Customer-Specific Teams for Agile Software Evolution

Helena Holmström Olsson
Dept. of Computer Science
Malmö University
Malmö, Sweden
helena.holmstrom.olsson@mah.se

Jan Bosch
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
jan.bosch@chalmers.se

Hiva Alahyari
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
hiva.alahyari@chalmers.se

***Abstract*—For more than a decade, agile methods have shown successful for increasing responsiveness to customer needs. As a major characteristic, agile methods advocate close customer collaboration in the early phases of software development. However, research on how to maintain agile ways of working during software evolution is scarce. In this paper, we address the need to establish and maintain agile ways of working during software evolution. We direct our attention to large-scale software development where development companies struggle to meet the needs of a large customer base. The contribution of this paper is two-fold. First, we propose customer-specific teams as a way to reap the benefits of agile methods in the evolution phase of large-scale software development. Second, we confirm the use of these teams as successful for improving customer responsiveness, customer satisfaction and feature quality in the subsequent phases of software evolution.**

Keywords-agile methods, large-scale software development, software evolution, customer-specific teams

I. INTRODUCTION

Market uncertainties, competitive pressures, and the constant need for shortened development cycles call for software development practices that are flexible, responsive and adaptive. During the last decade, a variety of agile methods designed to enhance development teams' ability to respond to change have emerged. In emphasizing the use of iterations and development of small features, agile methods have increased the ability for software development companies to accommodate changing customer requirements [1]. In particular, agile methods have shown their capacity in involving customers during the requirements engineering, design and development. As can be seen in previous research, there are a number of techniques focusing on how to efficiently involve customers in the daily practices of software development [2, 3, 4], and in methods such as Scrum and XP the customer is viewed as an important part of the software development team.

However, while agile development methods, and implementation and adoption of systems developed according to agile practices are common [5], the notion of software evolution, and how to maintain close customer collaboration and responsiveness to customer requests after product deployment, is not extensively studied in the agile community. As noted in our previous research [6], despite successful accounts on how to involve customers in the pre-deployment phases, there is still the need for ways to maintain this collaboration in the post-deployment phase in which software systems develop, grow and change. As part of this process, continuous corrections, adjustments and improvements need to be made to the system in order to address changing customer requirements, to address new customer groups and to cater for new technological innovations [7].

In this paper, we address the need to involve customers and increase responsiveness in the software evolution phase. In particular, we direct our attention to large-scale software development where companies struggle to meet the needs and desires of a large customer base. We see that while close collaboration might be difficult to achieve with all customers, advantages can be achieved by having customer-specific teams, i.e. designated development teams that work exclusively with one selected customer in order to quickly respond to their particular needs and requests. As a result, the company can benefit from the knowledge gained from this collaboration when responding to other customers. Based on interviews with three customer-specific teams, two customer unit representatives and product, program, and integration managers at the market leading company in global telecommunication systems, we show that the use of customer-specific teams is a promising way of working in order to reap the benefits of agile practices in the post-deployment phase of large-scale software development.

The contribution of this paper is two-fold. First, we propose customer-specific teams as a way to involve customers in the evolution phase of large-scale software development. Second, our study confirms the use of these teams as successful for improving (1) customer responsiveness, (2) customer satisfaction and (3) feature quality in the subsequent phases of software evolution.

The paper is organized as follows. In section 2, we introduce agile methods. While outlining the advantages with these methods, we recognize that benefits have not necessarily been reaped in the software evolution phase. We describe the software evolution phase, the importance of customer involvement, and we introduce customer-specific teams. In section 3, we describe our research site and method. Section 4 presents the interview findings, and in section 5 we discuss these findings. Finally, section 6 presents the conclusions.

II. BACKGROUND

A. Agile software development

During the last decade agile methods have dramatically changed the way software development is performed. Unlike traditional development methods characterized by sequential phases and heavy upfront planning, agile methods deal with unpredictability and change by relying on people and close customer collaboration rather than formalized processes [8]. Agile methods are characterized by short development cycles, collaborative decision-making, rapid feedback loops, and continuous integration of code changes into the product baseline [9]. Today, many different agile methods are in use [10]. During the last decade, XP and Scrum have become well established in small-scale, as well as large-scale software development. While XP is basically a collection of well-known software engineering practices taken to the extreme [3], Scrum is a simple, low-overhead process for managing software development [11, 12]. Although agile methods differ in details and techniques, overall agile principles such as 'flexibility', 'working code' and 'customer collaboration' lie at the heart of all of them. In more detail, the agile manifesto (http://agilemanifesto.org) presents twelve principles that characterize agile software development. While these principles were initially developed for smaller software development organizations, evidence indicate that large software-intensive organizations operating in complex global development environments are in the process of deploying agile methods as part of their de-facto approach to software development.

B. Software evolution

Software evolution is concerned with the changes to a software system after its deployment to a customer. Although software engineering practices traditionally focused on the pre-deployment phases in which rigorous requirements engineering activities take place, it has become clear that many relevant activities for a software system take place after commercial deployment of the system. Cook et al [13], describe all industrial software systems as systems that co-evolve with its context over time. As recognized already a decade ago [14], software evolution needs to be quick and responsive in order for companies to stay competitive. For example, large-scale businesses or governmental organizations have to respond fast to changing environmental or competitive conditions such as modifying systems to support new products or services. Likewise, to create systems assembled from different components and allow for a successful co-evolvement of these have become increasingly important in today's software evolution processes. Interestingly, the whole notion of software development is moving away from being regarded as an activity that takes place within the boundaries of an organization [15]. Instead, software development is becoming an activity that includes externally developed components and applications, as well as external stakeholders that increase the value of the core product. From a software evolution perspective this notion calls for mechanisms to include the surrounding environment in the evolution process, and a need to embrace the innovative capacity of a large customer base.

While there are a number of approaches to support evolution of software [16], changing requirements still constitute a challenge. Often, they are seen as problematic and referred to as requirements volatility and requirements creep [17]. Clearly, a paradigm shift is required towards software evolution being the norm, rather than the exception in software engineering [18]. Also, and based on the shortcomings recognized in previous research, we need ways in which to better involve customers in this continuous modification and correction process where the majority of the system's functionality evolve.

C. Customer involvement

Customer involvement has been a prominent research topic in the field of information systems (IS) and Human Computer Interaction (HCI) for a long time, and there are several techniques, e.g. prototyping techniques, focus groups, face-to-face meetings, workshops, customer segmentation and lead-user feedback etc., with which to tap into customer knowledge. Recently, agile methods have complemented already existing techniques with development and management practices that emphasize customer involvement and customer representation during development [11]. Contrary to these fields of research, customer involvement has not been a major research topic in the software engineering (SE) tradition and even though customers are considered critical for the success of a system, methods and techniques for how to efficiently involve them during development have not been the focus of attention [19]. Instead, SE research provides a rich body of knowledge on requirements engineering, and the involvement of customers is recognized as critical mainly in the pre-development phases. As can be seen in Davis [20], one reason for the focus on requirements engineering is the fact that detecting an error during pre-development phases is far less expensive than detecting an error during the design and coding phase of a system. In addition to research on requirements engineering, verification and validation is another characteristic of SE research. Verification and validation is the process of checking that a software system meets the specifications set by the customer, and that it fulfills its intended purpose, i.e. a way to ensure and evaluate the quality of a software system [21]. While the concepts are tightly interrelated, they refer to different things. Boehm [22] succinctly explained the difference between them by saying that in validation you check if you build the 'right product', while in verification you check if you build the 'product right'. Traditionally, these activities were placed at the end of the systems development cycle when major parts of the system were already built. However, and as a result of more complex systems, fast-changing technology and the increasing costs related to discovering errors late in the process, verification and validation activities have become a way to continuously involve customers in the checking of the system that is being built.

As can be seen above, there is a number of ways to involve customers in the pre-development and development phases of software development, and the agile methods have

contributed to closer collaboration with customers [4, 11]. However, and as recognized both in SE research as well as in other domains, the subsequent phase of software evolution in which continuous modification and correction of the software is made is not fully explored in terms of customer involvement. In particular, knowledge on how to maintain agile ways of working closely with customers also after commercial deployment is lacking. Often, customer feedback tend to concern the current system and its weaknesses and hence, works as good input for correcting errors and maintain functionality, but not necessarily as rewarding input for extension, expansion and innovation of new functionality. To achieve this, more direct customer collaboration is needed to allow for development teams to continuously learn about changing customer behavior and expectations.

D. Customer vs. product needs

Successful software systems have a significant number of customers and these customers tend to have many ideas on how the product can serve their needs. As can be seen in recent research, [23], management of ideas, and especially management of innovation, is critical for the success of software evolution. In this process, strategic customers are looking to enforce the inclusion of requirements that are important to them in the roadmap for the next release of the system. Typically, this leads to a tension between two conflicting interests. On the one hand, the development organization seeks to achieve scale in terms of implementing as many new features to as many customers as possible. On the other hand, the development organization needs to show responsiveness to its customers, especially the most strategic ones, and seeks to minimize the delay between a customer request and the deployment of a solution meeting that request. In most organizations, the focus is on achieving scale. Therefore, customers that ask for unique solutions are viewed as problematic since it usually causes disadvantages in terms of added complexity in product version control [24]. For the subsequent discussion in this paper, we introduce two concepts, i.e. the notion of 'customer-unique features' and 'customer-first features'. A customer-unique feature is a feature that will only have relevance to one specific customer, and the likelihood of any other customer requesting the same feature is low. Most often, customer-unique features are considered problematic and preferably kept outside the main product baseline. A customer-first feature, on the other hand, is a feature that is requested by one customer, but that most likely will be requested by other customers in the future. A customer-first feature presents an excellent opportunity to develop a feature in close collaboration with one customer and, through knowledge gained in this collaboration, reduce the amount of re-work needed when developing this feature to other customers.

E. Customer-specific teams

Customer-specific teams are designated teams that work exclusively with one selected, and highly prioritized customer, in order to quickly respond to their particular needs and requests after product deployment. Customer-specific teams have proven useful for improving long-term relationships and for adding value to customers. As one successful account, LaNasa reports on teams working closely with customers to enhance customer value and strengthen customer relationships [25]. These teams are organized as cross-functional teams with competencies spanning organizational boundaries. In particular, customer-specific teams engage with customers in order to improve, modify and enhance functionality in the post-deployment phase of software development and hence, allow for a development organization that can quickly respond to specific customer needs during the evolution phase of the system. The knowledge gained in working closely with one selected customer can then be transformed into generic value for a large customer base. Similar to agile teams [2] and to lean software development practices [26], an important feature of customer-specific teams is the close access to the customer and the short feedback loop between the team and the customer. With the opportunity for direct customer contact on a daily basis, the use of customer-specific teams can further reduce cycle time with the intention of having significantly faster deliveries of improvements and new functionality.

III. RESEARCH SITE AND METHOD

A. Research site

This study was conducted in close collaboration with the world-leading provider of telecommunication systems. The company offers end-to end solutions for mobile communication and they develop telecommunication infrastructure components. The company is transforming their development organization into development units including a number of agile, cross-functional teams involving software architects, designers, developers, testers etc. The intention is that the cross-functional teams are accountable for developing a complete feature from the formulation of requirements until release to the product baseline and deployment at customer site. As a result of this autonomous team structure, a team can be allocated to a specific customer if needed, i.e. work as a customer-specific team, for a period of time. From being more of an experiment a few years ago, customer-specific teams have become a natural part of the development organization and a way to maintain close customer collaboration and responsiveness to customer requests also in the evolution phase of large-scale software development.

The development organization involved in this study has about 30 cross-functional teams that are located at two different development sites in different time zones. Each team consists of 7-8 members. To compile and manage releases, product and program management, feature integration management and release verification support the cross-functional teams. The teams are involved in development of one of the nodes in the 3G networks that includes a wide range of customer requested features as well as advanced support for mobility management. Today, the product contains about 5 million lines of code and its customers are mobile operators all over the world.

B. Research method

Our paper reports on a six months (January – June 2012) case study [27]. During this time, we studied three customer-specific teams serving different customers, two customer units with which two of the teams interact, and we involved program, product, and integration management at the main development site. The main data collection method used was semi-structured interviews with open-ended questions [28]. In total, 17 interviews were conducted. In the three customer-specific teams we interviewed the team leader, the system manager, the system designer and the function tester. At each customer unit we interviewed the person with which the customer-specific teams interact, and who has direct contact with the customer. Finally, we conducted interviews with a program manager, a product manager and an integration leader at the main development site in order to capture the context in which the customer-specific teams operate. In our research, we explore the following questions:

- *How can benefits of agile practices, i.e. close customer collaboration and responsiveness to customer requests, be established and maintained in the evolution phase of large-scale software development?*
- *In what way do customer-specific teams (CST) contribute to the evolution phase of large-scale software development?*

The research questions were formulated in close collaboration with company representatives. All interviews were conducted in English and each interview lasted for about one hour. As part of our interview guide, we asked questions related to the following themes: (1) software evolution and introduction of CST's, (2) customer collaboration techniques and how CST's contribute to these, (3) customer feedback loops and responsiveness to emerging requests, (4) customer satisfaction and understanding of customer needs, and finally (5) feature quality and prioritization of functionality. For each theme, we had 2-4 questions and sometimes we also asked follow-up questions in order to clarify, or further explore, a particular topic. While the major part of the interviews were conducted on-site and face-to-face with the different team members and managers, interviews with the customer unit representatives were telephone interviews involving representatives in the US and in Asia. During all interviews, we were two people sharing the responsibility, i.e. one researcher asking the interview questions and taking notes, and one company representative from the main development site focusing on taking notes and adding to the discussion if needed, i.e. if there were misunderstandings in company specific information and/or difficulties in understanding company specific abbreviations etc. To address the risk of bias, and the presence of a company representative having a negative impact on the interviews, we made sure that all questions were asked by the researcher and that the company representative took a passive role. In retrospective, the presence of the company representative was helpful in that misunderstandings could be easily addressed and whenever a problem occurred in terms of interpretation of content we could have a direct dialogue. After each interview the notes were shared among the three researchers as well as the two company representatives involved in the study. This allowed for further elaboration on the empirical material, as well as any clarifications that were needed. In addition, two workshop sessions were held with a larger group of company representatives at the main development site in order to share, discuss and confirm the interview findings. Finally, e-mail correspondence was used as a follow-up in order to clarify any misunderstandings in the analysis of the data.

In terms of data analysis, the open coding technique originating from grounded theory was adopted [29]. Grounded theory is a method involving both inductive and deductive thinking and typically, the researcher wants to discover the participants' main concerns, how they understand a certain situation and how they try to resolve a problem they encounter. In the open coding process, written data from field notes or interview transcripts are conceptualized line by line by identifying codes. During our data analysis, the empirical material, i.e. the interview transcripts, were read through several times by all researchers. When reading, we looked for phrases, expressions, words etc. that described a certain phenomenon, a certain feeling or opinion, and we grouped those similar to each other together in categories (using different colors to keep different coding categories apart). For example, we found 'quality aspects' to be a reoccurring theme in several of the interviews, and there were many statements concerning the challenge of maintaining high quality code/architecture while also being fast and responsive to customers. All these aspects, i.e. expressions, experiences etc. around quality were noted in the margins of the interview transcript, and later grouped together in a category that we named 'feature quality'. In similar, statements and comments about interaction, response time and development cycles were coded and later grouped together in categories such as 'customer collaboration' and 'customer feedback loops'. After iterating the text among the researchers, and after having grouped together expressions and statements concerning similar phenomena, we had categories with words and expressions describing 'communication', 'coordination', customer satisfaction', 'customer collaboration', 'customer feedback loops', and 'feature quality'. These categories emerged as a result of the data analysis and as a result of working our way through the interview transcripts. The codes and the categories highlight the respondents' concerns, their experiences and their profound knowledge of the specific situation.

A problem that has been identified in relation to qualitative research is that different individuals may interpret the same data in different ways [30]. This problem was addressed in two ways. First, the grounded theory method of data analysis prescribes coding processes that provide a traceable, documented justification of the process by which conclusions are reached. Second, we used a 'venting' method, i.e. a process whereby results and interpretations are discussed with professional colleagues [31]. The findings were presented and discussed with academic colleagues and expert practitioners in detail after each interview and at the two workshop sessions.

C. Validity and generalizability of results

As noted by Maxwell [32], qualitative researchers rarely have the benefit of previously planned comparisons, sampling strategies, or statistical manipulations that control for possible threats. Instead, qualitative researchers must try to rule out validity threats after the research has begun by using evidence collected during the research itself to make alternative hypotheses or interpretations implausible. One important aspect of validity is construct validity [28] that reflects to what extent the operational measures that are studied represent what the researcher has in mind and what is reflected in the interview questions. To address this aspect, we started each interview with an introduction in which we shared our understanding of agile practices and software evolution with the interviewee. In this way, the researchers and the interviewee had a shared understanding of the topic before the interview. With respect to external validity, i.e. to what extent it is possible to generalize the findings, our contribution is related to (1) the drawing of specific implications and (2) the contribution of rich insight [27]. Based on our interviews, we present implications in a particular domain of action. Our study brings together empirical insight that allows for a deep understanding of the domain, and the findings we present should be regarded as insights valuable for other companies interested in how to maintain agile ways of working during software evolution.

IV. CASE STUDY FINDINGS

In this section, we present our interview findings. We do so by describing the experiences from the time before customer-specific teams were introduced, and until today's situation in which these teams have become a natural part of the development organization. We summarize our interview findings in **Table 1**. In this summary, we confirm the use of customer-specific teams as successful for improving customer responsiveness, customer satisfaction, and feature quality in the evolution phase of large-scale software development.

	Customer responsiveness	Customer satisfaction	Feature quality
Team A	・Faster feedback and response ・Closer to a specific customer ・Direct communication with a specific customer ・Easy to get in contact with the customer	・More frequent usability tests ・Strong support after delivery ・Regular meetings with customers and customer units ・Close interaction with customer units	・Continuous discussion about requirements ・Opportunity to discuss what customers really need ・More frequent acceptance tests
Team B	・Give important customers what they want ・Bypass the normal heavy process ・Meet special demands ・Extra attention to customer	・Better understanding of how features will be used/not used ・Be pro-active, foresee and anticipate problems ・Customers feel they are in control of what they get	・Opportunity to test in advance and in direct contact with a specific customer ・Test in field – learn more about specific needs
Team C	・Flexibility ・Faster deliveries ・Customer get a feature that is requested only by them ・Adapt faster to customer needs	・Special treatment ・Tailor features ・Understand customer use of a feature ・Customers get what they want when they want it	・More adapted and flexible processes and testing procedures
Customer Unit A	・Closer to developers ・Direct communication ・Short feedback loops	・Customers get extra attention ・Customers feel special	・Test directly in the field with a specific customer
Customer Unit B	・Faster deliveries ・Customers decide when to get a feature	・Better understanding of how a feature will be used/not used	・Learn what requirements mean instead of having assumptions
Product Manager	・Seamless interaction between developers and customers	・Additional opportunities to understand our customers	・Learn about specific needs ・Facilitate good testing
Program Manager	・Closer and more responsive ・Less disruptive ・Focus on one customer	・Only value-adding development	・Bundle customer-specific features
Integration Manager	・Special treatment for important customers ・Bypass traditional release cycles	・Better understanding of customers ・Develop the right things immediately	・Daily discussions ・Better understanding of quality

Table 1. Summary of the interview findings from the customer-specific teams, units, and managers included in the study.

A. Experiencing traditional practices

Our case company is involved in large-scale software development with a distributed team structure. For the purpose of this study, one development unit was involved and we got the opportunity to interview members of three customer-specific teams serving different customers. The customer-specific team members we interviewed are physically co-located, i.e. all team members work in the same physical and geographical location. A few years ago, the unit was involved in projects of a typically traditional nature. At that time, project teams were large and development cycles were long. The development organization was separated into expert disciplines, e.g. architecture, design, test, and maintenance, and the different disciplines were usually kept apart, communicating mainly via hand-overs of code or documentation. At that time, development was sequential with a rigorous planning phase in the beginning of each project and a major characteristic was that customers were involved during requirements engineering. In most cases, a separate organization took over the responsibility for system evolution. The challenges the organization experienced at that point were the following:

- Changes in requirements caused major re-work resulting in an expensive evolution process.

- Considerable time was spent on communication and coordination between different organizations, disciplines and teams, which made the evolution process slow.
- Evolution was concerned only with correcting errors and bugs, i.e. things that could have been done right if developers had closer collaboration with the customer.
- Feedback loops were slow and by the time a request or a bug report reached the development team this team was already involved in another project.
- Teams had difficulties in responding to customer-specific needs and requirements that emerged after product deployment.

B. *Adopting agile practices*

After experiencing the difficulties mentioned above, the organization started adopting agile practices. A major change was the introduction of cross-functional teams. Instead of having separate disciplines, the company re-organized into units in which a number of cross-functional teams operate. Each cross-functional team includes system design, development, test etc., allowing for each team to take full responsibility for a feature. Even though there were problems in the beginning, such as for example difficulties in adjusting to working closely with other disciplines, challenges in taking full responsibility of a feature, and uncertainties in how to stay flexible while at the same time meet with project milestones, the teams quickly got used to an environment characterized by incremental delivery, minimal up-front planning and continuous adjustment of plans. In taking full responsibility for the features they developed, the team members got the opportunity to stay involved during the entire development cycle of a feature and the problems associated with communication- and coordination in between disciplines decreased. In adopting agile practices, the company experienced a number of benefits such as shortened development cycles, closer customer collaboration during development of features, and more frequent deliveries of features. However, despite the adoption of agile practices, the company still experienced several challenges in relation to software evolution:

- Even though development cycles were shortened, the company had difficulties in maintaining short feedback loops after product deployment.
- The adoption of agile practices did not resolve the issue of how to maintain flexibility to changing customer requests after product deployment.

C. *Introducing customer-specific teams*

After having experimented with agile practices and cross-functional teams for a few years, the company decided to introduce customer-specific teams. Customer-specific teams are teams that work exclusively for a selected customer. Although the initial plan was to have the existing cross-functional team structure cater for both generic development as well as customer-specific requests, the company decided to introduce a number of designated teams that could be allocated exclusively for one customer if there was a need for improvement of functionality that was not part of the generic product. While the customer-specific teams operate within the existing structure and look as all other cross-functional teams in terms of competence, they are intended to significantly improve the company's ability to respond to customer-specific feature requests that emerge during the evolution phase. The long-term goal is to transfer knowledge gained from working closely with specific customers to the development of generic functionality that serves the large customer base. In this way, re-work efforts will decrease, the evolution process will become less expensive, and customer-first features will be efficiently transferred to the main product as part of the continuous modification and correction process. Our interviews show on a number of benefits associated with customer-specific teams:

- Faster feedback and direct communication with customers resulting in increased responsiveness.
- Flexibility to bypass planned release cycles allowing for faster deliveries and deployment of functionality.
- Increased team and customer satisfaction, as teams and customers have direct communication.
- Reduced disruption of work tasks, allowing team members to focus on the needs of one customer.
- Seamless environment in which customers feel there is room for their ideas and their feedback.
- Opportunity to learn about feature usage, allowing development teams to anticipate customer needs.
- Flexible testing procedures directed towards one customer.
- Continuous dialogue between customers and teams about emerging and changing requirements.

As part of our study, interviewees were asked to confirm the use of customer-specific teams (CST) for increasing customer satisfaction, feature quality, customer responsiveness etc., on a scale 0-6 (0 = completely disagree and 6 = completely agree). In **Figure 1**, we summarize our findings by showing the numbers assigned by the three customer-specific teams as well as the management representatives.

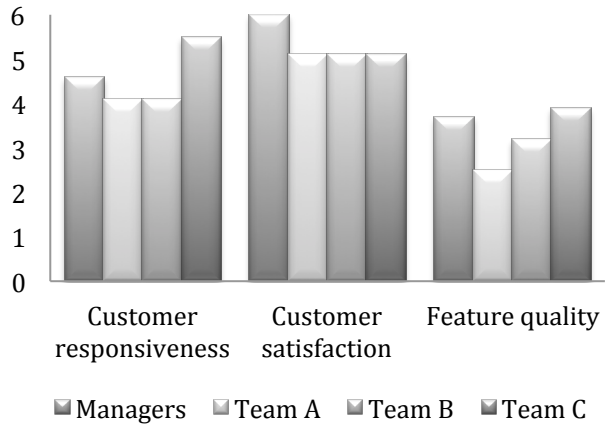

Figure 1. Team and management perspective on benefits with CST use during software evolution.

V. DISCUSSION

In this study, we explore the way in which benefits of agile practices can be established and maintained in the evolution phase of software development. In particular, we are interested in customer-specific teams and the way in which they can contribute to fast and responsive customer collaboration. Below, we present the findings related to agile principles and the benefits they provide, as well as customer-specific teams and the characteristics of these.

A. Reaping the benefits of agile practices

Traditionally, activities related to software evolution are viewed as expensive, resource demanding and time consuming [33]. There is significant research on cost estimation, effort estimation models and ways in which the different phases of software evolution can be efficiently managed [34]. Often, the evolution process is pictured as a 're-active' process in which software systems are adjusted in accordance with customer needs, but in which there is limited opportunities to embrace innovative ideas and continuous customer feedback.

In our study, we explore a company in which traditional practices was the norm for a long time. Large development projects, long development cycles and difficulties in handling changing customer requests after system delivery were familiar problems [21]. Also, and as is common in the area of software product lines [16], customer needs and requests of similar kind were grouped together in product-specific requirements, resulting in a number of different 'product families'. As time went by, however, the situation became too complex and the software evolution process was considered inefficient as well as unsatisfactory from both a developer, as well as from a customer perspective. At this point in time, the evolution process was re-active in nature with a focus on maintaining current functionality and only to some extent improving the existing product. As can be seen in our study, there are a number of problems associated with this approach, i.e. long feedback cycles, slow response time and difficulties in transferring customer-first features into generic functionality that serves a large customer base.

To address this situation, the company adopted agile development practices by introducing cross-functional teams with full responsibility for the features they develop. Instead of having separate disciplines and a complex organizational structure for customer collaboration, the teams have access to all competences needed as well as closer, and more frequent, collaboration opportunities with the customers. As can be seen in studies on agile development [2, 4], common benefits include shortened development cycles, more frequent delivery of features, and as a result of this, opportunities for more frequent customer feedback. In our study, we see that by introducing cross-functional teams with full responsibility for feature development, a number of benefits from agile development practices could be reaped, and instead of a discipline-oriented team structure, the company succeeded in establishing an autonomous team structure that could better respond to emerging customer needs during development as well as evolution of features.

B. Customer-specific teams for agile software evolution

While the adoption of agile practices addressed many of the problems experienced with traditional development, and significantly improved the overall situation at the company, the evolution phase remained a challenge. Especially, close customer collaboration was difficult to establish and maintain while at the same time achieving scale and responsiveness to requests from a large customer base. To further address this situation, customer-specific teams, i.e. teams that could be allocated to a specific customer for a period of time, were introduced. Based on findings from our study, we see that these teams significantly contributed to the evolution phase in the company we studied. First, team members and customer unit representatives highlight the many advantages of having a direct contact that allows for fast feedback loops once the system is deployed. The idea of not being disrupted, and the opportunity to focus on only one customer, is appreciated. The general feeling is that the way in which customer-specific teams allow for a close relation to a selected customer is satisfying, and with the opportunity to bypass the planned release cycle in order to deliver functionality more frequently, they add significantly to improved customer responsiveness during software evolution. Moreover, our interviewees recognize the opportunity to get continuous customer feedback on how functionality is used as a way for them to learn about feature usage. Managers involved in our study recognize the extra attention given to prioritized customers and they view this as important to enhance customer satisfaction. Likewise, and as critical in agile development [4], team members and customer unit representatives mention regular meetings, close interaction and continuous collaboration as vital for customer satisfaction after product deployment.

Finally, the interview study reveals interesting findings in relation to customer-specific teams and the opportunity to improve feature quality. The interviewees mention the frequency of acceptance tests, the opportunity to test in field, and the possibility to adapt test processes as rewarding. Also, both team members and customer unit representatives acknowledge the increased opportunity to continuously discuss and negotiate emerging requirements. While testing and evaluation of functionality is often considered a bottle neck in software development [35], and an activity that is both complex and insufficient, the impression after having heard customer-specific team members talk about it is that testing, on the opposite, is highly rewarding and something they appreciate if only circumstances allow for a collaborative and customer-oriented process.

VI. CONCLUSIONS

In this paper, we address the need to involve customers in the software evolution phase. In particular, we direct our attention to large-scale software development where companies struggle to meet the specific needs and requests of a large customer base. The contribution of this paper is two-fold. First, we propose customer-specific teams (CST) as a way to establish and maintain agile practices in the evolution phase of large-scale software development. By having designated teams working closely with selected

customers many of the benefits characterizing agile practices can be reaped. Customer-specific teams increase flexibility and speed, and as a result, facilitate efficient knowledge transfer between customer-first requests and generic functionality of interest for the large customer base. In our study, we identify a number of CST characteristics that lead to positive effects during software evolution:

- CST's have direct communication with customer units and allow for short feedback cycles and fast response to emerging requests.
- CST's have the opportunity to bypass the planned release cycle and adjust releases to one particular customer.
- CST's have the ability to focus on one task without being disrupted by other customer requests.
- CST's allow for a creative environment in which they continuously learn about customers and feature usage.

Second, our study confirms customer-specific teams as successful for improving (1) customer responsiveness, (2) customer satisfaction and (3) feature quality in the subsequent phases of software evolution.

REFERENCES

[1] N. D. Fogelström, T. Gorschek, M. Svahnberg, and P. Olsson, The Impact of Agile Principles on Market-Driven Software Product Development, Journal of Software Maintenance and Evolution: Research and Practice, 2010, Vol: 22, pp. 53-80.

[2] J. Highsmith, and A. Cockburn, Agile Software Development: The business of innovation, Software Management, September, 2001, pp. 120-122.

[3] K. Beck, Embracing Change with Extreme Programming. Computer, 1999, vol. 32, no. 10, pp. 70–77.

[4] C. Larman, C., Agile and Iterative Development: A Manager's Guide. Addison-Wesley, 2004.

[5] D. Mishra, D., and A. Mishra, Complex software project development: agile methods adoption, Journal of Software Maintenance and Evolution: Research and Practice, 2011, vol. 23, pp. 549–564.

[6] H. H. Olsson, H. Alahyari, and J. Bosch, Climbing the "Stairway to Heaven": A multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software, In Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications, September 5-7, Cesme, Izmir, Turkey, 2012.

[7] K. H. Bennett, V. T. Rajlich, and R. Mohamad Mazrul, Legacy System: Coping with success. IEEE Software, 1995, pp.19-23.

[8] A. Cockburn, Agile Software Development. Boston: Addison-Wesley 2002.

[9] J. Highsmith, The great methodologies debate: Part 2, Cutter IT Journal, 2002, vol. 5.

[10] P. Abrahamsson, J. Warsta, M. Siponen, and J. Ronkainen, New Directions on Agile Methods: a comparative analysis. In Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, 2003, pp. 244–254.

[11] K. Schwaber, and M. Beedle, Agile Software Development with Scrum. Upper Saddle River, NJ: Prentice-Hall, 2002.

[12] B. Fitzgerald, G. Harnett, and K. Conboy, Customizing Agile Methods to Software Practices. European Journal of Information Systems, 2006, vol 15, no. 2.

[13] S. Cook, R. Harrison, M. M. Lehman, and P. Wernick, Evolution in Software Systems: Foundations of the SPE Classification Scheme. J. Softw. Maintenance & Evolution: Research. and Practice, 2006, vol 18, no. 1, pp. 1–35.

[14] N. Chapin, J. E. Hale, Md. K. Khan, F. Ramil, and W-G. Tan, Types of software evolution and software maintenance, Journal of Software Maintenance and Evolution: Research and Practice, 2001, vol. 13, pp. 3-30.

[15] J. Bosch, Software ecosystems; Taking software development beyond the boundaries of the organization, Editorial in the Journal of Systems and Software; Special issue on Software Ecosystems, 2012, vol. 85, no. 7.

[16] J. Bosch, Design and use of software architectures: adopting and evolving a product-line approach, Addison-Wesley, 2000.

[17] C. Jones, Strategies for managing requirements creep, IEEE Computer, 1996, vol. 9, no. 6, pp. 92-94.

[18] V. Rajlich, Changing the paradigm of software engineering, Communications of the ACM, 2006, vol. 49, no. 8, pp. 67-70.

[19] J. Iivari, H. Isomäki, and S. Pekkola, The User – the great unknown of systems development: reasons, forms, challenges, experiences and intellectual contributions of user involvement, Editorial in Information Systems Journal, 2010, vol. 20, pp. 109-117.

[20] A.M. Davis, Software Requirements, Prentice-Hall: New Jersey, 1993.

[21] I. Sommerville, Software Engineering, 6th Edition, Pearson Education: Essex, England, 2001.

[22] Boehm, B.W. (1989). Software Risk Management. IEEE Computer Society Press.

[23] M. Neumann, A. Riel, and D. Brissaud, IT-supported innovation management in the automotive supplier industry to drive idea generation and leverage innovation, Journal of Software Maintenance and Evolution: Research and Practice, 2011, July.

[24] J. Bosch, Maturity and evolution in software product lines: Approaches, artefacts and organization, In Proceedings of the Second Software Product Line Conference (SPLC2), 2002, pp. 257-271.

[25] J.M. LaNasa, Building customer teams to deliver on your company's value proposition, Velocity, 2002.

[26] M. Poppendieck, and T. Poppendieck, Lean Software Development; An agile toolkit, Addison-Wesley, 2003.

[27] G. Walsham, Interpretive case studies in IS research: Nature and method, European Journal of Information Systems, 1995, vol. 4, pp. 74-81.

[28] P. Runesson and M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering, 2009, vol, 14.

[29] J. Corbin and A. Strauss, Basics of Qualitative Research: Grounded Theory Procedures and Techniques, Sage, California, 1990.

[30] B. Kaplan and D. Duchon, Combining qualitative and quantitative methods in IS research: A case study, MIS Quarterly, 1988, vol, 12, no, 4, 571–587.

[31] J. Goetz and D. LeCompte, Ethnography and Qualitative Design in Educational Research, Academic Press, Orlando, 1984.

[32] J.A. Maxwell, Qualitative research design: An interactive approach, Sage: Los Angeles, 2013.

[33] B. P. Lientz and E. B. Swanson, Software Maintenance Management, A study of the maintenance of computer application software in 487 data processing organizations. Addison-Wesley, Reading MA, 1980.

[34] A. De Lucia, E. Pompella and S. Stefanucci, Effort estimation for corrective software maintenance, In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, 2002.

[35] S. McConnell, Code Complete (2nd ed.). Microsoft Press, 2004, p. 29.