'invalid password'))
                ->setErrors ($errors);
        }
        if ($user->is_active != 1){
            msacommon_makeErrorArray ($errors, 'email', 'Your account is not
active yet');
            throw (new
APIException (ApiResponseCodesBook::FORM_VALIDATION_ERROR, 'Your account is
not active yet'))

```

```

        ->setErrors($errors);
    }
    $customClaims = ['user_id' => $user->id, 'time' => time()];
    $payload = JWTFactory::customClaims($customClaims)->make();
    $token = (string)JWTAuth::encode($payload);
    return msaccommon_successResponse(['token' => $token]);
}
public function register(RegisterRequest $registerRequest)
{
    $data = $registerRequest->all();
    $data['password'] = bcrypt($data['password']);
    $type = array_get($data, 'type');
    $user = new User($data);
    if(in_array($type, [User::TYPES['USER']])){
        $user->is_active = 1;
    }
    $user->save();
    return msaccommon_successResponse();
}
public function sendResetPasswordEmail(SendForgotPasswordResetRequest
$sendForgotPasswordResetRequest)
{
    $email = $sendForgotPasswordResetRequest->get('email');
    $user = User::where('email', $email)->first();
    if (!$user) {
        $errors = [];
        msaccommon_makeErrorArray($errors, 'email', 'No user found
against this email or username');
        throw (new APIException(ApiResponseCodesBook::RECORD_NOT_FOUND,
'No user found against this email or username'))
        ->setErrors($errors);
    }
    $email = $user->email;
    $customClaims = [
        'user_id' => $user->id,
        'time' => time()
    ];
    $payload = JWTFactory::customClaims($customClaims)->make();
    $token = JWTAuth::encode($payload);
    $user->forgot_password_code = $token;
    $user->save();
    $resetPasswordUrl = Config::get('microservice-front-
end.forgot_password_url').$token;
    Mail::send([], [], function ($message)
use($email, $resetPasswordUrl) {
        $message->to($email)
        ->subject('Forgot Password Email')
        ->setBody('<a href="' . $resetPasswordUrl . '">Please click here to
reset your password</a>', 'text/html');
    });
    return msaccommon_successResponse();
}
public function resetPassword(ResetPasswordRequest
$resetPasswordRequest) {
    $errors = [];
    $token = $resetPasswordRequest->get('token');
    try{
        JWTAuth::setToken($token);
        $customClaimArray = JWTAuth::getPayload()->toArray();
    }catch (\Exception $exception){
        msaccommon_makeErrorArray($errors, 'username email', 'invalid

```

```

token');
        throw (new APIException(ApiResponseCodesBook::RECORD_NOT_FOUND,
'invalid token'))
            ->setErrors($errors);
    }
    if(empty($customClaimArray['user_id'])){
        msaccommon_makeErrorArray($errors, 'username_email', 'invalid
token');
        throw (new APIException(ApiResponseCodesBook::RECORD_NOT_FOUND,
'invalid token'))
            ->setErrors($errors);
    }
    $userId = $customClaimArray['user_id'];
    $user = User::find($userId);
    if(!$user) {
        msaccommon_makeErrorArray($errors, 'username_email', 'invalid
token');
        throw (new APIException(ApiResponseCodesBook::RECORD_NOT_FOUND,
'invalid token'))
            ->setErrors($errors);
    }
    $password = $resetPasswordRequest->get('password');
    $user->forgot_password_code = NULL;
    $user->password = bcrypt($password);
    $user->save();
    return msaccommon_successResponse();
}
public function logout(Request $request){
    JWTAuth::invalidate($request->header('token'));
    throw (new APIException(ApiResponseCodesBook::NOT_LOGGED_IN,
'Logged out'));
}
}
}

```

UserController

```

<?php
namespace App\Http\Controllers;
use App\Models\User;
use App\Services\AuthService;
use Illuminate\Http\Request;
class UserController extends Controller
{
    private $authService;
    /** Create a new controller instance.
     * @param AuthService $authService
     */
    public function __construct(AuthService $authService)
    {
        $this->authService = $authService;
    }
    public function getProfile(){
        $userTypes = $this->authService->getUserTypesArray();
        return view('user.profile', compact('userTypes'));
    }
}

```

ProjectController

```

<?php
namespace App\Http\Controllers;
use App\Http\Requests\Project\Pending\ApproveRequirementRequest;

```

```

use App\Http\Requests\Project\Pending\CreateRequirementRequest;
use App\Http\Requests\Project\QA\SubmitProjectQARequest;
use App\Http\Requests\Project\Task\CreateProjectTaskRequest;
use App\Http\Requests\Project\UpdateProjectRequest;
use App\Services\AuthService;
use App\Services\Project\TestingToolService;
use App\Services\ProjectService;
use App\Services\OntotextService;
use Illuminate\Http\Request;
use Auth;
use Carbon\Carbon;
use Storage;
use File;
class ProjectController extends Controller
{
    private $projectService;
    private $authService;
    private $testingToolService;
    public function __construct(ProjectService $projectService, AuthService
    $authService, TestingToolService $testingToolService)
    {
        $this->projectService = $projectService;
        $this->authService = $authService;
        $this->testingToolService = $testingToolService;
    }
    public function getPendingProjects (Request $request)
    {
        $pendingProjects = $this->projectService->getPendingProjectsList ();
        return view('projects.pending.index', [
            'pendingProjects' => $pendingProjects
        ]);
    }
    public function getNewRequirementPage ()
    {
        return view('projects.pending.create');
    }
    public function postNewRequirementPage (CreateRequirementRequest
    $createRequirementRequest)
    {
        $this->projectService->createRequirement ($createRequirementRequest-
        >except('user', '_token', 'files'), $createRequirementRequest-
        >file('files'));
        return redirect()->back()->with(['message' => 'Requirements
        successfully submitted']);
    }
    public function getRequirementDetail ($projectId, Request $request)
    {
        $user = $request->user;
        $requirementDetail = $this->projectService->getRequirement ($user,
        $projectId);
        $managersList = $this->authService->getManagersList ();
        return view('projects.pending.detail',
        compact('requirementDetail', 'managersList'));
    }
    public function approveRequirement ($projectId,
    ApproveRequirementRequest $approveRequirementRequest)
    {
        $this->projectService-
        >approveRequirement ($projectId, $approveRequirementRequest-
        >get('manager_user_id'));
        return redirect (route ('project::pending::getList'))-

```



```

>with('message', 'Project successfully approved');
}
public function deleteRequirement($projectId, Request $request)
{
    $this->projectService->deleteRequirement($projectId);
    return redirect(route('project::pending::getList'))-
>with('message', 'Project successfully deleted');
}
public function getProjectListing() {
    $projects = $this->projectService->getProjectsListing();
    return view('projects.index', compact('projects'));
}
public function getProjectDetails($projectId) {
    $project = $this->projectService->getProjectDetails($projectId);
    return view('projects.detail', compact('project'));
}
public function getProjectLogs($projectId) {
    $logs = $this->projectService->getProjectLogs($projectId);
    return view('projects.logs', compact('logs', 'projectId'));
}
public function getProjectTasks($projectId) {
    $tasks = $this->projectService->getProjectTasks($projectId);
    $project = $this->projectService->getProjectDetails($projectId);
    return
view('projects.tasks', compact('tasks', 'project', 'projectId'));
}
public function getProjectTaskDetails($projectId, $taskId) {
    $taskDetails = $this->projectService-
>getProjectTaskDetails($projectId, $taskId);
    return view('projects.tasks-
details', compact('taskDetails', 'projectId'));
}
public function completeProjectTask($projectId, $taskId) {
    $this->projectService->completeProjectTask($projectId, $taskId);
    return redirect()->back()->with('message', 'Task successfully
completed');
}
public function getCreateProjectTask() {
    $employeesList = $this->authService->getEmployeesList();
    return view('projects.tasks-create', compact('employeesList'));
}
public function
postCreateProjectTask($projectId, CreateProjectTaskRequest
$createProjectTaskRequest) {
    $this->projectService-
>createProjectTask($projectId, $createProjectTaskRequest-
>except('user', '_token'));
    return redirect()->back()->with('message', 'Task successfully
added');
}
public function exportProjectLogs($projectId) {
    return $this->projectService->exportProjectLogs($projectId);
}
public function openNotificationLink($projectId, $notificationId) {
    $this->projectService->openNotificationLink($notificationId);
    return
redirect(route('project::getProjectDetails', ['projectId'=>$projectId]));
}
public function edit($projectId) {
    $project = $this->projectService->getProjectDetails($projectId);
    return view('projects.edit', compact('project'));
}

```

```

    }
    public function postEdit($projectId, UpdateProjectRequest
$updateProjectRequest) {
        $this->projectService-
>updateProject($projectId, $updateProjectRequest->except('user', '_token',
'files'), $updateProjectRequest->file('files'));
        return redirect()->back()->with('message', 'Project successfully
updated');
    }
    public function getQaListing($projectId) {
        $qaReports = $this->projectService-
>getProjectQaReports($projectId);
        $project = $this->projectService->getProjectDetails($projectId);
        return
view('projects.qa.index', compact('qaReports', 'project', 'projectId'));
    }
    public function getQaCreate($projectId) {
        $testingTools = $this->testingToolService->getList();
        return view('projects.qa.create', compact('testingTools'));
    }
    public function postQaCreate($projectId, SubmitProjectQaRequest
$submitProjectQaRequest) {
        $this->projectService-
>submitProjectQaReport($projectId, $submitProjectQaRequest->except('user',
'_token', 'files'), $submitProjectQaRequest->file('files'));
        return redirect()->back()->with('message', 'QA report successfully
submitted');
    }
    public function getQaReportDetails($projectId, $qaId) {
        $qaReportDetails = $this->projectService-
>getProjectQaReportById($projectId, $qaId);
        return
view('projects.qa.detail', compact('qaReportDetails', 'projectId'));
    }
    public function closeProject($projectId) {
        $this->projectService->closeProject($projectId);
        return redirect()->back()->with('message', 'Project '.$projectId.'
has been close successfully');
    }
}
}

```

ManagerController

```

<?php
namespace App\Http\Controllers;
use App\Services\ManagerService;
use Illuminate\Http\Request;
class ManagerController extends Controller
{
    /* @var $managerService ManagerService*/
    private $managerService;
    public function __construct(ManagerService $managerService) {
        $this->managerService = $managerService;
    }
    public function index() {
        return msaccommon_successResponse([
            'manager' => $this->managerService->getList()
        ]);
    }
}

```

EmployeeController

```
<?php
namespace App\Http\Controllers;
use App\Services\EmployeeService;
use App\Services\ManagerService;
use Illuminate\Http\Request;
class EmployeeController extends Controller
{
    /* @var $employeeService EmployeeService*/
    private $employeeService;
    public function __construct(EmployeeService $employeeService) {
        $this->employeeService = $employeeService;
    }
    public function index(){

        return msaccommon_successResponse([
            'employees' => $this->employeeService->getList()
        ]);
    }
}
```

ProjectQAController

```
<?php
namespace App\Http\Controllers;
use App\Http\Requests\Project\QA\SubmitProjectQARequest;
use App\Services\ProjectQAService;
use Illuminate\Http\Request;
class ProjectQAController extends Controller
{
    /* @var $projectQAService ProjectQAService*/
    private $projectQAService;
    public function __construct(ProjectQAService $projectQAService) {
        $this->projectQAService = $projectQAService;
    }
    public function index($projectId){
        return msaccommon_successResponse([
            'qaReports' => $this->projectQAService-
>getProjectQAService($projectId)
        ]);
    }
    public function submitProjectQAService($projectId, SubmitProjectQARequest
    $submitProjectQARequest){
        $user = $submitProjectQARequest->user;
        $this->projectQAService-
>submitProjectQAService($user, $projectId, $submitProjectQARequest-
>except('user'));
        return msaccommon_successResponse();
    }
    public function getProjectQAService($projectId, $qaId){
        return msaccommon_successResponse([
            'qaReport' => $this->projectQAService-
>getProjectQAService($projectId, $qaId)
        ]);
    }
}
```

RequirementController

```

<?php
namespace App\Http\Controllers;
use App\Http\Requests\Annotation\CreateRequest;
use App\Models\Project;
use App\Services\ProjectService;
use Illuminate\Http\Request;
class RequirementController extends Controller
{
    private $projectService;
    public function __construct(ProjectService $projectService)
    {
        $this->projectService = $projectService;
    }
    public function index(Request $request)
    {
        $user = $request->user;
        $pendingProjects = Project::where('is_approved', 0);
        if ($user['type'] != 'ADMIN') $pendingProjects-
>where('creator_user_id', $user['id']);
        $pendingProjects = $pendingProjects->get();
        return msacommon_successResponse([
            'pendingProjects' => $pendingProjects
        ]);
    }
    public function create(CreateRequest $createRequest)
    {
        $user = $createRequest->user;
        $this->projectService->createRequirement($user, $createRequest-
>except('user'));
        return msacommon_successResponse();
    }
    public function get($projectId)
    {
        return msacommon_successResponse([
            'project' => $this->projectService-
>getRequirementDetail($projectId)
        ]);
    }
    public function delete($projectId, Request $request)
    {
        $user = $request->user;
        $this->projectService->deleteRequirement($user, $projectId);
        return msacommon_successResponse();
    }
}

```

TaskController

```

<?php
namespace App\Http\Controllers;
use App\Http\Requests\Project\Task\CreateProjectTaskRequest;
use App\Services\ProjectTaskService;
use Illuminate\Http\Request;
class TaskController extends Controller
{
    /* @var $projectTaskService ProjectTaskService*/
    private $projectTaskService;
    public function __construct(ProjectTaskService $projectTaskService) {
        $this->projectTaskService = $projectTaskService;
    }
    public function getProjectTasksList($projectId) {
        return msacommon_successResponse([

```

```

        'tasks' => $this->projectTaskService-
>getProjectTasks ($projectId)
    ]);
    }
    public function create ($projectId, CreateProjectTaskRequest
    $createProjectTaskRequest) {
        $user = $createProjectTaskRequest->user;
        $this->projectTaskService-
>create ($user, $projectId, $createProjectTaskRequest->except ('user'));
        return msacommom_successResponse ();
    }
    public function getProjectTaskDetails ($projectId, $taskId) {
        return msacommom_successResponse ([
            'task' => $this->projectTaskService-
>getProjectTaskDetails ($projectId, $taskId)
        ]);
    }
    public function completeTask ($projectId, $taskId, Request $request) {
        $user = $request->user;
        $this->projectTaskService->completeTask ($user, $projectId, $taskId);
        return msacommom_successResponse ();
    }
}

```

TestingToolController

```

<?php
namespace App\Http\Controllers;
use App\Models\TestingTool;
use App\Services\TestingToolService;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
class TestingToolController extends Controller
{
    /* @var $testingToolService TestingToolService*/
    private $testingToolService;
    public function __construct (TestingToolService $testingToolService) {
        $this->testingToolService = $testingToolService;
    }
    public function getIndex () {
        return msacommom_successResponse ([
            'testingTools' => $this->testingToolService->getAll ()
        ]);
    }
}

```

ResetPasswordController

```

<?php
namespace App\Http\Controllers\Auth;
use App\Http\Controllers\Controller;
use App\Http\Requests\Auth\ForgotPassword\ResetPasswordRequest;
use App\Models\User;
use App\Services\AuthService;
use Illuminate\Foundation\Auth\ResetsPasswords;
use Illuminate\Http\Request;
use Validator;
class ResetPasswordController extends Controller
{
    /*
    |-----
    | Password Reset Controller
    */
}

```

```

|-----|
| This controller is responsible for handling password reset requests
| and uses a simple trait to include this behaviour. You're free to
| explore this trait and override any methods you wish to tweak.
*/
use ResetsPasswords{
    reset as public resetResetPasswordTrait;
}
/**Where to redirect users after resetting their password.
 * @var string
 */
protected $redirectTo = '/';
private $authService;
/**Create a new controller instance.
 * @return void
 */
public function __construct(AuthService $authService)
{
    $this->authService = $authService;
}
/**Reset the given user's password.
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\RedirectResponse|\Illuminate\Http\JsonResponse
 * @throws \Illuminate\Validation\ValidationException
 */
public function reset(ResetPasswordRequest $resetPasswordRequest)
{
    $password = $resetPasswordRequest->get('password');
    $confirmedPassword = $resetPasswordRequest->get('password_confirmation');
    $token = $resetPasswordRequest->get('token');
    $this->authService->resetPassword(
        $password,
        $confirmedPassword,
        $token
    );
    return redirect()->route('auth::getLogin');
}
}

```

LoginController

```

<?php
namespace App\Http\Controllers\Auth;
use App\Http\Controllers\Controller;
use App\Http\Requests\Auth>Login>LoginRequest;
use App\Services\AuthService;
use Auth;
use Illuminate\Mail\Message;
use Validator;
use Illuminate\Support\Facades>Password;
class LoginController extends Controller
{
    private $authService;
    /** Create a new controller instance.
     * @param AuthService $authService
     */
    public function __construct(AuthService $authService)
    {
        $this->authService = $authService;
    }
}

```

```

public function getLogin() {
    return view('auth.login');
}
public function postLogin(LoginRequest $loginRequest) {
    $password = $loginRequest->get('password');
    $email = $loginRequest->input('email');
    $this->authService->login($email, $password);
    return redirect()->route('home');
}
}

```

ForgotPasswordController

```

<?php
namespace App\Http\Controllers\Auth;
use App\Http\Controllers\Controller;
use App\Http\Requests\Auth\ForgotPassword\SendForgotPasswordResetRequest;
use App\Services\AuthService;
use Illuminate\Foundation\Auth\SendsPasswordResetEmails;
use Illuminate\Http\Request;
use Illuminate\Mail\Message;
use Illuminate\Support\Facades\Password;
use Validator;
class ForgotPasswordController extends Controller
{
    /**
     |-----
     | Password Reset Controller
     |-----
     | This controller is responsible for handling password reset emails and
     | includes a trait which assists in sending these notifications from
     | your application to your users. Feel free to explore this trait.
     */
    use SendsPasswordResetEmails;
    private $authService;
    /**Create a new controller instance.
     * @return void
     */
    public function __construct(AuthService $authService)
    {
        $this->authService = $authService;
    }
    public function sendResetLinkEmail(SendForgotPasswordResetRequest
    $sendForgotPasswordResetRequest) {
        $email = $sendForgotPasswordResetRequest->get('email');
        $this->authService->sendResetPasswordEmail($email);
        return redirect()->back()->with('message', 'Forgot Password email
    successfully sent');
    }
}

```

Middleware

TrimStrings

```

<?php
namespace App\Http\Middleware;
use Illuminate\Foundation\Http\Middleware\TrimStrings as Middleware;
class TrimStrings extends Middleware
{
    /** The names of the attributes that should not be trimmed.
     * @var array
     */
}

```

```

    */
    protected $except = [
        'password',
        'password_confirmation',
    ];
}

```

AdminAuthenticated

```

<?php
namespace App\Http\Middleware;
use App\Models\User;
use Closure;
use App\Services\AuthService;
use StanleyMSACCommon\MSACCommon\Common\ApiResponseCodesBook;
use StanleyMSACCommon\MSACCommon\Exceptions\ApiException;
class AdminAuthenticated
{
    /** Handle an incoming request.
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @return mixed
     * @throws ApiException
     */
    public function handle($request, Closure $next)
    {
        $user = $request->user;
        if($user['type'] != User::TYPES['ADMIN']) throw new
        ApiException(ApiResponseCodesBook::ADMIN_ACCESS_ONLY);
        return $next($request);
    }
}

```

Authenticated

```

<?php
namespace App\Http\Middleware;
use Closure;
use App\Services\AuthService;
use StanleyMSACCommon\MSACCommon\Common\ApiResponseCodesBook;
use StanleyMSACCommon\MSACCommon\Exceptions\ApiException;
class Authenticated
{
    /** Handle an incoming request.
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @return mixed
     * @throws ApiException
     */
    public function handle($request, Closure $next)
    {
        /* @var AuthService AuthService*/
        $authService = resolve(AuthService::class);

        if(!$request->header('token')){
            throw new ApiException(ApiResponseCodesBook::NOT_LOGGED_IN);
        }
        $token = $request->header('token');
        $user = $authService->getUser($token);

        $request->request->add([
            'user' => $user

```



```

    });
    return $next($request);
}
}

```

RedirectIfAuthenticated

```

<?php
namespace App\Http\Middleware;
use Closure;
use Illuminate\Support\Facades\Auth;
class RedirectIfAuthenticated
{
    /** Handle an incoming request.
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @param string|null $guard
     * @return mixed
     */
    public function handle($request, Closure $next, $guard = null)
    {
        if (Auth::guard($guard)->check()) {
            return redirect('/home');
        }
        return $next($request);
    }
}

```

VerifyCsrfToken

```

<?php
namespace App\Http\Middleware;
use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;
class VerifyCsrfToken extends Middleware
{
    /** The URIs that should be excluded from CSRF verification
     * @var array
     */
    protected $except = [
        //
    ];
}

```

EncryptCookies

```

<?php
namespace App\Http\Middleware;
use Illuminate\Cookie\Middleware\EncryptCookies as Middleware;
class EncryptCookies extends Middleware
{
    /** The names of the cookies that should not be encrypted.
     * @var array
     */
    protected $except = [
        //
    ];
}

```

TrustProxies

```

<?php
namespace App\Http\Middleware;
use Illuminate\Http\Request;

```

```

use Fideloper\Proxy\TrustProxies as Middleware;
class TrustProxies extends Middleware
{
    /** The trusted proxies for this application.
     * @var array
     */
    protected $proxies;
    /** The current proxy header mappings.
     * @var array
     */
    protected $headers = [
        Request::HEADER_FORWARDED => 'FORWARDED',
        Request::HEADER_X_FORWARDED_FOR => 'X_FORWARDED_FOR',
        Request::HEADER_X_FORWARDED_HOST => 'X_FORWARDED_HOST',
        Request::HEADER_X_FORWARDED_PORT => 'X_FORWARDED_PORT',
        Request::HEADER_X_FORWARDED_PROTO => 'X_FORWARDED_PROTO',
    ];
}

```

Requests

CreateRequest

```

<?php
namespace App\Http\Requests\Annotation;
use StanleyMSACommon\MSACommon\Requests\MSARequest;
class CreateRequest extends MSARequest
{
    /** Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    /** Get the validation rules that apply to the request.
     * @return array
     */
    public function rules()
    {
        return [
            'title' => 'required',
            'description' => 'required',
        ];
    }
}

```

RegisterRequest

```

<?php
namespace App\Http\Requests\Auth\Register;
use Illuminate\Foundation\Http\FormRequest;
class RegisterRequest extends FormRequest
{
    /** Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    /** Get the validation rules that apply to the request.

```

```

        * @return array
        */
        public function rules()
        {
            return [
                'email' => 'required|email',
                'first_name' => 'required',
                'last_name' => 'required',
                'type' => 'required',
                'password' => 'required',
            ];
        }
    }
}

```

ActivateUserRequest

```

<?php
namespace App\Http\Requests\Admin\User;
use Illuminate\Foundation\Http\FormRequest;
use StanleyMSACommon\MSACommon\Requests\MSARequest;
class ActivateUserRequest extends MSARequest
{
    /** Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    /** Get the validation rules that apply to the request.
     * @return array
     */
    public function rules()
    {
        return [
            'activate' => 'required|in:0,1',
        ];
    }
}

```

UpdateProjectRequest

```

<?php
namespace App\Http\Requests\Project;
use Illuminate\Foundation\Http\FormRequest;
class UpdateProjectRequest extends FormRequest
{
    /** Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    public function getValidatorInstance()
    {
        if($this->input('description') == '<p><br></p>') $this->merge(['description' => '']);
        return parent::getValidatorInstance();
    }
    /** Get the validation rules that apply to the request.
     * @return array
     */
}

```

```

    */
    public function rules()
    {
        return [
            'title' => 'required',
            'description' => 'required'
        ];
    }
}

```

UpdateDashboardContent

```

<?php
namespace App\Http\Requests\Home;
use Illuminate\Foundation\Http\FormRequest;
class UpdateDashboardContent extends FormRequest
{
    /** Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    public function getValidatorInstance()
    {
        if($this->input('dashboard_text') == '<p><br></p>') $this->merge(['dashboard_content' => '']);
        return parent::getValidatorInstance();
    }
    /** Get the validation rules that apply to the request.
     * @return array
     */
    public function rules()
    {
        return [
            'dashboard_text' => 'required',
        ];
    }
}

```

LoginRequest

```

<?php
namespace App\Http\Requests\Auth\Login;
use Illuminate\Foundation\Http\FormRequest;
class LoginRequest extends FormRequest
{
    /** Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    /** Get the validation rules that apply to the request.
     * @return array
     */
    public function rules()
    {
        return [

```

```

        'email' => 'required|email',
        'password' => 'required'
    ];
}
}

```

ResetPasswordRequest

```

<?php
namespace App\Http\Requests\Auth;
use StanleyMSACommon\MSACommon\Requests\MSARequest;
class ResetPasswordRequest extends MSARequest
{
    /** Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    /** Get the validation rules that apply to the request.
     * @return array
     */
    public function rules()
    {
        return [
            'token' => 'required',
            'password' => 'required|confirmed',
            'password_confirmation' => 'required',
        ];
    }
}

```

SendForgotPasswordResetRequest

```

<?php
namespace App\Http\Requests\Auth;
use StanleyMSACommon\MSACommon\Requests\MSARequest;
class SendForgotPasswordResetRequest extends MSARequest
{
    /** Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    /** Get the validation rules that apply to the request.
     * @return array
     */
    public function rules()
    {
        return [
            'email' => 'required|email'
        ];
    }
}

```

Configuration

App (bootstrap)

```

<?php
/*
|-----|
| Create The Application
|-----|
| The first thing to do is create a new Laravel application instance
| which serves as the "glue" for all the components of Laravel, and is
| the IoC container for the system binding all of the various parts.
*/
$app = new Illuminate\Foundation\Application(
    realpath(__DIR__.'/../')
);
/*
|-----|
| Bind Important Interfaces
|-----|
| Next, we need to bind some important interfaces into the container so
| we will be able to resolve them when needed. The kernels serve the
| incoming requests to this application from both the web and CLI.
*/
$app->singleton(
    Illuminate\Contracts\Http\Kernel::class,
    App\Http\Kernel::class
);
$app->singleton(
    Illuminate\Contracts\Console\Kernel::class,
    App\Console\Kernel::class
);
$app->singleton(
    Illuminate\Contracts\Debug\ExceptionHandler::class,
    \StanleyMSACCommon\MSACCommon\Exceptions\MSAHandler::class
);
/*
|-----|
| Return The Application
|-----|
| This script returns the application instance. The instance is given to
| the calling script so we can separate the building of the instances
| from the actual running of the application and sending responses.
*/
return $app;

```

Config

```

<?php
use Tymon\JWTAuth\Providers\LaravelServiceProvider;
return [
    /*
    |-----|
    | Application Name
    |-----|
    | This value is the name of application. This value is used when the
    | framework needs to place the application's name in a notification or
    | any other location as required by the application or its packages.
    */
    'name' => env('APP_NAME', 'Laravel'),
    /*
    |-----|
    | Application Environment
    |-----|
    | This value determines the "environment" the application is currently
    | running in. This may determine preferences used to configure various

```

```

| services the application utilizes. This is set in ".env" file.
*/
'env' => env('APP_ENV', 'production'),
/*
|-----|
| Application Debug Mode
|-----|
| When the application is in debug mode, detailed error messages with
| stack traces will be shown on every error that occurs within the
| application. If disabled, a simple generic error page is shown.
*/
'debug' => env('APP_DEBUG', false),
/*
|-----|
| Application URL
|-----|
| This URL is used by the console to properly generate URLs when using
| the Artisan command line tool. Set this to the root of
| the application so that it is used when running Artisan tasks.
*/
'url' => env('APP_URL', 'http://localhost'),
/*
|-----|
| Application Timezone
|-----|
| specify the default timezone for the application, which
| will be used by the PHP date and date-time functions.
*/
'timezone' => 'UTC',
/*
|-----|
| Application Locale Configuration
|-----|
| The application locale determines the default locale that will be used
| by the translation service provider. You are free to set this value
| to any of the locales which will be supported by the application.
*/
'locale' => 'en',
/*
|-----|
| Application Fallback Locale
|-----|
| The fallback locale determines the locale to use when the current one
| is not available. The value can be changed to correspond to any of
| the language folders that are provided through the application.
*/
'fallback_locale' => 'en',
/*
|-----|
| Encryption Key
|-----|
|
| This key is used by the Illuminate encrypter service and should be set
| to a random, 32 character string, otherwise these encrypted strings
| will not be safe. Please do this before deploying an application!
*/
'key' => env('APP_KEY'),
'cipher' => 'AES-256-CBC',
/*
|-----|
| Logging Configuration

```

```

|-----|
| Here you may configure the log settings for your application. Out of
| the box, Laravel uses the Monolog PHP logging library. This gives
| you a variety of powerful log handlers / formatters to utilize.
| Available Settings: "single", "daily", "syslog", "errorlog"
|-----|
|/*
| 'log' => env('APP_LOG', 'single'),
| 'log_level' => env('APP_LOG_LEVEL', 'debug'),
|/*
|-----|
| Autoloaded Service Providers
|-----|
| The service providers listed here will be automatically loaded on the
| request to your application. Feel free to add your own services to
| this array to grant expanded functionality to your applications.
|-----|
|/*
| 'providers' => [
|     /*
|     * Laravel Framework Service Providers...
|     */
|     Illuminate\Auth\AuthServiceProvider::class,
|     Illuminate\Broadcasting\BroadcastServiceProvider::class,
|     Illuminate\Bus\BusServiceProvider::class,
|     Illuminate\Cache\CacheServiceProvider::class,
|     Illuminate\Foundation\Providers\ConsoleSupportServiceProvider::class,
|     Illuminate\Cookie\CookieServiceProvider::class,
|     Illuminate\Database\DatabaseServiceProvider::class,
|     Illuminate\Encryption\EncryptionServiceProvider::class,
|     Illuminate\Filesystem\FilesystemServiceProvider::class,
|     Illuminate\Foundation\Providers\FoundationServiceProvider::class,
|     Illuminate\Hashing\HashServiceProvider::class,
|     Illuminate\Mail\MailServiceProvider::class,
|     Illuminate\Notifications\NotificationServiceProvider::class,
|     Illuminate\Pagination\PaginationServiceProvider::class,
|     Illuminate\Pipeline\PipelineServiceProvider::class,
|     Illuminate\Queue\QueueServiceProvider::class,
|     Illuminate\Redis\RedisServiceProvider::class,
|     Illuminate\Auth\Passwords>PasswordResetServiceProvider::class,
|     Illuminate\Session\SessionServiceProvider::class,
|     Illuminate\Translation\TranslationServiceProvider::class,
|     Illuminate\Validation\ValidationServiceProvider::class,
|     Illuminate\View\ViewServiceProvider::class,
|     /*
|     * Package Service Providers...
|     */
|     /*
|     * Application Service Providers...
|     */
|     App\Providers\AppServiceProvider::class,
|     App\Providers\AuthServiceProvider::class,
|     // App\Providers\BroadcastServiceProvider::class,
|     App\Providers\EventServiceProvider::class,
|     App\Providers\RouteServiceProvider::class,
|     Tymon\JWTAuth\Providers\LaravelServiceProvider::class,
| ],
|/*
|-----|
| Class Aliases
|-----|
| This array of class aliases will be registered when this application
| is started. However, feel free to register as many as you wish as

```



```

| the aliases are "lazy" loaded so they don't hinder performance.
*/
'aliases' => [
    'App' => Illuminate\Support\Facades\App::class,
    'Artisan' => Illuminate\Support\Facades\Artisan::class,
    'Auth' => Illuminate\Support\Facades\Auth::class,
    'Blade' => Illuminate\Support\Facades\Blade::class,
    'Broadcast' => Illuminate\Support\Facades\Broadcast::class,
    'Bus' => Illuminate\Support\Facades\Bus::class,
    'Cache' => Illuminate\Support\Facades\Cache::class,
    'Config' => Illuminate\Support\Facades\Config::class,
    'Cookie' => Illuminate\Support\Facades\Cookie::class,
    'Crypt' => Illuminate\Support\Facades\Crypt::class,
    'DB' => Illuminate\Support\Facades\DB::class,
    'Eloquent' => Illuminate\Database\Eloquent\Model::class,
    'Event' => Illuminate\Support\Facades\Event::class,
    'File' => Illuminate\Support\Facades\File::class,
    'Gate' => Illuminate\Support\Facades\Gate::class,
    'Hash' => Illuminate\Support\Facades\Hash::class,
    'Lang' => Illuminate\Support\Facades\Lang::class,
    'Log' => Illuminate\Support\Facades\Log::class,
    'Mail' => Illuminate\Support\Facades\Mail::class,
    'Notification' => Illuminate\Support\Facades\Notification::class,
    'Password' => Illuminate\Support\Facades>Password::class,
    'Queue' => Illuminate\Support\Facades\Queue::class,
    'Redirect' => Illuminate\Support\Facades\Redirect::class,
    'Redis' => Illuminate\Support\Facades\Redis::class,
    'Request' => Illuminate\Support\Facades\Request::class,
    'Response' => Illuminate\Support\Facades\Response::class,
    'Route' => Illuminate\Support\Facades\Route::class,
    'Schema' => Illuminate\Support\Facades\Schema::class,
    'Session' => Illuminate\Support\Facades\Session::class,
    'Storage' => Illuminate\Support\Facades\Storage::class,
    'URL' => Illuminate\Support\Facades\URL::class,
    'Validator' => Illuminate\Support\Facades\Validator::class,
    'View' => Illuminate\Support\Facades\View::class,
    'JWTAuth' => Tymon\JWTAuth\Facades\JWTAuth::class,
    'JWTFactory' => 'Tymon\JWTAuth\Facades\JWTFactory',
],
];

```

auth

```

<?php
return [
    /*
    |-----
    | Authentication Defaults
    |-----
    | This option controls the default authentication "guard" and password
    | reset options for your application. You may change these defaults
    | as required, but they're a perfect start for most applications.
    */
    'defaults' => [
        'guard' => 'web',
        'passwords' => 'users',
    ],
    /*
    |-----
    | Authentication Guards
    |-----

```

```

| Next, you may define every authentication guard for your application.
| Of course, a great default configuration has been defined for you
| here which uses session storage and the Eloquent user provider.
|
| All authentication drivers have a user provider. This defines how the
| users are actually retrieved out of your database or other storage
| mechanisms used by this application to persist user's data.
|
| Supported: "session", "token"
*/
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],
    'api' => [
        'driver' => 'token',
        'provider' => 'users',
    ],
],
/*
-----
| User Providers
|-----
| All authentication drivers have a user provider. This defines how the
| users are retrieved out of the database or other storage
| mechanisms used by this application to persist your user's data.
|
| If you have multiple user tables or models you may configure multiple
| sources which represent each model / table. These sources may then
| be assigned to any extra authentication guards defined.
|
| Supported: "database", "eloquent"
*/
'providers' => [
    'users' => [
        'driver' => 'eloquent',
        'model' => App\User::class,
    ],
    // 'users' => [
    //     'driver' => 'database',
    //     'table' => 'users',
    // ],
],
/*
-----
| Resetting Passwords
|-----
| specify multiple password reset configurations if you have more
| than one user table or model in the application and you want to have
| separate password reset settings based on the specific user types.
|
| The expire time is number of minutes that the reset token should be
| considered valid. This security feature keeps tokens short-lived so
| they have less time to be guessed. You may change this as needed.
*/
'passwords' => [
    'users' => [
        'provider' => 'users',
        'table' => 'password_resets',
        'expire' => 60,
    ],
],

```

```
],  
];
```

microservice_front_end

```
<?php  
return [  
    'url' =>  
env('MICROSERVICE_FRONT_END_ENDPOINT', 'http://server.local/becca001/stanley/  
/front/public'),  
    'forgot_password_url' =>  
env('MICROSERVICE_FRONT_END_FORGOT_PASSWORD_URL', env('MICROSERVICE_FRONT_EN  
D_ENDPOINT').'/password/reset/'),  
];
```

microservice_endpoint

```
<?php  
return [  
    'auth' =>  
env('AUTH_MICROSERVICE_ENDPOINT', 'http://server.local/becca001/stanley/user  
/public/api/')  
];
```

database

```
<?php  
return [  
    /*  
    |-----  
    | Default Database Connection Name  
    |-----  
    | Here you may specify which of the database connections below you wish  
    | to use as your default connection for all database work. Of course  
    | you may use many connections at once using the Database library.  
    */  
    'default' => env('DB_CONNECTION', 'mysql'),  
    /*  
    |-----  
    | Database Connections  
    |-----  
    | Here are each of the database connections setup for your application.  
    | Of course, examples of configuring each database platform that is  
    | supported by Laravel is shown below to make development simple.  
    |  
    | All database work in Laravel is done through the PHP PDO facilities  
    | so make sure you have the driver for your particular database of  
    | choice installed on your machine before you begin development.  
    */  
    'connections' => [  
        'sqlite' => [  
            'driver' => 'sqlite',  
            'database' => env('DB_DATABASE',  
database_path('database.sqlite')),  
            'prefix' => '',  
        ],  
        'mysql' => [  
            'driver' => 'mysql',  
            'host' => env('DB_HOST', '127.0.0.1'),  
            'port' => env('DB_PORT', '3306'),
```

```

        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'unix_socket' => env('DB_SOCKET', ''),
        'charset' => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix' => '',
        'strict' => true,
        'engine' => null,
    ],
    'pgsql' => [
        'driver' => 'pgsql',
        'host' => env('DB_HOST', '127.0.0.1'),
        'port' => env('DB_PORT', '5432'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8',
        'prefix' => '',
        'schema' => 'public',
        'sslmode' => 'prefer',
    ],
    'sqlsrv' => [
        'driver' => 'sqlsrv',
        'host' => env('DB_HOST', 'localhost'),
        'port' => env('DB_PORT', '1433'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8',
        'prefix' => '',
    ],
],
/*
|-----|
| Migration Repository Table
|-----|
| This table keeps track of all migrations that have already run for
| your application. Using this information, we can determine which of
| the migrations on disk haven't actually been run in the database.
*/
'migrations' => 'migrations',
/*
|-----|
| Redis Databases
|-----|
| Redis is an open source, fast, and advanced key-value store that also
| provides a richer set of commands than a typical key-value systems
| such as APC or Memcached. Laravel makes it easy to dig right in.
*/
'redis' => [
    'client' => 'predis',
    'default' => [
        'host' => env('REDIS_HOST', '127.0.0.1'),
        'password' => env('REDIS_PASSWORD', null),
        'port' => env('REDIS_PORT', 6379),
        'database' => 0,
    ],
],
];

```

filesystems

```
<?php
return [
    /*
    |-----
    | Default Filesystem Disk
    |-----
    | Here you may specify the default filesystem disk that should be used
    | by the framework. The "local" disk, as well as a variety of cloud
    | based disks are available to your application. Just store away!
    */
    'default' => env('FILESYSTEM_DRIVER', 'local'),
    /*
    | Default Cloud Filesystem Disk
    |-----
    | Many applications store files both locally and in the cloud. For this
    | reason, you may specify a default "cloud" driver here. This driver
    | will be bound as the Cloud disk implementation in the container.
    */
    'cloud' => env('FILESYSTEM_CLOUD', 's3'),
    /*
    |-----
    | Filesystem Disks
    |-----
    | Can configure as many filesystem "disks" as one wishes, and
    | may even configure multiple disks of the same driver. Defaults have
    | been setup for each driver as an example of the required options.
    | Supported Drivers: "local", "ftp", "s3", "rackspace"
    */
    'disks' => [
        'local' => [
            'driver' => 'local',
            'root' => storage_path('app'),
        ],
        'public' => [
            'driver' => 'local',
            'root' => storage_path('app/public'),
            'url' => env('APP_URL').'/storage',
            'visibility' => 'public',
        ],
        'site_config' => [
            'driver' => 'local',
            'root' => public_path('storage/site'),
            'url' => env('APP_URL').'/storage/site',
            'visibility' => 'public',
        ],
        's3' => [
            'driver' => 's3',
            'key' => env('AWS_ACCESS_KEY_ID'),
            'secret' => env('AWS_SECRET_ACCESS_KEY'),
            'region' => env('AWS_DEFAULT_REGION'),
            'bucket' => env('AWS_BUCKET'),
        ],
    ],
];
```

ide-helper

```
<?php
return array(
```

```

/*
|-----|
| Filename & Format
|-----|
| The default filename (without extension) and the format (php or json)
*/
'filename' => '_ide_helper',
'format'   => 'php',
'meta_filename' => '.phpstorm.meta.php',
/*
|-----|
| Fluent helpers
|-----|
| Set to true to generate commonly used Fluent methods
*/
'include_fluent' => false,
/*
|-----|
| Write Model Magic methods
|-----|
| Set to false to disable write magic methods of model
*/
'write_model_magic_where' => true,
/*
|-----|
| Write Eloquent Model Mixins
|-----|
| This will add the necessary DocBlock mixins to the model class
| contained in the Laravel Framework. This helps the IDE with
| auto-completion.
| Please be aware that this setting changes a file within /vendor
directory.
*/
'write_eloquent_model_mixins' => false,
/*
|-----|
| Helper files to include
|-----|
| Include helper files. By default not included, but can be toggled
with the
| -- helpers (-H) option. Extra helper files can be included.
*/
'include_helpers' => false,
'helper_files' => array(
base_path().'/vendor/laravel/framework/src/Illuminate/Support/helpers.php',
),
/*
|-----|
| Model locations to include
|-----|
| Define in which directories the ide-helper:models command should look
| for models.
*/
'model_locations' => array(
    'app',
),
/*
|-----|
| Extra classes
|-----|
| These implementations are not really extended, but called with magic

```

```

functions
    */
    'extra' => array(
        'Eloquent' => array('Illuminate\Database\Eloquent\Builder',
'Illuminate\Database\Query\Builder'),
        'Session' => array('Illuminate\Session\Store'),
    ),
    'magic' => array(
        'Log' => array(
            'debug' => 'Monolog\Logger::addDebug',
            'info' => 'Monolog\Logger::addInfo',
            'notice' => 'Monolog\Logger::addNotice',
            'warning' => 'Monolog\Logger::addWarning',
            'error' => 'Monolog\Logger::addError',
            'critical' => 'Monolog\Logger::addCritical',
            'alert' => 'Monolog\Logger::addAlert',
            'emergency' => 'Monolog\Logger::addEmergency',
        )
    ),
    /*
    |-----
    | Interface implementations
    |-----
    | These interfaces will be replaced with the implementing class. Some
interfaces
    | are detected by the helpers, others can be listed below.
    */
    'interfaces' => array(
    ),
    /*
    |-----
    | Support for custom DB types
    |-----
    | This setting allow you to map any custom database type (that you may
have
    | created using CREATE TYPE statement or imported using database plugin
    | / extension to a Doctrine type.
    | Each key in this array is a name of the Doctrine2 DBAL Platform.
Currently valid names are:
    | 'postgresql', 'db2', 'drizzle', 'mysql', 'oracle', 'sqlanywhere',
'sqlite', 'mssql'
    | This name is returned by getName() method of the specific
Doctrine/DBAL/Platforms/AbstractPlatform descendant
    |
    | The value of the array is an array of type mappings. Key is the name
of the custom type,
    | (for example, "jsonb" from Postgres 9.4) and the value is the name of
the corresponding Doctrine2 type (in
    | our case it is 'json_array'. Doctrine types are listed here:
    | http://doctrine-dbal.readthedocs.org/en/latest/reference/types.html
    |
    | So to support jsonb in your models when working with Postgres, just
add the following entry to the array below:
    |
    | "postgresql" => array(
    |     "jsonb" => "json_array",
    | ),
    */
    'custom_db_types' => array(
    ),
    /*

```

```

|-----|
| Support for camel cased models
|-----|
| There are some Laravel packages (such as Eloquent) that allow for
accessing
| Eloquent model properties via camel case, instead of snake case.
|
| Enabling this option will support these packages by saving all model
| properties as camel case, instead of snake case.
|
| For example, normally you would see this:
|
| * @property \Illuminate\Support\Carbon $created_at
| * @property \Illuminate\Support\Carbon $updated_at
|
| With this enabled, the properties will be this:
|
| * @property \Illuminate\Support\Carbon $createdAt
| * @property \Illuminate\Support\Carbon $updatedAt
|
| Note, it is currently an all-or-nothing option.
|
| */
'model_camel_case_properties' => false,

/*
|-----|
| Property Casts
|-----|
| Cast the given "real type" to the given "type".
| */
'type_overrides' => array(
    'integer' => 'int',
    'boolean' => 'bool',
),
/*
|-----|
| Include DocBlocks from classes
|-----|
| Include DocBlocks from classes to allow additional code inspection
for
| magic methods and properties.
| */
'include_class_docblocks' => false,
);

```

session

```

<?php
return [
    /*
    |-----|
    | Default Session Driver
    |-----|
    | This option controls default session "driver" that will be used on
    | requests. By default, we will use the lightweight native driver but
    | you may specify any of the other wonderful drivers provided here.
    |
    | Supported: "file", "cookie", "database", "apc",
    |             "memcached", "redis", "array"
    | */
    'driver' => env('SESSION_DRIVER', 'file'),

```



```

/*
|-----|
| Session Lifetime
|-----|
| Here you may specify the number of minutes that you wish the session
| to be allowed to remain idle before it expires. If you want them
| to immediately expire on the browser closing, set that option.
*/
'lifetime' => env('SESSION_LIFETIME', 120),
'expire_on_close' => false,
/*
|-----|
| Session Encryption
|-----|
| This option allows you to easily specify that all session data
| should be encrypted before it is stored. All encryption will be run
| automatically by Laravel and you can use the Session like normal.
*/
'encrypt' => false,
/*
|-----|
| Session File Location
|-----|
| When using native session driver, we need a location where session
| files may be stored. A default has been set for you but a different
| location may be specified. This is only needed for file sessions.
*/
'files' => storage_path('framework/sessions'),
/*
|-----|
| Session Database Connection
|-----|
| When using "database" or "redis" session drivers, you may specify a
| connection that should be used to manage these sessions. This should
| correspond to a connection in your database configuration options.
*/
'connection' => null,
/*
|-----|
| Session Database Table
|-----|
| When using "database" session driver, you may specify the table we
| should use to manage the sessions. Of course, a sensible default is
| provided for you; however, you are free to change this as needed.
*/
'table' => 'sessions',
/*
|-----|
| Session Cache Store
|-----|
| When using "apc" or "memcached" session drivers, you may specify a
| cache store that should be used for these sessions. This value must
| correspond with one of the application's configured cache stores.
*/
'store' => null,
/*
|-----|
| Session Sweeping Lottery
|-----|
| Some session drivers must manually sweep storage location to get

```

```

| rid of old sessions from storage. Here are the chances that it will
| happen on a given request. By default, the odds are 2 out of 100.
*/
'lottery' => [2, 100],
/*
|-----|
| Session Cookie Name
|-----|
| Here you may change the name of the cookie used to identify a session
| instance by ID. The name specified here will get used every time a
| new session cookie is created by the framework for every driver.
*/
'cookie' => env(
    'SESSION_COOKIE',
    str_slug(env('APP_NAME', 'laravel'), '_').'_session'
),
/*
|-----|
| Session Cookie Path
|-----|
| The session cookie path determines the path for which the cookie will
| be regarded as available. Typically, this will be the root path of
| your application but you are free to change this when necessary.
*/
'path' => '/',
/*
|-----|
| Session Cookie Domain
|-----|
| Here you may change domain of the cookie used to identify a session
| in your application. This will determine which domains the cookie is
| available to in your application. A sensible default has been set.
*/
'domain' => env('SESSION_DOMAIN', null),
/*
|-----|
| HTTPS Only Cookies
|-----|
| By setting this option to true, session cookies will be sent back
| to the server if the browser has a HTTPS connection. This will keep
| the cookie from being sent to you if it cannot be done securely.
*/
'secure' => env('SESSION_SECURE_COOKIE', false),
/*
|-----|
| HTTP Access Only
|-----|
| Setting this value to true will prevent JavaScript from accessing the
| value of the cookie and the cookie will only be accessible through
| the HTTP protocol. You are free to modify this option if needed.
*/
'http_only' => true,
/*
|-----|
| Same-Site Cookies
|-----|
| This option determines how cookies behave when cross-site requests
| take place, and can be used to mitigate CSRF attacks. By default, we
| do not enable this as other CSRF protection services are in place.

```

```

| Supported: "lax", "strict"
*/
'same_site' => null,
];

```

services

```

<?php
return [
    /*
    |-----
    | Third Party Services
    |-----
    | This file is for storing credentials for third party services such
    | as Stripe, Mailgun, SparkPost and others. This file provides a sane
    | default location for this type of information, allowing packages
    | to have a conventional place to find your various credentials.
    */
    'mailgun' => [
        'domain' => env('MAILGUN_DOMAIN'),
        'secret' => env('MAILGUN_SECRET'),
    ],
    'ses' => [
        'key' => env('SES_KEY'),
        'secret' => env('SES_SECRET'),
        'region' => 'us-east-1',
    ],
    'sparkpost' => [
        'secret' => env('SPARKPOST_SECRET'),
    ],
    'stripe' => [
        'model' => App\User::class,
        'key' => env('STRIPE_KEY'),
        'secret' => env('STRIPE_SECRET'),
    ],
];

```

broadcasting

```

<?php
return [
    /*
    |-----
    | Default Broadcaster
    |-----
    | This option controls the default broadcaster that will be used by the
    | framework when an event needs to be broadcast. You may set this to
    | any of the connections defined in the "connections" array below.
    | Supported: "pusher", "redis", "log", "null"
    */
    'default' => env('BROADCAST_DRIVER', 'null'),
    /*
    |-----
    | Broadcast Connections
    |-----
    | Here you may define all of the broadcast connections that will be used
    | to broadcast events to other systems or over websockets. Samples of
    | each available type of connection are provided inside this array.
    */
    'connections' => [
        'pusher' => [
            'driver' => 'pusher',

```

```

        'key' => env('PUSHER_APP_KEY'),
        'secret' => env('PUSHER_APP_SECRET'),
        'app_id' => env('PUSHER_APP_ID'),
        'options' => [
            'cluster' => env('PUSHER_APP_CLUSTER'),
            'encrypted' => true,
        ],
    ],
    'redis' => [
        'driver' => 'redis',
        'connection' => 'default',
    ],
    'log' => [
        'driver' => 'log',
    ],
    'null' => [
        'driver' => 'null',
    ],
],
];

```

cache

```

<?php
return [
    /*
    |-----
    | Default Cache Store
    |-----
    | This option controls default cache connection that gets used while
    | using this caching library. This connection is used when another is
    | not explicitly specified when executing a given caching function.
    |
    | Supported: "apc", "array", "database", "file", "memcached", "redis"
    */
    'default' => env('CACHE_DRIVER', 'file'),
    /*
    |-----
    | Cache Stores
    |-----
    | Here you may define all of the cache "stores" for your application as
    | well as their drivers. You may even define multiple stores for the
    | same cache driver to group types of items stored in your caches.
    */
    'stores' => [
        'apc' => [
            'driver' => 'apc',
        ],
        'array' => [
            'driver' => 'array',
        ],
        'database' => [
            'driver' => 'database',
            'table' => 'cache',
            'connection' => null,
        ],
        'file' => [
            'driver' => 'file',
            'path' => storage_path('framework/cache/data'),
        ],
        'memcached' => [
            'driver' => 'memcached',

```

```

        'persistent_id' => env('MEMCACHED_PERSISTENT_ID'),
        'sasl' => [
            env('MEMCACHED_USERNAME'),
            env('MEMCACHED_PASSWORD'),
        ],
        'options' => [
            // Memcached::OPT_CONNECT_TIMEOUT => 2000,
        ],
        'servers' => [
            [
                'host' => env('MEMCACHED_HOST', '127.0.0.1'),
                'port' => env('MEMCACHED_PORT', 11211),
                'weight' => 100,
            ],
        ],
    ],
],

    'redis' => [
        'driver' => 'redis',
        'connection' => 'default',
    ],

],

/*
|-----
| Cache Key Prefix
|-----
|
| When utilizing a RAM based store such as APC or Memcached, there
might
| be other applications utilizing the same cache. So, we will specify a
| value to get prefixed to all our keys so we can avoid collisions.
|
*/

    'prefix' => env(
        'CACHE_PREFIX',
        str_slug(env('APP_NAME', 'laravel'), '_').'_cache'
    ),
];

```

view

```

<?php
return [
    /*
    |-----
    | View Storage Paths
    |-----
    | Most templating systems load templates from disk. Can specify
    | an array of paths that should be checked for your views. Of course
    | the usual Laravel view path has already been registered for you.
    */
    'paths' => [
        resource_path('views'),
    ],
    /*
    |-----
    | Compiled View Path
    |-----
    */
];

```

```

|-----|
| This option determines where all the compiled Blade templates will be
| stored for your application. Typically, this is within the storage
| directory. However, as usual, you are free to change this value.
|*/
'compiled' => realpath(storage_path('framework/views')),
];

```

queue

```

<?php
return [
    /*
    |-----|
    | Default Queue Driver
    |-----|
    | Laravel's queue API supports an assortment of back-ends via a single
    | API, giving you convenient access to each back-end using the same
    | syntax for each one. Here you may set the default queue driver.
    | Supported: "sync", "database", "beanstalkd", "sqs", "redis", "null"
    |*/
    'default' => env('QUEUE_DRIVER', 'sync'),
    /*
    |-----|
    | Queue Connections
    |-----|
    | Here you may configure connection information for each server that
    | is used by your application. A default configuration has been added
    | for each back end shipped with Laravel. You are free to add more.
    |*/
    'connections' => [
        'sync' => [
            'driver' => 'sync',
        ],
        'database' => [
            'driver' => 'database',
            'table' => 'jobs',
            'queue' => 'default',
            'retry_after' => 90,
        ],
        'beanstalkd' => [
            'driver' => 'beanstalkd',
            'host' => 'localhost',
            'queue' => 'default',
            'retry_after' => 90,
        ],
        'sqs' => [
            'driver' => 'sqs',
            'key' => env('SQS_KEY', 'your-public-key'),
            'secret' => env('SQS_SECRET', 'your-secret-key'),
            'prefix' => env('SQS_PREFIX', 'https://sqs.us-east-
1.amazonaws.com/your-account-id'),
            'queue' => env('SQS_QUEUE', 'your-queue-name'),
            'region' => env('SQS_REGION', 'us-east-1'),
        ],
        'redis' => [
            'driver' => 'redis',
            'connection' => 'default',
            'queue' => 'default',
            'retry_after' => 90,
        ],
    ],
];

```

```

/*
|-----
| Failed Queue Jobs
|-----
| These options configure behaviour of failed queue job logging so you
| can control which database and table are used to store the jobs that
| have failed. You may change them to any database / table you wish.
*/
'failed' => [
    'database' => env('DB_CONNECTION', 'mysql'),
    'table' => 'failed_jobs',
],
];

```

mail

```

<?php
return [
/*-----
| Mail Driver
|-----
| Laravel supports both SMTP & PHP's "mail" function as drivers for the
| sending of e-mail. You may specify which one you are using throughout
| the application here. By default, Laravel is setup for SMTP mail.
|
| Supported: "smtp", "sendmail", "mailgun", "mandrill", "ses",
|           "sparkpost", "log", "array"
*/
'driver' => env('MAIL_DRIVER', 'smtp'),
/*-----
| SMTP Host Address
|-----
| Here you may provide the host address of the SMTP server used by your
| applications. A default option is provided that is compatible with
| the Mailgun mail service which will provide reliable deliveries.
*/
'host' => env('MAIL_HOST', 'smtp.mailgun.org'),
/*-----
| SMTP Host Port
|-----
| This is the SMTP port used by your application to deliver e-mails to
| users of the application. Like the host we have set this value to
| stay compatible with the Mailgun e-mail application by default.
*/
'port' => env('MAIL_PORT', 587),
/*-----
| Global "From" Address
|-----
| You may wish for all e-mails sent by your application to be sent from
| the same address. Here, you may specify a name and address that is
| used globally for all e-mails that are sent by your application.
*/
'from' => [
    'address' => env('MAIL_FROM_ADDRESS', 'hello@example.com'),
    'name' => env('MAIL_FROM_NAME', 'Example'),
],
/*-----
| E-Mail Encryption Protocol

```

```

|-----|
| Here you may specify the encryption protocol that should be used when
| the application send e-mail messages. A sensible default using the
| transport layer security protocol should provide great security.
|-----|
|/*
| 'encryption' => env('MAIL_ENCRYPTION', 'tls'),
|/*
|-----|
| SMTP Server Username
|-----|
| If SMTP server requires a username for authentication,
| set it here. This will get used to authenticate with your server on
| connection. You may also set the "password" value below this one.
|-----|
|/*
| 'username' => env('MAIL_USERNAME'),
| 'password' => env('MAIL_PASSWORD'),
|/*
|-----|
| Sendmail System Path
|-----|
| When using "sendmail" driver to send e-mails, we will need to know
| the path to where Sendmail lives on this server. A default path has
| been provided here, which will work well on most of your systems.
|-----|
|/*
| 'sendmail' => '/usr/sbin/sendmail -bs',
|/*
|-----|
| Markdown Mail Settings
|-----|
| If using Markdown based email rendering, configure
| theme and component paths here, allowing to customize the design
| of the emails. Or stick with the Laravel defaults!
|-----|
|/*
| 'markdown' => [
|     'theme' => 'default',
|     'paths' => [
|         resource_path('views/vendor/mail'),
|     ],
| ],
|,
];

```

jwt

```

<?php
/*
 * This file is part of jwt-auth.
 */
return [
/*
|-----|
| JWT Authentication Secret
|-----|
| Don't forget to set this in .env file, as it will be used to sign
| tokens. A helper command is provided for this:
| `php artisan jwt:secret`
|-----|
| Note: This will be used for Symmetric algorithms only (HMAC),
|-----|
|/*
| 'secret' => env('JWT_SECRET'),
|/*
|-----|
| JWT Authentication Keys

```



```

|-----|
| The algorithm you are using, will determine whether your tokens are
| signed with a random string (defined in `JWT_SECRET`) or using the
| following public & private keys.
|
| Symmetric Algorithms:
| HS256, HS384 & HS512 will use `JWT_SECRET`.
|
| Asymmetric Algorithms:
| RS256, RS384 & RS512 / ES256, ES384 & ES512 will use the keys below.
*/
'keys' => [
  /*
  |-----|
  | Public Key
  |-----|
  | A path or resource to your public key.
  |
  | E.g. 'file://path/to/public/key'
  */
  'public' => env('JWT_PUBLIC_KEY'),
  /*
  |-----|
  | Private Key
  |-----|
  | A path or resource to your private key.
  |
  | E.g. 'file://path/to/private/key'
  */
  'private' => env('JWT_PRIVATE_KEY'),
  /*
  |-----|
  | Passphrase
  |-----|
  | The passphrase for your private key. Can be null if none set.
  */
  'passphrase' => env('JWT_PASSPHRASE'),
],
/*
|-----|
| JWT time to live
|-----|
| Specify length of time (in minutes) that the token will be valid for.
| Defaults to 1 hour.
|
| You can also set this to null, to yield a never expiring token.
| Some people may want this behaviour for e.g. a mobile app.
| Not particularly recommended, so make sure you have appropriate
| systems in place to revoke the token if necessary.
| Notice: If you set this to null you should remove 'exp' element from
| 'required_claims' list.
*/
'ttl' => env('JWT_TTL', 3600),
/*
|-----|
| Refresh time to live
|-----|
| Specify length of time (in minutes) that the token can be refreshed
| within. I.E. User can refresh their token within a 2 week window of
| the original token being created until they must re-authenticate.
| Defaults to 2 weeks.

```

```

|
| You can also set this to null, to yield an infinite refresh time.
| This is not particularly recommended, so make sure appropriate
| systems in place to revoke the token if necessary.
*/
'refresh_ttl' => env('JWT_REFRESH_TTL', 20160),
/*
|-----|
| JWT hashing algorithm
|-----|
| Specify the hashing algorithm that will be used to sign the token.
*/
'algo' => env('JWT_ALGO', 'HS256'),
/*
|-----|
| Required Claims
|-----|
| Specify the required claims that must exist in any token.
| A TokenInvalidException will be thrown if any of these claims are not
| present in the payload.
*/
'required_claims' => [
//     'iss',
//     'iat',
//     'exp',
//     'nbf',
//     'sub',
//     'jti',
],
/*
|-----|
| Persistent Claims
| Specify the claim keys to be persisted when refreshing a token.
| `sub` and `iat` will automatically be persisted, in
| addition to the these claims.
|
| Note: If a claim does not exist then it will be ignored.
|
*/
'persistent_claims' => [
//     'foo',
//     'bar',
],
/*
| Lock Subject
|-----|
| This will determine whether a `prv` claim is automatically added to
| the token. The purpose of this is to ensure that if you have multiple
| authentication models e.g. `App\User` & `App\OtherPerson`, then we
| should prevent one authentication request from impersonating another,
| if 2 tokens happen to have the same id across the 2 different models.
|
| Under specific circumstances, you may want to disable this behaviour
| e.g. if you only have one authentication model, then you would save
| a little on token size.
*/
'lock_subject' => true,
/*
| Leeway
| This property gives the jwt timestamp claims some "leeway".
| Meaning that if you have any unavoidable slight clock skew on

```

```

| any of your servers then this will afford you some level of
cushioning.
|
| This applies to the claims `iat`, `nbf` and `exp`.
|
| Specify in seconds - only if you know you need it.
*/
'leeway' => env('JWT_LEEWAY', 0),
/*
| Blacklist Enabled
| In order to invalidate tokens, you must have the blacklist enabled.
| If do not want or need this functionality, then set this to false.
*/
'blacklist_enabled' => env('JWT_BLACKLIST_ENABLED', true),
/*
| Blacklist Grace Period
| When multiple concurrent requests are made with the same JWT,
| it is possible that some of them fail, due to token regeneration
| on every request.
| Set grace period in seconds to prevent parallel request failure.
|
*/

'blacklist_grace_period' => env('JWT_BLACKLIST_GRACE_PERIOD', 0),
/*
| Cookies encryption
| By default Laravel encrypt cookies for security reason.
| Set it to true if you want to decrypt cookies.
*/
'decrypt_cookies' => false,
/*
| Providers
|-----
| Specify the various providers used throughout the package.
*/
'providers' => [
    /*
    |-----
    | JWT Provider
    |-----
    | Specify provider that is used to create and decode the tokens.
    */
    'jwt' => Tymon\JWTAuth\Providers\JWT\Lcobucci::class,
    /*
    | Authentication Provider
    |-----
    | Specify the provider that is used to authenticate users.
    */
    'auth' => Tymon\JWTAuth\Providers\Auth\Illuminate::class,
    /*
    | Storage Provider
    |-----
    | Specify provider that is used to store tokens in the blacklist.
    |
    */
    'storage' => Tymon\JWTAuth\Providers\Storage\Illuminate::class,
],
];

```

Console

Kernel

```
<?php
namespace App\Console;
use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     * @var array
     */
    protected $commands = [
        //
    ];

    /**
     * Define the application's command schedule.
     * @param \Illuminate\Console\Scheduling\Schedule $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {
        // $schedule->command('inspire')
        //     ->hourly();
    }

    /**
     * Register the commands for the application.
     * @return void
     */
    protected function commands()
    {
        $this->load(__DIR__.'/Commands');
        require base_path('routes/console.php');
    }
}
```

Exceptions

Handler

```
<?php
namespace App\Exceptions;
use Exception;
use Illuminate\Foundation\Exceptions\Handler as ExceptionHandler;
class Handler extends ExceptionHandler
{
    protected $dontReport = [
        //
    ];
    protected $dontFlash = [
        'password',
        'password_confirmation',
    ];

    /**
     * Report or log an exception.
     * @param \Exception $exception
     * @return void
     */
    public function report(Exception $exception)
    {
        parent::report($exception);
    }
}
```

```

/**
 * Render an exception into an HTTP response.
 * @param \Illuminate\Http\Request $request
 * @param \Exception $exception
 * @return \Illuminate\Http\Response
 */
public function render($request, Exception $exception)
{
    return parent::render($request, $exception);
}
}

```

Public

Index

```

<?php
define('LARAVEL_START', microtime (true));
/*
|-----|
| Register The Auto Loader
|-----|
| Composer provides a convenient, automatically generated class loader for
| our application. We just need to utilize it! Simply require it
| into the script here so, do not have to worry about manual
| loading any classes later on.
*/
require __DIR__.'../vendor/autoload.php';
/*
|-----|
| Turn On The Lights
|-----|
| This bootstraps the framework and gets it ready for use, then it
| will load up this application so that we can run it and send
| the responses back to the browser and delight our users.
*/
$app = require_once __DIR__.'../bootstrap/app.php';
/*
|-----|
| Run The Application
|-----|
| Once we have the application, we can handle the incoming request
| through the kernel, and send the associated response back to
| the client's browser allowing them to enjoy the creative
| and wonderful application we have prepared for them.
*/
$kernel = $app->make(Illuminate\Contracts\Http\Kernel::class);
$response = $kernel->handle(
    $request = Illuminate\Http\Request::capture()
);
$response->send();
$kernel->terminate($request, $response);

```

server

```

<?php
/** Laravel - A PHP Framework for Web Artisans
 * @package Laravel
 */
$uri = urldecode(
    parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH)

```

```
);
// This file allows us to emulate Apache's "mod_rewrite" functionality from
// built-in PHP web server. This provides a convenient way to test a
Laravel
// application without having installed a "real" web server software here.
if ($uri !== '/' && file_exists(__DIR__.'/public'.$uri)) {
    return false;
}
require_once __DIR__.'/public/index.php';
```

API

API - User

```
<?php
use Illuminate\Http\Request;
/*
|-----
| API Routes
|-----
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
*/
Route::group(['middleware'=>\App\Http\Middleware\Authenticated::class],function (){
    Route::get('dashboard-text','HomeController@getDashboardContent');
});
Route::group(['prefix'=>'auth'],function (){
    Route::post('login','AuthController@login');
    Route::post('register','AuthController@register');
    Route::post('send-forgot-password-email','AuthController@sendResetPasswordEmail');
    Route::post('reset-password','AuthController@resetPassword');
Route::group(['middleware'=>\App\Http\Middleware\Authenticated::class],function (){
    Route::get('logout','AuthController@logout');
});
});
require_once __DIR__.'/api/user.php';
require_once __DIR__.'/api/manager.php';
require_once __DIR__.'/api/employee.php';
require_once __DIR__.'/api/admin.php';
```

API - application

```
<?php
/*
|-----
| API Routes
|-----
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
*/
require_once __DIR__.'/api/admin.php';
require_once __DIR__.'/api/requirement.php';
require_once __DIR__.'/api/testing-tool.php';
require_once __DIR__.'/api/notification.php';
Route::group(['middleware'=>\StanleyMSACCommon\MSACCommon\Http\Middleware\Authentication::class],function (){
    Route::get('/', 'ProjectController@getIndex');
```

```

Route::group(['prefix'=>'{projectId}'],function () {
    Route::get('/', 'ProjectController@get');
    Route::get('complete', 'ProjectController@completeProject');
    Route::post('update', 'ProjectController@updateProject');
    Route::get('logs', 'ProjectController@getProjectLogs');
    Route::group(['prefix'=>'qa'],function () {
        Route::get('/', 'ProjectQAController@index');
        Route::post('/', 'ProjectQAController@submitProjectQAReport');
        Route::get('{qaId}', 'ProjectQAController@getProjectQaById');
    });
    Route::group(['prefix'=>'tasks'],function () {
        Route::get('/', 'TaskController@getProjectTasksList');
        Route::post('/', 'TaskController@create');
        Route::get('{taskId}', 'TaskController@getProjectTaskDetails');
        Route::get('{taskId}/complete', 'TaskController@completeTask');
    });
});
});

```

requirement

```

<?php
Route::group(['middleware'=>\StanleyMSACCommon\MSACCommon\Http\Middleware\Authentication::class, 'prefix'=>'requirement'],function () {
    Route::get('/', 'RequirementController@index');
    Route::post('/', 'RequirementController@create');
    Route::get('{projectId}', 'RequirementController@get');
    Route::get('{projectId}/delete', 'RequirementController@delete');
});

```

Testing-tool

```

<?php
Route::group(['prefix' => 'testing-
tool', 'middleware'=>\StanleyMSACCommon\MSACCommon\Http\Middleware\Authentication::class], function () {
    Route::get('/', 'TestingToolController@getIndex');
});

```

```

<?php
Route::group(['prefix' => 'testing-tool'], function () {
    Route::post('/', 'Admin\TestingToolController@create');
    Route::get('/testing-tools-
count', 'Admin\TestingToolController@getTestingToolsCount');
});

```

admin

```

<?php
Route::group(['prefix' =>
'admin', 'middleware'=>[\App\Http\Middleware\Authenticated::class, \App\Http\
Middleware\AdminAuthenticated::class]],function () {
    Route::post('dashboard-text', 'AdminController@updateDashboardContent');
    require_once __DIR__ . '/admin/user.php';
    require_once __DIR__ . '/admin/backup.php';
});

```

manager

```

<?php
Route::group(['middleware'=>\App\Http\Middleware\Authenticated::class, 'prefix' => 'manager'],function () {

```

```
Route::get('/', 'ManagerController@index');
});
```

user

```
<?php
Route::group(['middleware'=>\App\Http\Middleware\Authenticated::class, 'prefix' => 'user'], function () {
    Route::get('/', 'UserController@getUser');
    Route::get('{id}', 'UserController@getUserById');
    Route::get('admin-users-list', 'UserController@adminUsersList');
});
```

employee

```
<?php
Route::group(['middleware'=>\App\Http\Middleware\Authenticated::class, 'prefix' => 'employee'], function () {
    Route::get('/', 'EmployeeController@index');
});
```

backup

```
<?php
Route::group(['prefix' => 'backup', 'middleware'=>\App\Http\Middleware\AdminAuthenticated::class], function () {
    Route::get('create-database-dump-file', 'Admin\BackupController@createDatabaseDump');
    Route::get('download-backup-file', 'Admin\BackupController@downloadBackupFile');
});
```

notification

```
<?php
Route::group(['prefix' => 'notification', 'middleware'=>\StanleyMSACCommon\MSACCommon\Http\Middleware\Authentication::class], function () {
    Route::get('/', 'ProjectNotificationController@index');
    Route::get('{notificationId}/read', 'ProjectNotificationController@markAsRead');
});
```

Broadcast

```
<?php
/*
|-----
| Broadcast Channels
|-----
| Here you may register all of the event broadcasting channels that your
| application supports. The given channel authorization callbacks are
| used to check if an authenticated user can listen to the channel.
*/
Broadcast::channel('App.User.{id}', function ($user, $id) {
    return (int) $user->id === (int) $id;
});
```


Console

```
<?php
use Illuminate\Foundation\Inspiring;
/*
|-----
| Console Routes
|-----
| This file is where you may define all of your Closure based console
| commands. Each Closure is bound to a command instance allowing a
| simple approach to interacting with each command's IO methods.
*/
Artisan::command('inspire', function () {
    $this->comment(Inspiring::quote());
})->describe('Display an inspiring quote');
```

Web

```
<?php
/*
|-----
| Web Routes
|-----
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/
Route::get('/', function () {
    return view('welcome');
});
```

Database

DatabaseSeeder

```
<?php
use Illuminate\Database\Seeder;
class DatabaseSeeder extends Seeder
{
    /** Run the database seeds.
     * @return void
     */
    public function run()
    {
        // $this->call(UsersTableSeeder::class);
    }
}
```

UserFactory

```
<?php
use Faker\Generator as Faker;
/*
|-----
| Model Factories
|-----
| This directory should contain each of the model factory definitions for
| your application. Factories provide a convenient way to generate new
| model instances for testing / seeding your application's database.
*/
$factory->define(App\User::class, function (Faker $faker) {
    return [
```

```

        'name' => $faker->name,
        'email' => $faker->unique()->safeEmail,
        'password' =>
'$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdIVF1Yg7B77UdFm', // secret
        'remember_token' => str_random(10),
    ];
});

```

Tests

CreatesApplication

```

<?php
namespace Tests;
use Illuminate\Support\Facades\Hash;
use Illuminate\Contracts\Console\Kernel;
trait CreatesApplication
{
    /**
     * Creates the application.
     * @return \Illuminate\Foundation\Application
     */
    public function createApplication()
    {
        $app = require __DIR__.'/../bootstrap/app.php';
        $app->make(Kernel::class)->bootstrap();
        Hash::setRounds(4);
        return $app;
    }
}

```

TestCase

```

<?php
namespace Tests;
use Illuminate\Foundation\Testing\TestCase as BaseTestCase;
abstract class TestCase extends BaseTestCase
{
    use CreatesApplication;
}

```

ExampleTest (UnitTest)

```

<?php
namespace Tests\Unit;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;
class ExampleTest extends TestCase
{
    /**
     * A basic test example.
     * @return void
     */
    public function testBasicTest()
    {
        $this->assertTrue(true);
    }
}

```

ExampleTest (FeatureTest)

```

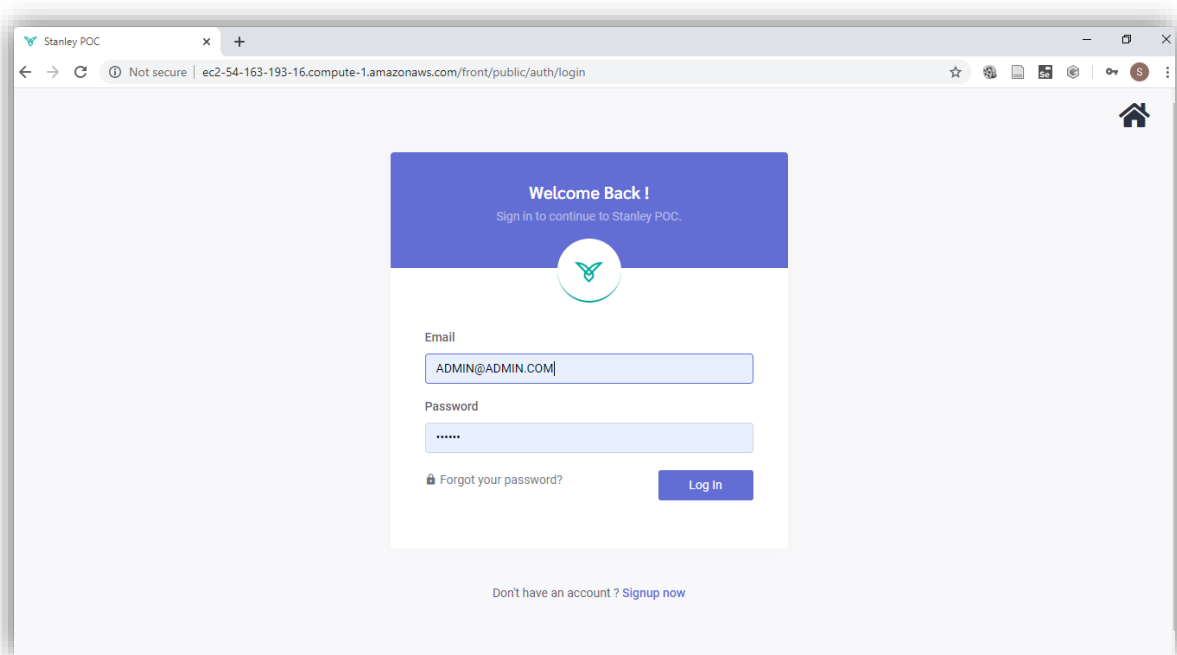
<?php
namespace Tests\Feature;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

```

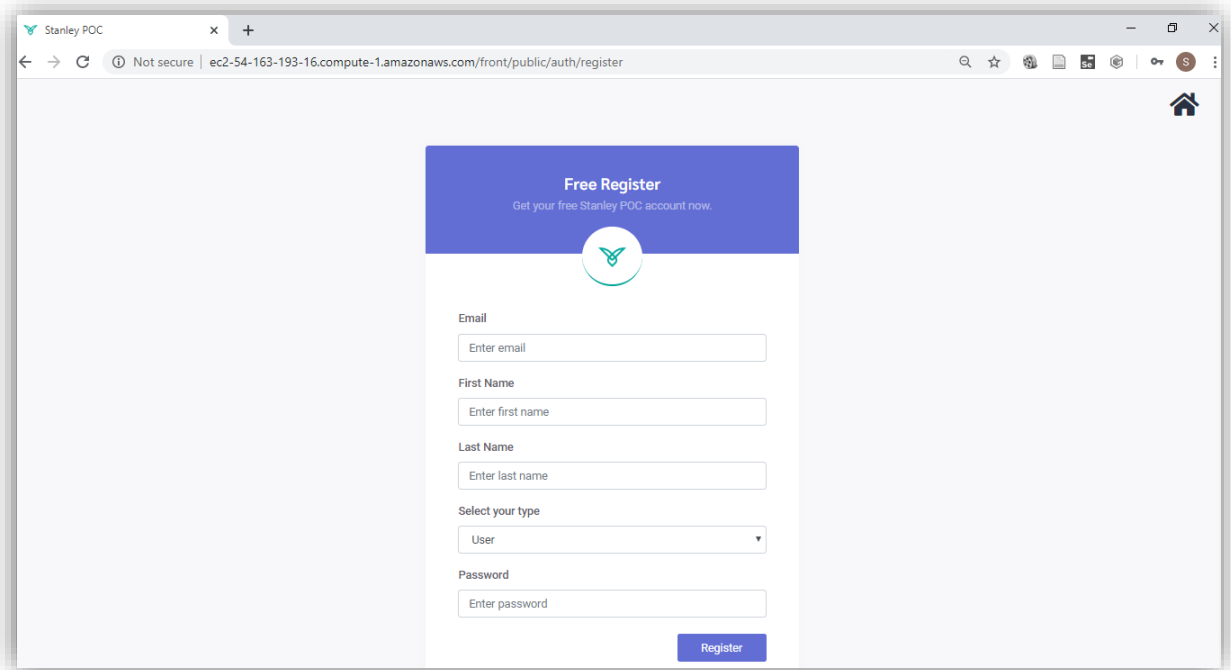
```
class ExampleTest extends TestCase
{
    /** A basic test example.
     * @return void
     */
    public function testBasicTest ()
    {
        $response = $this->get ('/');
        $response->assertStatus (200);
    }
}
```

C. POC IMPLEMENTATION OF FUNCTIONALITY

Application – Home (Index)



Registration

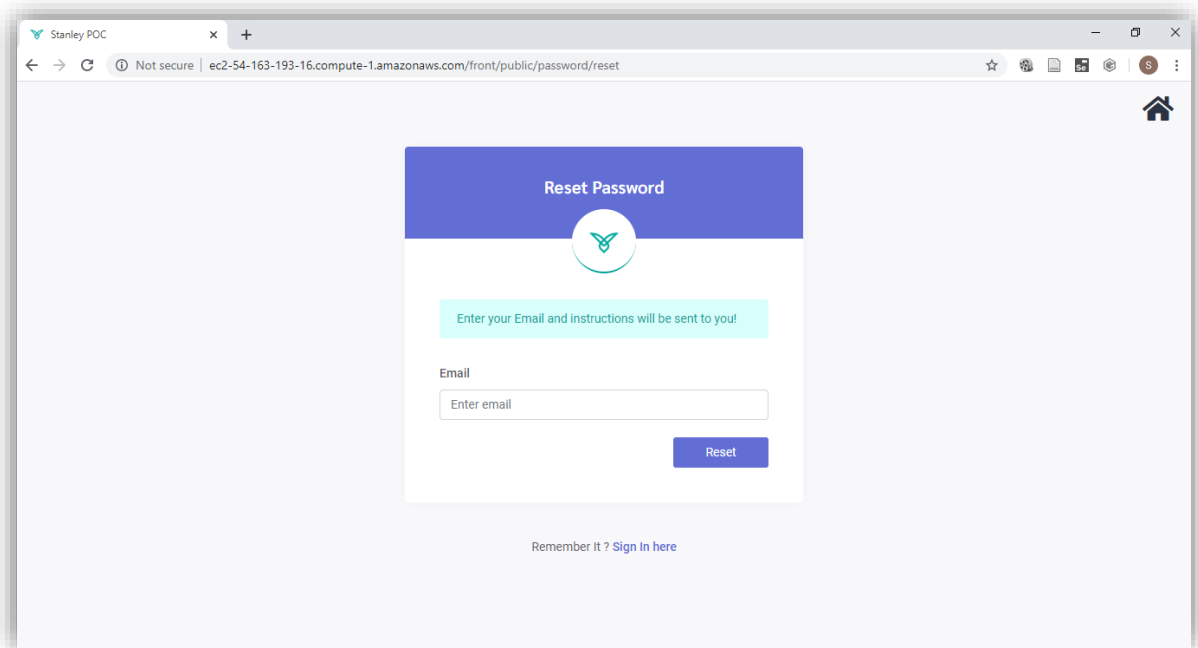


The screenshot shows a web browser window with the Stanley POC logo in the top left. The address bar shows the URL: `ec2-54-163-193-16.compute-1.amazonaws.com/front/public/auth/register`. The page features a blue header with the text "Free Register" and "Get your free Stanley POC account now." Below the header is a circular logo with a stylized bird. The registration form includes the following fields:

- Email:** A text input field with the placeholder "Enter email".
- First Name:** A text input field with the placeholder "Enter first name".
- Last Name:** A text input field with the placeholder "Enter last name".
- Select your type:** A dropdown menu with "User" selected.
- Password:** A text input field with the placeholder "Enter password".

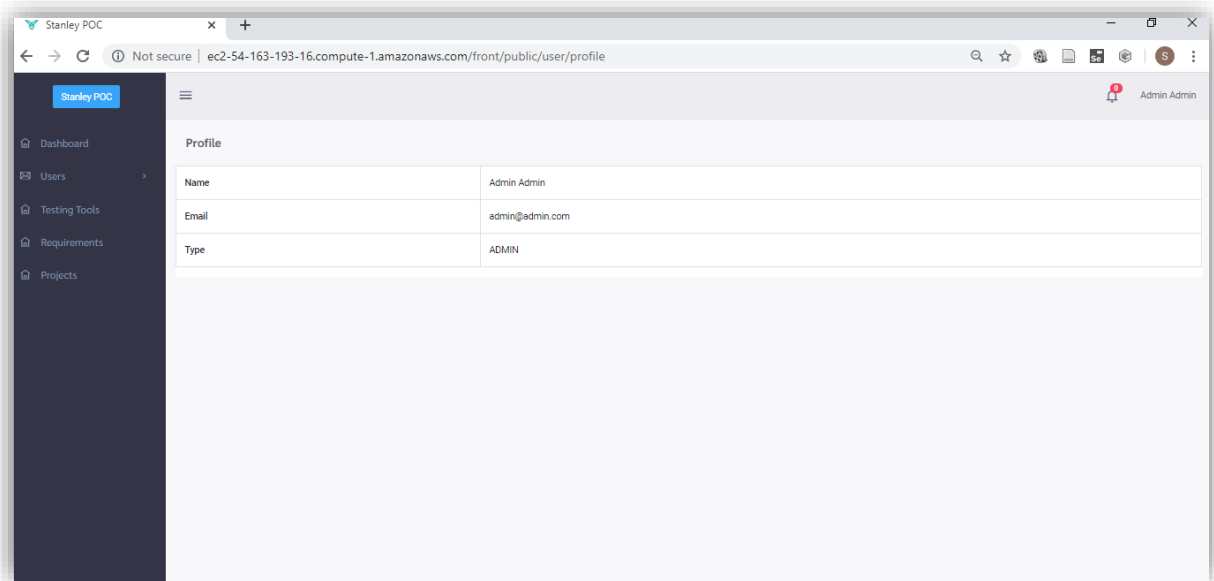
A blue "Register" button is located at the bottom right of the form.

Reset Password

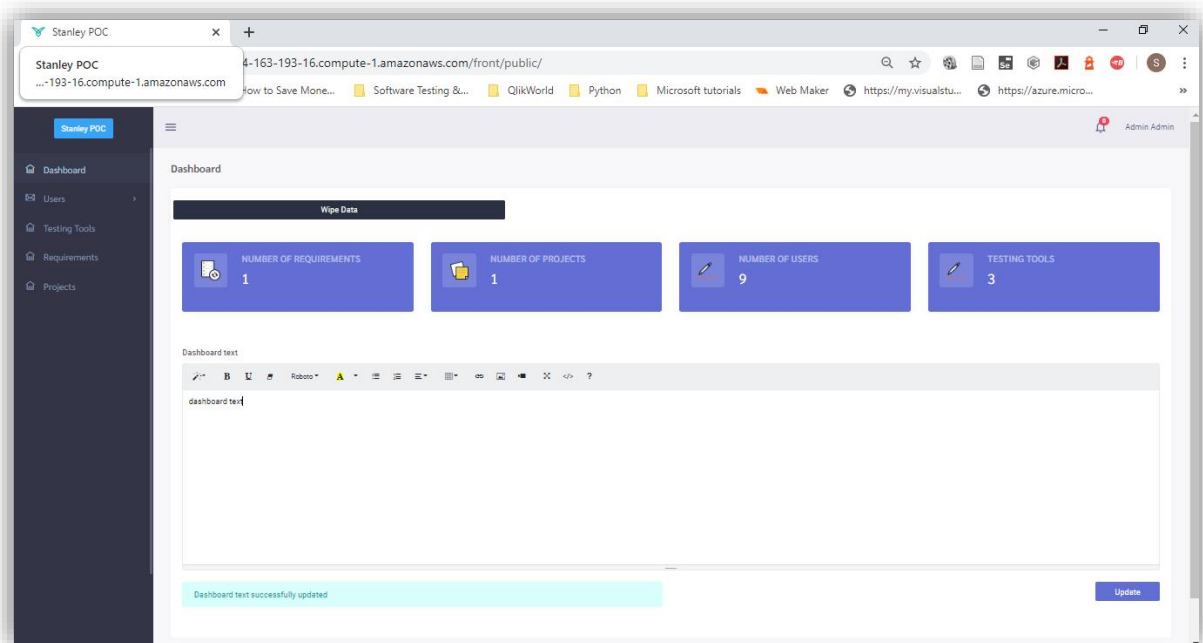


The screenshot shows a web browser window with the Stanley POC logo in the top left. The address bar shows the URL: `ec2-54-163-193-16.compute-1.amazonaws.com/front/public/password/reset`. The page features a blue header with the text "Reset Password" and a circular logo with a stylized bird. A light blue message box contains the text: "Enter your Email and instructions will be sent to you!". Below this is an "Email" section with a text input field and the placeholder "Enter email". A blue "Reset" button is located at the bottom right of the form. At the bottom of the page, there is a link: "Remember It ? [Sign In here](#)".

Admin profile



Admin Profile - Dashboard



Admin Profile – Projects

The screenshot shows a web application interface for 'Stanley POC'. The left sidebar contains navigation links: Dashboard, Users, Testing Tools, Requirements, and Projects. The main content area is titled 'Projects' and displays a table with the following data:

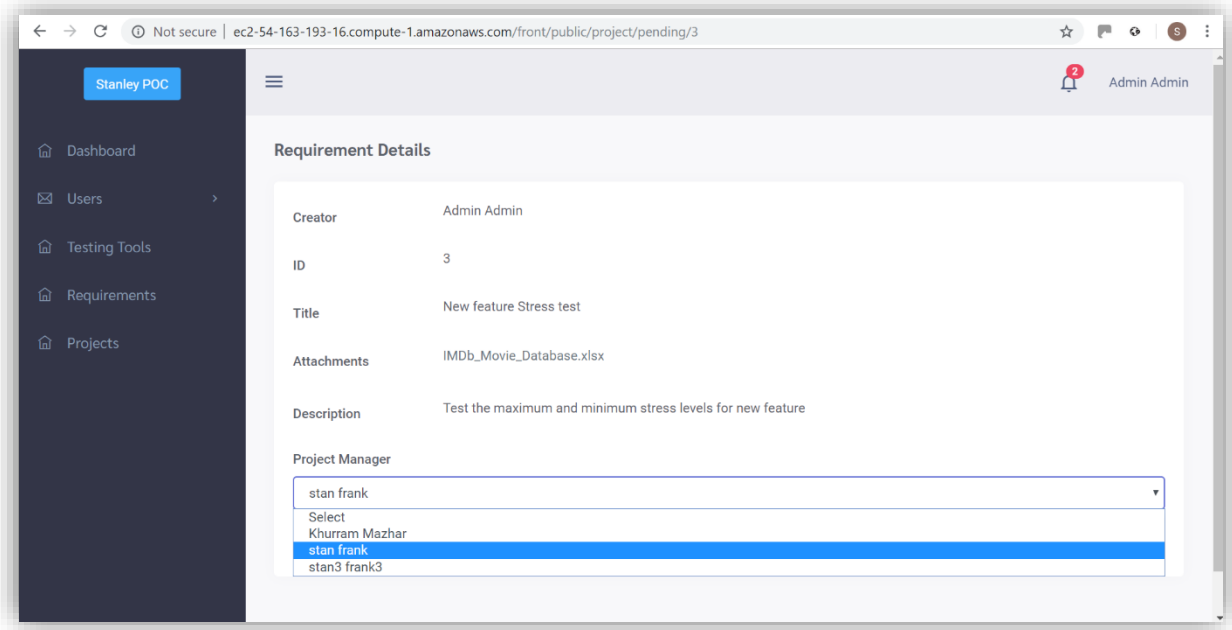
#	Title	Created	Status	Actions
1	asdadasd	2019-09-01 21:39:48	Completed	Details
2	New feature Load test	2019-09-07 22:45:15	Pending	Details Edit Close
3	New feature Stress test	2019-09-07 22:46:50	Pending	Details Edit Close
4	Test	2019-09-08 00:41:21	Pending	Details Edit Close
5	Requirement 1 for stan1(user1)	2019-09-11 04:05:45	Pending	Details Edit Close

Admin Profile – Pending Projects

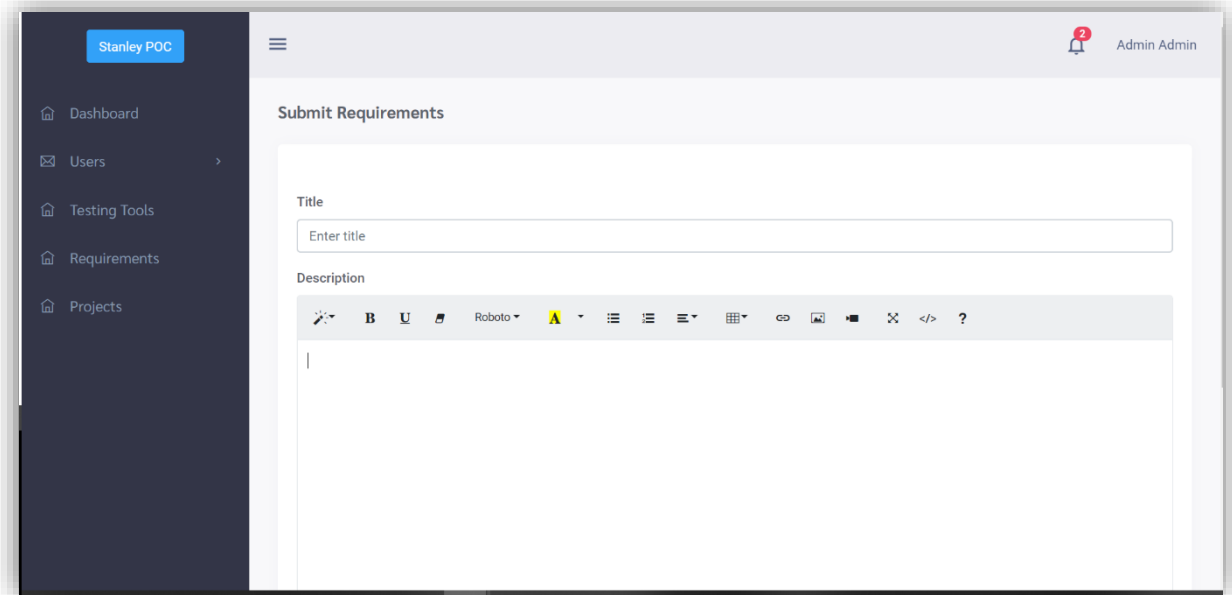
The screenshot shows the 'Pending Projects' section of the web application. The left sidebar is the same as in the previous screenshot. The main content area is titled 'Pending Projects' and includes a 'Submit New Requirement' button in the top right. The table below contains the following data:

#	Title	Created	Actions
3	New feature Stress test	2019-09-07 22:46:50	Details Delete
6	Requirement 2 for stan1(user1)	2019-09-11 04:06:37	Details Delete
7	Requirement 1 for stan4(user)	2019-09-11 04:12:26	Details Delete
8	Requirement 2 for stan4(user)	2019-09-11 04:12:55	Details Delete
9	requirement amendment for requirement ID 5	2019-09-11 04:17:56	Details Delete

Admin Profile – Requirements



Admin Profile – Submit Requirements



Admin Profile - Active Users

Stanley POC

Admin Admin

Active Users

#	Name	Email	Type	Created	Status	Actions
5	Segun Adedugbe	segun.adedugbe@gmail.com	USER	2019-09-08 00:39:16	Active	De-activate
6	stan frank	stanleyewenike@icloud.com	MANAGER	2019-09-08 01:47:23	Active	De-activate
7	stan1 frank1	stanleyewenike@gmail.com	USER	2019-09-11 03:47:31	Active	De-activate
8	stan2 frank2	stanleyewenike@yahoo.co.uk	EMPLOYEE	2019-09-11 03:48:44	Active	De-activate
10	stan4 frank4	Stanley.Ewenike@staffs.ac.uk	USER	2019-09-11 03:51:42	Active	De-activate
12	Joe Bloggs	joe@bloggs.com	USER	2019-09-17 12:53:29	Active	De-activate

Admin Profile - Pending Users

Stanley POC

Admin Admin

Pending Users

#	Name	Email	Type	Created	Actions
1	Khurram Mazhar	mkhurram900@gmail.com	USER	2019-09-01 21:26:31	Activate
3	Khurram Mazhar	mkhurram900@yahoo.com	MANAGER	2019-09-01 21:34:44	Activate
4	Khurram Mazhar	mkhurram900@gmail.com1	EMPLOYEE	2019-09-01 21:40:17	Activate
9	stan3 frank3	stancity@hotmail.com	MANAGER	2019-09-11 03:50:53	Activate
11	stan5 frank5	ew009265@student.staffs.ac.uk	EMPLOYEE	2019-09-11 03:52:41	Activate

Admin Profile – Testing Tools

Testing Tools List

Create New Tool

#	Title	Created
1	Loader.io	2019-09-04 01:21:57
2	Load Impact	2019-09-04 01:27:49
3	Test	2019-09-17 13:18:39

Admin Profile – Add Testing Tools

Add Testing Tool

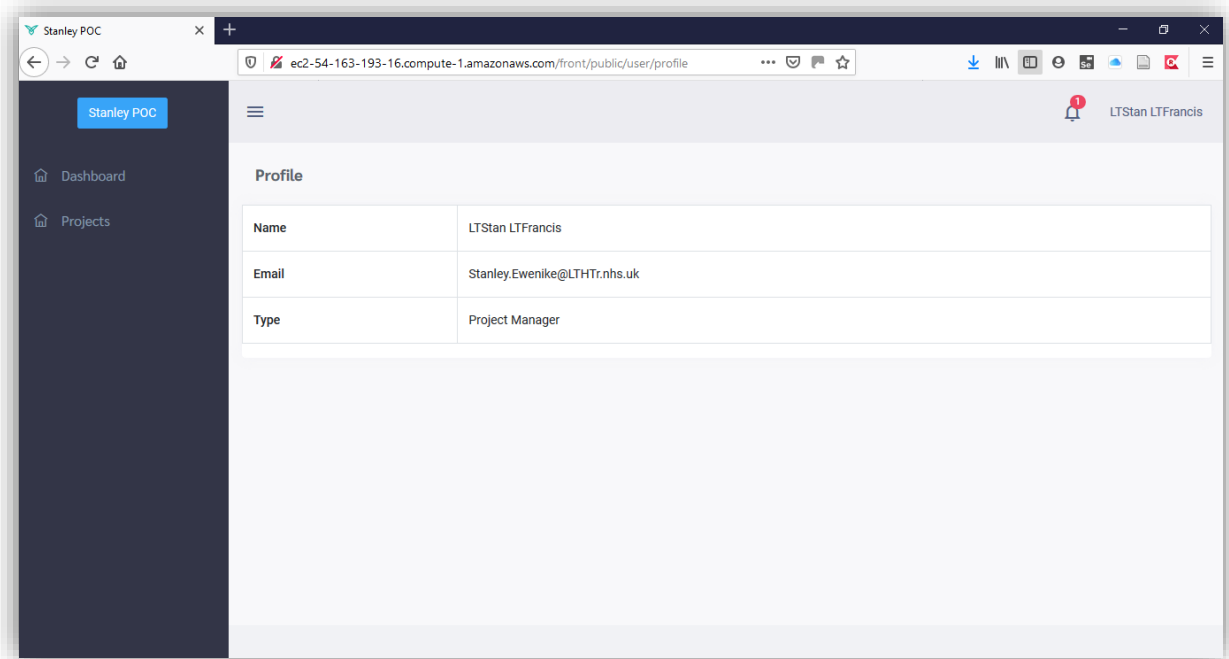
Title
Enter Name

URL
Enter Url

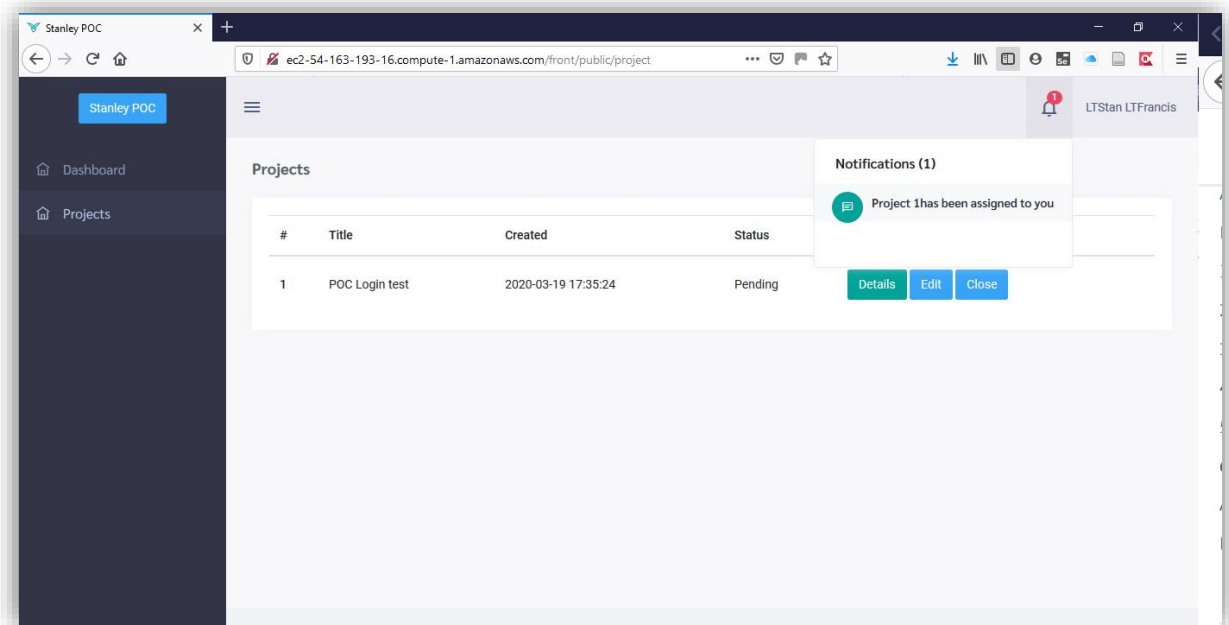
Description

Rich text editor toolbar: Bold, Underline, Font Color, Background Color, Bulleted List, Numbered List, Indent, Outdent, Link, Unlink, Source Code, Help.

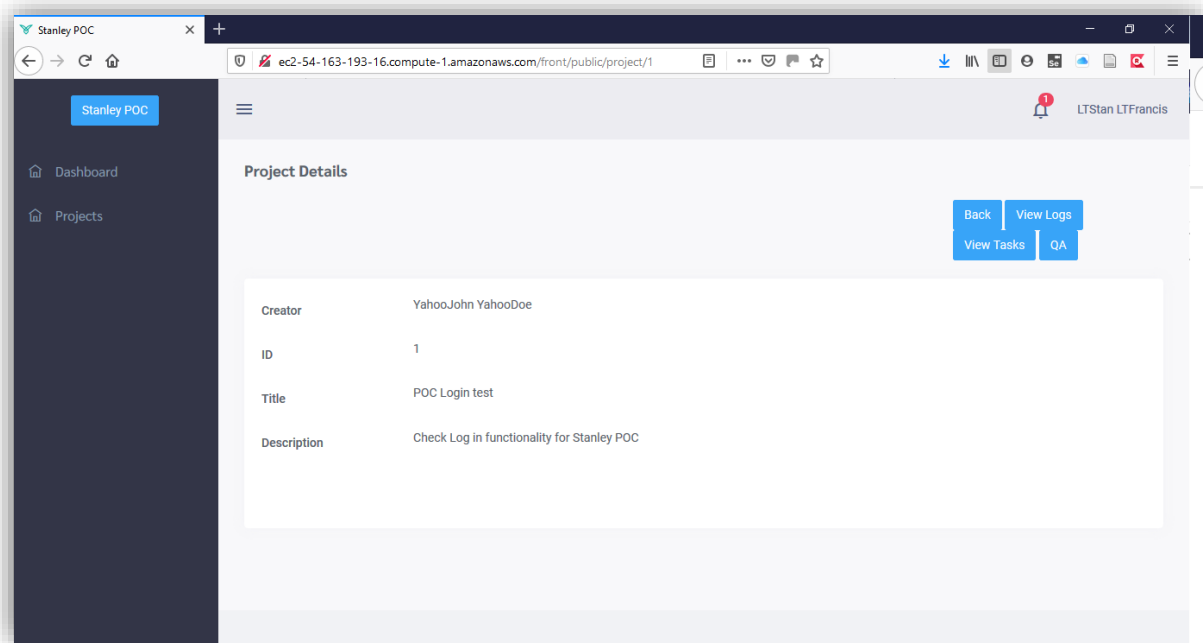
Project Manager Profile



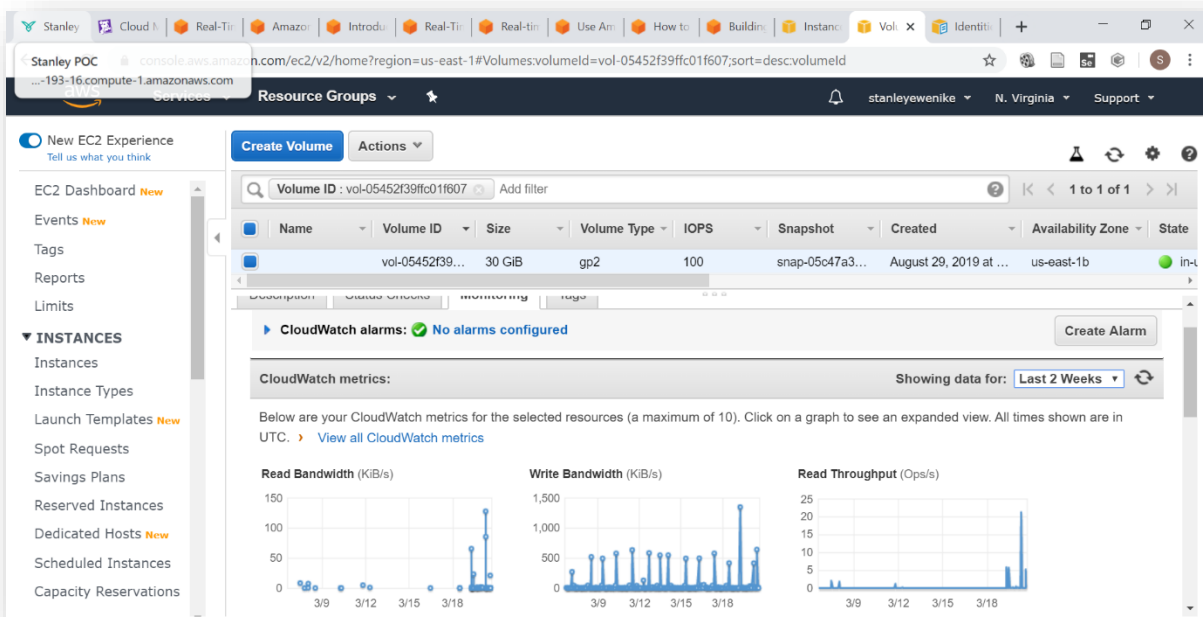
Project Manager Profile - Projects

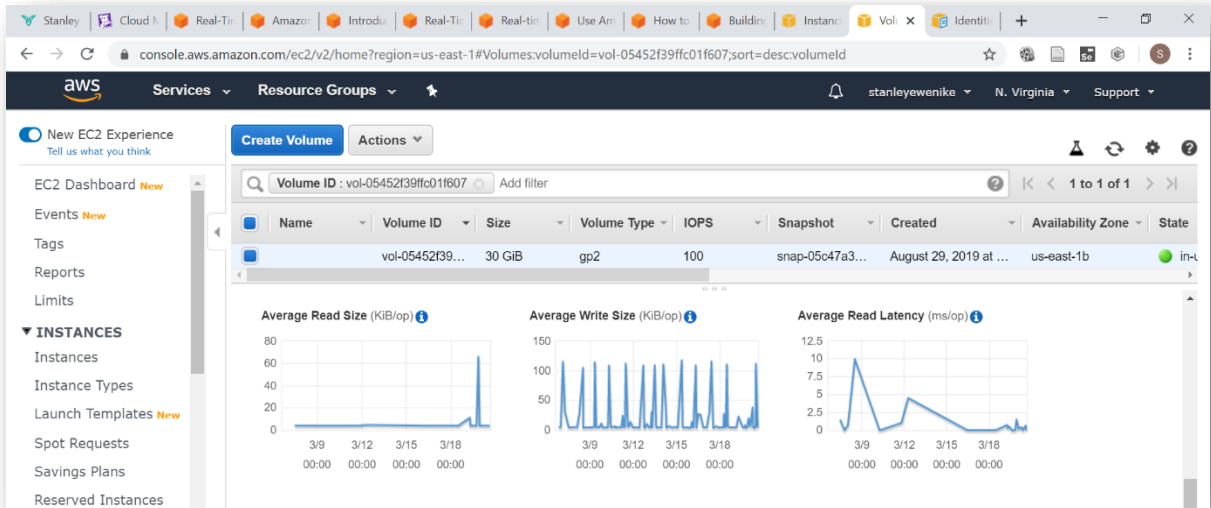
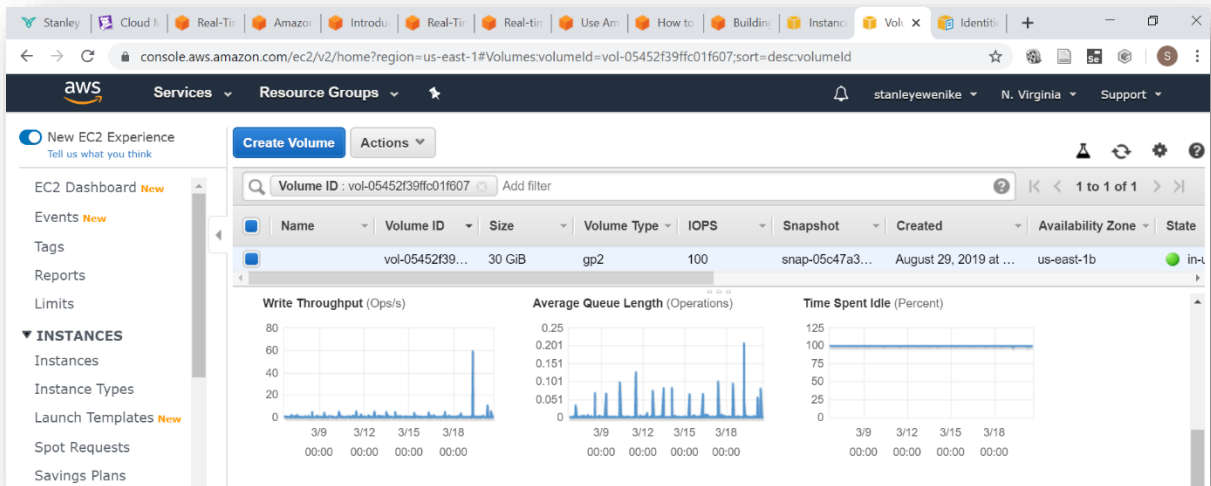


Project Manager Profile - Project details



Some metrics from testing activity for Test scenario





Source code for API call

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Stanley POC</title>
  <meta content="Admin Dashboard" name="description" />
  <meta content="Themesbrand" name="author" />
  <link rel="shortcut icon" href="assets/images/favicon.ico">
  <!-- App favicon -->
  <link rel="shortcut icon" href="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/images/favicon.ico">
  <!-- App css -->
  <link href="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/css/bootstrap.min.css" rel="stylesheet" type="text/css" />
  <link href="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/css/metismenu.min.css" rel="stylesheet" type="text/css">
```

```

<link href="http://ec2-54-163-193-16.compute-1.amazonaws.com/front/public/assets/css/icons.css"
rel="stylesheet" type="text/css" />
<link href="http://ec2-54-163-193-16.compute-1.amazonaws.com/front/public/assets/css/style.css"
rel="stylesheet" type="text/css" />
</head>
<body class="pb-0">
<div class="home-btn d-none d-sm-block">
<a href="index" class="text-dark">
<i class="fas fa-home h2"></i>
</a>
</div>
<div class="wrapper-page">
<div class="card overflow-hidden account-card mx-3">
<div class="bg-primary p-4 text-white text-center position-relative">
<h4 class="font-20 m-b-5">Welcome Back !</h4>
<p class="text-white-50 mb-4">Sign in to continue to Stanley POC.</p>
<a href="index" class="logo logo-admin">

</a>
</div>
<div class="account-card-content">
<form class="form-horizontal m-t-30" method="post">
<input type="hidden" name="_token" value="kVOUp4aoLrdSqqQcdJAeXDqgtPwRQ1TbIDNhfXJ7">
<div class="form-group">
<label for="email">Email</label>
<input type="email" class="form-control" id="email" placeholder="Enter username"
name="email"
value="">
</div>
<div class="form-group">
<label for="password">Password</label>
<input type="password" class="form-control" id="password" name="password"
placeholder="Enter password">
</div>
<div class="form-group row m-t-20">
<div class="col-sm-6">
<a href="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/password/reset">
<i class="mdi mdi-lock"></i> Forgot your
password?
</a>
</div>
<div class="col-sm-6 text-right">
<button class="btn btn-primary w-md waves-effect waves-light" type="submit">Log
In</button>
</div>
</div>
</div>
</form>
</div>
</div>
<div class="m-t-40 text-center">
<p>Don't have an account ?</p>
<a href="http://ec2-54-163-193-16.compute-1.amazonaws.com/front/public/auth/register"
class="font-500 text-primary"> Signup
now </a>

```

```

    </p>
  </div>
</div>
<!-- end wrapper-page -->
<!-- App's Basic Js -->
<script src="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/js/jquery.min.js"></script>
<script src="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/js/bootstrap.bundle.min.js"></script>
<script src="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/js/metisMenu.min.js"></script>
<script src="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/js/jquery.slimscroll.js"></script>
<script src="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/js/waves.min.js"></script>
<!-- App js-->
<script src="http://ec2-54-163-193-16.compute-
1.amazonaws.com/front/public/assets/js/app.js"></script>
</body>
</html>

```

#YAML source script for test scenario (Plugin tool used – Blazemeter)

```

modules:
  nose:
    ignore-unknown-actions: true
execution:
  - executor: selenium
    scenario: 1 Check Login functionality-Selenium
    blazegrid: true
    iterations: 1
    capabilities:
      browserName: chrome
    locations:
      harbor-5ab3c64ec8589f914c7b25db: 1
      harbor-5d25f94f9950ce73cd105f53: 1
      harbor-5d25f95f206f10730f21f1b4: 1
scenarios:
  1 Check Login functionality-Selenium:
    generate-flow-markers: true
    headless: false
    timeout: 60s
    think-time: 0s
    requests:
      - label: Test
        actions:
          - 'resizeWindow(1366,625)'
          - 'go(chrome://newtab/)'
          - go(http://ec2-54-163-193-16.compute-1.amazonaws.com/front/public/auth/login)
      - label: Enter valid email
        actions:
          - clickByID(email)
          - typeByID(email): ADMIN@ADMIN.COM
          - clickByCSS(body.pb-0)
      - label: Enter valid password
        actions:
          - clickByID(password)
          - typeByID(password): '123123'

```

- *clickByCSS(body.pb-0)*
- *label: Authenticate by clicking Login*
- actions:*
 - *clickByCSS(button.btn.btn-primary.w-md.waves-effect.waves-light)*
 - *clickByLinkText(Admin Admin)*
 - *clickByLinkText(Logout)*
- *label: Enter invalid email*
- actions:*
 - *clickByID(email)*
 - *typeByID(email): admin@staff.com*
- *label: Enter valid password*
- actions:*
 - *clickByID(password)*
 - *typeByID(password): '123123'*
- *label: Authenticate by clicking Login*
- actions:*
 - *clickByCSS(button.btn.btn-primary.w-md.waves-effect.waves-light)*
- *label: Enter valid email*
- actions:*
 - *clickByID(email)*
 - *typeByID(email): ADMIN@ADMIN.COM*
- *label: Enter invalid password*
- actions:*
 - *clickByID(password)*
 - *typeByID(password): test123*
- *label: Authenticate by clicking Login*
- actions:*
 - *clickByCSS(button.btn.btn-primary.w-md.waves-effect.waves-light)*
- *label: Enter invalid email*
- actions:*
 - *clickByCSS(body.pb-0)*
 - *typeByID(email): janedoe@sky.com*
- *label: Enter invalid password*
- actions:*
 - *clickByID(password)*
 - *typeByID(password): newfiles243*
- *label: Authenticate by clicking Login*
- actions:*
 - *clickByCSS(button.btn.btn-primary.w-md.waves-effect.waves-light)*
- *label: leave email field blank*
- actions:*
 - *clickByCSS(div.account-card-content)*
 - *typeByID(email): ''*
 - *clickByCSS(body.pb-0)*
- *label: Leave password field blank*
- actions:*
 - *clickByCSS(body.pb-0)*
- *label: Authenticate by clicking Login*
- actions:*
 - *clickByCSS(button.btn.btn-primary.w-md.waves-effect.waves-light)*
- *label: Check forgot password is working*
- actions:*
 - *clickByCSS(div.col-sm-6 > a)*
 - *clickByID(email)*
 - *typeByID(email): ew009265@student.staffs.ac.uk*
 - *clickByCSS(button.btn.btn-primary.w-md.waves-effect.waves-light)*

Metadata for Test scenario

```
{
  "sessionId": "77b5d51e1fe41006a7d71754f2fcc86b",
  "browserName": "chrome",
  "rotatable": false,
  "acceptInsecureCerts": true,
  "browserConnectionEnabled": false,
  "handlesAlerts": true,
  "databaseEnabled": false,
  "unexpectedAlertBehaviour": "ignore",
  "cssSelectorsEnabled": true,
  "acceptSslCerts": true,
  "hasTouchScreen": false,
  "networkConnectionEnabled": false,
  "setWindowRect": true,
  "version": "69.0.3497.92",
  "takesHeapSnapshot": true,
  "mobileEmulationEnabled": false,
  "javascriptEnabled": true,
  "applicationCacheEnabled": false,
  "goog:chromeOptions": {
    "debuggerAddress": "localhost:43111"
  },
  "pageLoadStrategy": "normal",
  "platform": "Linux",
  "chrome": {
    "chromedriverVersion": "2.44.609551 (5d576e9a44fe4c5b6a07e568f1ebc753f1214634)",
    "userDataDir": "/tmp/.org.chromium.Chromium.YssWPI"
  },
  "webStorageEnabled": true,
  "locationContextEnabled": true,
  "takesScreenshot": true,
  "webdriver.remote.sessionid": "77b5d51e1fe41006a7d71754f2fcc86b",
  "nativeEvents": true
}
```

Check-Login-functionality.jmx source file

```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="2.4" jmeter="4.0">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Check Login functionality"
      enabled="true">
      <stringProp name="TestPlan.comments">This test plan was created by the BlazeMeter converter v.2.3.14.
      Please contact support@blazemeter.com for further support.</stringProp>
      <boolProp name="TestPlan.functional_mode">>false</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">>false</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments">
      <collectionProp name="Arguments.arguments"/>
    </TestPlan>
  </hashTree>
</jmeterTestPlan>
```



```

</elementProp>

<stringProp name="TestPlan.user_define_classpath"></stringProp>

</TestPlan>

<hashTree>

  <HeaderManager guiclass="HeaderPanel" testclass="HeaderManager" testname="HTTP Header
manager">

    <collectionProp name="HeaderManager.headers">

      <elementProp name="Accept" elementType="Header">

        <stringProp name="Header.name">Accept</stringProp>

        <stringProp
name="Header.value">text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.9</stringProp>

        </elementProp>

        <elementProp name="Upgrade-Insecure-Requests" elementType="Header">

          <stringProp name="Header.name">Upgrade-Insecure-Requests</stringProp>

          <stringProp name="Header.value">1</stringProp>

        </elementProp>

        <elementProp name="User-Agent" elementType="Header">

          <stringProp name="Header.name">User-Agent</stringProp>

          <stringProp name="Header.value">Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36</stringProp>

        </elementProp>

        <elementProp name="DNT" elementType="Header">

          <stringProp name="Header.name">DNT</stringProp>

          <stringProp name="Header.value">1</stringProp>

        </elementProp>

      </collectionProp>

    </HeaderManager>

  <hashTree/>

  <Arguments guiclass="ArgumentsPanel" testclass="Arguments" testname="User Defined Variables"
enabled="true">

    <collectionProp name="Arguments.arguments">

      <elementProp name="BASE_URL_1" elementType="Argument">

        <stringProp name="Argument.name">BASE_URL_1</stringProp>

        <stringProp name="Argument.value">ec2-54-163-193-16.compute-1.amazonaws.com</stringProp>

```

```

    </elementProp>
  </collectionProp>
</Arguments>
<hashTree/>
  <ConfigTestElement guiclass="HttpDefaultsGui" testclass="ConfigTestElement" testname="HTTP Request
Defaults" enabled="true">
    <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
      <collectionProp name="Arguments.arguments"/>
    </elementProp>
    <boolProp name="HTTPSampler.concurrentDwn">true</boolProp>
    <boolProp name="HTTPSampler.image_parser">true</boolProp>
    <intProp name="HTTPSampler.concurrentPool">6</intProp>
  </ConfigTestElement>
<hashTree/>
  <DNSCacheManager guiclass="DNSCachePanel" testclass="DNSCacheManager" testname="DNS Cache
Manager" enabled="true">
    <collectionProp name="DNSCacheManager.servers"/>
    <boolProp name="DNSCacheManager.clearEachIteration">true</boolProp>
  </DNSCacheManager>
<hashTree/>
  <AuthManager guiclass="AuthPanel" testclass="AuthManager" testname="HTTP Authorization Manager">
    <collectionProp name="AuthManager.auth_list"/>
  </AuthManager>
<hashTree/>
  <CookieManager guiclass="CookiePanel" testclass="CookieManager" testname="HTTP Cookie Manager"
enabled="true">
    <collectionProp name="CookieManager.cookies"/>
    <boolProp name="CookieManager.clearEachIteration">true</boolProp>
  </CookieManager>
<hashTree/>
  <CacheManager guiclass="CacheManagerGui" testclass="CacheManager" testname="HTTP Cache
Manager">
    <boolProp name="clearEachIteration">true</boolProp>
    <boolProp name="useExpires">false</boolProp>

```

```

</CacheManager>

<hashTree/>

<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Thread Group"
enabled="true">

  <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>

  <elementProp name="ThreadGroup.main_controller" elementType="LoopController">

    <boolProp name="LoopController.continue_forever">false</boolProp>

    <stringProp name="LoopController.loops">1</stringProp>

  </elementProp>

  <intProp name="ThreadGroup.num_threads">1</intProp>

  <intProp name="ThreadGroup.ramp_time">1</intProp>

  <boolProp name="ThreadGroup.scheduler">false</boolProp>

  <longProp name="ThreadGroup.duration">0</longProp>

  <longProp name="ThreadGroup.delay">0</longProp>

</ThreadGroup>

<hashTree>

  <TransactionController guiclass="TransactionControllerGui" testname="Test" enabled="true">

    <boolProp name="TransactionController.includeTimers">false</boolProp>

  </TransactionController>

  <hashTree>

    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="http://ec2-
54-163-193-16.compute-1.amazonaws.com/front/public/auth/login" enabled="true">

      <elementProp name="HTTPSampler.Arguments" elementType="Arguments">

        <collectionProp name="Arguments.arguments"/>

      </elementProp>

      <boolProp name="HTTPSampler.follow_redirects">true</boolProp>

      <boolProp name="HTTPSampler.use_keepalive">true</boolProp>

      <stringProp name="HTTPSampler.protocol">http</stringProp>

      <stringProp name="HTTPSampler.domain">${BASE_URL_1}</stringProp>

      <intProp name="HTTPSampler.port">0</intProp>

      <stringProp name="HTTPSampler.path">front/public/auth/login</stringProp>

      <stringProp name="HTTPSampler.method">GET</stringProp>

    </HTTPSamplerProxy>

  </hashTree>

```

```

    <ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer"
enabled="true">
    <stringProp name="ConstantTimer.delay">0</stringProp>
</ConstantTimer>
<hashTree/>
</hashTree>
</hashTree>
<TransactionController guiclass="TransactionControllerGui" testname="Navigate to POC URL"
enabled="true">
    <boolProp name="TransactionController.includeTimers">>false</boolProp>
</TransactionController>
<hashTree>
    <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="http://ec2-
54-163-193-16.compute-1.amazonaws.com/front/public/auth/login" enabled="true">
    <elementProp name="HTTPSampler.Arguments" elementType="Arguments">
    <collectionProp name="Arguments.arguments"/>
</elementProp>
<boolProp name="HTTPSampler.follow_redirects">>true</boolProp>
<boolProp name="HTTPSampler.use_keepalive">>true</boolProp>
<stringProp name="HTTPSampler.protocol">http</stringProp>
<stringProp name="HTTPSampler.domain">${BASE_URL_1}</stringProp>
<intProp name="HTTPSampler.port">0</intProp>
<stringProp name="HTTPSampler.path">front/public/auth/login</stringProp>
<stringProp name="HTTPSampler.method">GET</stringProp>
</HTTPSamplerProxy>
<hashTree>
    <ConstantTimer guiclass="ConstantTimerGui" testclass="ConstantTimer" testname="Constant Timer"
enabled="true">
    <stringProp name="ConstantTimer.delay">45017</stringProp>
</ConstantTimer>
<hashTree/>
</hashTree>
</hashTree>
</hashTree>

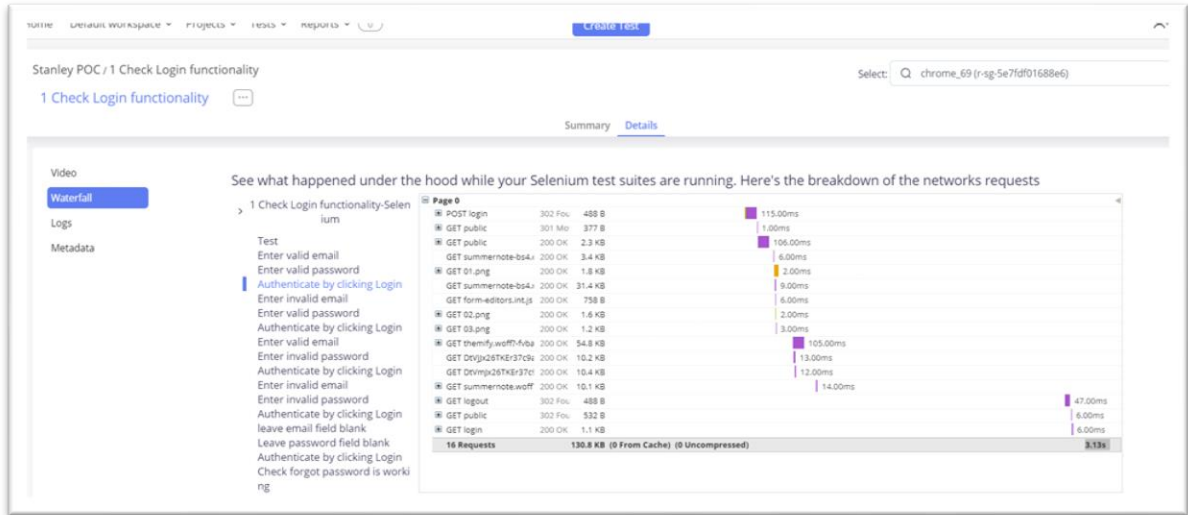
```

</hashTree>

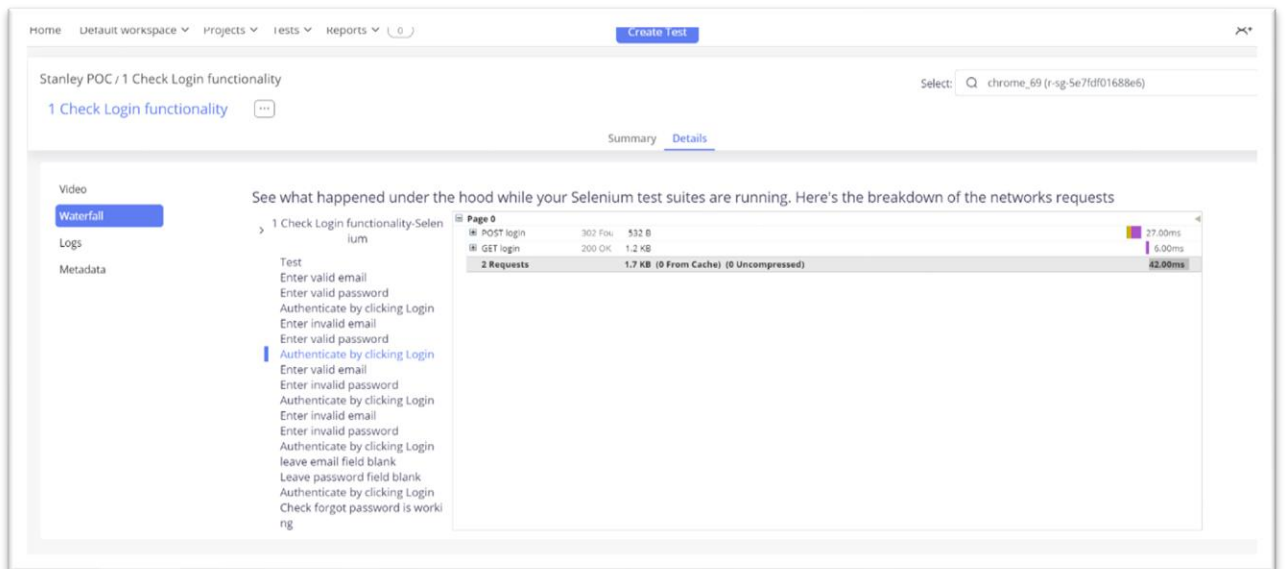
</hashTree>

</jmeterTestPlan>

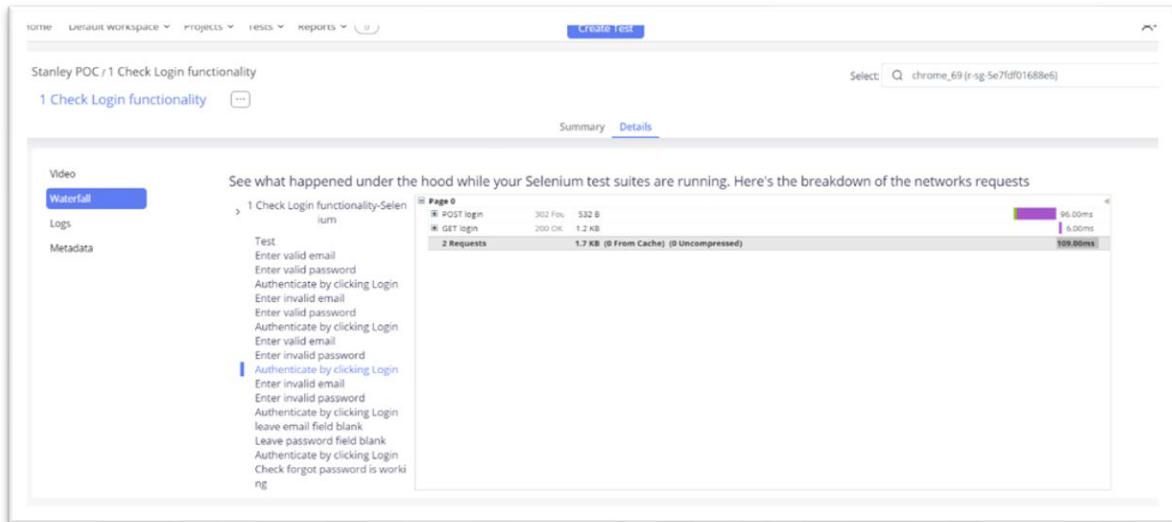
Results for test case 1



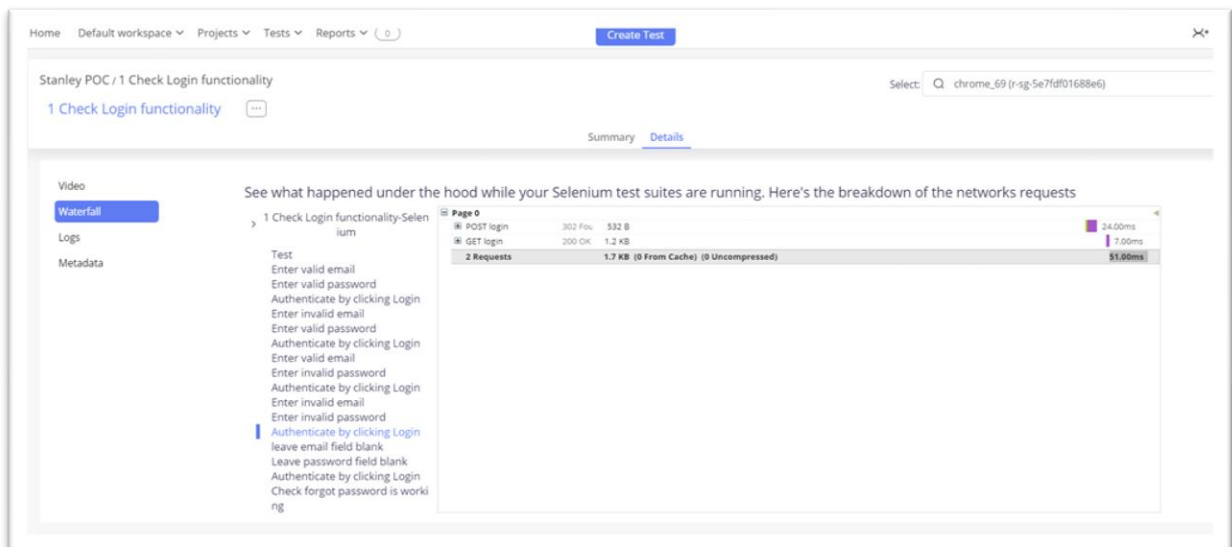
Results for test case 2



Results for test case 3



Results for test case 4



Results for test case 5

Home Default workspace Projects Tests Reports (6) Create Test

Stanley POC / 1 Check Login functionality Select: chrome_69 (r-sg-5e7fd01688e6)

1 Check Login functionality

Summary Details

Video

Waterfall

Logs

Metadata

See what happened under the hood while your Selenium test suites are running. Here's the breakdown of the networks requests

1 Check Login functionality-Selenium

Test

- Enter valid email
- Enter valid password
- Authenticate by clicking Login
- Enter invalid email
- Enter valid password
- Authenticate by clicking Login
- Enter valid email
- Enter invalid password
- Authenticate by clicking Login
- Enter invalid email
- Enter invalid password
- Authenticate by clicking Login
- leave email field blank
- Leave password field blank
- Authenticate by clicking Login
- Check forgot password is working

Page 0

POST login	302 Fox	532 B	13.00ms
GET login	200 OK	1.2 KB	5.00ms
2 Requests			28.00ms

1.7 KB (0 From Cache) (0 Uncompressed)

Results for test case 6

Home Default workspace Projects Tests Reports (6) Create Test

Stanley POC / 1 Check Login functionality Select: chrome_69 (r-sg-5e7fd01688e6)

1 Check Login functionality

Summary Details

Video

Waterfall

Logs

Metadata

See what happened under the hood while your Selenium test suites are running. Here's the breakdown of the networks requests

1 Check Login functionality-Selenium

Test

- Enter valid email
- Enter valid password
- Authenticate by clicking Login
- Enter invalid email
- Enter valid password
- Authenticate by clicking Login
- Enter valid email
- Enter invalid password
- Authenticate by clicking Login
- Enter invalid email
- Enter invalid password
- Authenticate by clicking Login
- leave email field blank
- Leave password field blank
- Authenticate by clicking Login
- Check forgot password is working

Page 0

GET reset	200 OK	1.1 KB	6.00ms
POST email	500 Int	669.9 K	628.00ms
GET favicon.ico	404 Not	303 B	2.00ms
3 Requests			637ms

671.3 KB (0 From Cache) (0 Uncompressed)