DD2482 Evolution of DevOps

Per Arn (perarn@kth.se), Tobias Gabi Goobar (tobiasgg@kth.se)

May 2022

1 Introduction

Software Development is an ever changing practice. During its inception in the middle of the 20th century, programming was relatively weak and had few uses as well as quite troublesome to implement. Submitting code on punch cards only to receive a syntax error 3 days later was hardly feasible. As the industry grew, the need for efficiency did the same. This spawned ideas such as Agile Development and DevOps, which are now considered by many to be state of the art. Although many problem areas now have efficient solutions, new practices and ideas continue to revolutionize the way of working. As the demand for fast paced development and lower costs continues to increase the pressure to continuously improve does not show any signs of stopping.

2 The Inception of Software

The first description of what should best be described as a computer was given by Charles Babbage in 1837 [5]. He described a machine which could perform arithmetic operations as well as many other operations which would be considered to be fundamental to programming such as looping and conditional branching [16]. The input to this computer was to be given via punch cards. Although this computer was at the time only theoretical, its structure and concepts would be used in the future. During the middle of the 1930's, Konrad Zuse created the first programmable electro magnetic calculator [24].

When general purpose computers started seeing the light of day in the late 40's and early-to-mid 50's, the popularity of programming grew. Initially, programmers would write their computer programs onto punch cards in assembly languages. However, this turned out to be far too tedious and errors were bound to happen far too often. Any mistakes on the punch card resulted in having to throw away the card and start from scratch on a new card [22]. Coupled with the scarcity of computers, this meant that programmers who sent their punch cards to be evaluated by a computer could have to wait several hours or even days for their sought computer response only to hear that there was a typo on their punch cards. This was not a feasible way of programming seeing how it

was highly inefficient [4].

A solution to this was to implement a way of communicating with the computer in a way that makes more sense to the programmer. In 1957 the higher level programming language Fortran originated. This programming language made it feasible for scientific and engineering personnel to delve into the area of programming and although it did not solve the issue with human errors in writing code, it reduced the errors frequency [18]. Following the rise of Fortran, higher level languages dominated the programming area during the 60's seeing how writing machine code or assembly languages shrunk in popularity. Prioritizing higher level languages over machine or assembly code has since more or less been a fact among programmers.

3 The IT-boom

With the introduction of the Personal Computer in the 1980s large companies started to form around the production of software and the industry grew immensely. Companies such as Microsoft, Apple and IBM started developing software at a large scale with lots of developers working on the same systems. The increased scale of development brought difficulties. As the programs became larger and more developers started to work on projects together the maintenance of the code became more cumbersome [28]. A great tool for collaboration that surfaced during this time was *Version Control*. A version control system (VCS) is a system that tracks changes made to files and simplifies the composition and management of different versions of software. One of the first version control systems was SCCS (Source Code Control System) which was released in 1975. This system paved the way for many more version control systems in the years to come, such as RCS (Revision Control System) in 1982, CVS (Concurrent Version Systems) in 1986 and finally Git in 2005 [15].

The slow development pace of software during this era was another large problem that plagued the industry. The development period for a software product was on average around three years. The slow pace of production caused many software projects to become obsolete even before being finished, and therefore being scrapped. This quandary was what became known as the Application Development Crisis [28].

One of the development methodologies that was used frequently during this time was the *Waterfall model*. The waterfall model consists of multiple phases, each cascading into the next. The phases are[1]

- **Specification/Requirements**

  Requirements are established together with the client.

- **Analysis**

  Models and schemas are created.

[1] There are several different versions of the waterfall model with slightly different phases.

- **Design**

  The system architecture is defined.

- **Development**

  The code is developed.

- **Testing**

  The code is tested.

- **Operations**

  The completed system is installed and maintained.

[6] [27]

The linear non-iterative nature of this model had several issues. Once requirements were established they could no longer be changed, this can be a serious issue since new ideas and realizations during the project from either the client's or the developers' side could not be incorporated. You essentially had to get it right the first time, which is generally quite uncommon. These problems spawned the idea of a more dynamic iterative software development methodology, this idea would later become known as *Agile Development* [6].

4 The Agile Revolution

It is of great importance to be efficient when it comes to developing software. To this end, several different methods of developing software have been proposed throughout the years. During the 90's, the software methods which dominated the industry were commonly heavyweight methods, meaning that they were very "regulated, planned and micromanaged" [21]. One example of such a method is the above mentioned Waterfall model. In response to this, lightweight methods were conceived, whose primary goal was to guide the development of software through looser restrictions that are far easier for the developer to follow [25].

In 2001, 17 software developers who all cherished the lightweight approach met and discussed their practices and how they were best followed. Their meeting resulted in the Manifesto for Agile Software Development which represents their values on lightweight software development and how agile software development should be conducted [7]. According to the manifesto, one should have the following values when developing software: Individuals and interactions over processes and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation, Responding to change over following a plan [12].

The manifesto also describes 12 principles which should be followed in software development, as can be seen in figure 1 beneath. Today, more and more companies follow the values and principles brought up in the Manifesto when developing software, which is a testimony to its importance and value [7].

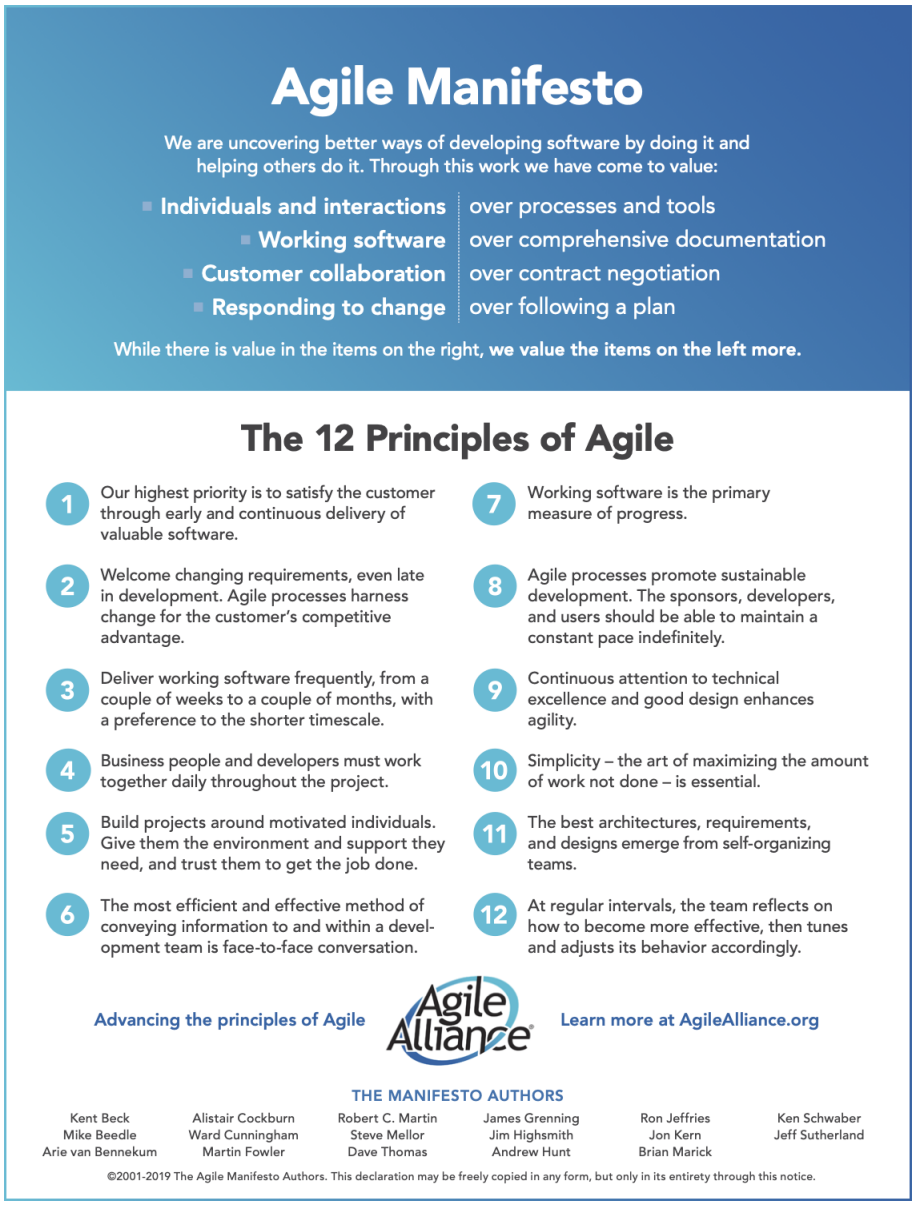

Figure 1: The 12 principles of Agile development [13].

5 DevOps

The term DevOps was coined in 2007 and essentially describes the principles and values one should aim towards following when developing software and performing IT operations [1]. Efficiency is a must when developing software and can be achieved in different ways, with DevOps being the common approach for many. DevOps aims to increase efficiency by minimizing the time spent in the life cycle of system development [26] and "provide continuous delivery with high software quality" [23]. Many of the core building blocks of DevOps originate from agile principles such as valuing continuous delivery integration, ensuring to always have a stable build which can be built further upon, stressing the importance of communication and having a neat culture for all stakeholders involved in the projects. A problem with agile software development, which, granted, did thrive after the Agile Revolution, was that although the software development teams were efficient, they lacked in communication with the IT operations teams. The solution to this was combining the agile software development with IT operations and thus DevOps originated [1].

DevSecOps is an extension of DevOps which aims to include the ever-growing important aspect of Computer Science that is security into software development. Considering how important security is nowadays with the society transforming more and more into digitalization, it makes sense that DevOps sought to include this in their practices. The idea is to incorporate security at all times during development of an artifact as opposed to adding security to an already finished artifact [29].

6 Current best practices

It has been more than 10 years since the concept of DevOps was first introduced to the software industry. The practices have matured and DevOps is now standard among most companies in the industry. The following concepts summarizes the current state of DevOps and can be considered best practices within the software world.

6.1 Version Control System (VCS)

A VCS is used to continuously develop, track changes and enable collaboration between developers. Branching is used heavily to distinguish different versions.

6.2 Continuous Integration (CI)

CI solves the problem of merging code from a large number of developers working on the same codebase at the same time. By committing multiple times per day and running automated tests and builds on every commit it is easy to merge different contributions and find the "breaking" commit in case of a failure. The high frequency of commits also avoids situations where a local branch has

diverged too much from the main branch and the merging of said branches becomes highly difficult ("merge hell"). [17]

6.3 Continuous Testing

Automated test are run on *only* those parts of the system affected by a commit. This allows for tests to be run on each commit even on a large system where running every single test would take too long. The tests consist of unit tests, integration tests, performance tests and end-to-end tests. [8]

6.4 Continuous Delivery/Deployment (CD)

CD consists of two parts, Continuous Delivery and Continuous Deployment. Together these two processes make sure that each change to the codebase is continuously deployed to the production environment, with each successful commit. Continuous Delivery automates the process of safely pushing changes to production and Continuous Deployment focuses on automating the actual deployment [9].

6.5 Continuous Security

In addition to testing, building and deploying, automated security checks are added in the CI/CD pipeline. Furthermore, developers are also trained in security [11].

6.6 Agile project management

The dynamic and iterative approach of agile allows for constant evolution of a software project. It also enables goal-alignment between developers, operations and customers[21].

7 Future of DevOps

Even though many problems within large-scale software development has been solved, there still is room for improvement. The software industry is dynamic and new practices and tools are introduced frequently. The following is a list of concepts that are starting to make their way into DevOps and will most likely shape the future of software.

7.1 AIOps & MLOps

A current trend that is making its way into software development and DevOps is the usage of Artificial Intelligence and Machine Learning to generate advanced analytics to boost the performance of the operations team [19]. Furthermore, it is likely that we will see the growth of AI-developer tools, such as GitHub Copilot, to accelerate the writing of code and tests [14].

7.2 Serverless

The concept of Serverless Computing involves deploying systems on external servers provided by vendors that are paid on an as-used basis. The main driving force behind switching to Serverless is to avoid the massive costs of having a server infrastructure and the risk of maintenance [3] [20].

7.3 Chaos Engineering

Chaos Engineering is a practice that involves experimenting on a system to improve confidence in the system's stability. By subjecting a system to chaotic behaviours, such as crashes and severed network connections, one can simulate critical events that can happen in a production environment. This can uncover weaknesses in a system that are otherwise not found by standard testing practices [2] [10].

References

[1] Atlassian. *DevOps principles*. URL: https://www.atlassian.com/devops/what-is-devops.

[2] Principles of Chaos. *Principles of Chaos Engineering*. Mar. 2019. URL: https://principlesofchaos.org.

[3] Cloudflare. *What is serverless computing?* URL: https://www.cloudflare.com/en-gb/learning/serverless/what-is-serverless/.

[4] Dale Fisk. *Programming with Punched Cards*. 2005. URL: http://www.columbia.edu/cu/computinghistory/fisk.pdf.

[5] John Graham-Cumming. *Let's build Babbage's ultimate mechanical computer*. Dec. 2010. URL: https://www.newscientist.com/article/mg20827915-500-lets-build-babbages-ultimate-mechanical-computer/?ignored=irrelevant.

[6] Growin. *A Brief History of Software Development Methodologies*. Dec. 2021. URL: https://www.growin.com/blog/history-of-software-development-methodologies/.

[7] Tomas Gustavsson. *Agil projektledning*. Sanoma utbildning.

[8] Tom Hall. *DevOps Best Practices*. URL: https://www.atlassian.com/devops/what-is-devops/devops-best-practices.

[9] Harness. *Continuous Delivery Vs. Continuous Deployment: What's The Difference?* Nov. 2021. URL: https://harness.io/blog/continuous-delivery/continuous-delivery-vs-continuous-deployment/.

[10] Ori Hoch. *A Look Into DevOps Future – Top DevOps Trends?* July 2021. URL: https://www.incredibuild.com/blog/a-look-into-devops-future-top-devops-trends.

[11] Marc Hornbeek. *Best of 2021 – Nine Pillars of DevOps Best Practices*. Dec. 2021. URL: https://devops.com/nine-pillars-of-devops-best-practices/.

[12] et al. Kent Beck Mike Beedle. *Manifesto for Agile Software Development*. 2001. URL: http://agilemanifesto.org.

[13] et al. Kent Beck Mike Beedle. *Principles behind the Agile Manifesto*. URL: https://www.agilealliance.org/wp-content/uploads/2019/09/agile-manifesto-download-2019.pdf.

[14] Gaurav Kumar. *What Does GitHub's Copilot Mean for the Future of the Software Industry?* Oct. 2021. URL: https://www.bairesdev.com/blog/what-githubs-copilot-mean-for-the-future/.

[15] Taryn McMillan. *A History of Version Control*. Sept. 2021. URL: https://blog.tarynmcmillan.com/a-history-of-version-control.

[16] Computer History Museum. *The Engines*. URL: https://www.computerhistory.org/babbage/engines/.

[17] Max Rehkopf. *What is continuous integration?* URL: https://www.atlassian.com/continuous-delivery/continuous-integration.

[18] Jean E. Sammet. "Programming Languages: History and Future". In: *Commun. ACM* 15.7 (July 1972), pp. 601–610. ISSN: 0001-0782. DOI: 10.1145/361454.361485. URL: https://doi.org/10.1145/361454.361485.

[19] ScienceLogic. *What is AIOps?* URL: https://sciencelogic.com/solutions/aiops.

[20] Veritis. *Future of DevOps: Top 6 DevOps Trends in 2022 and Beyond*. URL: https://www.veritis.com/blog/future-of-devops-top-6-devops-trends-in-2022-and-beyond/.

[21] Wikipedia. *Agile software development*. URL: https://en.wikipedia.org/wiki/Agile_software_development.

[22] Wikipedia. *Computer programming in the punched card era*. URL: https://en.wikipedia.org/wiki/Computer_programming_in_the_punched_card_era.

[23] Wikipedia. *DevOps*. URL: https://en.wikipedia.org/wiki/DevOps.

[24] Wikipedia. *Konrad Zuse*. URL: https://en.wikipedia.org/wiki/Konrad_Zuse.

[25] Wikipedia. *Lightweight methodology*. URL: https://en.wikipedia.org/wiki/Lightweight_methodology.

[26] Wikipedia. *Systems development life cycle*. URL: https://en.wikipedia.org/wiki/Systems_development_life_cycle.

[27] Wikipedia. *Waterfall model*. URL: https://en.wikipedia.org/wiki/Waterfall_model.

[28] Micah Yost. *A Brief History of Software Development*. Jan. 2018. URL: https://medium.com/@micahyost/a-brief-history-of-software-development-f67a6e6ddae0.

[29] Kev Zettler. *DevSecOps Tools*. URL: https://www.atlassian.com/devops/devops-tools/devsecops-tools.