Advances in Streamlining Software Delivery on the Web and its Relations to Embedded Systems

Kasper Hirvikoski

Master's thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, February 22, 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Kasper Hirvikoski

Työn nimi — Arbetets titel — Title

Advances in Streamlining Software Delivery on the Web and its Relations to Embedded Systems

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	February 22, 2015	12

Tiivistelmä — Referat — Abstract

Abstract.

ACM Computing Classification System (CCS):

Avainsanat — Nyckelord — Keywords
keyword

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

Software delivery on the web has over the years evolved into a rather established process. A software is developed iteratively through multiple stages, which ensure the user's requirements and the quality of the product or service. These stages form what is called the deployment pipeline [Fow06, HF11, Fow13a, Fow13b].

The deployment pipeline nowadays usually consists of at least three stages: development, staging and production. Organisations alter these stages depending on their size and needs. Using modern iterative and incremental processes, a software is developed feature-by-feature by iterating through these stages. Development starts in the development stage where developers build the feature requested by the customer or user. The feature is then tested in the staging phase, which represents the production setting. When the feature has been validated, it is then deployed to production. If necessary, each stage can be repeated until the feature is accepted. The stages are short and features are deployed frequently — in some cases even multiple times a day [O'R11, Sny13, Rub14].

Software engineering consists of various different processes and practises for ensuring the quality of the product or service — nowadays more or less based on Agile and Lean ideologies and practises [Ōno88, BBvB+01a, Fow05, Mon12]. At the low level, developers use source code management to keep track of changes to the software and to collaborate with other team members. To reinforce that the features work as intended, developers write automated test cases. Teams can also use more social methods — such as reviewing each other's code — to validate the implementations. These practises form the basis for Continuous Integration and Continuous Deployment [Fow06, HF11, Fow13a, Fow13b]. Software changes are frequently integrated, tested and deployed — automatically in each stage. The first two form Continuous Integration and the latter Continuous Deployment. If any stage fails, the process starts from the beginning.

The web enables the use of the deployment pipeline and its practises in an unprecedented way [KLSH09]. Due to the distributed nature of the web, software can be deployed as needed and the user always sees the newest version without the need of any interaction. This eases the use of many cutting-edge methods [KLSH09]. Deploying software as needed has allowed developers to experiment with different implementations of a feature. These changes can target anything from a more optimised algorithm to something more user-faced, such as improvements to the user experience of a product [KLSH09]. These practises have started to formalise as Continuous Experimentation [FGMM14].

Not all software can be developed easily this way. Many embedded systems, which have a dedicated function within a larger mechanical or electrical system, require hardware to accompany the software. This presents a variety

of challenges to overcome. Hardware can require thorough planning and iterating can take time. Contexts such as cross-platform support, robotics, aerospace and other embedded systems pose interesting cases. Many of these contexts can at a glance seem regarded as models for more traditional sequential software engineering processes with heavy planning, documentation and long development phases. However, even NASA's earlier missions have iterated on the successes and failures of previous ones. Even though it can be more difficult, hardware can be build and tested iteratively with new approaches such as prototyping and 3D-printing.

This raises an interesting research topic — *presenting the advances in streamlining software delivery on the web and relating its practises and their advantages and challenges to embedded systems.* Using case studies to identify which Agile and Lean practises are used, how they could be improved and how new practises could be incorporated to these settings. Moreover, the aim is to identify which modern Continuous Integration, Delivery and Experimentation practises are used. Can we determine how they compare to the way the web is utilised as a platform?

The hypothesis is that there should be no reason why these practises could not be successfully used and cleverly adapted to hardware settings. My research method for this thesis was reviewing the current practises in literature and industry. I also conducted several semi-structured interviews with the industry working on leading embedded systems to get a view on if and how the deployment pipeline has changed the development of hardware related products.

This thesis is structured into seven chapters. Following the introduction, Chapter 2 outlines how software delivery has progressed from a structureless process following a code-and-fix mentality to what is now considered the leading edge of iterative development. This sets the scene for understanding the logic behind being adaptive to change and how the user is an essential part of the process. Chapter 3 describes how software delivery has embraced primarily a three staged pipeline for deploying new features to the user. This deployment pipeline consists of a development, staging and a production stage. Chapter 4 discusses how the web has provided an effective platform for the deployment pipeline by streamlining and automating many of the practised used by modern development. Chapter 5 delves into the challenges related to delivering software that is firmly linked to hardware. It deliberates about how the deployment pipeline could be integrated into these embedded systems. Chapter 6 presents the results from the interviews I conducted from the field. The idea is to incite discussion — through the view of people working on the embedded field — about what is the current state of software delivery in these settings and how it could be improved. Finally Chapter 7 concludes this work by making some conclusions about the gathered knowledge.

2 Software Delivery

Software development has changed notably in the past few decades. Going back, it was not until 1968, when the term software engineering was introduced by the NATO Science Committee [NR69]. By that time, it was considered that software development had drifted into a crisis, where a wider gap was forming between the objectives and end-results of software projects. Additionally, it was getting increasingly difficult to plan the cost of development. A collective effort was put in place to establish a more formalised method for software development — similar to traditional engineering. It was considered necessary that the foundation for delivering software should be more theoretical with laid principles and practises [NR69]. By 1969, the term software engineering had become well-established [BR70].

Software development processes began to form. Notably in 1970, Winston W. Royce published a paper that described a formal approach for sequentially developing a software based on previously used practises [Roy70]. It was only later named as the waterfall model [Boe88, LB03]. The process consists of multiple stages that should be carried after the previous has been reviewed and verified. See figure 1. It begins by mapping the requirements for the entire software, then proceeding to designing the architecture, followed by implementing the plan, verifying the result is according to the set requirements, and finally maintaining the product [Roy70]. Each stage is planned and documented thoroughly. However, contrary to what has been referred, Royce presented the model as a flawed, non-working model [Roy70]. If any of the stages fail, serious reconsideration of the plan or implementation might be necessary. Therefore sequentially following the stages would not produce what was intended and inevitably previous stages would need to be revisited [Roy70]. Nevertheless, this was overlooked and the waterfall model became the dominant software development process for software standards in government and industry for the time-being [Boe88, LB03].

Waterfall-oriented models are considered heavy. The waterfall model has been criticised as too controlled and managed, documentation-oriented and over-incremental [Boe88, LB03]. More lightweight iterative processes were proposed as opponents for incremental software development in the later part of the nineteen hundreds [LB03]. In fact, early applications of iterative and incremental development dates as far back as the mid-1950s [LB03]. Fast-forward to 2001, when a group of software developers met to discuss new lightweight development principles. As the result of these discussions, a manifesto for Agile software development was published [BBvB$^+$01a]. Four principles were proposed for Agile software development: focusing on individuals and interactions over processes and tools, focusing on working software over comprehensive documentation, focusing on customer collaboration over contract negotiation and responding to change over following a plan. The manifesto does not dismiss the value of the latter, but considers the former

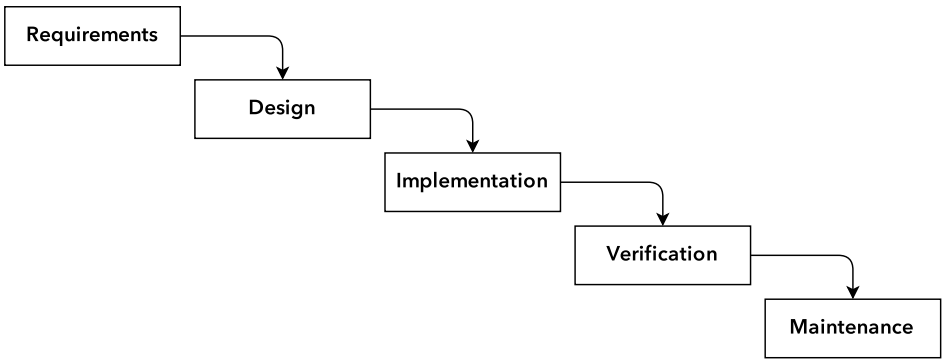

Figure 1: Waterfall Model

more valuable [BBvB+01a]. Iterative processes started to gain mainstream traction [LB03]. Software development was considered as an ongoing process, where a product should be build in small increments, iteratively going through the development stages. Instead of planning, designing and implementing the whole software, the software should be build iteratively by repeating all of these steps in shorter more manageable parts. See figure 2. Hence, any issue or miss-communication could be discovered early and fixed accordingly.

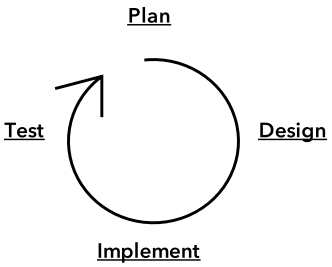

Figure 2: Iterative Development

2.1 Adapting to Requirements

The demands for software products are continuously shifting. It is not always obvious what the users want. In some cases, users do not know what they are looking for, until you show them what they need. An average client has little knowledge on how software products work or how they are built. Therefor, it

is exceedingly difficult for a client to map specifically what they require from a software product. In most cases, rigorously planning a software beforehand will not work. Agile development tries to create a framework, where processes and practises can take these requirements into consideration.

Prominently, being "agile" means effectively responding to change. These course corrections are rapid and adaptive. The highest priority is to satisfy the customer trough continuously delivering valuable software from early on [BBvB+01b]. Software should be delivered frequently in short increments. These increments, also referred as iterations in Agile development, should take no more than a couple of weeks to a couple of months — the shorter the better. Throughout the project, teams respond to change by having effective communication among all stakeholders for the product daily. The best means for conveying information is face-to-face conversation [BBvB+01b]. A stakeholder represents the views for the users or clients. By taking the stakeholders as part of the team, developers can react when something is not working as intended. Studies show that developers see the ongoing presence of stakeholders helpful for development [DD08]. An Agile process is driven by the customers descriptions of what is required [BBvB+01b]. These requirements may be short-lived and that must be kept in focus. Changes are unavoidable. Users' desires evolve and this must be harnessed to the customer's competitive advantage [BBvB+01b]. It is not uncommon for requirements to change even late in development.

One key premiss for Agile development is to reduce the burden of the process. Working software is the primary measure of progress [BBvB+01b]. A process should not hinder the work of a team — on the contrary it should permit the team to function to its full extent. By organising the team to be in control of the process, the framework facilitates rapid and incremental delivery of software. Projects should be based on motivated individuals [BBvB+01b]. Motivation is maintained by creating a constructive environment and giving the necessary support when needed. Trusting the team is of the utmost importance [BBvB+01b].

2.2 Ensuring Quality

Assuring quality is not an easy task. ISO 9000-standard defines quality as the extent of how well the characteristics of a product or service fulfil all of the requirements, the needs and expectations, set by the stakeholders [ISO05]. IEEE defines software quality as the degree to which a system, component or process meets the specified requirements as well as the customer's and user's needs or expectations [IEE06]. Both definitions focus strongly on fulfilling the user's needs.

Software development is challenging. Users perceive quality as working software, but most of all emphasising good technical design and implementation makes the development process easier. People, time and money are

limiting factors for ensuring quality. Strict deadlines and scarce resources have direct effects. Furthermore, human factors play a considerable role. Several empirical studies reinforce the significance of Agile development processes and practises as improving quality in software [SS10]. Evidently being "agile" should in the long term make development more predictable and eventually lead to shorter development times and minimised costs. This provides an environment for being adaptive.

In addition to focusing on satisfying the customers needs, Agile development promotes continuous attention on technical excellence and good design practises [BBvB+01b]. Even so, this should not be accomplished by hindering simplicity. Simplicity maximises the amount of work that can be accomplished. The Agile Manifesto states that the best architectures, requirements and designs emerge from self-organising teams [BBvB+01b]. After regular intervals, the team members reflect on how they have performed and how they can become more effective. This is how the team can then tune and adjust its behaviour appropriately.

2.3 Processes and Practises

Processes and practises assist the development process. They create the framework and guidelines within a team can develop a suitable environment to deliver software [Kni07]. They also help to maintain quality. Modern development strongly focuses on Agile and Lean ideologies and practises [Ōno88, BBvB+01a, Fow05, Mon12].

At the low level, developers use source code management to keep track of changes to the software and to collaborate with other team members. Source code management enables multiple developers to work on a single project, while also creating a history for the entire project. When a problem arises, developers can go back in time to look at the source code at any given point in time. To ensure features work as intended, developers use automated test cases to verify expected behaviour. There is a clear correlation between higher test coverage resulting in fewer errors in software [MNDT09]. Tested code has a better change of detecting errors than untested code. Teams can also use more social methods — such as reviewing each other's code — to validate the implementations. Pair programming, coding dojos and hackathons provide tools for improving skills and solving complex problems together.

Agile development only provides a framework for software delivery. It does not specify concretely how development should be organised. Most notably, Scrum and Extreme Programming (XP) have created a structure for Agile development [LB03, SS10]. Scrum provides a framework for managing development. It focuses on how development should be planned, managed and scheduled. It does not provide any strict practises, instead it gives guidelines for how customer requirements should be discovered, prioritised,

and how the development of these features is split into iterations.

Scrum has been strengthened with practises such as test-driven development (TDD) and pair programming. These are defined in XP. In test-driven development, features are developed by writing the expectations for a feature as tests before actually implementing the code. In pair programming, developers develop features in pairs.

2.4 From Agile to Lean

As time has elapsed, developers have simplified software delivery even more. Agile has turned into Lean. Being "lean" means reducing the amount of "waste" around software development. Iterations have turned into building single features at a time. Instead of building a frame for a car, a development process should essentially start with building a bicycle first. To evaluate an idea, developers should begin by developing a minimum viable product (MVP) to validate the implementation has value. Being "adaptive", has transformed into quantitatively assessing what effects changes have. This build-measure-learn cycle has transformed how features are developed and validated. See figure 3.

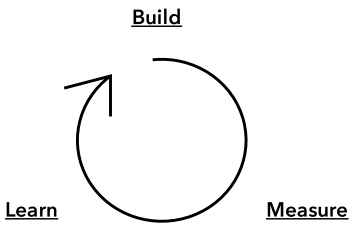

Figure 3: Build-Measure-Learn Cycle

3 Deployment Pipeline

Deployment pipeline.

3.1 Development

Development.

3.2 Staging

Staging.

3.3 Production

Production.

4 Using Web as a Platform

Using Web as a platform.

4.1 Continuous Integration

Continuous Integration.

4.2 Continuous Deployment

Continuous Deployment.

4.3 Continuous Experimentation

Continuous Experimentation.

5 Towards Embedded Systems

Towards embedded systems.

5.1 Using Hardware as a Platform

Using hardware as a platform.

5.2 Adapting for Deployment Pipeline

Adapting for deployment pipeline.

6 Cases from Embedded Settings

Software is considered “soft”, hardware “hard”. It is not always obvious how products or features that combine software with hardware can be developed piece-by-piece. This combination, usually referred as an embedded system, provides challenges in being agile and adaptive. I conducted several semi-structured interviews with the industry working on leading embedded systems to get a view on if and how the deployment pipeline has changed the development of hardware related products? The interview consisted of the following topics:

Process

1. Do you consider that your organisation follows the principles and practises of Agile and Lean development?
2. If so, has this recently changed the way you develop products or features into production?
3. Do you approach development from the point-of-view of the whole product or by single features?
4. Please describe the process behind developing an idea into a single feature. How long does it take?
5. Do you recognise distinct development, staging and production environments in your process?
6. If so, are these automated?

Adapting to Change

7. How easy or hard is it to adapt changes in hardware related products?
8. Can you deploy software changes automatically or even remotely?
9. How do you keep software and hardware development in sync?
10. How short iterations do you use to adjust for feedback from your stakeholders?

Experimentation

11. Do you have an automated process for deploying or experimenting a feature?
12. How do you experiment with software related to hardware?
13. How do you value an idea (prototypes, minimum-viable products, A/B testing)?
14. Related to hardware, has new approaches such as electronic testing platforms, 3D-printing or laser-cutting changed your process?

7 Conclusions

Conclusions.

References

[BBvB+01a] Beck, Kent, Beedle, Mike, Bennekum, Arie van, Cockburn, Alistair, Cunningham, Ward, Fowler, Martin, Grenning, James, Highsmith, Jim, Hunt, Andrew, Jeffries, Ron, Kern, Jon, Marick, Brian, Martin, Robert C., Mellor, Steve, Schwaber, Ken, Sutherland, Jeff, and Thomas, Dave: *Manifesto for Agile software development*, 2001. `http://agilemanifesto.org`, (accessed 13 January 2015).

[BBvB+01b] Beck, Kent, Beedle, Mike, Bennekum, Arie van, Cockburn, Alistair, Cunningham, Ward, Fowler, Martin, Grenning, James, Highsmith, Jim, Hunt, Andrew, Jeffries, Ron, Kern, Jon, Marick, Brian, Martin, Robert C., Mellor, Steve, Schwaber, Ken, Sutherland, Jeff, and Thomas, Dave: *Principles behind the Agile Manifesto*, 2001. `http://agilemanifesto.org/principles.html`, (accessed 10 February 2015).

[Boe88] Boehm, Barry W.: *A spiral model of software development and enhancement*. Computer, 21(5):61–72, May 1988, ISSN 00189162. `http://dx.doi.org/10.1109/2.59`.

[BR70] Buxton, John N. and Randell, Brian (editors): *Software Engineering Techniques: Report of a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27–31 Oct. 1969, Brussels, Scientific Affairs Division, NATO*. 1970.

[DD08] Dybå, Tore and Dingsør, Torgeir: *Empirical studies of agile software development: A systematic review*. Information and Software Technology, 50(9–10):833–859, 2008, ISSN 09505849. `http://dx.doi.org/10.1016/j.infsof.2008.01.006`.

[FGMM14] Fagerholm, Fabian, Guinea, Alejandro Sanchez, Mäenpää, Hanna, and Münch, Jürgen: *Building blocks for continuous experimentation*. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, RCoSE 2014, pages 26–35, New York, NY, USA, 2014. ACM, ISBN 9781450328562. `http://doi.acm.org/10.1145/2593812.2593816`.

[Fow05] Fowler, Martin: *The new methodology*, 2005. `http://martinfowler.com/articles/newMethodology.html`, (accessed 13 January 2015).

[Fow06] Fowler, Martin: *Continuous integration*, 2006. `http://martinfowler.com/articles/continuousIntegration.html`, (accessed 13 January 2015).

[Fow13a] Fowler, Martin: *ContinuousDelivery*, 2013. http://martinfowler.com/bliki/ContinuousDelivery.html, (accessed 13 January 2015).

[Fow13b] Fowler, Martin: *DeploymentPipeline*, 2013. http://martinfowler.com/bliki/DeploymentPipeline.html, (accessed 13 January 2015).

[HF11] Humble, Jez and Farley, David: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. The Addison-Wesley Signature Series (Fowler). Addison-Wesley, 2011, ISBN 9780321601919.

[IEE06] IEEE: *IEEE standard for developing a software project life cycle process*. IEEE Std 1074-2006 (Revision of IEEE Std 1074-1997), 2006. http://dx.doi.org/10.1109/IEEESTD.2006.219190.

[ISO05] ISO: *Quality management systems — fundamentals and vocabulary*. 2005. http://iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=42180.

[KLSH09] Kohavi, Ron, Longbotham, Roger, Sommerfield, Dan, and Henne, Randal M.: *Controlled experiments on the web: survey and practical guide*. Data Mining and Knowledge Discovery, 18(1):140–181, 2009, ISSN 13845810. http://dx.doi.org/10.1007/s10618-008-0114-1.

[Kni07] Kniberg, Henrik: *Scrum and XP from the Trenches*. InfoQ, 2007, ISBN 9781430322641.

[LB03] Larman, Craig and Basili, Victor R.: *Iterative and incremental developments: a brief history*. Computer, 36(6):47–56, June 2003, ISSN 00189162. http://dx.doi.org/10.1109/MC.2003.1204375.

[MNDT09] Mockus, Audris, Nagappan, Nachiappan, and Dinh-Trong, Trung T.: *Test coverage and post-verification defects: A multiple case study*. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 291–301, October 2009. http://dx.doi.org/10.1109/ESEM.2009.5315981.

[Mon12] Monden, Yasuhiro: *Toyota Production System: An Integrated Approach to Just-In-Time, 4th edn.* CRC Press, 2012, ISBN 9781439820971.

[NR69] Naur, Peter and Randell, Brian (editors): *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*. 1969.

[Ōno88] Ōno, Taiichi: *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1988, ISBN 9780915299140.

[O'R11] O'Reilly: *Velocity 2011: Jon Jenkins, "velocity culture"*. YouTube, 2011. `https://youtube.com/watch?v=dxk8b9rSKOo`, (accessed 13 January 2015).

[Roy70] Royce, Winston W.: *Managing the development of large software systems*. In *Proceedings of IEEE WESCON*, 1970.

[Rub14] RubyKaigi: *Continuous delivery at GitHub - RubyKaigi 2014*. YouTube, 2014. `https://youtube.com/watch?v=Rhvri5cozTc`, (accessed 13 January 2015).

[Sny13] Snyder, Ross: *Continuous deployment at Etsy: A tale of two approaches*, 2013. `http://slideshare.net/beamrider9/continuous-deployment-at-etsy-a-tale-of-two-approaches/`, (accessed 13 January 2015).

[SS10] Sfetsos, Panagiotis and Stamelos, Ioannis: *Empirical studies on quality in agile practices: A systematic literature review*. In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pages 44–53, September 2010. `http://dx.doi.org/10.1109/QUATIC.2010.17`.