J Braz Comput Soc (2013) 19:257–274
DOI 10.1007/s13173-012-0096-0

ORIGINAL PAPER

Scrum adoption and architectural extensions in developing new service applications of large financial IT systems

Tuomas Ihme

Received: 7 February 2012 / Accepted: 3 December 2012 / Published online: 29 December 2012

Abstract The use of modern agile software development methods in large organisations requires tailoring agile development to the organisations' needs. This study concentrated on studying integrating software product line and agile application development in the context of large and complex financial IT systems. The study was conducted by interviewing a wide representation from the case organisation's personnel developing in-house software for the company's own use. The results indicate that the guidelines of the Scrum method for agile project management are valid also for the studied company type. However, each project in the studied company should acknowledge the constraints set by other projects and the complex technical infrastructure, data security issues and system portfolio in the organisation's product line platform. In order to promote re-use and avoid risks, means and mechanisms are needed to coordinate and synchronise multiple projects and their releases on business level, the project's compatibility with its constraints on system architecture level and the direction of software architecture on project team level.

Keywords Software product line · Agile software development · Scrum · Software architecture · Qualitative study

1 Introduction

Experience from agile software development (ASD) in large organisations and domains such as automobiles, telecommunications, finance and medical devices shows the necessity of tailoring agile principles, methods and practices to the project's environment and organisational requirements [1–10]. A real obstacle for the adoption of ASD is agile methods not generally guiding architecting large-scale systems [11–13].

In large new product development organisations, the perspective has to be extended from individual release projects of most agile methods to longer-term product evolution and portfolio management and, when needed, to software product line (SPL) development [8,14]. Northrop [15] defines that "A software product line is a set of software-intensive systems that share a common, managed feature set satisfying a particular market segment's specific needs or mission and that are developed from a common set of core assets in a prescribed way." SPL practices have been used in a wide range of organisations and domains and the use has produced significant organisational advantages such as better quality, decreased cost, decreased labour needs, decreased time to market, and ability to move into new markets faster [16].

ASD's highest priority "is to satisfy the customer through early and continuous delivery of valuable software" (agile manifesto). The SPL approach's goals include reducing time to market, increasing productivity, improving quality, enhancing agility and gaining cost effectiveness and efficiency [8,17]. In addition to common goals, ASD and SPL have complementary properties [18]. Even though complementary, these two approaches are relatively independent of each other and have only a few conflicts [18]. The above mentioned give motivation for considering combining SPL engineering and ASD. However, combining may be difficult as the means of these approaches to achieve their goals can be contradictory [19,20]. ASD and SPL approaches have philosophical differences in design and change management strategies [19,21]. SPL engineering aims to develop software products using software platforms and mass customisation,

T. Ihme (✉)
VTT Technical Research Centre of Finland,
P. O. Box 1100, 90571 Oulu, Finland
e-mail: Tuomas.Ihme@vtt.fi

whereas ASD focuses on delivering single products for the customer [18]. Agile methods and particularly the fundamental eXtreme Programming (XP) process [22] does not explicitly support the development of artifacts for reuse [19,23]. Recently, there has been increased interest in exploring the possibilities of the benefits of combining ASD and SPL approaches [18–21,24–26].

The industrial case study of Petersen and Wohlin [27] indicates that moving from a plan-driven to an incremental software development approach with agile practices can provide added value. A survey of 72 IBM software developers suggests that agile developers consider software architectures as important and supportive as agile values [28]. Architecture plays a key role in SPL development [15]. Booch [29] argues that "all good software-intensive architectures are agile" and need to be socialised in the tribal memory of all stakeholders. However, more empirical research concerning the combination of agile and traditional methods is needed [30]. A general model of software architecture design reveals that incremental or ongoing architectural evaluation and analysis are important research topics [31]. The review of Dyba and Dingsoyr [11] reveals that current empirical studies still lack reports related to architectural design in ASD. Furthermore, ASD in large projects and the co-existence of ASD and architecture [3] in different problem classes are among the most burning research questions from practitioners [12].

The adoption of ASD in the banking sector can be nontrivial [32–34]. This study concentrated on studying integrating SPL and ASD approaches in the context of large financial IT systems. Thus, this study aims at answering the following research questions:

- RQ1: What type of challenges does a large financial IT systems company encounter in its software development?
- RQ2: What type of steps should a large company in the financial sector take while combining SPL and agile approaches in their software development?

In this study, for example, we found that interviewed architects, developers and decision makers recommended the following first priority things towards ASD in their organisation: removing organisational impediments to ASD, prioritising products and projects, setting prioritised requirements and milestones for the projects, directing software architecture development to support agile project realisation and acknowledging limitations set by technical infrastructure. The results of this study propose first steps for the case organisation for moving towards ASD. The proposed steps are synthesised based on analysing the interviewees' recommendations, their positive qualities of old practices and the limitations set by the organisation's technical infrastructure. The results of this study have implications on business, architecture and project team level management.

The study was conducted empirically by interviewing a wide representation from the personnel and stakeholders of complex software projects in a software development department of a large financial company. The interviews covered software development in a versatile manner, including business people, software architects, technical writers, software designers, test personnel, and other responsible managers and decision-makers. The research utilised a semi-structured thematic interview approach (see e.g. [35]). The interviews were conducted in a qualitative manner, allowing the interviewees to explain and clarify the cases and topics as entities. In addition to the interviews, separate data gathering workshops were organised for decision-makers and software engineering people and one feedback workshop for all interviewees and participants. As a secondary data source, the researchers also reviewed various project document examples.

The structure of the paper is as following. Section 2 provides an overview of related research on large financial IT systems and on integrating SPL and ASD approaches. Section 3 presents the research methodology and procedure. The findings are presented and discussed in Section 4. Section 5 discusses research contributions and managerial implications. Section 6 discusses the case study's limitations. Finally, Section 7 concludes the paper with final remarks.

2 Literature review

Large financial IT systems suffer from shortcomings and agility challenges such as duplicated business functionality across multiple sub- systems and tightly coupled parts of the infrastructure [34]. A lesson is that interpreting agile approaches in the specific organisational context of a large financial enterprise remains challenging [10]. The systems tend to include legacy subsystems and monolithic functionality which can be hard to decompose into small independent pieces for agile software production [32,33,36]. Management of large and mature banking-sector organisations sees big risks in the big-bang adoption of agility. ASD methods do not promote formal documentation that may be required for regulatory, company policy and maintenance reasons within the financial services community [37,38]. Augustine et al. [36] have developed six practices (XP practices) for managing large and mission-critical agile projects: small teams (seven to nine members), guiding vision, simple rules, free and open access to information, light management style and adaptive leadership. The practices do not address reuse or architecture.

Agile and SPL approaches promote collaboration [25]. Agile principles emphasise collaboration between customers and developers (agile manifesto), whereas SPL approaches expect collaboration between SPL asset builders and product developers [25]. Poort et al. [39] have discovered how

emotional and interpersonal relationship challenges such as conflicts, trust and willingness to share knowledge significantly correlate with successful architectural knowledge-sharing and project success. Agile principles even encourage changes in requirements. Variation points allow a SPL to accept unanticipated changes, in addition to anticipated changes in requirements [20,25]. The scope of a SPL defines those entities and projects that are within the context of a product line [15], whereas an agile project team works within an implicit project scope defined by the customer, yet the scope of each iteration cycle is explicit. Agile approaches emphasise the importance of producing working software early and frequently starting from the first iteration cycle [40]. Product development teams in a SPL produce working software early as well by assembling and configuring SPL assets [25].

In spite of the obvious synergies between SPL and agile approaches, their competing philosophies can make their integration difficult [19,25]. During SPL's product development activity, design is proactive and upfront-oriented when a product team creates product-specific solutions by specialising SPL assets, including a flexible, up-front designed SPL architecture for a family of products [41]. In contrast, ASD's emerging software-development thinking includes a repetitive emergent design process [42]. Thapparambil [43] claims that "Refactoring is the primary method to develop architecture in the agile world." The refactoring of complex systems and SPLs often leads to unexpected evolution conflicts [7,44]. Software architecture is a key factor in a SPL's success [45], whereas the XP [22] and Scrum [40] method books are known for paying very little explicit attention to software architecture and call for the non-consideration of architectural features that do not have immediate interest for the current iteration [18–20,46,47]. The agile approaches weakly prescribe holistic system wide analysis and design and the role of system-level architects [9,48]. Multiple issues should be addressed for reconciling apparent conflicts between ASD and architecture in a particular project or organisation [3]. How much architectural activity is needed in the project's context? What design decisions are architecturally significant? When those significant decisions should be made? Who owns architectural issues? What methods and practices will be used to design software architecture? How much of an explicit architectural description and documentation is needed? How to find a right balance between the costs of architecture and functionality?

Successful SPLs are possible both in large and small organisations and systems [20,15], whereas agile methods are just one subset of the means to achieve and improve agility in large-scale software product development e.g. [5,9,14,49,50]. Conboy et al. [51] argue that the agile principles and their method instantiations such as XP and Scrum are largely naive to the concept of agility in information systems development. In large development organisations, some amount of existing infrastructure will be necessary to minimise large scale refactoring and to help diverse teams build features and components that integrate easily [13]. Flexible software architecture and software platforms are among the key factors of the level of agility in large-scale agile new product development [14]. A hybrid project management approach [52] with both traditional and agile practices may be the most suitable for increasing adaptation and value generation [48] in large organisations and systems. Denne and Clealad-Huang [53] describe an approach that helps agile and traditional projects to determine which requirements should be implemented first and which platform and architectural elements are needed and when they are needed. They demonstrate their approach using an example project for the construction of a financial services portal.

Motorola's tailored version of XP [8] comprises the development of a coarse-grained baseline architecture that provides sufficient guidance to support the creation of SPL assets. Most agile methods weakly support cross-team communication problems. Nokia's solution to this problem is to minimise the need for the cross-team communication [8] between agile and non-agile teams. Different teams usually have their own interfaces with the SPL asset team. Thus the teams can decide which agile practices are suitable to their needs independently of each other. Lessons learned applying the Software Engineering Institute's (SEI) SPL approach indicate that development processes can be according to agile methods, but the used process must be defined and followed [15,54]. Mohan et al. [55] have created guidelines that help top management, project managers and developers make SPL practices more agile. The guidelines are based on the SEI's report of an industrial case study [54]. The findings from the recent industrial case study of Hanssen and Fægri [18] indicate that SPL engineering and ASD can complement each other in a medium-sized software company.

3 Research methodology and procedure

The research process is described in Fig. 1. ASD methods were first studied by using existing literature as the key source. The case company was selected as it represents the large financial enterprises with scarce agile research. The case company had challenges in their old software development practices, hence attempted to seek improvements through agile methods.

The case selected for this research is a software development department of a large financial enterprise. The department develops systems and applications for satisfying the banking, investment, loan and insurance service needs of

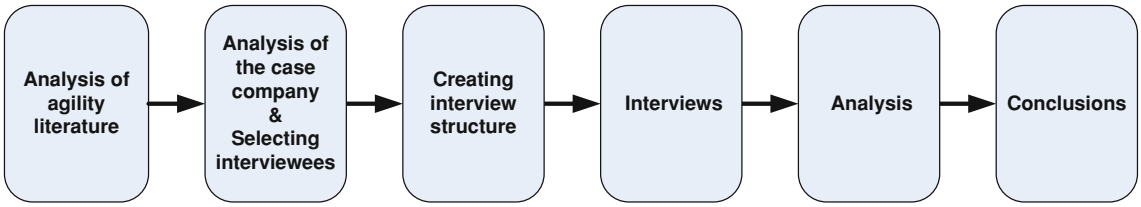

Fig. 1 The research process

the enterprise's different private and corporate customers. It uses SPL engineering for improving and maintaining the effectiveness and efficiency of its product development activities that lean on the reuse of assets and technologies of a complex SPL platform. The common nominator for development projects lies in the SPL platform's technical system infrastructure, multi-channel architecture and common reusable assets. In addition, the projects' business functionality typically penetrates across multiple systems in the case company's system portfolio. Products and applications are developed for the company's own use, not for sale or licensing. Software development is conducted mainly in-house using the company's own resources. Before, the department has developed software following plan-driven software development methods. Recently, they have carried out several pilot projects for capturing experiences on agile development methods.

The case company was examined in order to obtain an understanding of the company's operations. A kick off meeting was organised in the beginning together with a wide representation from the case company. This representation included business managers, software architects, software designers, test personnel, and other responsible managers. Interview questions were formulated based on the obtained understanding. The interview questions included common questions to all the interviewees and also focused questions for different specialist groups.

The prolonged involvement (a four-year period), triangulation (multiple data sources, multiple researchers, different data collection methods as well as interpretive, positivistic and critical research perspectives), member checking (feedback from interviewed persons), and audit trail (interviews were recorded and transcribed professionally) strategies [50] were used to reduce the validity threats of the study in terms of reactivity, researcher bias and respondent bias. The study consisted of 21 interviews, including 33 specialists from the case organisation's different areas and managerial levels. Table 1 shows a summary of the interviews. The interviewees included decision-makers, business people and software engineering people. Software engineering people were managers, architects, lead designers, technical writers, software designers and testers. Seven of these interview sessions concentrated on software architecture experts purposely selected to maximise diversity on architectural stakeholder concerns. The length of each interview was 1.5–2 h. The research utilised a semi-structured thematic interview approach (see e.g. [35]). The interviews were conducted informally, in a qualitative manner, allowing the interviewees to explain and clarify the cases and topics as entities. Interviews were recorded and transcribed resulting in some 20 to 30 pages for each interview. The researchers also took notes during the interview sessions. In addition to the interviews, separate data gathering workshops were organised for decision-makers and software engineering people (25 participants) and one feedback workshop for all interviewees and participants. Three managers were interviewed after several pilots during a three-year period. As a secondary data source, the researchers also reviewed various project document examples. Therefore, this research represents the software development activities in the case company in a versatile manner.

Each interview session started with a brief interviewee and researcher introductions. The introduction included the name of the interviewee or interviewees, their organisations, position, professional background, experience, and current work focus. The information management and communication model [56] was utilised by the interviewees in each interview session. The interviewees described their own tasks by using the model, including work inputs and outputs, weak points, unnecessary elements, rewards, means of communication, project's internal and external personnel, stakeholders, organisations, systems and documents used for communication. The interviewees were also asked for their opinions on the advantages, disadvantages, challenges and risks relating to the old software process as well as short-term and long-term needs and proposals for improving the process. The interviewees were asked about their know-how of ASD and their opinions on the potential possibilities, advantages and disadvantages of agile practices, principles and thinking in their work.

The source material was processed and further analysed in order to make conclusions. The results were categorised based on the case company's software process, business functions and software engineering. Software engineering was divided into further categories. The results of the analysis are presented in the next section.

Table 1 A summary of interviews

Aspect	Comment
An interview instrument (10 pages)	• Common questions to all the interviewees and focused questions for different specialist groups
21 interviews The length of each interview was 1.5–2 h	• 33 specialists from the case organisation's different areas at and the managerial levels • Two researchers interviewed one or two subjects • One subject was interviewed in each seven interview session of architects • The researchers also took notes during the interview sessions
Recording interviews	• Full recoding of interviews
Transcription of recordings	• Using a professional transcription company • Transcribed interview data in some 20 to 30 pages for each interview
Identifying, classifying and storing interesting quotes in tables	• Multiple researchers in quoting
Identifying specific results and conclusions	• Feedback from subjects before final results • The experts of the company reviewed final results • All researchers participated in the writing final results

4 Results and discussion

4.1 Current state analysis

4.1.1 Old practices described

Before transition steps towards agility, the case organisation has followed a sequential waterfall style development process in their software development, even though concept development is iterative. Their old software development process includes business pre-studies, idea crystallisation, concept development, software code design and testing, and piloting and introduction. Formal review checkpoints are applied between each development phase to assess obtained and documented results. These checkpoints also act as decision points for following phases.

The case organisation utilises SPL architectures that are divided into six main sub-areas: business architecture, information architecture, software architecture, integration architecture, platform technology architecture, and security architecture. Each of these sub-areas has a nominated architect team; however, a single architect can work for several architect teams.

The studied SPL in the case organisation aims to ensure that adequate solutions are available for projects, including both infrastructure and technical architecture techniques. A support system in the form of a database enables maximal utilisation of previously developed components, solutions, coding instructions and system-level documents. The case organisation obligates developers to utilise previously developed solutions and sub-systems when creating new products. This ensures production quality, user capacity, cost efficiency, scalability, and information and network security.

The existing SPL architecture does not encourage radical development ideas, but aims to utilise multi-channel architecture for applications enabling product creation for all target groups. Multi-channel architecture is seen as an adequate basis for effective application development. This practice is well established in the case organisation, even if it may not be the most modern one.

Initial description of a system product concept is rooted to project business goals described by the business unit. The description of a system product includes depictions of all the relevant interfaces and relevant sub-systems. The description is conducted by one or two lead designers who communicate and negotiate the technical aspects with numerous architectural experts and other specialists responsible for sub-systems.

In addition to developing assigned system products, projects also generate generic components that are available for other projects. Architecture guidelines aim to ensure that projects develop generic solutions and do not excessively create project-specific or overlapping solutions.

A structural description document for a product includes both functional and technical descriptions. This document acts as a medium, but more importantly is developed through different discussions and meetings. Based on the final reviewed version of this document, a project steering group decides on product realisation.

In order to avoid unwanted surprises, occasionally proof of concept prototypes are constructed and performance tests are carried out to confirm architecture functionality already

during concept development. In some cases, proof of concept prototypes are constructed even before actual requirements definition.

4.1.2 Current challenges

The case company representatives experience the old ways of working to contain many challenges, they also believe that agility could rectify some of the old issues. Different personnel groups experience the need for agility in different ways. Software designers believe that agile approach is beneficial, while architecture designers see agility to also contain risks and potentially challenging.

One of the main problems, highlighted by the interviewees, is the serial development model, without iterative feedback, allowing results of development projects to become obsolete, resulting in products not meeting expectations. In addition, freezing specifications too early, without feedback, causes problems, especially if project implementation is delayed. These problems are seen to be partially a result of weak communication and interaction between those responsible for business and application development. Project steering group is seen as the only true contact interface the business people have with development projects. Project steering meetings are the only times when the business group is informed on development projects on face-to-face basis, at other times; it is the documentation that acts as the communication medium. This results in the business group not having adequate understanding or visibility over the progress of development, making it difficult to make decisions in a timely manner. Developed applications may include unnecessary features, or even worse, miss features that are seen vital for business.

Software architects see the business goals in the documentations as unclear, lacking information and without adequate requirements prioritisation. Once the business group has accepted requirements specifications, the next time they provide feedback is typically only after project realisation. Persons defining features conduct their feature definitions from the business perspective, resulting in software designers having difficulties in understanding the features. Currently, development ideas of software developers are not adequately appreciated. Software designers would prefer the features to be defined from the perspective of component-based architecture, which however is not adequately conducted.

The interviewees also experience problems in communication and interaction within the development organisation, within different development phases and projects. Communication is strongly documentation-based, and it is seen very difficult to confirm correct realisation, should the written requirements be ambiguous. As the transition from a development phase to another is governed by documentation, parallel development is minimal, potentially extending project durations. The old ways of working are seen not to support co-learning and distributing best practices. Also, division of projects into smaller entities is governed by the organisational structure resulting in further communication challenges.

Increasing expenses is as a challenge, seen to be caused by inefficiency that is partially due to planning overly secure schedules. Initial estimated schedules and expenses are rarely reviewed after the beginning of a project. Projects are typically completed as planned, but with higher than expected expenses. According to some of the interviewees, loose schedules result in ineffective resource utilisation, increasing expenses. According to some of the interviewees, the indicators used to measure project success are not optimal as they mainly concentrate on keeping to schedules and budget instead of product success.

The case company's system portfolio is a very complex system of systems, making it virtually impossible for an individual to perceive big enough entities. The interviewees experience that there are not enough specialists who are capable of this type of perception. In addition, architecture designers experience the project diversity as a problem from the perspective of making architectural guidelines. Software designers are expected to produce other documentation, aside software code, further explaining the code. However, requirements for this additional documentation are neither clear, nor are its quality and necessity assessed.

4.1.3 Agility challenges

Experience from several pilot projects shows that the transition towards agility is not a single step, but a series of smaller ones. There are still many technical and managerial agility problems to solve. One pilot project typically solves some agility difficulties and highlights several new open problems for solving in successor pilots. Agile methods seem to be more suitable for new product development projects than maintenance projects. According to interviewed project managers and software engineering people in their workshop, organisation-wide agile transformation within the large case organisation is hard. A high turnover of agile project team members is a problem.

The very complex system nature of the studied company's operational environment hinders the realisation of single projects, including cost control and through times. New projects are having to acknowledge dated infrastructure technology slowing down the introduction of new technological solutions. Current monolithic application supply system is a limiting factor, even a barrier for agility. Large distribution versions of production systems are launched about two times a year, making change implementation inflexible and slow.

The interviewed technical architects believe that realising business requirements should be possible to proceed in an incremental manner. They prefer technical solutions to

be taken to the supply system in smaller increments than currently. In addition, testing required by the supply system should be possible to conduct on these smaller increments. According to interviewed testers, early agile pilot projects showed one big problem: the current development environment does not allow developers to test bigger entities.

However, it is not desired to apply agility, including numerous short iteration cycles, to all aspects. The common nominator for company projects lies in the technical infrastructure, in multi-channel architecture, which is proven, but inflexible. The interviewed infrastructure architect has an opinion that infrastructure is not to be developed using agile methods. The incrementality and iterativity are effective for business requirements, but harmful for interfaces between projects and technical infrastructure. These interfaces should be known and fixed, and remain the same, not only during a project, but also for the entire application life-cycle. On the other hand, there have been needs to update the system product concept and architectural guidelines during project realisation.

According to interviewees, the rapid, annual or bi-annual, change of technologies and application development models causes risks as also the change process may be challenging to manage. Those responsible for business are feared to demand, based on agility, quicker results than is actually possible, or may even ask for solutions that are not immediately possible.

4.1.4 Agility proposals by architects, developers and decision makers

The *chief designer* proposes starting from minimum requirements and functionalities, and taking through smaller increments. Prototypes could be used as the starting point. As projects have been relatively small in the past the chief designer does not see obstacles for agility. Agile working method has the benefit of smart guys being able to use their own know-how and creativity and actively influence project realisation.

An *infrastructure architect* views any changes to business requirements during projects as positive. Developing software architecture ought to be directed to support business agility within the framework defined by the infrastructure.

A *data management architect* sees an opportunity for shorter throughput with the application of agile methods. ASD is interlinked to project realisation, but does not help the preceding phases requiring a lot of time and money. Also the architectural guidelines ought to be made agile. The data management architect understands agility as architects being more involved in project realisation and taking a role in managing changes.

Developers propose starting from removing organisational impediments to ASD and communication and interaction within the development organisation. Prioritising products, projects and requirements at business level ought to be the first step.

According to the *decision makers*, having the business responsibility, the aim is to realise the most profitable solutions and the business group and application development must cooperate efficiently. Managing resources is challenging when there are several simultaneous projects. Projects must be prioritised and be divided into milestones. Project teams are given authority and development responsibility resulting in better commitment.

4.2 New solution

The new solution is synthesised based on the positive qualities of the old practices, analysing the interviewees' recommendations, and the limitations set by the organisation's technical infrastructure. The solution's project management guidelines have their main origins in the Scrum literature because Scrum has demonstrated to be linearly scalable to support multiple project teams and platform companies where architecture is of critical importance [57]. ASD practices such as shared ownership, responsibility, knowledge and skills as well as direct and open communication make many things in development projects more transparent. The goal is to enable realising the most lucrative solutions swiftly and cost efficiently. For achieving the goal, the aim is to increase the agility of the development process and the visibility and transparency of requirements, design constraints and the rationale behind design decisions.

The key aspects the new solution was expected to address:

- prioritising products and applications
- prioritising projects and setting milestones for them
- acknowledging limitations set by technical infrastructure
- tailoring processes project specifically
- prioritising project requirements
- the visibility of the actual achievements of agile project teams
- making decisions timely
- interacting between application development and business
- interacting effectively within application development organisation
- increasing involvement of architects in project realisation
- increasing responsibility for development teams
- rotating staff between release concept planning and development

This paper's focus is on how requirements and architectural issues permeate the concept planning and project realisation phases. A data flow diagram (DFD) is used in Figs. 2 and 3 to illustrate the phases. An activity is represented as

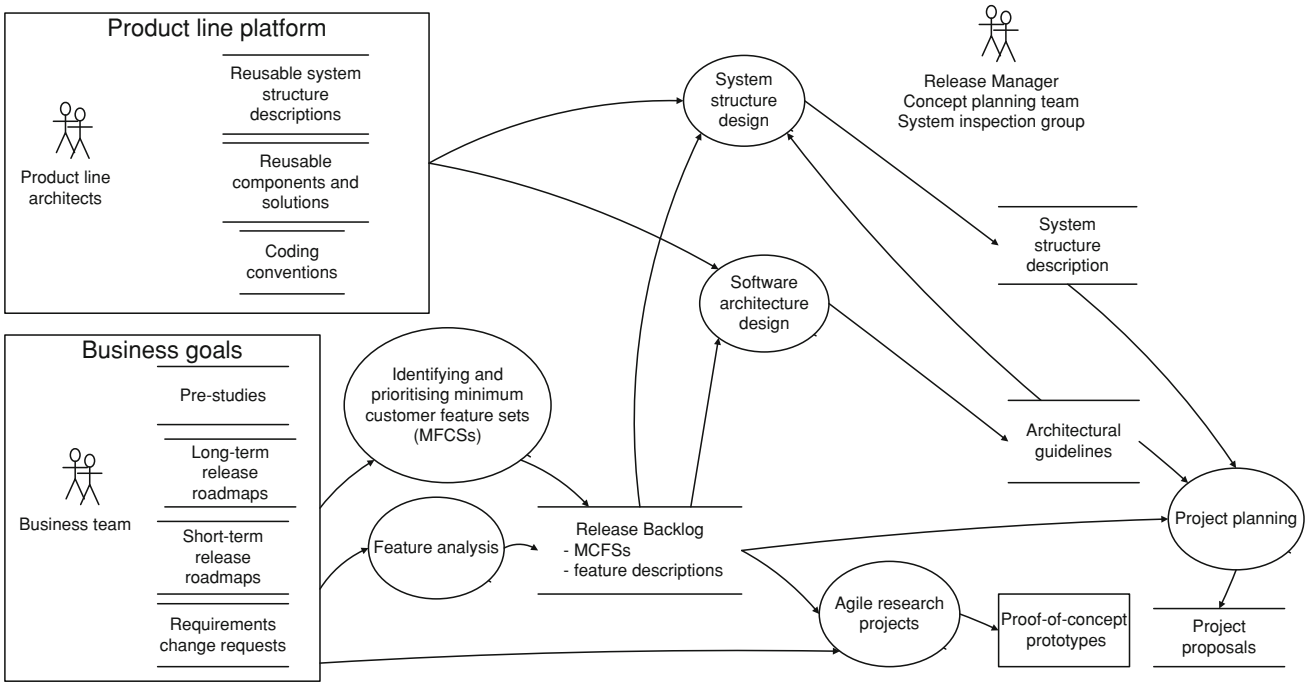

Fig. 2 New release concept planning activities

a circle. A flow is represented by an arrow. Flows are used to describe the movement of information from one part of the system to another part. Stores represent data and information at rest. The notation for a store is two parallel lines. A terminator represented as a rectangle. Terminators are used to describe external entities with which the system communicates.
A special actor symbol is used to describe key roles during a development phase.

4.2.1 Acknowledging business goals in the new solution

The case organisation's business pre-studies should include long-term planning and goal setting in relation to, for example, business, technologies, end-users' needs, products and applications (see the Business goals terminator in Fig. 2). A long-term roadmap should identify and prioritise long-term requirements, select requirements for long-term releases of an application and schedule these releases. Based on the long-term roadmap, the business team will identify and prioritise short-term requirements, select requirements for short-term releases and schedule these releases. The development of a complex product or application may comprise of several successive releases.

4.2.2 Release concept planning in the new solution

Release concept planning should provide background information and rationale for decisions on successful release development projects. A DFD in Fig. 2 illustrates the new release concept planning activities.

4.2.3 Release backlog

Initial customer features for release backlog are to be created based on information in release roadmaps. Customer features could also come from the business team or directly from the release manager. Feature proposals and change requirements may also come from lead designers, project managers and SPL architects. A release manager will create a feature description document for each feature including initial requirements. Concept planning team members describe each feature's implementation, workload estimate and relationships between the feature and architectural parts of the system.

The business team and the release manager will identify, prioritise and select minimum customer feature sets (MCFSs, also known as user stories) for the release and determine MCFS sequencing. A MCFS is the smallest set of functional, non-functional and architectural features that delivers a subset of the requirements returning some value, enabling feedback from the business team. A MCFS is released as one independent entity.

4.2.4 Agile research projects

Separate agile research projects will be carried out in order to explore larger issues for building release backlogs. The

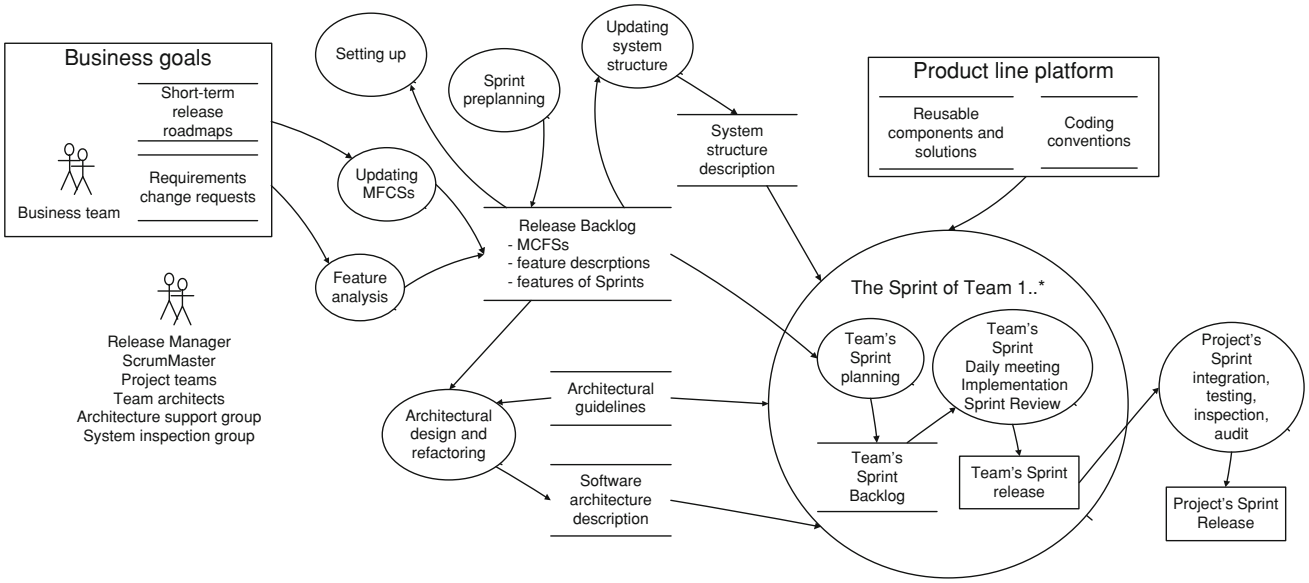

Fig. 3 New release development activities

business team will provide feedback on user interface ideas and core functionality of releases. Research projects will conduct feasibility studies on integrating existing SPL solutions and complex features. Proof-of-concept prototypes, produced by research projects, proactively address changes before they actually occur. Research projects have separate research backlogs. The tasks of research projects may include tackling surprises and, consequently, controlling and directing their progress is difficult. Therefore, research projects sprints are shorter, often two weeks, than the sprints of application projects. The steering groups of research projects should meet after each sprint. Every research project sprint does not necessarily deliver new executable business functionalities.

4.2.5 Architectural guidelines

A concept planning team should include one or more lead designers that create the system structure description document based on information from the release backlog, existing system structure descriptions and person-to-person discussions with numerous architectural experts and other specialists responsible for sub-systems and other projects. The document specifies interfaces between releases and information systems and the technical infrastructure. It also describes alternatives, possibilities, limitations and boundary conditions set by SPL platform for agile development and changes in the release. Important system structure elements are identified as system architecture features for the release backlog. The system structure description document particularly focuses on the overall architecture and key interfaces that are anticipated to be stable during projects.

The concept planning team should include software architects from information architecture and software architecture sub-areas. The architects create architectural guideline document and identify the important parts of the software architecture design as architectural features for the release backlog. The architectural guidelines define and recognise independent modules and architectures for releases, utilising either existing or potentially new solutions. Architectural guideline document includes recommendations for release sprints for forming project teams and for planning content and schedules. To minimise cross-team and cross-project communication problems, the document defines their roles and identifies interfaces for different software components. For example, user interface can often be clearly separated from the rest of the system, and the development of user interfaces requires special skills, tools and SPL artefacts.

4.2.6 Project planning

A large release backlog should be implemented by several projects that produce sub-releases for the entire release. Successive projects are preferred over concurrent projects due to the complexity of dependencies between concurrent projects. One project may produce one or more sub-releases that can be considered project management milestones. A minimum criterion to start agile release development is that the content of the release backlog, architectural descriptions and project plan are ready for the first sprint.

4.2.7 Agile release development in the new solution

Figure 3 illustrates new release development activities. This DFD diagram depicts how new revised business goals are addressed during software production.

Trials with minimal functionality (the Setting up activity in Fig. 3) are needed for setting up technical development environment and for making sure that everything is ready for release implementation. Trials may include minimal functionality with communication paths between the main system architectural components being exercised for finding a skeletal system for building up the system incrementally. The trials do not necessarily address only the highest priority backlog items but rather they may be selected based on the structure of the application and key interfaces between teams, releases, infrastructure, reusable components, external systems and other projects.

MCFSs guide sprint iterations particularly from the business viewpoint. Release manager / business team may reconsider the release backlog before each sprint (see the Updating MFCSs and Feature analysis activities in Fig. 3). Changing release backlog items can cause compulsory modification needs in the system structure description document (the Updating system structure activity in Fig. 3). These modifications could be carried out by team architects and the architecture support group under the control of the system inspection group. Sprint schedules should be kept fixed whenever possible.

Internal structure of a release and relevant documentation in the software architecture description document can evolve iteratively, sprint by sprint, along with new high priority items in the release backlog (the Architectural design and refactoring activity in Fig. 3). In addition, architectural solutions of previous sprints may require refactoring when new and large features are included. Also, refactoring often has side-effects outside an individual team's focus area. Team architects should be responsible for architectural modifying and designing more detailed architectural solutions for sprints. The architecture support group should support architectural design and refactoring activities.

Sprint pre-planning sessions (the Sprint preplanning activity in Fig. 3) are needed to evaluate whether high priority release backlog items include enough information for moving them from the preliminary state to a ready state. Features in the ready state can be inserted into following sprint backlogs. The release manager, ScrumMaster, team architects and other team representatives usually participate in sprint preplanning sessions. There can also be other experts who have experience with high priority items in release backlogs. The release manager should have the responsibility on refining the information. Should the development involve more than one team, the pre-planned release backlog items are to be divided among the teams. This may require further refining the backlog items and synchronising the work between teams. The identified preceding dependencies between MCFSs and architectural features help dividing the development work among multiple teams.

Team's sprint planning (the Team's Sprint planning activity in Fig. 3) can focus on selecting ready state features to be implemented during the next sprint. The exact scope of each sprint will be fixed in the beginning of the sprint. Project teams continuously re-factor code without planned architectural tasks (the Team's Sprint implementation activity in Fig. 3). They are obligated to consult the architectural design and refactoring activity when encountering problems while adopting system structure descriptions, software architecture descriptions and architectural guidelines. Individual teams are not at liberty to modify architecture design in the system structure description document and clearly identified architectures and interfaces in the software architecture description document, because those modifications tend to have side-effects on the whole system, the work of other teams or other projects.

According to the continuous integration practice, project teams frequently integrate their work to identify conflicts between developers quickly. Each integration is verified by an automated build and tests to detect integration errors as quickly as possible. Integration testing validates software architecture. In the case of several project teams and large systems, continuous integration requires multiple integration environments and multiple builds done in a sequence.

During the inspection activities at the end of each sprint, project teams prepare for the sprint audit (the Sprint integration, testing, inspection, audit activity in Fig. 3). They check the status of their own implementation. They carry out pre-tests to ensure the presentation of their results to the release manager. The sprint audit activity validates the results of the finished sprint assuring that the requirements and needs of the business team are met. During the sprint audit, project teams present their results to the release manager, or if releasing MCFSs, to other business team members

4.2.8 Inspection, verification, validation and acceptance in the new solution

Release managers should accept feature descriptions before taking them into the release backlog. The business team accepts or rejects the release backlog and the project plan and decides whether to start the agile implementation of the release. During implementation, the business team accepts changes in release-level strategies and priorities. The release manager accepts changes in individual features in the release backlog.

The system inspection group should inspect and verify that the structure of the developed software conforms to the infrastructure of the SPL platform. The group also accepts the system structure description document at the end of the concept planning phase. The group evaluates, verifies and accepts all changes in the system structure description document during the release development phase. Common

interfaces between project teams are reviewed in joint meetings by all the involved teams. Team architects inspects changes in architectural guidelines and software architecture description document by the end of each sprint. Project teams inspect their own work during sprint inspection activities. The release manager accepts the results of the sprints.

During the delivery phase, the system inspection group accepts the system structure description document. The architecture support group reviews and accepts the software architecture description document. The business team accepts the delivered release as a whole.

4.3 Old solutions and potential new solutions provided by agility

Tables 2, 3, and 4 summarise previous chapters and compares the old and proposed new solutions from three different perspectives of business, general, and software. Table 2 presents old and new solutions at business level.

Table 3 presents old and new solutions at general level.

Table 4 presents old and new solutions at software production level.

4.4 A summary of Scrum extensions

Scrum is an iterative and incremental framework to create or evolve long-lived applications or products either for the market or for internal use within an organisation [58]. Scrum is a move from a project-centric development model towards a continuous development model. There is no traditional project with the beginning and end dates or no traditional project manager. Scrum offers guidelines for agile project management but it does not explicitly address engineering practices.

Scrum can also be used for one-time initiatives or true projects [58]. Most of the case department's software development initiatives are relatively small, true projects. The projects' business functionality typically penetrates across multiple systems in the case company's system portfolio including banking, investment, loan and insurance services. Project team members are experts from all needed service sectors. This results in a high turnover of project team members. There is no sufficient work for a dedicated long-lived team after a completed application due to the project diversity. The common nominator for development projects lies in the SPL platform's technical system infrastructure, multichannel architecture and common reusable assets. The platform is stable from the viewpoint of application development. The result of each project is a part of the large distribution versions of production systems that are launched about two times a year. Applications are developed for the company's own use.

Release planning is described in several Scrum descriptions, although Scrum does not require release planning or long-term product strategies [58,59]. Our solution briefly describes how the business team's long-term and short term planning should support ASD (see Sect. 4.2.1 Acknowledging business goals in the new solution).

Scrum includes a preparation phase [57] or Pregame [60] for developing a plan and product backlog for a product or a project. The preparation phase corresponds to the release concept planning phase in the new solution of this paper (see Fig. 2). The activities of Pregame and this study's release concept planning include some iterations and intermediate checkpoints. Pregame's planning step corresponds to the release backlog and project planning activities in this study's solution. Many items of Pregame's architecture step correspond to the architectural guidelines activity in this study's solution (see Sect. 4.2.5 Architectural guidelines).

This study's release backlog corresponds to Scrum's release backlog [57]. Whereas initial customer features for this study's release backlog are created based on information in release roadmaps, Scum's release backlog items are selected from the product backlog for the current release. Schwaber [61] identifies the staging process for defining non-functional requirements for scaling a project. For scaling a project to use multiple teams, Schwaber and Beedle [40,61] recommend to add into the product backlog non-functional requirements for creating business architecture, system architecture, a more detailed product and technical architecture, infrastructures and development environments to support multi-team development. In our model, the non-functional features in the release backlog are needed not only for scaling the project to support multiple teams but also for specifying the boundaries and interfaces between the results of the project and the whole system and other projects. The concept planning team describes each feature's implementation and relationships between the feature and architectural guidelines. Requirements should be grouped from the business perspective (see Sect. 4.2.3 Release backlog).

According to the Scrum primer [58], the product backlog may include also research work. In our model, research work will be done in separate research projects (see Sect. 4.2.4 Agile research projects). The main reason for this decision is that the tasks of research projects are expected to often bring surprises. Shorter sprints are needed for controlling and directing the progress of the tasks in these projects. Scrum has been used to build Internet applications and financial applications on handheld and web devices [57]. However, an empirical study [62] in 10 US companies indicates that a hybrid mix of agile (such as XP and Scrum) and plan-driven methods these might be most appropriate for Internet software. The critical requirements of the case company's applications such as security, scalability, and robustness are challenging issues in the high-speed development of Internet software

Table 2 Business level issues

Old solution	New solution
The business group describes short-term business goals and defines fixed business goals for separate development projects. The project business goals are not changed during development	Based on prioritised long-term roadmaps, the business team identifies and prioritises short-term requirements, selects the requirements for short-term releases and schedules these releases. Implementation can start once the content of the release backlog is ready for the first sprint
The business group is informed only through project steering group meetings	The release manager represents the business team in face-to-face meetings during release concept planning and development phases
Business value is seen only at the very end of the project, all at once	A large release backlog may be implemented by several projects that produce sub-releases for the entire release. One project may produce one or more sub-releases that can be considered project management milestones
The business group aims to complete and freeze specifications before go/no-go decisions on proceeding to software code design and testing	Release manager/ business team may reconsider the release backlog before each sprint
Business goals in documentations are unclear, lacking information and without adequate requirements prioritisation	Initial customer features for release backlog are created based on information in release roadmaps. Release manager creates a feature description document for each feature including initial requirements. Concept planning team members describe requirements implementation and relationships between features and the system. Workload is estimated
After accepting requirements prior to software design and testing, business group provides feedback only after project realisation	Minimum customer feature set (MCFS) to guide sprint iterations from business viewpoint. After each sprint, project teams present their results to release manager and business team

[62]. In our solution, separate agile research projects will be carried out in order to explore Internet user interface ideas for building release backlogs and for getting early feedback from the business team.

According to Schwaber [60], Pregame's planning step includes verification of management approval and funding. In our model, the business team makes go/no-go investment decisions on proceeding to software code design and testing based on the project proposals produced by the project planning activity in the end of the release concept planning phase (Fig. 2).

According to Schwaber and Beedle [40,61], one team works via as many sprints as required for scaling the project's infrastructure and architecture. Non-functional requirements should be mixed with business functionality, because business functionality must be demonstrated at the end of every sprint. The Setting up activity in Fig. 3 addresses the implementation of the non-functional features in the release backlog that are needed to scale the project to support multiple teams and to find a skeletal system for building up the system incrementally.

According to Scrum [57,61], means and mechanisms are needed to coordinate and synchronise multiple Scrum teams and their efforts. Most reported coordination examples are from the cases where teams are isolated or distributed across geographies [57]. In our case, project teams work in the same city block. In this study, the architectural design and refactoring activity is used to coordinate and control the direction of the project architecture (see Fig. 3). The project sprint integration, testing, inspection and audit practices are used to create the release of the project's sprint from the sprint releases of the project teams. In addition, the system inspection and architecture support groups coordinate the project's compatibility with and effects on the SPL platform's technical system infrastructure, on common reusable assets and information architecture and on other related projects. The business team coordinates and synchronises multiple projects and their releases.

One lesser known Scrum's guideline is that five to ten per cent of each sprint should be devoted to refining the product backlog [58]. Scrum does not tell how to do this but focused workshops are often used for this work [58]. In our solution,

Table 3 General issues

Old solution	New solution
Communication and interaction are strongly documentation based also within the development organisation	Rotating staff between SPL platform and release concept planning and between release concept planning and development phases increases willingness to share knowledge and reduces conflicts
The old ways of working do not support co-learning and distributing best practices	The communicative roles of release manager, ScrumMaster and team architects. The system inspection group and the architecture support group
Projects do not have chief designer level expertise	The role of team architect
Project management is based on the conventional specification-planning-execution-control cycle	Project management emphasises adaptation and value generation
Description of a system product includes all the relevant interfaces and relevant sub-systems	System structure document describes also alternatives, possibilities, limitations and boundary conditions set by SPL platform for ASD
	Important system structure elements are identified as system architecture features for the release backlog
Architecture guidelines aim to ensure that projects develop generic solutions, not project-specific or overlapping solutions	Architectural guidelines document includes recommendations for release sprints for forming project teams and for planning content and schedules
	To minimise cross-team and cross-project communication problems, the document defines their roles and identifies interfaces for different software components
	The guidelines define and recognise independent modules and architectures for releases, utilising either existing or potentially new solutions
Documentation is expected to be perfect and changing it is considered as a matter of authority	ASD embraces changing requirements
Those preparing architectural guidelines do not enforce them. Realisation may differ from guidelines	Project teams consult all team architects when encountering problems while adopting system structure, software architecture descriptions and architectural guidelines

release backlog items are refined in sprint preplanning sessions (the Sprint preplanning activity in Fig. 3).

According to Schwaber and Beedle [40], Scrum's team is constrained only by the issues that have been set prior to the start of a sprint, e.g., the product backlog items the team has selected and organisations standards and conventions. The team is also responsible for using and conforming to any existing architectures and other issues that have been set prior to the start of a sprint. In our model, individual project teams are constrained by selected release backlog items, the system structure and software architecture descriptions, coding conventions and architectural guidelines with reuse recommendations (Fig. 3).

There are three primary roles in Scrum: Product Owner, ScrumMaster and Team [57]. This study's release manager corresponds to Scrum's Product Owner. The release manager servers as a proxy for the business team of the case company. The release manager accepts changes in individual features in the release backlog. This study's ScrumMaster (a certified ScrumMaster) correspond s to Scum's ScrumMaster. This study's project team differs from Scrum's team in that individual teams are required to consult all team architects when encountering problems while adopting system structure, software architecture descriptions and architectural guidelines. Those problems tend to have side-effects on the whole system, the work of other teams or other projects. Otherwise the sprints of each project team follow the Scrum framework's [57] roles, ceremonies and artefacts (The Sprint of Team 1..* activity in Fig. 3).

In addition to the three roles of Scrum, Scrum and CMMI (capability maturity model integration) Level 5 recommends that the team is likely to include the roles and expertise such as domain experts, system engineers, software engineers, architects, programmers, analysts, QA experts, testers and UI designers [57]. It has been reported that, in the context of large products, architects in production Scrum teams have weekly assembled in an architecture group for controlling the direction of the project architecture [57]. After scaling

Table 4 Software production issues

Old solution	New solution
Sequential waterfall style development process in software code design and testing phases	Incremental and iterative development process in software code design and testing phases
In order to avoid surprises, proof of concept prototypes are occasionally constructed and performance tests are carried out to confirm architecture functionality already during concept development	Proof of concept prototypes are developed in separate agile research projects Feasibility studies are conducted on integrating existing SPL solutions and complex features in the release backlog
Technical infrastructure is common nominator for projects	Architecture support group and system inspection group are formed to carry out compulsory modification needs
Separate architecture support groups can be established to support project realisation, should significant issues require	Project team architects are responsible for architectural modifying and designing more detailed architectural solutions for sprints Individual project teams are not allowed to modify architecture design
Designers find it difficult to understand requirements documents as they have been prepared from business perspective	Trials with minimal functionality Feature analysis during release development Sprint preplanning sessions preparing workload estimates and pre-planned and clear features for implementation
Designers concentrate on their tasks without a view to surroundings	Architectural communication between project teams, team architects, architecture support group and system inspection group
System product concept and architectural guidelines are frozen for designers	
Designers are expected to maintain an application log, its quality is, however, not controlled	Architecture support group reviews and accepts the software architecture description document
Development projects encounter challenges relating to costs, project division, communication, and interaction	To avoid communication problems, split development is favoured wherever possible using prioritised roadmaps. A large release backlog may be split into several projects. Each project can also be split into several sub-projects

a project to use multiple teams, each new team includes a member of the original team, who serves as an expert on the project's infrastructure and architecture [61]. In our model, one or more concept planning team members become project team architects and serve as experts on the project's system and software architecture. The project team architects in the architectural design and refactoring activity are responsible for architectural modifying and designing more detailed architectural solutions for sprints.

Scrum includes continual inspection in Scrum sprints [57]. The product owner accepts the results of the sprints [57]. In addition to these, our model includes additional roles and responsibilities in the inspection, verification, validation and acceptance of the results of the sprints and related documentation (see Sect. 4.2.8 Inspection, verification, validation and acceptance in the new solution).

5 Research contributions and managerial implications

Managers who wish to start utilising ASD, but who have realised the direct use of these methods as unfeasible, may benefit of the results of this study. Especially companies developing solutions to be integrated into complex IT products can utilise the results. The transition towards agility is not a single step, but a series of smaller ones. This study provides an example on potential actions required to improve the match between business goals and the results of development projects. In addition, the cost efficiency of product development can be improved through increased flexibility during the development.

The results of this study have implications on business level responsibilities. Business teams are responsible for the visibility and transparency of requirements. They are responsible for creating prioritised long-term roadmaps, identifying and prioritising short-term requirements, selecting requirements for short-term releases and scheduling releases. Prioritised initial customer features for release backlog are created based on information in release roadmaps. The business team is responsible for subdividing the implementation of a large release backlog into several projects that produce sub-releases for the entire release. The business team is responsible for appointing a business-oriented release manager for each project. The business team is responsible for

specifying minimum customer feature sets to guide the milestones and sprint iterations from business viewpoint. The release manager and the business team have better than one iteration's accuracy of information about the progress and status of projects. Therefore, the business team is able and responsible for redirecting projects after each iteration cycle, if deemed necessary.

The results of this study have implications on architecture level management. Architects are responsible for the visibility and transparency of design constraints and the rationale behind design decisions. System level design decisions are architecturally very significant and should be explicitly documented during all project phases. Many reuse decisions are architecturally significant and are documented and communicated via architectural guidelines. These architecturally significant decisions should be made early and mainly before implementation. However, the results of this study indicate that these definitions should not be frozen, but changed during development according to gained understanding. The system inspection group owns system level architectural issues. Software architects from information architecture own reuse issues. Team architects are responsible for the internal structure of releases and relevant architectural documentation. This study highlights the role of staff rotation as when specifications are neither stable nor perfect, the tacit knowledge and understanding of personnel becomes the key.

This study points out how ScrumMaster and team architects should have a more communicative role to ensure adequate links between project team, business team, architecture support group, system inspection group, designers, and such to assure that work is conducted efficiently and any problems are addressed openly and swiftly.

6 Limitations

The limitations of this study include, the study concentrating on a single case and including no more than 33 interviewees and three workshops, somewhat weakening the generaliseability of the results. Participants especially decision-makers and interviewed software architecture experts were experienced and had worked for a long time (6–20 years) with the plan-driven and platform-oriented product-line architecture approach of the case company. On the other hand, their experience of agile approaches was limited to some agile pilot projects. All interviewees were aware of the gained experience of the pilots. The "big picture" presentation manner of the findings is expected to promote the transferability of the results. Further research is needed on the experience of the use of the developed process framework.

No significant discrepancy was found in the opinions of the participants about challenges and solutions in combining the SPL and ASD cultures in their projects. Participants were purposely selected to maximize diversity of stakeholder concerns and, partially therefore, many of the opinions were complementary to each other. The interview instrument's information management and communication model of the interviewees was found a useful countermeasure to reduce architecture-related misconception and misunderstanding threats to validity of the study.

Tian and Cooper [19] argue that an established agile method is not a suitable choice in a financial product line application due to the security critical nature of this application. ASD methods do not promote formal documentation that may be required within the financial services community [37,38]. These reflect the results of this study that the direct use of the basic and well-known practices of agile methods is unfeasible in the context of the case organisation. Cao and Ramesh [36] argue that formal requirement specifications can benefit from frequent reprioritisation of requirements. This is in line with the results of this study.

The results of this study also show that the transition towards agility in the case organisation is not a single step, but a series of smaller ones. This reflects the findings of Lycett et al. [33] that management of large and mature organisations sees big risks in the big-bang adoption of agility. Several industrial experiences (e.g., [2,24,63,64]) indicate that a small group of experts in a research project or an exploration phase before agile production is often needed to prepare key requirement and architecture issues to ensure that a project is in a situation where agile methods are feasible. This is in line with the results from early agile pilot projects in the case organisation. Therefore, the results of this study emphasise release concept planning before agile production. Tyree and Akerman [34] believe that architecture decisions are the missing link between traditional and agile architecture documentation in the context of large financial IT systems. This is in line with the results of this study that emphasise rationale for decisions on successful release development projects.

The proposed first steps of this study for moving towards ASD include no concrete proposals for more agile system testing, validation and releasing practices. This is in line with several industrial companies that, after more than 5-year experience in tailoring agile methods, still use and do not see big problems in using company-specific less agile system testing, validation and releasing practices (e.g., [24,50]).

Brown's case study focused on a broad view of agility at a large bank [10]. The focus of this research was on small in-house software development projects (few agile teams with fewer than ten members) for the company's own use thus excluding maintenance projects and the use of sub-contractors. This study focused on improving the product development activity [15] of the case organisation's SPL. From these viewpoints, this study is similar to the case of Ali Babar et al. [24]. The key findings of the two studies have

many similarities such as the exploration phase before agile production although their case organisations and domains were very different.

Augustine et al. [36] have developed practices for managing large and mission-critical agile (XP practices) projects. Their financial service case project included more than 120 members. The article includes only one remark about architecture: "developers struggled with simple design in light of the large legacy code base." In contrast to the article, the results of this study emphasise acknowledging limitations set by the case organisation's complex dated technical infrastructure and multi-channel architecture. The case organisation's IT system has similar shortcomings and agility challenges as many other large financial IT systems: business functionality across multiple sub-systems and tightly coupled parts of the infrastructure [34].

Despite the above and potentially other limitations, the findings from this study are expected to increase the understanding of combining complex product line architectures and ASD within the context of large financial IT systems.

7 Conclusions

ASD is an important development method for improving software development. However, experiences from ASD especially in large organisations highlight the need of extending and tailoring ASD to organisational requirements. This study concentrated on studying the adoption and architectural extensions of the Scum method in developing new service applications of large financial IT systems. The study was conducted by interviewing a wide representation from the personnel and stakeholders of software projects and analysing the potential of using agile methods for improving software development.

The interviewees highlighted the old serial development model, without iterative feedback, allowing results of development projects to become obsolete and products failing to meet expectations. Inefficiency caused by planning overly secure schedules is seen to increase expenses. In addition, the old ways of working are seen not to support co-learning and distributing best practices. Communication is strongly documentation based, with weak interaction, making it very difficult to confirm correct realisation, should the written requirements be ambiguous.

Scrum does not require release planning. In the solution of this study, the business team should coordinate and synchronise multiple projects and their releases via a prioritised long-term roadmap and short-term release plans. In Scrum, the product owner develops a plan and backlog for a project. In the studied case, the result of each project is a part of a large distribution release. Each project should acknowledge the constraints set by other projects and the complex infrastructure, multi-channel architecture, data security issues and system portfolio in the organisation's SPL platform. In the case of this very complex system portfolio, it is virtually impossible for an individual to perceive large enough entities. Therefore, lead designers are needed to collect the constraints from the release backlog, existing system structure descriptions and person-to-person discussions with numerous architectural experts and other specialists responsible for sub-systems and other projects. In order to minimise risks and promote reuse, system, software architecture and reuse guidelines documents are still seen of vital importance for specifying the constraints. In addition, research work will be done in separate agile research projects.

Scrum offers guidelines for coordinating and synchronising multiple and distributes teams. Project teams in the case company work in the same city block. In the solution of this study, team architects collaborate in coordinating and controlling the direction of the project architecture. The system inspection and architecture support groups coordinate the project's compatibility with and effects on the SPL platform's technical system infrastructure, common reusable assets, information architecture and other related projects.

Projects' business functionality typically penetrates across multiple systems in the case company's system portfolio including banking, investment, loan and insurance services. Project team members are experts from all needed service sectors. This often results in a high turnover of project team members. There is no sufficient work for a dedicated long-lived team after a completed application due to the project diversity. Therefore, it seems to be difficult to achieve all Scrum's aims for a continuous development model and dedicated long-lived teams in the context of large financial IT systems.

Using agile methods in the case company would enable better match between developed products and business requirements. The results of this study propose steps for a large IT systems company for moving towards ASD starting by developing new service applications using Scrum. The results of this study have implications on business, architecture and project team level management. Many implications relate to the visibility and transparency of requirements, design constraints and the rationale behind design decisions.

In addition to the addressed limitations of this study, recommended future research could include more testing the proposed actions towards agility. Also, a follow-up study after an adequate period, would be interesting to check how the transition towards agility has progressed.

Acknowledgments This research has been carried out within the EVOLVE ITEA2 and Varies Artemis projects funded by the National Technology Agency of Finland (Tekes) and VTT. Many sincere thanks

are due to all the fellow researchers in the case project. The author would also like to thank Dr. Pekka Belt, Dr. Matti Mottonen and Dr. Janne Harkonen for their support in writing this article.

References

1. Bowers J, May J, Melander E, Baarman M, Ayoob A (2002) Tailoring XP for large system mission critical software development. In: Wells D, Williams L (eds) Extreme programming and agile methods—XP/Agile Universe 2002. Springer, Berlin, pp 269–301.
2. Kähkönen T (2005) Life cycle model for a software process improvement project deploying an agile method. In: Andersin H, Niemi E, Hirvonen V (eds) The proceedings of the international conference on agility, ICAM 2005, 27 June 2005. Helsinki University of Technology, Otaniemi, Finland, pp 225–232
3. Abrahamsson P, Babar MA, Kruchten P (2010) Agility and architecture: can they coexist? IEEE Softw 27(2):16–22. doi:10.1109/MS.2010.36
4. Kruchten P (2007) Voyage in the agile memeplex: agility, agilese, agilitis, agilology. ACM Queue 5(5):38–44
5. Vanhanen J, Jartti J, Kähkönen T (2003) Practical experiences of agility in the Telecom industry. In: Mearchesi M, Succi G (eds) The proceedings of extreme programming and agile processes in software engineering, XP 2003, 25 May 2003, Genova, Italy. Springer, Berlin, pp 279–287
6. Evans I (2006) Agile delivery at British Telecom. Methods Tools 14(2):19–26
7. Karlsson E-A, Andersson L-G (2000) XP and large distributed software projects. In: Succi G, Marchesi M (eds) Extreme programming examined: selected papers from the XP 2000 Conference. Addison-Wesley, New York, pp 119–134
8. Lindvall M, Muthig D, Dagnino A, Wallin C, Stupperich M, Kiefer D, May J, Kähkönen T (2004) Agile software development in large organizations. IEEE Comput 37(12):26–34
9. McMahon P (2006) Lessons learned using agile methods on large defense contracts. CrossTalk 19(5):25–30
10. Brown A (2011) A case study in agile-at-scale delivery. In: Sillitti A, Hazzan O, Bache E, Albaladejo X (eds) The proceedings of the12th international conference on agile software development, XP 2011, 10 May 2011, Spain. Springer, Berlin, pp 266–281
11. Dyba T, Dingsoyr T (2008) Empirical studies of agile software development: a systematic review. Inf Softw Technol 50(9–10):833–859
12. Freudenberg S, Sharp H (2010) The top 10 burning research questions from practitioners. IEEE Softw 27(5):8–9
13. Leffingwell D (2011) Agile software requirements: lean requirements practices for teams, programs, and the enterprise. Addison-Wesley, Upper Saddle River
14. Kettunen P, Laanti M (2008) Combining agile software projects and large-scale organizational agility. Softw Process Improv Pract 13(2):183–193
15. Northrop L (2002) SEI's software product line tenets. IEEE Softw 19(4):32–40
16. van der Linden FJ, Schmid K, Rommes E (2007) Software product lines in action. Springer, Berlin
17. Clements P, Jones L, McGregor J, Northrop L (2006) Getting there from here: a roadmap for software product line adoption. Commun ACM 49(12):33–36
18. Hanssen G, Fægri T (2008) Process fusion: an industrial case study on agile software product line engineering. J Syst Softw 81(6):843–854
19. Carbon R, Lindvall M, Muthig D, Costa P (2006) Integrating product line engineering and agile methods: flexible design up-front vs. incremental design. In: The proceedings of the 1st international workshop on agile product line engineering, 21 August 2006, Baltimore, MD USA, pp 1–8
20. Tian K, Cooper K (2006) Agile and software product line methods: are they so different. In: The proceedings of the 1st international workshop on agile product line engineering, 21 August 2006, Baltimore. Maryland, USA, pp 1–8
21. McGregor J (2008) Mix and match. J Object Technol 7(6):7–13
22. Beck K, Andres C (2004) Extreme programming explained: embrace change, 2nd edn. Addison Wesley Longman, Inc., Reading
23. Turk D, France R, Rumpe B (2005) Assumptions underlying agile software-development processes. J Database Manage 16(4):62–87
24. Ali Babar M, Ihme T, Pikkarainen M (2009) An industrial case of exploiting product line architectures in agile software development. In: The proceedings of the 13th international software product line conference, 24 August 2009, San Francisco. CA, USA. ACM, New York, pp 171–179
25. McGregor J (2008) Agile software product lines, deconstructed. J Object Technol 7(8):7–19
26. Noor M, Rabiser R, Grünbacher P (2008) Agile product line planning: a collaborative approach and a case study. J Syst Softw 81(6):868–882
27. Petersen K, Wohlin C (2010) The effect of moving from a plan-driven to an incremental software development approach with agile practices: an industrial case study. Empirical Softw Eng 15(3):654–693. doi:10.1007/s10664-010-9136-6
28. Falessi D, Cantone G, Sarcia' SA, Calavaro G, Subiaco P, D'Amore C (2010) Peaceful coexistence: agile developer perspectives on software architecture. IEEE Softw 27(2):23–25
29. Booch G (2010) An architectural oxymoron. IEEE Softw 27(5):96
30. Hansson C, Dittrich Y, Gustafsson B, Zarnak S (2006) How agile are industrial software development practices? J Syst Softw 79(9):1295–1311
31. Hofmeister C, Kruchten P, Nord R, Obbink H, Ran A, America P (2007) A general model of software architecture design derived from five industrial approaches. J Syst Softw 80(1):106–126
32. Christou I, Ponis S, Palaiologou E (2010) Using the agile unified process in banking. IEEE Softw 27(3):72–79
33. Lycett M, Macredie RD, Patel C, Paul RJ (2003) Migrating agile methods to standardized development practice. Computer 46(6):79–85
34. Tyree J, Akerman A (2005) Architecture decisions: demystifying architecture. IEEE Softw 22(2):19–27
35. Merton R, Fiske M, Kendall P (1990) The focused interview: a manual of problems and procedures, 2nd edn. The Free Press, New York
36. Augustine S, Payne B, Sencindiver F, Woodcock S (2005) Agile project management: steering from the edges. Commun ACM 48(12):85–89
37. Cao L, Ramesh B (2008) Agile requirements engineering practices: an empirical study. IEEE Softw 25(1):60–67
38. Coram M, Bohner S (2005) The impact of agile methods on software project management. In: Proceedings of the 12th IEEE international conference and workshops on the engineering of computer-based systems (ECBS'05). IEEE, pp 393–370
39. Poort E, Pramono A, Perdeck M, Clerc V, van Vliet H (2009) Successful architectural knowledge sharing: beware of emotions. In: Mirandola R, Gorton I, Hofmeister C (eds) The proceedings of the 5th international conference on the quality of software architectures, QoSA 2009, 24 June 2009. East Stroudsburg, PA, USA, Springer, Berlin, pp 130–145
40. Schwaber K, Beedle M (2002) Agile software development with Scrum. Prentice-Hall, Upper Saddle River
41. Krueger C (2002) Eliminating the adoption barrier. IEEE Softw 19(4):29–31

42. Nerur S, Balijepally V (2007) Theoretical reflections on agile development methodologies. Commun ACM 50(6):79–83
43. Thapparambil P (2005) Agile architecture: pattern or oxymoron? Agile Times 6(1):43–48
44. Mens T, Tourwe T (2004) A survey of software refactoring. IEEE Trans Softw Eng 30(2):126–139
45. McGregor J, Northrop L, Jarrad S, Pohl K (2002) Initiating software product lines. IEEE Softw 19(4):24–27
46. Sharifloo A, Saffarian A, Shams F (2008) Embedding architectural practices into Extreme Programming. In: The Proceedings of 19th Australian conference on software engineering, 25 March 2008, Perth. Western Australia, IEEE, pp 310–319
47. Holcombe M, Thomson C (2008) Seven years of XP–50 customers, 100 projects and 500 programmers–lessons learnt and ideas for improvement. In: Abrahamsson P, Baskerville R, Conboy K, Fitzgerald B, Morgan L, Wang X (eds) The Proceedings of the 9th international conference on agile processes and eXtreme programming in software engineering, 10 (June 2008) Limeric. Ireland. Springer, Berlin, pp 104–113
48. Boehm B (2006) Some future trends and implications for systems and software engineering processes. Syst Eng 9(1):1–19
49. Fraser S, Rising L, Ambler S, Cockburn A, Eckstein J, Hussman D, Miller R, Striebeck M, Thomas D (2006) A fishbowl with piranhas: coalescence, convergence or divergence? In: The proceedings of dynamic languages symposium in companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, 22 (October 2006) Portland, Oregon. USA. ACM, New York, pp 937–939
50. Karlström D, Runeson P (2006) Integrating agile software development into stage-gate managed product development. Empiric Softw Eng 11(2):203–225. doi:10.1007/s10664-006-6402-8
51. Conboy K, Fitzgerald B, Golden W (2005) Agility in information systems development: a three-tiered framework. In: Baskerville R, Mathiassen L, Pries-Heje J, DeGross J (eds) Business agility and information technology diffusion. Springer, New York, pp 35–49
52. Fernandez DJ, Fernandez JD (2008) Agile project management–agilism versus traditional approaches. J Comput Inf Syst 49(2): 10–17
53. Denne M, Cleland-Huang J (2004) Software by numbers: low-risk, high-return development. Prentice Hall, Upper Saddle River
54. Clements P, Northrop L (2002) Salion, Inc.: a software product line case study, CMU/SEI-2002-TR-038. Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
55. Mohan K, Ramesh B, Sugumaran V (2010) Integrating software product line engineering and agile development. IEEE Softw 27(3):48–55
56. IEEE (1998) Standard for information technology–software lifecycle processes, IEEE-Std-12207. IEEE, New York
57. Sutherland J, Schwaber K (2007) The Scrum papers: nuts, bolts, and origins of an agile process. http://www.scrumalliance.org. Accessed 7 September 2012
58. Deemer P, Benefield G, Larman C, Vodde B (2010) The Scrum primer. http://www.scrumalliance.org. Accessed 7 September 2012
59. Paulk M, Davis N, Maccherone L (2011) On empirical research into Scrum. http://www.cs.cmu.edu/~mcp/agile/oersa.pdf. Accessed 14 June 2011
60. Schwaber K (1995) SCRUM development process. In: The Proceedings of tenth annual conference on object-oriented programming systems, languages, and applications (OOPSLA'95), workshop on business object design and implementation, 15 October 1995, Austin. Texas, USA, pp 117–134
61. Schwaber K (2004) Agile project management with Scrum. Microsoft Press, Redmond
62. Baskerville R, Ramesh B, Levine L, Pries-Heje J (2006) High-speed software development practices: what works, what doesn't. IT Professional 8(4):29–36
63. Drobka J, Noftz D, Rekha R (2004) Piloting XP on four mission-critical projects. IEEE Softw 21(6):70–75
64. Kruchten P (2004) Scaling down large projects to meet the agile sweet spot. IBM. http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/aug04/5558.html. Accessed 30 October 2009