DOI: 10.46541/978-86-7233-386-2_43

25th International Scientific Symposium
Strategic Management and Decision Support Systems
in Strategic Management

19th May, 2020, Subotica, Republic of Serbia

Zoran Dragičević
PhD student, University of Novi Sad, Faculty of Economics in Subotica, Serbia
zoran.dragicevic021@gmail.com

Saša Bošnjak
University of Novi Sad, Faculty of Economics in Subotica, Serbia
bsale@efs.uns.ac.rs

AGILE DEVELOPMENT PROCESS IN THE SOFTWARE FACTORY OF THE FUTURE

Abstract: The advent of the cloud and software ecosystems, the expansion of digital technologies and new business models, the digital transformation and the 4th Industrial Revolution bring new challenges to the implementation of agile software development processes, which create new needs for process and project management. These challenges are especially related to the development of complex, distributed software systems, their security, scaling, distribution and development environment, rapid delivery and maintenance. Agile software development processes emerged 20 years ago in response to the challenges and problems caused by the bureaucratic, predictive nature of traditional methodologies. Since then, they have had a tremendous impact on changing the way software is developed, due to a greater focus on people, fast iterations, direct communication and quick feedback. However, in order to remain agile in their application, agile development processes have had to transform and adapt to new challenges, with a rethinking of some of the fundamental aspects of agility. In this regard, the question arises of what the agile development process looks like in the software factory of the future. To answer this question, applying multiple case study and comparative analysis, this paper investigates, describes, and compares agile software development processes in some of the leading software companies, such as Google, Amazon, Facebook, and Twitter. The results of the research will contribute to a better understanding of the basic principles of agility and agile development process in the software factory of the future.

Key words: agility, agile development process, software factory of the future

INTRODUCTION

The emergence of the cloud and software ecosystems, the expansion of digital technologies and new business models, the digital transformation and the 4th Industrial Revolution bring new challenges to the implementation of agile software development processes, which create new needs for process and project management (Kruchten, 2019). These challenges are particularly related to the development of complex, distributed software systems, security, scaling, distribution of the development and production environment, rapid delivery and maintenance. The processes of digital disruption and transformation are changing the way software projects are run and implemented. In the digital age, the measure of success is not delivering software on time, within budget, with requirements from a year ago, but delivering value in the eyes of the customers, improved service, learning and revenue. This requires generating ideas, selecting the most promising ideas, and finding the shortest path that will confirm their value. In this regard, agile software development is no longer only related to how software is developed, but also to how digital technologies, products, and services are evolving using increasingly better, faster and more robust infrastructure and tools (Kelly, 2018).

Agile software development processes, such as XP, Scrum, Lean Software Development, Kanban, etc., emerged almost 20 years ago in response to the challenges and problems caused by the bureaucratic, predictive nature of traditional methodologies. Since then, they have had a tremendous impact on changing the way software is developed, due to a greater focus on people, fast iterations, direct communication, and quick feedback. However, in order to remain agile in their use, agile development processes have had to transform and adapt to new challenges, with a rethinking of some of the fundamental aspects of agility (Kruchten, 2019). On the other hand, there has been a shift in focus from processes and people towards integration technologies, cloud, and DevOps/CD practices, while the increasing use of microservices has renewed interest in software architecture and design (Zimmermann, 2016). Microservices are gaining in importance in all industries because they allow flexible delivery mechanism and migration of monolithic architectures towards highly flexible service-oriented architecture. Therefore, by 2021, 80 percent of the development of all applications on cloud platforms is expected to be based on microservices (Larrucea *et al.*, 2018). Third, development and production infrastructure and tools are gaining in importance as the rapid development of technology and tools has allowed new approaches, such as continuous software engineering, to meet new challenges and old demands - faster, better and cheaper (O'Connor *et al.*, 2017).
Having all this in mind, the question arises of what an agile development process looks like in a software factory of the future. In this paper, for better understanding, multiple case study has described software development processes in some of the leading global technology companies, such as Google, Amazon, Facebook, and Twitter. Obtained results are then compared to identify linking elements that represent agility drivers of the development process in the software factory of the future. The rest of the paper is organized as follows: Section 2 presents the research methodology; Section 3 results of multiple case study at Google, Amazon, Facebook, and Twitter, as well as results of comparative analysis; Section 4 discusses the results, limitations of research, and related work, while the final section presents the conclusions, implications, and directions for further research.

2. METHODOLOGY

The aim of the research is to identify, through a better understanding of the development process of leading technology companies such as Google, Amazon, Facebook and Twitter, the linking elements that have a key impact on the agility of the development process in the software factory of the future. In this regard, the next research questions are defined:

RQ-1*: What does the software development process look like at some of the leading technology companies Google, Amazon, Facebook and Twitter?*

RQ-2*: What linking elements are key agility drivers of the development process in the software factory of the future?*

In order to answer the first research question, a descriptive case study method was conducted in the first phase of the research. For the purposes of the research, multiple cases from some of the leading global technology companies were selected: Google, Amazon, Facebook and Twitter. From different perspectives, a key business, organizational, and cultural aspects, that significantly influence the software development process, as well as key engineering, management, and team practices applied in software development, have been considered. Various sources available on the Internet have been used to collect the data, including peer-reviewed papers, reports, analysis, blogs and websites, which include employee testimonials. In order to answer the second research question, in the next phase of the research, the results from the first phase were subjected to comparative analysis, that is, compared against different perspectives, to identify the linking elements that have a key influence on the agility of the software development process in leading technology companies. These linking elements represent key agility drivers of the development process in the software factory of the future.

3. RESULTS

In accordance with the research methodology, this section first presents the results of multiple case study describing software development in a broader context, taking into account different business, organizational and cultural aspects, followed by management, engineering and team practices that are critical to the success of leading technology companies Google, Amazon, Facebook and Twitter (***RQ-1***). The results of the comparative analysis are then presented, where, from different perspectives, a comparative overview of the results of the multiple case study is given and then identified linking elements that have a key impact on the agility of the software development process (***RQ-2***).

3.1 Google

Google is one of the leading global technology companies specializing in Internet-related services and products, which include web search and advertising, cloud computing, software and hardware. From its beginnings, the company has focused on developing its proprietary algorithms to maximize efficiency in organizing information on a global network. Accordingly, Google's corporate mission is "*to organize the world's information and make it universally accessible and useful*" (Thompson, 2019), in a way "*to provide access to the world's information in one click*" (Thompson, 2019), to reach new heights through rapid innovation and business growth. Google has a multi-functional organizational structure that groups resources and processes based on business functions to support product development teams with a minimum hierarchy. In addition, Google has created a very positive, flexible and creative incubator environment. The founders believed that the 80/20 rule would encourage workers to devote one day a week to work on ideas that interest them and deserve their attention, so many new products and services were a direct result of these "side jobs". There are many reasons for Google's success, some of them are next great engineering, management, and team practices, which have evolved over time based on the accumulated knowledge and wisdom of a large number of extremely talented people (Vise, 2007; Henderson, 2017; Smithson, 2019; Thompson, 2019):

- *Single Source Repository*: Most of the source code is stored in a single repository and is accessible to all engineers. Complete development occurs on the main branch, with the automated testing system being frequently launched. Each subtree has owners, anyone can modify the code, but only the subtab owner of the code approves the changes. Larger teams have a so-called *build cop* member, which helps engineers to quickly fix problems and bugs.
- *The Build System*: All engineers use the same distributed build system to compile source code, bind, and execute tests, which is reliable, simple, and very fast because it runs on hundreds or thousands of machines, and caching results in the cloud. There are tools that automatically run tests when initializing a code revision and/or before commit changes to the repository.
- *Code Review*: Web-based tools for code auditing with integrated email are used. All changes to the repository must be reviewed by at least one engineer who is automatically suggested by a special tool, while the author of the change may also propose a reviewer by himself to avoid waiting. Engineers are encouraged to keep every change small. The result of the code review is automatically sent to the email list, everyone is free to comment on the changes. There is an experimental part of the repository with code that does not require review.
- *TDD (Test-Driven Development)*: The emphasis is put on testing, which is widely represented. All production code must have unit tests, and mocking frameworks are also used. In addition, integration, regression and stress tests are used before delivery to production. There are tools that measure code coverage by tests.
- *Active Bug Tracking*: A special system is used to track process execution, errors, user requests and issues. Usually (not generally) teams analyze open issues in their components, prioritize them, and assign tasks to individual engineers. Some teams have a particular responsibility for analyzing and marking errors, while others do so within regular meetings.
- *Multiple Programming Languages*: Several programming languages (C ++, Java, Python, Go, JavaScript) are used, for which there are writing style instructions, as well as many domain-specific languages, for which interoperability is used the so-called Protocol Buffer. Uniformity is key to facilitate development, even when there is an enormous amount of code and different languages. A unique set of commands is used for common engineering tasks.
- *Debugging and Profiling Tools*: There are libraries that provide numerous tools for debugging production servers, automatically dumping and logging on failure, tracking incoming/outgoing calls, resource usage, profiling, etc.
- *Deployment pipeline*: Release-related jobs are mostly done by software engineers, and only a few DevOps teams have release engineers. The release is done frequently, on a weekly or 4-day basis, some individual teams on a daily basis, with the automation of most engineering jobs during the release. Frequent deliveries keep engineers motivated at a high level and speeds up the complete process by allowing fast iterations and feedback, and more opportunities to respond rapidly. The release candidate goes first to the staging server for further integration tests, usually by sending copies of real-world requests from a small number of users or by the development team. Then the release candidate goes to the canary server where it is tested with some of the real-time production data, and eventually gradually, within a couple of days, it is shipped to production data centers.
- *Launch approval*: Release of any changes visible to the user or significant design changes require approval from multiple people outside the DevOps team to ensure that the code meets the requirements regarding regulations, privacy, security, reliability, functionality, etc. This is done using a special tool that monitors audit requirements and approvals to ensure compliance with the defined delivery process for each product.
- *Post-mortems*: For any significant problem that arises, everyone involved is required to write a post-mortem document describing the incident, impact, timeline, causes, possible solutions, and action plan. The focus is on the problem and how to avoid it in the future, not the determination of individual guilt.
- *Frequent rewrites*: Most of the software is rewritten from scratch every two years. Although expensive at first glance, it is key for increased agility and the long-term success of Google. During this period, requirements have usually changed a lot, technology and environment have also changed, which significantly affects the needs, desires and expectations of customers. Also, over time it has been accumulated a lot of complexity and technical debt, so rewriting the code can eliminate complexity, especially in the part related to old requirements that are no longer relevant. In addition, it is a way to transfer knowledge and increase the sense of ownership of a product by new team members, and that is key to productivity. This increases the mobility of engineers and participation in various projects, which contributes to the dissemination of new ideas, and ensures that the code is written using modern technologies and methodologies.
- *20% time*: Engineers can spend up to 20% of their working time on projects of their choice, without the approval of the manager, to stay motivated and to prevent burnout.
- *Objectives and Key Results - OKRs*: Both individuals and teams at each level of the company have to define and document the quarterly and annual goals as well as report on the progress towards their realization. OKRs are

visible throughout the company, except for confidential projects. OKRs are not a basis for measuring individuals' performance but are a mechanism for communicating and promoting individuals through social incentive.

- *Project approval*: There is no uniform approval process of projects. Managers at all levels are personally responsible for the projects implemented by their teams.
- *Corporate reorganizations*: Reorganization can be caused by major project cancellations, project defragmentation across multiple geographic locations, merging or splitting teams, and changes to the reporting chain. Frequent reorganizations are used to increase efficiency and to avoid or reduce the effects of Conway's law, the consequence of which is that the software architecture reflects the organizational structure.
- *Annual hackweeks*: Innovation is encouraged to maintain an annual hackathon, during which engineers can work on innovative projects. However, due to regular commitments, the response rate is relatively small (5-20%).
- *Facilities*: The workspace is open and quite crowded to support communication, with venues changing frequently while meetings are organized in a specially equipped video conference room.
- *Training*: Employee training is supported in a number of ways: initial training for new employees, various online training, courses and studies at external institutions.
- *Internal Transfers*: Employee transfers between different parts of the company are supported in order to spread knowledge and improve communication.
- *Bottom-up performance appraisal and rewards*: Feedback is highly valued, so engineers can reciprocate positive reviews for a cash bonus and/or special recognition for good work. In addition, there are bonuses to performance and a very careful process to ensure that the right people are promoted. Job loss due to poor performance is possible, but in practice it is extremely rare.

3.2 Amazon

Amazon is one of the technology companies with the greatest economic and cultural influence in the world, which as the world's largest online seller opted for on-line commerce, cloud computing services, distribution of digital content, computer hardware and software, and artificial intelligence. Amazon’s mission statement is “*we strive to offer our customers the lowest possible prices, the best available selection, and the utmost convenience*” (Gregory, 2019), in a way „*to be Earth’s most customer-centric company, where customers can find and discover anything they might want to buy online*“ (Gregory, 2019). Amazon focuses on an obsession with customers (rather than a focus on competition), a passion for innovation, a commitment to operational excellence, and long-term thinking. Amazon has a clean functional organizational structure that uses business functions as the basis for merging parts of an organization to support global functional-based groups, global hierarchies, and geographical divisions. Amazon uses various software development methodologies for software development at Amazon, such as Scrum, XP, Crystal, and other hybrid methodologies, with teams choosing whether and how to apply the methodology. As long as the team makes progress, with measurable results, management gets the team out of the way, moreover, management often does not know how the development process works. Accordingly, some of the key engineering, management and team practices at Amazon are (Hicks, 2013; Novak, 2016; Meyer, 2017, 2019; Weiss, 2018; Gregory, 2019; Hoff & Exner, 2019):

- *Bottom-up Planning*: Planning is a bottom-up because the teams are closest to the customers and they know what the customers want. The 6-page operational plan is done on an annual basis, where each team explains what it would do with existing ones and what additional resources they will need. Managers based on team plans make their plan 6 pages and so on. Based on plans, resources are allocated to teams.
- *Decomposition & microservices*: Initially monolithic software architecture was decomposed into microservices. Microservices have one purpose, they are simple, autonomous, and communicate via APIs. On the other hand, the organization was decomposed into the so-called *two-pizza* teams.
- *Automatization & Testing*: Everything is maximally automated, from the build and release to delivery, where multiple types of automated tests are implemented: functional, integration, web-based interface tests, stress and performance tests, all of which are subject to monitoring and measurement. The goal is to detect errors as early as possible, ideally in the ECT (Edit-Compile-Test) phase. All this allows changes to be delivered more often while maintaining high quality, ie deliveries are faster and of better quality.
- *Focus on Security*: The issue of security permeates the whole process of development and delivery, so software engineers have to deal with programming, architecture, testing and security aspects. In order to incorporate security into the development process, DevSecOps is practiced.
- *Threat model and Architecture*: The new project begins with developers working on the threat model and architecture, as they are closest to the user, and then the threat model is reviewed by security engineers. Developers are responsible for architecture, which is reviewed in collaboration with architect or chief engineer, whose role is to audit and teach. The same is the case with security and testing, because the team owns the development process and other roles are intended to teach the team. The code is reviewed by other engineers before being committed to the repository. During the build of the static code analysis was performed and then the changes are queued for release with additional testing.
- *Delivery pipeline*: The delivery process is implemented through a fully automated delivery pipeline, to allow for faster and more reliable, continuous, multiple daily deliveries (an average delivery of every 11.7 per second) and is used by all teams. The delivery process is pessimistic, as it constantly tries to find a reason why deliveries will

not interfere, both before production and in production. Changes are gradually introduced into production by availability zones; in case of problems rollback is done. Controls are built-in through an automated delivery pipeline with the application of local and global policies and rules. Each team can have its own rules, but for commit there must be 70% code coverage for unit tests. There are general delivery rules, but they can be customized and applied at the team, organizational, or company level. The delivery mechanism contains best practices, which are the result of years of experience, to ensure that people do not do the wrong thing, while automation allows the delivery process to be executed in the right way each time.

- *Ops culture*: Operations are of the utmost importance, and leaders are expected to model what is important, which is why they spend a lot of time on operations, while each team must present and explain each deviation of indicators at a weekly meeting dedicated to operations.
- *Two-pizza teams*: The entire organization is divided into *two-pizza* teams. *Two-pizza* teams are small, autonomous, decentralized and agile DevOps teams, which act like startups. They take responsibility and ownership of their microservices, from start to finish, dealing with customers, development, testing, support, and maintenance. They define goals and are given the resources to achieve them. Ideas come from teams because they are closest to the users, while management evaluates and approves requests. Management acts as a board of directors that manages startups, audits and monitors the realization of their goals and metrics. Teams can have experts in various competencies, such as web developer, SW engineer, product manager, documentation writer, marketer, etc.
- *Communication and Consistency*: Communication between teams is sometimes difficult because teams are separated. In these situations, multiple teams may do the same thing, but this risk is accepted and subsequently resolved so that the process does not slow down. In such situations, in order to maintain consistency, teams are often refactored by creating a new team and a new service for which the new team will take responsibility.
- *Monitor & Measure Everything*: Everything is monitored and measured, promoting the Ops culture and the importance of data (data culture). The Weekly Business Review (WBR) meetings use dashboards to analyze the results of monitoring client-side resources, applications in staging and the production environment. There is canary monitoring that initiates positive and negative tests before changes are delivered to production.
- *Post-mortem*: Critical errors are corrected as soon as possible. The logs and metrics (5 Whys) are then analyzed in detail to determine what happened, why the problem occurred, why the problem was not detected earlier, and how to guarantee that such errors will not happen again. It is the content of a separate document that is subject to further analysis by various levels of management, down to top management.

3.3 Facebook

Facebook is one of the leading technology companies, which is basically an online social media and social network. Facebook has become the largest social network in the world, with products used by over 2 billion people worldwide to communicate, share ideas and support. Facebook's corporate mission is "*to give people the power to build community and bring the world closer together*" (Smithson, 2019), in a way that „*people use Facebook to stay connected with friends and family, to discover what's going on in the world, and to share and express what matters to them*" (Smithson, 2019). Facebook focuses on supporting corporate efficiency in international business growth, data privacy and security issues. Facebook has a matrix organizational structure, which supports the creativity, innovation of corporate functionally-based teams, geographic and product divisions. Facebook is defined by a unique culture that rewards impact, encouraging employees to be brave and solve the problems they care about most in working in small teams, moving quickly through iterations to develop new products. Kent Beck, one of the founders of the agile movement and having a background in Facebook, describes Facebook as a real agile company, not because it implements one or the other agile development process, but because it does not over-plan and over-design, it is very flexible, ready for change and does not try to predict the future. Accordingly, some of the key engineering, management and team practices at Facebook are (Bird, 2013; Feitelson *et al.*, 2013; Rossi, 2017; Lombardo, 2018; Messersmith, 2019; Smithson, 2019):

- *Common codebase*: Engineers work directly on common code, without branching. In response to emergencies, engineers can investigate source code unrelated to their regular work, submitting updates or at least detailed reports of errors found.
- *Perpetual development*: The key features of Facebook are speed and growth - the growth in the number of code engineers, the growth of these engineers' overall activity, and the growth of the database itself, with a clear awareness that the system will be developed indefinitely. A product cannot be defined in advance but must evolve at a continuous and rapid rate.
- *Continuous deployment*: Continuous delivery means that iterations of new or modified small-scale code are delivered continuously as soon as they are ready. This solves integration problems on the one hand and enables real-time experimentation on the other. Due to the limited scope of code changes, it reduces but does not eliminate, the risk of something going wrong. However, this does not mean that the new code is immediately available to users. For this, a special Gatekeeper tool is used, which controls which users can see certain features. This allows incremental delivery and testing of new services without exposing them to customers. Gatekeeper also allows the new code to be shut down in case of problems, to disable parts of the user interface or to check the stability of the system with stress tests.

- *Deployment pipeline*: The new code is delivered at high speed - once a week, using a quasi-continuous system. After unit tests, the new code is audited by another software engineer using the Phabricator tool. Subsequently, the software engineers, in conjunction with the release engineer, implement commit and push activities, which automatically starts accumulated regression tests, with user interface testing using the Watir and WebDriver tools. In addition, the new code is tested through its internal use. The result of the whole process is more likely that the delivered code will not cause a problem in the system. Perflab tool is used to test the impact of new code on performance, before the new code is put into production. Standard weekly delivery to production is realized in several stages. The first phase (H1) involves delivery to internal servers accessible only to engineers, in order to perform the last round of testing by engineers who changed the code. Phase Two (H2) is the delivery to thousands of machines that serve a small fraction of real users. If the new code does not cause errors, then in the third phase (H3) its full delivery to the production servers is made. In the case of an error, the code is corrected and the entire process is repeated, allowing the production code to be restored to the previous version. During the delivery of the new code, all engineers who worked on it must be available online, which is automatically checked by the release system.
- *Continuous Monitoring*: The complete system is continuously monitored using internally developed tools such as Claspin.
- *DevOps Team*: With a huge number of developers (over 1000) and several engineers posting daily and weekly deliveries to production, there is no dedicated quality assurance team or any other role or team whose main task is testing. There is no dedicated team responsible for testing.
- *Personal responsibility*: The personal responsibility of software engineers writing code is more important than external quality control, as software engineers must support the operational use of their software, which further motivates them to write good code and test it thoroughly. The dedication and personal responsibility of software engineers is critical, both for the smooth functioning of the system and for maintaining quality in scaling. It emphasizes that methodologies and tools alone are not sufficient, as they can always be misused.
- *Experimenting*: In order to respond to the challenges of scaling, different teams almost always experiment with multiple alternative solutions with different advantages and disadvantages, whereby whatever solution is chosen, the others are stored as a reserve.
- *Self-select work*: The scheduling of work assignments is driven by software engineers. The engineers themselves choose the tasks they will work on.
- *Training and mobility:* All new employees receive in-depth training where they are introduced to code, culture, and processes, and then select a team that they feel will enable them to express their potential, in line with the company's priorities. It is also possible to move from one team to another. One way to support mobility is hackamonth, where engineers join another team to work on new ideas for a few weeks in the domain of those teams. After that, the engineer can officially join another team as well.
- *Innovative hackathons:* Innovation is encouraged by breaking the routine with frequent, daily hackathons. The hackathons are focused and intense, fostering interactions between all parts of the company - not just the engineers, but also the finance, legal and other departments.
- *Sustainable work rates*: The reverse of personal responsibility is the responsibility of the engineers to themselves. Due to the culture of continuous development, the concept of sustainable work rate is also promoted. Hacking culture does not involve extra hours, and no delivery of the new code is scheduled for the weekend.
- *A/B testing*: Although engineers have first-hand domain experience, they need to test innovation on real users to see what works. Testing on real users under scaling conditions is possible and provides more accurate and immediate feedback.
- *Learning from experience*: Learning from experience and mistakes is more important and useful than punishing those responsible for failure and seek to blame for failure.
- *Impact*: There is a significant focus on the 6-month audit cycle, where every engineer has to demonstrate impact across the company by presenting their key metrics, results and personal impact of those results.

3.4 Twitter

Twitter is one of the most innovative companies in the world, and is a global platform for microblogging, conversation, marketing and social networking through the Internet, connecting its users with people, information, ideas, opinions and news. Twitter mission statement is "*to give everyone the power to create and share ideas and information instantly without barriers*" (MSA, 2019), in a way that "*believing in free expression and the thought that every voice has the power to impact the world*" (MSA, 2019). Twitter has a clean functional organizational structure, which supports creativity and innovation to continue to be globally relevant and meaningful. Twiter promotes intersectionality and diversity, as well as a culture of inclusivity and transparency. What makes it unique, along with its development history, is its rapid scaling to prevent site congestion or collapse with a rapid increase the number of user's requests. Although Twiter employs Scrum and agile engineering practices, the result of agile transformation is a rather ad-hoc development process in the work of highly autonomous teams, based on a simple agile working framework and lean model. Accordingly, some of the key engineering, management and team practices at Twitter are (Aniszczyk, 2013; Raffi, 2013; Isaac, 2014; Dymond, 2016; Kostenko, 2016; Aiello, 2018; Chong, 2018; Ngahane & Goodsell, 2018; MSA, 2019):

- *Growth Mindset*: There is an intense focus on a mindset that promotes growth and engineering culture that drives engineers to embrace nothing but excellence.
- *Move Fast*: The key priority is to deliver software at high speed, but fast-moving requires both proper metrics and process monitoring to achieve a balance between speed and quality, as most of the costs of software development are related to maintenance. It is also emphasized that free experimentation with new functionalities requires that the impact of changes on the rest of the system be isolated.
- *Processes are Guidelines — not rules*: Processes are guidelines, not rules - and they update as the team/company grows. When a certain level is crossed it becomes difficult to follow things without a standard procedure because of the difficulty in sharing knowledge that grows exponentially. Each new project is accompanied by a technical design document containing goals, non-goals, possible solutions, success criteria and potential risk factors, in order to standardize the design process. This forces the Chief Engineer to think about the whole scenario, end-to-end, sourcing requirements, considering the pros and cons of different alternatives, selecting the solution and linking the solution to the success metrics. Work is planned in two-week sprints, and each sprint begins with a list of tasks that are prioritized according to resource availability and urgency. The implementation of selected tasks is monitored to identify bottlenecks and, most importantly, to find out if all tasks can be completed on time. At the end of each sprint is a retrospective, where is considered what went well and what could be improved. Wins are celebrated and failures are analyzed so that the same mistakes will not be made in the future.
- *Blameless Culture*: Embracing risks and making mistakes are key to building a flexible, resilient and fast-growing company. A work environment that does not seek and condemn the culprit for a mistake is promoted, but the learning from mistakes is promoted. The cause of the error is vigorously searched for in order to eliminate the error at the source of its occurrence, as well as the cause of its occurrence, to prevent its recurrence. It is considered very important to allow everyone to work on something that is important and to be truly committed to what they do, as well as the ability to immediately create and share information and ideas in an open and vivid atmosphere.
- *Focusing on Quality*: Disseminating team knowledge plays a crucial role in enabling long-term stability and eliminating technical debt. Knowledge is viewed in two dimensions - what is known (*breadth*) and how well it is known (*depth*). For each team, a *bus factor* is defined, which indicates the number of team members who can leave the team before the project is stopped due to lack of expertise. To eliminate the risk, various initiatives are being launched - investing 30 minutes a day in the documentation, sharing team members' expertise *with tech brown bag* expertise, and assigning team members tasks that are out of domain during a sprint to get a chance to learn a different technology steak or code. The chances of success are also increased by making small, incremental changes, with minimal risk.
- *Experiments*: To build a learning culture, one has to experiment. A lot is being invested in infrastructure to allow people to experiment and test a new ideas. This is the way to get new product.
- *Tim scaling*: The key thing about scaling a team is that new teams still have some structure in place to know what they need to do, but to keep the ability to experiment and iterate quickly.
- *Microservices*: A rapidly growing site is causing architectural problems (eg 143.200 tweets/second during World Cup 2010), where a monolithic application could not withstand scalability requirements, which required the implementation of good encapsulation and modularity practices. For this reason, microservice architecture was implemented that supports scaling and allows for an increase in 10-100x performance, asynchronous Event-Driven Design (EDD), parallel execution and replication. Failures are inevitable in a distributed environment, so the use of microservices isolates failures by defining clear context boundaries and related business logic by designing a loosely coupled service model. Microservice architecture is a flexible platform based on which new functionalities can be developed and delivered faster.
- *Team Authonomy & Ownership*: The goal is to retain the autonomy of each individual team as much as possible, with the organizational structure set up so that teams can work together. Clear context boundaries, service orientation, and microservices make it possible to increase team independence by making the team fully responsible for their microservices. Engineering is (mostly) based on teams that can develop their own microservices independently and very quickly, which has a huge impact on operations.
- *Runtime Configuration and Testing*: Delivery of new code into production involves coordination between a large number of different services. On the other hand, it is difficult to run tests in a completely isolated environment. Therefore, instead of the staging phase, a configured production environment is used for testing, to take advantage of realistic data in scaling conditions. This uses the Decider tool, which functions as a switch, with multiple systems across the infrastructure responding to this change in near real-time. This means that software may come into production, but a particular function may not be "active". This also allows for percentage inclusion, such as the functionality available for a certain percentage of requests or users. In this way, the system can be tested in a completely "off" and safe setting, as long as it is not certain that it is functioning properly and that the system can withstand a high load. All this reduces the need for coordination at the team level.
- *Production ML Pipelines*: In order to improve the delivery process, advanced artificial intelligence is used, and so that engineers can focus more on modeling, experimentation and user experience, production ML (Machine Learning) pipelines are applied. The automation, scheduling and sharing of ML pipelines use the ML Workflows special tool.

- *Monitoring & Governance*: Tools (Zipkin and Viz) are used to monitor and manage microservices to facilitate the collection of microservice performance data and to support data-based decision making. Thus, the Zipkin tool enables a visual representation of the time required to fulfill requirements.
- *The Best People Own Problems*: Twitter is a very small company in terms of the volume and impact it makes in the world, making it easy for small teams to make a big impact. It's not uncommon to see a team of 10 people making tens and hundreds of millions of dollars in revenue a year. In a way, this makes Twitter work very special, as it allows engineers to take responsibility for solving problems, especially in dangerous situations where the best people take responsibility, face problems, and solve them.
- *Reward System*: Opportunities for learning and growth for individuals are supported and provided, with a balance of work and leisure.
- *Training*: From an organizational point of view, the company can be seen as a school because new employees do not have the necessary knowledge, so intensive training aims to get them to work as soon as possible.
- *Mobility*: At the individual level, every quarter, employees can find an engineering role in the team that wants to join him. This creates a free market for talent within the company.
- *Bottom-up Promotions*: There is a transparent system of nomination for promotion by colleagues, not by management, ie. promotion is not dependent on the free will of the manager, but on feedback from colleagues, which is then evaluated by a team of engineers.

3.5 Comparative analysis

Based on the results of multiple case study, from different perspectives, important agility drivers and practices have been identified, related to software development in leading technology companies: Google, Amazon, Facebook and Twitter (Table 1).

Table 1: Key agility drivers and practices of the software development process in leading technology companies

Perspective	Google	Amazon	Facebook	Twitter
Business	-reaching new heights -rapid innovations -business growth -speed	-obsession to customers -passion for innovation -operative excellence -long-term thinking	-corporate efficiency -global growth -privacy -security	-global relevance and significance -growth -speed and quality
Organization	-multi-functional -minimal hierarchy	-functional -global hierarchy	-matric -flexible	-functional -like school
Culture	-innovativeness -openness	-research ideas -risk-taking	-hacking -improvement	-innovativeness -transparency
Development	-20% of the time to work independently -one repository -build system -code review -TDD -debugging tools -bug tracking -eliminating the cause of the problem -DevOps -frequent replacements	-bottom-up planning -model of threats and architecture -startup projects -decomposition and microservices -TDD -automation of testing -DevSecOps -continuous delivery -removing the cause of the problem (5 Why)	-common source code -independent choice of tasks -perpetual development -experimentation -TDD -continuous delivery -sustainable pace of work -DevOps - code review -shared files	-processes are guidelines -minimal documentation -focus on speed and quality -experimentation -TDD / EDD -15-day sprints -retrospectives -microservices & APIs -incremental change -the best people own problems
Operations	-delivery pipeline -approval of every delivery -daily, 4-day or weekly delivery -canary monitoring	-deployment pipeline -multiple daily deliveries (every 11.7 seconds) -pessimistic delivery -canary monitoring	-deployment pipeline -delivery 2 times a day and once a week -continuous monitoring -real-time experiments -A/B testing	-deployment pipeline -runtime configuration and testing -ML pipelines -continuous monitoring -A/B testing

Team & People	-frequent reorganization -hackathons -open workspace -intensive training -internal transfers -social incentives -bottom-up promotions	-communication and consistency -*two-pizza* teams -startup culture -refactoring teams -Ops based data culture	-the impact of individuals on the company is valued -training and mobility -innovative hackathons -learning from experience - personal responsibility	-team autonomy -blameless culture -scaling the team -training and mobility -*bus factor* -*tech brown bag* -bottom-up promotions
Technology	-tools for testing, code review, test coverage and error tracking, profiling and monitoring	- tools for testing, static analysis, delivery, performance, monitoring	- tools for code and interface testing, incremental delivery and gradual introduction, performance, stress testing, monitoring	- tools for testing, production environment configuration, automated ML, monitoring, management

Source: Authors

Comparative analysis, from different perspectives, identifies linking elements related to business, organizational, cultural aspects, as well as engineering, management and team practices of Google, Amazon, Facebook, and Twitter, which have a key influence on the agility of the software development process in the software factory of the future:

1. ***Business focus on speed, innovation and** growth:* The driving force behind all development activities is a business focus on continuous growth based on accelerated innovative development.
2. ***Flexible organizational structure**:* Instead of the traditional, hierarchical structure, flexible organizational structures, such as functional and matrix, are used to support intensive training, mobility, rapid learning and accountability of employees.
3. ***Innovative hacking culture**:* A hacker culture is promoted where the willingness to experiment, innovate, be open and transparent is especially appreciated.
4. ***Key engineering practices in agile software development (Dev)**:* Instead of a predefined methodology, test-driven development (TDD) practice is implemented, supported by common code, code review, test automation and monitoring of code coverage by tests. The different unit, functional, integration, web-based and regression tests are used in order to detect errors as early as possible and to eliminate the cause of the problem, in order to deliver quality code continuously in the production. DevOps practice is widely used, where the development team not only engages with customers, development, and testing, but supports the operational use and maintenance of its components, or DevSecOps practice, which puts security at stake. DevOps practice is accompanied by incremental changes and continuous delivery (CD) changes of small scale, on a weekly, daily or multiple daily. To support continuous, frequent deliveries, service orientation and decomposition to microservices are applied, which become the ownership and responsibility of small, autonomous (two-pizza) teams. This implies an increased focus on architecture, planning and modeling, on the one hand, but also new opportunities for experimentation and startup projects, on the other, to achieve a balance of speed and quality. Due to the development of new technologies, accumulation of technical debt, change of complexity and requirements, problems of scaling in the cloud environment, as well as an increase of the sense of ownership of the product by new team members, complete replacement of the software or its key parts is periodically rewritten and replaced, with the use of new technologies and development practices.
5. ***Key engineering practices in agile operations (Ops)**:* Deployment pipeline is a key operations-related practice that allows for continuous development and rapid, multiple daily deliveries. Deployment pipelines can include a development, test, staging and production environment where, with the use of the appropriate configuration, testing and continuous canary monitoring tools. It supports real-time experimentation and A/B testing with pessimistic delivery. The high degree of automation and application of ML gives DevOps team members more time to work on modeling, experimentation and user experience.
6. ***Flexibility of agile DevOps team**:* Selection of the right people, intensive training and bottom-up promotion is crucial for the autonomy, flexibility, scaling and refactoring of DevOps teams. There is a transfer of knowledge between people and teams due to the mobility of engineers and/or internal transfers of engineers between different teams. Open workspace, innovative hackathons, blameless culture and social incentives contribute to open communication, experience-based learning, startup thinking and taking responsibility. Data-driven decision making is promoted and the impact of the individual is valued.
7. ***Increasing importance of tools**:* DevOps/CD practices, deployment pipeline, as well as the implementation of microservices, rely heavily on tools. In such a context, the implementation of software solutions without tools support is difficult to achieve. In the development process, these are tools for code management, testing and review, measuring code coverage for tests, error management, auditing, and static analysis. With regard to operations, tools for continuous code delivery, production environment configuration, incremental deployment and A/B testing, monitoring, profiling, performance monitoring, automated ML and management are implemented.

4. DISCUSSION

The engineering practices of some of the leading technology companies, such as Google, Amazon, Facebook, and Twitter, have much in common with agile software development, as they allow the software to evolve and deliver at high speed, while perhaps the greatest specificity is how individual accountability and team responsibility can replace specialization, methodologies and formalized procedures. The agile engineering practices are not isolated but fit with the engineering culture and code control process, so that the needs for rapid development and delivery cycles are aligned with the needs for monitoring, robustness, and quality. In addition to the aforementioned agile engineering practices, the principles of leadership, sense of urgency, operational excellence, social incentive, scaling and decision making based on real data are also promoted. Technological advances have enabled the development of tools that contribute to raising agility to a new level, while business based on technological innovation and digital strategies requires an agile approach. To maintain competitive advantage, agile DevOps teams need to increasingly shorten the delivery cycle and customer feedback loop, and drastically reduce the gap between business, team, and technology. Although it cannot be determined accurately, the impact of the practices described on the success of leading technology companies is critical. The described key engineering, management, and team practices are the result of years of work by a large number of managers and engineers, and have withstood the test of time.

There are several limitations to the research conducted. First, it concerns how data was collected, because there was no possibility of direct access to resources and first-hand data, e.g. using (semi) structured interview techniques or questionnaires. To overcome this limitation, online testimonies from both former and active engineers were used. Another limitation is related to the quality of the data itself, given that very few peer-reviewed papers have been identified related to the software development process in selected companies. This is why multiple gray sources, such as testimonials from blogs and website employees, have been used.

According to the information available, there is no similar research that focuses on the software development process in leading technology companies. Hewage *et al.* (2018) conducted a comparative analysis of big data methods and techniques applied by Google, Amazon, Facebook and Twitter, based on sixteen peer-reviewed papers publications from 2007-2015, but this research does not in any way address the software development process.

5. CONCLUSIONS

This paper investigates, describes and compares software development processes in some of the leading software companies, such as Google, Amazon, Facebook, and Twitter, to identify linking elements that have a key impact on the agility of the development process in the software factory of the future. For this purpose, multiple case study and comparative analysis were applied.

The results obtained indicate that there are different linking elements when comparing business, organizational and cultural aspects, as well as engineering, management and team practices, which can be said to have a key impact on the agility of the software development process in the software factory of the future. The identified linking elements and key agility drivers in the software factory of the future are: 1) business focus on speed, innovations, and growth, 2) flexible organizational structure, such as a functional or matrix; 3) innovative, hacker culture, 4) key engineering practices in agile software development, such as TDD, DevOps, CD, microservices, etc., 5) key engineering practices in agile operations, such as automated deployment pipeline, A/B tests, stress tests, canary monitoring, ML pipeline, 6) highly flexible and agile DevOps teams, such as two-pizza teams, and 7) an increasingly significant role of tools to support engineering practices in agile software development and operations. In this regard, it can be expected that in the software factory of the future, individual accountability and team responsibility will replace specialization, formalized procedures and agile methodologies in the form we have known them so far. Also, it can be expected further shortening the delivery cycle and users' feedback loops with continuously reducing the gap between business, development team and technology.

The results of the research will contribute to a better understanding of the basic principles of agility and agile development process in the software factory of the future. Further research may go towards further empirical evaluation of the results obtained, especially in the context of the development of complex business software systems.

REFERENCES

Aiello, C. (2018) Jack Dorsey just blew up Twitter's org structure to prepare it for the next decade. Available at: https://www.cnbc.com/2018/06/28/twitter-is-restructuring-as-its-product-executive-steps-down.html (Accessed: 2 March 2020).

Aniszczyk, C. (2013) Evolution of The Twitter Stack. Available at: https://www.slideshare.net/caniszczyk/twitter-opensourcestacklinuxcon2013 (Accessed: 2 March 2020).

Bird, J. (2013) This is how Facebook develops and deploys software. Should you care? Available at: https://dzone.com/articles/how-facebook-develops-and (Accessed: 2 March 2020).

Chong, Z. (2018) What I've learned in 1 year at Twitter. Available at: https://www.freecodecamp.org/news/what-ive-learned-in-1-year-at-twitter-65150f5d4af2/ (Accessed: 2 March 2020).

Dymond, R. (2016) Does Twitter use agile? Available at: https://www.quora.com/Does-Twitter-use-agile (Accessed: 2 March 2020).

Feitelson, D. G., Frachtenberg, E. & Beck, K. L. (2013) 'Development and Deployment at Facebook', IEEE Internet Computing, 17(4), pp. 8–17.

Gregory, L. (2019) Amazon.com Inc.'s Mission Statement & Vision Statement. Available at: http://panmore.com/amazon-com-inc-vision-statement-mission-statement-analysis (Accessed: 2 March 2020).

Henderson, F. (2017) 'Software Engineering at Google', in CoRR.

Hewage, T. N., Halgamuge, M. N., Syed, A. & Ekici, G. (2018) 'Review: Big data techniques of google, Amazon, Facebook and Twitter', Journal of Communications, 13(2), pp. 94–100.

Hicks, C. (2013) What software development methodology do the developers at Amazon use? Available at: https://www.quora.com/What-software-development-methodology-do-the-developers-at-Amazon-use (Accessed: 1 March 2020).

Hoff, T. & Exner, K. (2019) How is software developed at Amazon? Available at: http://highscalability.com/blog/2019/3/4/how-is-software-developed-at-amazon.html (Accessed: 5 March 2020).

Isaac, M. (2014) Twitter SVP Chris Fry Breaks Down How His Engineering Org Works. Available at: https://www.vox.com/2014/1/2/11621872/twitter-svp-chris-fry-breaks-down-how-his-engineering-org-works (Accessed: 2 March 2020).

Kelly, A. (2018) The Future of Agile Is Digital. Available at: https://www.agileconnection.com/article/future-agile-digital (Accessed: 6 March 2020).

Kostenko, A. (2016) How Twitter Works. Available at: https://www.slideshare.net/ITARENA/how-twitter-works-arsen-kostenko-technology-stream (Accessed: 2 March 2020).

Kruchten, P. (2019) 'The End of Agile as We Know It', in IEEE/ACM International Conference on Software and System Processes (ICSSP). Institute of Electrical and Electronics Engineers (IEEE), pp. 104–104.

Larrucea, X., Santamaria, I., Colomo-palacios, R. & Ebert, C. (2018) 'Microservices', IEEE Software, 35(3), pp. 96–100.

Lombardo, J. (2018) Facebook Inc.'s Organizational Structure. Available at: http://panmore.com/facebook-inc-organizational-structure-analysis (Accessed: 2 March 2020).

Messersmith, M. (2019) Kent Beck fired from Facebook. Available at: https://rackandstack-tech.blog/2019/10/15/kent-beck-fired-from-facebook/ (Accessed: 2 March 2020).

Meyer, P. (2017) Amazon.com Inc.'s Organizational Culture Characteristics. Available at: http://panmore.com/amazon-com-inc-organizational-culture-characteristics-analysis (Accessed: 2 March 2020).

Meyer, P. (2019) Amazon.com Inc.'s Organizational Structure Characteristics. Available at: http://panmore.com/amazon-com-inc-organizational-structure-characteristics-analysis (Accessed: 2 March 2020).

MSA (2019) Twitter Mission Statement Analysis, Mission Statement Academy. Available at: https://mission-statement.com/twitter/ (Accessed: 2 April 2020).

Ngahane, S. & Goodsell, D. (2018) Productionizing ML with workflows at Twitter. Available at: https://blog.twitter.com/engineering/en_us/topics/insights/2018/ml-workflows.html (Accessed: 2 April 2020).

Novak, A. (2016) Most Companies Deploying Code Weekly, Daily, or Hourly. Available at: https://blog.newrelic.com/technology/data-culture-survey-results-faster-deployment/ (Accessed: 2 March 2020).

O'Connor, R. V., Elger, P. & Clarke, P. M. (2017) 'Continuous software engineering—A microservices architecture perspective', Journal of Software: Evolution and Process, 29(11), pp. 1–12.

Raffi (2013) New Tweets per second record, and how! Available at: https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how.html (Accessed: 2 March 2020).

Rossi, C. (2017) Rapid release at massive scale, Facebook Engineering. Available at: https://engineering.fb.com/developer-tools/rapid-release-at-massive-scale/ (Accessed: 2 March 2020).

Smithson, N. (2019) Google's Organizational Structure & Its Characteristics. Available at: http://panmore.com/google-organizational-structure-characteristics-analysis (Accessed: 2 April 2020).

Thompson, A. (2019) Google's Mission Statement and Vision Statement. Available at: http://panmore.com/google-vision-statement-mission-statement (Accessed: 2 March 2020).

Vise, D. (2007) ‘The Google Story’, Strategic Direction. Emerald Group Publishing Limited, 23(10), pp. 117–133.

Weiss, J. (2018) The Software Development Process at Amazon. Available at: http://aws-de-media.s3.amazonaws.com/images/AWS_Summit_2018/June6/Kumo/20180606 - Berlin Summit - Day 1 - 12.15 - 13.00 The Software Development Process at Amazon.pdf (Accessed: 3 March 2020).

Zimmermann, O. (2016) ‘Microservices tenets: Agile approach to service development and deployment’, Computer Science - Research and Development. Springer Berlin Heidelberg.