Industrial Challenges of Scaling Agile in Mass-Produced Embedded Systems

Ulrik Eklund[1], Helena Holmström Olsson[1], and Niels Jørgen Strøm[2]

[1] Dept. Computer Science, School of Technology, Malmö University
SE-205 06 Malmö, Sweden
{ulrik.eklund,helena.holmstrom.olsson}@mah.se
[2] Grundfos A/S
DK-8850 Bjerringbro, Denmark
njstroem@grundfos.com

Abstract When individual teams in mechatronic organizations attempt to adopt agile software practices, these practices tend to only affect modules or sub-systems. The short iterations on team level do not lead to short lead-times in launching new or updated products since the overall R&D approach on an organization level is still governed by an overall stage gate or single cycle V-model.

This paper identifies challenges for future research on how to combine the predictability and planning desired of mechanical manufacturing with the dynamic capabilities of modern agile software development. Scaling agile in this context requires an expansion in two dimensions: First, scaling the number of involved teams. Second, traversing necessary systems engineering activities in each sprint due to the co-dependency of software and hardware development.

Keywords: software engineering, agile development, agile methods, large-scale agile software development, project management, embedded systems, embedded software, software and hardware co-dependency.

1 Introduction

The embedded systems industry is currently in significant transition, i.e. markets becoming more fast-changing and unpredictable, customer requirements becoming increasingly complex, rapidly advancing technologies and the constant need to shorten time-to-market of new products. Moreover, while the ability to manufacture high-quality mechanical systems is still critical, it is no longer the only differentiator and what makes a company competitive. During the last two decades, electronics and software have been introduced into many products, and embedded systems companies are becoming increasingly software-intensive with software being the key differentiator [1]. This requires a significant shift in the ways-of-working within these companies, and currently many large companies within the embedded systems domain struggle with the alignment of hardware and software development cycles and practices [1].

T. Dingsøyr et al. (Eds.): XP 2014 Workshops, LNBIP 199, pp. 30–42, 2014.

In response to this, agile methods advocating flexibility, efficiency and speed are seen as an increasingly attractive solution [2], and highly relevant also in the embedded Systems domain [3]. Typically, agile methods emphasize the use of short iterations and incremental development of small features, with the intention to increases the ability for companies to accommodate fast changing customer requirements as well as turbulent and fluctuating market needs [2,4].

However, when agile practices are introduced for software teams in a mechatronics environment without careful consideration it just results in different development cycle-times for hardware, and even more so in mechanics, compared to software development, with the longest cycle determining the lead-time for the complete product. For production equipment depending on the product design it becomes even worse, because investments and lead-times for the manufacturing setup are even more difficult to do with short iterations. This is the main difference between scaling agile in domains where only software teams are concerned, and in embedded domains also concerned with mechanical design and manufacturing.

This position paper presents a set of research challenges relevant when agile practices are scaled beyond a single team in organisations developing and delivering mass-produced embedded systems and into combining mechanical, hardware and software disciplines in the agile practices. The challenges are based on concerns of member companies in a Nordic research partnership with eight international industry companies and three universities.

2 Empirical Data

The main empirical data source for the challenges identified in this paper was a workshop conducted within a Nordic software research partnership[1] in November 2013, where seven companies presented their most important research challenges within software engineering. Three of the companies mentioned a set of challenges with agile development in large organisations as a top priority for future research within the partnership. The challenges presented in this paper are a synthesis of these presented challenges, elucidated by examples from two of the companies; Grundfos and Volvo Cars.

Grundfos is the world's largest manufacturer of circulator pumps, many controlled by embedded software. The examples provided for this paper from Grundfos serves to highlight the challenges all companies experienced in combining agile and waterfall development in a mechatronics environment; where there are different development cycle-times for hardware, and even more so in mechanics, compared to software. For production equipment depending on the product design it becomes even worse because investments and lead-times for the manufacturing setup are even more difficult to do with short iterations.

The synthesized challenges were also corroborated by data from three in-depth case studies on agile development at Volvo Cars, another of the partnership companies; the cases being published in e.g. [5].

[1] `http://www.software-center.se`

3 Background

3.1 Software in Embedded Products

Software is prevalent in many products manufactured today; cars, washing machines, mobile phones, airplanes, satellites and industrial devices, e.g. pumps [6]. The embedded software controls the behaviour of the product and is most often critical for the success of the product. Typically these products are developed in large, and sometimes very complex, industrial projects with a more or less elaborate R&D process governed by a stage gate model to arrive at the finished design of the product. Even though many companies are in a transition towards delivering services deployed on already delivered hardware and mechanics, they still heavily rely on the financial transaction taking place when the physical product goes from the company to the customer.

The software in an embedded system increases in size exponentially over time [6], and software is increasingly so being a crucial element and one of the most important drivers for innovations, e.g. in the car industry [7] and the pump industry. But the manufacturing and delivery setup of a new car or pump model is presently still a heavier investment than the software budget. A product example from Grundfos shows that the software budget for a new pump was between 5 and 10% of the total project investments.

The most common approach to develop embedded software, according to a mapping study [8], is to use an integration-centric approach, summarized as: Early in the development cycle requirements are allocated to software and hardware components. This is usual done by a central systems engineering team or architect. A number of development teams then implement the requirements allocated to their component. All of the teams are usually synchronized according to a common project model. After the finalization of the components the integration phase starts where all components are integrated to form the complete systems and system level testing takes place, where most integration problems are found and resolved [9]. This cycle may repeat 1-5 times, and it is common that the integration points in time are scheduled according to a stage-gate model [10]. An integration cycle is typically six months or more, meaning that a development project can have a lead-time of multiple years.

Example from the Pump Domain. Grundfos typically defines 4 to 6 review series of a PCB design to ensure quality before launching a new product. This approach is agile on the team level, and every cycle takes between 4 and 8 weeks dependent on complexity and where in the process the cycle takes place.

At the system integration level (integrating the product with other related products in a system) an example from Grundfos of a complex system showed a total integration and test phase of 9 months with additional bug fix cycles afterwards - giving a single integration and test phase of approximately a year. This leads to a lead-time before production could start measured in years, rather than months.

3.2 Mechanical and Hardware Development

The typical culture of company with a heavy tradition of mechanical engineering is to focus on predictability and doing so by trying to foresee activities many months ahead because of the constraints linked to mechanical manufacturing. Traditionally mechatronics manufacturing companies freeze the design at a certain point in a stage-gate model and after that the mechanical design does not change, instead focus it's activities on optimizing the manufacturing, sales and delivery processes. The purpose of the stage gate model is, at certain stages, to ensure the feasibility of releasing large investments, not only for development but in particular for manufacturing. Developing a mechanical part for a product often includes developing and investing in very expensive manufacturing tools with long lead times, expanding the development cycle for mechanics to up to 12 months or more. If the company is already established in a mature domain, e.g. the car industry, these type of activities are highly optimized, with much know-how of the company directed to running such projects.

Software may be strongly dependent on mechanical structures because of software modelling etc. and since there is a very weak link between software and mechanics cycle times (typically weeks vs. many months) the final verification of the software/mechanics interface cannot take place until much later, even if models, simulations and fast prototyping such as e.g. 3D printing is utilized. Sometimes this late verification can lead to less optimal solutions where issues are solved in software even though they would have been better solved in mechanics, had it been possible to use an agile approach.

3.3 Agile Software Development

For more than a decade, agile development methods have gained much popularity and become widely recognized within the field of software engineering. The methods promise shorter time-to-market, as well as higher flexibility to accomodate changes in requirments and thereby, increase companies' ability to react and respond to evolving customer and market needs [4,11,12]. While there are a number of different agile methods, they typically emphasize close customer collaboration, iterative development and small cross-functional development teams. Also, team autonomy and end-to-end responsibility are reported as important characteristics permeating the methods [13]. As recognized by Kettunen and Laanti [14], the concept of agile is multi-dimensional, and there are many reasons for companies to adopt agile ways-of-working. Typically, most companies introduce agile methods to increase the frequency in which they release new features and new products, and as a way to improve their software engineering efficiency. According to Dingsøyr et al. [15] agility embraces lean processes with an emphasis on realizing effective outcomes, and common for agile methods is that they entail the ability to rapidly and flexibly create and respond to change in the business and technical domains [15].

Today, there exist a number of different agile methods, with Extreme programming (XP) and Scrum being the two most common ones. XP focuses on

the programming practice itself and prescribes a set of practices for developers, e.g. pair programming and continuous unit testing. In addition, it includes practices such as user stories and iterative planning as a support for management in their requirements prioritization processes [16]. Scrum, on the other hand, focuses more on the process for the development team, i.e. how to prioritize, track and optimize team performance, and how to continuously evaluate and follow-up with the customer what is being implemented [17]. Although different in focus, both these methods emphasize the importance of working in short sprints, to constantly reprioritize what is being developed, and to test and validate new software functionality in rapid cycles.

Originally, agile methods evolved to meet the needs of small and co-located development teams [14]. Currently, and due to many successful accounts [18,19] agile methods have become attractive to a broad variety of companies, including companies involved in large-scale development of embedded systems, and there are attempts such as Industrial XP and Scrum of Scrums aiming at scaling agile methods [20]. However, with characteristics such as hardware-software interdependencies, heavy compliance to standards and regulations, and limited flexibility due to real-time functionality [21], development of embedded systems challenges the traditional concept of agile practices.

3.4 Agile Development of Embedded Software

Currently, companies producing embedded systems are in the process of deploying agile methods, and several attempts to scale agile methods to include development of mass-produced systems can be identified [22,20,23].

Some organizations developing mass-produced system have successfully introduced agile development on the team level where individual teams are allowed to define their own ways of working to facilitate speed, short iterations and delivery quality when developing their components. The experiences of doing this are generally positive according to two literature studies by [3] and [24].

However, the applicability of agile methods is not without challenges in large-scale development of software intended for mass-produced systems [25]. Companies also often discover misalignments between the agile methods and their already established ways-of-working when attempting to adopt agile practices in a large-scale setting [26]. One reason is that many large-scale development companies practice agile in a way that is not consistent with the original agile ideas, and that the translation of the original ideas to a context of mass-production is difficult.

Ronkainen and Abrahamsson [27] identified four main characteristics that would affect adoption of agile methods under strict hardware constraints, typical of most embedded systems:

- Meeting hard real-time requirements, e.g. performance
- Experimenting is part of the systems development, many technological constraints are difficult to ascertain until actual hardware and mechanics is available.

- High-level designs and executable documentation are not sufficient, information shared between teams tend to be detailed and implementation-specific
- Embedded development is test driven by nature, but some of the core ideas of agile are problematic to implement when doing software/hardware co-design (e.g. write tests first, run every unit test at least daily)

Greene [28] describes how elements from Scrum and XP were used in a firmware project to deal with changing hardware interfaces for a new family of 64-bit microprocessors. Some of the constraints they had to satisfy were; consistent firmware interfaces across the entire processor family, architecture features that are better, cheaper, or more flexibly implemented in firmware than hardware, and workarounds for processor errata. Some of the challenges they had to deal with were

- Turnaround time for silicon from the factory of more than a month.
- Detailed quarterly planning of schedules, which quickly became obsolete.
- Too specialized team members will little cross-domain firmware knowledge.
- lack of test coverage, and no regression tests when changes were made. Reliance on outside groups to find problems.
- Poor code maintainability, due to overly optimized and complex code.

Cordeiro et al. [29] proposes an agile method for developing embedded software under stringent hardware constraints. The aim to: Resolve the trade-off between flexibility and performance, fulfill hardware constraints, support a flow from specification to implementation, propose novel test techniques, and use an incremental approach where the developer can validate a system specification in each iteration. They solve this by proposing three sets of parallel processes organized in three process groups: System Platform Processes, Product Development Processes, and Product Management Processes. The method assumes that a system designer chooses the system components from an already existing platform library to instantiate a given product. Both this and the previous example only concerned a very limited number of involved developers, less than 10 developers in 1-2 teams.

A conclusion is that teams in an integration-centric organization that attempt to adopt agile software practices have difficulties in scaling them beyond the team level. The adopted agile practices typically only affect modules or sub-systems, as seen in figure 1 below. The product as a whole is still developed with an integration-centric approach, as described in section 3.1, with the mechanics and manufacturing schedules also controlling the software development.

Even if agile teams try to follow a platform-oriented approach focusing on developing prioritized cross product features, individual stage-gate projects still require a certain amount of functionality bound to product-specific hardware and mechanics. This makes the agile overall prioritization process difficult to perform. One or two major products with large investments can draw all the attention making it difficult to do the right prioritization of feature development across the full range of products.

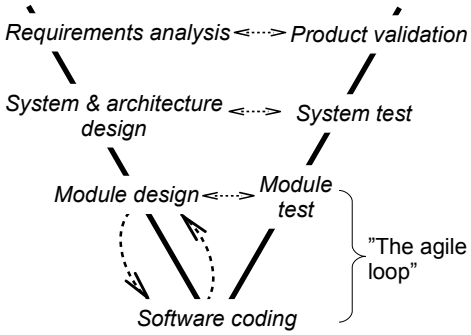

Fig. 1. The agile iteration on team level seen in the context of a typical systems engineering approach

The shift towards agile is complex for companies developing embedded systems since they are often used to heavyweight sequential processes also outside of R&D; additional challenges are e.g:

- dependencies to a number of suppliers and sub-contractors [5], with some software subcontractors tied up in sourcing agreements,
- software interfacing with hardware and mechanics, and
- certification processes.

As a result the development teams need to spend effort to align the internal team practices to the overall product development and release processes (see e.g. [30]). All this also means that the short iterations on team level do not lead to short lead-times in launching new or updated products.

4 Industrial Challenges of Scaling Agile

The key agile principle of delivering software frequently [31] contrasts with the situation described in section 3.4.

The long-term prediction and associated lead-times forced upon software development teams in this context leads to lack of flexibility in case market needs change during the development project. If an organisation was fully adhering to all agile principles it would in theory be possible to deploy new software throughout the entire life-cycle of the product if economically viable.

Not being able to exploit agile software development and adapting stage-gate models to agile software development also leads to a continuation of notoriously poor predictability when developing software, something which is prevalent also for embedded software.

4.1 Challenge of Uniting Agility with Stage-Gate Development

The principal challenge is how to combine the planning and achieved predictability associated with mechanical manufacturing with the dynamic planning

capabilities of modern agile software development; in practice this means large mechatronics companies need to solve the challenge of how to scale agile software development beyond short iterations on the team level.

This challenge is a major obstacle to allowing differentiated lead-times towards start of production (SOP) depending on the size or complexity of the wanted software features rather than depending on investments in mechanical manufacturing according to a stage-gate process. Rephrased; it means that while the start of the product project is demanded by the activities necessary for mechanical and manufacturing development, the development of a specific software feature can start independently of this while still aiming at the same SOP, as seen in figure 2.

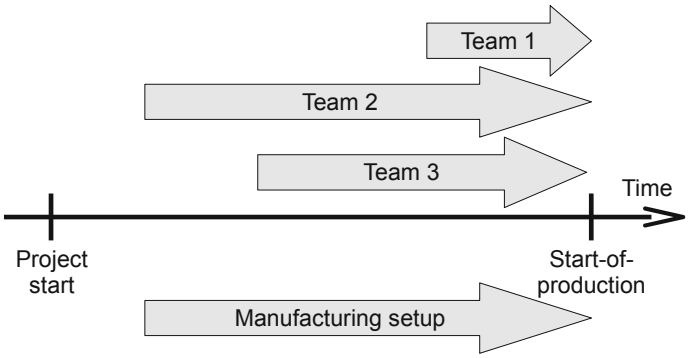

Fig. 2. Different sub-projects are allowed differentiated lead-times towards Start-of-Production (or start-of-deployment)

4.2 Challenge of Scaling the Number of Involved Teams

Scaling agile in this context is a challenge in two dimensions, as seen in figure 3: First, scaling the number of involved teams, this is usually what ?scaling? in the context of agile means. Second, scaling up the necessary system engineering activities in the iterations/sprints prescribed by different agile methodologies.

A complex product today, e.g. a car, has up to a hundred development teams doing software and embedded development, and twice that numer of teams doing mechanical development. Currently these teams are synchronized by all adhering to the same schedule according to a stage-gate process. The need for such large-scale development requires mid range and long range planning mechanisms beyond the standard sprint pattern of plan/commit, execute, and demo/adapt used for individual teams [32]. Typically such mechanisms involve release planning and road mapping of product portfolios, as described by the Scaled Agile Framework [32] or by Disciplined Agile Delivery [33].

The Scaled Agile Framework[2] presents guidelines on how to plan releases when demanded, while the individual teams work and deliver continuously in agile iterations. The involved teams are part of an agile release train that provides the program-level value according to the program backlog. These program backlogs

[2] `http://scaledagileframework.com/`

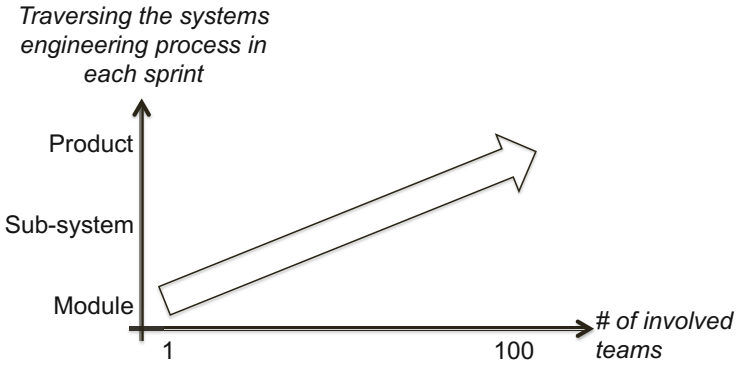

Fig. 3. Scaling agile in the context of mass-produced embedded systems is a challenge in two dimensions

are prioritized according to a portfolio backlog that realize the value streams that proved a continuous flow of value to the business, customer or end user.

However, existing large-scale agile methodology frameworks such as these do not address the challenges particular to the embedded domain (identified by e.g. [27]), and especially not all system engineering challenges regarding large-scale manufacturing.

4.3 Challenge of Scaling System Engineering Activities

The second dimension in figure 3 is traversing the systems engineering process in each sprint, i.e. not being confined to iterate each module separately in each sprint, but also allow re-prioritization of system-wide features and properties. This means that each team must have the ability or support to perform activities at all abstraction levels in the V-model in figure 1, for example doing system wide tests. This second dimension is what distinguishes agile development in mass-produced embedded systems, and can be considered the novel research challenge.

A trivial example of a system engineering activity in a sprint would be if it is necessary to have access to a physical property, such as fluid flow or temperature, in order to realize a specific feature. A system engineering choice would be to either try to estimate this based on other data or to use a sensor to directly measure the physical value with higher accuracy. The former choice could be implemented purely in software, while the latter would entail a change in the physical and electronics design of the system, incurring a cost, and possibly a lead-time, penalty. In a safety-critical system both choices may be necessary to implement for redundancy.

A related difficulty in this dimension concerns cross-functional team expertise and component interdependencies [25]. Usually, organizations realize that many components in a large-scale system are technically very difficult and interdependent, and require years of experience to be fully understood by developers. To solve this they therefore often organize in specialist or component teams

with exclusive access rights to key components occasionally leading to bottleneck situations. This is in contrast with the basics of agile where teams are self-contained and are able to solve their tasks independently in each sprint. As a result, many large-scale organizations experience long lead times before the development teams can implement anything useful in a component.

5 Discussion on Solutions

Our preliminary assumption is that the solutions to the challenges above not only lies within the process dimension, it is a question of implementing agile practices on an enterprise scale. We therefore expect a holistic approach is needed, weighing in business, architectural, and organizational aspects, besides scaled processes.

Typical software architectures for embedded systems are monolithic, having a static structure for every instantiation and variation is achieved by variation points in the components, usually by de-selecting code. An architecture that supports continuous integrations, including system tests, must probably be based on composition instead, allowing a creative selection and configuration of components and most of the tailoring towards specific products is achieved through different component configurations developed by various agile teams.

Some other architectural patterns supporting large-scale agile systems development would be suitable hardware device abstractions, and mechanism allowing for device composition supporting necessary security and safety integrity levels. However, monitoring of architectural and/or organizational dependencies and subsequent actions to resolve these dependencies is necessary. For example, even if proper hardware abstractions are made, new functionality may require new low level features to be implemented by the aforementioned component/specialist team causing dependency problems to the team implementing a new customer feature. Causes for this could both be architectural and organizational. Remedies to consider for mitigating this could be refactoring, spreading the necessary knowledge, establishing mentors to be able to immediately stand in and facilitate what is necessary, establish task force capacity, and these activities need to be iterative as well.

Martini et al. [34] identifies factors that inhibit the speed of organisations with a large number of small, independent and fast teams. The teams suffer from excessive inter-team interactions, which may lead to paralysis. Some of their recommendations to manage such factors, complementing current agile practices, are establishing cross-team roles with part-time domain experts and architects, and allow for programmed available time for other concerns, and not only synchronizing e.g. planning among SCRUM masters.

6 Summary

Mechanical and manufacturing development have very long lead-times compared to software development iterations of 2-6 weeks, reconciling this is a challenge when shortening lead-times towards start-of-production. The goal would be to

allow differentiated lead-times towards SOP depending on the size or complexity of the wanted software features, i.e. the development of a specific software feature can start independently of other features while still aiming at the same SOP. Related to this overall challenge we identified a number of additional challenges:

- Embedded system companies have already established ways-of-working for systems engineering which need to be considered.
- Individual stage gate product projects still require a certain amount of functionality bound to product-specific hardware and mechanics making a platform approach with agile overall prioritization difficult to perform.

Scaling agile software development in this context is then a question of scaling agile in two dimensions: First increasing the involved number of teams and utilize agile practices for mid- and long-range planning such as release planning and road mapping. Many large-scale development companies practice agile in a way that is not consistent with the original agile ideas, and that the translation of the original ideas to a context of mass-production is difficult. This is already a growing research field, as seen in [35], which gives some examples of smaller challenges:

- To coordinate work between agile teams.
- To effectively structure the organization and collaborate in large projects, especially when the organization is distributed.
- To plan large projects and control the scope.
- To understand the role of architecture in large-scale agile.

Second, scaling the system engineering activities executed in each sprint, to a truly iterative practice instead of a stage-gated planned approach. A cross-functional team must have the ability or support to perform activities at several abstraction levels in a systems engineering V-model in each iteration or sprint. This is a novel challenge, particular to the embedded domain. We can see a set of associated challenges that needs to be addressed in this domain, regardless of project size:

- Embedded systems have specific product requirements, e.g. safety, which are not obviously addressed by agile practices such as XP or Scrum.
- The feedback loop with customers and management is quite long due to the business model of delivering physical products in exchange of a financial transaction, and manufacturing constraints how short this can be.
- Mechanical and manufacturing development emphasises long-term predictability, and is usually successful in achieving this. This contrasts with the desire of short-term agility and poor long-term predictability of software development.
- Component interdependencies affect cross-functional teams requiring special expertise. Components in a large-scale system are technically very difficult and interdependent, and require years of experience to be fully understood by developers.

The call to action is to broaden the research on scaling agile to address the identified challenges particular to developing mass-produced embedded systems, and thus solving actual industrial needs.

References

1. Bosch, J., Eklund, U.: Eternal embedded software: Towards innovation experiment systems. In: Dingsøyr, T., Moe, N.B., Counsell, S., Tonelli, R., Gencel, C., Petersen, K. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 19–31. Springer, Heidelberg (2012)
2. Dzamashvili Fogelström, N., Gorschek, T., Svahnberg, M., Olsson, P.: The impact of agile principles on market-driven software product development. Journal of Software Maintenance and Evolution: Research and Practice 22(1), 53–80 (2010)
3. Albuquerque, C.O., Antonino, P.O., Nakagawa, E.Y.: An investigation into agile methods in embedded systems development. In: Murgante, B., Gervasi, O., Misra, S., Nedjah, N., Rocha, A.M.A.C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2012, Part III. LNCS, vol. 7335, pp. 576–591. Springer, Heidelberg (2012)
4. Williams, L., Cockburn, A.: Agile software development: It's about feedback and change. Computer 36(6), 39–43 (2003)
5. Eklund, U., Bosch, J.: Applying agile development in mass-produced embedded systems. In: Wohlin, C. (ed.) XP 2012. LNBIP, vol. 111, pp. 31–46. Springer, Heidelberg (2012)
6. Ebert, C., Jones, C.: Embedded software: Facts, figures, and future. Computer 42(4), 42–52 (2009)
7. Broy, M.: Challenges in automotive software engineering. In: Proceedings of the International Conference on Software Engineering, Shanghai, China, pp. 33–42. ACM (2006)
8. Eklund, U., Bosch, J.: Archetypical approaches of fast software development and slow embedded projects. In: Proceedings of the Euromicro Conference on Software Engineering and Advanced Applications, Santander, Spain, pp. 276–283. IEEE (2013)
9. Bosch, J., Bosch-Sijtsema, P.: From integration to composition: On the impact of software product lines, global development and ecosystems. Journal of Systems and Software 83(1), 67–76 (2010)
10. Cooper, R.: Stage-gate systems: A new tool for managing new products. Business Horizons 33(3), 44–54 (1990)
11. Larman, C., Vodde, B.: Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum, 1st edn. Addison-Wesley Professional (2008)
12. Highsmith, J., Cockburn, A.: Agile software development: The business of innovation. Computer 34(9), 120–127 (2001)
13. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology 50(9-10), 833–859 (2008)
14. Kettunen, P., Laanti, M.: Combining agile software projects and large-scale organizational agility. Software Process: Improvement and Practice 13(2), 183–193 (2008)
15. Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N.B.: A decade of agile methodologies: Towards explaining agile software development. Journal of Systems and Software 85(6), 1213–1221 (2012)
16. Beck, K.: Extreme programming: A humanistic discipline of software development. In: Astesiano, E. (ed.) ETAPS 1998 and FASE 1998. LNCS, vol. 1382, pp. 1–6. Springer, Heidelberg (1998)
17. Schwaber, K.: Scrum development process. In: Proceedings of the ACM Conference on Object Oriented Programming Systems, Languages, and Applications, pp. 117–134 (1995)

18. Abrahamsson, P., Warsta, J., Siponen, M., Ronkainen, J.: New directions on agile methods: A comparative analysis. In: Proceedings of the International Conference on Software Engineering, pp. 244–254 (2003)
19. Holmström Olsson, H., Alahyari, H., Bosch, J.: Climbing the "stairway to heaven". In: Proceeding of the Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Izmir, Turkey (2012)
20. McMahon, P.: Extending agile methods: A distributed project and organizational improvement perspective. In: Systems and Software Technology Conference (2005)
21. Kaisti, M., Mujunen, T., Mäkilä, T., Rantala, V., Lehtonen, T.: Agile principles in the embedded system development. In: Cantone, G., Marchesi, M. (eds.) XP 2014. LNBIP, vol. 179, pp. 16–31. Springer, Heidelberg (2014)
22. Kerievsky, J.: Industrial XP: Making XP work in large organizations. Executive Report. Cutter Consortium, 6(2) (2005)
23. Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., Stahl, D.: The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, MD, USA, pp. 348–356 (2013)
24. Shen, M., Yang, W., Rong, G., Shao, D.: Applying agile methods to embedded software development: A systematic review. In: Proceedings of the International Workshop on Software Engineering for Embedded Systems, pp. 30–36. IEEE (2012)
25. Heikkilä, V.T., Paasivaara, M., Lassenius, C., Engblom, C.: Continuous release planning in a large-scale scrum development organization at ericsson. In: Baumeister, H., Weber, B. (eds.) XP 2013. LNBIP, vol. 149, pp. 195–209. Springer, Heidelberg (2013)
26. Badampudi, D., Fricker, S.A., Moreno, A.M.: Perspectives on productivity and delays in large-scale agile projects. In: Baumeister, H., Weber, B. (eds.) XP 2013. LNBIP, vol. 149, pp. 180–194. Springer, Heidelberg (2013)
27. Ronkainen, J., Abrahamsson, P.: Software development under stringent hardware constraints: Do agile methods have a chance? In: Marchesi, M., Succi, G. (eds.) XP 2003. LNCS, vol. 2675, pp. 73–79. Springer, Heidelberg (2003)
28. Greene, B.: Agile methods applied to embedded firmware development. In: Proceedings of the Agile Development Conference, pp. 71–77. IEEE Computer Society (2004)
29. Cordeiro, L., Mar, C., Valentin, E., Cruz, F., Patrick, D., Barreto, R., Lucena, V.: An agile development methodology applied to embedded control software under stringent hardware constraints. SIGSOFT Softw. Eng. Notes 33(1), 5:1–5:10 (2008)
30. Karlström, D., Runeson, P.: Integrating agile software development into stage-gate managed product development. Empirical Software Engineering 11(2), 203–225 (2006)
31. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for agile software development (2001)
32. Leffingwell, D.: Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, 1st edn. Addison-Wesley (2011)
33. Ambler, S.W., Lines, M.: Disciplined Agile Delivery, 1st edn. IBM Press (2012)
34. Martini, A., Pareto, L., Bosch, J.: Improving businesses success by managing interactions among agile teams in large organizations. In: Herzwurm, G., Margaria, T. (eds.) ICSOB 2013. LNBIP, vol. 150, pp. 60–72. Springer, Heidelberg (2013)
35. Dingsøyr, T., Moe, N.B.: Research challenges in large-scale agile software development. SIGSOFT Softw. Eng. Notes 38(5), 38–39 (2013)