ICAM 2005 International Conference on Agility
Helsinki July 27–28, 2005

The Use of Architectural Patterns in the Agile Software Development of Mobile Applications

Tuomas Ihme, Pekka Abrahamsson
VTT Technical Research Centre of Finland,
P.O.Box 1100, FIN-90571 Oulu, Finland

Abstract: ***Architectural design patterns capture proven solutions of skilled designers to many recurring design problems. However, these patterns may lead to large solutions and overengineering, which are considered alarm signals from the viewpoint of agility . This paper reports the results of two case studies focusing on the adoption of architectural design patterns in agile development of mobile applications for real markets. The Agile Architecture Line Approach and Model were applied in the case studies. The used approach and model strive for a light and robust architecture design framework for mobile applications and services. In this approach, current architectural knowledge concerning available patterns and solutions will be captured during the Architecture Line Definition phase taking place before production. Based on the experience gained from the first case project, more emphasis was laid on capturing the current architectural knowledge about the patterns and solutions proven useful and effective in similar applications running on the used platform. The patterns are augmented before production with suitable supporting information so as to enable them to help inexperienced designers to improve the quality of mobile applications developed in nine-week agile projects in concordance with agile values. This paper demonstrates empirically that architectural design patterns can help to develop viable software architectures and to document them in a useful way, as applied in the challenging context of tough time-to-market demads, the mobile development environment and the J2ME platform. This paper further shows that pattern-based rationale of design decisions and architectural components can be a key success factor in designing mobile software and improving its quality. The empirical results of this paper are presented in a manner enabling practitioners to utilize the proposed solutions in similar projects.***

Keywords: Agile methodologies; Architectural patterns; Extreme programming; Mobile software; New product development

1. Introduction

Agile software development solutions [for an overview, see 1]can be seen to provide a good fit for the mobile application environment due to its high volatility and tough time-to-market demands. Mobile applications are generally quite small and majority of them are developed by small software teams.

While, so far, mobile commerce applications have not been very successful, telecommunications companies believe that the situation will change in the near future. This would lead to a widespread adoption of mobile services in combination with mobile commerce applications [2]. At present, basically anyone with the required skills can develop applications for mobile terminals thanks to open platform technologies such as Symbian and Java.

The few published articles on mobile software development argue for the existence of specific design challenges compared to traditional desktop applications development [3-5]. This study focuses on agile development of mobile applications from the viewpoint of using architectural design patterns.

Architectural design patterns [6-8] describe the expertise of experienced developers on solving recurring architectural design problems in specific design contexts. It has been argued that patterns help developers to create new software more effectively, while simultaneously maintaining a high quality of developed software. Furthermore, patterns are most effective when they enhance communication not only among designers but also between designers and other project stakeholders.

Agile software development relies on a defined set of values and principles guiding the development (see www.agilemanifesto.org). The principles and values motivating agile modeling practices, as drawn from the Agile Manifesto, have been presented by Ambler [9]. Similarly, the documentation of patterns is motivated by a set of values [10]. There are several similarities between the values stated by the promoters of agile solutions and those shared by the pattern authors. Contrary to a common belief that agile software development neglects the viewpoint of architecture development, many agile experts call for explicit attention to good design and architecture early in the development [e.g., 11, 12].

Many patterns represent the bottom-up engineering-oriented type. Examples of such patterns can be found in [6], [7] and [8]. They can be used to provide restructuring targets

in agile refactoring [13] activities for enhancing software modularity, maintainability, and reusability [6, 14].

The adoption of patterns has, however, proven difficult [6, 8, 15]. Only few patterns describe the architectural expertise of experienced mobile software developers [8] and there are only some publications available discussing examples and experiences of applying trusted general-purpose patterns in the construction of mobile applications. Kerievsky [16] argues that "unless people devote significant study to patterns, they will be in danger of misunderstanding them, overusing them, and overengineering with them." In eXtreme Programming (XP), patterns are most useful when people are familiar with them and use them in a disciplined XP way. Patterns are often implemented in their most primitive forms in early design phases and these implementations are altered or upgraded later [16].

The pattern-specific guidelines of pattern descriptions are insufficient for applying patterns in building complex real-world software architectures [8, 15]. The pattern descriptions fail to address the integration of patterns into a partial design, the combination of patterns to larger design structures, the application order of a given set of patterns, and the resolution of problems that cannot be solved by a single pattern in isolation [15]. Some pattern collections, e.g., [6-8], include useful but collection-specific pattern selection procedures.

This paper addresses the challenges introduced above and reports the results of two case studies where a mobile service and a mobile extension to an existing pc-based information system was developed. Both case projects used an agile software development approach called Mobile-D [17], which is based on XP practices. This paper focuses on the adoption of architectural design patterns in a situation where the developers are not familiar with related concepts, which can be seen as a common situation in mobile development environments.

The paper is organized as follows. The next section introduces the research design of the study. The paper continues with a summary of the supporting material of the case projects for the adoption of architectural design patterns. The main findings of the cases is then presented. Finally, some conclusions are drawn.

2. Research Design

Developing mobile applications is a challenging task due to the specific demands and technical constraints of mobile development. Very little is known about the use of architectural patterns a) in agile software development of b) mobile software systems from the practical viewpoint.

Mobile-D, an agile software development approach [17] based on the practices of eXtreme Programming, or XP, which is the best known agile method [11], is one of the pioneer approaches for mobile application development. The predecessor project [for details, see 18] for the case studies of this paper, i.e. the zOmbie and bAmbie projects, also showed that no patterns can found or utilized in 8-week Mobile-D projects without any support for the adoption. Due to these issues, the Mobile-D approach and two case projects were selected for studying if architectural design patterns could be used in the development of mobile applications for improving software architecture.

2.1 Research setting

- *Agile Architecture Line Model*

The Agile Architecture Line Model in Figure 1 provides the design and documentation context and process phases for adopting the architectural patterns in the case projects. Figure 1 presents the Architecture Line Model from the viewpoint of systematic piecemeal growth of architectural design and documentation in the different phases of the Mobile-D process. Architectural patterns represent one form of creating, understanding, communicating, and documenting viable architectural solutions.

The Agile Architecture Line Model [originally introduced in 19] strives for a light and robust architecture design frame for mobile applications and services. It aims at producing maintainable and expandable products with trusted infrastructures. The Architecture Line concept involves selecting and using architectural design patterns in a systematic manner over the course of agile projects. It relies on agile values and principles and attempts to mitigate the risks of overengineering in tough time-to-market development environments.

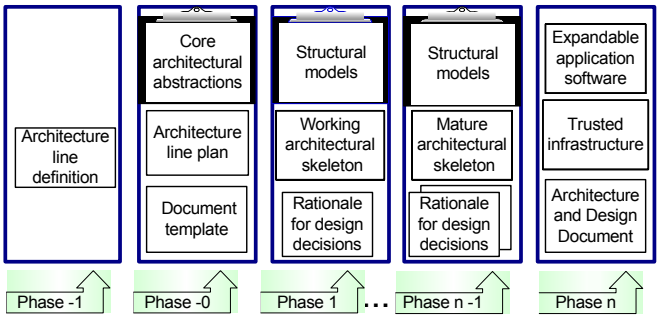

Figure 1. Agile Architecture Line Model

The growth of the architecture in an agile software development project progresses over a series of phases. Phase -1 includes activities performed prior to project commission and launch. Phase 0 consists of activities related to the first iteration of product development. Piecemeal and systematic growth of architectural understanding progress over subsequent phases as shown in Figure 1.

- *2.1.1 Case studies*

The zOmbie project team was developing a mobile service application for active investors for an easy access to financial services involving trading in the stock market and visualizing real-time market data and stock exchange indexes. The bAmbie project team was developing a mobile extension to an

existing production planning system enabling sales personnel to visualize the on-line the state of factory production anywhere in the world at any given time. The cases shared a common development concept: "From a scratch idea to a marketable java-based mobile application in nine weeks, running on several mobile phone terminals". The concept was realized via rapid iteration cycles and very small software releases of the Mobile-D process using the minimum-footprint J2ME platform on client mobile phones.

The zOmbie team included five 5th to 6th year university students and a research scientist with several years of experience in software development but very limited experience with the Java language and the J2ME platform. The bAmbie team consisted of three university students and one designer from the customer organization with no experience of the J2ME platform but with sound knowledge of the existing pc-based information system. None of the group members were working in both of the teams. All the students involved had participated in an introductory course on software architecture and architectural design patterns prior to the projects. The course had included a small exercise on the use of the J2ME platform.

The development environments of the two projects were almost identical. However, the MySQL database was used for the stock market data in the zOmbie project and the Oracle database for the production system data in the bAmbie project.

Both case projects involved six similar iterations or phases of the Mobile-D process (Figure 1): Phase 0 spanned one week, Phases 1 to 3 two weeks, and Phases 4 and 5 one week. Both case studies required one extra enhancement iteration at the end of the project in addition to the initial eight-week development schedule.

The applications of the cases shared the architecturally important key functionality of retrieving data from a database and displaying it on mobile phone terminals. The customers in the case projects did not specify any quality requirements for the applications. The main difference between the cases regarding the adoption of architectural design patterns was to be found in the supporting material created during Phase -1 prior to the start of the case projects and its use during the projects. The cases shared similar mobile applications, teams, development environments, the same agile development process, and J2ME, one of the most common platforms for mobile applications. Accordingly, the number of disruptive variables was very small. The zOmbie and bAmbie cases could thus be considered appropriate for this study focusing on the adoption of architectural design patterns in an agile development of mobile applications.

2.2 Research method

Cunningham [20] looks upon action research as one form of case study research. In action research, the focus is more on what practitioners do rather than what they say they do [21]. The multiple case study research as suggested by Yin [22] is the principal guiding research method in our study. Oquist [23] further points out that action produces knowledge for guiding practice, which also lays down the guidelines of this study.

The support group of the case projects included an experienced software architect with sound expertise in patterns. He contributed to the study by preparing and presenting supporting material for the adoption of architectural design patterns during Phase -1 (see Figure 1) before the teams started to work on the case projects. Action research requires that reality is modified using planned interventions [24]. The planned interventions took place between the agile projects. Retrospective project post-mortem workshops [25] and interviews at the end of each project were used for capturing experiences and learning from individual projects. This information augmented by action point lists was used for starting improvement actions for the Agile Architecture Line Model and for developing supporting material for subsequent case projects.

During the case projects, Phases 0 to n in Figure 1, the support group architect provided expert guidance on patterns whenever needed and requested by developers and particularly via project retrospectives. The Post-Iteration Workshop technique [25] was used for conducting project retrospectives after all iterations of the projects so as to improve the software development process, also including architectural design and documentation aspects. The support group architect did not intentionally intervene in the case projects since the goal of the study was to investigate whether an inexperienced team was able to make use of provided architectural patterns if supported by training material.

The architectural design and documentation issues, including utilization of proposed patterns, were totally optional for the developers due to the agile values (i.e., trust teams and working software over comprehensive documentation) adopted in the projects because the customer did not require architectural issues.

The data collection involved the use of the resulting documentation, code and design artifacts. In addition, consultation requests from the developers, project retrospectives conducted after iterations and retrospective project post-mortems and interviews conducted after the end of the projects were collected for research purposes.

2.3 Research problems

The main characteristics of the experimental setup were, as described above:

- The development concept "From a scratch idea to a marketable java-based mobile application in nine weeks, running on several mobile phone terminals",
- Inexperienced teams, and
- An agile development process with an emphasis on the totally optional utilization of the proposed patterns.

This research aims to study if architectural design patterns can be used for improving the software architecting process of

Mobile-D projects in the given experimental setup. This general aim is divided into the following specific problems according to the issues identified in the introduction:

- How to select and present patterns in the training material so as to motivate and facilitate their use by the project team?
- How can patterns help the team to create new software effectively, while simultaneously maintaining the quality of the developed software at a high level?
- How can patterns help the team to develop useful and effective software architectures? Northrop [26] identifies the following three fundamental reasons why software architecture is important: a basis for mutual understanding, consensus and communication among stakeholders, a manifestation of early design decisions about a system, and a transferable abstraction of the system. A useful architectural model with associated documentation has to be capable of describing adequately what the externally visible properties and roles of the components in the models are, the relationships among those components, and how the components cooperate to accomplish the purpose of the system [26].
- Are the Mobile-D approach with its agile values and the Agile Architecture Line Model suitable to be used with patterns?
- How can patterns be used as a documentation aid to preserve vital design information that helps to maintain and evolve the developed product after the project? Agile methods generally tend to ignore problems that may occur after a specific software product has been developed [1].

3. Supporting material for the adoption of patterns

An experienced software architect in the support group of the case projects was responsible for preparing supporting material for the adoption of architectural design patterns as well as for presenting the material during the training courses of the project teams before the start of the projects.

3.1 The zOmbie case

When preparing the supporting material during Phase –1, the architect had neither enough time nor enough information about the application idea of the zOmbie project for analyzing the application and technology domains. Therefore, the supporting material of the zOmbie case mainly included general information about architectural design patterns and their selection and also about architectural drivers. The material included some more detailed information about the following styles and patterns [7, 27]: the client-server style, the tiered style, the deployment style, the module decomposition style, and the Layers pattern. The material also included information about Java-specific J2EE patterns along with an example of a potential system-level architecture.

3.2 The bAmbie case

The supporting material of the bAmbie case included information about the following styles and patterns [6, 7, 27]: the Three-Tier-Client-Server style, the Broker pattern, the Model-View-Controller pattern, the Proxy pattern, the Façade pattern, the Layers pattern, the Singleton pattern, the deployment style, and the module decomposition style.

Based on the negative experience gained from the zOmbie case, the supporting material of the bAmbie case was complemented by information about examples [e.g., 28] of concrete implementations of most of the patterns on the Micro Edition of the Java 2 Platform (J2ME) with the Mobile Information Device Profile (MIDP). Based on these examples, the supporting material provided a small and manageable set of potential core abstractions for structuring each subsystem of the system to be developed. The abstractions and their relations resulted from the integration of pattern examples in the supporting material. The abstractions and their relations composed potential architectural skeletons for the system to be developed. They were also used in the proposal for software architecture design document template as examples of the use of patterns in documenting the rationale of architectural decisions and solutions.

4. Findings from the case studies

4.1 Empirical evidence of benefits: the cyclomatic complexity

Literature has produced little empirical evidence of the benefits of using architectural patterns in practice. In order to shed some light on this issue, the cyclomatic complexities of the zOmbie and bAmbie cases were calculated and compared. As noted earlier, the difference in the analyzed cases was the use of architectural patterns, which were used in bAmbie but not in zOmbie.

Cyclomatic Complexity	Risk Evaluation
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
greater than 50	untestable program (very high risk)

Table 1. Cyclomatic complexity values and risk of defects (the QJ-Pro tool available from http://checkstyle.sourceforge.net/)

Cyclomatic complexity measures the number of linearly-independent paths through a program module. It is a measure of the minimum number of possible paths through the source and therefore the number of required tests. Therefore, as Table

1 shows, a higher cyclomatic complexity value suggests a higher risk of defects.

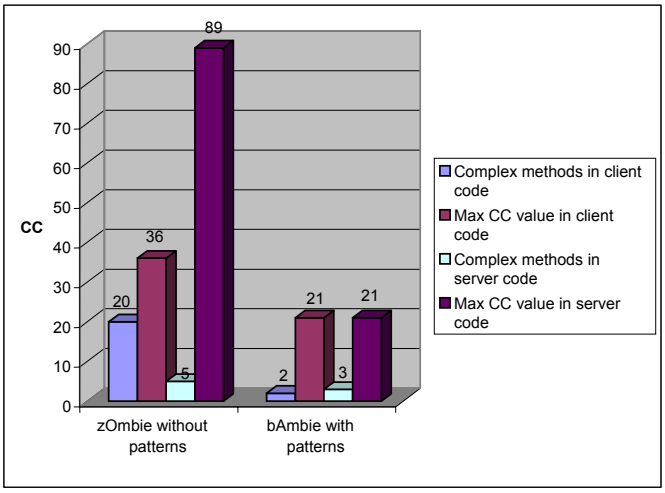

Figure 2. Cyclomatic complexity of code in two projects.

Figure 2 shows that the use of design patterns can clearly reduce program complexity. In bAmbie the client software showed two complexity points larger than 10, while in zOmbie this value was as great as 20. On the server side, the difference between the respective values was not quite as great, but in zOmbie the greatest complexity value in server code was as large as 89, which is certainly not a desirable result in view of the target values presented in Table 1.

4.2 Answers to research problems

The research problems in Section 2 are further divided into more specific questions (in italics) and attached with the answers obtained to these questions.

How can a small set of suitable and useful patterns be pre-selected if no quality requirements are specified and only a scratch idea about the functionality of the product is available? The rapid iteration cycles and very small software releases of the Mobile-D process require that the quality of the developed software is constantly maintained at a good enough level in terms of, for example, modifiability, testability, expandability and integrability. Many architectural design patterns support these kinds of quality attributes but, if not applied carefully, they may also cause problems in terms of performance, which must be of high level in the final application. The results showed that the preparation of useful pattern supporting material requires not only enough time and sound expertise on patterns and architecting, but also specific knowledge about the use of patterns in similar applications running on the specified platform. This specific knowledge can be acquired by analyzing application and technology domains, particularly focusing on pointers to potential pattern and architecture resources, such as pattern and style catalogues, examples of similar applications, and typical system and software architectures. Pattern collections such as [6, 7] and [8] include useful but collection-specific pattern selection procedures. Pre-applied architectural design patterns in the example material [e.g., 28] of the J2ME platform provider proved highly useful. The architecturally important key functionality of the system needed to be crystallized from the scratch idea of the products for searching examples of similar applications. In these cases, the key functionality was to retrieve data from a database and to display it on mobile phone terminals.

How can selected patterns be presented in the pattern supporting material in a manner that would make the project team willing to utilize them? The results showed that no patterns will be found or utilized if appropriate supporting material for their adoption is missing. In the worst case scenario, the teams are in danger of misunderstanding the patterns. Thus, the adoption of patterns without any working examples on similar problems and platform contexts is likely to fail. The supporting material of the bAmbie case included information about examples of concrete implementations of most of the patterns on the used platform. These examples enabled the supporting material to provide a small and manageable set of potential core abstractions for structuring each subsystem of the system to be developed. The abstractions and their relations resulted from the example integration of the patterns. The abstractions were also used in a proposal for software architecture and design document template as examples of the use of patterns in documenting the rationale of architectural decisions and solutions. The results showed that the teams had no time to use more than the presented part of the supporting material.

How can patterns help the team to create new software more effectively? The results showed that patterns in the supporting material helped the bAmbie team to capture a comprehensive, appropriate and understandable set of core architectural components at the beginning of the project. This contributed to enhanced productivity in terms of logical lines of code per person month and better task estimation in the bAmbie case.

How can patterns be used as a documentation aid to preserve vital design information that helps developers evolve software iterations more efficiently and maintain a high quality of the developed software? Pre-defined patterns and pattern-based core architectural abstractions in the supporting material allowed the team to create a comprehensive and appropriate set of core architectural components at the beginning of the project for structuring each subsystem of the system. The patterns and abstractions provided excellent targets and design information for refactoring the architecture of the system. The zOmbie case, which did not employ patterns led to the simplest architecture at the beginning of the project and the same architecture was maintained during the whole project, which led to large and complex components in the final architecture as shown in Figure 2.

How can patterns help the team to develop software architecture descriptions that can be used as a basis for mutual understanding, consensus and communication not only in the project team but also among other project stakeholders during the project? Two non-standard software models were created during Phase 1 in the zOmbie project (Layer Description and Composite Model). The models included a large number of different kinds of weakly defined components and relationships among components. These models, however, failed to describe software architecture in a useful way [26] for stakeholders outside the project team. The non-standard software models of the bAmbie case on flip charts were usable for architectural communication among stakeholders during the development because the components in the models were derived from high-quality core abstractions in the pattern supporting material.

How can patterns help the team to develop software architecture descriptions that represent the manifestation of early design decisions? Some indications of early design decisions, but no design rationale, regarding the partitioning of the software system can be identified from the development time models of the zOmbie case. The design decisions of the software models of bAmbie on flip charts could, however, be traced and they were also included the pattern supporting material. Patterns encouraged the team to document also the design decisions and rationale that were not based on the information in the supporting material. The software architecture descriptions in the final document represent the manifestation and rationale of design decisions derived from the descriptions of the used patterns.

How can patterns help the team to create software architecture models that are transferable across successive iterations as well as across similar systems? The Layer Description of the zOmbie case was transferable across successive iterations. Since the other models on flip charts and in the final document list almost all classes in the system, they had to be modified according to class insertions or removals. The software models of bAmbie on flip charts were transferable across successive iterations due to their background of modeling examples provided by the supporting material. The software architecture descriptions in the final document were transferable also across similar systems, e.g., they could be applied to the architecture of the system developed in zOmbie.

Is the Agile Architecture Line Model suitable for using patterns? The results of the bAmbie project showed that patterns could be used in the fast-cyclic Mobile-D agile development process for new software via pre-defined pattern-based core architectural components while simultaneously maintaining the desired phasing and pacing rhythm of the process. However, some problems were also encountered. More effort and guidelines is needed for the development of supporting material for the adoption of patterns.. The Mobile-D process included the idea of creating a communication channel from the cell phone to the database during the first one-week iteration. The bAmbie project followed this idea and ended up with run-time architecture and architectural components that were too difficult to restructure for integrating the Model-View-Controller (MVC) pattern [7]. The particular importance of the application order of patterns at the beginning of the project should be emphasized more in the guidelines of the Mobile-D process.

Are the adopted agile value of "trust teams" and particularly the derivative application of the "trust teams" value, the totally optional utilization of the proposed patterns, suitable for the adoption of patterns? The Mobile-D approach is based on XP practices. XP expects the majority of the project team members to be on expert level. While the 'trust teams' value is well suited to teams of experts, it may cause problems for more inexperienced teams, such as those involved in this study. Better results from the viewpoint of using patterns and other architectural issues would have been archived if the project team had been obligated to consult the support group when encountering problems in the adoption of proposed patterns and core architectural abstractions.

Is the agile value "working software over comprehensive documentation" suitable for the adoption of patterns? All Mobile-D projects have followed this value. The results of the bAmbie case suggest that patterns can help and encourage the project team to document knowledge about software architecture in a brief and salient form [11] during the development period and particularly in the final documentation. The architectural documentation of the bAmbie project was as brief as that of the zOmbie project while simultaneously being more salient [11] in discussing the overall architectural design rationale and focusing more on the essential and high-level structures in the system. The agile project teams were not too keen to document software architectures, so it was important that patterns would support their documentation work while not increasing their documentation effort.

How can patterns be used as a documentation aid to preserve vital design information that helps to maintain and evolve the developed product after the project? Software architecture knowledge remained tacit in the zOmbie project and had remained tacit in its predecessor mainly due to a common misunderstanding that the agile value of "working software over comprehensive documentation" neglects the viewpoint of software architecture. The software architecture descriptions in the final documents of these projects were not useful from the viewpoint of the criteria presented in [26]. This may cause serious problems for maintenance and further development activities after the projects particularly in case the project team changes. In bAmbie, quite reversely, the high-quality pattern-based software architecture descriptions recorded in the bAmbie final document have been used as a basis for mutual understanding, consensus and communication among all stakeholders after the project, including continuation projects along with a study course work and a university graduation thesis. This was achieved particularly via the software architecture and design document template proposal in the supporting material including examples of the

use of patterns in documenting the rationale of architectural decisions and solutions.

5. Conclusions

This paper has argued that mobile applications are developed in a turbulent business environment, which requires the ability to react rapidly to changing market needs. It was suggested that agile software development solutions are targeted for this type of environment. Architectural design patterns describe the expertise of experienced developers on fundamental structuring principles of software systems and in solving recurring design problems in specific design contexts. Very little is known about the application of these patterns in the context of agile development of mobile application. Empirical evidence, experiences and lessons-learned regarding the application of architectural patterns are in short supply. This paper makes an attempt to shed light on these issues.

The results of the adoption of architectural design patterns in two case studies on the development of mobile applications have been described. The applications were developed from a scratch idea in 9 weeks with rapid iteration cycles and very small software releases using an XP-based agile method called the Mobile-D as the development frame and the Agile Architecture Line Model. of Mobile-D as the architectural design frame. The Agile Architecture Line Model defines the development of viable architecture solutions for mobile software applications in a stepwise and phase-based manner. The applications shared an architecturally important key functionality of retrieving data from a database and displaying it on mobile phone terminals. The team members were not highly experienced with software architecting, the J2ME platform, or architectural design patterns. The customer did not require the use of patterns. Thus, the use of patterns was optional for both of the project teams. The context, therefore, was highly challenging for the adoption of architectural design patterns. The difficulties faced have also been documented and reported here.

The results show that patterns are not likely to be found or utilized if appropriate support for their adoption is missing. In the worst-case scenario, the teams are in danger of misunderstanding the patterns. Thus, the adoption of patterns is bound to fail if there are no working examples on similar problems or platform contexts available. Software architecture descriptions were not useful [26] in the final documents of the zOmbie project, in which no patterns were used. The zOmbie application was not modular enough.

The bAmbie project showed that patterns can support agile values also in difficult adoption contexts such as those of the case projects if the patterns have been augmented before production with suitable information in the supporting material. This addition captures current architectural knowledge about the patterns and solutions that have proven to be useful and working in similar applications running on the same platform. The integration of the patterns resulted in a set of potential core abstractions and their relations. This enabled the team to make good use of patterns for developing viable software architectures and documenting them in a useful way and also for creating sound rationale for the trusted and high quality architectural solutions of the project. Pattern-based rationale of design decisions and architectural components proved to be a key success factor in designing and refactoring software architecture under the challenging development concept of this study. Empirical evidence in terms of code complexity demonstrated the impact of the use of patterns on achieving more balanced software structures.

The augmented patterns supported agile development values such as the Simplicity and Communication values of Agile Modeling [9] and also the agile principle of “Continuous attention to technical excellence and good design enhances agility” in the context of the bAmbie project. The principle of optional use of supporting material originated from the agile value of “trust teams”. This caused problems in the application order of patterns. The data and empirical material presented in the paper serve as a useful pointer for practitioners developing mobile applications and researchers interested in agile development and use of architectural patterns.

The Agile Architecture Line Model has been empirically tested in four Mobile-D case studies. The future research will focus on industrial cases and on radical acceleration of product development by developing agile architecture lines for series of products and for initiating software product lines and application frameworks.

Acknowledgements

The main research work of this paper has been carried out within the ITEA-Agile project funded by the National Technology Agency of Finland (Tekes) and VTT. The authors wish to thank the zOmbie and bAmbie teams for valuable and fruitful co-operation.

References

1. Boehm, B. and R. Turner. 2003. *Balancing Agility and Discipline*, Boston: Addison-Wesley.

2. Blazevic, V., A. Lievens, and E. Klein. 2003. "Antecents of project learning and time-to-market during new mobile service development," *International Journal of Service Industry Management*, 14(1):120-147.

3. Dahlgren, F. 2001. "Future Mobile Phones - Complex Design Challenges from an Embedded Systems Perspective," *Engineering of Complex Computer Systems*.

4. Yoshimi, B., et al. 2002. "Lessons Learned in Deploying a Wireless, Intranet Application on Mobile Devices," *Workshop on Mobile Computing Systems and Applications (WMCSA'02)*.

5. Konchnev, D.S. and A.A. Terekhov. 2003. "Surviving Java for Mobiles," *Pervasive Computing*, (April/June):90-95.

6. Gamma, E., et al. 1995. *Design patterns: Elements of Reusable Object-Oriented Software*, New York: Addison-Wesley.

7. Buschmann, F., et al. 1996. *Pattern-Oriented Software Architecture - A System of Patterns*, Chichester, England: John Wiley & Sons.

8. Schmidt, D., et al. 2000. *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*: John Wiley & Sons.

9. Ambler, S.W. 2002. *Agile modeling*, New York: John Wiley & Sons, Inc.

10. Schmidt, D., R. Johnson, and M. Fayad. 1996. "Software Patterns," *Communications of the ACM*, 39(10):36-39.

11. Martin, R. 2002. *Agile Software Development, Principles, Patterns, and Practices*, Upper Saddle River, New Jersey: Prentice Hall.

12. Beck, K. 2000. *Extreme programming explained: Embrace change*, Reading, MA.: Addison Wesley Longman, Inc.

13. Fowler, M. 1999. *Refactoring: Improving the design of existing code*, Boston: Addison-Wesley.

14. Keriewsky, J. 2004. *Refactoring to Patterns*, Boston: Addison-Wesley.

15. Buschmann, F. 1999. Building Software with Patterns, in *Proceedings of the Fourth European Conference on Pattern Languages of Programming and Computing, Bad Irsee, Germany, 8 - 10 July 1999*. 1999. p. 58.

16. Keriewsky, J. 2000. "Patterns and XP," in *Extreme Programming Examined: Selected Papers from the XP 2000 Conference. XP 2000 Conference, Cagliari, Italy*, G. Succi and M. Marchesi, Editors, Addison-Wesley: New York. p. 207-220.

17. Abrahamsson, P., et al. 2004. "Mobile-D: An Agile Approach for Mobile Application Development," *Object-oriented programming systems, languages, and applications,* Vancouver, BC, CANADA, pp. 174 - 175.

18. Abrahamsson, P. and J. Koskela. 2004. "Extreme programming: A survey of empirical results from a controlled case study," *ACM-.IEEE International Symposium on Empirical Software Engineering (ISESE 2004),* Redondo Beach, CA, USA, pp. 73 - 82.

19. Ihme, T. 2001. "An architecture line structure for command and control software," *27th EUROMICRO Conference,* Warsaw, Poland: IEEE Computer Society Press, pp. 90-96.

20. Cunningham, J.B. 1997. "Case study principles for different types of cases," *Quality and quantity*, 31:401-423.

21. Avison, D., et al. 1999. "Action Research," *Communications of the ACM*, 42(1):94-97.

22. Yin, R.K. 1994. *Case Study Research Design and Methods,* 2nd ed.: Sage Publications.

23. Oquist, P. 1978. "The epistemology of action research," *Acta Sociologica*, 21:143-163.

24. Susman, G.I. and R.D. Evered. 1978. "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly*, 23:582-603.

25. Salo, O., et al. 2004. "Self-Adaptability of Agile Software Processes: A Case Study on Post-Iteration Workhops," *5th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004),* Garmisch-Partenkirchen, Germany: Springer.

26. Bass, L., P. Clements, and R. Kazman. 1998. *Software Architecture in Practice*, Reading, Massachusetts: Addison-Wesley.

27. Clements, P., et al. 2002. *Documenting Software Architectures, Views and Beyond*, New York: Addison-Wesley.

28. Yuan, M. 2004. *Enterprise J2ME, Developing Mobile Java Applications*, Upper Saddle River, NJ: Prentice Hall.

Authors' biographies:

Lic. of Tech. **Tuomas Ihme** is a senior research scientist at VTT Technical Research Centre of Finland. His professional experience involves several years in industry as a software engineer and a project manager, more than 20 years experience in development and research of embedded software, and management of industrial development projects as well as national joint research projects. His areas of

expertise are software architectures in embedded and wireless systems, architecture design methods and tools, architecting component software, product line architectures, and architectural patterns. His research interests are currently centered on agile software architectures in embedded systems. He has authored more than thirty scientific publications focusing on methods, tools, reusability, quality, and architecture of embedded software.

Prof. **Pekka Abrahamsson** is a senior research scientist at VTT Technical Research Centre of Finland. He received his Ph.D. on "The role of commitment in software process improvement" from University of Oulu in 2002. His research interests are currently centred on agile software production, embedded systems and mobile application development. His current responsibilities include managing an AGILE-ITEA project (http://www.agile-itea.org), which involves 22 organizations from 8 European countries. He has coached several agile software development projects in industry and authored 50+ scientific publications focusing on software process and quality improvement, commitment issues and agile software development. His team has developed the Mobile-D methodology for mobile application development. His professional experience involves 5 years in industry as a software engineer and a quality manager.