Artefacts and Agile Method Tailoring in Large-Scale Offshore Software Development Programmes

Article *in* Information and Software Technology · March 2016

DOI: 10.1016/j.infsof.2016.03.001

1 author:

Julian Michael Bass
University of Salford
92 PUBLICATIONS **507** CITATIONS

SEE PROFILE

Artefacts and Agile Method Tailoring in Large-Scale Offshore Software Development Programmes

Julian M. Bass

University of Salford, The Crescent, Salford, Manchester, UK

Abstract

Context: Large-scale offshore software development programmes are complex, with challenging deadlines and a high risk of failure. Agile methods are being adopted, despite the challenges of coordinating multiple development teams. Agile processes are tailored to support team coordination. Artefacts are tangible products of the software development process, intended to ensure consistency in the approach of teams on the same development programme.
Objective: This study aims to increase understanding of how development processes are tailored to meet the needs of large-scale offshore software development programmes, by focusing on artefact inventories used in the development process.
Method: A grounded theory approach using 46 practitioner interviews, supplemented with documentary sources and observations, in nine international companies was adopted. The grounded theory concepts of open coding, memoing, constant comparison and saturation were used in data analysis.
Results: The study has identified 25 artefacts, organised into five categories: feature, sprint, release, product and corporate governance. It was discovered that conventional agile artefacts are enriched with artefacts associated with plan-based methods in order to provide governance. The empirical evidence collected in the study has been used to identify a primary owner of each artefact and map each artefact to specific activities within each of the agile roles.
Conclusion: The development programmes in this study create agile and plan-based artefacts to improve compliance with enterprise quality standards and technology strategies, whilst also mitigating risk of failure. Management of these additional artefacts is currently improvised because agile development processes lack corresponding ceremonies.

Keywords: Agile software development, software development artefacts, scrum, large-scale, enterprise, offshore, outsourced, grounded theory, process tailoring

Email address: j.bass@salford.ac.uk (Julian M. Bass)
URL: https://www.seek.salford.ac.uk/profiles/JBass.jsp (Julian M. Bass)

1. Introduction

Practitioners managing large-scale offshore software development programmes appear to find it increasingly attractive to blend elements of both plan-based and agile software development methods, with the result being a pragmatic tailoring of agile methods which accommodates organisational constraints, governance requirements and geographical distribution. This article explores process tailoring by investigating artefact inventories, using empirical data collected from industry practitioners at all levels representing nine international companies.

Agile methods have been proposed as way to avoid project failures [1]. Risk of project failure is reduced each time a software increment is delivered, since the highest priority requirements are selected for development during each increment and each increment is used to gather client and user feedback. Increments are delivered regularly and each comprises a carefully defined fragment of the overall development effort. This contrasts with plan-based methods in which risks progressively rise until product handover at the end of the project. There is evidence that agile methods improve both software development productivity and product quality [2]. However, such agile methods are conventionally associated with small, co-located development teams.

The scaling of agile methods to large-scale software development programmes has attracted interest from practitioners [3, 4, 5], and has been identified as a priority area for researchers [6, 7]. For example, the scrum of scrums approach supports multiple concurrent scrum teams [3]. Teams working in parallel with each other need coordination, consequently the scrum masters from each team work together to coordinate activities, manage dependencies and avoid the duplication of effort [8, 9].

Outsourcing is the process of procuring products or services from a third-party vendor or provider [10]. In onshore outsourcing the third party vendor is located in the same territory as the client organisation. Onshore outsourcing is not the focus of this study. Offshore outsourcing involves a geographically remote third-party vendor, often separated from the client organisation by significant temporal and cultural distance. In contrast, some client organisations establish their own in-house offshore development centres. Offshore development (whether in-house or outsourced) can help establish a presence in emerging markets or benefit from their anticipated lower cost base.

The focus of this study is on large-scale development programmes comprising a system under development or integrated portfolio of related products. Inevitably, large-scale systems involve the integration of new features into an existing code base sometimes called a legacy system. In this study, large-scale consists of at least 25 developers configured into multiple cooperating teams working together for a period of 9 months or more.

Various forms of artefact are used to negotiate, record and disseminate decisions made during the development process. An artefact is a tangible product or by-product produced during the development of software, typically including: models, designs, reports and source code. The artefacts act as boundary objects between the different technical specialisms of stakeholders involved in

the development programme [11].

Development teams are required to record and share design decisions to avoid duplication of effort and resolve dependencies. The artefacts record the results of negotiations between stakeholder groups, decisions made and decisions revised. Written records contradict the agile manifesto which advocates focus on working software over comprehensive documentation [12]. So, large-scale agile development programmes dictate forms of documentation to coordinate the activities of groups and teams, yet agile methods advocate focus on working code. Thus, artefacts represent an area of tension between traditional plan-based methods and agile methods. As a consequence, artefacts can provide insights in to the tailoring of agile methods in large-scale development programmes.

To enhance understanding of software development process tailoring in large-scale offshore agile software development programmes, this research explores practitioner interactions with the artefacts used. The main research question for this study is: "how do practitioners describe the inventory of artefacts they use in large-scale offshore software development programmes?"

This primary research question is further explored using two subsidiary research questions: "how do the artefacts map to software development processes used in large-scale offshore software development programmes?" and "how do these practitioner descriptions contribute to our understanding of artefacts in agile method tailoring in large-scale offshore software development programmes?"

In order to answer these research questions the author has conducted qualitative empirical research with nine international companies engaged in large-scale offshore agile software development programmes; leading to 46 open-ended semi-structured interviews with practitioners ranging from senior executives to novice testers and developers. In addition, documentary sources describing development process standards and guidelines have been reviewed and workplace observations have been conducted.

The main contribution of this article is a systematic description of practitioner interactions with the artefacts created in large-scale offshore agile software development programmes. The project teams in this study use several agile techniques, notably: daily stand-up meetings, short iterations, prioritized backlogs, iteration planning, retrospectives and release planning. These techniques have been identified as most popular by respondents to a well established industry survey [13].

Five categories of artefacts emerge from the empirical data collected in this study: feature, sprint, release, product, and programme governance. A taxonomy is then established that relates these artefacts to their role in the software development process. An actor within the development process is identified as primary owner of each artefact. Further, information sources and information consumers for each artefact are derived from the data obtained during this study. It is suggested that agile processes are missing ceremonies for managing certain artefacts and that agile processes need to be enhanced with additional scrum of scrum ceremonies to manage these artefacts in large-scale offshore software development programmes.

The rest of this paper is structured as follows. In the next section a review of related work is undertaken, with agile methods summarised, along with a brief review of global software development, where the use of agile methods in large-scale offshore software development programmes is considered. The article then introduces the research method adopted, providing information on the selected research sites, data collection methods and analysis undertaken. Findings are then presented, organised into sections on programme governance, product, release, sprint, and feature artefacts. At the end of the article, the findings are discussed, and the limitations of the work are presented, along with suggestions for further work and conclusions.

2. Agile Software Development

There are a range of agile software development methods that are increasingly being adopted in large-scale offshore software development programmes, including Feature Driven Development, Scrum, Extreme Programming (XP) and Lean Software Development [14]. These software development methods build upon three key themes in software engineering: development using short iterations, feature-driven development and the close interaction with customers.

Short iterations are now widely used in software development, providing frequent and regular feedback on smaller scale development activities, rather than more traditional six or nine month development efforts. Within an agile team, short iterations help to identify the causes of any development delays, while the relevant development team also gains feedback from external stakeholders regarding its compliance with agreed requirements. Short iterations require the continuous integration of software code artefacts as they are combined, tested and released [15]. The use of automated software tools migrates code artefacts between the platforms used for build, integration and testing activities.

In feature driven development, team members work together in self-organising teams to produce end-to-end functionality [16]. Each feature is designed to meet a business need and is just one element of a much larger system. Such features must include all the necessary technical components (including databases, network communications and user interface screens) needed to solve that specific business problem, and thus provide end-to-end functionality. Features are self-contained and independent, making them well suited for managing from the initial requirements stage through design and implementation to final testing. Staff members in feature teams either possess all the necessary skills needed to build all the technical components to implement a particular feature; or occasionally they work together in small groups to bring the required skills together. The feature team concept is in contrast to teams organised around technical specialisms, where, for example one team has the skills to develop user interfaces with another team responsible for developing databases.

The close collaboration with customers has been advocated as a way to make detailed application domain knowledge available to the team [17]. Customers, or more specifically, product owners in a scrum, should be able to identify and prioritise requirements and define the test criteria for the successful completion

of each requirement. Product owners can also take responsibility for assessing the number and severity of outstanding issues in a code base, and for approving the release to any clients.

2.1. Large-Scale Agile Methods

There are currently four main themes of concern in using agile methods on large-scale development programmes: scaling, portfolio management, inter-team coordination and architecture [18]. The Scaled Agile Framework (SAFe) is targeted at large enterprises and has three main layers: team, programme and portfolio [3, 19]. SAFe comprises an implementation strategy, nine lean agile principles and extends the scrum of scrums concept. The framework uses a structured programme of training and large scale organisation change to support adoption. The scrum master and product owner roles are envisaged along with selected engineering practices from XP.

The scrum of scrums meeting, in which scrum masters from cooperating teams meet to provide each other status updates, has previously been investigated [20, 21]. It was found that, in large-scale projects, the scrum of scrums meeting membership can grow making it difficult to expedite the meeting in the recommended 15 minutes. Further, scrum of scrum meeting participants did not find it useful to listen to status updates from teams working on relatively unrelated aspects of the development programme. Both studies [20, 21] found practitioners preferring to conduct a more frequent series of smaller, more focused, scrum of scrum meetings to share information about related aspects of the development programme, alongside a less frequent large scrum of scrums meeting involving all stakeholders.

Release planning, in large-scale agile development programmes, is an on-going process comprising regular scoping and prioritisation decisions [15]. This approach to release planning is in contrast with traditional approaches which create the release plan at an early stage of development process, a release plan which is adhered to throughout subsequent phases.The release planning decision making may be decentralised to include development team members [15] or it may be more centralised in the product owner job function informed development team members where technical dependencies are present [22].

To retain flexibility in the face of changing requirements, architecture is not fixed at the start of an agile development programme. In small agile teams, the whole team takes on-going responsibility for developing architecture during the development project. However, in large-scale agile, teams cooperate to build the overall system, sometimes subject to corporate governance or third party development standards, and so cannot realistically be jointly responsible for architecture. Eckstein [23] identifies four approaches to architecture development in large-scale agile programmes: community of practice, chief architect, technical service team, technical consulting team. In the community of practice model, a self-selecting small group of specialists representing each team work together to derive the architecture. The chief architect approach, in contrast, uses a technically skilled individual to coordinate the architecture needs of the

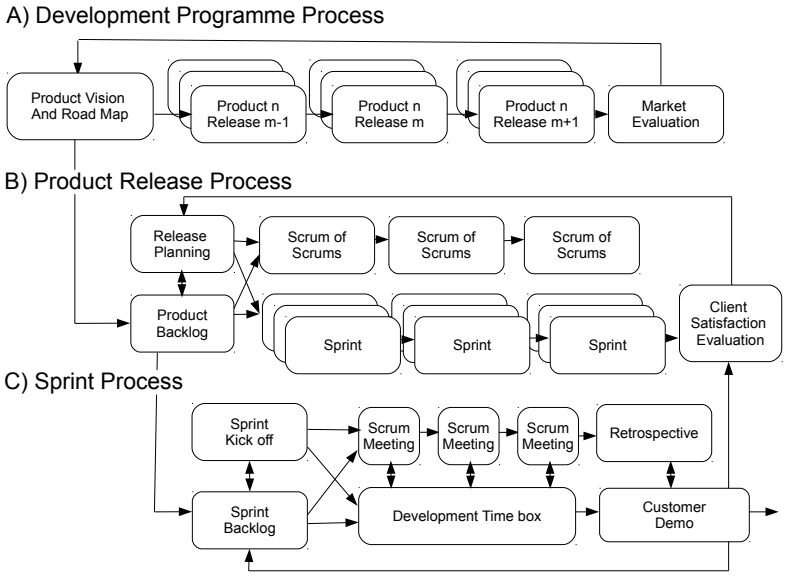

Figure 1: Overall Scrum of Scrums Process (adapted from [22])

cooperating teams, this is similar to the Technical Product Owner role identified in [22]. In development programmes with complex architecture needs, a specialist team is formed called the technical service team [24]. The technical service team provides architecture development services to the feature teams. Some organisations impose corporate architecture governance on teams across development programmes, in such cases a technical consulting team can provide architecture support to features teams combining the community of practice or chief architect and technical services team models [23].

The overall scrum of scrums process, as shown in Figure 1, is comprised of the individual sprints shown in (C), the co-ordinated product releases illustrated in (B), and the portfolio of multiple products that form the software product lines of (A). The sprints in Figure 1 (C) consist of sprint planning to prepare a sprint backlog and conduct of a sprint kick off meeting; with scrum meetings held daily throughout the development period culminating in the demonstration of the working code to a customer, followed by a retrospective. In Figure 1 (B) a number of sprints are included in a release, whilst in Figure 1 (A) a set of products are produced through a series of releases.

2.2. Agile Method Tailoring

Agile method tailoring describes the overall process of selecting or adapting software practices and comprises two main approaches, contingency-based

method selection and method engineering [25]. Contingency-based method selection assumes that software development methods are not universally applicable but that teams or organisations should select a method in its entirety, dependent upon their project context. In contrast, using the method engineering approach, development teams construct a bespoke new process using method fragments [26].

For example, a method engineering approach appears to have been used at Intel Shannon where aspects of both scum and XP were adopted [27]. A recent systematic literature review of empirical agile tailoring research papers suggests that the method engineering approach is more popular with project teams [28]. Tailoring interventions can relate to stakeholders, project life cycle, project organisation and knowledge building [29].

2.3. Agile Artefacts

Conventionally, agile development methods consist of roles, practices and artefacts. Previous studies have investigated these roles, including self-organising teams [16], on-site customers [17], product owners [22] and scrum masters [20]. There have also been a number of studies considering engineering practices, conducted as a part of the XP process including pair programming [30, 31, 32], test driven development [33], continuous integration [34] and refactoring [35]. These studies have found that effective XP teams exhibit a shared sense of responsibility, and a relaxed yet rhythmic approach to working; with the resulting good quality code a jointly owned and valued asset [36].

There have also been a number of attempts to identify software development artefacts. Broadly, these artefacts can be categorised in terms of planning, requirements, development, testing and change management [37, 38]. Three patterns of artefact use within requirements engineering have been identified: solution oriented, function oriented and problem oriented [39]. Within these broad categories specific agile artefacts including the product backlog, sprint backlog, burn down chart, task, issue and working software are located. A more detailed ethnography has focused on two specific agile artefacts the physical user 'story card' and the 'wall', a large board displaying project status [40].

3. Method

This research adopts a grounded theory approach, using the empirical data gathered from research sites in selected international companies, as shown in Table 1. Data collection used documentary evidence, workplace observation of software development practices and semi-structured face-to-face practitioner research interviews, which were recorded and subsequently transcribed. The unit of analysis is the set of software practitioners working on development programmes in selected organisations. Data analysis was conducted using a grounded theory approach [41].

3.1. Research Sites

The nine international companies investigated, shown in Table 1, perform either off-shoring (companies B, F and I) or outsourcing (companies A, C, D, E, G and H). Off-shoring is used to access and cultivate specialist technical skills from around the world, with both off-shoring and outsourcing offering access to lower cost skills than traditional in-house onshore specialists. Five of the companies have achieved Capability Maturity Model Integration (CMMI®) maturity level 5 accreditation.

Company B is a well-known Internet business. It has an in-house development capability based in California, USA and has also established a development team in India, to attract a wide range of specialist skills while also reducing staffing costs. In contrast, Company F has interests in the industrial products space. Company F has headquarters in Europe, and also has research and development centres in India and other territories across the world. Work assignments in these organisations are allocated according to the critical mass of technical of expertise within specialist groups, in order to avoid the duplication of competencies. The selected IT services companies (Companies A, C, D, G and H) are each well-known vendors in the world-wide software and/or IT service outsourcing sectors. The two largest companies investigated in this study have a turnover of almost €8 billion and over US $1.5 billion.

The selected companies have head offices located in Germany, India and the USA, although the research sites chosen were exclusively in the UK and India, due to budgetary constraints. The interviews were conducted in Bangalore, India, London, England, Delhi, India, and Glasgow, Scotland, between January 2010 and September 2014.

Two phases of sampling were undertaken, in order to decide which companies to include in the study: a snowball sampling technique ([42] pp. 237; [43] pp. 37), which was followed by intensity sampling ([42] pp. 234). During the first phase, former co-workers and other professional contacts provided access to study participants, who then provided access to development teams in other companies. During this initial, exploratory phase of the study, semi-structured interviews were conducted with a broad range of companies (Companies A, B, C, F and G).

Employing snowball sampling techniques provided access to perspectives drawn from a range of different stakeholders. For example, Company C exclusively used agile software development methods, while Companies E and G provided access to some agile sceptics, who had negative experiences to report.

During the second phase of the study, intensity sampling ([42] pp. 234) was used to obtain a greater richness and depth in the study, by accessing a larger number of interview participants with different responsibilities in the same company or software development programme. Triangulated perspectives were provided by developers, quality assurance testers (QAs), project managers, development programme managers and corporate-level executives in Company H and Companies D and E.

Table 1: Participating Companies, Industry Sectors and Interviewee Job Titles

Company	Company Sector	Interviewee Job Titles	Interviewee Projects and Programmes
Company A, Bangalore	IT Service Provider	Programme Manager Senior Project Manager Team Member	Customer Relationship Management
Company B, Bangalore	Internet	Engineering Manager Product Manager	Web Mail Web Calendar
Company C, Bangalore	Software Service Provider	Development Manager	Rail Booking
Company D, Bangalore (Offshore Provider to Company E)	Software Service Provider	Project Manager Product Owner Scrum Master (3) QA Lead Team Member	Marketing Campaign Management Customer Relationship Management
Company E, London	Enterprise CRM	Programme Manager Project Manager Director of Engineering	Banking Marketing Campaign Management Customer Relationship Management
Company F, Bangalore	Industrial Products	Scrum Master	Healthcare Instruments
Company G, Bangalore	IT Service Provider	Engagement Manager	Media Entertainment
Company H, Delhi	IT Service Provider	Chief Technology Officer Corporate Lead Architect General Manager Human Resources Delivery/Programme Manager (3) Project/Senior Project Manager (3) Scrum Master (2) Technical Analyst/ Consultant/Specialist (6) Team Member (9) Business Analyst	Airline Customer Service Flight Booking
Company I, Glasgow, Scotland	Customer Relationship Management	Chief Operating Officer	Customer Relationship Management

Details of the enterprise software development programmes selected for the study, including team size and development method, are shown in Table 2. As mentioned previously, in this study, a large software development programme is defined as a minimum of 25 developers engaged for a duration of nine months or more. The overall team size will be much larger when analysts and other support staff are included. Three of the development programmes are geographically distributed (Company A, CRM insurance; Company G, healthcare; and Company F, healthcare). While the remaining 17 development programmes use various configurations of onshore clients and offshore teams. All the development programmes in the study and shown in Table 2, are for commercial revenue generation and are not for internal IT infrastructure applications.

In summary, research sites were selected to provide replication using snowball sampling in the first phase of the study, with intensity sampling employed in the second phase to enhance both the depth and richness of the data, with the aim of increasing data reliability through participant triangulation. Using both snowball and intensity sampling methods is a combination sampling approach that provides methodological triangulation to the sample selection. This methodological triangulation also acts to minimise researcher bias in the study.

3.2. Data Collection

Three types of secondary data were used to support the study: corporate process guidelines, project and development programme documentation, and technical reports or white papers. Some of the participating companies also made commercially confidential process guidelines available. These guidelines provided details of corporate agile practices, roles, policies and recommendations. However, it was not possible to obtain access to such documentation from all of the companies, due to the extreme commercial sensitivity of the contents. Some documentation produced for specific software development programmes, including design and architecture documents, has also been explored. Publicly available white papers, technical reports, case studies and descriptions of vendor capabilities were also reviewed. Such documents are usually produced to provide potential customers with marketing information. Direct observation of working practices and work place environments was enabled by on-site visits. Some secure work environments were visited with coordination meetings (stand-up meetings) of both co-located and distributed scrum teams observed at Companies C, D and H. The on-site visits were also used to investigate the arrangements in place for distributed scrum coordination meetings, using both video and audio conferencing technologies. Furthermore, various informal (sometimes off-site) discussions with executives, project managers and development team members were conducted.

The primary data used in the study was gathered from 46 face-to-face semi-structured interviews conducted with practitioners, as detailed in Table 1. The recordings of these interviews were then transcribed using a specialist commercial transcription service and reviewed to ensure verbatim transcription. The interviews were conducted using open-ended interview guide approach. An example of the semi-structured interview guide used in this research is presented

Company	Project	Team Size	Development Process	Project Type
A	CRM (Insurance)	325	RUP[a] Distributed BA Team	Bespoke
	CRM (Banking)	50	Scrum	Bespoke
	CRM (Healthcare)	75	Scrum	Bespoke
B	Internet (Calendar)	25	Scrum	SPL[b]
	Internet (Mail)	25	Scrum	SPL
	Internet (Options)	25	Scrum	SPL
C	Transport (Rail Ticketing)	40	XP	Bespoke
D/E	Marketing (Campaign Management)	25	Scrum	SPL
	Enterprise CRM (Core)	20	Scrum	SPL
	Enterprise CRM (Banking)	12	Scrum	Bespoke
	Enterprise CRM (Credit Card)	20	FDD[c] (Capsule Development)	SPL
	Enterprise CRM (Financial Services)	25	RUP	Bespoke
F	Healthcare (Instruments)	1000	Scrum	SPL
	Industrial Automation	200	Scrum	SPL
G	Media Entertainment	50	Scrum	Bespoke
	Healthcare	180	Scrum	SPL
H	Travel (Loyalty)	30	Scrum	SPL
	Travel (Airline Reservation)	25	Scrum	Bespoke
	Risk Management and Insurance	30	Scrum	SPL
I	Enterprise CRM	190	Scrum	SPL

Table 2: Development Programme Details

[a]RUP, Rational Unified Process
[b]SPL, Software Product Line
[c]FDD, Feature Driven Development

in the Appendix. The interviews were open-ended to provide respondents with the opportunity to raise any issues or concerns that were outside the scope of the scripted interview questions. Interviews were typically conducted in small meeting rooms exclusively booked for the purpose on company premises.

3.3. Data Analysis

The audio interviews and corresponding verbatim transcripts were carefully reviewed to ensure consistency. The transcript text was then imported into a qualitative data analysis software tool, in this case Nvivo V9 [44].

The grounded theory analysis began with the identification of concepts contained within the interview data [45]. These interview concepts were coded and then compared within and between interviewees. These interview concepts were then iteratively grouped and refined into selected categories. Therefore, interview concepts were combined to create categories which were then themselves coded, listed and compared within and between interviewees. There were four main aspects of the data analysis used in this research.

3.3.1. Open Coding

A sentence-by-sentence approach to interview transcript coding was adopted in this research. Each code was represented by a short descriptive phrase. The codes were handwritten onto hard copies of the interview transcripts, during the early stages of analysis. This approach offered a quick and easy way to start analysis, identifying initial codes, that were then collated. These early stage codes were tentative and evolved quickly as data analysis progressed. Later, the data analysis software tool NVivo [44] was used to record and formalise the coding process. Concept classification was used to organise the large volume of data into categories [46], with the categories becoming saturated as the data collection progressed [47]. This categorisation formed the basis of the subsequent grounded theory [41].

3.3.2. Memoing

Memo writing was used to capture and sharpen topics identified using open coding as they develop into categories. Each memo was short, often informal, essay on the topic which includes selected quotations to provide primary evidence. Some memos build upon field notes, that were initiated during data collection, to capture interesting topics raised during interviews. The memo writing helps to clarify, refine and sharpen categories, evolving as new transcript data is added [46, Chapter 12].

3.3.3. Constant Comparison

When using constant comparison, the researcher iterates back and forth between data collection and analysis. Further, constant comparison is used to compare incidents that apply to each category, integrate categories and their properties, delimit the theory, and write the theory [45, pp.105]. This approach is "close to the common-sense approach which one might use when trying to

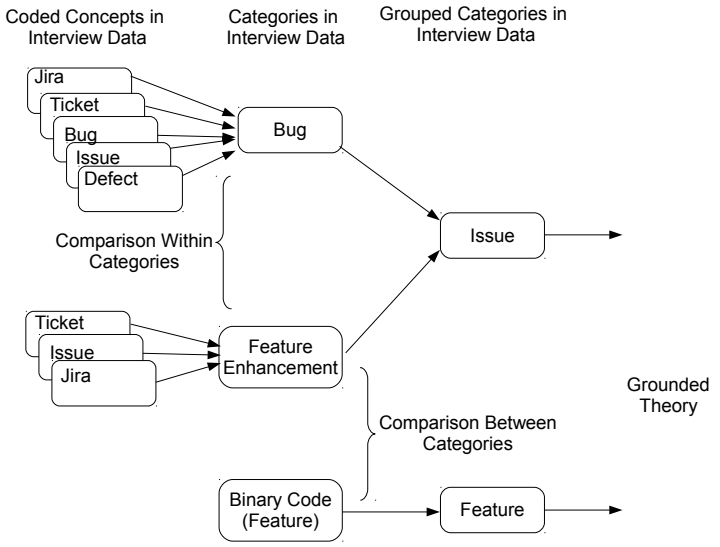

Figure 2: Grounded Theory Development Example

understand something which is complex and puzzling"[48, pp. 148]. Interview transcript codes were compared with each other at two levels: within the same organisation or project team and with outside organisations and teams. In this way, the codes were honed over time using constant comparison.

The data analysis process is illustrated in Figure 2. Practitioners describe managing artefacts called Jiras, tickets, bugs, issues or defects. Analysis of the interview transcripts shows the handling of these artefacts can be grouped. Similarly, practitioners describe a range of activities surrounding feature enhancements. Feature enhancements are clearly related to features, but analysis of the interview transcripts shows their handling within the software development process has more in common with other Jiras, bugs and issues. The grouping and categorisation has been performed iteratively, as data collection has proceeded.

3.3.4. Saturation

At early stages of the research, conducting practitioner interviews with a new company or new project team causes re-appraisal of the categories identified. New artefacts, development practices and stakeholders are discovered at the new research sites. As the study evolves, and the number of respondents increases, the richness and detail of the grounded theory is enhanced. At the later stages of the research, analysis of new research sites provides evidence that is consistent with the grounded theory categories already identified. Gradually, each new research site and practitioner interview has less and less impact on the

categorisation. Saturation has occurred in the research when new research sites or interviews no longer result in new categories in the area of research under investigation.

To summarise, a combination of open coding, memoing, constant comparison and saturation was adopted for data analysis in this research. Early in the analysis, topics were identified using sentence-by-sentence analysis of transcript data. These topics were recorded in memos, which were refined and honed using constant comparison within project teams and between organisations. As the volume of interview data expanded, the topics evolved into categories that form the basis of the grounded theory, presented in the findings. Saturation has been achieved as analysis of new practitioner interview transcripts no longer result in the emergence new categories.

4. Findings

The artefacts identified in this study are organised into five levels of abstraction: programme governance, product, release, sprint and feature. Each of these categories is discussed in turn.

4.1. Programme Governance

Five artefacts have been identified in this study that are used across development programmes: risk assessment, programme architecture standards, test plans, contracts and reference architectures. These artefacts are created to coordinate cooperating agile development teams and mitigate risk of development programme failure by providing a layer of oversight and governance.

4.1.1. Risk Assessment

Risk assessment is used to identify potential sources of risk and estimate the likelihood of occurrence. For each risk identified, mitigating actions are listed and described. In large-scale development programmes, potential financial losses are greater, the possibility of damage to reputation is higher and the consequences of schedule overruns are more severe than on smaller projects. Risk assessment is not normally associated with agile development methods, it is "not really an agile-specific thing. It's for any kind of project" (Delivery Manager, Company H).

An important driver for conducting risk assessments in large-scale development programmes is complexity:

> "if something is seen as technically very complex, it will come up as part of the risk [assessment] of that particular project, and then you will have to [decide] how you mitigate that risk" (Delivery Manager, Company H).

Risks maybe identified around several dimensions such as: client relationships, solutions, team composition, operational issues and delivery. For large projects the risk assessment is performed regularly, as part of project monitoring

> "we have the practice of risk assessment every month and at the time of defining the minimum viable product... to review and update the risks for every Sprint" (Programme Manager, Company A)

Team members are involved in the regular risk assessments

> "the Agile coach and Scrum master work with team and support them to review the risks" (Programme Manager, Company A)

Whereas team composition and solution risks maybe reviewed at the end of each sprint, client risks will be reviewed less frequently, perhaps quarterly. Operational and delivery risks are reviewed only after major milestones, such as product delivery. Architectural design is used to mitigate any technical complexity identified during the risk assessment process.

4.1.2. Architecture Standards

Architecture standards are used to ensure software systems are well structured, internally consistent, simple to understand, easy to maintain and satisfy non-functional requirements. Proposed enhancements to existing systems are thus checked and subject to approval to ensure compliance with corporate technical strategy. Systems "need to have some technical governance on how changes are made" (Lead Architect, Company H). Making unauthorised changes to systems is discouraged and "there can be certain serious implications, if there is no governance out there" (Lead Architect, Company H). In CMMI® level 5 accredited organisations, such as several of the companies in this study, architectural decisions must be recorded in a form that can be disseminated to team members.

Architecture standards are required when the adoption of a new technology or design approach is being considered. The impact of adopting such a new approach must be carefully considered from different perspectives:

> "we had a governance process where you have people from Bangalore, Pune and the UK participating, and they were thrashing out, 'what would we achieve by [adopting] a service oriented architecture?' 'what is our vision of service oriented architecture?' And then define some standards for it" (Development Manager, Company C).

The standards developed by governance authorities are then disseminated to development teams "team members, should have thorough understanding of the domain as well as technical architecture" (Scrum Master, Company H). In turn, development teams can promote their own demands for revisions or extensions to the existing recommended standards:

> "if I'm a developer working on a story that needs some amendment to a service, or which demands a creation of a new service, I could go to the service governance board" (Development Manager, Company C).

Architecture standards are then disseminated in order to encourage consistency of software structure among development teams.

4.1.3. Test Plan

In plan-based software development methods, the emphasis on testing is towards the end of the project: "in a waterfall model, you will actually budget for a QA cycle towards the end of the development cycle" (Development Manager, Company C). In contrast, when using agile methods, each iteration involves testing: "we are taking the [testing] budget and we are spreading it across the development cycle" (Development Manager, Company C). Members of the test team are assigned to work within multi-functional sprint teams.

However, in large-scale offshore outsourced software development programmes, user acceptance test teams are often separate from the development teams "our testing team and development team are different" (Senior Software Engineer, Company H). Several practitioners argue that independent test teams result in higher quality code because developers may reproduce logical errors in both tests and the implemented software. In contrast, when test teams are independent, "testing is always different under different perceptions" (Senior Software Engineer, Company H).

Test plans describe the overall approach to testing for the development programme. The test plan articulates the resources required for different stages of testing. The schedule for conducting the different types of testing is described in the plan, which is particularly important where personnel from a third-party user acceptance test team are required, as is more common on large-scale offshore development programmes than smaller projects. The test plan also describes the deployment of personnel, whether centralised in a dedicated test team, or distributed in multi-functional development teams.

4.1.4. Contracts

Contracts between outsourcing vendors and clients are are always delicate, potentially adversarial and sometimes the scene of outright conflict. The complexity and duration of large-scale development programmes mean that change is inevitable, "I've never been involved in a tender process whereby the deliverable has been what they tendered for. You know, in bigger projects, there's always change" (Chief Operating Officer, Company I). The two main types of contract observed in this study are fixed price and time and materials (T&M) "when we draw up contracts, it will be a fixed price, or a T&M contract" (Delivery Manager, Company H). Both contract models can be used to accommodate change during the contract. A T&M contract enables ongoing billing for work completed, while in "a fixed-price project, you'll look for change requests, to make it more [like] time and materials" (Chief Operating Officer, Company I). In contrast, in-house offshore organisations, such as Companies B and F, use an internal project approval process to allocate resources for development programmes and projects.

4.1.5. Reference Architecture

In large-scale offshore software development programmes, reference architectures are required in order to maintain consistency in design between cooperating development teams working on the same development programme. Outsourcing vendors are typically subject to client approval processes to ensure development programmes adhere to prevailing architecture standards:

> "a technical design authority looks at a reference architecture for a [development] programme or for a solution. So, governance is in place" (Lead Architect, Company H).

The reference architecture is then used to disseminate the approved architecture to stakeholders in a development programme:

> "this requires almost every individual to put a lot of effort into understanding the architecture of the product, and the common patterns of issues that we're getting. So this responsibility is shared in the team" (Scrum Master, Company D).

The reference architecture simplifies software maintenance and facilitates staff deployment across project teams in the development programme. The reference architecture is created to ensure development teams consistently follow the agreed design approaches.

The reference architecture defines how the software will meet non-functional requirements, such as performance or security goals:

> "to be able to implement these features, or new concept, we need to do certain design changes or maybe architecture changes. So before we can start working on a particular feature, these are the engineering tasks they need to perform. There could be certain performance guidelines we need to meet, there are security guidelines we need to meet. Engineering managers or teams, they start to look at the pre-requisites before we start working on any major features" (Product Manager, Company B).

In large-scale offshore software development programmes, the reference architecture sustains the project beyond the needs of the functional code in the current iteration:

> "So while doing the designing with the current architecture, we are [also] trying to make a design that it is adaptable to future requirements as well. We make the design more generalised [and] try to make more decoupling. So that future requirements can be catered for in this [architecture]" (Technical Analyst, Company H).

4.2. Product Artefacts

This study identifies four artefacts used at the product level: product backlogs, product architecture implementation, user acceptance tests and product release binaries.

4.2.1. Product Backlog

The product owner elicits and prioritises requirements in the form of a product backlog. Items in the backlog are re-prioritised, as the development programme progresses, by the product owner, "it is a product owner's responsibility to make sure product backlog should be continuously evolving" (Programme Head, Company H). Product grooming, the process of constantly re-prioritising requirements, is necessary to ensure that the next sprint tackles the highest value development items. In Company H, for example, "if [product backlog items]... are not sequenced properly, whatever we are delivering will not give that priority value to the end user or the business user" (Programme Head, Company H).

In large scale development programmes, higher levels of complexity cause a greater number of technical dependencies between product backlog items.

Scrum masters help the product owner to identify technical dependencies between backlog items and assist in prioritising items, "it's the scrum master and the product owner's responsibility to make sure the backlog is prioritised (Developer, Company D). However, development teams members do not usually have the opportunity to influence items in the product backlog, "we as team members don't really have the access to the [product] backlog" (Developer, Company D).

The diverse stakeholder communities in large-scale development communities increase the likelihood of conflicting requirements:

> "different regions like APAC and Europe and North America come with some requirements, this is what they want to get done" (Engineering Manager, Company B).

The product owner identifies and reconciles the needs of the different parts of the client organisation. Successful product owners have the necessary authority to perform this conflict resolution function, "we keep re-triaging [the product backlog] because we have too many mandatories" (Engineering Director, Company E). In Company H, if the "product backlog... is not groomed properly, or in advance [of iteration planning], we really find [it] a challenge" (Programme Head, Company H).

These findings show that product owners create and continuously revise the product backlog during the software development programme, with scrum masters providing technical advice regarding dependencies between features.

4.2.2. Product Architecture Implementation

The project teams analysed in this study conducted architecture implementation prior to development iterations, at the same time as release planning. Architecture implementation changes precipitated by a new requirement may well impact on the work of multiple development teams, and so is therefore subject to scrutiny outside of any specific team. Practitioners often refer to architecture implementation as high level design.

Architecture implementation changes can arise from new functional requirements.

> "In our company, the project manager comes up with high level feature list first ... and tells the overall team, 'to implement these features, we need to do certain design changes or maybe architecture changes'. We see if the design is okay in a couple of sprints, then start working on actual features" (Product Manager, Company B).

There is an awareness that architecture decisions can have an impact beyond the scope of a particular feature or iteration, "if you've taken a good decision it will help you, if you've taken a bad decision, then you will have to go back and fix it" (Development Manager, Company C).

4.2.3. Other Product Artefacts

Two other product artefacts identified in the research are user acceptance testing and product release code binaries. User acceptance testing (UAT) follows integration and regression tests "after two/three sprints we have UAT release testing for two weeks" (Scrum Master, Company H). Sometimes the UAT is conducted within the development team, "all the developers and QA are executing the [automated test] scripts" (Scrum Master, Company H), but it is often conducted by a third party team on behalf of clients, "once [the code] is released from the development team, the UAT team picks [it] up and starts doing testing" (Scrum Master, Company D).

The development teams in this study do not release code at the end of each iteration, rather multiple iterations are integrated into product release code binaries. A freeze on new features occurs when all the functionality has been implemented during the final sprint in the release, "once the code complete cycle happens, from now on the developers are not going to merge anything in that branch" (QA Lead, Company D). Product owners then approve release, if test results exceed the required quality threshold "we can ask the product owner, 'can we go ahead with the release cycle'?" (QA Lead, Company D).

4.3. Release Artefacts

Practitioners in this study identify four artefacts applied at the release process level: regression tests, release code binaries, release plans and integration tests.

4.3.1. Regression Tests

Functional regression testing is used to confirm the correct behaviour of previously implemented software, after new features are added. Regression testing is "to make sure the new feature does not affect the older features" (Software Developer, Company D).

Despite being automated, running full regression test can be very time consuming. "Regression tests take too long to do [on] every sprint. So we don't run a regression suite, it is like three and a half days to run a full regression suite, so we don't run that" (Director of Engineering, Company E). Similarly, at Company H regression testing too time consuming to perform at the end of

each sprint "And after three to four sprints we have release testing. And release testing normally [lasts] two weeks" (Test Analyst, Company H).

Practitioners argue that time consuming regression tests are an important reason why code is not released to clients at the end of each iteration in large-scale offshore development programmes.

4.3.2. Release Code Binaries

In large-scale offshore software development programmes, the release process includes governance checks on releases to minimise any customer dissatisfaction arising from poor quality releases. In the companies analysed in this study, release plans, integration code binaries and release candidates are produced prior to any product release to clients.

Therefore, a release will commonly consist of a series of iterations, often involving multiple teams. A strategy is prepared for each release, "on the basis of the new features we have implemented... I will come up with a strategy. How much time my team requires for doing the [release]" (QA Lead, Company D).

The presence of multiple scrum teams (whether co-located or geographically distributed) can lead to problems in integrating the code bases under development by different teams.

> "They follow their sprint cycle, we follow our sprint cycle. Finally, they match and we take their code base merge into our one and give it to QA as a single release and then it goes into production" (Product Manager, Company B).

Similarly, "as a scrum master, the challenges were... code synchronising problems, task updating problems" (Scrum Master, Company D).

Scrum masters must therefore adopt and disseminate an integration coordination strategy. "Suppose Team A has a 'show and tell' [customer demo] tomorrow, say Wednesday. On Monday there would be a code freeze for that particular branch. On that branch, no other team members from Team B or Team C would be allowed to check in any code into that particular branch. So for two days, because there would be show and tell preparation, for two days the branch would be blocked by that particular team" (Project Manager, Company H).

4.3.3. Other Release Artefacts

Two other artefacts, release plans and integration tests, are identified from the data in this study and are summarised here. Large projects require a release plan to schedule product delivery, often coordinated with external events such as TV or print media advertising campaigns "I'd advocate more agile feature set delivery [but] there would still be a master plan of everything you're trying to build" (Chief Operating Officer, Company I) and "there's a six-monthly road map discussion... where they have a high-level look at the things we are trying to achieve this year" (Engineering Manager, Company B). Integration tests are typically used to identify problems with data or control flows through

the software system, "to validate what you're building is right, and get early visibility of issues, [you] do a happy path end-to-end test, quite early on in the project" (Chief Operating Officer, Company I).

4.4. Sprint Artefacts

Five artefacts have been identified that support the sprint development process: sprint backlog, user story estimates, burn down chart, status board and sprint code binaries.

4.4.1. Sprint Backlog

Practitioners argue that incremental approaches reduce initial time to market and improve resilience to change during development programmes. In plan-based methods "you go for a requirement analysis, then design, and then coding. And by the time you actually go for delivery... the market situation has changed" (Scrum Master, Company F). Reducing time to market and getting feedback on product releases is attractive, "[we want to] come in the market as soon as possible... with newer ideas" and "you don't know your customers face-to-face, so it's pretty critical to get the product out [and] get their feedback" (Engineering Manager, Company B). However, agile methods are less attractive for when negotiating between external providers, "in systems integration we need to interact with multiple vendors... they don't have any knowledge of agile" (Engagement Manager, Company G).

In large development programmes, 'ad hoc' features emerge during the conduct of the project. As such 'ad hoc' features are not a part of the functional requirements gathering process. However, 'ad hoc' features are often needed to address other corporate needs. For example:

> "In a company like ours, because there are multiple products, they might come up with a security issue or a new way of registration, a new way they handle some cookies. Suddenly, every team in our company needs to get this particular thing done by a particular date. Suddenly, those things become priority, then obviously we need to drop a few things on the next sprint to pick up those [new things]" (Product Manager, Company B).

The sprint backlog is the mechanism used to add 'ad hoc' features into the development pipeline.

4.4.2. User Story Estimates

In scrum, story points are often used to estimate work: "we have projects that are using story points for estimation" (Programme Manager, Company H) and "we give a story point [estimate] for each feature" (Senior Developer, Company D). An advantage of using story points is that a burn down chart can be generated during each sprint and a velocity can be calculated for the team at the end of the iteration. However, another common approach to estimation is known as 'T-shirt sizing' which divides effort estimates into the broad categories of small, medium and large:

> "Everybody gives their own T-shirt sizes to their particular story. If somebody is saying that it's a large size story, then we listen to him talking about why it is a large story. And if we all agree, then it's a large story" (Scrum Master, Company H).

Tee shirt sizing is less onerous to perform than story point estimation because it is less precise. However, tee shirt sizing does not provide the numerical estimates that can be used to produce burn-down charts or perform velocity calculations.

A collaborative approach among team members is sometimes used, "we check whether we have the capacity to do those stories or not. We do the planning, we do the estimates" (Scrum Master, Company H). The task estimation performed by team members is then communicated to managers, "a product manager says to the development team 'tell me how much time we'll need to finish this and [we'll] commit to that'." (Product Manager, Company B). However, on a number of development programmes estimates are performed by managers, in negotiation with clients, and presented to the teams.

4.4.3. Other Sprint Artefacts

Three other sprint artefacts identified in the research are burn down charts, sprint code binaries and status board. A burn down chart is used to record and disseminate completion of user stories in scrum, "we maintain a burn down chart on a day to day basis" (Senior Project Manager, Company H). Within each team, the source code for new features must be integrated with the source code for existing features "we integrate the new code and the old code every week" (Developer, Company H). A physical status board is used to disseminate the progress of features through the development process, "the card wall is nothing but a picture of how your iteration is going. We have stories which are marked there at different stages of development" (Development Manager, Company C) or, "on the Kanban board we have all the different columns created, that is, to be 'done, 'verified', 'in progress', 'impediments'." (Developer, Company H). However, in geographically distributed development teams, a physical board is less useful. Software tools were thus used to disseminate information to dispersed team members, "we used an internal tool which actually creates the Kanban board virtually [i.e. an online representation of the Kanban board]" (Technology Consultant, Company H).

4.5. Feature Artefacts

Practitioners in this study identified seven artefacts that describe aspects of specific features: user stories, detailed designs, source code, test criteria, unit tests, issues and feature code binaries.

4.5.1. Source Code

User stories drive the source code development process "we pick the user story from the proxy [product owner], and then work from user stories" (Developer, Company H). The source code produced by the teams analysed in this

study usually follows a feature driven development approach, as has been mentioned. Features are used in order to encourage cohesion of closely related functions, and independence of unrelated functions, to avoid undesirable coupling "there should be no interdependency between two different features" (Software Developer, Company D).

The features require a vertical slicing of the project architecture into end-to-end functionality, thus a user story comprises elements of user interface, business logic functionality and data storage, with team members working together to provide these features as end-to-end functional units. For example,

> "[agile] is aimed at providing customers [with] the most satisfaction... basically it's designed around vertical slicing of the project... let's say we have completed the server side for ten functionalities. But we haven't created the UI for all of them, so we can't say to the customer that those things are all working" (Developer, Company H).

This is in contrast with conventional project organisation where teams are organised around the technical specialisms needed for horizontal layers, such as database design in the persistence layer or user interface development for the presentation layer. Advocates of plan-based approaches saw risks in the feature driven development approach, for example:

> "At that time the US [team approach] was to do every thing, in little 'capsules.' They developed the requirements and then [implemented the] program and tested a capsule then move to the next one... This struck us as a bit dangerous. Because you could get 10 capsules on and find something in a capsule that affected an earlier capsule that [consequently] needed reworking" (Programme Manager, Company E).

Large-scale offshore software development programmes often create new software in order to re-implement, modify or integrate with the source code of a legacy system. Sometimes projects may have to re-implement the legacy code, "So you had a piece of old code which has to be re-written" (Development Manager, Company C), while other projects will have to change the behaviour of legacy code "[we have a] piece of code which was [previously] attempted at, but which doesn't really match what [the client] wants" (Development Manager, Company C). This may include adding new features to interact with existing legacy code. "You somehow had to combine them, marry them together, and then form a transition plan so that you can reach where the clients want it" (Development Manager, Company C).

Large-scale offshore software development programmes are complex, with multiple interacting programming languages and software technologies, "we are using Java, we are using C, [and] some other code is in CORBA" (Developer, Company H).

> "It's really difficult to understand everything because it involves Java, C++ and [an international flight booking] server. [The international flight booking] API is third party area, it's [integrating their] libraries and your own code" (Senior Developer, Company H).

The development teams in this study developed features using source code artefacts comprising multiple programming languages, which integrated with existing or legacy systems.

4.5.2. Issues (Defects and Feature Enhancements)

On large-scale offshore software development programmes, there are often many more known issues and bugs than can be fixed within the time and resources available. Consequently, the issues are prioritised according to severity and the likely impact on end users. Selected issues are then fixed within development teams and the solutions subsequently tested.

The companies analysed in this study reported using automated software tools to manage these issues. For example, in Company B:

> "I'll create a ticket in my system, [then developers will update the ticket saying] 'we have progressed a bit and only this part is left.' I automatically get an email notification [saying] that the ticket has had an update" (Product Manager, Company B).

Issues are constantly reviewed and prioritised, "we have a process where we go through each and every open defect and we see if [it] needs to be fixed or not" (Architect, Company D), and "there is a certain number of Jiras [a colloquial name for issues tracked using the Jira software tool] per sprint which I like the team to take up, so the quality of the line product improves" (Director of Engineering, Company E).

This study has identified three different strategies used for handling issues: a dedicated maintenance team, a subset of development team members assigned to issue resolution and periodic sprints focusing specifically on issues. Company D has a sustaining team that focuses on bugs, issues and infrastructure upgrades (such as migration to new operating system releases, or support for new browser releases). For instance, "the sustaining group takes care of bug fixes in the previous releases" (Architect, Company D). Whereas, on one project in Company H, a number of developers are assigned to dealing with these issues, "we have two or three developers in the whole team who are assigned to defect removal" (Test Analyst Company H).

The relevant bug fixes are then collated into releases. For example,

> "in [the] case of minor issues, we plan for a minor release, which is typically handled by the sustaining team. We may have to give technical help to the sustaining team" (Scrum Master, Company D).

Furthermore, in each of these minor releases, a significant number of issues are resolved, "so for every release, we have close to 200 defects fixed" (Architect, Company D).

The development teams in this study manage large numbers of issues caused by the complexity and scale of their development programmes.

4.5.3. Other Feature Artefact Findings

This research has found evidence of five other feature-level artefacts: user stories, detailed designs, test criteria, unit tests and feature code binaries which are summarised here. User stories are developed during requirements analysis by product owners, a "product owner is making a [user] story" (Scrum Master, Company H). Detailed designs are needed to meet complex requirements approved prior to implementation "This particular story could be complex, so we put an extra check with that story, we will first create the design of that story [then] the design will get approved, and then we will move it to development" (Developer, Company H). Test criteria are defined for each user story and sometimes the test criteria are recorded on physical story cards "[we] put the acceptance criteria on the story itself" (Scrum Master, Company F). Unit testing is conducted to improve the quality of software produced by a team, "When you think you've got every logical point accomplished on a story, you would actually run test cases to make sure they are all fine" (Development Manager, Company C). The feature code can then be uploaded to a shared repository, "when you've got every logical point accomplished on a story, you've run test cases to make sure they are all fine, and then you check-in the story [to the shared repository]" (Development Manager, Company C).

In summary, feature artefacts comprise user stories which are produced and prioritised by product owners. Then low level design and source code development for the story is conducted. Subsequently, source code is compiled into feature code binaries, unit tests are executed and the code binary is uploaded to a repository.

5. Discussion

The application of agile methods in large-scale and geographically distributed software development programmes is an area of emerging interest [7]. Whereas agile early adopters tended to use engineering practices from the XP process [1], this research confirms more recent findings showing an increase in the popularity of scrum process orchestration practices [13, 49].

Previous research has focused on specific artefacts, such as the story card and wall [40], or on artefacts developed during a specific development phase, such as requirements engineering [39]. Others have explored the evolution of specific artefacts during the development process [50]. In contrast, the research presented here focuses on artefacts across the development life cycle but has only investigated large-scale agile development programmes. It is argued that the artefacts used in large-scale agile development programmes are a superset of those used in smaller projects.

Artefacts in software engineering are a locus of tension and controversy between production of working code and documentation for record keeping when

scaling agile methods to large development programmes. On one hand the agile community deprecates various forms of documentation in favour of working code [12] and yet larger scale projects require coordination between separate, yet concurrently working, development teams [9]. Further, high maturity software quality assurance processes, such as used by companies with CMMI® maturity level 5 accreditation, are often perceived to advocate detailed record keeping. Agile methods are thought by practitioners to generally discourage artefact production. While plan-based methods, encourage artefact production, for review and dissemination, to externally demonstrate high levels of quality assurance. Hence, this study focuses on artefacts to shed light on the tailoring of agile methods in a large-scale software development programme context. The interest here then is in practitioner interactions with artefacts and not in the artefacts per se.

> RQ1 "how do practitioners describe the inventory of artefacts they use in large-scale offshore software development programmes?"

The findings presented here confirm earlier research [37, 38] categorising artefacts in terms of planning (e.g. test plans, sprint plans and release plans), requirements (e.g. product backlog and sprint backlog), development (e.g. feature source code and feature code binaries), testing (e.g. unit tests and regression tests) and change management. However, the findings presented here identify additional artefacts which do not neatly fit into these categories, such as risk assessments and programme architecture standards. This suggests the need for a new category of agile 'governance' artefacts in large-scale software development programmes.

In addition, previous work [39] on requirements artefacts identifies: business risk analysis (risk assessment), release plans, test criteria which are confirmed by the findings presented here. However, agile requirements artefacts, such as product and sprint backlogs, are missing from that study.

The development teams in this study have recognised the value of physical artefacts, such as story cards and the kanban board [40], however, the geographical distribution of team members has more commonly necessitated their implementation using software tools.

Meyer lists impediments as artefacts [51], but the practitioners in this study did not seem to perceive impediments in exactly this way, instead viewing impediments more commonly as information that is available outside the team, or decisions from client representatives or other stakeholders, but which are required by team members in order to make progress with their work.

> RQ2 "How do the artefacts map to software development processes used in large-scale offshore software development programmes?"

Five categories of artefact have emerged from the empirical data in this study: feature, sprint, release, product and development programme governance. The feature and sprint categories are standard in agile methods. However, the artefacts in the release, product and programme governance categories are less frequently discussed in the agile literature. The artefacts identified in this study

form a taxonomy that can be mapped to the software development process. The development processes represented in Figure 1 comprise sprint, product release and development programme layers. Since features are produced within sprints and products are comprised of releases, we can map artefacts to the scrum of scrums process, as shown in Figure 3.

Several artefacts identified in this study, such as reference architectures, architecture implementation and release plans provide a means for communication between cooperating scrum teams [8, 9]. However, there appears to be a paucity of agile ceremonies to accompany the artefacts shared by cooperating scrum teams, the management of shared artefacts is not clearly defined, and currently agile teams improvise to fill this gap. The absence of agile ceremonies for managing shared artefacts presents a challenge for practitioners. New agile ceremonies to support artefacts shared across cooperating scrum teams, such as risk assessment, architecture standards and reference architectures should become a routine part of the scrum of scrums process.

The reference architecture artefact identified here is an example of a core asset from agile software product line engineering [52]. In a sense, the reference architecture is a constraint on the freedom of self-organising development teams to develop their own architecture. However, on large-scale development programmes architecture is a core asset to ensure inter-operability of software sub-systems and enable comprehensibility.

Four main themes have been identified for large-scale agile development programmes: scaling, portfolio management, inter-team coordination and architecture [18]. There is a paucity of agile artefacts for scaling, the teams in this study used the scrum of scrums meetings to identify and resolve dependencies between teams, but this seemed to be a recurring challenge. In this study, there is comparatively little evidence of interaction between projects and the central release plan used for portfolio management. In terms of inter-team coordination, this study found no evidence of artefacts for establishing shared values between teams. This study found strong evidence for the use of architecture standards, reference architectures and product architecture implementations.

> RQ3 "How do these practitioner descriptions contribute to our understanding of artefacts in agile method tailoring in large-scale offshore software development programmes?"

Taken together, the five artefact categories: feature, sprint, release, product and development programme governance, provide evidence of agile method tailoring in large-scale offshore software development programmes. This study confirms other recent studies showing that the method engineering approach [26] is more popular with project teams [28].

Using the study data and these sources, the artefacts identified can be mapped to agile roles and the activities within roles and is shown in Table 3. This tentative, and somewhat simplified mapping of artefacts to activities helps to lay the basis for integrating the expanded artefact set into a large-scale agile development programme. This could form the basis for developing new

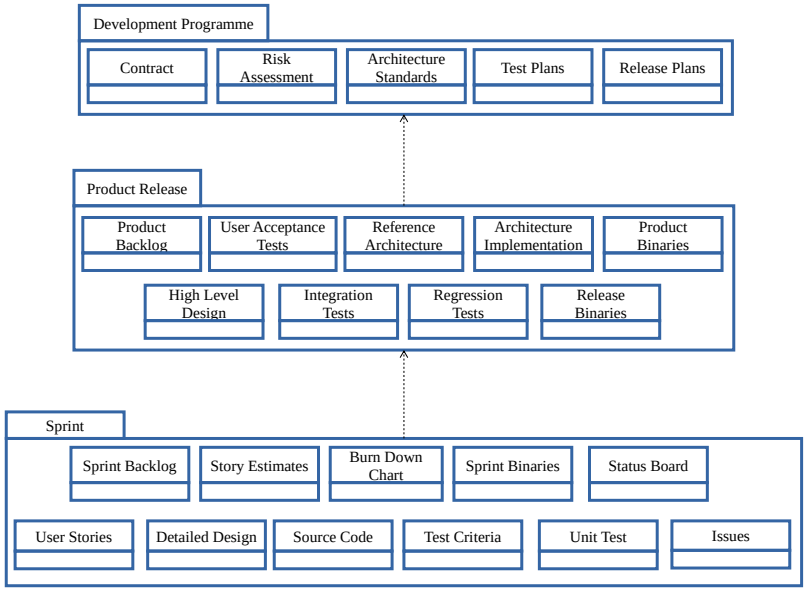

Figure 3: Mapping of Selected Artefacts to Scrum of Scrum Process

agile ceremonies to support the creation and management of the full artefact set.

As already mentioned, the artefacts identified in this study act as boundary objects enabling communication between diverse project stakeholders [53]. Boundary objects play an important role in knowledge sharing in cross-cultural software development teams [54]. Boundary objects can be identified at three levels: syntactical, semantic and pragmatic [11]. Syntactic boundary objects enable simple transfer of knowledge between distinct technical specialisms or across organisational boundaries, such as burn down charts and status boards. Semantic boundary objects enable translation or interpretation of information across the boundary, such as user stories and test criteria. While, pragmatic boundary objects require information transformation or negotiation between competing interests, such as product backlogs and risk assessments.

The increased number of stakeholders and the complexity of their intra- and inter-organisational relationships may encourage the proliferation artefacts in large-scale offshore software development programmes. It has been suggested that the process of constructing boundary objects helps the formation of shared organisational identities [55]. It has also been shown that boundary object use reduces conflict duration and reduced the time to identify and resolve conflict in global virtual teams [56]. Thus, stakeholders may be using creation of boundary objects to mitigate conflict and diffuse tension around potential areas of controversy in the development programme.

Table 3: Artefact Mapping to Agile Role Activities

Artefact	Creation Activity (within Primary Role)	Information Sources	Primary Information Consumer
Product Backlog	Product Sponsor	Market, Existing Clients Potential Clients	Development Team
Release Plan	Proxy Product Owner	Product Sponsor	Development Team
User Stories	Proxy Product Owner	Product Sponsor	Development Team
Reference Architecture	Technical Product Owner	Technology Trends Product Backlog	Development Team
Architecture Standards	Technical Product Owner	Technology Trends	Development Team
Risk Assessment	Proxy Product Owner	Contract	Product Sponsor
Contract	Product Sponsor	Product Backlog, Client	Proxy Product Owner
Test Criteria	Proxy Product Owner	Client	Development Team
Burn Down Chart	Scrum Master	Feature Binaries, Unit Tests	Proxy Product Owner
Status Board	Scrum Master	Feature Binaries, Unit Tests	Development Team Proxy Product Owner Product Sponsor
Sprint Backlog	Scrum Master	Product Backlog	Development Team
Product Binaries	Development Team	Release Binaries	Development Team
Test Plans	Tester	Product Backlog, Test Criteria Architecture Implementation	Development Team
Integration Tests	Tester	Sprint Binaries, Test Criteria	Scrum Master
Regression Tests	Tester	Sprint Binaries, Test Criteria	Product Owner
Release Binaries	Development Team	Sprint Binaries	Clients, Product Sponsor
Sprint Binaries	Development Team	Source Code	Development Team
Unit Tests	Tester	User Story, Test Criteria	Development Team
Issues	Tester	Unit Tests, Integration Tests Regression Tests, UATs	Development Team
User Story Estimates	Development Team	User Stories, Architecture Implementation	Proxy Product Owner
Detailed Design	Development Team	User Stories, Architecture Implementation	Development Team
Source Code	Development Team	Detailed Design, User Stories	Development Team
Feature Binaries	Development Team	Source Code	Proxy Product Owner

5.1. Limitations

Four important tests for evaluating the quality of descriptive empirical social research, from a positivist standpoint, are: objectivity, reliability, internal validity and external validity [57, 48]. However, Adolph et al. [41] draw on Lincoln and Guba [58] to establish corresponding criteria that are more appropriate for an interpretive stance: confirmability, dependability, internal consistency and transferability. These positivist and interpretive criteria have been contrasted in [59]. In this view, the positivist and interpretive quality criteria are intellectually incommensurable [59, Pg. 91]. In contrast, [43] adopt a stance they describe as "critical realist", preferring to avoid "discussing how 'goodness criteria' flow from epistemological positions" [43, Pg. 277] and grouping the issues together for consideration in the following categories.

5.1.1. Objectivity/Confirmability

In seeking to demonstrate that the findings of the research are representative, the research questions, research sites, data collection from documentary sources and practitioner interviews as well as the use of 'Glaserian' grounded theory such as open coding, memoing, constant comparison and saturation have all been described in some detail. This is intended to show the actual sequence of steps used in the conduct of the research. A potential weakness of this work is that it has been conducted by a sole researcher. As a researcher with prior first hand experience of both large-scale plan-based and small team agile methods, I was deeply curious to objectively understand how these large-scale teams select artefacts from these methods in their development programmes. In consequence, efforts have been made to objectively present the contrasting approaches of the development programmes investigated.

5.1.2. Reliability/Dependability

The goal of reproducible research has been pursued by collecting data across a wide range of appropriate settings and respondents and also by demonstrating meaningful parallelism across sources. Conducting the study in multiple organisations helps avoid bias from respondents giving monolithic and uniform answers, perhaps due to management pressure. Making the research sites anonymous is intended to avoid bias from respondents acting in pursuit of some perceived marketing objectives. Thus, the reliability of categories in this research has been sought not only by gaining the perspective of different actors within development programme teams and their surrounding organisational contexts but also by interviewing actors from different organisations. The risk of researcher bias has been minimised by obtaining feedback on early drafts of this article from key informants. This allowed validation of key findings from the research participant perspective.

5.1.3. Internal Validity/Internal Consistency

Here I want to establish that the research findings are credible, consistent and 'truthful.' The research is intended to present a descriptive understanding of the

artefacts selected in large-scale offshore software development programmes. It is intended that the account of findings presented is sufficiently comprehensive, coherent and authentic to make that case.

It was necessary to establish that the development teams were actually using agile methods and not simply deviating from text book methods through ignorance or due to organisational constraints. This was achieved through comparison with non-agile teams from Companies A and E, which were using RUP processes. Further dimensions of comparison were provided by Company C which was using XP rather than the more popular Scrum approach used in the other development programmes and Company B were senior management were promoting enterprise-wide adoption of scrum.

To maximise internal credibility sources of evidence have been used, including observation, documentary sources as well as practitioner interviews through conducting studies at nine different companies. A combination of snowball and intensity sampling was used, with snowball sampling providing data from a wide selection of projects, while the intensity sampling adopted at Companies D, E and H allowed for in-depth contact with a range of respondents, including corporate executives, project portfolio managers, project managers and various development team members. This intensity sampling provided varied sources of evidence and different perspectives on the software development processes used.

5.1.4. External Validity/Transferability

The purpose of this study has been to explore artefacts within large-scale offshore software development programmes, which operate within a particular climate of commercial pressures and quality assurance constraints. The study attempts to show that governance and architecture artefacts are required to meet these specific pressures and constraints. Within the large-scale offshore software development programme context, efforts to enhance transferability have been made by conducting the studies across nine companies and by gaining access to a wide range of project stakeholder respondents.

There has been no attempt to collect data relating to small and medium sized software development projects. Similarly the artefacts used in entirely co-located projects have not been explored. This contrasts with the approach taken in large scale surveys such as [60] which conflates findings from these different contexts in order to draw general conclusions. As a consequence of the research design employed, it is not appropriate to attempt to generalise the findings and conclusions presented here to small and medium sized co-located projects.

The intention of the article is to demonstrate a chain of evidence from the actual artefact inventories used in large-scale offshore development programmes to the findings, conclusions and contribution of the research. The interview extracts from respondents are intended to provide authentic evidence of current practice.

Companies B, F and I operate an in-house offshore development model, whereas the other companies are engaged in outsourced offshore. This might suggest that the data set is somewhat skewed towards outsourcing.

6. Conclusions

This article uses a grounded theory approach to investigate artefact inventories used in large-scale offshore software development programmes, using qualitative analysis of practitioner interview transcripts and supported by documentary sources and workplace observations. The study has focused on a sample of nine international companies and interviews with 45 practitioners. The grounded theory concepts of open coding, memoing, constant comparison and saturation were used to analyse the data that was triangulated across the selected companies as well as within project teams and corporate-level executives.

The research contribution of this study lies in its practitioner descriptions of the 25 artefacts created and managed in large-scale offshore agile software development programmes. This research confirms that the large-scale offshore software development programmes in this study use a range of conventional agile process artefacts. Agile artefacts are: user stories, issues (new feature requests, feature enhancements or defects), detailed designs, test criteria, unit tests, feature source code, sprint backlogs, story point estimates, burn down charts, status boards and the sprint code binaries. The artefacts used confirm that the teams have adopted elements of agile culture, and use agile process ceremonies such as daily review (scrum) meetings, collaborative work estimations and empirical measures of progress. However, a large, physically visible chart showing project status, regularly updated to show progress was used in few projects, with geographically distributed project teams more likely to use software tools to manage a 'virtual' status board.

A further contribution of the article is to provide a mapping of artefacts to a scrum of scrums software development process. At the release and product level artefacts identified in this study are: release plans, integration tests, regression tests, binary code releases, product backlog, user acceptance tests, reference architectures, product architecture implementation and product code binaries. While at the development programme level, artefacts used are: contracts, risk assessments, architecture standards, test plans and release plans.

The teams in this study skilfully blend agile artefacts with conventional plan-based artefacts such as reference architectures, risk assessments, regression tests, architecture standards, third-party user acceptance tests, release plans and product releases. Some of these plan-based artefacts reflect the need for co-ordination among software development teams working on the same product or aspects of the same development programme. Several of the project teams were using a software product line model in which releases contributed towards products which in turn contribute toward the overall development programme. However, other artefacts reflect quality assurance and governance culture among diverse development programmes occurring within the same organisation.

Thus, cooperating agile teams need shared artefacts such as: product backlog, reference architecture, architecture implementation, risk assessment, architecture standards and test plans in order to coordinate their development activity. However, currently there are no agile ceremonies specifically designed to create and refine any of these artefacts. The teams in this study improvise,

using programme governance groups, product owner teams or scrum of scrums meetings to overcome these challenges to the expansion of agile methods to large-scale offshore software development programmes. It is suggested that new agile ceremonies are needed, which could be used to create, review and refine these artefacts in keeping with the iterative and incremental ethos of the agile programmes investigated.

There is a risk that additional artefacts and new ceremonies detract from a healthy focus on producing working code. However, the additional artefacts are needed to provide governance oversight, coordinate the activities of cooperating teams, ensure a consistent approach is taken in areas such as software structure and testing, support correct sequencing where were there are dependencies and avoid duplication of effort. Further research is needed to identify a minimum set of additional artefacts required for this domain of software development practice.

Mapping the artefacts to agile job roles helps to identify actors with primary responsibility for creating and refining the artefacts. Drawing on other studies, it is possible to identify activities conducted within job roles [22, 16, 20] and map the artefacts to these specific activities. Mapping artefacts to activities provides a tentative first step toward defining new agile ceremonies for creating and managing additional artefacts required on large-scale offshore software development programmes. The mapping can also be used for staff training and development to disseminate best practice.

7. Acknowledgements

I am grateful to the companies and interviewees who participated in this research. Thanks also go to the students of the Executive MBA at the Indian Institute of Management, Bangalore; who facilitated access to several participating companies. The International Institute for IT, Bangalore provided hospitality during several research visits. The research benefited in part from travel funding from the UK Deputy High Commission, Bangalore; Science and Innovation Network; and Robert Gordon University, Aberdeen, UK. Accommodation and sustenance was provided by Company H during the data collection visit to Delhi, India.

References

[1] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: A systematic review, Information and Software Technology 50 (910) (2008) 833–859. doi:10.1016/j.infsof.2008.01.006.
URL http://www.sciencedirect.com/science/article/pii/S0950584908000256

[2] T. Dybå, T. Dingsøyr, What do we know about agile software development?, IEEE Software 26 (5) (2009) 6–9. doi:10.1109/MS.2009.145.

[3] D. Leffingwell, Scaling software agility: Best practices for large enterprises, Addison Wesley, Boston, MA, USA, 2007.

[4] C. Larman, B. Vodde, Scaling Lean and Agile Development, Addison Wesley, Upper Saddle River, NJ, USA, 2008.

[5] S. W. Ambler, Agile software development at scale, in: B. Meyer, J. Nawrocki, B. Walter (Eds.), Balancing agility and formalism in software engineering, Vol. 5082 of lecture notes in computer science, Springer Berlin Heidelberg, 2008, pp. 1–12. doi:10.1007/978-3-540-85279-7_1.

[6] S. Freudenberg, H. Sharp, The top 10 burning research questions from practitioners, Software, IEEE 27 (5) (2010) 8–9. doi:10.1109/MS.2010.129.

[7] T. Dingsøyr, N. B. Moe, Research challenges in large-scale agile software development, SIGSOFT Softw. Eng. Notes 38 (5) (2013) 38–39. doi:10.1145/2507288.2507322.
URL http://doi.acm.org/10.1145/2507288.2507322

[8] J. Vlietland, H. van Vliet, Towards a governance framework for chains of scrum teams, Information and Software Technology 57 (0) (2015) 52 – 65. doi:10.1016/j.infsof.2014.08.008.
URL http://www.sciencedirect.com/science/article/pii/S0950584914001992

[9] J. Vlietland, R. van Solingen, H. van Vliet, Aligning codependent Scrum teams to enable fast business value delivery: A governance framework and set of intervention actions, Journal of Systems and Software 113 (2016) 418–429. doi:10.1016/j.jss.2015.11.010.
URL http://www.sciencedirect.com/science/article/pii/S0164121215002435

[10] S. Sahay, B. Nicholson, S. Krishna, Global IT Outsourcing: Software Development across Borders, 1st Edition, Cambridge University Press, 2003.

[11] P. R. Carlile, A pragmatic view of knowledge and boundaries: Boundary objects in new product development, Organization Science 13 (4) (2002) 442–455. doi:10.1287/orsc.13.4.442.2953.
URL http://dx.doi.org/10.1287/orsc.13.4.442.2953

[12] Kent Beck, James Grenning, Robert C. Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, Andrew Hunt, Ken Schwaber, Manifesto for Agile Software Development (2001).
URL http://agilemanifesto.org/

[13] Version One, The 9th Annual State of Agile Survey, http://stateofagile.com/, [accessed 25-06-2015].

[14] T. Dingsøyr, S. Nerur, V. Balijepally, N. B. Moe, A decade of agile methodologies: Towards explaining agile software development, Journal of Systems and Software 85 (6) (2012) 1213 – 1221, special Issue: Agile Development. doi:10.1016/j.jss.2012.02.033.
URL http://www.sciencedirect.com/science/article/pii/S0164121212000532

[15] V. Heikkilä, M. Paasivaara, C. Lassenius, C. Engblom, Continuous release planning in a large-scale scrum development organization at ericsson, in: H. Baumeister, B. Weber (Eds.), Agile Processes in Software Engineering and Extreme Programming, Vol. 149 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2013, pp. 195–209. doi:10.1007/978-3-642-38314-4_14.
URL http://dx.doi.org/10.1007/978-3-642-38314-4_14

[16] R. Hoda, J. Noble, S. Marshall, Self-organizing roles on agile software development teams, IEEE Transactions on Software Engineering 39 (3) (2013) 422–444. doi:10.1109/TSE.2012.30.

[17] A. Martin, The role of the customer in agile projects, PhD thesis, Victoria University of Wellington, New Zealand (2009).

[18] T. Dingsøyr, N. Moe, Towards principles of large-scale agile development, in: T. Dingsøyr, N. Moe, R. Tonelli, S. Counsell, C. Gencel, K. Petersen (Eds.), Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation, Vol. 199 of Lecture Notes in Business Information Processing, Springer International Publishing, 2014, pp. 1–8. doi:10.1007/978-3-319-14358-3_1.
URL http://dx.doi.org/10.1007/978-3-319-14358-3_1

[19] Scaled Agile Framework.
URL http://www.scaledagileframework.com/

[20] J. M. Bass, Scrum master activities: Process tailoring in large enterprise projects, in: Global Software Engineering (ICGSE), 2014 IEEE 9th International Conference on, 2014, pp. 6 – 15. doi:10.1109/ICGSE.2014.24.

[21] M. Paasivaara, C. Lassenius, V. T. Heikkilä, Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work?, in: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement, ESEM '12, ACM, New York, NY, USA, 2012, pp. 235–238. doi:10.1145/2372251.2372294.

[22] J. M. Bass, How product owner teams scale agile methods to large distributed enterprises, Empirical Software Engineering 20 (6) (2015) 1525 – 1557. doi:10.1007/s10664-014-9322-z.
URL http://dx.doi.org/10.1007/s10664-014-9322-z

[23] J. Eckstein, Architecture in large scale agile development, in: T. Dingsøyr, N. Moe, R. Tonelli, S. Counsell, C. Gencel, K. Petersen (Eds.), Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation, Vol. 199 of Lecture Notes in Business Information Processing, Springer International Publishing, 2014, pp. 21–29. doi:10.1007/978-3-319-14358-3_3.
URL http://dx.doi.org/10.1007/978-3-319-14358-3_3

[24] C. Larman, B. Vodde, Practices for Scaling Lean and Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum, Addison Wesley, Upper Saddle River, NJ, 2010.

[25] K. Conboy, B. Fitzgerald, Method and developer characteristics for effective agile method tailoring: A study of xp expert opinion, ACM Trans. Softw. Eng. Methodol. 20 (1) (2010) 2:1–2:30. doi:10.1145/1767751.1767753.
URL http://doi.acm.org.salford.idm.oclc.org/10.1145/1767751.1767753

[26] S. Brinkkemper, Method engineering: engineering of information systems development methods and tools, Information and Software Technology 38 (4) (1996) 275 – 280, method Engineering and Meta-Modelling. doi:http://dx.doi.org/10.1016/0950-5849(95)01059-9.
URL http://www.sciencedirect.com/science/article/pii/0950584995010599

[27] B. Fitzgerald, G. Hartnett, K. Conboy, Customising agile methods to software practices at Intel Shannon, European Journal of Information Systems 15 (2) (2006) 200–213. doi:10.1057/palgrave.ejis.3000605.
URL http://dx.doi.org/10.1057/palgrave.ejis.3000605

[28] A. S. Campanelli, F. S. Parreiras, Agile methods tailoring a systematic literature review, Journal of Systems and Software 110 (2015) 85 – 100. doi:http://dx.doi.org/10.1016/j.jss.2015.08.035.
URL http://www.sciencedirect.com/science/article/pii/S0164121215001843

[29] G. Kalus, M. Kuhrmann, Criteria for Software Process Tailoring: A Systematic Review, in: Proceedings of the 2013 International Conference on Software and System Process, ICSSP 2013, ACM, New York, NY, USA, 2013, pp. 171–180. doi:10.1145/2486046.2486078.
URL http://doi.acm.org/10.1145/2486046.2486078

[30] K. M. Lui, K. C. C. Chan, J. Nosek, The effect of pairs in program design tasks, IEEE Transactions on Software Engineering 34 (2) (2008) 197–211. doi:10.1109/TSE.2007.70755.

[31] V. Balijepally, R. Mahapatra, S. Nerur, K. H. Price, Are two heads better than one for software development? The productivity paradox of pair programming, MIS Quarterly 33 (1) (2009) 91 – 118.

[32] J. E. Hannay, E. Arisholm, H. Engvik, D. I. K. Sjoberg, Effects of personality on pair programming, IEEE Transactions on Software Engineering 36 (1) (2010) 61 – 80. doi:10.1109/TSE.2009.41.

[33] J. W. Wilkerson, J. F. Nunamaker, R. Mercer, Comparing the defect reduction benefits of code inspection and test-driven development, IEEE Transactions on Software Engineering 38 (3) (2012) 547–560. doi:10.1109/TSE.2011.46.

[34] M. A. Cusumano, Extreme programming compared with Microsoft-style iterative development, Commun. ACM 50 (10) (2007) 15–18. doi:10.1145/1290958.1290979.

[35] T. Mens, T. Tourwe, A survey of software refactoring, Software Engineering, IEEE Transactions on 30 (2) (2004) 126–139. doi:10.1109/TSE.2004.1265817.

[36] H. Sharp, H. Robinson, An ethnographic study of XP practice, Empirical Software Engineering 9 (4) (2004) 353–375. doi:10.1023/B:EMSE.0000039884.79385.54.

[37] M. Kuhrmann, D. Mendez Fernandez, M. Grober, Towards artifact models as process interfaces in distributed software projects, in: Global Software Engineering (ICGSE), 2013 IEEE 8th International Conference on, 2013, pp. 11–20. doi:10.1109/ICGSE.2013.11.

[38] H. Femmer, M. Kuhrmann, J. Stimmer, J. Junge, Experiences from the Design of an Artifact Model for Distributed Agile Project Management, in: 2014 IEEE 9th International Conference on Global Software Engineering (ICGSE), 2014, pp. 1–5. doi:10.1109/ICGSE.2014.9.

[39] D. Méndez Fernández, S. Wagner, K. Lochmann, A. Baumann, H. de Carne, Field study on requirements engineering: Investigation of artefacts, project parameters, and execution strategies, Information and Software Technology 54 (2) (2012) 162–178. doi:10.1016/j.infsof.2011.09.001.
URL http://www.sciencedirect.com/science/article/pii/S0950584911001820

[40] H. Sharp, H. Robinson, M. Petre, The role of physical artefacts in agile software development: Two complementary perspectives, Interacting with Computers 21 (1) (2009) 108–116.

[41] S. Adolph, W. Hall, P. Kruchten, Using grounded theory to study the experience of software development, Empirical Software Engineering 16 (4) (2011) 487–513. doi:10.1007/s10664-010-9152-6.

[42] M. Q. Patton, Qualitative Research & Evaluation Methods, 3rd Edition, Sage Publications, Inc, Thousand Oaks, CA, USA, 2002.

[43] M. B. Miles, A. M. Huberman, Qualitative Data Analysis: An Expanded Sourcebook, 2nd Edition, Sage Publications, Inc, Thousand Oaks, CA, USA, 1994.

[44] NVivo 9 help, http://help-nv9-en.qsrinternational.com/nv9_help.htm, [accessed 10-09-2013].

[45] B. G. Glaser, A. L. Strauss, Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine, Chicago, IL., USA, 1967.

[46] B. G. Glaser, Doing Grounded Theory: Issues and Discussions, Sociology Press, Mill Valley, USA, 1998.

[47] B. G. Glaser, Basics of Grounded Theory Analysis: Emergence vs. Forcing, Sociology Press, Mill Valley, USA, 1992.

[48] C. Robson, Real World Research, 3rd Edition, John Wiley and Sons Ltd., Chichester, UK, 2011.

[49] J. M. Bass, Influences on Agile Practice Tailoring in Enterprise Software Development, in: Agile India, IEEE, Bangalore, India, 2012, pp. 1 – 9. doi:10.1109/AgileIndia.2012.15.
URL http://doi.ieeecomputersociety.org/10.1109/AgileIndia.2012.15

[50] Z. Xing, E. Stroulia, Analyzing the evolutionary history of the logical design of object-oriented software, IEEE Transactions on Software Engineering 31 (10) (2005) 850–868. doi:10.1109/TSE.2005.106.

[51] B. Meyer, Agile!: The Good, the Hype and the Ugly, Springer, New York, USA, 2014.

[52] J. Díaz, J. Pérez, P. P. Alarcón, J. Garbajosa, Agile product line engineeringa systematic literature review, Software: Practice and Experience 41 (8) (2011) 921–941. doi:10.1002/spe.1087.
URL http://dx.doi.org/10.1002/spe.1087

[53] S. L. Star, J. R. Griesemer, Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39, Social Studies of Science 19 (3) (1989) 387–420. doi:10.1177/030631289019003001.
URL http://sss.sagepub.com/content/19/3/387.abstract

[54] M. Barrett, E. Oborn, Boundary object use in cross-cultural software development teams, Human Relations 63 (8) (2010) 1199–1221. doi:10.1177/0018726709355657.
URL http://hum.sagepub.com/content/63/8/1199.abstract

[55] U. Gal, K. Lyytinen, Y. Yoo, The Dynamics of IT Boundary Objects, Information Infrastructures, and Organisational Identities: The Introduction of 3d Modelling Technologies into the Architecture, Engineering, and Construction Industry, European Journal of Information Systems 17 (3) (2008) 290–304. doi:10.1057/ejis.2008.13.
URL http://dx.doi.org/10.1057/ejis.2008.13

[56] J. Iorio, J. E. Taylor, Boundary object efficacy: The mediating role of boundary objects on task conflict in global virtual project networks, International Journal of Project Management 32 (1) (2014) 7–17. doi:10.1016/j.ijproman.2013.04.001.
URL http://www.sciencedirect.com/science/article/pii/S0263786313000409

[57] R. K. Yin, Case Study Research: Design and Methods, 4th Edition, Sage Publications, Inc, Thousand Oaks, CA, USA, 2009.

[58] Y. S. Lincoln, E. G. Guba, Naturalistic Inquiry, 1st Edition, SAGE Publications, Inc, Beverly Hills, Calif, 1985.

[59] S. Gasson, Rigor in Grounded Theory Research: An Interpretive Perspective on Generating Theory from Qualitative Field Studies, in: M. Whitman, A. Woszczynski (Eds.), The Handbook of Information Systems Research, Hershey, PA, USA, 2004, pp. 79–102.
URL http://www.igi-global.com/chapter/rigor-grounded-theory-research/30344

[60] S. de Cesare, M. Lycett, R. D. Macredie, C. Patel, R. Paul, Examining perceptions of agility in software development practice, Commun. ACM 53 (6) (2010) 126–130. doi:10.1145/1743546.1743580.

Appendix

The interviews were conducted using an open-ended, semi-structured, interview guide approach. There was some modification of the interview guide as the study evolved. An example recent interview guide is provided here.

Interview Guide, Agile Method Tailoring

Agile Processes

- What agile methods and practices are you using?
- Would you describe agile methods as being successful for you? In what ways?
- What challenges have you encountered with agile methods?

Scaling to Enterprise Projects

- Describe any software tools or technologies that you use to support agile methods?
- Have you adapted agile methods because of the geographical distribution of the team?
- Have you adapted agile methods because the client organisation was geographically distributed?
- Have you adapted agile methods because of a particularly large team?
- Have you used agile methods in a context with demanding regulatory compliance? What adaptations did you make?
- Have you used agile methods in a particularly complex domain context? What adaptations did you make?
- Have you used agile methods on a particularly technically complex project? What adaptations did you make?
- Have you used agile methods with an especially complex range of stakeholder relationships?
- What adaptations did you make?
- Have you adapted agile methods for use on a strategically important enterprise architecture programme?

Social Media/Cloud

- What forms of social media or electronic communication are used in the projects?
- What forms of cloud computing services are used in the projects?

Learning

- What do you think about learning, particularly in the offshore situation? How do you distribute knowledge and skills around the team?

Future Perspectives

- What future trends do you foresee in your use of agile methods?
- If there was one thing you could change about the way agile methods are used at [Company H] what would it be?
- What advice would you give to improve transitioning to offshore agile?

Any other comments

- Do you have any further comments on agile methods?

About Your Project(s)

Now I want to ask some questions about you and your project. These details will be kept confidential.

- What project are you working on currently? How many projects?
- How is the project team structured (for management purposes)?
- How is the project team organised geographically?
- What is the project domain? What is the project purpose?
- How large is the project in terms of team size? In terms of value?