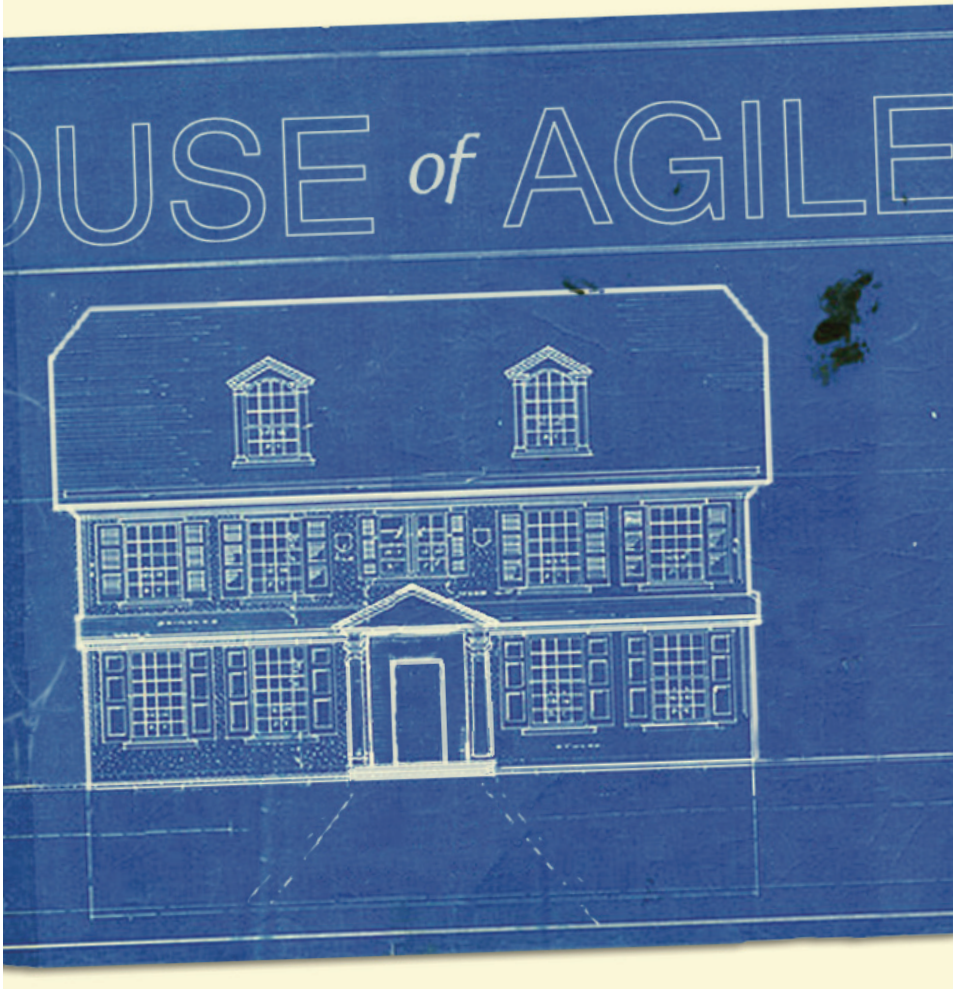
ITEA
INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Agile software development of embedded systems

AGILE

Newsletter 1/2006



Agile embedded framework - the "House of Agile" will be published on December 15th, 2006 in http://www.agile-itea.org.

PROJECT MANAGERS´ GREETINGS

AGILE project is gearing towards its final phases. The results have been outstanding both in industrial and academic sense. The consortium members have executed more than 80 industrial trials with different agile methods, practices and tools. These results are being packaged in a unique web portal capturing the principal findings of the project.

This newsletter presents a tip of our results´ ice berg. Dive in for an exciting experience!

Agile greetings,
Pekka Abrahamsson, VTT
Ko Dooms, Philips

CONTENTS:

>> AGILE IN BRIEF

AGILE-ITEA project presents a new software development technology capable of producing a significant improvement for the competitive position of the embedded software industry in Europe. The project develops agile software development solutions for the embedded domain. The goal is to to increase the reliability, productivity and reduce the risk of embedded software development. The solutions are validated by industrial trials in many different application domains.

XP 2006 in Oulu, Finland.

Kent Beck and Barry Boehm at XP2006

XP 2006, i.e. "the 7th International Conference on eXtreme Programming and Agile Processes in Software Engineering" was organized in Oulu, Finland in June 2006 by VTT Technical Research Centre of Finland and the University of Oulu.

As the largest agile conference ever organized in Europe, the five day program included altogether 18 tutorials, five keynote speeches, 22 research and experience paper presentations, 20 different workshops and activities, three cutting edge panel discussions and poster demonstrations, as well as 20 round-the-clock open-space events self-organized by the enthusiastic agile crowd. All together, 283 industrial and academic professionals from 25 countries around the world, and from over 100 different companies, gathered together to discuss their needs and ideas for incorporating agile methodologies into the production models. In this respect, the conference can undoubtedly be regarded as a huge success in terms of both quality and quantity!

The conference was also honoured with the presence of two legends in software industry; Barry Boehm (Professor of Software Engineering, Computer Science Department Director at USC Center for Software Engineering and creator of, e.g., COCOMO, Spiral Model, and the Theory W approach) and Kent Beck (Founder and Director of Three Rivers Institute, the father of methodology and one of the founders of the Manifesto).

In all, the success of XP2006 is one indicator that agile methodologies and their variants are here to stay. The enormous and ongoing interest of both the industry and academia shows no signs of subsiding. However, it is also evident that even though the agile ideologies and methodologies have emerged since the mid 1990's, there is still a great need to keep the debate and evolution moving upwards and onwards. In fact, a majority of software organizations functioning in complex human and technical ecosystems are still contemplating whether to make the transition towards agile software development and how they should actually proceed in pursuit of increased quality and customer satisfaction, and decreased lead-time and costs.

XP2007 will be organized in June, 2007 in Como, Italy. Welcome to join the agile crowd!

Outi Salo
Organizing Chair of XP 2006, Research Scientist
outi.salo@vtt.fi
Pekka Abrahamsson
Program Chair of XP 2006, Research Professor
pekka.abrahamsson@vtt.fi

Material available:
http://virtual.vtt.fi/inf/pdf/symposiums/2006/S241.pdf
(Collection of Tutorials, Workshops, Activities and Key Note Speeches of XP2006). http://www.xp2006.org, http://agile.vtt.fi

INSIDE VIEW

Behind Every Great Idea: THE VALUES AND PRINCIPLES OF AGILE

"Much is made of Agile practices, but in order to really revolutionise your software development, you need to start thinking in an agile way in order to behave in an agile way", says **Sean Hanly**, MD of Exoftware.

Discussions on Agile software methods usually focus on agile practices and their benefits and challenges for teams. The actual values and principles of Agile often take a back seat. Even during Agile transformations, companies often relegate Agile values and principles to inspirational talks.This makes sense in one way. As agile practices are concrete and very specific, they seem like a logical starting point for companies eager for change.

However, if we slavishly implement Agile practices without really appreciating the values and principles of Agile, we are really missing the mark. By blindly following the practices without thought to the particular situation, we are likely to fail. Maybe we won't fail as badly as we would without Agile but somewhere down the road the process will inevitably splutter to a halt .

To prevent this unseemly end, Agile principles - not just practices - must be embedded into an organisation, so companies can apply Agile thinking rather then just practices. A deep understanding of Agile thinking allows companies select the right Agile practice, and adapt it for their given context. In his way, companies begin to think agile and then begin to act agile, regardless of what practices they are using. Companies that make this cognitive leap are able to deal with almost any software or business situation.

Let's take the practice of daily stand ups. We can ask ourselves: how do we deal with this as a large, distributed team? How do we deal with offsite partners and clients? What are daily stand ups really forcing us to think about?

Daily stand ups are really about getting us to have as much face-to-face communication everyday. Interacting with our team as much as possible. And why is this important? So that our software is delivering exactly what the customer wants and any issues or questions can be resolved quickly. As we begin to really dig deeper into the values and principles of Agile we can make intelligent decisions on how to adapt and apply practices to specific situations. For our large, distributed team, this may mean doing things like team stand ups that report into project wide stand-ups. It may mean making use of video-conferencing or sending team ambassadors.

So while, on the surface, daily stand-ups may seem unthinkable in a large distributed team, by delving deeper into Agile values we can start to find new applications for Agile ideas.

When considering Agile, we shouldn't underestimate its most powerful aspect which is its ability to be extremely adaptable.

In this respect, Agile has taught us to disregard convention and focus more on the core goal of any software project: to deliver real business value to customers.

Sean Hanly,, MD of Exoftware

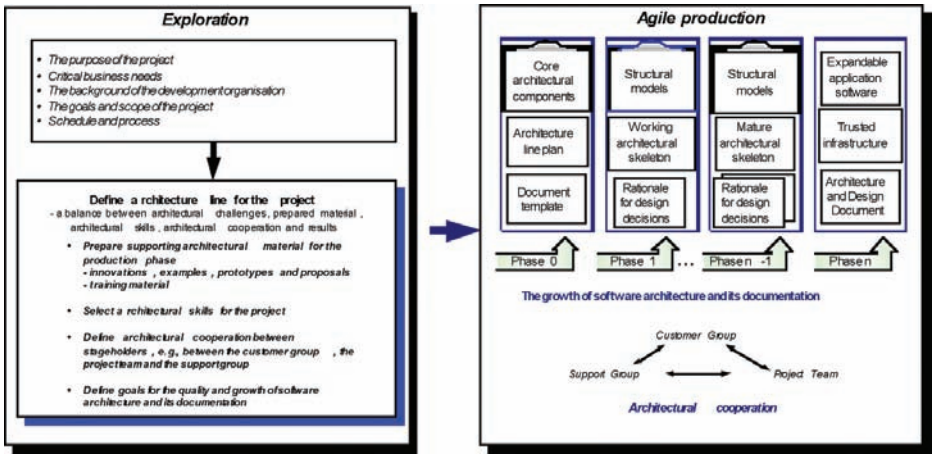
exoftware Be Agile

AGILE ARCHITECTURE LINE

Agile architecture line allows project teams to produce expandable prototypes and products with desired quality attributes such as flexibility and maintainability. It helps product management to maintain focus not only on the critical architectural issues of the current product but also, when needed, on the future directions of production releases to the marketplace.

The agile architecture line of a project is created by identifying, analysing and making decisions about critical issues that will have a strong but often indirect influence on the architecture of the software to be developed. During the early preparation of the project, the issues can be determined from the purpose and critical business needs of the product as well as from the background of the organisation or organisations constructing and maintaining the product.

The architecture line definition before production aims a balance between architectural challenges, prepared material, architectural skills and architectural cooperation so that the desired structural quality properties of the product will be achieved while simultaneously maintaining the desired phasing and pacing rhythm of the agile production phase. The agile architecture line of the project will be customized to the goals and scope of the project and its resources and development process because different projects, teams and systems need different architectural approaches.



The phases in agile architecture line development

The agile architecture line decisions and selections must be adjusted to the talents of available architects and application domain experts as well as available related architectural software assets and development tools. The prepared supporting architectural material enables the project team to use existing architectural innovations that have proved useful and working on similar problems and platform contexts. Cooperation between architects in different groups such as the customer group, the project team and the support group is needed because, for example, the chief architect of an organisation can seldom work as a team member of one specific project.

Core architectural components and their relations are important issues in the beginning of the production phase. Pattern-based components provide useful information for their rationale and also for architectural communication. The final and optimised version of the architecture will be documented in the short and silent software architecture and design document.

The agile architecture line approach has been empirically tested in many Mobile-D™ application development cases. The results support the assumption that customized architecting, on which the agile architecture line is based, can be a key success factor in the agile development of mobile and embedded software.

For more information, contact
Tuomas Ihme (tuomas.ihme@vtt.fi)

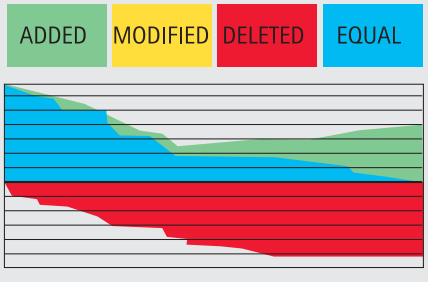
500.000 lines of agile code

The Modena project assignment was to revive an inherited client – server software solution for online DRM based media distribution. The inherited software solution consisted of 800 KLOC and roughly 8 pages of documentation. This software was supposed to be functionally complete and had to be brought to a quality level ready for product release. In addition some components based on proprietary standards had to be redesigned to base them on open standards.

Using a team of experienced senior software designers, the software was reverse-engineered and partly redesigned. Because of the unavoidable unfamiliarity with the code a process was set up to minimize the chance of regression caused by mistakes or misinterpretations. The build, test and reporting process was fully automated using SoftFab. Daily, it generated new builds of the software product, ran static and dynamic tests on the software, generated documentation from the source code and generated statistics on code size and coverage. The SoftFab environment also allowed for a complete focus by the team on the real challenges. Test driven development was done to keep the redesigns small and compact and to validate changes early in the process. Pair programming in combination with pair review was used to validate re-designs of components and major changes in components.

On a weekly basis, the automated process was used to release a working version of the product to the customer. These versions were reviewed and discussed with the customer on a weekly basis to determine the new priorities for development. As the software was also needed in a few pilot test environments, scrums were used to get extra focus to get the required functionality for these pilots working on time.

Using all these techniques the software was successfully revived, the functionality extended and the code size reduced, which resulted in a very satisfied customer (4.9/5 point scale). The same techniques also had a positive impact on the productivity data and defect injection data. The project showed excellent scores, on a total code size of 422 KLOC, the project's productivity was 8 times faster than the industry average with a quality rating 3.5 times better than the industry average.



Code size changes over time in the Modena project

PHILIPS Maarten Nielen
maarten.nielen@philips.com

Architecture Centric Agility

A sound component based software architecture is crucial for embedded systems development. Although some agile ideologists oppose it, agile practices can and need to be combined with traditional software architectural design. K.U.Leuven is studying how to work out and maintain such a 'big up front design' in an agile way. This article gives a first look at the research on hand.

As people are starting to apply new agile practices in all sorts of software projects, they often come to value certain more classical practices, but are not sure whether or how to integrate them both. A nice example of such a 'classic' practice is software architectural analysis and design. XP has proven that small-sized projects can significantly reduce their up-front design activities by making use of a well-balanced mix of collective code ownership, refactoring and testing. However, a good software architecture still remains paramount to the success of large and/or long term software projects.

Software architecture represents the overall structure or structures of the software system, comprising software components, the externally visible properties of those components, and the relationships among them. It is important to enable communication among stakeholders and to provide a solid and proven foundation on which to base the rest of the design. When the software architecture can define suitable components for an application, - the components can be developed relatively independently of one another, potentially by different teams; - the application may be upgraded in smaller increments by upgrading only some of the components that comprise the application; - components may be shared between applications, creating opportunities for reuse, but also creating inter-project dependencies.

K.U.Leuven's experience of architectures (including embedded component based architectures and agent based architectures) has lead to the research and development of component frameworks, resource-aware systems with built-in Quality-of-Service (QoS) and context mechanisms and supporting methodologies and middleware. Currently we are studying the integration of lightweight software components within an agile development ecosystem. Through the process of adapting and complementing agile principles we are developing a prototype of an architecture-centric agile development process.

One of the major differences between this and the approach of most agile methods is the refactoring barrier (see Figure 1). Refactoring may cause changes to ripple throughout the entire application. Once changes entail component interface

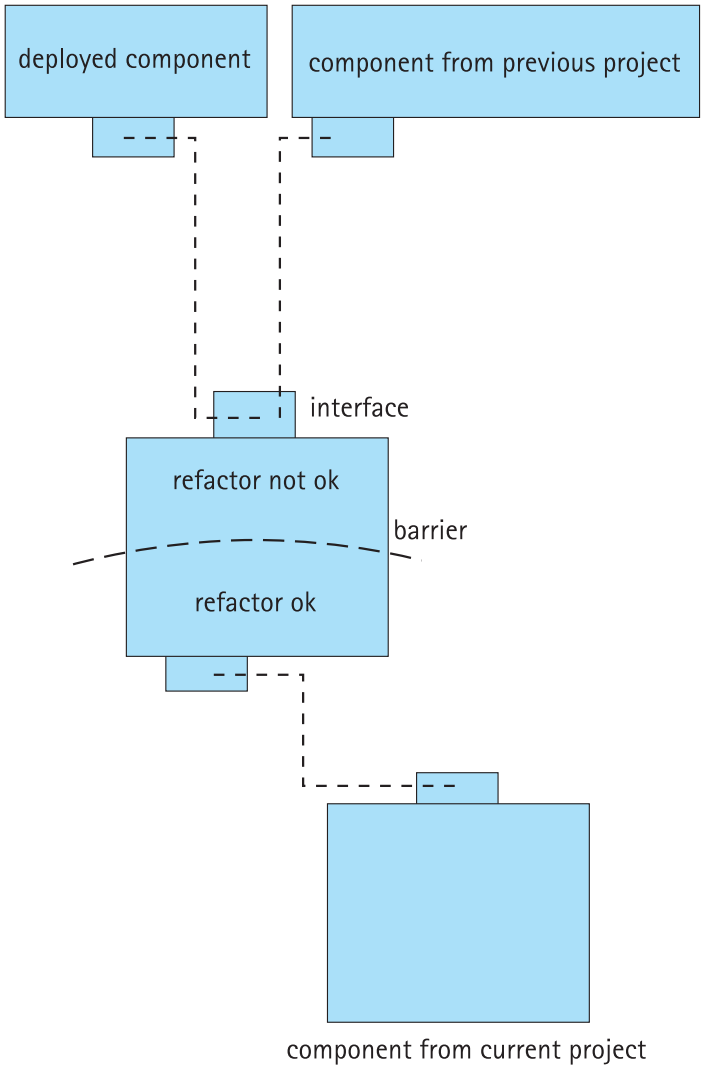


Figure 1: The refactoring barrier prevents modifying part of the software architecture.

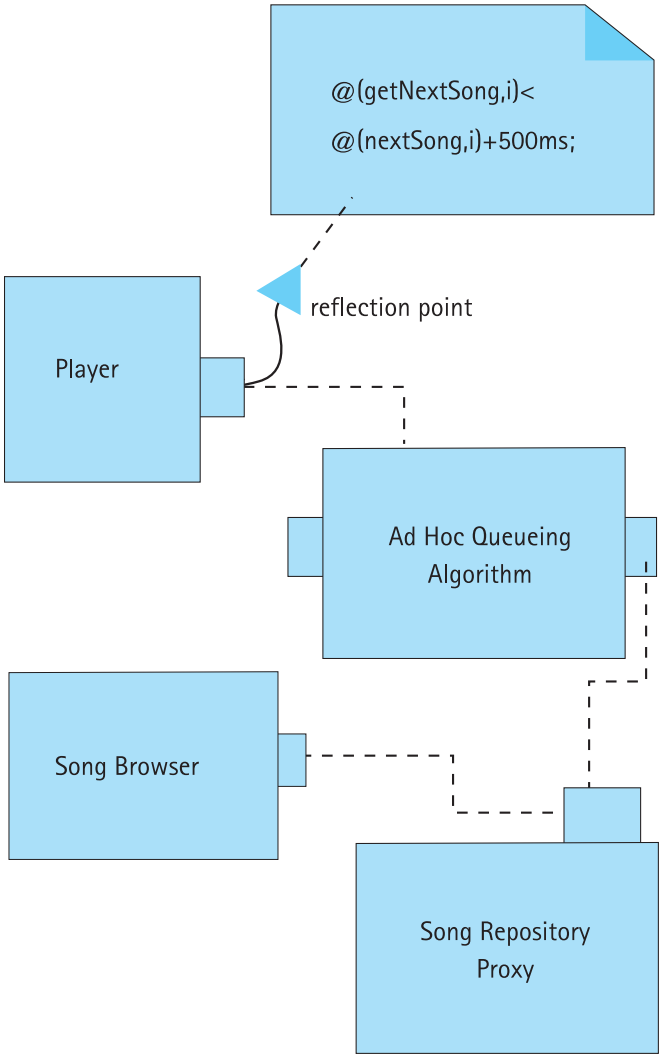


Figure 2: A simple software architecture diagram embedding a timing constraint.

changes, they break the refactoring barrier and with it some major reuse properties. Indeed refactoring does not change the application's overall behavior, yet it changes individual component behavior. A mixed approach will try to restrict the refactoring towards the interiors of a component without affecting its interfaces. In fact, components can be seen as application parts the behavior of which should also be preserved during refactorings. As such, ripple effects can be restricted to individual components instead of spreading out through the whole application.

Another major difference with traditional agile design is the explicit treatment of non-functional constraints. Software components form an important technology that encourages reuse and sound architecture-centric design. Traditional software architecture design activities are up-front, top-down and documentation-oriented and they are driven by quality attributes of the system to be developed. Agile features or stories on the other hand describe mostly functional behavior. Although these stories may be used to incorporate non-functional concerns, they are hardly suited for it. Most agile methods realize the simplicity principle in architectural design by assuming that architectural design is high-level design without explicit quality attributes. Implicitly, the methods lean on object-oriented and bottom-up approaches, neglecting the importance of the architecture for the realization of the quality attributes.

Software architectural design puts forward the use quality scenarios in order to realize the required quality attributes. These structured scenarios could very well complement agile functional descriptions. Of course, agility requires that all behavior is automatically tested. That is why we take architecture one step further and embed constraints directly into the architecture using a concept called reflection points. Reflection points allow precise description of non-functional behavior that can be automatically tested. They may also be used to detect exceptional situations at runtime, improving software behavior and robustness. Figure 2 describes a simple architecture with a timing constraint that is realized within a testable reflection point.

Other aspects of our research on architecture-centric agile practices includes tool support for component unit testing and static and dynamic architectural evolution.

KATHOLIEKE UNIVERSITEIT LEUVEN

Andrew Wils
Andrew.Wils@cs.kuleuven.be

Stefan Van Baelen
Stefan.VanBaelen@cs.kuleuven.be

From awkward processes to efficient agile processes

PROCESSES WITHIN ORGANISATIONS

Organisations are striving for efficiency through the improvement of their business and software processes. In the past, a wide range of notations were in use to describe processes at different levels within organisations. This variety produces interoperability problems because of the different notations and the complexity to share and construct processes. Standardising the process descriptions considerably reduces the challenge of adoption. One possible standardisation approach is to use metamodels to describe both business and software processes. These standard models are not widely used in the Industry but they provide conceptual representations in a consistent and formal way. The usage of models compliant with a standardised metamodel provides a common understanding for sharing and to constructing processes. In addition, this conceptual representation allows the elements involved in a process to be checked for consistency (e.g., work products, in/out, etc.). The development of a model-based framework also provides a formal way to apply agile patterns in the organisation.

AGILE PROCESSES WITHIN ORGANISATIONS

Organisations interested in agile practices need a way to define their agile processes. The definition of agile processes contains specifications suitable for projects characterised by their tight delivery schedules [1]. Technology changes and business requirement increase the uncertainty of software projects. This encourages organisations to shift towards adopting agile practices. However the wide range of agile methods complicates their adoption.

The definition of a framework for agile patterns allows the application of agile practices in different organisational contexts. In project runtime, agile patterns address different aspects of the development process and they can be combined to suit the specific requirements of a project.

Modelling agile patterns enhances the benefits of their usage and facilitates the adoption of agile process initiatives.

A MODEL BASED FRAMEWORK FOR AGILE PATTERNS

Modelling a set of agile patterns provides several benefits for potential adopting organisations. These benefits arise from the synergy between the usage of patterns and the usage of models. However, an appropriate infrastructure for defining and managing agile processes has to be established as well.

We define the underground metamodel within this infrastructure as one which is able to instantiate patterns as models. Once the agile patterns are defined using this approach, users can adopt specific and proven agile practices to solve specific problems. The resulting processes can be stored in the organisation's set of standard processes, once their validity has been proven.

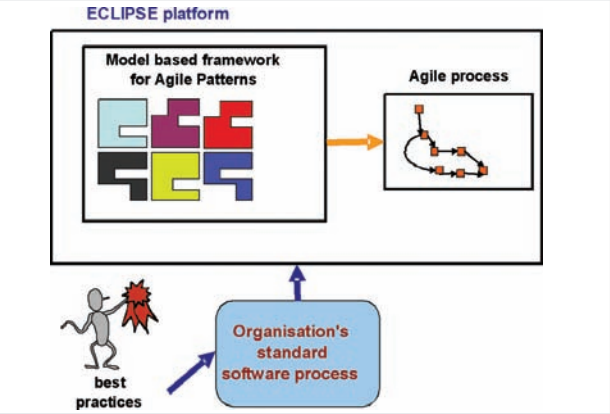


Figure : Overview of our approach

INFRASTRUCTURE

A metamodel for agile patterns is developed in order to define and to use consistently the definition of these patterns. This metamodel is based on MOF (Meta Object Facility), and more precisely on EMOF (Essential MOF).

This infrastructure is defined within the Eclipse open platform in line with the standardisation initiative of the Object Management Group (OMG) to define software processes (Software Process Engineering Metamodel 2.0). http://www.eclipse.org/proposals/beacon/main.php We describe patterns in terms of three main elements: problem, solution and consequences. Solution defines a set of activities that resolve the described problem.

Figure 1 shows the overview of our approach. Users can combine patterns to solve their problems quickly. Moreover users can choose convenient activities from the organisation's set of standard software processes database.

Benefits in practice

Agile supporters promote the idea of "Working software over comprehensive documentation" [2] and the usage of this model-based framework completely supports this principle. Moreover, it provides additional benefits. In particular, the approach facilitates the update of the software processes with synergies and experience gained in different projects. In this way, agile pattern models stakeholders are able to share the successfully applied practices amongst different parts of organisations and amongst projects.

The usage of the model-based framework automates the processes definition activities reducing the effort needed to start using agile practices.

References

1. Pekka Abrahamsson et al "Agile software development methods: state of the art review", AGILE-PÄIVÄ, Pori, Finland . Presentation (2003)
2. Agile Manifesto, http://www.agilemanifesto.org
3. T. Bozheva, Maria Elisa Gallo, Framework of Agile Patterns, Proceedings of EuroSPI (2005)

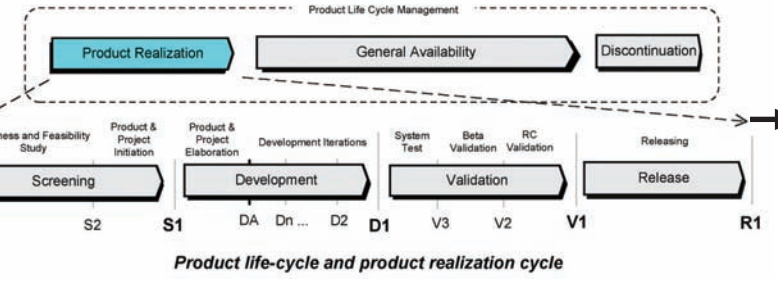
AXabier Larrucea
xabier.larrucea@esi.es

Teodora Bozheva
teodora.bozheva@esi.es

First company-wide agile R&D process established

THE STARTING POINT –MATURE DEVELOPMENT PROCESS

F-Secure's Product Realization Process was an established and mature way of developing successful products (Figure 1). The process was mainly based on Rational unified process and OMT++. The process type and milestones we used are most likely familiar to vast majority of software companies. While several attempts were made to achieve a truly iterative and incremental development, several problems were faced.



Product life-cycle and product realization cycle

Figure 1. F-Secure's old product development model

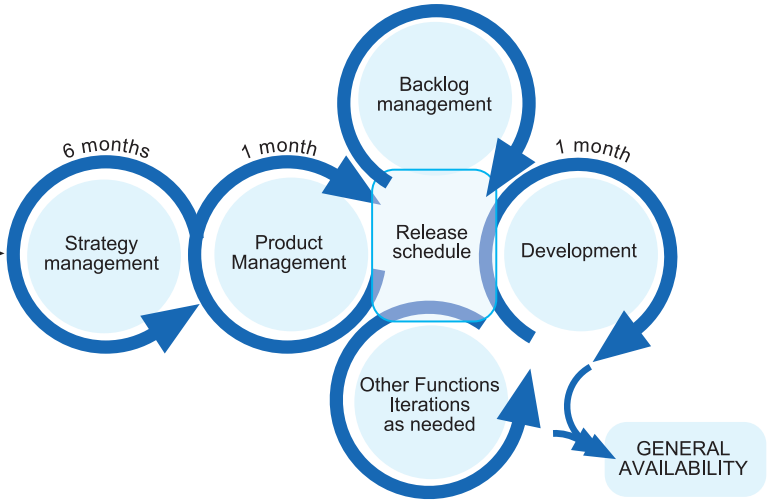


Figure 2. F-Lex –Agile development process framework

In the AGILE-ITEA project, F-Secure piloted extensively different agile methods and practices. The results of the pilots were impressive in both technical and business sense. A decision was made to truly renew the product development process. The new process called F-LEX (Figure 2) describes new processes and practices in F-Secure product life cycle management and their interfaces to business operations and processes.

F-LEX is targeted for the whole corporation, outlining the processes and practices, which are commonly agreed and followed in each organization and project. F-LEX is based on Incremental & Iterative and Agile models such as Scrum, Evolutionary development, Crystal, Extreme Programming and Mobile-D™.

The new process is highly disciplined. Deviating from the agreed practices is not allowed without a clearly stated reason. The new process is continuously being developed. Anyone can add change requests to the F-LEX backlog.

The change takes time and requires high level commitment from all organizational levels. The business benefits are clear, however. F-Secure continues to extend the process to all other business areas.

Principal problems in the old model

- Frozen requirements changed continuously
- Heavy planning and documentation throughout the development
- Workload for change management grew exponentially towards the end of the project
- 90% done syndrome
- Validation phase was always much longer than planned despite of vast automation
- Culture: relaying on individuals who are high on rank on organisational hierarchy

New process model: Benefits gained

- Working culture is changing
- projects are guided by strategy, resources and schedule are fixed, the scope is not
- much better visibility to strategy, operations and projects
- Our customers are truly involved

New process model: Challenges yet to overcome

- Multi-site, large projects
- Tool support /especially product and project management
- Having a shippable release after each iteration
- Multi-projects and multi-products synchronisation
- Enabling innovation, innovation management

Jari Still
jari.still@f-secure.com

Agility on the way to CMMI

Problem:

Over the past years, the Capability Maturity Model®'ae (CMM) and Capability Maturity Model®'ae Integration (CMMISM) have been widely used to assess organizational maturity and process capability around the world . Many organizations are now used to regular CMMI assessments and appraisals. They have confidence in CMMI because of its extensive descriptions of how the various good practices fit together .

Most recently, agile software development methods, practices and techniques have aroused widespread interest from software development organizations. The methods have promised to increase customer satisfaction [4, 5], to produce high quality software and to accelerate development times especially in organizations working in dynamic markets. Currently, companies wishing to utilize CMMI as a tool for the continuous improvement of their processes are also deploying agile methods. However, the question "To what extent CMMI admits practices for agile based software development?" is still open and is case dependent. Thus, mechanisms are needed about how to continuously evaluate the success of agile software development projects in the light of CMMI.

Solution:

In the AGILE project, a number of projects in F-Secure, Hantro, Nokia and Nemetschek were evaluated in assessments in which the goal was to support the adaptation of agile principles and methods by providing results useful for starting agile based improvement efforts. In these cases, mappings between CMMI specific goals and agile practices were used as a central tool in the assessments. These mappings can provide a roadmap for achieving a level of CMMI maturity and ability to stay "agile" on the way. Examples of the mapping are given in Table 1.

Table 1. CMMI goals and agile practices

CMMI specific goal	Scrum Practices	XP Practices	Mobile-D™'99 practices
Manage requirements	Product and Sprint backlogs; Sprint planning, Sprint reviews; Self-organizing teams	Real customer involvement, Stories	Product backlogs; Stories; Planning days; Release days
Establish estimates	Sprint Planning; Tasks and effort estimations for 1- to 4-week releases	Stories	Planning Days; Task cards with estimates of effort on information radiator
Develop a Project Plan	Sprint Planning; Product backlog; Sprint backlog	Quarterly cycle, weekly cycle, Incremental deployment, Real customer involvement	Planning days; Information radiator; Product backlog
Obtain commitment to the Plan	Sprint planning, Sprint Review; Self-organizing teams;	Real customer involvement, Whole team	Planning days, Task cards on information radiator; Self-organizing teams; Release day

In the studied cases, CMMI was found to be useful in ensuring that all focal software development viewpoints were taken into account in the assessment, whereas mappings were found to clarify the connections between the agile and plan-driven processes and thus to ease both the analysis and understanding of the assessment results. However, it also became clear that agile practices cannot fully cover the requirements for a formal CMMI assessment for the following reasons:

- During a formal (SCAMPI) CMMI Assessment a number of direct artefacts (documents) need to be presented to demonstrate the implementation of the specific and generic practices of CMMI. At the same time most of the Agile practices do not "encourage" the creation and storage of such documentation.
- While most of the specific practices of CMMI for the process areas of ML2 (and some from ML3) can be mapped to corresponding agile practices, this rarely can be done for the generic CMMI practices.

Further Research:

The assessment results have subsequently been used in improvement initiatives in which the case organizations' processes have shifted towards agility. It is concluded that the proposed approach for assessing agile software development using mapping produces useful results for starting agile-based improvement efforts which can lead to achieving a level of CMMI maturity with the introduction of some "non-agile" practices required by the standard. In the future, the study of assessment in the agile context will continue by addressing the needs of the different maturity levels of CMMI and how they can be achieved with maximum agility in the software development process.

Minna Pikkarainen, minna.pikkarainen@vtt.fi
Youri Metschev, ymetchev@nemetschek.bg
Boyan Angelov,BAngelov@nemetschek.bg
Annukka Mäntyniemi, annukka.mantyniemi@vtt.fi

Agile Methods and Open Source Software

Agile Methods (AMs) are a very recent development, but many of their basic principles have been around for a while and were inherited from the lean production pioneered in the '60s at Toyota for the production of cars. Moreover, many practices on which AMs are based have a long tradition in software development. For instance, unit testing has been used since the '60s. However, one of the major achievements of AMs is the integration of all these well-established principles and practices with some other more recent ones, such as pair programming.

The Open Source (OS) movement has a long tradition as well. However, it came into being as a way of licensing software not as a development method. Moreover, people producing OS software use a wide range of different approaches for software development. Even if it is not possible to define an OS development method, there are some basic principles and approaches that have become common in several OS communities.

Surprisingly, or maybe not, there are many basic principles and development techniques that are similar in AMs and OS Development (OSD). Three of the four principles of AMs are completely embraced by OSD:

- Working software over comprehensive documentation: documents, other than the source code, are reduced to a minimum to minimize the huge effort required to keep them updated and, in the event of misalignment all the effort spent in producing them is wasted. Important documentation can be generated on the fly using reverse engineering tools able to extract useful information from the source code and from the developers' written comments.
- Responding to change over following a plan: the software environment is highly dynamic and the ability to adapt the code to new needs is of paramount importance for success.
- Individuals and interaction over processes and tools: individuals are the main asset for any software project and their abilities deeply affect the result. The main idea is that a small team with high quality people using a 'light' process can perform better than a large team with average developers controlled through a 'heavy' process.

The last principle (customer collaboration versus contract negotiation) can be applied to OSD only partially, since it is not common for OS developers to deal with contracts. Moreover, it is very frequent that the customer and the developer are in agreement with each other.

However, there are some big differences in some approaches to development. For instance, most AMs involve co-localized teams, while OSD is rarely carried out in this way. One of the strongest features of OS is the distributed approach. This single feature greatly affects the organization of the work and the communication among team members. For this reason, the usage of communication tools is a key factor for success. Both synchronous and asynchronous tools are used, for instance instant messaging, forum, mailing lists, etc. Whereas in agile teams, the communication is completely different: usually, it is direct without any tools.

The analysis of commonalities and differences between AMs and OSD is in the early stages, but it is interesting to understand how some development approaches have evolved over time and whether they produce actual benefits in terms of software quality and customer satisfaction.

Alberto Sillitti, Alberto.Sillitti@unibz.it
Giancarlo Succi, Giancarlo.Succi@unibz.it

Introducing Agility in the Avionics Software World

The K.U.Leuven university joined forces with avionics equipment supplier Barco to take a look at how XP and other agile practices can improve a software process for the development of avionics software.

K.U.Leuven and Barco found out how agility can still be used in the stringent mission-critical domain, identifying 'safe spots' to introduce agile practices.

The success of agile software development is creating a pressure for application domains with less flexible software development processes.

The avionics software industry is confronted with late requirements changes and asked to provide shorten release cycles. While agile processes seem an obvious solution to deal with these demands, at the same time people are cautioned and advised to consider a more disciplinary approach for the development of mission-critical software. Fortunately for the avionics software world, agile improvements can be made while still respecting the major certification guidelines (known as RTCA DO-178B).

AGILITY ANALYSIS

Avionics guidelines do not impose a specific software development life-cycle process. Rather, they require delivery of multiple documents and records to verify traceability and testing of all requirements. Tools and reused software do not escape from these guidelines.

The bottlenecks that we can alleviate with agile techniquesare presented in Figure 1. Agile processes such as XP aim for an ideal, flattened curve, allowing a constant development pace. At the beginning of a project, certification driven software development follows these curves. We call this the software phase of the project.

A first divergence can be seen in the figure when the software is prepared to get tested in the field. Here, the process slows down because of hardware dependencies and acceptance testing. We call this the embedded phase. An even more significant slowdown is encountered when the software is ready to be certified. In this stadium, that we call certification phase, the software is presumed bug-free, but much documentation and manual testing is needed to provide certification artifacts. We found we could considerably flatten the steep curve of a regular DO-178B driven process. However, changing existing avionics software impose a difficult to handle certification overhead in the form of traceability management and manually supervised testing.

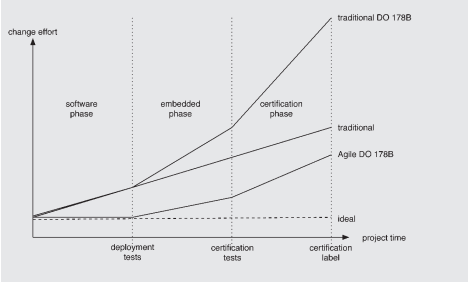
LESSONS LEARNED

Software phase: Communicate regularly and early in the development process and deliver incrementally functional prototypes;

Embedded phase: Improve communication with non-software people by applying project feedback based practices, such as daily meetings and post-iteration workshops;

Certification phase: Minimize the amount of changes and avoid refactoring. Treat certification documents like source code. Auto- generate documents or use an agile document preparation practice. Use pair programming and collective code ownership for independent reviewing of code.

By understanding the limitations of the development phases, software development can be made agile while still respecting strict certification guidelines. However, certification authorities need to acknowledge that agile software development can yield software that is at least as safe as before. Providing the authorities with evidence of this remains a task for the industry.



More information on this subject can be found in:
Wils, A., Van Baelen, S., Holvoet, T., De Vlaminck, K., Agility in the Avionics Software World, XP'2006, 17-22 June 2006, Oulu, Finland

KATHOLIEKE UNIVERSITEIT LEUVEN

Andrew Wils
Andrew.Wils@cs.kuleuven.be

Stefan Van Baelen
Stefan.VanBaelen@cs.kuleuven.be

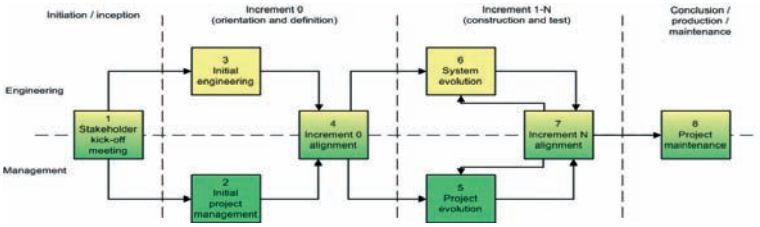
BARCO Visibly yours

Stijn Rammeloo
Stijn.Rammeloo@barco.com

Agile project execution tooling

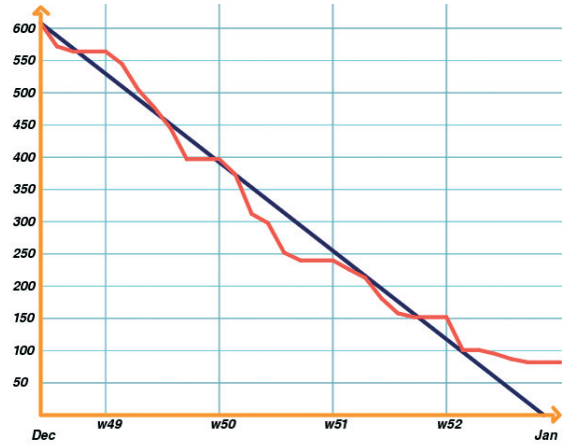
Doing agile with tools?

Most traditional projects depend on multiple complex and expensive tools to help execute the project. Although the aim would be to avoid using these tools in an agile project, it would be optimal to have a simple lightweight tool to guide the user in the agile way of working and store just enough information for all the stakeholders. At Philips Applied Technologies we created a tool just for that, called the Project Assist Tool (PAT).



Process information

PAT also has the ability to follow a described process. The figure below shows the overall agile process followed within Applied Technologies. The tool allows you browse through the process and track the steps taken within a project.



Features

The project assist tool was created for agile projects. It features:

- An agile process guide (the process steps, progression in the process)
- Project information (members, FTE, duration, budget, orders, risks, stakeholders etc.)
- Backlog (define and manage stories, prioritize , add business value, effort estimations, etc.)
- Increments (manage increments and increment plans, activities done for a story)
- Activities (working on stories, estimated hours, hours left, hours spent, owners, etc.)
- Project execution and tracking (budget left, rolling forecast , expenses, burn down graphs, etc.)
- Generation of information (progress reports, orders, budget left, etc.)
- WIKIs (project WIKIs for use within project but readable to anyone , general WIKIs, process WIKIs, etc.)
- Storing data for your project (reflection information, goals, etc.)

Burn down

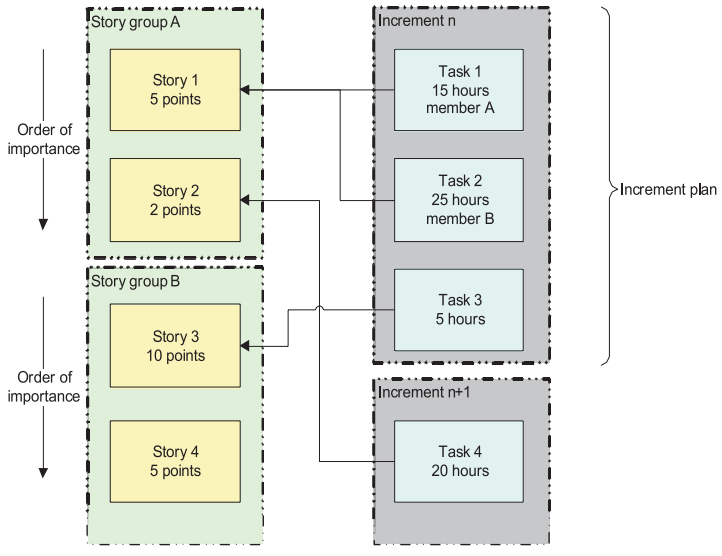
With PAT you can monitor your increment progress as well as your overall project progress via burn down charts. For an increment burn down the X axis is the increment duration and the Y-axis is the maximum amount of effort you can do taking the current FTE"d5s into account. The blue line is the ideal one and the orange line the actual effort based on estimated "d4to-go-hours"d5

Information radiation

The PAT tool can be seen as a project information radiator. It shows information for all stakeholders in the form of management information, requirements, deadlines, dependencies, what people are working at the moment and more. With the help of WIKI information is even shared between projects and to everyone that is interested.

PAT Planning and tracking basics

Within PAT you estimate your effort by means of stories which form the basic product requirements. These stories are given points based on their importance (business value). The most important stories are implemented in the first couple of increments. Activities are created in an increment to implement a story (as demonstrated in the figure below). The total amount of activities that fit in an increment form the increment plan. The assigned members can fill in their hours spent and what they think they still need for finishing the activity on a daily basis. From this information the burn down (as shown on the left) is generated.



PHILIPS

More information
Peter.Boon@philips.com

AGILE

Facts:

AGILE- Agile development of embedded systems

21 partners, 9 countries, 178 person years

1.4.2004 - 31.12.2006

http://www.agile-itea.org

Project coordinator: Pekka Abrahamsson, VTT (pekka.abrahamsson@vtt.fi)