Optimising agile development practices for the maintenance operation

nine heuristics

Heeager, Lise Tordrup; Rose, Jeremy

Published in:
Empirical Software Engineering

DOI (link to publication from Publisher):
10.1007/s10664-014-9335-7

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Heeager, L. T., & Rose, J. (2015). Optimising agile development practices for the maintenance operation: nine heuristics. *Empirical Software Engineering*, *20*(6), 1762-1784. https://doi.org/10.1007/s10664-014-9335-7

Empir Software Eng
DOI 10.1007/s10664-014-9335-7

Optimising agile development practices for the maintenance operation: nine heuristics

Lise Tordrup Heeager · Jeremy Rose

Abstract Agile methods are widely used and successful in many development situations and beginning to attract attention amongst the software maintenance community – both researchers and practitioners. However, it should not be assumed that implementing a well-known agile method for a maintenance department is therefore a trivial endeavour - the maintenance operation differs in some, important respects from development work. Classical accounts of software maintenance emphasise more traditional software engineering processes, whereas recent research accounts of agile maintenance efforts uncritically focus on benefits. In an action research project at Aveva in Denmark we assisted with the optimisation of SCRUM, tailoring the standard process to the immediate needs of the developers. We draw on both theoretical and empirical learning to formulate seven heuristics for maintenance practitioners wishing to go agile.

Keywords Software maintenance · Agile methods · SCRUM · Action research

1 Introduction

Software developers have for more than a decade been concerned with a change of perspective in favour of the lightweight flexible approaches incorporated in agile ideas and practices (Senapathi 2010; Conboy 2009; Dybå and Dingsøyr 2008; Souza et al. 2005). Agile methods, including Crystal Clear, Agile Unified Process, Lean Development, Dynamic Systems Development Methods, Scrum and XP have become increasingly popular for development projects (Sidky et al. 2007), with commercial adoption outstripping the ability of researchers to keep up (Abrahamsson et al. 2009). The research evidence for the success of these approaches remains slim, though there is work addressing success factors (Misra et al. 2010); nevertheless agile is clearly here to stay, and agile is a accepted option for development projects.

Communicated by: Hakan Erdogmus

L. T. Heeager (✉)
Department of Business Administration, Aarhus University, 8000 Aarhus C, Denmark
e-mail: lith@asb.dk

J. Rose
Department of Informatics, Skövde University, PO Box 408, SE-54128 Skövde, Sweden
e-mail: jeremy.rose@his.se

J. Rose
Department of Computer Science, Aalborg University, Aalborg, Denmark

Published online: 11 October 2014

Lehman's first law of software evolution states that 'a program that is used undergoes continual change or becomes progressively less useful' (Lehman 1980). Therefore most software systems are maintained and further developed after their initial development. According to the IEEE, software maintenance is defined as: the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment (Standards Coordinating Committee of the Computer Society of the 1990). Maintenance, though under-researched, is by far the predominant activity in software engineering. The majority of commercial projects are built over a previously developed codebase (Svensson and Host 2005), and maintenance represents typically 90 % of the total cost of a typical software product (de Souza et al. 2006). Cockburn (2006) argues that agile methods represent an evolutionary kind of development in which maintenance activities are subsumed in an iterative sequence of software deliveries, but there is no evidence that this has become common practice. Commercial software projects most typically have an agreed delivery point, after which the system is then moved on to maintenance, which is often carried out by specialised maintenance engineers, working in their own department. Maintainers face multiple work-practice challenges in the same way that developers do and also seek flexible and efficient processes and appropriate methodological support. Much of the literature on software maintenance deals with traditional methods and software engineering techniques, but software maintenance departments are well aware of agile methods and have started experimenting with them. A few pioneering research studies suggest that agile practices have several advantages in software maintenance (Choudhari and Suman 2010; Rudzki et al. 2009).

Nevertheless, software maintenance differs from development in several important respects (Charette et al. 1997); for instance development does not normally depend on the comprehension of an existing program, as maintenance does. Agile methods were developed for software development (not for software maintenance). It is a reasonable assumption that the adoption of an agile method in the maintenance context may pose considerable challenges. The principles behind optimising an agile method for a maintenance operation are poorly researched and in need of clarification, and there is therefore a need to investigate how agile methods can successfully be adapted to maintenance situations. Researchers should understand which advantages and disadvantages the methods have for the maintenance operation, and how the well-known approaches should be customized. They need to develop theory in the area of agile maintenance, and guidance for maintenance practitioners wishing to adopt agile practices. In this article we therefore ask the question: how can agile methods be optimized to fit the practice of software maintenance?

The empirical part of the research concerns an action research study of the adoption of Scrum into the work practice of the maintenance department at Aveva, an international software company specialising in enterprise and design solutions for large engineering plants and the shipping industry. The focus of the study was on management of agile software maintenance. The maintenance group in Denmark concentrates on enterprise systems for shipping, with a global portfolio of customers. They comprise nine developers and two managers, and had been working with Scrum for about 9 months at the time the study started. A group of three researchers from Aalborg University assisted in the task of helping the new agile process to run smoothly.

2 Research Process

The research is framed as two complimentary studies (Fig. 1): a literature study and an action research project. The literature study identifies relevant maintenance literature and previous research into agile maintenance, and investigates three questions:

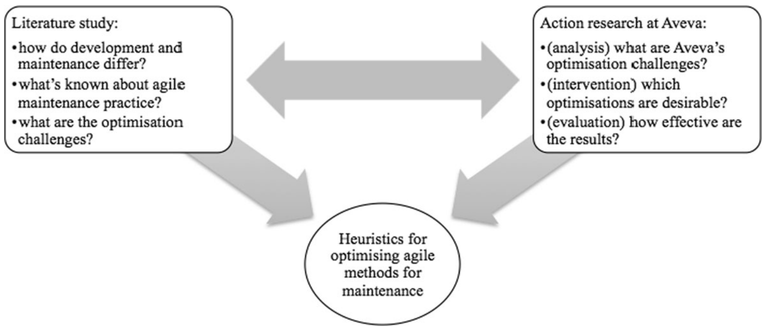
Literature study:
•how do development and maintenance differ?
•what's known about agile maintenance practice?
•what are the optimisation challenges?
Action research at Aveva:
•(analysis) what are Aveva's optimisation challenges?
•(intervention) which optimisations are desirable?
•(evaluation) how effective are the results?
Heuristics for optimising agile methods for maintenance

Fig. 1 Overall framing for the research

- how do software development and maintenance differ?
- what is known about agile maintenance practice?
- which challenges may be expected when optimizing an agile method for maintenance?

The action research study was a cooperation between the maintenance unit at Aveva and the information systems group at Aalborg University. Action research is an appropriate form of research for optimising organisational performance (Susman and Evered 1978) and accepted both in information systems research (Baskerville and Wood-Harper 1998) and software engineering research (Runeson and Höst 2009). The objective of the research was to optimize the Scrum implementation at Aveva. It consisted of three activities: initial analysis, intervention and evaluation; the empirical data were collected over a year (November 2011 to October 2012). Table 1 summarizes the action research design.

In action research it is important to maintain a research focus (McKay and Marshall 2001); we maintain an iterative connection between theoretical literature and action research study by encouraging the literature study to influence issues under consideration in action research project and vice versa. Nine issues (see Table 2) under consideration are investigated theoretically, from the literature, and empirically, in the case study; the resulting heuristics are grounded in these issues.

The rest of paper is structured as follows: in the next section the theoretical background identifying differences between software development and maintenance, the state of the art of agile software maintenance, and expected optimisation challenges are outlined. The learning is presented in a summary table organised around the nine numbered issues. In the next section, the case company is introduced, and the three stages of the action research (analysis,

Table 1 Action research design

Initial analysis	Intervention	Evaluation
9 qualitative interviews	Meetings with management	Evaluating questionnaires
Observations of stand-up and planning meetings	Seminar with the team, determining the concrete strategies	8 speed interviews
Seminar presentation	Observations of planning meetings	Seminar presentation and validation
Interview coded and analysed using Nvivo	Seminars and observations audio recorded	Interviews coded and analysed using Nvivo

intervention and evaluation) described. The major points of learning are again identified in a summary table based on the nine issues. Section 5 develops nine heuristics for maintenance departments adopting agile methods, which are derived from the theoretical and empirical analyses, and the conclusion develops implications for research and practice.

3 Challenges for Optimisation of Agile Methods for Maintenance

This section describes the theoretical background for the study. It presents an analysis of the differences between software development and maintenance, a comparison of agile software development and agile maintenance, and nine challenges for implementing agile methods in maintenance.

3.1 Software: Maintenance is not Development

The literature defines software maintenance as "modification of a software product after delivery" (Bennett and Rajlich 2000); hence as a stage that take place after software development. During maintenance, software systems move away from their original state (Stachour and Collier-Brown 2009) into continuous evolution (Sukumaran and Sreenivas 2005). Maintenance work can be understood as a lifecycle (as system development sometimes also is) with four stages (Kung and Hsu 1998):

- Introduction stage: the first few months after the system goes live, normally with low usage; identification and resolution of initial problems
- Growth stage: growth of system usage exposes deeper level technical problems (such as performance issues), bugs in less-used features and mismatches with work processes requiring changes
- Maturity stage: during this stage, major enhancement projects occur that test the limits of the technologies and functionality embedded in the application software
- Decline stage: the system reaches the limits of the embedded technologies, and users require software renovation – the system is patched until it is replaced.

However, this lifecycle can be considerably longer than the development life cycle, and maintenance work may cover several overlapping system cycles. It is therefore more often organised as a departmental unit than a time-limited project. A classical typology of maintenance types was provided by Lientz and Swanson (1980) – maintenance is:

- Adaptive – responding to changes in the software environment
- Perfective – responding to new user requirements
- Corrective - fixing errors
- Preventive – preventing foreseeable problems in the future

Maintenance is mis-represented as bug fixing (corrective maintenance) since approximately 75 % of it is adaptive or perfective - effectively new development. This may be a good argument for adopting a agile development method, even though some other premises may be different. Whereas development projects usually only deal with one instance of a system at a time; maintenance is often complicated by dealing with customers with different customizations of several versions of a software system (Bennett and Rajlich 2000). The result may be many small independent tasks with limited cohesion, rather that the naturally homogenous tasks in a development project. Tasks are usually initiated by customers, leading to a change mini-cycle (Bennett and Rajlich 2000):

- Request for change
- Planning phase: program comprehension, change impact analysis
- Change implementation: restructuring for change, change propagation
- Verification and validation
- Re-documentation

For minor changes, further customer involvement may not be necessary, and the large-scale requirements and analysis tasks of a development project are replaced by program comprehension - a demanding skill, requiring both understanding and interrelation of domain, situation and program models, and the generation and testing of hypotheses and high-level strategies (Vans et al. 1999). Up to 50 % of the effort in such a maintenance cycle can be devoted to program comprehension (Bennett and Rajlich 2000). However, maintenance requires a skill-set commensurate with development work (Taylor et al. 1998), encompassing not only technical skills in programming and de-bugging, but also analytical skills such as requirements analysis, business skills such as understanding work-process and social skills for user liaison.

Much research in software maintenance has been focused on traditional software engineering techniques such as modelling techniques and UML, (Arisholm et al. 2006; Dzidek et al. 2008), estimation, (Buchmann et al. 2011; De Lucia et al. 2003; Nguyen et al. 2011), risk management, (Charette et al. 1997; Sherer 1997), statistical process control (De Lucia et al. 2003; Popovic et al. 2001; Ware et al. 2007; Zanker and Gordea 2006), quality, (Ghods and Nelson 1998), metrics, (Hall and Lineham 1997; Popovic et al. 2001; Ware et al. 2007), post mortem, (De Sousa et al. 2004) and testing, (Sukumaran and Sreenivas 2005). The major thrust of this research -in line with standard textbooks such as Pigoski (1996) and Grubb & Takang (2003) - is directed towards increasing discipline (Boehm and Turner 2003) in the maintenance operation, rather than agility.

In summary, many common assumptions about development work are poorly transferable to the maintenance operation. Development is usually organised as a project, in a matrix-structured organisation, whereas a stable group (structured like a department) often carries out maintenance. Release planning and personnel responsibilities are not necessarily synchronized, which keeps maintenance organizations from attaining a sense of cohesion (Charette et al. 1997). Maintenance organizations do not often have a project completion date to rally around. There is only the next task, and the one after that, and the one after that, ad infinitum. If there are releases, they are a mixture of enhancements and corrections for different customers, which further reduce the sense of focused purpose. Individuals frequently work alone on tasks and often have sole responsibility for the task. They are not necessarily working as part of a team, in which individuals communicate with one another during their work. The tasks can be ad-hoc, and have no intrinsic relationship with each other. Maintenance planning is constantly interrupted by urgent requests from customers whose own businesses may be threatened by system failures or changes that must be accommodated. The traditional measures of success in development work revolve around delivering a unified product at specific points in time at an acceptable cost with sufficient quality to satisfy a single customer, however these need some adjustment in the case of the maintenance operation.

3.2 Agile Maintenance

It is generally accepted that agile methods share a group of common characteristics (Prochazka 2011; Abrahamsson et al. 2002), which include an iterative development process, focused work objectives around delivery points, small teams working closely together, close customer involvement, face-to-face communication, light documentation, frequent testing, intrinsic motivation through collective ownership, knowledge transfer through openness, and a focus on a high quality of code and product. These characteristics are also understood to function well, at least in some kinds of development situations.

A smaller body of research work investigates the application of these characteristics to maintenance work. Since we have argued that there are significant differences between development and maintenance engineering, we do not assume that the tools and techniques associated with agile development will necessarily produce good results in maintenance (Polo et al. 2002). The agile maintenance studies are generally positive in tone and highlight advantages: agile methods help speed up the process (Choudhari and Suman 2010) and improve communication between members of the maintenance team (Rudzki et al. 2009). Other cases demonstrate improved user satisfaction and team motivation (Prochazka 2011) and an increase in code quality (Choudhari and Suman 2010; Rudzki et al. 2009; Pino et al. 2012). An iterative lifecycle can be seen as advantageous for maintenance (Choudhari and Suman 2010), due to the short-term nature of the work (Shaw 2007). Whereas some maintenance engineers easily adopt pair programming, and find that it improves the quality of the code (Poole and Huisman 2001; Poole et al. 2001), others regard it as pointless and time consuming (Kajko-Mattsson and Nyfjord 2009) - these findings mirror the agile development literature. There are no documented examples of maintenance teams with on-site customers, but close cooperation with customers is regarded positively (Svensson and Host 2005). Scrum meetings have been successfully implemented in maintenance (Rudzki et al. 2009). Maintenance departments traditionally respect documentation (though they do not necessarily trust it) since they expect other engineers to work on their code. They are therefore naturally suspicious of agile attempts to reduce documentation to a minimum (Souza et al. 2005). Agile testing practices work well in maintenance situations; frequent testing is necessary, and suites of automated tests reduce the overhead of performing them (Poole and Huisman 2001; Poole et al. 2001). Regression testing (which agile methods encourage through iterative practices) is important to detect knock-on effects of maintenance improvements (Thomas 2006). Collective ownership proves a mixed blessing; many maintenance jobs are small, distinct and specialised and there is a natural tendency for the developer with the necessary specialised knowledge to take responsibility for them (Svensson and Host 2005). Openness leads to visibility of tasks and their allocation, which is also highly valued in maintenance (Poole and Huisman 2001; Poole et al. 2001). However, the principle of simple design can be difficult to apply, as teams are constrained by the design decisions of the original developers, and have to allow for future implementations and corrections (Svensson and Host 2005).

A certain amount of redesign both of agile practices and of maintenance work process is commonly necessary when agile methods are adopted by maintenance engineers, and all studies point to implementation difficulties. The few case studies available are largely practice reports; the principles behind optimising an agile method for a maintenance operation are not discussed.

3.3 Challenges for Optimising Agile Methods for Maintenance

We have discussed the principal differences between development and maintenance work, and looked at how agile characteristics operate in the few existing studies of agile maintenance. We summarise this literature-based investigation as a series of challenges for the optimisation of agile methods in the maintenance environment:

- in relation to the agile characteristic of iterative development: maintenance work can be organised into sprints (iterations), but the task synergy and common goal assumed in development work are usually missing in maintenance (Poole and Huisman 2001)

- in relation to focused work objective: maintenance sprints are subject to interruption by urgent customer demands and there are few common delivery points or integrated releases (Bennett and Rajlich 2000)
- in relation to teams working closely together: maintainers predominantly work on individual tasks on many different systems and system variants (Bennett and Rajlich 2000)
- in relation to close customer involvement: maintenance engineers typically work with many customers with many small change requests, they are seldom on-site, and there is often little close interaction (Bennett and Rajlich 2000)
- in relation to face-to-face communication: this is not always necessary or productive in maintenance, due to the diverse nature of the tasks undertaken (Kitchenham et al. 1999)
- in relation to light documentation: maintainers consider documentation necessary to maintain the integrity of the evolving system, and to help with program comprehension in future maintenance work (de Souza et al. 2006; Prochazka 2011)
- in relation to frequent testing: comprehensive system testing is usually impractical, testing limited to immediate impacts of current fix (Junio et al. 2011)
- in relation to motivation through collective ownership: collective ownership may be irrelevant where maintainers are working on independent tasks in multiple codebases, motivation is difficult without common tasks and delivery points (Kitchenham et al. 1999; Singer 1998)
- in relation to knowledge transfer through openness: openness and information sharing may be of little value if tasks are un-related, and there is no coherent release or natural deadline (Bennett and Rajlich 2000)

Table 2 summarizes the results of the literature investigation.

4 Optimising Scrum at Aveva's Maintenance Operation

4.1 The Aveva Operation

Aveva is a large worldwide software company specialising in engineering software for the plant and marine industries. They have technology centres in six European countries and the USA and India, and sales and support operations around the world.

The maintenance department at Aalborg works primarily with MARS (an enterprise management system for the ship-building industry) and consists of a small team of 8–9 persons with two integrated managers. These two managers and other team members with accumulated experience were in charge of keeping the contact to the customers. The team services a portfolio of customers around the world with heavily customized implementations. The team members in the maintenance department were mainly women (two men) with very different background and a great difference in years of experience. While about half of the team members had 10+ years of experience, the remainders were very new in software development and maintenance. The team primarily consisted of Danish citizens but also included a few other nationalities. Only a few of the team members had practical experience with agile methods.

4.2 The Pre-research-State

The maintenance operation at Aveva Aalborg was for many years based around a conventional task management system, with dedicated maintainers servicing their own customers. Though this approach generated a good rapport between individual maintainers and their customers and a high

Table 2 9 optimisation challenges for implementing agile practice in the maintenance area

Agile characteristics	Agile software development	Agile software maintenance	Agile maintenance optimisation challenges	Issue no
Iterative development	Agile methods advocate an iterative development strategy (Beck et al. 2001), developers work in fixed time frames (Schwaber and Beedle 2001)	Iterative lifecycle advantageous for maintenance (Choudhari and Suman 2010), due to short-term nature of work (Shaw 2007)	Maintenance work can be organised into sprints (iterations), but there is not necessarily any task synergy or common goal (Poole and Huisman 2001)	1
Focused work objective	The goal of the short iterations is to give the developers a time to work without interruptions (Schwaber and Beedle 2001)	No current research raises the issue of how to keep the objective of focused work even though maintenance has several urgent customer demands	Sprints are subject to interruption by urgent customer demands and there are few common delivery points or integrated releases (Bennett and Rajlich 2000)	2
Teams working closely together	Agile team are small in order to be able to work together closely. XP advocates that the team sit together and use pair programming (Conboy and Fitzgerald 2010)	Studies undecided: pair programming advantageous and improves code quality (Poole and Huisman 2001; Poole et al. 2001), or too expensive to pursue (Kajko-Mattsson and Nyfjord 2009)	Maintainers may work on many individual discrete tasks on different system variants (Bennett and Rajlich 2000)	3
Close customer involvement	The customer plays a central role in agile development. XP recommends an on-site customer (Beck et al. 2001), Scrum has the role of the product owner (Schwaber and Beedle 2001)	Close cooperation with customers desirable, but no examples of on-site customers (Svensson and Host 2005)	Many customers, many small change requests, seldom on-site, little close interaction (Bennett and Rajlich 2000)	4
Face-to-face communication	Face-to-face communication is seen as essential. This is largely supported by frequent meetings (Schwaber and Beedle 2001)	Scrum meetings have been implemented with success in maintenance (Rudzki et al. 2009)	Face-to-face communication not always necessary or productive due to diverse nature of work (Kitchenham et al. 1999)	5
Light documentation	Interactions are seen as more important than documentation (Conboy and Fitzgerald 2010)	Documentation traditionally important in maintenance, agile methods deficient (Souza et al. 2005)	Documentation considered necessary to maintain integrity of evolving system (de Souza et al. 2006; Prochazka 2011)	6
Frequent testing	Test-driven development advocated (Beck), a working increment of the system developed and tested in each sprint (Schwaber and Beedle 2001)	Frequent automated tests regarded as important in agile maintenance (Poole and Huisman 2001; Poole et al. 2001); especially regression testing (Thomas 2006)	Comprehensive system testing usually impractical, testing limited to immediate impacts of current fix (Junio et al. 2011)	7
				8

Table 2 (continued)

Agile characteristics	Agile software development	Agile software maintenance	Agile maintenance optimisation challenges	Issue no
Motivation through collective ownership	Working in self-organizing teams with common responsibility for tasks (Schwaber and Beedle 2001) and code (Beck et al. 2001) enhances team motivation	Despite collective ownership, most of the time the person with specialist knowledge of the system will be responsible (Svensson and Host 2005)	Collective ownership irrelevant where maintainers are working on independent tasks in multiple codebases, motivation difficult without common delivery points (Kitchenham et al. 1999; Singer 1998)	
Knowledge transfer through openness	Information on the project and its status must be provided in the open space (Conboy and Fitzgerald 2010)	Openness leads to visibility of tasks, highly valued in maintenance (Poole and Huisman 2001; Poole et al. 2001)	Openness and information sharing may be of little value if tasks are un-related, and there is no coherent release or natural deadline (Bennett and Rajlich 2000)	9

level of specialised knowledge of these customers' implementations, it also led to mismatching of experience and tasks (a very experienced maintainer with a trivial task), work troughs and peaks for individual maintainers which could sometimes induce long working hours and stressful deadlines, and vulnerability in the case of absence through, for example, sickness.

They switched to Scrum in 2010, implementing a method with following Scrum elements:

- The Scrum master role is assigned to one of the maintainers
- The Product owner role is assigned to several internal persons, due to multiple customer relations
- Highly self-organized team
- Sprints of 2 weeks length
- Planning meetings before each sprint
- Daily 5-minute scrum meetings
- Tasks collected into a prioritized product backlog
- A subset of tasks is chosen for the sprint backlog
- A large task board is used to keep track of the sprint progress

Even though they had succeed in implementing the majority of the practices of Scrum, the method immediately demanded modifications – there was no natural overall product owner or on-site customer, and release dates were chosen arbitrarily because they worked with batches of (often unrelated) jobs made unpredictable by emergency calls from customers that had to be attended to immediately. They initially decided to use fourteen-day sprints for fast turnover, and adopted planning poker for estimation (practices more usually associated with XP). The scrum master role was assigned to one of the more experienced maintainers; her task was to lead the team through planning meetings, daily stand-up meetings and retrospectives (every second month). The new Scrum approach introduced shared ownership; everybody was involved in estimation, had a say in prioritising tasks, selected their tasks from the common scrum board (instead of responding to their own customers) and took responsibility for completing their own tasks on time. Even though the maintenance team at Aveva overall was pleased with moving to Scrum, they found that an adaptation and optimisation of the method was necessary.

5 Optimising Agile Maintenance Practice at Aveva

5.1 Analysis

The study was initiated with an analysis based on semi-structured qualitative interviews (Myers and Newman 2007) with developers and managers of the software maintenance team (in total 9 interviews). A stand-up meeting and a planning meeting were observed. The interviews and the observations were audio-recorded; the audio files were later coded and analysed using NVivo 9 and the relevant passages were transcribed. The analysis focused on documenting current practice with Scrum and identifying strengths and weaknesses (optimisation needs). The whole team attended a seminar to validate the analysis results and determine focus points for the intervention. The initial analysis identified six inter-connected issues with the group's use of Scrum. Table 3 gives these and their interview coding frequency. The coding frequency is not included for generalization of the findings, but as an indicator of which issues were focused on by the maintenance team.

Table 3 Analysis findings

Findings	Frequency	Explanation
Looser relationships with customers compared with the earlier work-practice	5	Close relationships between individual maintainers and customers in the earlier work-practice provided both good feedback and a form of relational motivation derived from satisfying customer needs. These became less evident when Scrum was introduced.
Sprint goal not consistently reached	70	A mixture of emergency jobs (which could not be planned for) and difficulties with estimation (partly caused by incomplete use cases) meant that the sprint plan was not consistently met, and unfinished tasks were held over to the next sprint.
Poor motivation and lack of feedback for maintainers compared with the earlier work-practice	15	Not meeting the sprint goal was de-motivating, especially where earlier types of feedback and motivation were less evident than before.
Administration of Scrum is time consuming, particularly the meetings	29	Meetings were long inefficient (particularly planning meetings) consuming resources that could not be billed to customers. Planning poker was only partially effective; with some incomplete use cases, and maintainers who did not necessarily know anything about the task, customer and installation being estimated. Many discussions involved only a minority of the more experienced maintainers - leaving the others unable to contribute.
Lack of shared knowledge of individual customer installations	11	Maintainers were often expected to take the next task from the Scrum board which were prioritised in order, resulting in them working on an installation or a task with which they were not familiar and needed support.
Unhelpful use cases with low communication value for fellow maintainers	13	Use cases had traditionally been written in a shorthand form that signalled their content for the writer/implementer (often the result of a telephone call or email with a customer). This didn't provide a secure foundation for group estimation and programming by a different maintainer. Maintainers provided examples of both poor practice and good.

5.2 Intervention

At a seminar presentation of the findings presented above, the group identified the administration issue and the use case problem as important starting points for the intervention, though many of the problems can be seen as inter-related. The intervention covered optimisations in four areas:

- Use cases. A use case template was developed from the examples of good practice identified earlier, and implemented in the task-handling system as a non-compulsory structure. In addition the team adopted the practice of rejecting poorly formulated use-cases at the planning meeting and asking for them to be re-written. It was expected both that estimation would improve, and that time spent clarifying tasks would be reduced. Both should contribute to reaching sprint goals.

- Estimation. Practice was changed so that maintainers with special responsibility for customers estimated their own customers' use cases before planning meetings. Group estimation (planning poker) was not eliminated entirely, but confined to estimates that were found questionable by the team. This change was designed both to improve estimation quality and reduce the time spent on planning poker in planning meetings.
- Planning meetings. The planning meetings were re-organised so that a backlog printout was distributed in advance of the meeting and the team asked to prepare by reading through it. They could then spot poor use cases and estimates they did not agree with. The scrum master was also given powers to curtail lengthy or irrelevant discussion. The intention was to create more efficient, better-focused and shorter meetings.
- Metrics. A lightweight portfolio of metrics was introduced. The team identified the parameters that defined their successful working environment as: employee satisfaction (interesting tasks, team spirit, appropriate pressure), cooperation (meeting results, involvement in decisions, quality (errors found at test, customer-reported errors) and estimation accuracy. Metrics were developed combining data from the task-management system and the questionnaire, and the process was automated so that the metrics could be graphically represented for each planning meeting with minimal effort. Metrics served both to help visualize progress in the other intervention areas and to provide an alternative form of motivation for the team.

The intervention strategy was organised in collaboration with the two managers and initiated at a seminar where the whole group was invited to refine the concrete improvement suggestions. During the intervention three planning meetings were observed and the team was followed closely.

5.3 Evaluation

The progress of the intervention was evaluated through questionnaires every second week and eight qualitative speed-interviews (one developer had resigned since the initial analysis). The interviews focused on the actions taken during the intervention, while the questionnaires monitored the overall levels of satisfaction on agreed parameters. Audio recordings of the evaluation interviews were coded and analysed using Nvivo 9; the results were presented and validated at a seminar with the whole team.

The evaluation showed that the quality of the use cases had improved. The template developed for the use cases was considered really useful but there was also room for improvement. Estimation accuracy remained somewhat variable, thought the team managed two sprints that were perfectly estimated. The evaluation showed how changing the ground rules had resulted in more efficient planning meetings. The team thought the meetings now provided the same benefits/information in a shorter time and continued with the new practice. They acknowledged, however, that preparation time was longer, and that much of this fell on the scrum master. The metrics were adopted as standard practice, providing a useful overview of the progress of the group and a starting point for retrospectives.

Table 4 summarizes learning from the action research project.

Table 4 Summary of learning from the action research project

Issue no.	Agile maintenance optimisation challenges	Learning from the action research	Optimisations to Scrum
1	Maintenance work can be organised into sprints (iterations), but there is not necessarily any task synergy or common goal (Poole and Huisman 2001)	Sprints at Aveva were made up of collections of (usually unrelated) tasks chosen on the basis of urgency and keeping the portfolio of customers happy. Achieving the sprint goals (and monitoring progress through a burn-down chart) were therefore somewhat arbitrary objectives, motivated more by professional pride than the natural structure of the work.	Engineers were motivated by comparing sprints to establish a curve of improving performance. A minimal suite of low cost automated metrics was developed to support this common objective.
2	Sprints are subject to interruption by urgent customer demands and there are few common delivery points or integrated releases (Bennett and Rajlich 2000)	The maintenance department at Aveva handled some larger tasks, but seldom involving more than on customer implementation; new product releases were developed elsewhere at the company. Sprints were almost always interrupted by urgent customer requests, which made both estimation and achieving the sprint goals difficult.	Some time for unexpected tasks was factored into the sprint planning, though the scale of the tasks remained unpredictable, especially during short sprints.
3	Maintainers may work on individual tasks on different system variants (Bennett and Rajlich 2000)	Most of the tasks at Aveva were of this discrete nature – the challenge therefore became to share knowledge so that engineers were equipped to work on many different tasks in different environments.	Pair programming was not considered appropriate, but the team developed an informal mentoring system so that engineers with experience of a particular problem or system variant could support less experienced colleagues.
4	Many customers, many small change requests, seldom on-site, little close interaction (Bennett and Rajlich 2000)	Most change requests were submitted by email. Individual engineers knew some customers well through years of interaction, and senior members of the team visited important customers. The agile objective of an on-site customer was impractical, and the move to collective ownership of the work meant breaking to some extent with the old customer representative structure.	The team adopted a multiple product owner strategy – giving those members with existing relationships with customers special responsibilities for estimation, mentoring and handling tricky problems
5	Face-to-face communication not always necessary or productive due to diverse nature of work (Kitchenham et al. 1999)	Some aspects of the planning meetings were time-consuming or ineffective because of a lack of common experience. For instance, planning poker for estimation yielded poor results in cases where very	The team reverted to trusting the estimates of the engineers with relevant experience in many cases, retaining the right to challenge estimates where they had a reason to disagree. The scrum master intensified preparation of the meetings – more time

Table 4 (continued)

Issue no.	Agile maintenance optimisation challenges	Learning from the action research	Optimisations to Scrum
		few engineers had knowledge of the task, or experi ence with the customer's system variant.	consuming for her, but less burdensome for the team as a whole.
6	Documentation considered necessary to maintain integrity of evolving system (de Souza et al. 2006; Prochazka 2011)	Paradoxically, use cases required better documentation with the move to common ownership. This was because tasks were allotted by sprint priority in the sprint and use cases therefore needed to communicate to an engineer who might have little prior knowledge of customer or system variant.	Structured use cases introduced, other documentation practices maintained.
7	Comprehensive system testing usually impractical, testing limited to immediate impacts of current fix (Junio et al. 2011)	Agile practices fitted well with the engineer's practice of testing each fix before implementation, but fully automated testing of many system variants was impractical.	The move to collective ownership meant that engineers could test each other's fixes instead of testing their own; this was regarded as a safer practice that encouraged shared learning.
8	Collective ownership irrelevant where maintainers are working on independent tasks in multiple codebases, motivation difficult without common delivery points (Kitchenham et al. 1999; Singer 1998)	The engineers did not find collective ownership irrelevant; they valued the knowledge transfer involved, the ability to work on a broader variety of tasks and systems, the opportunity to improve their professional skills, and the more effective distribution of work which led to fewer bottlenecks. These were all good motivators, as was the sense of departmental professionalism.	The principle was accepted in principle – many small modifications supporting shared learning and good communication needed to make it work in practice
9	Openness and information sharing may be of little value if tasks are un-related, and there is no coherent release or natural deadline (Bennett and Rajlich 2000)	Openness and information sharing were valued – they gave a better overview of distribution of tasks and monitoring of progress, with learning and mentoring opportunities.	The traditional Scrum boards were implemented with few minor modifications, but modifications were necessary for planning meetings, particularly in respect to estimation.

6 Nine Heuristics for Maintainers Adopting Agile Methods

Though Scrum is targeted at software development and not software maintenance, several of its practices are advantageous for maintenance projects as well. Aveva experienced benefits including 1) improvement of code quality, 2) improvement of team morale, 3) an increased visibility of the project and 4) better communication and knowledge sharing. These results align with previous reports in the literature examined in section 3. Adoption is, however, not trivial. As with software development adoption, methods need to be tailored to fit the situation (Boehm 2002; Nerur et al. 2005). Our theoretical analysis suggests that there are also differences between maintenance and development work that would indicate that modifications might be necessary. At Aveva several optimizations were necessary in order to improve the fit of Scrum, and their optimisation work continues. In this respect our study reinforces the work of Svensson and Host (2005), and we recommend caution with other studies that uncritically report benefits of agile implementation in maintenance.

This section combines lessons from both theoretical and empirical investigations and presents nine heuristics for maintainers adopting agile methods, derived from the issues presented in Tables 2 and 4.

Heuristic 1 (derived from Tables 2 and 4, issue 1) *Use sprints to organize the maintenance work; balance the needs of the portfolio of customers*

Previous research shows how iterations can be used in a maintenance context, both 2 week iterations (Poole and Huisman 2001) and 4 week iterations (Nawrocki et al. 2002). Maintenance tasks often have a short-term nature and shorter iterations are therefore preferable (Shaw 2007). Short iterations were also considered advantageous for at Aveva. However the issue of lack of task synergy and the absence of a common goal have not previously been considered. This issue arises because of the number of different customers with outstanding maintenance requests at any time. At Aveva, they adopted two principles to structure their sprints. The first was that, as far as possible, all customers with current requests should be served in the sprint, even if all of their tasks could not be completed. This avoids customer dissatisfaction through lack of attention. The second was that customers with urgent (business threatening) request should be prioritised. Hanssen et al. (2009) found that software quality was sometimes compromised by the desire to complete the sprint tasks on time. This was not the case at Aveva - their code quality improved when working in sprints. This may be related to better testing practices (see heuristic 7).

Heuristic 2 (derived from Tables 2 and 4, issue 2) *Allow for unexpected urgent customer requests*

Agile methods advocate focused development iterations where the team works without interruptions (Schwaber and Beedle 2001), although it's not necessarily easy or desirable to avoid them (Heeager and Nielsen 2009; Pikkarainen et al. 2008). In maintenance work, urgent customer requests may occur at any time (Bennett and Rajlich 2000). There are consequently two types of tasks: those that can be planned (releases, new version, minor bug fixes) and those that cannot (urgent, acute fixes). Sprint interruptions and mid-cycle reprioritization are therefore inevitable (Poole and Huisman 2001). The literature on agile maintenance does not provide much guidance on how to handle urgent customer demands. Pino et al. (2012) suggest having two different types of maintenance sprints - a short one for unexpected demands and a longer one for tasks that can be planned. Aveva found that they had to attend to urgent customer requests even during very short sprints. They introduced a buffer (Schwaber and Beedle 2001) - some sprint time which was not allocated in advance - however the size of this buffer was hard to predict. Sprint disruptions need to be managed in order to avoid frustration and demotivation in relation to planning and sprint goals.

Heuristic 3 (derived from Tables 2 and 4, issue 3) *Structure team learning where team members work on discrete tasks*

Agile methods advocate shared ownership of the code (Cockburn 2006), whereas maintainers tend to become specialists in a few customer systems (Singer 1998). Shared ownership of tasks independent of customers helps optimise the workload across the team, and increases team morale. At Aveva, shared ownership of tasks was considered one of the strengths of Scrum - the workload was no longer distributed unevenly and the maintainers were able to help each other out in busy periods. The maintainers went from being specialists in a few customer systems to becoming generalists working with several customer systems. This change required a strong focus on team learning and knowledge transfer – achieved in agile methods by close cooperation. These benefits are reported both in the agile development literature (Dybå and Dingsøyr 2008) and the more limited agile maintenance literature (Rudzki et al. 2009). The maintainers at Aveva also experienced improvement in internal knowledge sharing with the adoption of Scrum. Daily stand-up meetings served as a place for identifying knowledge in the team and as a basis for team learning. Pair programming didn't really work; instead the team adopted an internal mentoring practice. Those engineers working on an unfamiliar system identified someone who had experience with it and asked for help, often starting with a detailed explanation of the task as specified in the use case.

Heuristic 4 (derived from Tables 2 and 4, issue 4) *Structure multiple customer relationships*

Maintenance is characterized by multiple customer relationships using multiple systems or system variants (Bennett and Rajlich 2000). Scrum advocates a product owner role taken by a single customer representative (Schwaber and Beedle 2001); XP advocates having an onsite customer (Beck and Andres 2004). Implementing this role in development work is not simple (Paulk 2002), as few customer organizations can make so much time available (Heeager and Nielsen 2013). In maintenance work a different logic must clearly operate; however the existing literature provides no recommendations. At Aveva several engineers had established relationships with customers they had previously been responsible for. These engineers were assigned the role of product owners; they were responsible for receiving the tasks, writing initial use cases (using their specialist customer knowledge) where necessary, and being advocates for their respective customers in the planning process. Multiple product owners implies a co-operative process for prioritizing backlogs, otherwise some customers' requests may win at the expense of other customers. Aveva adopted the principle that every customer should experience some progress in a sprint; after that the relative urgency of the request as expressed by the individual product owner should determine priority.

Heuristic 5 (derived from Tables 2 and 4, issue 5) *Balance documentation and face-to-face communication appropriately*

Traditional software development methods (Paetsch et al. 2003) and traditional maintenance approaches (de Souza et al. 2006; Prochazka 2011) are more dependent on documentation than agile methods, which to a larger degree depend on face-to-face communication (Nerur et al. 2005). Maintainers constantly work with changing systems and program comprehension, so they understand the need for good documentation. Agile proponents advocate keeping the documentation to a minimum, in order to avoid unnecessary overhead in writing and maintaining specifications (Beck et al. 2001). The maintenance team at Aveva sometimes found face-to-face communication unproductive, especially where they could not contribute because they had no detailed knowledge of customer, system variant, or task (see heuristic 9). Their need for documentation increased because of shared ownership: they had to work on a variety of tasks with which they might not be familiar, even when colleagues with specific domain knowledge were not available to help (see heuristic 6).

Heuristic 6 (derived from Tables 2 and 4, issue 6) *Write (minimally) structured use cases for communication*

Task specification was a case in point where Aveva needed good documentation; the minimal user stories of agile practice tuned out to be inadequate for their needs. Use case diagrams and specifications are regarded important in maintenance (de Souza et al. 2005, 2006), for representing requirements and developing tests (Choudhari and Suman 2014; Kajko-Mattsson and Nyfjord 2009). At Aveva the writer of the use case was (in many cases) no longer the engineer undertaking the task. The use case had to specify the task in such a way that any of the team's engineers could understand it, regardless of their level of experience with the customer or system. The more experienced maintainers travelled frequently to visit customer sites, and were not always around to explain. Nevertheless, with many small tasks, and customers unwilling to pay for documentation considered unnecessary, use cases had to remain lean and efficient - an agile principle. The solution was to write a short structured use case, where the structure ensured that all the necessary information was included. The task receiver could complete some trivial tasks without a use case, whereas especially complex tasks sometimes required more analysis and design work.

Heuristic 7 (derived from Tables 2 and 4, issue 7) *Strengthen collective ownership and learning by testing each other's code*

Agile methods advocate automated testing, early and often (Beck and Andres 2004). In maintenance work, testing the system adequately is a difficult challenge (Kajko-Mattsson 2008). Comprehensive system testing is usually impractical and testing is often limited to the immediate impacts of the current fix (Junio et al. 2011). Studies show that testing remains a challenge in agile maintenance environments (Poole and Huisman 2001). Test-driven development is not easy to implement (Kajko-Mattsson 2008), and when implemented not observed in practice (Hanssen et al. 2009), with negative impacts on software quality. Good documentation is necessary for comprehensive testing, but not always present (de Souza et al. 2005; Kajko-Mattsson and Nyfjord 2009; Nawrocki et al. 2002). Nevertheless test-driven development can sometimes improve maintenance code quality (Choudhari and Suman 2014). Aveva found it impractical to have fully automated tests of the many system variants. They continued with their practice of testing every fix before releasing it, but nevertheless found a way to improve quality through collective ownership. Engineers switched to testing each other's fixes instead of their own; this is a more effective practice and encouraged shared learning.

Heuristic 8 (derived from Tables 2 and 4, issue 8) *Find team motivators to replace the natural motivation of reaching the end of a sprint or project*

Team motivation is a general problem in software maintenance (Kitchenham et al. 1999). Agile developers strive for motivation and morale in the team through close cooperation (Cockburn and Highsmith 2002), and committing to tasks and sprint deadlines (Schwaber and Beedle 2001). Pair programming also strengthens team motivation (Poole and Huisman 2001) - though this presupposes a certain amount of task synergy and shared experience, which is sometimes missing in maintenance work. Agility in software maintenance has been shown to enhance team morale (Prochazka 2011), and Aveva experienced an improvement in team morale immediately after introducing Scrum. However, maintenance work lacks the natural project form of development work, where tasks have synergies, the work has combined team delivery points and the project has an end point. Maintenance work is never 'done,' nor does reaching the end of a sprint bring the overall project goal closer – there is just another batch of maintenance jobs in the next sprint. Aveva's experience was that the planning difficulties outlined earlier made it hard to reach the sprint goal, which undermined morale instead of improving it. They tried, and abandoned pair programming; where the tasks are discrete it's more effective to work alone.

Several factors helped with motivation. A combination of planning experience and the introduction of sprint buffers, together with better estimation, improved sprint performance. Visibility of the team's effort improved through an informative workspace and frequent communication (in line with the findings of Poole and Huisman (2001)) which also increased morale. The introduction of a light set of metrics allowed for motivation across sprints, providing a mechanism for comparing sprints on simple parameters, and for monitoring the evolution of the department's progress. Motivations concerned with finishing a project, implicit in the agile approach for development, were here replaced with motivations concerned with professionalism in the operation of a department.

Heuristic 9 (derived from Tables 2 and 4, issue 9) *Keep the planning meetings short and effective*

Agile methods have a strong focus on face-to-face communication and information sharing facilitated through frequent meetings (Beck et al. 2001). Rudzki et al. (2009) conclude that this causes no additional cost in Scrum projects, however inexperienced teams spend more time planning (Beck and Andres 2004) and it is necessary to learn how to keep meetings short and efficient (Heeager 2012; Moe et al. 2010). Several characteristics of maintenance work can result in long and tedious planning meetings: multiple systems and customers result in many smaller tasks, and shorter sprints imply more meetings. Engineers may have little knowledge of some of the customers and systems being discussed, and therefore be unable to contribute much of value. The team at Aveva worked hard to improve the quality of their meetings. Long rounds of planning poker were discontinued in favour of pre-estimates from experienced engineers, which the meeting could accept or question. A printed backlog was produced before the meeting to give the maintainers a chance to prepare. The scrum master was authorised to be more direct in controlling the progress of the meeting. The result was an improvement in meeting quality, at the cost of some more preparation time for the scrum master.

7 Conclusions

Agile methods have become accepted within software development and are also beginning to spread to software maintenance teams. However there is relatively little research on agile maintenance; we asked the research question: how can agile methods be optimized to fit the practice of software maintenance? We investigate the question both from a theoretical perspective, and from an empirical perspective. From a theoretical perspective we argue that agile methods (when applied to development work) have certain relatively well-understood characteristics, however maintenance work differs from development work in certain key respects, and therefore it's reasonable to expect a number of challenges when implementing agile methods in a maintenance department. Table 2 outlines those challenges. From the empirical perspective we can find many of those challenges echoed in our optimisation work at Aveva, and outline some solutions that the maintenance team developed to improve their agile work practice. Our findings confirm that agile practices can have various benefits for maintenance departments. Aveva experienced improvement of code quality, improvement in team morale, an increased visibility of progress, and better communication and knowledge sharing amongst the team members. However those benefits may not occur automatically and some optimisation will normally be necessary to accommodate both the different character of maintenance work and local practice differences. We use both theoretical and empirical lessons to formulate nine heuristics for the optimisation process:

1. Use sprints to organize the maintenance work; balance the needs of the portfolio of customers
2. Allow for unexpected urgent customer requests
3. Structure team learning where team members work on discrete tasks
4. Structure multiple customer relationships
5. Balance documentation and face-to-face communication appropriately
6. Write (minimally) structured use cases for communication
7. Strengthen collective ownership and learning by testing each other's code
8. Find team motivators to replace the natural motivation of reaching the end of a sprint or project
9. Keep the planning meetings short and effective

This is an action research study with the limitations that this implies; for example limited control over process and outcomes. The research is inevitably somewhat driven by the problem-based needs of the practitioners, rather than a pre-formulated research agenda. Action research suffers from difficulties establishing rigour and objectivity according to conventional positivist natural science traditions. It is often preoccupied with organizational problem solving at the expense of transferable theoretical understandings. Experimental controls are not available and the exploration of relevant hypotheses through statistical methods normally precluded by the demands of the situation. In addition, the agile maintenance area is poorly developed in terms of theoretical sophistication and there are few well-established causal theories or testable hypotheses available on which rigorous controls could be established. Neither is action research a suitable vehicle for developing causal theory or testable propositions; there is lack of epistemological clarity in theory testing and development (Rose 1997). We offer a single case study, which does not preclude generalisation, but implies that other researchers should be sensitive to local variations and cultural differences when incorporating our conclusions into their own work.

Despite these limitations, action research does have one major advantage over other forms of research: relevance for practice is ensured through its empirical foundation and the purposeful interactions with practitioners. We consequently choose to present our findings as constructive heuristics, rather than theoretical propositions. Though the heuristics do not amount to a prescription for adapting an agile method to a maintenance department, practitioners may find this work useful as sensitization for the kinds of issues that may emerge with agile implementations, as early warnings for the kinds of optimisations that may be necessary, and as suggestions for practical improvements to their practices. Researchers may be more interested in its methodological implications, especially in the area of normative method theory for maintenance work. This is presently dominated by documentation- and technique-heavy prescriptions from conventional software engineering schools, and the present work can contribute to the emergence of targeted agile methods for maintenance by pointing at theoretical differences between development and maintenance work that need to be accommodated, and empirical problems with agile methods and their possible solutions.

Acknowledgments The authors would like to thank our dedicated collaborators at Aveva.
The project was partially funded by the European Union through the CAIN project.

References

Abrahamsson P, Salo O, Ronkainen J, Warsta J (2002) Agile software development methods: review and analysis. VTT, Finland

Abrahamsson P, Conboy K, Wang X (2009) 'Lots done, more to do': the current state of agile systems development research. Eur J Inf Syst 18(4):281–284

Arisholm E, Briand LC, Hove SE, Labiche Y (2006) The impact of UML documentation on software maintenance: an experimental evaluation. IEEE Trans Softw Eng 32(6):365–381

Baskerville R, Wood-Harper AT (1998) Diversity in information systems action research methods. Eur J Inf Syst 7:90–107

Beck K, Andres C (2004) Extreme programming explained: embrace change. Addison-Wesley Professional, USA

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 RC, Mellor S, Schwaber K, Sutherland J, Thomas D (2001) Manifesto for Agile Software Development. Accessed Web Page

Bennett KH, Rajlich VT (2000) Software maintenance and evolution: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ACM, pp 73–87

Boehm B (2002) Get ready for agile methods, with care. Computer 35(1):64–69

Boehm B, Turner R (2003) Observations on balancing discipline and agility. In: Proceedings of the Agile Development Conference, Salt Lake City, Utah, USA. IEEE Computer Society, pp 32–39

Buchmann I, Frischbier S, Putz D (2011) Towards an estimation model for software maintenance costs. In: 15th European Conference on Software Maintenance and Reengineering (CSMR) Oldenburg. IEEE Computer Society, pp 313–316

Charette RN, Adams KM, White MB (1997) Managing risk in software maintenance. IEEE Softw 14(3):43–50

Choudhari J, Suman U (2010) Iterative Maintenance Life Cycle Using eXtreme Programming. In: International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2010). IEEE, pp 401–403

Choudhari J, Suman U (2014) Extended iterative maintenance life cycle using eXtreme programming. ACM SIGSOFT Softw Eng Notes 39(1):1–12

Cockburn A (2006) Agile software development: the cooperative game. Addison-Wesley Professional, Boston

Cockburn A, Highsmith J (2002) Agile software development, the people factor. Computer 34(11):131–133

Conboy K (2009) Agility from first principles: reconstructing the concept of agility in information systems development. Inf Syst Res 20(3):329–354

Conboy K, Fitzgerald B (2010) Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion, vol 20. ACM Transactions on Software Engineering Methodology, vol 1. doi: 10.1145/1767751.1767753

De Lucia A, Pompella E, Stefanucci S (2003) Assessing the maintenance processes of a software organization: an empirical analysis of a large industrial project. J Syst Softw 65(2):87–103

De Sousa KD, Anquetil N, De Oliveira KM (2004) Learning software maintenance organizations. In: Advances in Learning Software Organizations. Springer-Verlag Berlin Heidelberg, pp 67–77

de Souza SCB, Anquetil N, de Oliveira KM (2005) A study of the documentation essential to software maintenance. In: Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information. ACM, pp 68–75

de Souza SC, Anquetil N, de Oliveira KM (2006) Which documentation for software maintenance? J Braz Comput Soc 12(3):31–44

Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: a systematic review. Inf Softw Technol 50(9–10):833–859

Dzidek WJ, Arisholm E, Briand LC (2008) A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Trans Softw Eng 34(3):407–432

Ghods M, Nelson KM (1998) Contributors to quality during software maintenance. Decis Support Syst 23(4):361–369

Grubb P, Takang AA (2003) Software maintenance: concepts and practice. Thomson Computer Press, London

Hall R, Lineham S (1997) Using metrics to improve software maintenance. BT Technol J 15(3):123–129. doi:10.1023/a:1018694404616

Hanssen GK, Yamashita AF, Conradi R, Moonen L (2009) Maintenance and agile development: Challenges, opportunities and future directions. In: International Conference on Software Maintenance, ICSM 2009., Edmonton, Alberta, Canada. IEEE, pp 487–490

Heeager LT (2012) Introducing agile practices in a documentation-driven software development practice: a case study. J Inf Technol Case Appl Res 14(1):3–24

Heeager LT, Nielsen PA (2009) Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study. In: Scheepers H DM (ed) Australasian Conference on Information Systems, Melbourne, Australien, 2009. p 205

Heeager LT, Nielsen PA (2013) Agile software development and the barriers to transfer of knowledge: an interpretive case study. In: Aanestad M, Bratteteig T (eds) Scandinavian conference on information systems. Springer, Oslo, pp 18–39

Junio GA, Malta MN, de Almeida Mossri H, Marques-Neto HT, Valente MT (2011) On the benefits of planning and grouping software maintenance requests. In: 15th European Conference on Software Maintenance and Reengineering (CSMR), Oldenburg. IEEE, pp 55–64

Kajko-Mattsson M (2008) Problems in agile trenches. In: Proceedings of the Second ACM-IEEE international symposium on Empirical Software Engineering and Measurement, Kaiserslautern, Germany. ACM, pp 111–119

Kajko-Mattsson M, Nyfjord J (2009) A model of agile evolution and maintenance process. In: 42nd Hawaii International Conference on System Sciences Hawaii. IEEE, pp 1–10

Kitchenham BA, Travassos GH, von Mayrhauser A, Niessink F, Schneidewind NF, Singer J, Takada S, Vehvilainen R, Yang H (1999) Towards an ontology of software maintenance. J Softw Maint 11(6):365–389

Kung H, Hsu C (1998) Software maintenance life cycle model. International Conference on Software Maintenance. IEEE Computer Society, Bethesda

Lehman MM (1980) on understanding laws, evolution, and conservation in the large-program life cycle. J Syst Softw 1:213–221

Lientz BP, Swanson EB (1980) Software maintenance management. Addison Wesley, Reading MA

McKay J, Marshall P (2001) The dual imperatives of action research. Inf Technol People 14(1):46–59

Misra SC, Kumar V, Kumar U (2010) Identifying some critical changes required in adopting agile practices in traditional software development projects. Int J Qual Reliab Manag 27(4):451–474

Moe NB, Dingsøyr T, Dybå T (2010) A teamwork model for understanding an agile team: a case study of a Scrum project. Inf Softw Technol 52(5):480–491

Myers MD, Newman M (2007) The qualitative interview in IS research: examining the craft. Inf Organ 17(1):2–26

Nawrocki JR, Walter B, Wojciechowski A (2002) Comparison of CMM level 2 and eXtreme programming. In: 7th European Conference on Software Quality Software Quality, Hensinki, Finland. Springer, pp 288–297

Nerur S, Mahapatra RK, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Commun ACM 48(5):73–78

Nguyen V, Boehm B, Danphitsanuphan P (2011) A controlled experiment in assessing and estimating software maintenance tasks. Inf Softw Technol 53(6):682–691

Paetsch F, Eberlein A, Maurer F (2003) Requirements engineering and agile software development. In: Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Linz, Austria. Citeseer, p 308

Paulk MC (2002) Agile methodologies and process discipline. Crosstalk-J Def Softw Eng 1(1):15–18

Pigoski TM (1996) Practical software maintenance: best practices for managing your software investment. Wiley, New York

Pikkarainen M, Haikara J, Salo O, Abrahamsson P, Still J (2008) The impact of agile practices on communication in software development. Empir Softw Eng 13(3):303–337

Pino FJ, Ruiz F, Garcia F, Piattini M (2012) A software maintenance methodology for small organizations: Agile_MANTEMA. J Softw Evol and Process 24(8):851–876

Polo M, Piattini M, Ruiz F (2002) Using a qualitative research method for building a software maintenance methodology. Softw Pract Experience 32(13):1239–1260

Poole C, Huisman JW (2001) Using extreme programming in a maintenance environment. IEEE Softw 18(6): 42–50

Poole CJ, Murphy T, Huisman JW, Higgins A (2001) Extreme maintenance. In: Software Maintenance. Proceedings. IEEE International Conference on, Florence, Italy. IEEE, pp 301–309

Popovic M, Atlagic B, Kovacevic V (2001) Case study: a maintenance practice used with real-time telecommunications software. J Softw Maint Evol Res Pract 13(2):97–126

Prochazka J (2011) Agile Support and Maintenance of IT Services. In: Information Systems Development, Prague, Czech Republic. Springer, pp 597–609. doi: 10.1007/978-1-4419-9790-6_48

Rose J (1997) Soft systems methodology as a social science research tool. Syst Res Behav Sci 14(4):249–258

Rudzki J, Hammouda I, Mikkola T (2009) Agile Experiences in a Software Service Company. In: 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA'09. IEEE, pp 224–228

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164

Schwaber K, Beedle M (2001) Agile software development with Scrum. Prentice Hall, Upper Saddle River

Senapathi M (2010) Adoption of software engineering process innovations: The case of agile software development methodologies. In: Agile Processes in Software Engineering and Extreme Programming, 11th International Conference, XP 2010, Trondheim, Norway. Springer, pp 226–231

Shaw S (2007) Using Agile Practices in a Maintenance Environment. Intelliware Development Inc

Sherer SA (1997) Using risk analysis to manage software maintenance. J Softw Maint Res Pract 9(6):345–364

Sidky A, Arthur J, Bohner S (2007) A disciplined approach to adopting agile practices: the agile adoption framework. Innov Syst Softw Eng 3(3):203–216

Singer J (1998) Practices of software maintenance. In: International Conference on Software Maintenance. IEEE, pp 139–145

Souza SCBd, Anquetil N, #225, Oliveira tMd (2005) A study of the documentation essential to software maintenance. Paper presented at the Proceedings of the 23rd annual international conference on Design of Communication: designing for pervasive information, Coventry, United Kingdom

Stachour P, Collier-Brown D (2009) You don't know jack about software maintenance. Commun ACM 52(11):54–58

Standards Coordinating Committee of the Computer Society of the I (1990) IEEE Standard Glossary of Software Engineering Terminology. vol IEEE Standard 610.12. Los Alamitos

Sukumaran S, Sreenivas A (2005) Identifying test conditions for software maintenance. Ninth European conference on software maintenance and reengineering. IEEE Computer Society, Manchester. doi:10.1109/CSMR.2005.32

Susman GI, Evered RD (1978) An assessment of the scientific merits of action research. Adm Sci Q 23:582–603

Svensson H, Host M (2005) Introducing an agile process in a software maintenance and evolution organization. In: Ninth European Conference on Software Maintenance and Reengineering, Manchester, United Kingdom. IEEE, pp 256–264

Taylor M, Moynihan E, Laws A (1998) Training for software maintenance. J Softw Maint Res Pract 10:381–393

Thomas D (2006) Agile evolution: towards the continuous improvement of legacy software. J Object Technol 5(7):19–26

Vans M, von Mayrhauser A, Somlo G (1999) Program understanding behavior during corrective maintenance of large-scale software. Int J Hum Comput Studies 51(1):31–70

Ware M, Wilkie FG, Shapcott M (2007) The application of product measures in directing software maintenance activity. J Softw Maint Evol Res Pract 19(2):133–154

Zanker M, Gordea S (2006) Measuring, monitoring and controlling software maintenance efforts. In: Thirteenth International Symposium on Temporal Representation and Reasoning, TIME 2006 Budapest. IEEE Computer Society, pp 103–110

Lise Tordrup Heeager holds a PhD (2012) from Aalborg University in Denmark. She is currently employed as assistant professor at the Information System Research group at the department of Business Administration at Aarhus University, Denmark. Her research interests include action research and case studies on systems development and system maintenance practice with an emphasis on agile software development and agile project management.

Jeremy Rose is Professor of Informatics at Skövde University, and Associate Professor at the Department of Computer Science at Aalborg University. He has worked with the PITNIT, SPV, DISIMIT, CaIN research projects in Denmark, and in a variety of engaged research and consulting roles. His research interests are principally concerned with IT and organizational change, IT and societal change, the management of IT, and systems development. He has published in management, systems, eGovernment and information systems journals and is associate editor at Communications of the AIS, Information Technology and People, and Systems, Signs and Actions. He was the founding director of the Centre for eGovernance at Aalborg University and a member of the steering committee for the Demo-Net European network of excellence. Current research themes include achieving value in eGovernment projects and managing innovation in small software companies (in connection with which he has recently finished a sabbatical project at the Judge Institute at Cambridge University funded by the Danish Research Council).