

A Faceted Taxonomy of Requirements Changes in Agile Contexts

Kashumi Madampe, *Graduate Student Member, IEEE*, Rashina Hoda, *Member, IEEE*,
and John Grundy, *Senior Member, IEEE*

Abstract—Background: Originally, developers aimed to identify most software requirements upfront in software development projects. However, agile methods explicitly encourage software requirements to be changed throughout development, i.e., many Requirements Changes (RCs) occur. **Objective:** The objective of this study is to better understand RCs and produce a taxonomy of RCs in agile contexts. **Method:** We ran a mixed-methods approach comprising a series of studies: an interview-based study (10 participants from New Zealand and Australia), a focused literature review, and an in-depth survey (40 participants world-wide). **Results:** Key characteristics of RCs in agile we found relate to different *types* and *forms*, agile RCs have multiple *reasons* and *sources*, they are brought by different *carriers*, and their emergence in agile is via a variety of *events*. **Summary:** The presented taxonomy provides a guide for software practitioners to use to help manage RC-related issues in agile contexts.

Index Terms—agile software development, requirements engineering, requirements changes, software engineering, taxonomy

1 INTRODUCTION

The IEEE Standard Glossary of Software Engineering [1] defines a “*requirement*” as “(a) a condition or capability needed by a user to solve a problem or achieve an objective; (b) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; (c) a documented representation of a condition or capability as in (a) or (b).” This original definition of a software requirement has not changed over time. However, the practical definition of a requirement has changed with the invention of agile software development methods – or what we call in this paper “Agile” for short. Agile encourages emergent requirements and incremental elicitation of requirements during development. This naturally leads to a great many *Requirements Changes* (RCs) throughout development [2], [3]. Agile teams must cope with many such RCs, which can pose a risk to the cost, quality, and schedule of the project [4].

There has been limited study of RCs in agile development contexts to date. Therefore, we decided to comprehensively study RCs in agile software development projects. Our aim is to help software teams to gain a better understanding of RCs and ultimately to better anticipate, react to, and handle different RCs during development.

First, we conducted a semi-structured interview based study¹ (IBS) to acquire a broad sense of the multi-faceted nature of RCs from an industrial perspective. We interviewed 10 software practitioners (indicated by IP<ID>) from New

Zealand and Australia who had their current or most recent projects driven by agile methods. We then used a Grounded Theory (GT) analysis that identified some key facets (Fn) of agile project RCs – *types* (F1), *forms* (F2), *reasons* (F3), *sources* (F4), *carriers* (F5), and *events* (F6).

In order to understand these potential agile RC facets more thoroughly, we conducted a focused literature review (FLR) on taxonomies of RCs in Software Engineering. We found that none of the existing studies have as yet comprehensively investigated these RC facets. Prior studies [5], [4] resulted in an RC taxonomy for software development based on origination *sources*, categorised as RCs originating from *market*, *organization*, *project vision*, *specification*, and *solution*. Nurmuliani et al. [6] presents *types* of RCs, *reasons* why they originate, and *sources* from which RCs originate as their key findings. They categorised *types* of RCs as additions, deletions, and modifications of requirements. Inpirom and Prompoon classified RCs according to analysis and design of software artefacts [7], but these were not applied to handling RCs during agile software development. Saher et al. [8] describe RCs in terms of *time of change*, *type*, *reason*, and *origin* of RC. However only one study they used actually focused on agile methods.

To discover more about agile project RCs in practice, we then ran a detailed practitioner survey². We obtained results from 40 software practitioners from Asia (26 participants), Oceania (9 participants), North America (3 participants), and Europe (2 participants). We asked them about RCs in their current or most recent project using agile methods. We utilised descriptive statistical analysis to analyse the quantitative data, and GT to analyse the qualitative data from the survey.

The definition of what is an RC varied from participant to participant in the IBS. For some, it was the change made

• K. Madampe, R. Hoda, and J. Grundy are with the HumaniSE Lab at Department of Software Systems and Cybersecurity, Faculty of Information Technology, Monash University, Wellington Road, Clayton, VIC 3800, Australia.
E-mail: kashumi.madampe@monash.edu

Manuscript received December 4, 2020; revised July Date, 2021.

1. Approved by The University of Auckland Human Participants Ethics Committee. Approval Number: 023015

2. Approved by Monash Human Research Ethics Committee. Approval Number: 23578

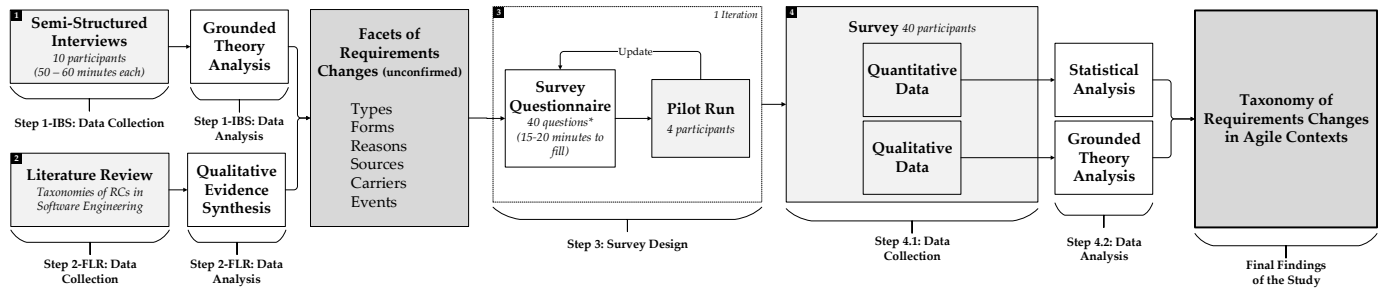


Fig. 1. Mixed-Methods Research Including Semi-Structured Interviews, Focused Literature Review, and In-Depth Survey (IBS: Interview Based Study, FLR: Focused Literature Review, *Included Questions which are Out of Scope of this Paper)

in the scope of the project. For others, it varied from prioritisation of a change in a user story, minor modifications to a current requirement, to major changes, such as user stories moved, changed, deleted, or added. Due to the range of definitions IBS participants had on RCs, we decided to adhere to a single definition in our survey. We derived the following definition of an RC from the IBS and FLR findings and use it in the rest of this paper:

A Requirements Change (RC) is an addition/modification/deletion/bug fix/or combination of these in terms of functional and non-functional requirements presented in any form such as a user story and a use case.

We used this definition at the beginning of our survey questionnaire so that participants' understanding of the term is in correspondence with our understanding. With this definition, we also aim to relate our findings closer to the use of requirements for the development team focus i.e., software changes made, rather than the customer focus, which is mostly on value delivered.

This study falls under "mixed-methods research" category [9] as we used a sequence of different data collection and analysis approaches. In this paper, we primarily focus on results from the practitioner survey, complimented with key findings from the IBS and FLR. The key contributions of this research include:

- 1) Insights from interviews (10 practitioners) and survey (40 practitioners), all experienced agile based software practitioners, on how RCs are handled in real-world agile software projects;
- 2) A taxonomy of agile project RCs. These include RC *types, forms, reasons, sources, carriers, and events*; and
- 3) A set of practical recommendations for software practitioners to better handle diverse RCs in team contexts.

The rest of this paper is structured as follows. Our detailed mixed-methods study approach including research questions is given in Section 2, and the end results of the study are given in Section 3. The key findings are discussed in Section 4 emphasising the avenues for further research, and practical recommendations for software practitioners. Threats to validity of the findings are given in Section 5. Finally, the paper ends by a Summary given in Section 6.

2 STUDY DESIGN

We adopted a mixed-methods approach to conduct this study, outlined in Fig. 1. We conducted an IBS, then an FLR,

TABLE 1
Demographic Data of IBS Participants (XA: Experience in Agile Methods; TX: Total Software Development Experience; XTA: Total Agile Experience; *Some participants played more than one role)

Location	# of Participants	Role*	# of Participants
New Zealand	8	Scrum Master	5
Australia	2	Business Analyst	2
XA	# of Participants	Manager	2
Scrum	10	Tester	1
Kanban	8	Architect	1
XP	5	Senior Consultant	1
Scrum XP combo	4	Head of Global Projects	1
FDD	3		
Spotify	2		
DSD	1		
CBM	1		
XT	# of Years	XTA	# of Years
Minimum	2	Minimum	1
Maximum	56	Maximum	18
Mean	19.75	Mean	7.26

and finally a survey. The details of each of these steps are given in subsequent sub-sections. The instruments for this study, including the pre-interview questionnaire, interview guide, survey questionnaire, and survey data are available online³.

2.1 [Step 1]: Interview-Based Study

We indicate the number of participants by "N" in our studies below. With the objective of understanding the multi-faceted nature of RCs in agile contexts, we conducted 10 semi-structured (N=7 face-to-face, N=3 online) interviews with agile practitioners in New Zealand (N=8) and Australia (N=2). Each interview lasted 50-60 minutes. The demographic information of the participants is given in Table 1. We analysed the collected data from the interviews using GT analysis procedures including open coding and constant comparison described in [10]. This analysis resulted in several categories for the RCs – key facets (F1–F6) being *types, forms, reasons, sources, carriers, and events*. In Table 2 we show some example raw data from the interview excerpts. We used these to refine the respective RC facets, through codes, and concepts.

3. <https://github.com/kashumi-m/ReplicationPackageRCTaxonomy>

TABLE 2
Examples of GT Analysis of IBS Data (SM: Scrum Master; BA: Business Analyst; SSA: Senior Solutions Architect)

Raw Data	Code	Concept	Category
<i>"What they said one week, becomes not what they have decided they absolutely need the next, and there's something completely different."</i> – IP7 [SM]	Completely different requirement	Addition	Type
<i>"You know, drop what they're working on, and start working on this thing straight away, and that's not just for changes where you've encountered a requirement that wasn't right, it's also new requirements coming in or new features or new stories."</i> – IP8 [SSA]	New requirements		
	New features		
	New user stories		
<i>"Requirement change? We do, for our team, we do live in this world of, we don't really know what's wrong sometimes (laughter). It's just a hidden bug, we have no idea."</i> – IP10 [SM, Manager, BA]	Hidden bug	Bug fix	
<i>"Requirement changes are broken into two groups in our organisation, so one are feature request or feature changes, and the others are bugs."</i> – IP2 [SM]	Bugs		
<i>"We usually just have them, and sometimes we use stories, other times it's just an item on the backlog really."</i> – IP7 [SM]	User stories	User stories	Form
<i>"Edge, E-D-G-E, right, edge cases. So whenever there's an edge case you've got to be clear as to what happens exactly on that edge. And then related to that is [product name] and [product name]. Most people, well in English there's no exclusive or, so that's an issue."</i> – IP3 [BA]	Edge cases	Missing requirements	Reason
<i>"Well change can be for many reasons. And as I just said it can be because of you working out better ways of doing things, your customer changing what's most important to them. It's not only what they do but what's most important. And of course there can be external changes which are either possibly because of competitive changes or because of regulatory changes."</i> – IP3 [BA]	Value for customer	Product strategy	
	External changes [competitive changes]		
	External changes [regulatory changes]		
<i>"Most requirement changes that I was involved with, were as the team developed the product, they found the things about the product; they do too."</i> – IP1 [Tester, SM]	Discovering RCs as a team	Technical team discussions	Source
<i>"I think of myself as an end user. What if this product comes to me? How do I feel about it? So it's not like my job only limited to a tester, I can product ideas. Hey I feel like I've seen it, what do you guys think? Hey I feel like checkbox is more appropriated. Because it's something related to rules or regulations. Checkbox sounds good. If it's male or female stuff, probably a radio button. That's how that works. So I could have opinions on agile. We have to be keep thinking. Your mind should be open."</i> – IP9 [Tester]	Suggesting RCs as a team		
<i>"Suppose we are talking about an application which is bus tracking. So you know hop card stuff. Previously you had to go to a machine or somewhere to check the hop card balance. But now it's already showing on you app. Suppose you are launching some app. And you know okay, it's like now you have no function like hop card balance. And suddenly what do you say, your application is on the trial mode, not with, just a demo card with public. And they came out and say, okay, the beta version is out. And we say oh it's nice, can you write the feedback for us. So we say it's really nice to see the hop card balance. So its mass people were like that and they wanted it. In that scenario they wanted that. So it gave them heads up."</i> – IP9 [Tester]	End user feedback	User reviews	
<i>"It can be, it can be anyone."</i> – IP8 [SSA]	Anyone	Anyone	
<i>"This is one of the area, marketing people bring the information. Same way support people also bring the information."</i> – IP5 [SM]	Marketing team	Marketing team	Carrier
	User-support team	User-support team	
<i>"they have their planning meeting"</i> – IP6 [SM]	Planning meeting	Planning meeting	Ceremony/ event
<i>"I think CR's can come in at any point, generally, customers accept to see at any point in the process."</i> – IP2 [SM]	Any point	Any point	

2.2 [Step 2]: Focused Literature Review

Information about the key facets of RCs found in the IBS was inconclusive and contradictory. To clarify and improve our findings, we conducted a Focused Literature Review (FLR) seeking existing categorisations and/or taxonomies of RC in Software Engineering to date. We used the search string “requirements change” AND taxonomy to search in digital libraries. There were only small number of studies focused on construction of an RC taxonomy. We found six studies focusing on categorising RCs, and two studies by McGee and Greer were very similar to each other. Among these six studies, five focused on traditional software development methods and only Saher et al.’s study [8] focused on agile software development. A summary of these studies’ key findings are given below.

McGee and Greer’s studies [4], [5] resulted in an RC taxonomy for software development based on *sources* of RCs, categorised as originating from *market*, *organization*, *project vision*, *specification*, and *solution*. Their study found that higher cost and value changes originate mostly from *organization* and *vision sources*. These *sources* involve cooperation of stakeholder groups with less control than the RCs originating from *specification* and *solution sources*. Additionally, except for RCs originating from market, RCs originating from other *sources* showed a considerable amount of difference in cost, value to the customer, and management considerations. They also found some key triggers and uncertainties for each classified *source*. Furthermore, they claim that using their RC taxonomy will help to manage RCs, understand RCs, and gain risk visibility.

Nurmuliani et al. [6], classified different *types* of RCs, *reasons* why they originate, and various *sources* from which the RCs originate. They categorised *types* of RCs as additions, deletions, and modifications of requirements. They found that defect fixing, missing requirements, functionality enhancement, product strategy, design improvement, scope reduction, redundant functionality, obsolete functionality, erroneous requirements, resolving conflicts, and clarifying requirements are the *reasons* that RCs originate. As *sources* of RCs, they found defect reports, engineering’s calls, project management consideration, marketing group, developers’ detailed analysis, design review feedback, technical team discussion, functional specification review, feature proposal review, and customer-support discussions. In addition to their RC taxonomy based on *type*, *reasons*, and *sources* of RCs, they also provided change request arrival rate and requirements volatility measure. This is the ratio of a certain kind of RC to the total number of RCs over a certain period of time during development.

Harker et al. [11] defined mutable, emergent, consequential, adaptive, and migration as RCs in their classification. They also defined environmental turbulence, stakeholder engagement in requirements elicitation, system and user development, situation action and task variation, and constraints of planned organizational development as the origins of RCs. Inpirom and Prompoon [7] classified RCs according to analysis and design of software artefacts. Their RC taxonomy is based on Unified Modeling Language diagram changes: use case diagrams, class diagrams, and sequence diagrams. They identified that most research has

TABLE 3
Comparison of Our Study with Existing Literature on RC Taxonomies (F1: Types; F2: Forms; F3: Reasons; F4: Sources; F5: Carriers; F6: Events; IonA: Impact on Artefacts; SD: Software Development)

	F1	F2	F3	F4	F5	F6	IonA
Requirements Changes Taxonomies in Traditional SD							
McGee and Greer [4], [5]				☑			
Nurmuliani et al. [6]	☑		☑	☑			
Harker et al. [11]	☑						
Inpirom and Prompoon [7]							☑
Requirements Changes Taxonomies in Agile SD							
Saher et al. [8]	☑		☑	☑			
Our Study	☑	☑	☑	☑	☑	☑	

focused on impact of RCs on source code, but they claim that impact of RCs on design diagrams is also required. Likewise, Basirati et al.’s study [12] shows that RCs impact software artefacts in various ways. They explored the changes in use cases and their further analysis of problematic RC changes resulted in their new taxonomy of RCs. They also found the local and temporal dispersion of RCs as difficult and risky.

The only taxonomy we found for RCs in agile software development was Saher et al.’s [8] literature based study. They described their taxonomy in terms of *time of change*, *type*, *reason*, and *origin* of RC. However, out of the 9 studies they used to build their RC taxonomy, 8 use traditional software development while only one study actually focused on agile software development methods. This highlights a continuing gap in Software Engineering research where a taxonomy of RCs in agile contexts is necessary, where a high number of RCs is expected. In this paper we have tried to fill this gap.

Type, *reason*, and *source* were repeatedly found in these prior studies of RCs, confirming that these three key facets are important when describing RCs. We decided to incorporate the closely related work from these studies with our own IBS findings to construct our candidate proposed agile project RCs taxonomy. We then aimed to confirm this RC taxonomy through our practitioner survey. Comparison of our study with the existing literature is given in Table 3. If any given study included a key facet of RC, we indicate it by a ☑. Even though our study did not focus on the impact of RCs on artefacts, Inpirom and Prompoon’s study [7] covers this (indicated by ☑ in Table 3).

2.3 [Step 3: Survey Development]: Research Questions Formulation and Survey Questionnaire Development

2.3.1 Research Questions Formulation

The key facets of RCs (labelled F1 – F6) we found through our IBS formed the basis of the research questions for our survey. We wanted to determine answers to each of the following key research questions:

- RQ1. What are the different *types* of RCs found in agile projects?** ^[F1] – We wanted to know what different types of RCs developers encounter and which are more or less common.

TABLE 4
Research Questions Formulation and Survey Questionnaire Development (IBS = Interview-Based Study; FLR = Focused Literature Review)

Key Facet	Element	IBS	FLR	RQ	Survey Question	Question Type
F1	Addition	☑	☑	}RQ1	In your experience, which of the following types of requirements changes have you received? <list of elements> + “anything else other than the above?”	Close-ended (Multiple selection with one text entry)
	Modification	☑	☑			
	Deletion	☑	☑			
	Bug Fix	☑	☑			
	Combination F1*					
F2	Epics F2*	☑		}RQ2	In which form are requirements changes most regularly captured on your project? <list of RC types>x <list of elements>	Matrix (Single selection for each element)
	User stories					
	Use cases F2*					
	Tasks F2*					
	Combination of forms F2* Verbally, not documented F2*					
F3	Functional enhancement	☑	☑	}RQ3	A requirements change can occur due to, <list of elements>+ “other”	Close-ended (Multiple selection with one text entry)
	Design improvement	☑	☑			
	Bug	☑	☑			
	Erroneous requirements	☑	☑			
	Redundant functionality	☑	☑			
	Obsolete functionality	☑	☑			
	Missing requirements	☑	☑			
	Requirements clarification	☑	☑			
	Scope reduction	☑	☑			
	Resolving conflicts	☑	☑			
Product strategy	☑	☑				
Need for refactoring F3*						
Outstanding technical debt F3*						
F4	Defect reports	☑	☑	}RQ4	Where to the requirements changes originate and how often? <list of RC types>x <list of elements + [frequency: never, sometimes, always]> + any other person who brings requirements changes originate and how often?	Matrix (Single selection of frequency for each element with one text entry)
	Individual developer’s detailed analysis					
	Product backlog reviews					
	Technical team discussions					
	User reviews					
	Marketing team					
	User-support discussions					
F5	Customer F5*	☑	☑	}RQ5	Who brings requirements changes to you and how often? <list of RC types>x <list of elements + [frequency: never, sometimes, always]> + any other person who brings the requirements changes and how often?	Matrix (Single selection of frequency for each element with one text entry)
	Product owner F5*					
	Agile coach F5*					
	Developer F5*					
	Other agile team member					
	Marketing team User-support team					
F6	During iteration planning	☑		}RQ6	When and how often do you receive requirements changes? <list of RC types>x <list of elements + [frequency: never, sometimes, always]> + any other occasions where you receive requirements changes and how often?	Matrix (Single selection of frequency for each element with one text entry)
	During daily standup F6*					
	During iteration review F6*					
	During iteration retrospective F6*					
	After releasing the complete product					

- RQ2.** In what *forms* are RCs documented? ^[F2] – We wanted to know how agile teams document their RCs and if this documentation takes different forms.
- RQ3.** What are the *reasons* that RCs originate? ^[F3] – We wanted to understand why RCs originate. i.e., the justification or the motivation for RC origination.
- RQ4.** What are the *sources* of RCs? ^[F4] – We wanted to know the human and non-human artefacts which lead to origination of RCs. i.e., where the RCs are obtained from.
- RQ5.** Who are the *carriers* of RCs to the agile development team? – We wanted to know who provides these RCs to the agile team. ^[F5]
- RQ6.** In what *events* do RCs originate? ^[F6] – We wanted to know where the RCs come from.

2.3.2 Survey Questionnaire Development

Table 4 illustrates how each of our research questions and associated survey questionnaire were developed. We consolidated the findings from the IBS with closely related work of McGee and Greer’s [4] and Nurmuliani et al.’s [6] that we found through the FLR to develop the in-depth survey

questionnaire. Certain elements of the key facets were not directly found from the IBS and the FLR. These are indicated in Table 4 with the corresponding key facet and an asterisk. Table 5 provides the rationale for including these elements in the survey.

Each close-ended question used to collect the data presented in this paper included one open-ended “Other” option as well. That allowed the participants to enter any other responses they wanted to include apart from the options we provided for the particular question. We followed Kitchenham et al.’s [14], [15] and Punter et al.’s [16] guidelines to design the survey. We followed Smith et al.’s work [17] on “improving developer participation rates in surveys” to support survey distribution. The distribution was invoked by the authority and credibility of us and our affiliation, and time of distribution. We observed that developers in our social media networks were more active during weekends. Also, our known contacts preferred doing the survey during the weekends. Therefore, we posted, and sent the survey to the participants on Saturday mornings AEDT.

TABLE 5
Rationale for Including Elements that were not found through the IBS
and FLR Studies

	Rationale
F1*	IP4 and IP7 mentioned the RCs without specifying a distinct type. Therefore, we hypothesised that there could be combinations of the RC types we found. Hence, F1* one was included in the survey. Furthermore, IP1 and IP2 stated "functionality/functional changes". Therefore, we further replicated the found elements under functional and non-functional requirements.
F2*	Even though, we did not find these forms through the IBS study, but as these are forms of documenting RCs in practice, we decided to include in the survey.
F3*	Even though the two <i>reasons</i> denoted by F3* are related, there are other drivers for deciding on code refactoring [13] than technical debt. Our interest in investigating whether these <i>reasons</i> on their own cause the origin of RCs led to have this as an option in the survey.
F5*	IP8 stated that "anyone brings RCs". Therefore, we included the known stakeholders as <i>carriers</i> for the participants to select in the survey.
F6*	IP2 mentioned that RCs can be originated "at any point". Therefore, we included the accustomed agile ceremonies in the survey.

2.4 [Step 4.1]: Survey Data Collection

After the initial survey questionnaire was finalised, we sent the survey to 2 Research Fellows and 2 Ph.D. students in Australia and New Zealand who had software development industrial experience. This pilot study enabled us to receive feedback in terms of time for completion and any other aspects such as wording. For example, one of the pilot study participants suggested changing the title of the survey so that the title stays in layman terms. They all recorded the time spent to complete the survey, so that we could get a rough idea of the actual time taken to fill out the survey questionnaire. Following that, we updated the introduction section in the survey so that our participants could get a clearer idea of the time they have to dedicate to fill out the questionnaire. We then distributed the survey via:

- posting the survey link on professional software development groups and in our profiles in social media such as *LinkedIn*, *Twitter*, and *Facebook*;
- sending the survey link to our known contacts in the software development industry; and
- *Agile Alliance* posting the survey link on their *LinkedIn*, *Twitter*, and *Facebook* channels.

The survey was available online for a period of a month. 106 participants started the survey and 42 participants fully completed the survey. Two responses from the completed responses were removed due to their feedback given at the end of the questionnaire mentioned that they completed the survey only to see what the questions were and that their answers were arbitrary.

2.4.1 Demographic Information of Survey Participants

We targeted only agile software practitioners. We confirmed this by the participants' demographics and their self confirmation on practising agile prior to beginning the survey. As given in Table 6, participants of our survey included developers, agile coaches/scrum masters, testers, business analysts, product owners, tech leads, and managers.

Participants had between 1 to 30 years of total experience in the software industry. Their agile experience ranged from

1 - 20 years in Scrum, Kanban, ScrumBan, XP, Scrum XP combo, Crystal, Feature Driven Development, Dynamic System Development, and SAFe. One of the participants (SP13) had not mentioned which agile methods that they had experience in. Another participant (SP28) had selected their age group as 20-25 even though his total experience in software development industry was indicated as 30 years. As we did not collect any contact information of the participants, we were not able to follow up to find the true information in both cases. The majority of the participants (N=26) were from Asia (N(Sri Lanka)=22, N(Singapore)=2, N(India)=2). 9 participants were from Oceania (N(Australia)=6, N(New Zealand)=3), 3 participants were from North America (N(United States of America)=2, N(Colombia)=1), and 2 participants were from Europe (N(United Kingdom)=1, N(Netherlands)=1).

2.4.2 Project Information from Survey Participants

Table 7 shows the project information from the survey participants. The majority of our participants had their current or most recent projects under the general Information Technology domain (N=23); categorised as new development (N=22); and used Scrum (N=28). The mean team size was 14 members, where the mean iteration length was 3.28 weeks. The participants followed the agile practices predominantly. But practising customer demos and pair-programming counted less in comparison with other agile practices. Practising collective estimations, review meetings, and retrospectives followed a similar pattern where the number of responses was the option "sometimes" were closer to the options "most of the time", and "always".

2.5 [Step 4.2]: Survey Data Analysis

Quantitative data from the survey was descriptively analysed using *Qualtrics* and *Microsoft Excel*. Qualitative data was analysed using *Microsoft Excel*. Similar to the IBS, qualitative data followed GT analysis as described in [10], where concepts and categories were generated through constant comparison.

3 SURVEY FINDINGS

3.1 What are the different types of requirements changes found in agile projects? (RQ1)

Participants were able to choose multiple *types*. The rest of the survey questions depended on the choices in this question, as they only had to answer the other questions regarding RC *types* that they selected in this question. We provided the acronyms, definitions, and examples (see Table 8) for the participants to use as a guide to answer this question. Additionally, we assumed that *deletion* and *combination* were understandable terms.

Table 8 shows the different *types* of RCs reported. The top most received RC *type* as reported by the participants was *FR addition* (N=33). It was followed by *FR bug fix* (N=29), and *FR modification* (N=28). Same amount of participants (N=25) reported that they received *FR deletion* and *NFR modification*. 23 out of the 40 participants reported that they receive *NFR additions* as RCs. *FR combination* (N=16), *NFR bug fix* (N=15), *NFR deletion* (N=12), *FR-NFR combination* (N=12), and *NFR*

TABLE 6

Demographics of the Survey Participants (P#: Participant ID; XT: Total Experience in Software Development Industry in Years; XTA: Total Experience in Agile in Years; XA: Experience with Agile Software Development Methods; SL: Sri Lanka; Au: Australia; USA: United States of America; CO: Colombia; NZ: New Zealand; SG: Singapore; UK: United Kingdom; NL: Netherlands; In: India)

P#	Age Group	Gender	Country	XT	XTA	XA	Role in the Project
SP1	26 - 30	Male	SL	2.5	2	Scrum, Kanban, ScrumBan	Developer
SP2	20 - 25	Male	SL	2.5	2.5	Scrum	Developer
SP3	26 - 30	Female	SL	2	2	Scrum	Agile Coach/Scrum Master, Developer
SP4	26 - 30	Male	SL	3.5	3.5	Scrum, Kanban	Developer
SP5	26 - 30	Female	SL	2	1	Scrum	Tester
SP6	26 - 30	Female	SL	2	2	Scrum	Tester
SP7	26 - 30	Male	SL	3	1	Scrum	Developer
SP8	20 - 25	Female	AU	2	1	Scrum, XP	Developer
SP9	20 - 25	Female	SL	2.5	2.5	Scrum, Feature Driven Development	Tester
SP10	20 - 25	Male	SL	5	8	Scrum, XP, Scrum XP combo, Kanban, Crystal, Feature Driven Development, Dynamic System Development	Agile Coach/Scrum Master
SP11	26 - 30	Female	SL	3.5	2.5	Scrum	Tester
SP12	26 - 30	Female	SL	2.5	2.5	Scrum	Business Analyst
SP13	41- 45	Male	USA	20	10	Not specified	Developer
SP14	31 - 35	Female	CO	9	3.5	Scrum	Business Analyst
SP15	31 - 35	Male	SL	6	1	Scrum, Kanban	Developer
SP16	31 - 35	Male	SL	10	10	Scrum	Agile Coach/Scrum Master, Product Owner, Developer, Tech Lead
SP17	26 - 30	Male	AU	4	2	Scrum	Developer
SP18	26 - 30	Female	SL	2.5	2.5	Scrum	Agile Coach/Scrum Master
SP19	31 - 35	Male	NZ	10	7	Scrum, XP, Kanban	Agile Coach/Scrum Master
SP20	46 - 50	Male	AU	16	5	Scrum	Product Owner
SP21	26 - 30	Female	SL	2	2	Scrum	Tester
SP22	26 - 30	Female	AU	3	2	Scrum, Kanban	Developer
SP23	36 - 40	Male	SL	12	8	Kanban	Developer
SP24	26 - 30	Male	SL	5	2.5	Scrum, Kanban, ScrumBan	Tester
SP25	Above 50	Male	AU	30	15	Feature Driven Development, Dynamic System Development, Primarily the Values & Principles of The Agile Manifesto	Manager
SP26	26 - 30	Female	NZ	1	1	Scrum	Tester
SP27	31 - 35	Male	SL	8	1	Scrum, Kanban	Developer
SP28	20 - 25	Male	AU	30	20	Scrum, XP, Scrum XP combo, Kanban	Product Owner, Manager
SP29	26 - 30	Female	SG	2	2	Scrum	Developer
SP30	26 - 30	Male	SG	2	1	Crystal	Developer
SP31	31 - 35	Male	UK	3	3	Scrum, Kanban, SAFe	Business Analyst
SP32	41- 45	Male	USA	23	15	Scrum, XP, Kanban	Developer, Manager
SP33	36 - 40	Male	NL	10	5	Scrum	Product Owner
SP34	31 - 35	Female	SL	9	5	Scrum, Kanban	Agile Coach/Scrum Master
SP35	20 - 25	Female	IN	3.5	3	Scrum, XP, Kanban, Feature Driven Development	Developer
SP36	31 - 35	Male	SL	9	4	Scrum	Tester
SP37	41- 45	Female	NZ	25	5	Scrum, Kanban	Product Owner
SP38	31 - 35	Male	SL	8.5	4	Scrum, Kanban, Feature Driven Development	Agile Coach/Scrum Master
SP39	31 - 35	Female	IN	13	2	Scrum	Agile Coach/Scrum Master, Product Owner
SP40	26 - 30	Male	SL	4	4	Scrum	Developer

combination (N=7) were also received by the participants as RCs. In addition, participant SP12 [Business Analyst] reported that she had experienced a *type* of an RC called “transition requirements addition”, which we considered as either a functional or a non-functional RC.

FR changes were the most commonly received RCs *type* in agile contexts and included FR additions, bug fixes, modifications, and deletions (N(Total FRs)=131). A substantial number of RCs were also NFR modifications and additions (N(NFRs)=82).

3.2 In what forms are requirements changes documented? (RQ2)

Even though agile does not encourage the use of detailed documentation [2], RCs must still be documented in some way for ease of use by the teams [18], [19]. RCs are documented in several *forms* such as *epics*, *user stories*, *use cases*,

and *tasks*. Along with these *forms*, we provided *combination of forms*, and *verbally, not documented* for the participants to choose. As shown in Fig. 2, participants chose the *forms* they used the most commonly for each type of RC. Below, we summarise results for each documentation *form* reported by survey participants.

Epics: Epics are groups of related user stories and are very widely used in agile contexts.

User Stories: Changed, new or deleted user stories are the most common *form* used to document almost all *types* of RCs, excepting for FR bug fixes, NFR additions, and NFR bug fixes.

Tasks: Tasks (fractions of work) are most common *form* used to document FR Bug fixes, NFR additions, and NFR bug fix RC *types*.

Use Cases: Use cases are widely used in more traditional software development methods but are also still in use in many agile contexts. They are a much less common

TABLE 7
Project Information of Survey Participants

Project Domain	# of Participants	Project Category	# of Participants	Agile Method Used	# of Participants
IT	23	New Development	22	Scrum	28
Healthcare	4	Software as a Service	9	Kanban	4
Manufacturing	3	Migration	3	ScrumBan	2
Finance & Banking	2	New Development & Migration	3	Feature Driven Development	2
HRM	2	Maintenance	1	Crystal	1
Defence	1	Maintenance with Continuously Adding New Features	1	Dynamic System Development	1
ERP	1	All	1	SAFe	1
Food Services	1			Scrum, XP, DevOps, Kanban	1
Telecom	1	Team Size	# of People	Iteration Length	# of Weeks
Tourism	1	Minimum	3	Minimum	2
Transport	1	Maximum	80	Maximum	10
		Mean	14.36	Mean	3.28
		Standard Deviation	15.54	Standard Deviation	2.25

Practices Followed (Order of the Bars in Each Graph Below: Never → Sometimes → About half the time → Most of the Time → Always)					
Short iterations/Sprints	█	Iteration Planning	█	User Stories	█
Product Backlog	█	Sprint Backlog	█	Collective Estimation	█
Daily Standup/Team Meeting	█	Release Planning	█	Pair Programming	█
Self-assignment	█	Customer Demos	█	Review Meetings	█
Scrum/Kanban Board	█	Definition of Done	█	Retrospectives	█

TABLE 8
Requirements Change Types

Acronym	Definiton
FR:	Functional Requirement
NFR:	Non-Functional Requirement
Bug Fix:	Correction in the codebase
Addition:	A new requirement arising due to a change in an existing requirement
Modification:	Modifying an actual requirement. E.g., Modify the actual user story/ split the user story/ change user story partially
Combination:	Combination of given types

Requirements Change Type	# of Responses
FR Addition	33
FR Bug Fix	29
FR Modification	28
FR Deletion	25
NFR Modification	25
NFR Addition	23
FR Combination	16
NFR Bug Fix	15
NFR Deletion	12
FR-NFR Combination	12
NFR Combination	7

form used to document RCs. They are sometimes used to document FR addition, FR modification, NFR addition, NFR modification, NFR deletion and FR-NFR combination RC types.

Verbally, Not Documented: Surprisingly, the RC types FR modifications, FR deletions, FR bug fixes, FR combinations, NFR modifications, and NFR bug fixes are verbally provided to the teams but were reported as often not formally documented. Not documenting such RCs may lead to serious concerns in project schedule, cost and impact the

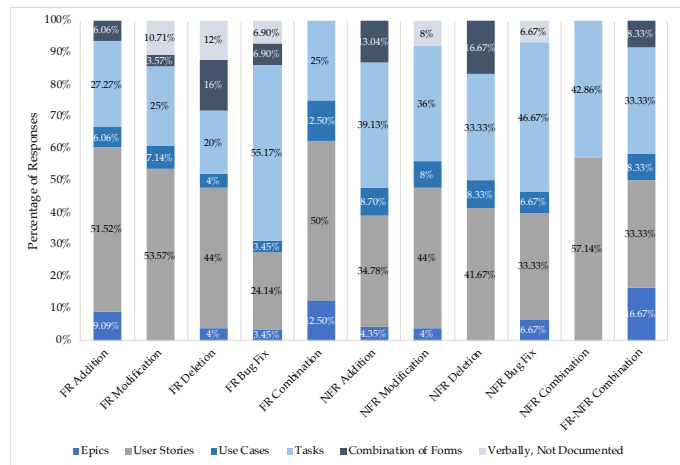


Fig. 2. Documentation Forms of Requirements Changes














quality of the software.

Combination of forms: Combination of forms was selected by the participants for a small number of RC types, including FR deletion, FR bug fixes, NFR deletion and FR-NFR combination.

3.3 What are the reasons that requirements changes originate? (RQ3)

We categorised the key reasons found through our IBS and FLR as software-centric and human-centric reasons, as shown in Table 9. By software-centric we mean the reasons that are directly linked to the software being developed. By human-centric we mean the reasons that are caused due to the various development and team and customer management approaches taken by the humans involved in the project.

TABLE 9
Categories of Reasons for Requirements Changes Origination

Reason	# of Responses
Functional enhancement 	32
Design improvement 	28
Bug  (Error arising in the codebase)	21
Erroneous requirements 	17
Redundant functionality 	9
Obsolete functionality 	9
Missing requirements 	27
Requirements clarification 	25
Scope reduction 	19
Resolving conflicts 	16
Product strategy 	15
Need for refactoring 	11
Outstanding technical debt 	10

As shown in Table 9, most participants (N=32; 80%) reported that the key *reason* for most RC origination is functional enhancement. Design improvement (N=28; 70%), missing requirements (N=27; 68%), requirements clarification (N=25; 63%), and bug (N=21; 53%) were chosen as the common *reasons* for RCs to occur by more than half of the participants. Scope reduction (N=19; 48%), erroneous requirements (N=17; 43%), resolving conflicts (N=16; 40%), product strategy (N=15; 38%), need for refactoring (N=11; 28%), and outstanding technical debt (N=10; 25%) were selected by 25% or more of the participants. Obsolete functionality and redundant functionality were selected (N=9; 23%) by participants as the least common *reasons* for origination of RCs.

Considering the responses which are above half of the sample size (>N=20), 3 out of 5 *reasons* (functional enhancement, design improvement, and bug) for RC occurrence we term software-centric. The other 2 *reasons* (missing requirements, and requirements clarification,) we term human-centric. Taking the rest of the responses (<N=20) into consideration, 5 out of 8 *reasons* (scope reduction, resolving conflicts, product strategy, need for refactoring, and outstanding technical debt) we term human-centric. The other 3 *reasons* (erroneous requirements, obsolete functionality, and redundant functionality) we term software-centric.

Additionally, certain opinions of the IBS participants had may be caused by “missing requirements”. Missing requirements can be due to:

Vagueness in the requirements: “And the requirement changes, like we have to, sometimes we actually have very bad requirements, it’s very vague, it’s just identified what’s happening, fix, test, and deploy I guess (laughter).” – IP10 [Scrum Master, Manager, Business Analyst]

and Omissions from requirements: “those are the sort of trivial, tricky little omissions from requirements.” – IP3 [Business Analyst]

We also found the following additional *reasons* reported for the origination of RCs in our open-ended responses of the survey:

Inadequate communication: “Individuals and interactions over processes and tools” is a core value in agile [2]. In order to have better interactions, high quality team and customer

communication is needed. Communication has been shown in many studies to be a key construct for better software development [20], [21]. However, if communication among the stakeholders is inadequate, origination of RCs can be expected as reported by the participants. In this case, it is required for the team to be mindful and overcome this issue so that unexpected RCs do not originate. Quoting an IBS participant who encountered similar experiences: “Change happens all the time unfortunately, and there’ll always be some who neglected to tell you some information” – IP7 [Scrum Master]

Inadequate documentation: Even though agile encourages “working software over comprehensive documentation” [2], as reported by our survey participants, inadequate documentation is a *reason* for many RC origination. Documenting the requirements to a sufficient level is therefore recommended. IP3 had a similar experience where omissions in requirements occurred due to inadequate documentation. Thus, leading to origination of RCs: “there’s information that needs to be exchanged and it wasn’t written down to start with. So you might refer to that as omissions in the requirements.” – IP3 [Business Analyst]

This is also inline with the previously mentioned reason “missing requirements”.

Rushed analysis when defining requirements: Even though the root causes are unknown for rushing the analysis of requirements, it causes the RCs to originate later: “The people defining what the software should do have either rushed their analysis...” - SP25 [Manager]

Wrong set of initial requirements: Having the wrong set of initial requirements unsurprisingly also causes RCs. RCs are necessary to redirect the software development in the correct direction so as to meet the actual customer requirements: “If the requirements change, that means the previous set of requirements was wrong, and who wants to build the wrong thing?” - SP32 [Developer/Manager]

3.4 What are the sources of requirements changes? (RQ4)

We define the human and non-human artefacts which lead to the origination of RCs as the RC *sources*. We found that RCs originate from *defect reports, individual developer’s detailed analysis, marketing team, product backlog review, technical team discussion, user reviews, and from user-support discussions*. The *sources* of each type of RC are shown in Table 10. We categorised these *sources* as *in-team software-centric* (●🖥️), *in-team human-centric* (●👤), *out-team software-centric* (○🖥️), and *out-team human-centric* (○👤). We define *in-team* to mean from the agile team, and we define *out-team* to mean from stakeholders outside the agile team. We define *software-centric* to mean the non-human artefacts directly linked to the software being developed. We define *human-centric* to mean the approaches taken in terms of activities by the humans involved in the software development process. Below we discuss the most commonly found RC *types* (cases where the number of responses were highest for the options: “average” and/or “more than average”) for each *source* category.

In-team Software-centric Sources: The most commonly found RC *types* from in-team software-centric *sources* were: Defect reports: **FR bug fix**, **NFR bug fix**, **NFR combination**; and Individual developer’s detailed analysis: **FR bug fix**,

TABLE 10
Sources of Requirements Changes (<Average Average >Average ; FR: Functional Requirement; NFR: Non-Functional Requirement)

	Defect Reports	Individual Developer's Detailed Analysis	Product Backlog Reviews	Technical Team Discussions	User Reviews	Marketing Team	User-Support Discussions
Functional Requirements Changes							
FR Addition	48.48% 42.42% 9.06%	42.42% 46.46% 4.04%	24.24% 54.55% 30.31%	24.24% 54.55% 21.21%	24.24% 54.55% 21.21%	51.52% 21.21% 27.27%	33.33% 33.33% 33.33%
FR Modification	50%	39.39% 10.10%	28.57% 51.52% 17.86%	28.57% 35.71% 35.71%	21.43% 57.14% 21.43%	51.52% 16.29% 32.16%	28.57% 46.43% 25%
FR Deletion	56%	4%	32%	32%	50%	46%	44%
FR Bug Fix	26.69% 44.83% 34.49%	17.24% 55.17% 27.59%	37.93% 44.83% 17.24%	27.59% 51.72% 20.69%	34.49% 44.83% 20.69%	38.42% 34.49%	37.93% 41.38% 20.69%
FR Combination	50%	43.75%	31.25%	37.50%	43.75%	43.75%	31.25%
Non-Functional Requirements Changes							
NFR Addition	45.22% 17.39% 37.83%	47.81% 52.17% 0%	39.13% 47.81% 13.06%	21.74% 48.87% 29.39%	34.70% 39.13% 26.16%	45.22% 38.42% 41.41%	43.48% 34.70% 21.74%
NFR Modification	52%	4%	36%	16%	28%	52%	32%
NFR Deletion	58.33%	50%	16.67%	25%	41.67%	41.67%	25%
NFR Bug Fix	20%	33.33%	26.67%	33.33%	33.33%	46.67%	20%
NFR Combination	42.86%	57.14%	42.86%	34.29%	42.86%	28.57%	28.57%
FR-NFR Combination	50%	41.67%	8.33%	33.33%	41.67%	5%	41.67%

TABLE 11
Carriers of Requirements Changes (<Average Average >Average ; FR: Functional Requirement; NFR: Non-Functional Requirement)

	Customer	Product Owner	Agile Coach	Developer	Other Agile Team Member	Marketing Team	User-Support Team
Functional Requirements Changes							
FR Addition	21.21% 38.39% 48.48%	15.15% 48.48% 36.36%	63.64%	48.48%	57.58%	63.64%	57.58%
FR Modification	21.43% 35.71% 42.86%	71.43% 53.57% 39.29%	71.43%	53.57%	64.29%	60.71%	57.14%
FR Deletion	40%	24%	64%	52%	56%	60%	72%
FR Bug Fix	27.59% 48.28% 24.14%	34.48% 41.38% 24.14%	58.62%	37.93%	44.83%	58.62%	44.83%
FR Combination	37.50%	31.25%	68.75%	50%	56.25%	62.50%	62.50%
Non-Functional Requirements Changes							
NFR Addition	38.43% 38.43% 39.13%	17.39% 56.52% 26.09%	56.52%	36.52%	65.22%	69.57%	68.87%
NFR Modification	24%	20%	60%	56%	56%	68%	52%
NFR Deletion	25%	16.67%	58.33%	52%	50%	66.67%	50%
NFR Bug Fix	6.67% 53.33% 40%	40%	66.67%	46.67%	46.67%	60%	33.33%
NFR Combination	28.57%	42.86%	57.14%	71.43%	42.86%	71.43%	71.43%
FR-NFR Combination	33.33%	50%	66.67%	50%	75%	66.67%	66.67%

NFR bug fix, FR combination, FR-NFR combination This suggests that bug fixes commonly originate from both defect reports and individual developer’s detailed analysis.

In-team Human-centric Sources: Product backlog reviews and technical team discussions are key sources for all RC types. This shows that for practices where team members work collectively, all RC types result.

Out-team Software-centric Sources: As reported by our participants, user reviews are a key source for all RC types except for FR deletion, FR combination, and NFR bug fix (same number of responses). As half of the participants claimed that FR deletions did not originate from user reviews, this is a possible indication of users being less focused on having FRs removed from the software.

Out-team Human-centric Sources: The marketing team is a key source for NFR combination RC type, and user-support discussions a key source for FR modification, FR deletion, FR bug fix, FR combination, NFR deletion, and FR-NFR combination RC types. Similar to in-team human-centric sources, user-support discussions also originate the majority of the RC types. This suggests that collective human actions are sources for more RCs.

3.5 Who are the carriers of requirements changes to the agile development team? (RQ5)

We define the carriers of RCs as the person or people the different RC types are brought to the agile team by. These are shown in Table 11. Similar to previous sections, below

we report the most common RC types carried by the RC carriers.

Customer: Participants reported that all RC types except FR-NFR combination are carried to the team by the customer. The most commonly carried RC type by the customer is NFR bug fix (average=53.33% and more than average=40%). This suggests that customers are more concerned about fixing bugs related to NFRs than other RC types.

Product Owner: Product owner acts as the carrier of RCs for all RC types except FR-NFR combination. The same number of responses were found for the options “less than average”, and “average” for NFR bug fix (40%), and NFR combination (42.86%). Among the RC types Product Owner carries, FR modification take the most prominent place (average=53.57% and more than average=39.29%).

Agile Coach: Perhaps unsurprisingly, the majority of the participants claimed that RC types are not brought to the team by agile coaches. This suggests that agile coaches do have responsibilities related to RCs such as carrying them to the team, but rather focus on their core task of guiding the team in terms of practising agile methods. However, one wonders whether coaches should highlight RCs they recognise the team should be addressing.

Developer: FR bug fix is the only RC type which our participants mentioned as carried by the developer to the team. The same number of responses were found for the options of “less than average”, and “average” for NFR bug fix (46.67%), and FR-NFR combination (50%). This aligns

with the finding on bug fixes and FR-NFR combinations being abundant in individual developer's analysis (Section 3.4).

Other Agile Team Member: NFR combination is the only RC *type* that is commonly *carried* to the team by other agile team members, as reported by our participants. This suggests that team members in the agile team, other than developers, are concerned more about NFR combinations whereas, developers are concerned more about bug fixes.

Marketing Team and User-Support Team: Our survey findings show that RC *types* are not *carried* to the team by these two particular parties. However, as stated in Section 3.4, the marketing team may themselves produce NFR combinations even though they do not *carry* the RCs to the team. In the case of user-support team acting as the *carrier* of an RC, the same number of responses were found for all three options "less than average", "average", and "more than average" for NFR bug fix (33.33% each). It is possible that the user-support team could act as the *carrier* of NFR bug fixes, as our participants mentioned that user support discussions result in NFR bug fixes (Section 3.6).

Apart from the *carrier* options we provided for the participants to choose in our survey, *carriers* who fall under the category "other agile team member" as reported by the participants included:

Security team: In some cases, a security team exists in the project and they have the ability to identify NFR changes and *carry* these to the agile team. This hints that security teams are subject matter experts in NFRs whereas customer/product owner brings other RCs: "Our security team most often identifies NFR changes." - SP37 [Product Owner]

Business Analysts, Delivery Leads, Development Manager: These three roles are connected to interaction with external stakeholders such as customers and users. Therefore, it can be said that, due to their engagement with external stakeholders, they may *carry* RCs in different circumstances to the team: "Other BAs, Delivery Leads & Development Manager (especially on critical requirements)." - SP25 [Manager]

3.6 In what events do requirements changes originate? (RQ6)

Agile "ceremonies" are development process events that formalise ways a team goes about performing different aspects of agile software development. We summarise the most common agile development process *events* that originate RC *types*. Results are summarised in Table 12.

During Iteration Planning: FR addition, FR modification, FR bug fix, NFR addition, NFR bug fix, FR-NFR combination are the most common RC *types* during this *event*. This indicates that RC additions and bug fixes of both FRs and NFRs are common during iteration planning. This could be due to the software delivery made in the previous iteration. In terms of bug fixes, this leads to the very interesting question: "does delivering in iterations produce more bug prone software?" The same number of responses were found for the options less than average, and average for the RC *types*: FR combination (43.75%), NFR addition (39.13%), and NFR modification (44%). We thus cannot infer whether these RC *types* originate during iteration planning or not.

During Daily Standup: FR bug fix and NFR bug fix are the most common RC *types* produced during this *event*. This

is an indication of software teams giving their formalised stand-up discussions priority to bug fixes over other RCs. Aligning with this, Stray et al. [22] emphasise that discussing and solving problems is one of the process characteristics of a daily standup. In addition, as the same amount of responses (42.86%) were found for less than average and average options for NFR combination, we cannot tell if the origination of NFR combination RC *type* is from daily standups.

During Iteration Review: All RC *types* except for NFR combination are RC *types* produced during this *event*. An iteration review is where the customer and the team discuss the delivery of the completed iteration. It is a key *event* for RCs to originate. All *types* of RCs are possible to originate here.

During Iteration Retrospective: None of the RC *types* are commonly produced in this *event*. Iteration retrospectives are mainly for the team to discuss their experience of the last completed iteration. It is often more about self-reflective team improvements. Our analysis suggests that teams do not seem to generate RCs in these retrospectives.

After Releasing the Complete Product: FR addition, NFR addition, NFR modification, NFR deletion are common RC *types* produced during this *event*. After releasing a completed, the main functional RC that is common is FR addition. Hence, the interest of the stakeholders is primarily in adding new functionalities to their software. As the majority of the RCs originating after delivery as reported by the participants were NFRs, this leaves us with the question whether NFRs are given less focus until complete delivery?

A few other *events* where RCs originate were also reported by participants.

Quarterly Planning Meetings: When the duration of the entire product development is long, such planning is required. In this case, some origination of RCs can be expected.

While Coding: It is common to see customers present if the team is onsite. In this case, the customer may directly provide RCs during development, as reported by participant SP28. Presence of a subject matter expert is possible on-site or off-site. However, how the presence of users at the working premises provides RCs was not clear. This also suggests that a free form of communicating RCs is done by various stakeholders: "The most common place is user/customer/SME directly working with the development team whilst coding the requirement." - SP28 [Product Owner/Manager]

Workshops and Testing: When workshop-based verbal communication and interactions are prominent, there is a high chance for RCs to originate. As testers have high attention to detail when writing test cases, testing is also a place likely for RCs to originate: "BA's "unboxing" (workshopping) new User Stories with Dev Team; Testers trying to build test cases." - SP24 [Tester]

Customer Demos: Usually customer demos are expected to occur during iteration reviews. However, our findings suggest that separate sessions for customer demos exist and RCs can originate during these customer demos.

TABLE 12

Events Where Requirements Changes Originate (<Average Average >Average; FR: Functional Requirement; NFR: Non-Functional Requirement)


	During Iteration Planning	During Daily Standup	During Iteration Review	During Iteration Retrospective	After Releasing the Complete Product
Functional Requirements Changes					
FR Addition	24.24% 48.48% 27.27%	63.64%	30.30% 6.06%	33.33% 45.45% 21.21%	45.45% 33.33% 21.21%
FR Modification	32.14% 35.71% 32.14%	64.29%	21.43% 14.29%	57.14% 32.14% 10.71%	46.43% 32.14% 21.43%
FR Deletion	40% 36% 24%	48%	36% 16%	28% 64% 8%	60% 28% 12%
FR Bug Fix	37.93% 44.83% 17.24%	34.48%	44.83% 20.69%	24.14% 55.17% 20.69%	55.17% 31.03% 13.79%
FR Combination	43.75% 43.75% 12.50%	50%	37.50% 12.50%	37.50% 50% 12.50%	50% 31.25% 18.75%
Non-Functional Requirements Changes					
NFR Addition	39.13% 39.13% 21.74%	56.52%	34.76% 4.76%	30.43% 52.17% 17.39%	65.22% 26.09% 8.70%
NFR Modification	44% 44% 32%	52%	32% 16%	36% 48% 16%	56% 32% 12%
NFR Deletion	41.67% 50% 8.33%	41.67%	33.33% 25%	33.33% 66.67%	66.67% 33.33%
NFR Bug Fix	26.67% 60% 13.33%	26.67%	53.33% 20%	20% 60% 20%	46.67% 40% 13.33%
NFR Combination	71.43% 14.29% 14.29%	42.86%	42.86% 14.29%	57.14% 42.86%	85.71% 14.29%
FR-NFR Combination	41.67% 50% 8.33%	41.67%	33.33% 25%	25% 66.67% 8.33%	58.33% 33.33% 8.33%

4 DISCUSSION

We discuss a new taxonomy of requirements changes in agile contexts that we developed from this study. We also discuss some key findings from our mixed-methods study and promising further areas for research.

4.1 Taxonomy of Agile Software Project Requirements Changes

Using information from our IBS, FLR, and survey, we developed a comprehensive taxonomy of agile software project RCs, shown in Fig. 3. The overview of the taxonomy, its facets, and relationships between them are illustrated in Fig. 4. The aim of this RC taxonomy is to assist software practitioners to: identify the respective *sources*, *carriers* and *events* of each and every *type* of RC in their project; gain a sense of common *forms* which RCs are documented in general and to use these forms at appropriate times during their agile software development practices; and better understand the key underlying *reasons* why RCs originate in their projects and to act accordingly to action these RCs. Based on our interviews, literature review, and survey responses, we believe this will help teams improve their understanding of agile software project RCs and assist with better managing RCs as they occur.

Some elements for different facets that we initially added were found less common after the analysis of our survey data. For example, participants seldom mentioned that they receive RCs *types* via an agile coach, marketing team, or through user-support team. These elements are shown in gray text in the taxonomy. However, some new elements were found during the survey (indicated by  in Fig. 3) and we incorporated these into the taxonomy. We encourage further research to study their usefulness in this RC taxonomy, and to find other elements that we have not yet captured.

Consider a self-organising agile team. A new developer, Kash joins the team and she is assigned to an ongoing development project. Kash wants to know about the current work the team is doing. Kash asks about this from her peers. Kash’s peers tell her that they receive a lot of RCs; but they do not have time to explain the multi-faceted nature of their RCs to Kash. Kash then goes through the product

backlog, the current sprint backlog, and communication the team has had. Kash is still lost. To address such issues, using our taxonomy the team members define what *types* of RCs they have received so far, in what *forms* they are documented, what are the main *reasons* of RC origination, what are their *sources*, who are the *carriers*, and *events* where they originated mostly. They find that the taxonomy has the capability of predicting the future of multi-faceted nature of RCs they will have. By going through this classification, the team finds key *reasons* that could have been addressed proactively to mitigate some unnecessary RCs and manage some unexpected RCs. Using the taxonomy as a framework, the team discusses their current approach to RCs in their next retrospective meeting. The team can see that they are not documenting RCs with sufficient details. The team finds that inadequate communication has also led to some unexpected RCs. The team identifies some unexpected sources and carriers of their RCs that need to be better identified and supported. They review their key agile events where most of their RCs originate to provide more proactive planning for their RCs. These discussions guided by the taxonomy result in some action points that the team decides to work on to improve their RCs management.

4.2 Key Findings and Recommendations

The majority of agile project RCs are functional RCs: The root causes likely include: All stakeholders are focused on functional requirements and/or; non-functional requirements are implemented well and do not require to be changed and/or; non-functional RCs are less common and/or; less focus is given to non-functional requirements. Exploring this further is worthwhile to confirm these four-fold root causes. Taking both functional and non-functional RCs into consideration, the majority are additions in agile projects. This is in contrast to traditional software development methods, as reported in our FLR, and agile projects seem to have many more RC additions. This suggests that stakeholders are primarily interested in adding new requirements during agile iterations and deliveries. It also leaves us with the question as to whether agile software project stakeholders are more interested in adding new require-

Taxonomy of Requirements Changes in Agile Contexts

- Types of Requirements Changes**
 - Functional/Non-Functional Requirements
 - Addition
 - Modification
 - Deletion
 - Bug Fix
 - Combination
 - Functional-Non-functional Requirements Combo
- Forms of RC Documentation**
 - Epics
 - User stories
 - Tasks
 - Use cases
 - Verbally not documented ☞
 - Combination of forms
- Reasons for RC Origination**
 - Functional enhancement ☞
 - Design improvement ☞
 - Bug ☞
 - Erroneous requirements ☞
 - Redundant functionality ☞
 - Wrong set of initial requirements ☞ ☞
 - Missing requirements ☞
 - Requirements clarification ☞
 - Scope reduction ☞
 - Resolving conflicts ☞
 - Product strategy ☞
 - Need for refactoring ☞
 - Outstanding technical debt ☞
 - Inadequate communication ☞ ☞
 - Inadequate documentation ☞ ☞
 - Rushed analysis when defining requirements ☞ ☞
- Sources of Requirements Changes**
 - Defect reports ●☞
 - Individual developer's detailed analysis ●☞
 - Product backlog reviews ●●
 - Technical team discussions ●●
 - User reviews ○☞
 - Marketing team ○●
 - User-support discussions ○●
- Carriers of Requirements Changes**
 - Customer
 - Product Owner
 - Developer
 - (Security Team BA Del. Lead Dev. Mgr.) ☞
 - Agile Coach
 - Marketing Team
 - User-support Team
- Events where RCs Originate**
 - During iteration planning
 - During daily standup
 - During iteration review
 - After releasing the complete product
 - Quarterly planning meetings ☞
 - (While coding Workshops Customer demos) ☞
 - During iteration retrospective

Fig. 3. Taxonomy of Requirements Changes in Agile Contexts (☞:Software-centric; ☞: Finding through survey. May require further investigation; ●: Human-centric; ●: In-team; ○: Out-team; BA: Business Analyst; Del. Lead: Delivery Lead; Dev. Mgr.: Development Manager)

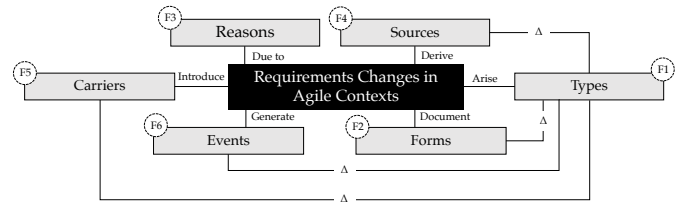


Fig. 4. Facets of Requirements Changes in Agile Contexts and Their Relationships (Δ: Variate)

ments rather than improving the existing requirements. We encourage researchers to investigate this in the future.

Even though bug fixes are not considered as a type of RC, as reported by some of our survey participants, bug fixes are prominent in agile contexts according to our findings: For example, “I do not think bug fix can be a requirement change. Requirement is a user need/problem and design depicts the solution suggested to solve the problem. A bug fix is needed to correct a design not a requirement.” - SP12 [Business Analyst]. If more and more bug fixes occur in every iteration, a team would not be able to deliver the product in a timely manner. This may also increase project technical debt. “Depending on the scenario often there’s actually a lot of technical debt, for example, very common to have technical debt inside an organisation which the team is attempting to address simultaneously while delivering a product. And sometimes only the product owner will know that, that it’s important enough to stop delivering something to the customer in order to address some of that debt. Because it will then give them a platform to springboard off, and produce things more faster, more quickly.” – IP6 [Scrum Master] In future, both aspects are worth investigating: whether bug fixes should be considered as a type of RC; and the relationship RCs have with addressing technical debt.

Our study did not find the reasons for the origination of bug fixes in particular. Knowing the root causes of bug fixes and remedying them accordingly could help to minimize the amount of bug fixes needed in agile iterations. On the other hand, predictive models such as Krishna et al.’s predictive model [23] which forecasts the number of bug reports to be expected given a list of issues could be used to be proactive and be ready to handle any future bugs.

RCs in agile are mostly documented in the form of user stories and tasks: This partially agrees with Wang et al.’s findings on user stories and use cases being the mostly used form of requirements representation in agile [24]. In addition, we found that a number of RCs are verbally communicated, but not documented, and different forms of documentation seem to suit different types, sources and carriers of RCs to the agile team. We suggest researchers study this to produce a mapping of most appropriate forms of documentation for the RC against type, source, and carrier of RCs.

The majority of reasons for the origination of RCs were software-centric: This suggests that stakeholders are interested in enhancing the functionalities of the software more than any other reason. Even though the majority of the reasons are software-centric, human-centric issues such as missing requirements, requirements clarification, and conflict resolvent also exist in a considerable amount. As

it is the customer who brings the RCs to the team the most, according to our findings, these human-centric issues as *reasons* depict that the level of engagement between customer and the team is probably low, even though the customers' attempt to be involved in the process is high. This impacts the project quality as better commitment from stakeholders is required to decrease project failure risks [25]. Corroborating this, Hoda et al. [26] highlight along with problems in gathering and clarifying requirements, prioritising, securing feedback, pressure to overcommit, loss of productivity, and business loss can be also caused by inadequate customer collaboration.

Similarly, reasons such as inadequate communication whereas intensive communication with customer lead to capture the RC [24], inadequate documentation, rushed analysis when defining requirements, and wrong set of initial requirements were also mentioned as reasons for RC origination:

The majority of our survey participants (N=28; 70%) used Scrum in their projects, and according to the *14th State of Agile Survey Report* [27], 58% of their respondents used Scrum. Positing this fact, Pikkarainen et al.'s findings [28] indicate that Scrum and XP do not allow stakeholders to communicate well, especially in situations where the number of stakeholders are high. This is a major concern for agile researchers to study in the future: how Scrum/XP hinders communication as a method by its own, which also causes origination of RCs.

Cao and Ramesh [29] found that rapidly changing competitive threats, stakeholder preferences, software technology, and time-to-market pressures (which we define as human-centric *reasons*) are key *reasons* for that pre-specified requirements become inappropriate. These can be viewed as the the root causes for the wrong set of initial requirements. As a remedy to this, Gall et al. [30] proposed a framework in which the requirements elicitation meetings are recorded, and important stakeholder statements are automatically extracted and stored in a database. However, their framework is most suitable for traditional software development where specific events are set to elicit requirements. If their approach is used in agile, all agile team *events* are required to be recorded, especially daily standups. This is because our findings suggest that many RCs originate at daily standups. This will have a number of practical implications given that standups are done daily.

The majority of the sources of RCs are human-centric: We found that the majority of RCs' origin is not directly related to software-centric artefacts, but instead to activities where humans heavily interact with each other, such as when building product backlog, undertaking project reviews and in technical team discussions. Therefore, as long as human interaction/collaboration is high, there is a chance for an RC to originate.

Using our taxonomy to help agile teams better understand and manage their project RCs: Our study suggests that teams need to be more knowledgeable about the range of *types* of RCs; in which *forms* they can be best documented; key *reasons* that can address proactively to minimise unexpected origination of RCs; which *sources* that RCs can be found; from whom their RCs can be expected to be *carried* to the team; and the *events* where most RCs can originate.

Document RCs with ample details: Despite agile software development's aim to minimise documentation, well-documented RCs are needed to improve project outcomes. Irrespective of the medium used to document the RC e.g. can be an online tool/a white board/or even post-its, the RCs need to be documented to a satisfactory level. Our results suggest that different RC documentation approaches suit different *types*, *sources* and *carriers* of RCs.

Maintain sustained and adequate communication about RCs: Even though this is repeatedly recommended by various studies [31], [26], our study found that communication issues result in more RCs, causing teams more work and impacting quality, delivery etc. We highlight the importance of sustained and adequate communication, which helps to capture and also to mitigate some unnecessary RCs from originating.

5 THREATS TO VALIDITY

Internal Validity: Our survey questionnaire was based on our initial IBS and FLR findings. Therefore, some RC elements may not have been presented for the participants to directly select. For example, "usage logs/issue trackers", could have been given as an option along with "defect reports" as a RC *sources*. Further research is needed to investigate other such missed RC elements to enhance our agile RC taxonomy. Answers to the questions in the survey may have depended on different situations of the participants. For example, a developer could play the role of an agile coach simultaneously and could act as the *carrier* of the RC. However, it is impractical to cover all such scenarios through a survey. Future research through in-depth interviews would reinforce the findings by complementing them with such scenarios. Even though we provided the definitions of the terminology we used, participants may have not grasped all cases we assumed. For example, one might think that decomposing an epic representing a single requirement into multiple user stories to have one acceptance criteria per user story as addition of RCs. This situation is dissimilar to where an epic representing multiple requirements decomposed to multiple user stories to document multiple requirements. Due to the nature of surveys, it is hard to provide the definitions for all choices and therefore to expect that all participants to share the same understanding of them. For example, in our case, one participant may consider the *reason* "need for refactoring" as a "design improvement" which we provided as a different choice for them to select. Such tangential choices pin a threat to validity to surveys by default.

The level of details the RC could vary from RC to RC. Thus, the utilisation of forms of RC may also vary. All forms, including use cases are heterogeneous. i.e., they are not homogeneous rule-based entities, and not used in the same way in all software development teams, or in all organisations. Therefore, the level of detail they capture may widely vary. For example, one team or organisation may have abstract level representation of the RC whereas another team or organisation may have a detailed level representation of the RC in the same form. In relation to the level of details, a special scenario could be that NFRs might not be explicitly documented, but could be hidden

behind an FR, or considered as something tacit. We did not capture the level of detail of the RC the participants documented. Hence, leaving a threat to the validity of our findings. Therefore, when replicating this study or when doing further research, we encourage the participants to capture the level of detail of the RC to sharpen the findings. The dropout rate of the survey completion was high as 106 participants started it but only 42 completed the survey and with what we determined to be sufficient quality answers. The main reason was that the survey was overly lengthy and it contained questions with matrices which took an excessive amount of time. Unfortunately, this was not reported by any of the participants who participated in our survey pilot study. However, when designing the survey, we placed the participant demographic questions at the end of the survey with the intention of giving participants enough time to answer the survey questions. A positive consequence of this was we were able to gather in-depth and detailed information about RCs from those who completed it. Even though we provided open-ended questions for the participants to provide other options, we found that many did not take the opportunity to give more detailed opinions in our open-ended questions. Collecting further data through in-depth structured interviews would further enrich our findings.

Construct Validity: Interview transcripts of the IBS and answers to the open-ended questions were coded and analysed by the first author. To mitigate potential bias, we had rounds of discussions between all authors of findings as they emerged. During the IBS, the first author, second author, and a Professional Teaching Fellow at The University of Auckland discussed the analysis. During the FLR, and the survey, first, second, and third author discussed the emerging categories to reach consensus.

External Validity: Given the sample size of our study, the findings and conclusions can not represent the entire global agile community. For example, the majority of the participants used Scrum as their agile software development method, but only a single participant had experience with SAFe. Therefore, the findings and conclusions we have derived are biased towards Scrum-based agile contexts. There was no equal distribution of the participants across the world. Our initial IBS was based solely on New Zealand and Australian participants, and the majority of our survey participants (65%) were from Asia. Therefore, the results may be biased towards practice of agile software development in some parts of the world. This territorial bias may hinder generalizing the findings to the entire agile global software development community.

6 SUMMARY

In this paper, we presented an analysis of RCs in agile software development projects. We carried out a mixed-methods approach comprising a series of studies: an interview-based study (10 agile software practitioners from New Zealand and Australia), a focused literature review, and an in-depth survey (40 agile software practitioners). The key facets of RCs in agile projects that we found are, *types, forms, reasons, sources, carriers, and events*. We produced a new taxonomy of agile software project RCs that can be

used by software practitioners to help guide their analysis, management, and actioning of RCs in their projects.

ACKNOWLEDGEMENT

This work is supported by a Monash Faculty of IT scholarship. Grundy is supported by ARC Laureate Fellowship FL190100035. Also, our sincere gratitude goes to *Agile Alliance*, to all the participants who took part in this study, and to the reviewers for their constructive feedback.

REFERENCES

- [1] "IEEE 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology."
- [2] K. Beck, M. Beedle, A. V. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," 2001.
- [3] R. Hoda, N. Salleh, and J. Grundy, "The Rise and Evolution of Agile Software Development," *IEEE Software*, 2018.
- [4] S. McGee and D. Greer, "A software requirements change source taxonomy," in *4th International Conference on Software Engineering Advances, ICSEA 2009, Includes SEDES 2009: Simposio para Estudantes de Doutorado em Engenharia de Software*, 2009.
- [5] S. McGee and D. Greer, "Software requirements change taxonomy: Evaluation by case study," in *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011*, pp. 25–34, 2011.
- [6] N. Nurmaliani, D. Zowghi, and S. Fowell, "Analysis of requirements volatility during software development life cycle," in *Proceedings of the Australian Software Engineering Conference, ASWEC*, 2004.
- [7] A. Inpirom and N. Prompoon, "Diagram change types taxonomy based on analysis and design models in UML," in *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, pp. 283–287, 2013.
- [8] N. Saher, F. Baharom, and O. Ghazali, "Requirement change taxonomy and categorization in agile software development," in *Proceedings of the 2017 6th International Conference on Electrical Engineering and Informatics: Sustainable Society Through Digital Innovation, ICEEI 2017*, vol. 2017-Novem, pp. 1–6, Institute of Electrical and Electronics Engineers Inc., 3 2018.
- [9] P. Ralph, N. b. Ali, S. Baltes, D. Bianculli, J. Diaz, Y. Dittrich, N. Ernst, M. Felderer, R. Feldt, A. Filieri, B. B. N. de França, C. A. Furia, G. Gay, N. Gold, D. Graziotin, P. He, R. Hoda, N. Juristo, B. Kitchenham, V. Lenarduzzi, J. Martínez, J. Melegati, D. Mendez, T. Menzies, J. Moller, D. Pfahl, R. Robbes, D. Russo, N. Saarimäki, F. Sarro, D. Taibi, J. Siegmund, D. Spinellis, M. Staron, K. Stol, M.-A. Storey, D. Taibi, D. Tamburri, M. Torchiano, C. Treude, B. Turhan, X. Wang, and S. Vegas, "Empirical Standards for Software Engineering Research," 10 2020.
- [10] A. Strauss and J. Corbin, *Basics of qualitative research techniques*. 1998.
- [11] S. D. Harker, K. D. Eason, and J. E. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering," in *Proceedings of the IEEE International Conference on Requirements Engineering*, pp. 266–272, IEEE Computer Society, 1993.
- [12] M. R. Basirati, H. Femmer, S. Eder, M. Fritzsche, and A. Widera, "Understanding changes in use cases: A case study," in *2015 IEEE 23rd International Requirements Engineering Conference, RE 2015 - Proceedings*, pp. 352–361, Institute of Electrical and Electronics Engineers Inc., 11 2015.
- [13] M. V. Mäntylä and C. Lassenius, "Drivers for software refactoring decisions," in *ISESE'06 - Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering*, vol. 2006, (New York, New York, USA), pp. 297–306, ACM Press, 2006.
- [14] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002.

- [15] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, pp. 63–92, Springer London, 2008.
- [16] T. Punter, M. Ciolkowski, B. Freimut, and I. John, "Conducting on-line surveys in software engineering," in *Proceedings - 2003 International Symposium on Empirical Software Engineering, ISESE 2003*, pp. 80–88, Institute of Electrical and Electronics Engineers Inc., 2003.
- [17] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013 - Proceedings*, pp. 89–92, 2013.
- [18] K. Madampe, R. Hoda, J. Grundy, and P. Singh, "Towards Understanding Technical Responses to Requirements Changes in Agile Teams," in *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20)*, (Seoul, Republic of Korea), p. 4, ACM, New York, NY, USA, 2020.
- [19] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements engineering and agile software development," in *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, vol. 2003-Janua, pp. 308–313, IEEE Computer Society, 2003.
- [20] I. Gizzatullina, "Empirical study of customer communication problem in agile requirements engineering," in *3rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*, (Lubeck), pp. 1262–1264, Association for Computing Machinery (ACM), 2019.
- [21] N. N. B. Abdullah, S. Honiden, H. Sharp, B. Nuseibeh, and D. Notkin, "Communication patterns of agile requirements engineering," in *Proceedings of the 1st Agile Requirements Engineering Workshop, AREW'11 - In Conjunction with ECOOP'11*, 2011.
- [22] V. Stray, D. I. Sjøberg, and T. Dybå, "The daily stand-up meeting: A grounded theory study," *Journal of Systems and Software*, vol. 114, pp. 101–124, 4 2016.
- [23] R. Krishna, A. Agrawal, A. Rahman, A. Sobran, and T. Menzies, "What is the connection between issues, bugs, and enhancements?: Lessons learned from 800+ software projects," in *Proceedings - International Conference on Software Engineering*, vol. 10, pp. 306–315, IEEE Computer Society, 5 2018.
- [24] X. Wang, L. Zhao, Y. Wang, and J. Sun, "The Role of Requirements Engineering Practices in Agile Development: An Empirical Study," in *Communications in Computer and Information Science*, vol. 432 CCIS, pp. 195–205, Springer Verlag, 2014.
- [25] G. Hoff, A. Fruhling, and K. Ward, "Requirement Prioritization Decision Factors for Agile Development Environments," tech. rep., 2008.
- [26] R. Hoda, J. Noble, and S. Marshall, "Agile Undercover: When customers don't collaborate," in *Lecture Notes in Business Information Processing*, vol. 48 LNBIP, pp. 73–87, Springer Verlag, 2010.
- [27] "14th Annual State of Agile Report — State of Agile," tech. rep.
- [28] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still, "The impact of agile practices on communication in software development," *Empirical Software Engineering*, vol. 13, pp. 303–337, 6 2008.
- [29] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *IEEE Software*, vol. 25, pp. 60–67, 1 2008.
- [30] M. Gall, B. Bruegge, and B. Berenbach, "Towards a framework for real time requirements elicitation," in *First International Workshop on Multimedia Requirements Engineering, MeRE'06*, 2006.
- [31] I. R. McChesney and S. Gallagher, "Communication and coordination practices in software engineering projects," *Information and Software Technology*, vol. 46, pp. 473–489, 6 2004.



Kashumi Madampe is a final year PhD candidate at Monash University, Melbourne, Australia. Ms. Madampe did part of her current research at The University of Auckland, New Zealand. Prior to the PhD candidature, she was in the software development industry as a project manager and a business analyst. Her research interests are requirements engineering, human and social aspects of software engineering, software repository mining, grounded theory, and natural language processing. She serves as the XP2021 poster co-chair, ASE2021 publicity and social media co-chair, and CHASE2021 social media chair. More details about her research can be found at <https://kashumim.com>. Contact her at kashumi.madampe@monash.edu.



Rashina Hoda is an Associate Professor in Software Engineering at the Faculty of Information Technology, Monash University, Melbourne. Her research focuses on human and social aspects of software engineering, socio-technical grounded theory, and serious game design. She serves on the IEEE Transactions on Software Engineering review board, IEEE Software advisory board, as ICSE2021 social media co-chair, CHASE 2021 program co-chair, and ICSE2023 SEIS co-chair. For details see www.rashina.com.

Contact her at rashina.hoda@monash.edu



John Grundy received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is an Australian Laureate fellow and a professor of software engineering at Monash University, Melbourne, Australia. He is an associate editor of the IEEE Transactions on Software Engineering, the Automated Software Engineering Journal, and IEEE Software. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at <https://sites.google.com/site/johngrundy/>. Contact him at john.grundy@monash.edu.