jamk

DevOps in mobile game development

An action research on applying DevOps practices in a mobile game development project

Tarmo Jussila

Master's Thesis
May 2021
Engineering and Technology
Technological Competence Management

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Description

Jussila, Tarmo

DevOps in mobile game development. An action research on applying DevOps practices in a mobile game development project

Jyväskylä: JAMK University of Applied Sciences, May 2021, 73 pages.

Engineering and Technology. Master's Degree Programme in Technological Competence Management. Master's Thesis.

Permission for web publication: Yes

Language of publication: English

Abstract

DevOps is a prevailing trend in the software industry that aims to unify development and operations activities and teams, while providing a quality-oriented approach to managing changes introduced to a software, often leveraging automation in the process.

The objectives of the research focused on practices and tools that enabled applying DevOps for the benefit of mobile game development, and more specifically a single mobile game development project that was being prepared for release during the research period. Furthermore, it was important to study the cultural impacts that DevOps had on the mobile game development team that was involved in the project.

The research was implemented using an action research strategy where the researcher was actively involved in the DevOps transformation process and was working as a part of the development team in the mobile game development project. Both the project and the development team were being researched, using group interviews and participant observation methods for gathering the research material.

As key results, continuous delivery and continuous monitoring practices were implemented into the mobile game development project using various tools that formed a so-called DevOps toolchain. Several automation pipelines were engineered to reduce the time and effort that the development team needed to spend on repetitive yet crucial tasks.

In conclusion, DevOps practices proved to be highly applicable in supporting mobile game development and the introduction of DevOps culture provided a quality-oriented environment where open communication was both welcomed and encouraged.

Keywords/tags (subjects)

DevOps, mobile game development, action research

Miscellaneous (confidential information)

No confidential information

jamk

Kuvailulehti

Jussila, Tarmo

DevOps mobiilipelikehityksessä. Toimintatutkimus DevOps-käytänteiden soveltamisesta mobiilipelikehitysprojektissa

Jyväskylä: Jyväskylän ammattikorkeakoulu, Toukokuu 2021, 73 sivua.

Tekniikan ala. Teknologiaosaamisen johtamisen tutkinto-ohjelma (YAMK). Opinnäytetyö (YAMK).

Verkkojulkaisulupa myönnetty: Kyllä

Julkaisun kieli: Englanti

Tiivistelmä

DevOps on ohjelmistokehitysalalla vallitseva trendi, jossa pyritään yhdistämään kehitykseen ja ylläpitoon liittyvien tiimien vastuita ja toimintoja, samalla pyrkien ohjelmiston laatulähtöiseen muutostenhallintaan, usein automaatiota hyödyntäen.

Tutkimuksen tavoitteet keskittyivät käytänteisiin ja työkaluihin, joilla DevOps-toimintamallin jalkautus mobiilipelikehityksen tarpeisiin mahdollistuisi. Tutkimus kohdistui yksittäiseen käynnissä olevaan mobiilipelikehitysprojektiin, jota valmisteltiin tutkimusjakson aikana julkaisuvalmiuteen. Lisäksi oli tärkeää tutkia DevOps-käytänteiden vaikutuksia projektissa toimivan mobiilipelikehitystiimin toimintakulttuuriin.

Tutkimuksessa käytettiin tutkimusotteena toimintatutkimusta, jossa tutkija oli aktiivisesti osallisena DevOps-muutosprosessin jalkauttamisessa ja toimi myös osana kehitystiimiä mobiilipelikehitysprojektissa. Tutkimuksen kohteina olivat sekä mobiilipelikehitysprojekti ja sitä kehittävä kehitystiimi, joista kerättiin tutkimusaineistoa ryhmähaastatteluiden ja osallistuvan havainnoinnin keinoin.

Keskeisinä tuloksina mobiilipelikehitysprojektissa otettiin käyttöön jatkuvan toimituksen ja jatkuvan monitoroinnin käytänteet, jotka mahdollistettiin moninaisilla työkaluilla, joista muodostui yhdessä niin kutsuttu DevOps-työkaluketju. Useita automaatioputkistoja toteutettiin toistuvien toimenpiteiden automatisoimiseksi, jotta kehitystiimin aikaa ja vaivannäköä pystyttiin säästämään.

Johtopäätöksinä DevOps-käytänteiden voitiin nähdä olevan varsin käyttökelpoisia mobiilipelikehityksen tukena ja DevOps-kulttuurilla oli laatulähtöinen vaikutus kehitysprojektin toimintaympäristöön, jossa avointa viestintää pidettiin mieluisana ja siihen myös kannustettiin.

Avainsanat (asiasanat)

DevOps, mobiilipelikehitys, toimintatutkimus

Muut tiedot (salassa pidettävät liitteet)

Ei salassapidettäviä liitteitä

Contents

Terminology 5
1 Introduction 6
1.1 The DevOps phenomenon 6
1.2 The employer of the research 6
1.3 Motivation for the research 7
1.4 The scope of the research 7
2 Theory 8
2.1 The definition of DevOps 8
2.1.1 Agile and DevOps 8
2.1.2 DevOps culture 9
2.1.3 DevOps automation 9
2.1.4 DevOps as a role 10
2.2 DevOps lifecycle and toolchain 10
2.2.1 DevOps lifecycle 10
2.2.2 DevOps toolchain 11
2.3 Continuous practices 12
2.3.1 Continuous integration 12
2.3.2 Continuous delivery 13
2.3.3 Continuous deployment 13
2.3.4 Other continuous practices 13
2.4 Applying DevOps 14
2.4.1 DevOps adaptation 14
2.4.2 Need for automation 14
2.4.3 Measuring the performance of DevOps 15
2.4.4 DevOps in small organizations 15
2.5 Mobile game development 16
2.5.1 Android and iOS 16
2.5.2 Google Play and App Store 16
2.5.3 Mobile game launch strategy 17
2.5.4 Live operations 18
2.5.5 QA testing 18
2.5.6 Unity game engine 19
3 Research objectives 20
3.1 Purpose 20

3.2 Objectives 21
3.3 Research questions 21
4 Implementation 22
4.1 Action research strategy 22
4.2 Subjects of research 23
4.2.1 Mobile game development project 23
4.2.2 The development team 24
4.3 Methods 24
4.3.1 Group interviews 24
4.3.2 Participant observation 25
4.4 Materials 26
4.4.1 Interview data 26
4.4.2 Observation data 26
4.5 Material analysis 26
4.5.1 Interview material analysis 26
4.5.2 Observation material analysis 27
4.6 Resourcing and scheduling 27
5 Results 28
5.1 Practices 28
5.1.1 Continuous integration 28
5.1.2 Continuous delivery 30
5.1.3 Continuous deployment 31
5.1.4 Automatic versioning 32
5.1.5 Automatic testing 33
5.1.6 Continuous monitoring 34
5.1.7 Continuous improvement 34
5.1.8 Summary of the practices 35
5.2 Tools 36
5.2.1 The planned DevOps toolchain 36
5.2.2 Selected tools to support the practices 39
5.2.3 Overview of the CI/CD pipelines for the game client 43
5.2.4 Continuous integration pipeline for nightly builds 45
5.2.5 Continuous integration pipeline for pull requests 46
5.2.6 Continuous delivery pipeline for iOS 47
5.2.7 Continuous deployment pipeline for remote settings 48
5.2.8 Continuous monitoring pipeline for backend 49

5.3 Culture 50
5.3.1 Quality orientation 50
5.3.2 Open communication 51
5.3.3 Eagerness to improve 51
5.3.4 Craving for automation 52
5.3.5 Favorability towards new tools 52
5.4 Summary of the results 53
6 Discussion 54
6.1 Reliability 54
6.1.1 Reliability of the theoretical basis 54
6.1.2 Reliability of methods and materials 54
6.1.3 Research ethics 55
6.2 Analysis of the research results in respect of the theoretical framework 56
6.2.1 Practices 56
6.2.2 Tools 56
6.2.3 Culture 57
6.3 Conclusion 58
6.3.1 Extensibility 58
6.3.2 DevOps is applicable in mobile game development 58
6.3.3 There is no one way to execute DevOps 58
6.3.4 DevOps is a journey not a destination 58
6.3.5 Appropriate automation 59
6.3.6 Understanding the benefits of Live Ops 59
6.3.7 QA testing is still the backbone of game testing 59
6.4 Suggestions for further research 60
References 61
Appendices 66
Appendix 1. Initial assessment interview with the development group 66
Appendix 2. Initial assessment interview with the steering group 67
Appendix 3. Conclusive interview with the development group 68
Appendix 4. Conclusive interview with the steering group 69
Appendix 5. Observation log sheet template 70
Appendix 6. Keywords and search expressions 71
Appendix 7. Data management plan 72

Figures

Figure 1: Illustration of a DevOps lifecycle 11
Figure 2: CI/CD practices in relation to each other 12
Figure 3: Simplified mobile game development project client-side and server-side structure . 23
Figure 4: Interview groups aligned with the development team 25
Figure 5: The planned DevOps toolchain 38
Figure 6: Continuous integration pipeline for nightly builds 45
Figure 7: Continuous integration pipeline for pull requests 46
Figure 8: Continuous delivery pipeline for iOS 47
Figure 9: Continuous deployment pipeline for remote settings 48
Figure 10: Continuous monitoring pipeline for backend 49

Tables

Table 1: DevOps practices and their prioritized target areas 35
Table 2: The planned DevOps toolchain with new tools separated 37
Table 3: DevOps practices, their prioritized target areas and the selected tools 39
Table 4: Unity Cloud Build targets for Android and iOS 44

Terminology

Android – A popular mobile operating system maintained by Google

Bitbucket – A version control service provider and a hosting platform

Docker – An operating system level virtualization product

Free-to-play – A game that is free to download and play but contains paid content

Git – A version control system for tracking changes in software development

Gitflow – Gitflow strategy is a systematic workflow to using Git version control

iOS – Apple's mobile operating system

Microsoft Azure – A public cloud platform that can be used to host applications

Pipeline – A set of chained processes that are executed in succession

Pull request – A review request in version control before changes are merged to a main branch

Serverless – On-demand service that can be used to execute actions without a separate server

TestFlight – Apple's test distribution service for iOS apps

Toolchain – A set of tools that are used together to perform complex software development tasks

1 Introduction

1.1 The DevOps phenomenon

DevOps is a phenomenon and a trend related to software development lifecycle. DevOps has grown in popularity since the early 2010s. DevOps usually refers to the development and operations practices that aim to streamline the process of software development lifecycle and the automation of recurring activities related to source code change management and the assurance of quality of the changes made in source code before the changes are released into production. DevOps is often considered as a continuation to the Agile methodologies and the aim of DevOps is to extend on Agile methodologies rather than to replace them. Since the term DevOps lacks an exact commonly acknowledged definition, the term is also used as an umbrella term with a broader definition that is referred to as the DevOps culture. DevOps culture is seen as the way the organization functions as a unified extended team rather than set of siloed teams working apart. This approach is said to advance the communication and collaboration between individuals working in the organization and overall resulting in better outcomes for the organization practicing it.

1.2 The employer of the research

The employer of research, Zaibatsu Interactive Oy, is a game and software industry company based in Jyväskylä, Finland. The company was founded in 2014 and its main field of activity is software design and software development. The net revenue of the company was over 1 million euros in the financial period ending in December 2020. In May 2021, the company employed 22 employees in total. The company had not previously used DevOps practices systematically or advocated for DevOps culture before the research, although many of the practices and principles commonly associated with DevOps were already familiar from the context other software development processes. The potential usefulness of DevOps had been recognized by the employer by the observable interest that the partners and clients of the company had towards DevOps. The employer had also recognized that DevOps is a prevailing trend in the software industry and that it might serve as a competitive advantage when utilized by the company itself. (Ultima, 2021.)

1.3 Motivation for the research

Supporting the company's ongoing strategic objectives was one of the key motivators that initiated the research. While the company was aiming to maintain its growth and stability with financially steadier software development, which was usually subcontracting in nature, it also had strategic goals set around scaling its game development activities in the near future. In contrast to software development, game development was more like product development with a substantially higher financial risk for the developing company, as only the published end-product would be capable of generating income, and only if the product were to be successful. In the initial research setting the company already had an ongoing mobile game project in production which was deemed as a prominent subject for research. The presumption of implementing DevOps practices and adapting DevOps culture was that it would provide a comprehensive and cost-effective way to manage a unified team that is both developing and supporting the game project without eventually breaking into separate specialized teams which would make communication between the teams harder and the management of the project more complex. Also, implementing DevOps was viewed as a preparatory measure that should be instituted before the upcoming product launch in order to be ready for scaling the activities around the game, and ensuring the rapid and quality-oriented capability of delivering game updates after a successful launch. Furthermore, the monitoring activities that occur after a release were also deemed important, and it was crucial to be able to automate monitoring and reporting of manifesting issues to the developing team so that issues could be resolved with as little delay as possible.

1.4 The scope of the research

The scope of the research was to study DevOps practices, tools, and cultural impacts of DevOps in a single mobile game development project that was being prepared for release during the research period. The project was chosen for the research because it of its strategic value for the company, and because it offered a tangible area of improvement for the research which was largely unknown for the company beforehand. The idea was to apply DevOps, generally considered a software industry practice used largely by enterprise-level companies, to support mobile game development. The matter of using DevOps specifically for mobile games has been previously discussed by Hargreaves (2016) in his encompassing Game Developers Conference presentation, but other than that and a few dispersed sidenotes the subject has largely evaded the attention of any scientific research.

2 Theory

2.1 The definition of DevOps

DevOps is commonly described as a set of software development practices that aim to streamline the processes between development and operations teams as well as accelerate the software release cycle while maintaining high quality (Mala, 2019, 16; Ebert, Gallardo, Hernantes & Serrano, 2016, 94). However, defining DevOps is not fully straightforward due to the lack of standardization on what exactly constitutes DevOps and what doesn't. In a broad definition DevOps can be seen as a culture or a philosophy that the whole organization is using for elevated collaboration and efficiency (Forsgren, Humble & Kim, 2018), but it can also be seen largely from a technical point of view in the form of practices and tools that help achieve the desired level of automation to enable development and operations to work jointly with lesser emphasis on wider cultural impact (Hüttermann, 2021, chapter 1). Furthermore, sometimes the term is narrowed into a singular role of one or multiple individuals carrying out various tasks throughout the entire software project lifecycle (Filipova & Vilão, 2018, chapter 2). Often the definition of the term varies depending on context and may contain elements from all of the previously mentioned aspects.

2.1.1 Agile and DevOps

DevOps shares many of the features of the Agile methodology. It is argued that DevOps should be seen as something that builds upon the Agile principles, and not as a replacement for Agile (Watts & Kidd, 2017). According to a comparative analysis of Agile and DevOps, Agile has more emphasis on bridging the gap between the developers and the customer whereas DevOps is seen as leaning towards bridging the gap between developers and operations (Brown, 2020). However, a certain consensus seems to exist that DevOps in fact should be conceptualized as an extension to the Agile principles, rather than the two being something separate (Mäkelä, 2019; Watts & Kidd, 2017), which makes DevOps complementary to Agile. In practice this may be lower the effort required to transform to using DevOps practices if the organization has already been familiarized with the Agile principles beforehand.

2.1.2 DevOps culture

Patrick Debois who coined the term DevOps in 2009 has stated that DevOps is in fact a human problem, which also brings forth the cultural significance of DevOps and the implication that DevOps within an organization is always unique to the people practicing it (Davis & Daniels, 2016, 57). Forsgren, Humble and Kim (2018) emphasize the meaning of DevOps as a comprehensive construct that covers the whole organization. They describe DevOps as transformative in nature and see it as a way to increase the performance of the entire organization. DevOps culture is characterized as having favorability towards constant change and adaptation in order for the organization and the individuals in it to improve. Built-in trust and psychological safety inside the organization is in key position in driving innovation, improving the feeling of shared responsibility and supporting advantageous experimental development (Forsgren et al., 2018, 29-40).

Forsgren and colleagues (2018) stress that the organization management has an important role in embracing DevOps culture and that management should enable and drive the adoption of DevOps culture through their own activities and behavior. Therefore company management is encouraged to build trust between teams, advocate for open communication and reward work that enables collaboration and leads to sharing of information and collective learning (Forsgren et al., 2018, 115-127).

2.1.3 DevOps automation

Automation that reduces repetitive manual work is considered as a crucial part of DevOps (Forsgren et al., 2018, 41-42; Ebert et al., 2016, 95). Continuous integration, continuous delivery and continuous deployment are software development practices that aim to validate that changes introduced to a software are functional before those changes are combined to the software and eventually released (Ebert et al., 2016, 95-96). Although the previously mentioned continuous automation practices do not originate with DevOps, they are commonly considered as essential DevOps practices. High degree of automation aims to reduce manual labor, decrease the number of issues and bugs and as a result increase the overall quality of the software. This kind of automation also enables faster release cycles for the software since time-consuming processes related to managing the quality and releasing the software are largely automated (Ebert et al., 2016, 95).

2.1.4 DevOps as a role

The term DevOps is also being used to describe a role of an individual. For example, the title DevOps engineer might be used as a titular job description of an employee working both on development and operations (Filipova & Vilão, 2018, chapter 2). However, this kind of titular role, either used for an individual or for a department, has met some critic since it comes across as if the DevOps practices and applying the culture are off-loaded to the hands of the few who carry the titular role, instead of a wider team, which is seen as conflicting with the idea that DevOps aims to unify the team with shared procedures (Hüttermann, 2021, chapter 1). Nevertheless, when the DevOps roles and responsibilities are expanded to cover the entire team, as in the DevOps team, the label is more fitting and more closely in line with the cultural aspects of DevOps.

2.2 DevOps lifecycle and toolchain

2.2.1 DevOps lifecycle

The DevOps lifecycle consists of phases that represent DevOps procedures as a repeating process. There are typically 6 to 8 separate phases defined in the lifecycle, sometimes even more. Service providers offering DevOps tools tend to provide their own specification of the DevOps lifecycle and its phases with an emphasis on the selection of services they supply, which contributes to the debatable accurate definition of a one DevOps lifecycle (“Atlassian DevOps”, n.d.; “GitHub DevOps”, n.d.; “GitLab DevOps”, n.d.). Consequently, the DevOps lifecycle should be seen as a demonstrative depiction of the most common practices in the development and delivery process of a software, not as a strictly defined standard. An illustration of a loop representing the DevOps lifecycle depicted by GitHub is seen in figure 1 (“GitHub DevOps”, n.d.).

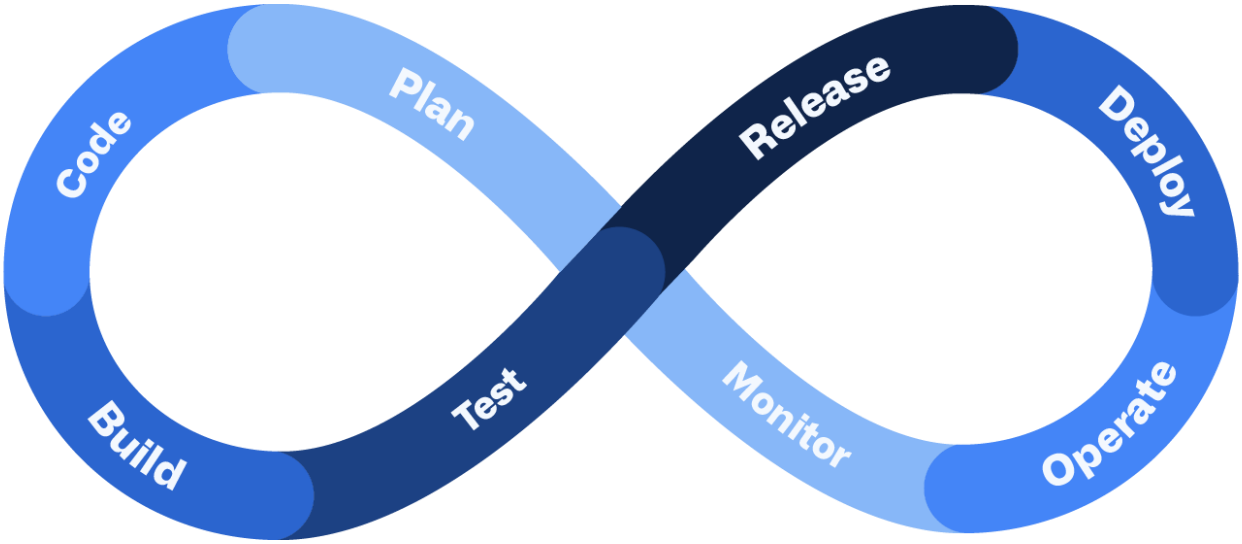

Figure 1: Illustration of a DevOps lifecycle

Since the phasing seen in figure 1 is widely popularized, as seen in various research reports and technical articles (Gokarna, 2020, chapter 3.4; Phulare, 2020; Storbakken, 2020), it is considered as the "traditional" DevOps lifecycle in the context of this research from here on out. The traditional DevOps lifecycle consists of phases such as planning, coding, building, testing, releasing, deploying, operating, and monitoring. Commonly the planning, coding, building, and testing are seen as development phases and releasing, deploying, operating, and monitoring are seen as operations phases (Storbakken, 2020). Although, in an integrated DevOps team the roles are expected to be less static since the lines between different phases and responsibilities might be blurry in practice.

2.2.2 DevOps toolchain

The DevOps toolchain is a set of tools that aid the development and operations team in ensuring the smooth execution of DevOps procedures and achieving the required level of automation to alleviate the repeating processes (Krohn, n.d.; Hiremath, 2021). The toolchain is often presented in a similar manner to the DevOps lifecycle, but supplemented with the tools that are supporting each phase of the lifecycle (Little, 2019; Phulare, 2020). Designing a DevOps toolchain of a software project is a wholesome undertaking and requires careful planning and understanding of the entirety of the software project as well as the business needs that fuel the project. Selected tools should be compatible with each other so that the smooth flow of work and the integrity of toolchain is secured (Krohn, n.d.). The demand for a concentrated set of DevOps tools is apparent in the emergent supply of DevOps tools by services providers ("Atlassian DevOps", n.d.; "GitHub DevOps",

n.d.; “GitLab DevOps”, n.d.). However, the process of selecting the tools for a toolchain in a given project should be based on the distinctive requirements of the project and all-in-one solutions might not be able to provide all the necessary tools for individual project needs.

2.3 Continuous practices

Continuous integration, continuous delivery, and continuous deployment, or in condensed form CI/CD, are software development practices that aim to automate various areas of testing, building, and releasing of software. While the practices are seen as key components of modern software development, they are also essential in practicing DevOps (Hall, n.d.), along with some other continuous practices. The CI/CD practices are closely related to each other but they each represent a different level of automation, as illustrated in figure 2 (Pittet, n.d.).

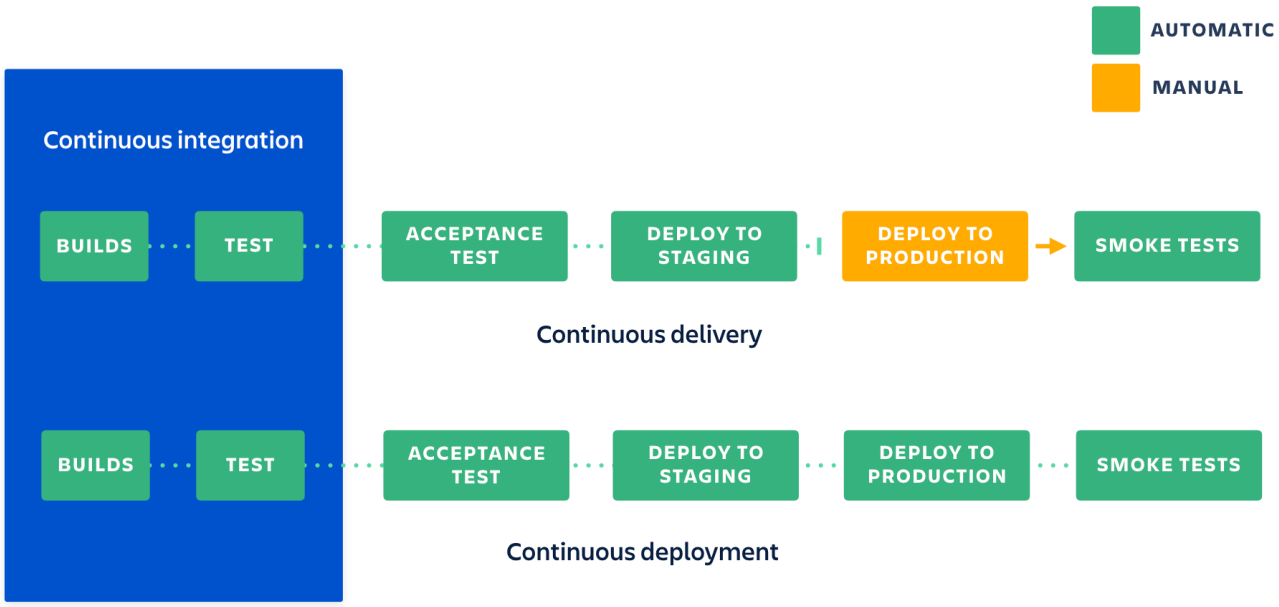

Figure 2: CI/CD practices in relation to each other

2.3.1 Continuous integration

Continuous integration is a software development practice in which changes are combined into the software as frequently as possible, so that the changes can be tested, and issues can be uncovered as soon as possible. Continuous integration usually consists of build automation and execution of automated tests. The end result of an integration attempt is either a successful integration, if the tests are passed, or an unsuccessful integration if the tests are not passed. Integration is

then followed by a report which states the results of automated tests. In case the integration fails the developer is notified to take action in order to fix the issues before attempting the integration again. (Pittet, n.d.)

2.3.2 Continuous delivery

Continuous delivery is an extension of continuous integration. In continuous delivery the acceptance testing is usually automated and following a successful acceptance test the software is released to a testing environment that closely resembles the production environment. However, deploying the software to the production environment is still a manual process that requires action from the developer. In continuous delivery it is typical to have automated smoke tests executed when the software is released to production, so that the most critical functionalities of the software are fully tested before going to production. (Pittet, n.d.)

Forsgren and colleagues (2018, 41-58) enforce five key principles that enable successful continuous delivery, which consist of building quality in, working in small patches, automating repetitive work, improving relentlessly and taking shared responsibility. In order to properly support continuous delivery they also stress the need for comprehensive configuration management and continuous testing that is ideally automated.

2.3.3 Continuous deployment

Continuous deployment is an extension for continuous delivery, in which the deployment to production is also fully automated without manual intervention required from the developer. Continuous deployment is seen as the most complete form of software automation, but it also sets the most requirements for comprehensive automated testing and requires extensive code coverage in order to ensure that all functionalities are working perfectly. (Pittet, n.d.)

2.3.4 Other continuous practices

Sometimes in the context of DevOps other continuous practices are introduced as well, such as continuous development or improvement, continuous feedback and continuous monitoring (Hall, n.d.). These are all essential practices that match with the phases of the DevOps lifecycle.

2.4 Applying DevOps

Adapting DevOps is a large undertaking for an organization of any size and initiating the DevOps transformation requires support from organizational management in order to succeed. Applying DevOps requires proficient technical knowledge of the tools and practices it entails, but also cultural willingness for change that spans the whole organization.

2.4.1 DevOps adaptation

Adapting DevOps should be seen as a continuous journey and the transformation to fully leveraging DevOps should be approached incrementally rather than as a large-scale change over a short period of time (Hall, n.d.). Forsgren and colleagues (2018, 115-127) highlight the importance of leadership in inspiring and enabling the DevOps transformation by removing obstacles that obstruct adapting DevOps and by creating a safe space for continuous learning and improvement in the organization. In order to initiate the DevOps adoption both the organizational management and the teams inside the organization require an understanding of the added value that DevOps is going to bring to the organization and the teams working inside it (Rajkumar, Pole, Adige & Mahanta, 2016). Otherwise the efforts to adapt new practices might turn out to be difficult to justify and the efforts are likely to fail. This might be especially challenging in large organizations that have already accustomed to working with separated departments for development and operations (Rajkumar et al., 2016). However, as Hargreaves (2016) has pointed out, the adaptation of DevOps should be viewed more as a different and more compact alignment of already existing departments rather than as something completely new and different that requires all-encompassing changes everywhere.

2.4.2 Need for automation

DevOps is often seen as highly favourabile towards to the use of automation, to the extent that it has been stated that the goal of DevOps is to “automate everything” (“DevOps Automation”, 2019). However, automation itself is not the end-goal of DevOps since the investment to automation should always stem from an actual need and it should be adapted to its purpose. Plainly, automation should save more time and effort that is invested into implementing it, and this should be estimated and measured by identifying the processes and work stages that are arduous and time consuming (“DevOps Automation”, 2019).

2.4.3 Measuring the performance of DevOps

Measuring the performance of DevOps is crucial to the businesses that pursuit DevOps, but also as important to the teams that are practicing it on a daily basis, because measurement enables clearer understanding of the areas that require mending, and identifying the issues allows overcoming them through improvement. Forsgren and colleagues (2018, 11-29) have defined four key metrics that enable measuring the performance of DevOps which are: deployment frequency, lead time for changes, mean time to recovery and change failure rate. The same four key metrics are also highlighted in the State of DevOps reports (Forsgren, Smith, Humble & Frazelle, 2019; Brown, Stahnke & Kersten, 2020). However, remarks have been made that organizations should not rely on predefined metrics for too long, because every metric is somewhat skewed to the point that people tend to find ways to make metrics appear better than they truly are, even subconsciously (Paul, 2015). Okes (2013, 3-4) has also stated that organizations that use metrics to measure their performance should define their own success metrics, instead of relying on predefined metrics.

2.4.4 DevOps in small organizations

Since the manifestation of DevOps the discussion surrounding it has been primarily focused on large enterprise organizations. Yet, the solutions that DevOps provides could benefit any organization regardless of their size when adapted accordingly. Hargreaves (2016) has also remarked the steep enterprise orientation of DevOps related documentation and tooling. Swartout (2012, 23) argues that small software businesses, out of necessity due to their small size, are usually already practicing behaviors that are closely in line with the sentiment of DevOps, since these organizations don’t have a strict division of teams and usually a small team of people is responsible for an entire software project. However, Swartout (2012, 24) continues to state that while small software businesses appear to practice behaviors similar to DevOps, they might be cutting corners and the approach is not as systematic when it comes to change management and version control, and generally there is a heavy reliance on manual activities instead of using automation. Schaub (2019) has argued that DevOps can be suited for an organization of any size, and that the transformation to using DevOps is probably easier for organizations that are still small and have low governance. Additionally, Schaub (2019) has pointed out that DevOps is becoming a necessity for small organizations as well, in order for them to remain competitive against larger competitors.

2.5 Mobile game development

Mobile game development is a rapidly evolving field of game development that is highly competitive and constantly growing. The most prominent mobile platforms are Android and iOS, while the largest publishing platforms are Google Play and App Store, on respective platforms (Lewis, 2020). The mobile games industry measured by revenue was estimated to account for 52% of the global games market, with revenues of over 90 billion US dollars, according to a 2021 forecast. This makes mobile gaming the biggest segment in the global games industry, and it has been estimated that the growth of mobile gaming will only continue to grow in the future (Wijman, 2021). Mobile games are increasingly becoming products that have a lifetime of years during which they need to be supported and updated constantly to keep the games relevant for players in order to remain competitive (Leung, 2017).

2.5.1 Android and iOS

Google's Android operating system and Apple's iOS operating system are the two largest mobile platforms, which covered around 99% of the global mobile device market in 2020 ("IDC Smartphone Market Share", 2021; "StatCounter Mobile OS Market Share Worldwide", 2020). When measured by units sold, the share between Android and iOS was approximately 85% and 15% in 2020, respectively (Goasduff, 2021; "IDC Smartphone Market Share", 2021). However, when measured by browsing activity, a larger share has been estimated for iOS, up to 28%, while the share for Android was averaging at 72% ("StatCounter Mobile OS Market Share Worldwide", 2020). While iOS has narrowly been the dominant platform in the United States ("StatCounter Mobile OS Market Share In USA", 2020) and has had a relatively high usage share in the western markets as well, Android was distinctly the dominant platform in India with almost 96% of the active devices being Android ("StatCounter Mobile OS Market Share In India", 2020), which is likely to be explained by a higher availability of Android devices with a lower pricing.

2.5.2 Google Play and App Store

Google Play for Android and App Store for iOS were the biggest publishing platforms for distributing mobile apps and games in 2021 and have remained so for many years. Both Google Play and App Store are widely available globally, however, a notable exception is China, where Google Play is not available due to regulations, leaving the market saturated with multiple different app store

operators. On iOS, App Store remains the only available legitimate distribution option globally. (Dogtiev, 2021.)

Google Play and App Store share similarities in their developer tools and procedures, but not without some differences. Both Google Play and App Store require the developer or organization to register to use their services and distribute apps using their platforms, bound by their terms. Both platforms have commission fees that are tied to the sales of an app on their platform, which was 30% in 2020, and 15% for businesses that make less than 1 million in sales on the platforms starting in 2021 (Leswing, 2020; Singh, 2021). Google Play and App Store require the developer to use platform-specific signing identity for the app artifacts that are distributed on their platforms in order to ensure their origin and safety for distribution. On iOS the distribution of builds is more complex than on Android, since even development build artifacts require signing, and distribution to non-registered devices requires uploading build artifacts to Apple's TestFlight service (Katz, 2018, 31-38). On Android development builds may be signed with a non-restricted debug key and uploading to Google Play is not required, which makes testing during development more straightforward ("Build and run your app", 2021). However, for platform-specific functionalities signing is needed on Android as well, although uploading to Google Play is not required for testing, but only for distribution. Additionally, both Google Play and App Store have review policies in place which require new apps and updates to existing apps to be reviewed according to platform-specific guidelines, which adds a mandatory delay to shipping app updates to end-users ("App Store App Review", n.d.; "Understand Play policies", 2020).

2.5.3 Mobile game launch strategy

Evaluating development phase mobile games with the use of analytical data has become an industry standard, and certain key metrics are expected to be achieved in order for a mobile game to become successful. Game industry experts (Telfer, 2016; Lancaric, 2018) have stated that a mobile game should be tested on actual players, usually with limited availability, as soon as possible in order to use the gathered data for optimizing the game to become more performant, and in order to determine whether the production is worthwhile to be continued at all. A soft launch is a test launch targeted at a limited audience in order to test the performance of a game before a prospected global launch. Soft launching a game is a common strategy at present that helps the developers of a game understand their audience better and lower the risk of failure before going

global. Soft launch might be a period of several months during which the game is tested in different countries and on different platforms with frequents updates to the game in order to produce new analytical insight. In case a soft launched game does not meet the metrics required, a very common scenario is to discontinue the game production, rather than committing to a global launch because of the financial risk. (Telfer, 2016; Lancaric, 2018.)

2.5.4 Live operations

Live operations, or Live Ops, refers to game content updates that aim to improve player experience and game metrics through frequent updates and adjustments that are based on player data and analytics (Jowsey, 2018; “Mobile Live Operations Best Practices”, n.d.). While live operations may refer to the act of creating content updates and adjustments at large, the term is also used to refer to certain type of updates that are served to the player without game client updates, effectively meaning that the player does not necessarily have to download an update to the game through the app distribution platform each time adjustments have been made to the game (Leung, 2017). Instead, the live operations team may use certain kind of tools that are built into the game client, that enable operating certain aspects of the game remotely (Raeburn, 2017). This kind of remotely controllable live operations enables adjusting the game almost instantaneously, instead through app updates which reach the players much slower. The approach of using remote tools also enables the development team to keep improving the game continuously without having to force continuous game client updates on the players. Since app stores require a review for each app update, delivering updates only through app store client updates would be time-consuming, and not only for the development team, but for the players as well.

2.5.5 QA testing

Quality assurance, or QA, is an ongoing testing process during a production of a game, as new functionalities and features are tested by a QA team through means of manual play testing (Chandler, 2014, chapter 2.7). A QA team tests a game against a test plan which contains test cases for the functionalities and features of a game that require testing (Chandler, 2014, chapter 1.5). Defects and issues uncovered by the QA team are reported to the development team, and once the development team has alleviated the issues, they are reported fixed by the development team,

and hence returned back to the QA team so that the absence of issues may the confirmed by testing (Chandler, 2014, chapter 1.5, chapter 2.7). Games in general are considered hard to test since they have complex logical structures that are hard to cover thoroughly by tests (Schultz & Bryant, 2017; Kaasila, 2015). Consequently, manual QA testing is seen as a cost-effective way to test such complicated systems, and often the method used is in fact black box testing, where the tester has no access to the source code while testing (Schultz & Bryant, 2017, chapter 6).

2.5.6 Unity game engine

Unity is a cross-platform game engine created by Unity Technologies. In addition to game development the Unity game engine has been used in other industries as well, such as film, engineering, construction and automotive industries ("Unity solutions", n.d.). Unity is one of the most popular game engines used in mobile game development, with estimates stating that up to 71% of the top 1000 mobile games in 2020 were made using the Unity game engine ("Unity company presentation", n.d.). While Unity is primarily known for its game engine, there are also many other tools and features Unity offers to support developers, which are either served as separate products, or as part of the Unity licensing. Namely, Unity offers Cloud Build service through its Unity Teams Advanced licensing, which enables cloud-based continuous integration with hosting and distribution of builds through the service ("Unity Cloud Build", n.d.). Similarly, tools for cloud-based diagnostics and analytics are also offered ("Unity Cloud Diagnostics", n.d.).

3 Research objectives

3.1 Purpose

The purpose of the research was to gain insight on the usability of DevOps specifically in mobile game development and in the use of the employer of the research, as DevOps was not very well known for the employer company beforehand. The intent was to learn which DevOps practices are best suited to support the needs of mobile game development, but also to provide information on the feasibility of the selected tools through practical means and experimentation, in order to understand what exactly is possible and what kind of investments would be required. The idea was to advance the use of DevOps during the research as extensively as reasonably possible and form a basis for further use of DevOps in the project. The employer of the research had expressed a desire to eventually expand the learnings of the research, if applicable, also into the use of other game development projects in the company, which also brought forth the need to study the cultural impact of DevOps to be able to evaluate the influence of DevOps on a development team, and in the long term in order to understand the wider impacts on the whole organization as well.

The employer company had previously used an on-premises continuous integration solution for creating build artifacts for mobile game projects. The existing solution was configured on a local machine that physically resided at the office of the company. The system was based on Jenkins, which is a commonly used continuous integration server that can be tailored to suit many kind of automation requirements (Riti, 2018, chapter 5). However, it had proven to be difficult to keep the system and the plugins related to it up to date, to the point that the system had to be maintained regularly in order to be usable at all. The physical nature of the build server setup was seen as an issue going forward, since also the hardware would have to be replaced at some point before it was going to become obsolete. This had imbued the desire to find a cloud-based solution for continuous integration and continuous delivery that would have low maintenance requirements and would be automatically up to date to suit the specific needs for building and distributing mobile game projects. Additionally, it was important to understand that iOS build distribution would be the constraint that would set the base requirements of the desired solution since iOS building and distribution requires that the build server runs macOS operating system.

3.2 Objectives

The main objectives of the research were focused on the practices and tools that would enable applying DevOps for the benefit of mobile game development, and more specifically for the benefit of a single mobile game development project that was in production, and being prepared for release, during the research period. Furthermore, it was important to study the cultural impact that applying DevOps had on the mobile game development team that was involved in creating the said project. At premise it was important to be able to prioritize the practices and tools that would be implemented during the research period, since it was crucial to be able to identify the bottlenecks of the current workflow and to focus on improving around those impediments in order to produce beneficial results for the research, and thereby lay ground for future developments.

3.3 Research questions

To further clarify the objectives of the research, a set of three (3) research questions were formed, to be approached in the following order:

1) Which DevOps practices are needed to most benefit the mobile game development project?
2) Which DevOps tools are needed for applying the practices in the mobile game development project?
3) How does applying DevOps impact the culture of the mobile game development team?

4 Implementation

The research was implemented using an action research strategy where the researcher was actively involved in the mobile game development project and within the development team which were being researched. The research was conducted using interviews and observation as the main research methods for gathering the research material that was then further analyzed.

4.1 Action research strategy

The research was implemented in the form of an action research, where the researcher was actively involved in the mobile game development project and in close contact with the development team, that were being researched. The reason for selecting action research as the research strategy was to enable initiating the DevOps transformation, and to gain practical insight on the use of DevOps in a project that was formerly not enforcing the use of DevOps practices or culture. The nature of action research is in fact driving transformation, and the aim is to both pursue change and to study that change when it actualizes (Kananen, 2014, 11-12). In an action research the subjects of the research are aware of the research taking place, as the researcher has an active and visible role in the research due to the transformational nature of the action research (Ojasalo, Moilanen & Ritalahti, 2009, 61).

There are several alternative phasings presented for action research. Rothwell (1999) has described action research in eight phases, which are mainly consistent with the more condensed six phases that Kananen (2009, 2014) has since presented. Similarly, Ojasalo, Moilanen and Ritalahti (2009, 60) have described action research process in four phases that repeat in a cycle. All of the previously mentioned phasings are quite similar and share the common structure that begins with defining the issue and planning a solution to alleviate it, which is then executed and evaluated before beginning a new cycle with the newly acquired learnings. Hence, action research has a cyclic pattern and finding successful results is based on repetitive experimentation and thorough reflection of the learnings of each cycle is important. The four-phase pattern that Ojasalo and colleagues (2009, 60) described was used in this research since it was simple and to the point. The repeating phases that were followed were planning, acting, observing and reflecting.

4.2 Subjects of research

4.2.1 Mobile game development project

The mobile game development project that was researched was an ongoing project that was being prepared for release during the research period. The game project was a so-called free-to-play action game with an online multiplayer, with the target release platforms being Android and iOS, and the target publishing platforms being Google Play and App Store.

The project had already been active for almost a year before the research started, so many of the technology decisions regarding the project had already been determined before the research began. The game project as a whole comprised of a game client and a backend. The game client was developed with the Unity game engine. The backend consisted of separate components such as TCP game servers, HTTP servers, and a SQL database. The TCP and the HTTP servers were developed with ASP.NET Core. The backend was containerized in Docker containers which were hosted in Microsoft Azure. The simplified structure of the project as a whole is visualized in figure 3.

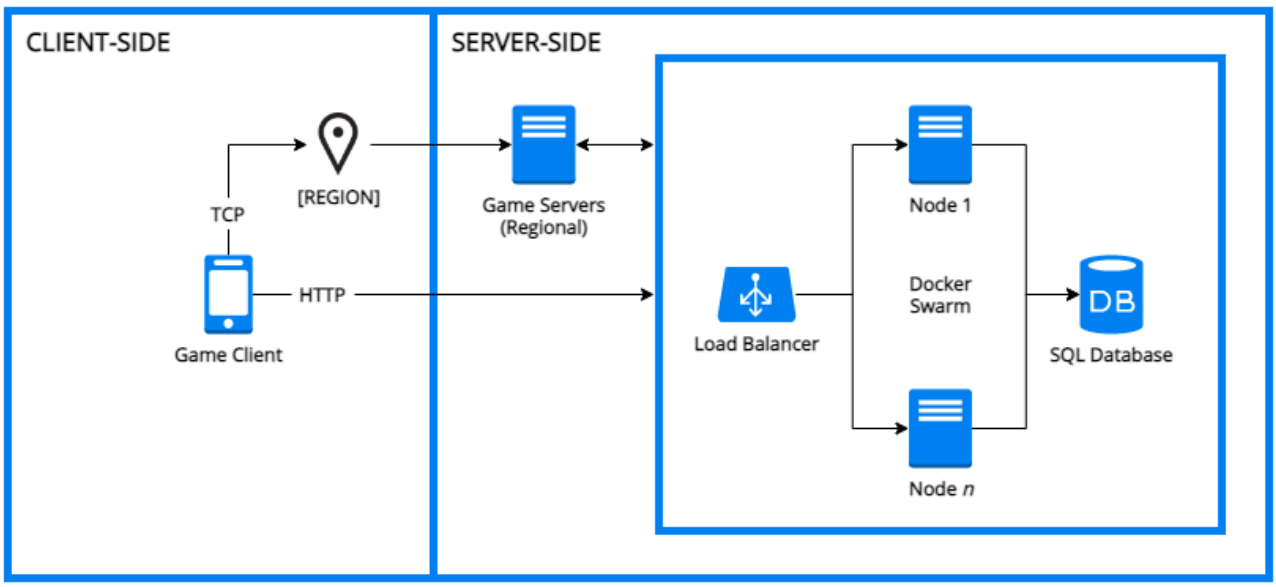

Figure 3: Simplified mobile game development project client-side and server-side structure

The project source code was stored in Bitbucket Git repositories, in three separate repositories: game client, HTTP server and TCP server resided in different repositories. Gitflow strategy was used in the project, but it was not fully enforced at the beginning of the research period.

4.2.2 The development team

The development team consisted of 11 people on average during the research period, with a few people moving between different projects during the research period, but mainly the development team structure was quite stable during the research period. The team composition had a project manager, 3 artists or designers, 6 programmers and a QA tester. Some of the previously mentioned roles had some overlap with different roles as the structure of the team was not as strictly defined in reality. For example, the managerial role was sometimes shifted between the project manager and the lead designer, and the QA tester was in actuality also doing programming and most of the programmers were at least partially participating in QA activities.

4.3 Methods

The methods used in the research were interviews and observation, that were executed in the form of group interviews and participant observation. The idea of the interviews was to deepen the understanding of the subject and to study the effects that had occurred during the research period, while observation aimed to support the validity of the research with information collected in a natural setting on a regular basis. Therefore, the chosen research methods complemented each other.

4.3.1 Group interviews

Group interviews were held in two occasions with two separate interview groups, which were the development group, and the so-called steering group. The development group consisted of individuals representing the core development team, who were in practice responsible for developing and operating the game project on a day-to-day-basis. The steering group consisted of individuals that represented the business perspective or had a managerial role in the project. Abovesaid grouping was established only for the means of the research and did not reflect the internal structure of the team in regard of everyday activities, since in practice there was only a single *development team,* or plainly *a team,* that encompassed all the individuals working on the project regardless of their position in the team. However, for research purposes it was insightful to interview the steering group and the development group in isolation to understand the business perspective and the development perspective and their viewpoints on DevOps separately. A total of 10 people from the development group and the steering group participated in the interviews. A referential

Venn diagram representing the alignment of the two interview groups with respect to the actual mobile game development team is visualized in figure 4.

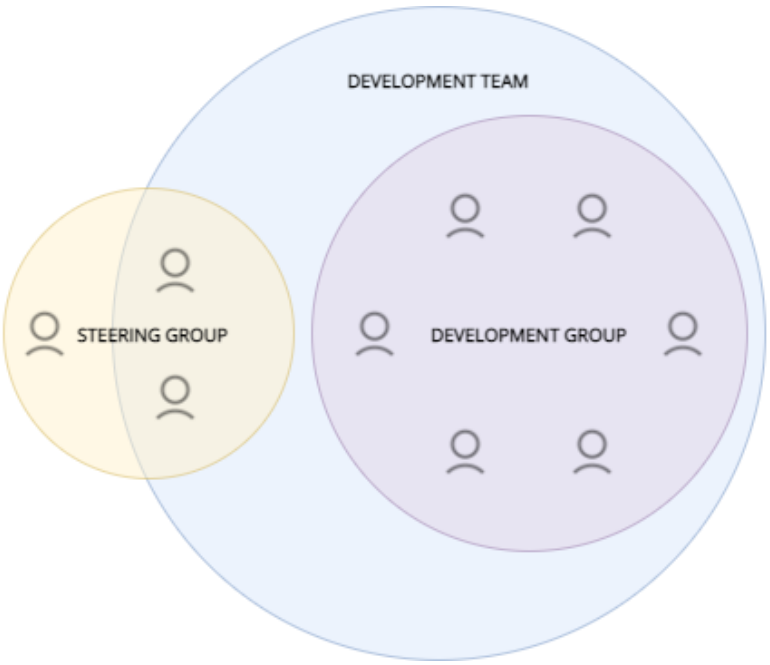

Figure 4: Interview groups aligned with the development team

Both interview groups, the development group, and the steering group, were interviewed twice: at the beginning and at the end of research period. The first interview was an initial assessment interview aiming to establish a baseline where to get started and to set the initial targets for the research, to be further defined during the action research period. The conclusive interview at the end of the research period aimed to reflect the transformation that had occurred throughout the research period and acted as a retrospective to the whole research.

4.3.2 Participant observation

Participant observation is a method of research where the researcher observes the research subjects in close proximity. The subjects were aware of the research taking place and knew they were being observed. In participant observation the researcher may take part in activities that are being researched, as was the case in this research as the transformation to using new practices and tools was desirable. The observation was executed systematically and regularly as part of daily team activities. The observation technique used was a form of systematic observation where the observation method was structured and analytical. (Ojasalo et al., 2009.)

4.4 Materials

The material that was collected for the research consisted of interview data and observation data. The material was gathered during a three-month research period taking place between February and early May 2021.

4.4.1 Interview data

The interview data was collected from a total of four interview sessions that were recorded, and transcribed into text in a proposition level accuracy, meaning that the key message of each interview statement was further analyzed and processed to form the results of the research. Interview data is discretionary data aiming to collect in-depth outlook on the research. (Kananen, 2014, 87.)

4.4.2 Observation data

The observation data was reported to an observation log sheet on a regular basis. Since the observation technique was structured, the log sheet structure was also predefined in order to analyze and process the data in a systematic manner (Kananen, 2014, 83-85). The observation data was collected during a three-month period at least twice a week.

4.5 Material analysis

All in all the research was highly qualitive due to the actionable nature of the research. This means that the layout of the research was quite open-ended. Therefore, actions to ensure the validity of the material analysis were taken into account from the beginning.

4.5.1 Interview material analysis

To support the validity of the interviews, different thematic viewpoints, notably the development and operations viewpoints, were presented in the interviews. Also, the grouping into the development group and the steering group was established so that different viewpoints could be evaluated without peer pressure possibly limiting the authenticity of the answers of either group. The interview material that was collected was transcribed into text at a proposition level accuracy, meaning that the key message was abstracted, and then it was further processed, anonymized, and analyzed in text format.

4.5.2 Observation material analysis

Observation data was collected in text format into a log sheet by the researcher. At the end of the research the log sheet excerpts were synthesized to form groupings with thematic correlations with similar data. This data was then brought together with the interview data in order to analyze the similarities and differences of the produced material as a whole.

4.6 Resourcing and scheduling

The research period was approximately a three-month-long period lasting from February 2021 to early May 2021. The scheduling was set to this time period because it was essential for the mobile game development project that the DevOps transformation could be initiated before the upcoming release of the game. The researcher was involved in the mobile game development project throughout the research period both in the roles of a developer and a technical lead, which offered a broad perspective into the day-to-day activities inside the development project. The researcher role was not unattached from the normal daily role during the research, which meant that the observation was conducted simultaneously among other daily activities. The interviews were scheduled to the beginning and to the end of the research period and the development team was involved in the interviews. Due to the transformational nature of the action research, the whole development team was actually involved in the research at least partially. In practice this meant that the practices and tools that were prioritized to be implemented were also tasked to the development team along with other development tasks. Hence, the development team and their participation in the research was essential to produce the outcomes. However, in regards of prioritization a higher prioritization was put to game-related feature tasks in case resources were scarce and prioritizations had to be made. While there wasn't a predefined resource or time budget set to the research, it was mutually agreed that the research should not obstruct the game development timeline which in turn had been agreed upon earlier.

5 Results

The results of the research are divided into three themes that match with the three research questions: practices, tools and culture. The results are based on the findings of the group interviews and the observational data that was gathered throughout the research, and on the concrete outcomes that were produced throughout the research especially in relation to the tooling.

5.1 Practices

The first research question was: **Which DevOps practices are needed to most benefit the mobile game development project?** To answer this question, the initial assessment interviews with both the development group and the steering group were in key position, since they laid out the foundation for the upcoming prioritization that was used for evaluating the most important practices that would most benefit the development team and the mobile game development project. The continuous observation and the iterative action research cycles then further supported the initial considerations and prioritizations. As to what was considered as being a "DevOps practice" in this context was any practice that was seen to advance the DevOps initiatives. This chapter introduces the practices that were deemed most beneficial to the mobile game development project, with the remark that the prioritization was also driven by both the scheduling and resourcing of the project, which meant that some of the otherwise beneficial practices had to be excluded from the scope of the research due to these limitations. The excluded practices that were deemed noteworthy are considered and discussed in the research conclusion.

5.1.1 Continuous integration

Supporting continuous integration was seen as critical to form the basis for any proper automation. At the start of the research period nightly builds were already in use in the mobile game development project using the Unity Cloud Build service, and while that was filling the daily minimum requirement for external QA testing, it was not enough to support the internal development processes and internal QA testing properly. The biggest concern about nightly integration pace was that the possible issues would be uncovered basically almost a day late, and not at the time of the integration or before it. If problematic code was passed to the development or production trunk of the project, it meant that the main branch was broken for everyone in the development

team, which was something that the team wanted to avoid because it was a waste of time and resources. While a more frequent integration was already possible with the Unity Cloud Build service, it was not desirable to use it because the building was quite slow if the queue started to form during the busy hours of the working day, and it didn't remove the issue that problematic code could still be integrated into the main branch altogether, and the issues would only be uncovered when they would already be in the main branch.

To overcome the issue that problematic code or broken features would get merged into the main branch, it was ideated that each of the newly opened pull request branches should be built in isolation and the resulting build should be linked to the pull request once ready. This was to make sure that in addition to the pull request code review the build would also get tested on a real device, and that the code would in fact compile successfully. It was agreed among the team that for each pull request to be approved a successful build had to be produced, and that the build had to be tested on a device in order for the pull requests to get approved. Beforehand, test builds for individual branches would have been made manually, which would have been time-consuming and hence was often avoided, which would result in features not getting properly tested before integration. The pull request build pipeline seemed like a resourceful idea because it in a way overcame the issue that Unity Cloud Build wasn't that fast when compared to manual building. Since pull requests weren't reviewed immediately after they were opened it left some leeway for the build to finish for each pull request. The usual rate of reviewing pull requests was either each morning when the developers were starting their workday before starting new tasks, or after previous tasks had been completed during the workday, which resulted in at least a few hours before new pull requests started to be reviewed, which was usually enough for builds to be completed even if there was a queue of multiple build requests on the build machine. In case the pull request pipeline wasn't able to produce a successful build or changes were later added to a pull request, it was deemed important that the build pipeline would also activate when the pull request was updated with a commit.

While the project was developed for two target platforms, Android and iOS, for automatic building purposes it was seen that automatically triggering the Android build for each pull request was enough at the base level. However, the ability to also order the iOS build manually was seen as important for features that were platform dependent. The difference between the build times for

Android and iOS was quite drastic, since the Android build would normally take from 30 to 45 minutes to be built, iOS builds took around 1 hour to 1,5 hours to complete. The Unity Cloud Build queue ja cooldown times were additionally added to these times, so the reasoning behind not always producing the iOS build was quite understandable, although in a perfect world it would have also been a candidate for full automation.

5.1.2 Continuous delivery

While the pull request pipeline that was covered in the previous section could already be seen as somewhat fulfilling the continuous delivery practice, since it was capable of delivering and distributing build artifacts, its main purpose was to aid the continuous integration process, i.e. support the quality-driven introduction of changes to the main branch in the version control system. Continuous delivery was considered to fulfill its definition of the delivery only in the context of delivering the build artifacts to an environment that resembles the production environment. In the context of the game client it meant either Google Play or App Store since those were the target publishing platforms for the game. The existing Unity Cloud Build pipeline was already able to deliver build artifact links to Slack, which was used for instant messaging by the development team.

It was clear from the start that the iOS building and distribution would be the constraint that would determine the requirements for the tooling in regards of the game client. As already hinted in the previous section, the iOS building was much more time-consuming than Android building. On top of that the iOS builds could only be created on a machine running macOS operating system, which alone set some hard minimum requirements for the CI/CD server. The goal was to be able to distribute the game client builds to the Google Play internal testing track on Android, and to TestFlight on iOS. The first priority was to automate the iOS delivery pipeline and the Android delivery pipeline was only considered secondary, since the process was much more straightforward for Android even when conducted manually. However, automating both automation pipelines was seen as desirable at least at some point in the future.

As continuous delivery is an extension to the continuous integration, it was seen that the continuous delivery pipeline should be able to create build artifacts for publishing platform distribution from the development and production branches of the game client repository on developer re-

quest. Since all the content of these main branches was to be secured by the previously mentioned continuous integration quality-assurance procedures that would cover all the pull requests it was surmised that this would already be enough to cover the quality of the main branches, and no further testing phases should be required for the continuous delivery pipelines.

5.1.3 Continuous deployment

Continuous deployment was initially seen as almost unreachable in the context of the research, as it was known to require a very high level of code coverage in order to be viable and trustworthy enough to be approached at all. In the context of the game client and backend continuous deployment had to be ruled out. However, in the context of the remote settings configuration that both the game client and backend were using as a global configuration source, it was deemed simple enough to become a reality. Basically, the remote settings configuration was a single JSON file that contained global settings that the game client fetched each it was launched, and similarly it was used by the backend upon certain requests. The remote settings file was in actuality located in the backend, and initially it was updated by the developers using an exposed JSON visual editor through the browser. However, this approach was deemed very error-prone since there was no proper validation or version control for the remote settings at this point. Once the settings were updated through the editor and saved, there was no way to return the previous version, which was a very problematic factor in the usage of the tool.

To overcome these issues, it was planned that the remote settings configuration should be placed in separate Git repository, where it would be versioned and handled similarly to the source code in other project repositories, i.e. enforcing the use of Gitflow strategy which would mean that the changes should be introduced through the use of pull requests that were then peer reviewed and validated before merging the changes to either development or production branches. The way that this was envisioned was that the JSON contents should be tested and validated automatically using some sort of JSON validation utility or a predefined schema, or the combination of both, before the contents would be considered deployable. Then, once the developer would open a pull request the JSON contents would be automatically validated, and after a successful validation and an approval from a peer review the pull request could be merged either to the development branch, where it would be automatically uploaded to replace test environment remote settings

during integration, or to the production branch, where it would be automatically uploaded to replace the live environment remote settings during integration.

5.1.4 Automatic versioning

Automatic versioning was realized to be essential basically regarding any of the previously mentioned continuous practices. Considering the game client, both Google Play and App Store have hard requirements for always providing unique incremental build numbers to the platform in order for build uploading to be successful. In Google Play the version code of each uploaded Android build has to be higher than the version code of the previous builds. In a similar manner in the App Store the bundle version has to be incremented to be higher for each uploaded build. While this seemed quite trivial at premise, it turned out that the incrementation could not be easily be automated using version control since the incremented number should be passed around between the version control and the build server in order to work, so it was quickly realized that the version incrementation should be handled by the build server itself and the build server should be able to determine which was the last uploaded build number in order to increment it correctly. Finally, it was outlined that the build pipeline number would be the most sensible reference point for incrementing the build number, as the pipeline number was always incremented and using it directly didn't require any separate solution for storing the information about latest build numbers.

Semantic versioning was introduced to the project during the research period; however, it was decided that the version numbering that was visible to the player would not be automated, but instead it would be increased manually depending on the contents of each game client update. Since the user-facing version number didn't have any hard requirements from either publishing platform, and the releases and updates were planned well beforehand, this was sufficient. The reason the semantic versioning was brought up was that it was a standardized way to version software using the major, minor and patch numbering logic. Previously only the major and minor numbers were used in the project, which was not seen as satisfactory especially since the tooling of the project was about to support the possibility of frequent patch updates. The semantic version number was also used to tag each release point of the game client in the version control to keep track of the release points.

Similarly the remote settings configuration had to be versioned in order to keep track of the latest settings configuration on the backend side. The versioning of the remote settings was quite simple to automate along with the pipeline, since the integer version was incremented each time a new version of the remote settings file was uploaded to the backend through the upload endpoint.

5.1.5 Automatic testing

While automatic testing is largely used in modern software development and the use of automatic testing has become a standard in the software industry, it has proven to be difficult to cover games with automatic testing, since there are a lot of moving parts - which are also changing at a rapid pace during development - and there are hardly any standardized test suites available for games when comparing the tooling that is available for web development technologies, for example. This leads to game developers having to develop their own test tools or having to lean on manual QA testing. While Unity offers some kind of testing tools through its package manager system, it was determined that the investment to properly benefit from automatic testing would require either bringing in a specialized developer that focuses on creating and maintaining tests in the project, or a lot of time from the current development team to cover the game project with tests. Since the game had not really been planned with testability in mind from the beginning, bringing in automated testing at this point was seemingly challenging. However, some ideas were brought forth about automatic play testing scenarios, that could act as smoke tests that would cover the most critical gameplay behaviour, for example the tutorial phase of the game, or simple menu scenarios. Execution of automated tests would have been apparently possible in the Unity Cloud Build process, as in the form of editor mode testing, but the real benefits would have been gained if the tests could have been executed on real device, which would have been even more complex to implement. In any case, the automatic testing of the game client could not be prioritized highly enough to be worthwhile during the research period. On the other hand the frequent building that was actualized through the planned build pipelines was already a test in itself regarding the ability to compile the source code.

Regarding the remote settings configuration the automatic testing was seen as undisputedly essential. In the case of the remote settings the testing requirement was more straightforward, and as mentioned in the previous sections, validations tests were planned to cover the remote settings configuration correctness with JSON validation and schema validation.

5.1.6 Continuous monitoring

Continuous monitoring of the game client performance after each update, and the continuous monitoring of the backend were seen as very important in ensuring the stability of the product without someone having to constantly monitor dashboards rigorously. The continuous monitoring that was set out to be implemented in the project was rooted in the idea that anomalies should be detected automatically, and those anomalies should be reported to the developers, preferably to Slack where they would reach the required people with no delays. That said, the monitoring logic was expected to have some sort of smart automation and the ability to detect out of place activity so that only the important deviations would get reported to developers. More precisely, any activity that is affecting a notable number of players should be reported to the developers, but issues that are under certain thresholds should not interrupt the developers from carrying out their regular activities. This was also experienced by the developers in practice when the first monitoring solutions were implemented without proper thresholds. Incessant false alarms lead to developers either ignoring or muting the alarm notifications altogether, which made the monitor alarms practically useless. However, after corrections and careful setup of proper thresholds this could be turned around and only the proper issue alerts were going to be delivered to developers.

5.1.7 Continuous improvement

Continuous improvement, while seemingly a no-brainer in any kind of project, was realized to be quite essential in order for the team to improve and for the product to improve as well. It was realized that if there weren't any improvements, it probably meant that things were probably getting worse over time, since the complexity of the project was increasing each time changes were contributed to it. This was evident from the bug and issue compilation list, which was constantly growing, and at times growing much faster than the team was able to make it shorter. Many continuous improvement attempts were actively made to improve the overall state of the project, like defining and enforcing a standardized convention for the source code, a requirement that had been overlooked before. Additionally, some refactoring guidelines were agreed upon within the team to also make sure that legacy code would eventually get improved over time and that there was plan for systematic refactoring of the source code. However, more than a technical practice the continuous improvement should be seen as a mindset that each of the developers in the project should maintain in order to make sure there is room for improvements.

5.1.8 Summary of the practices

To summarize the most important practices and their prioritized target areas, the previously covered practices were combined into table 1. As it became evident during the answering of the first research question, some of the practices were seen as more important to be applied to certain areas of the mobile game development project rather than covering all areas. Area in this context meant either the game client, the backend or the remote settings. Since this was the case, also the prioritized target areas were highlighted in the table in order to better support the answering of the second research question about the tools that would support the applying of these practices.

Table 1: DevOps practices and their prioritized target areas

Practice	**Prioritized target area**
Continuous integration	Game client
Continuous delivery	Game client
Continuous deployment	Remote settings
Automatic versioning	Game client Remote settings
Automatic testing	Remote settings
Continuous monitoring	Game client Backend
Continuous improvement	Game client Backend

5.2 Tools

The second research question was: **Which DevOps tools are needed for applying the practices in the mobile game development project?** To answer this question, the first research question about the most beneficial DevOps practices had to be already covered initially so that this research question could be approached in full. The nature of this research question was quite actionable, so answering it required both time and effort and active experimentation with the different tools that were assessed to be used in the mobile game development project. This chapter covers the tooling solutions that were implemented into the project during the research period, and also touches on the tool candidates that were considered to fulfill some of the prospected needs but were dropped out for one reason or another.

5.2.1 The planned DevOps toolchain

As the answers for the first research question indicated, many DevOps-related practices were deemed as important for the mobile game development project, some more than others, and some of the practices were deemed most important only in certain areas of project, for example in relation to game client or in relation the backend, but not necessarily as important for both. In order to clarify the scope of the tooling and the requirements that they posed on the whole project, it was important to be able to clarify the new and existing tools that were used at each phase of the DevOps lifecycle. To do this, composing a DevOps toolchain that made up the entirety of the project was seen as appropriate.

The planned DevOps toolchain was based on the requirements that were uncovered after answering the first research question and the toolchain is composed in table 2. In the table the new tools are also highlighted separately to clarify which new tools were introduced to the project in order to provide the answers to the second research question. However, it should be noted that also the existing tools were used more prudently along with the new tools, which means that also using the existing tools more extensively, by studying and utilizing their special features, contributed to the answering of the second research question. This was especially the case with Bitbucket and Unity Cloud Build which were originally underutilized in comparison to the features that they offered underneath.

Table 2: The planned DevOps toolchain with new tools separated

Lifecycle phase	**All tools**	**New tools**
Plan	Trello Slack Google Drive	
Code	Bitbucket Unity ASP.NET Core Visual Studio Visual Studio Code	
Build	Bitbucket Pipelines Unity Cloud Build Webhook Relay	Bitbucket Pipelines Webhook Relay
Test	Internal QA* External QA* NUnit	NUnit
Release	Bitbucket Unity Cloud Build Gitflow*	
Deploy	Azure Docker Fastlane App Store Google Play	Fastlane
Operate	Azure Remote settings Firebase Remote Config	Firebase Remote Config
Monitor	Unity Analytics Unity Cloud Diagnostics Firebase Analytics Azure Monitor UptimeRobot	Unity Cloud Diagnostics Azure Monitor UptimeRobot

**Not a tool but still pivotal in supporting the other tools*

The planned DevOps toolchain that was covered in table 2 was further clarified by creating a visualization of the toolchain and all of the tools that were used in different phases of the DevOps lifecycle, as seen in figure 5. The DevOps toolchain visualization was based on the traditional DevOps lifecycle as in the format that is often depicted when visualizing the lifecycle and its phases in relation to each other as a continuous loop.

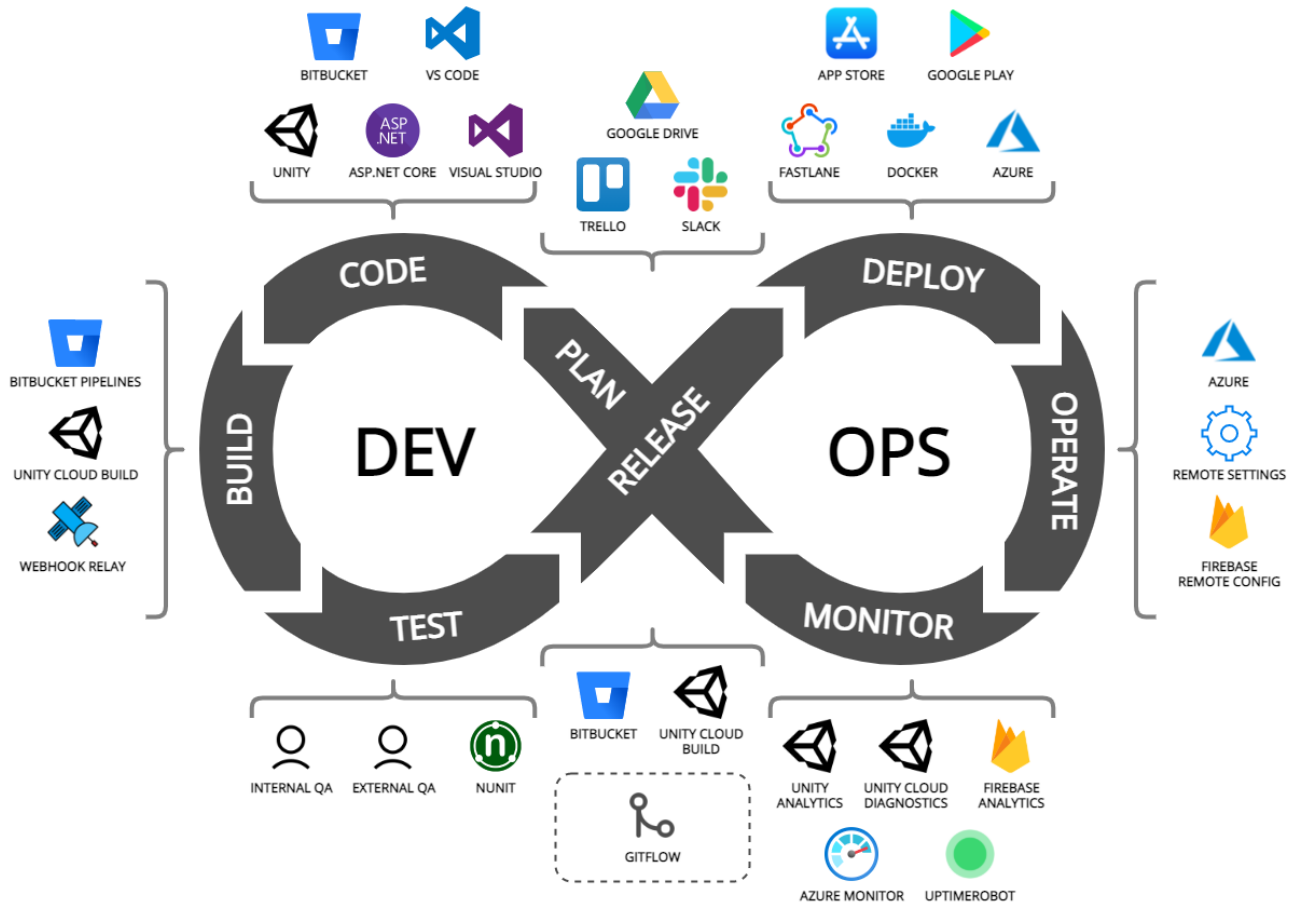

Figure 5: The planned DevOps toolchain

5.2.2 Selected tools to support the practices

To consolidate the findings about the most important practices and their prioritized target areas, which were introduced as the answers on the first research question, it was necessary to supplement them with the selected tools that were considered as the enablers of these practices in their prioritized target areas. The selected tools appended to the table of practices and their target areas is seen in table 3. Ergo, the selected tools in the table represent the new and in part existing tools that were listed in table 2, which are in this context associated with the practices and target areas that were seen desirable to be pursued in the research.

Table 3: DevOps practices, their prioritized target areas and the selected tools

Practice	**Prioritized target area**	**Selected tools**
Continuous integration	Game client	Bitbucket Pipelines Unity Cloud Build Webhook Relay
Continuous delivery	Game client	Bitbucket Pipelines Unity Cloud Build Fastlane
Continuous deployment	Remote settings	Bitbucket Pipelines
Automatic versioning	Game client Remote settings	Unity Cloud Build Bitbucket Pipelines
Automatic testing	Remote settings (Game client*)	Bitbucket Pipelines (NUnit*)
Continuous monitoring	Game client Backend	Unity Cloud Diagnostics Azure Monitor UptimeRobot Firebase Remote Config
Continuous improvement	Game client Backend	Developers**

**Was not considered as a priority, but was still covered by the research*

***Not a tool, since continuous improvement is more a cultural trait*

Bitbucket Pipelines

Bitbucket Pipelines is a serverless CI/CD tool offered as part of the Bitbucket Cloud product ("Bitbucket Pipelines", n.d.). Bitbucket Pipelines seemed like the obvious choice for automating some of the tasks that required automation, since the project was already hosted in Bitbucket Git repositories to begin with. The Bitbucket Pipelines was seen as easy to setup, since it was a serverless product in nature, meaning that the developer didn't have to worry about configuring a server in order to make use of the pipelines, but instead the pipeline instances were spun up automatically once they were needed. It was also seen to be desirable that the tooling choices would be closely aligned with the already existing tools so that their upkeep would be simple. Bitbucket Pipelines was mapped out as the candidate for fulfilling the continuous integration, continuous delivery and continuous deployment needs that were expressed, and was able fulfill these needs elegantly.

Webhook Relay

Webhook Relay is a webhook-related service that specializes in webhook forwarding and bidirectional tunneling ("Webhook Relay Introduction", n.d.). Webhook Relay was examined since Bitbucket Pipelines had monthly build time limitations attached to it, which would be easily exceeded if the pipeline instances were used for long-running tasks such as webhook polling between Bitbucket and Unity Cloud Build service, as was learned through experimentation. Webhook Relay pricing was relatively cheap, and the product featured very handy serverless functions that made the setup easy to manage overall. On the downside this was a "one more tool" in the toolchain, that could have been possibly avoided if the Bitbucket Pipelines weren't as limited with their monthly build time quotas and their hefty pricing.

NUnit

NUnit is a unit-testing framework for .NET languages. In the context of the Unity game engine a custom version of the NUnit library is shipped as part of the Unity Test Framework package distribution. The Unity Test Framework can be used for creating and running edit mode and play mode tests ("Unity Test Framework", n.d.). While the game client testing wasn't found as the most critical practice to be covered in the research, it was still a topic that had raised a lot of discussion in the group interviews, and for that reason the tooling related to that was brought up and covered, in case it would be a candidate for future improvements.

Fastlane

Fastlane is an app automation tool that helps to automate the releasing and deploying of iOS and Android apps and other tasks closely related to releasing and deploying of applications (“Fastlane Docs”, n.d.). Fastlane was already a familiar tool to some of the individuals in the development team as it had been previously used in the on-premises Jenkins server solution that the company had used before. Fastlane was also used by the Unity Cloud Build service under the hood, and it was possible to add custom Fastlane configurations to extend the cloud build configurations, for example to enable delivering builds to the App Store or TestFlight, and even to Google Play. However, a solution such as this was not readily available online for the picking, so some studying and experimentation was required in order to figure out the automation requirements for successful TestFlight distribution. Ultimately, a working solution was found and iOS builds could be automatically delivered to TestFlight directly from the Unity Cloud Build service.

Firebase Remote Config

Firebase Remote Config is a cloud service that enables the developer to adjust predefined values in the application without updating the application separately (“Firebase Remote Config”, n.d.). Essentially this makes Firebase a sort of live operations tool and enables a segmented setup of values to certain regions, for example. The tool can also be used for A/B testing certain features before they are rolled out to the whole player base. These features make Firebase Remote Config a good candidate for supporting the continuous monitoring initiative since the monitored findings from the segmented behaviour can be used to modify the game for the better. Firebase Remote Config could also be seen to support the continuous improvement practice, since the point of the segmentation and testing is to find out optimal configurations that would essentially improve the game overall.

Unity Cloud Diagnostics

Unity Cloud Diagnostics is a suite of cloud-based tools that enable developers to receive diagnostics data and feedback directly from their games (“Unity Cloud Diagnostics”, n.d.). Unity Cloud Diagnostics was seen as lucrative solution for tracking game performance since the tool was an integral part of the Unity engine and it was easy to activate, and relatively easy to configure to send notitifications to a mail address or to Slack in case errors or exceptions were received on the game clients of players. However, due to configuration limitations the Unity Cloud Diagnostics couldn’t be set up with custom thresholds, which meant the notifications were spitting out false positives from time to time, that were only affecting a small minority of players. This left something to be desired, perhaps a more specialized tool that would allow more tailoring.

Azure Monitor

Azure Monitor is solution for collecting and analyzing data of applications and services and measuring their performance. The Azure Monitor data can be used to setup alerts that can notify when issues or anomalies have been found in the monitoring data (“Azure Monitor alerts”, 2021). Azure Monitor was a natural choice to be used for monitoring and setting up the backend-related alerts since the backend resided in Azure. It was also relatively straightforward to setup the anomaly-based alerts and alert thresholds to certain services that needed to be monitored closely. Azure Monitor alerts were eventually configured to send automatic Slack alerts and SMS alerts to certain on-call-devices in order to keep the development team notified of the manifesting issues.

UptimeRobot

UptimeRobot is a web page uptime monitoring service that can be used for monitoring the availability of given web pages, or keywords on a web page with ease (“UptimeRobot FAQ”, n.d.). UptimeRobot was used as a temporary solution for monitoring the uptime of certain critical server addresses, and later it was used as a backup solution in addition to the more advanced Azure Monitor and alerts configured through Azure. To the benefit of the UptimeRobot it can be said that the service was extremely easy to setup and it served its purpose perfectly as a development-time monitor service when it was configured to monitor certain keywords in a publicly exposed server list in order to track their uptime.

5.2.3 Overview of the CI/CD pipelines for the game client

Several build targets for the game client were configured in the Unity Cloud Build service to enable a straightforward way to create build artifacts for both supported platforms, Android and iOS, and for each environment needed with distinct settings. Debug and staging builds were built from the development branch of the repository which contained the latest development state of the project, whereas release builds were built from the master branch of the repository which contained the production state of the project.

Debug builds were set up using debug signing with development mode enabled and they were preprocessed at build-time to allow the selection of the server type once the build was executed on device. Also, debug builds were configured so that the finished build artifacts were delivered to Slack in the form of a shareable link. Staging builds, in this context, meant builds that otherwise matched with builds using debug build settings, but were signed with release signing so that they could be delivered to Google Play internal testing on Android and to TestFlight on iOS. On Android, the AAB format (Android App Bundle) was required for Google Play, but the APK format (Android Application Package) was used for debug builds since builds using the format were easier to install on test devices directly. On iOS the Ad Hoc signing was used for debug builds to allow installation on registered devices that had been included in the provisioning profile. However, for staging and release builds the release signing was required and distribution for release signed builds was possible only through TestFlight and App Store.

Similarly to staging builds, release builds were using release signing but had development mode disabled, compression enabled and were preconfigured at build-time to use the production server. Additionally, release builds had automatic version code incrementation on each build, with Android builds using the Unity Cloud build pipeline number directly, and iOS builds using a date-based format for increasing the bundle version.

Android and iOS debug build targets were scheduled so that nightly builds would be delivered and linked to Slack each night so that the external quality assurance team could start testing the latest builds first thing in the morning. Other build targets required manual triggering by the developer to be built. The configured Unity Cloud build targets are seen in table 4.

Table 4: Unity Cloud Build targets for Android and iOS

Build target	**Branch**	**Artifact**	**Server**	**Distribution target**	**Method**
Android Debug	develop	APK	Test	Link/Slack	Automatic
Android Staging	develop	AAB	Test	Google Play Internal	Manual
Android Release	master	AAB	Production	Google Play Production	Manual
iOS Debug	develop	IPA	Test	Link/Slack (Ad Hoc)	Automatic
iOS Staging	develop	IPA	Test	TestFlight	Automatic
iOS Release	master	IPA	Production	App Store	Manual

Regarding table 4, the distribution target column meant the intended distribution target of the build artifact. Slack link distribution was trivial to setup using Unity Cloud Build service default functionalities, but distribution to publishing platforms, such as the App Store and the Google Play, required custom Fastlane configurations. For iOS, this kind of custom Fastlane configuration was setup and the distribution to TestFlight was automated, hence a continuous delivery pipeline for iOS had been created. For Google Play a similar continuous delivery automation was planned, but as it was not deemed as a priority in the research, it was not rushed. However, it still remains as a desirable target for automation at a later date, perhaps some time after the research has ended.

5.2.4 Continuous integration pipeline for nightly builds

The continuous integration pipeline that was used to produce the nightly builds in seen in figure 6. A similarly structured pipeline was ran each night to produce both Android and iOS development builds, which were then linked to Slack so that they could be picked up by the QA team in the morning. The pipeline setup was quite trivial as it could be fully setup in the Unity Cloud Build dashboard, as Unity Cloud Build supported Bitbucket as a version control source and Slack as a distribution target for link sharing out of the box.

Although this pipeline was used on a nightly level during the research period, there was the ability to use it more frequently, if needed. However, due the fact that the pull request pipeline was already put in place and it was building each pull request before integrating them to the development branch, it was chosen that this pipeline was only used for nightly builds to save bandwidth for the pull request builds during the working hours. Since no one in the development team was working at night, it was a great occasion to use the build server to do laborious automation tasks.

Figure 6: Continuous integration pipeline for nightly builds

5.2.5 Continuous integration pipeline for pull requests

In addition to the build targets seen in table 4, there were automatically created debug build targets created for each pull request through Bitbucket Pipelines. In theory all of these automatically created targets could have been used through the Unity Cloud Build user interface as well, although in practice these build targets were controlled through Bitbucket Pipelines and were disposed of after closing each pull request, hence they were only temporary targets used for testing changes introduced in pull requests. The pull request pipeline targets were created using the Android and iOS debug builds targets as the template configurations with minor changes, such as the branch being matched with the pull request. The flow of the pull request pipeline is shown in figure 7.

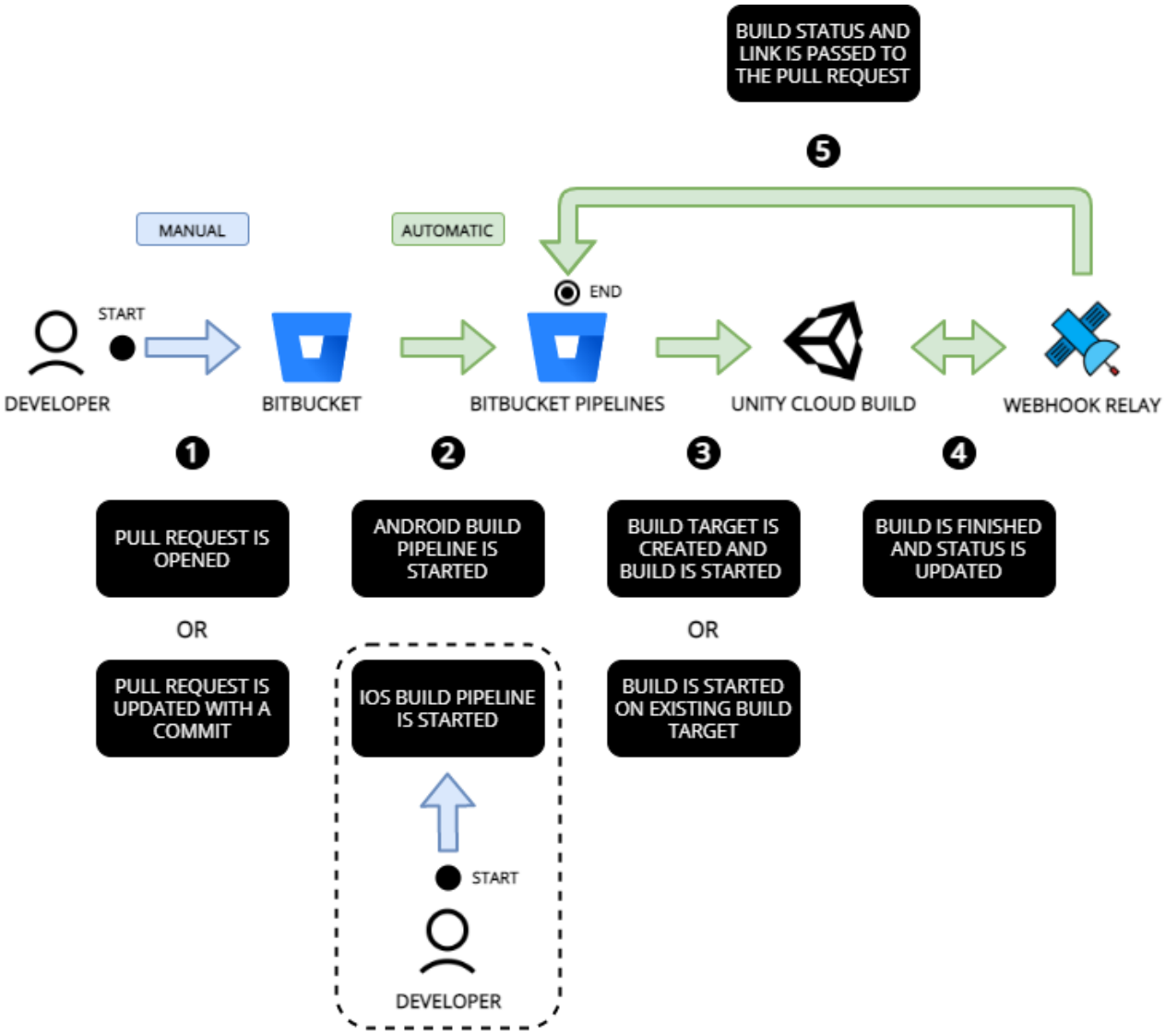

Figure 7: Continuous integration pipeline for pull requests

5.2.6 Continuous delivery pipeline for iOS

To support the continuous delivery practice requirement, the continuous delivery pipeline for iOS was created. The continuous delivery pipeline for iOS is visualized in figure 8. The essential automation tool in this context was Fastlane, which enables the distribution of the build artifacts directly from the Unity Cloud Build service to TestFlight, and the manual release of the TestFlight build to the App Store, if needed. In order to support custom Fastlane configurations, one had to be created and hosted in the game client repository and the path pointing to the Fastlane configuration had to setup in the Unity Cloud Build target configuration. The iOS builds could be ordered directly from the Unity Cloud Build dashboard and eventually after the order the build artifact was created and delivered to TestFlight.

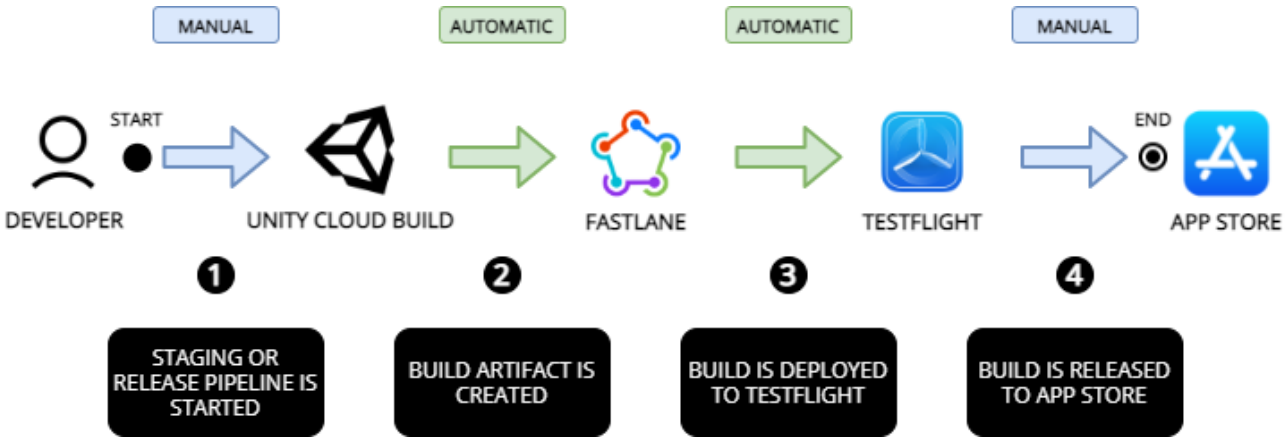

Figure 8: Continuous delivery pipeline for iOS

5.2.7 Continuous deployment pipeline for remote settings

The continuous deployment pipeline for remote settings can be seen in figure 9. As the requirement for automating the deployment of the remote settings configuration was quite simple, especially when considering how hard this requirement would be to fulfill on the game client or the backend side, it was decided that it should be approached. In actuality the remote settings configuration was hosted in the backend, and the configuration file was updated through a POST endpoint each time the pipeline was ran. Before the POST request was made, the contents of the configuration file were validated with both schema validation and generic JSON validation to avoid issues if the developer had mistakenly committed code with broken syntax.

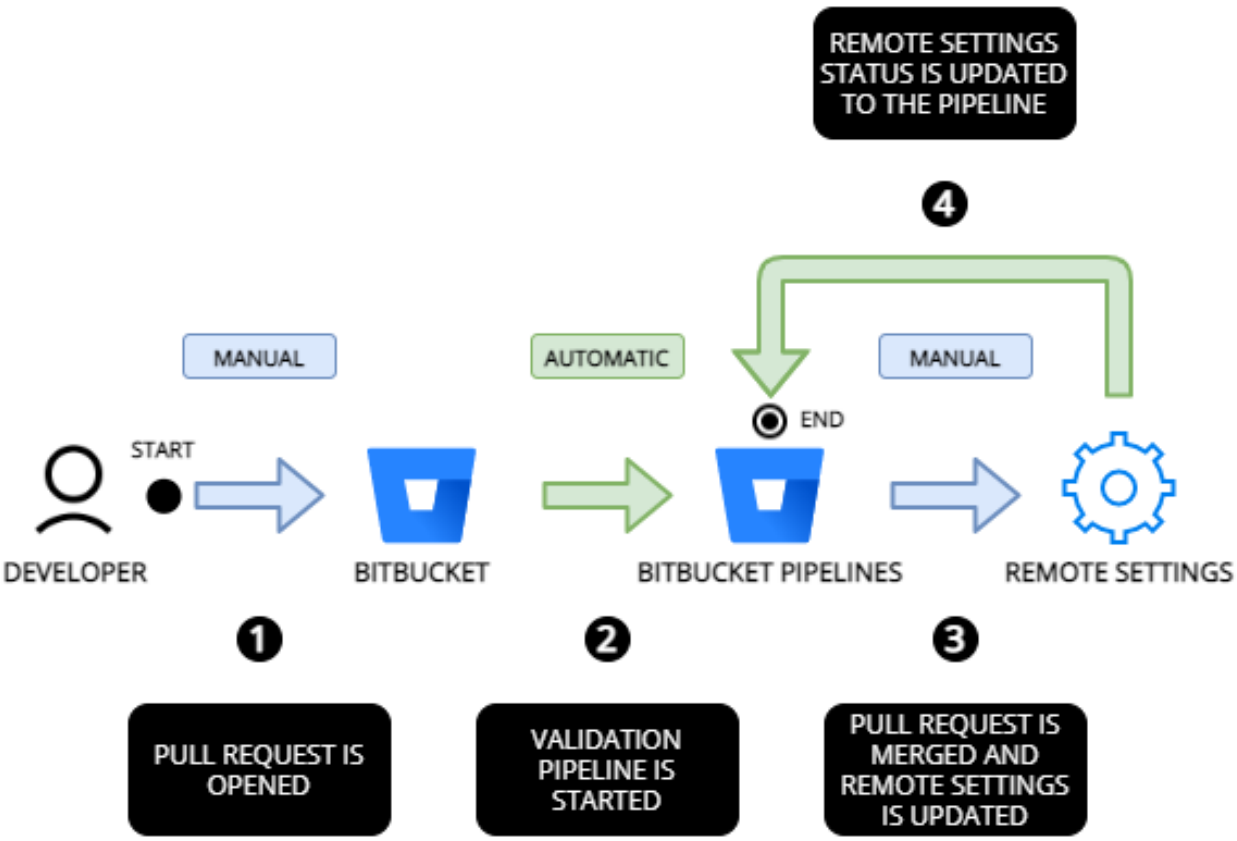

Figure 9: Continuous deployment pipeline for remote settings

5.2.8 Continuous monitoring pipeline for backend

In order to support the continuous monitoring practice on the backend side the Azure Monitor was setup with anomaly detection that was set to automatically trigger Azure alerts, that were then sent both to Slack and to the on-call-devices that some of the developers were possessing. Additionally the on-call-device was configured to start an alarm each time a SMS message with certain keyword content was received. An app called Priority Alerts was used to setup the keyword-based SMS alarms that would alert anyone who was in possession of the on-call-device ("Priority Alerts", n.d.). The continuous monitoring pipeline for the backend is visualized in figure 10.

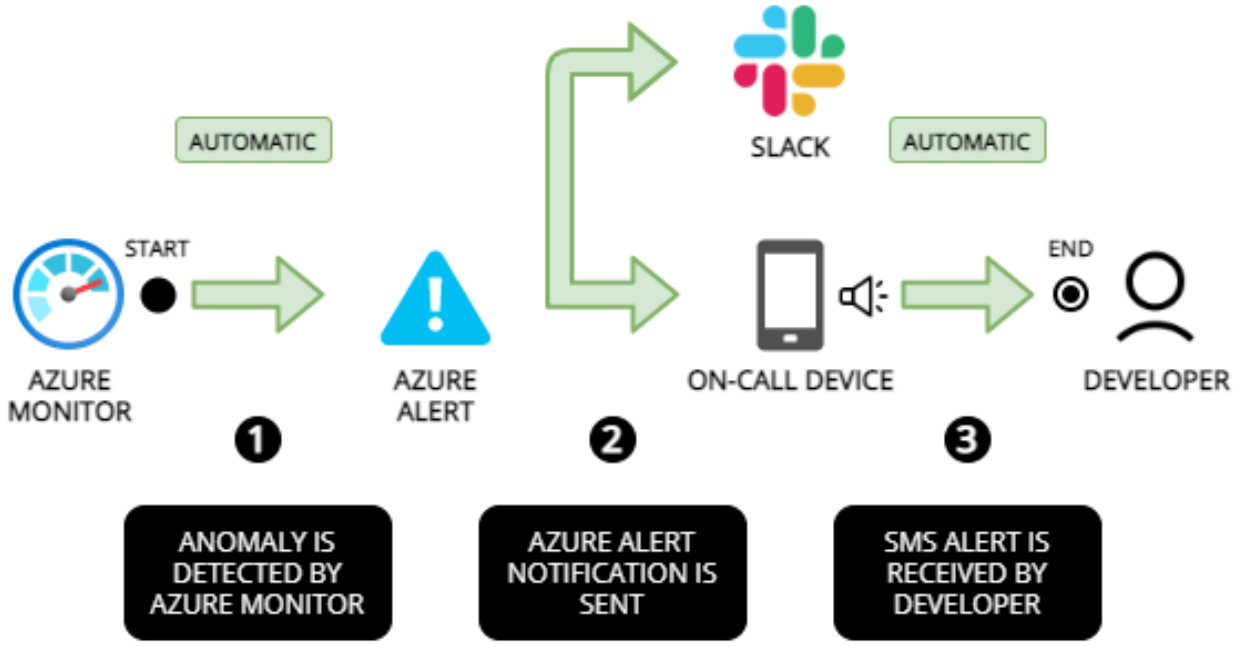

Figure 10: Continuous monitoring pipeline for backend

5.3 Culture

The third research question was: **How does applying DevOps impact the culture of the mobile game development team?** To answer the final research question, it was vital that the DevOps transformation had already been experienced and observed in practice and that the previous research questions had been answered, as they were related to bringing forth the actual applying of DevOps. Answering this question had a heavy reliance both on the observations that covered the whole research period and on the conclusive interviews that finally covered the impact of DevOps within the development team in depth.

5.3.1 Quality orientation

The quality orientation of the development team was identified as an important trait that had directly stemmed from the DevOps mindset that was slowly devolving into the minds of the development team, as the practices, tools and the DevOps culture was constantly highlighting quality as an important aspiration. The increasing quality orientation was evident from the spontaneous ideations for improving the quality of the project, that were witnessed throughout the research period. The quality approach seemed to touch on many areas of the project, since quality wasn't just considered through the experienced quality that the player would feel when playing the game, but also covering the quality of the internal development processes and the quality of the source code, for example. The quality orientation seemed to be fed by the fact that since after implementing the DevOps practices into the project, many of the processes were much more standardized and systematic than before. This also seemed to affect other previously loose internal processes to become more systematic, which could be witnessed as a sort of a snow-ball effect since it began affecting all the aspects of project. It could be also seen that since the proper automation processes were enabling the quality-oriented processes, such as test build automation, the quality orientation was easier to achieve and grasp. Since the introduction of the automatic pull request pipeline, the team also started to pay more attention to code reviews that were acting as quality gates along with the builds that were produced for each pull request. And since the code reviews were becoming more thorough it also led to the team having a higher sense of ownership over each line of code they submitted to the version control. At the same time, since code reviews are a collaborative effort, it seemed to also increase the shared feeling of ownership over the project's source code.

5.3.2 Open communication

The amount of open communication inside the development team seemed to increase throughout the research period. Not only was the communication more open, but it was also becoming more articulate as well. Certain expectations and an unofficial standardization were becoming to form around the communication style that the team was organically enforcing together. Common guidelines were agreed upon in Slack about which channels would be used for certain sort of communication and even new channels were opened in order to improve the direct communication between different parties that were involved in the project. Also, a factor affecting the communication was the introduction of monitoring notifications and alerts that were integrated with Slack. Since notifications were directly sent to certain channels, it also provided a natural way to discuss the contents of those notifications directly in Slack. Beforehand, since the issues weren't reported to Slack, it was not as likely that each anomaly would be conversed on, since they were hidden in some dashboards that were rarely inspected.

5.3.3 Eagerness to improve

The eagerness to improve both oneself and the project was inflating during the research period. Since the activities around the project were under heavy inspection and improvement, it naturally cast some anticipation on the team to dive into self-improvement as well. Since it was realized that improvements could occur even during a short period of time, which was the research period, it was also becoming evident that it would mean that also other improvements that might have been seen out of reach before might actually be easier to achieve than expected. This probably poured some hope into the self-improvement attempts that might have been suppressed before. It was also delightful to observe that the DevOps practices, tools and culture were met with such openness and no-one was downright questioning the plans to improve the project and the development team activities with the DevOps practices. On the contrary, the team was highly interested about DevOps and was excited to participate in improving the project together. This eagerness to improve did not seem to fade towards the end of research, as the past improvements were only acting as a fuel to further improvements. The continuous improvement also arose as a prioritized practice from the answers to the first research question, which amplifies the significance of the eagerness to improve continuously.

5.3.4 Craving for automation

Since the benefits gained from automation were easily seen and experienced by everyone in the development team, it also fueled to desire automate things even further. However, the craving for automation was already present in the initial assessment interviews so it did not develop only after the successful implementation of automation practices, but was already existing underneath, but perhaps somewhat repressed. Many of the development team individuals were hands on participating in the implementation of some of the automation tools, which might have ignited the craving for automating even more. The automation measures implemented into the mobile game development were largely viewed positively and automation was in general seen in a positive light. However, some indications could be picked up that automation could also be somewhat dreaded since it could in theory reduce the irreplaceableness of certain individuals in the team. These remarks were mostly downplayed by humor but there might have been a serious underlying concern about automation negatively affecting the otherwise secured position inside the development team. Most of the team was in agreement that automation was not in itself of any value, but that the automation should always aim to solve or alleviate a real-world issue in order to be of value, which was sensible and proved that the team was very conscious on the matter.

5.3.5 Favorability towards new tools

The research period was a relatively short period time with constant introduction of new tools, which might have ushered one to think that at some point the team would start rejecting new tools altogether, but ultimately this was not the case. Although initially there was some reluctance towards learning to use new tools, the positive consequences of adopting new tools started to push through because it was realized that the new tools weren't just adopted because changing them around was amusing, but because of the added value that they brought in. Once the team was becoming accustomed to adapting new tools, it started to become easier, and is likely to remain so going forward. Hence, it could be perceived that the favorability towards new tools had increased through experimentation and observably good results.

5.4 Summary of the results

This chapter aims to summarize the key results of the research and to answer each of the research questions in a concise format.

1) Which DevOps practices are needed to most benefit the mobile game development project?

Improving upon continuous integration and continuous delivery practices were seen to most benefit the game client development, while continuous deployment was seen to best support the remote settings configuration. Automatic versioning was considered to be important for both the game client and the remote settings, although automatic testing could only be prioritized for the simplified needs of the remote settings. Continuous monitoring and continuous improvement were seen as important practices and highly beneficial for the mobile game development project.

2) Which DevOps tools are needed for applying the practices in the mobile game development project?

Bitbucket Pipelines and Webhook Relay turned out as essential tools for bridging the gap between Bitbucket and Unity Cloud Build in terms of CI/CD practices. Fastlane was critical in enabling the continuous delivery of iOS builds to TestFlight. Azure Monitor that was configured with suitable anomaly alerts was able to fulfill the need for continuous backend monitoring. Unity Cloud Diagnostics turned out to be a satisfactory tool for monitoring the game client performance.

3) How does applying DevOps impact the culture of the mobile game development team?

Applying DevOps practices seemed to impact the mobile game development team mostly in very a positive way. In the end the team was more open-minded towards learning new things and taking on new tools. The eagerness to improve the working environment and oneself was clearly elevated. The quality-oriented mindset that the DevOps culture advocated for had clearly impacted the team positively. Also the increased amount of open communication was being encouraged inside the team. Additionally, the enthusiasm towards automation practices had grown.

6 Discussion

6.1 Reliability

The reliability of the conducted research is analyzed from the perspective of the theoretical basis and the methods and materials used for the research implementation. Additionally, the research ethics, which the research was conducted in respect to, is introduced.

6.1.1 Reliability of the theoretical basis

DevOps and the concepts associated with it have many definitions and even scholars are not in agreement with an exact definition of DevOps and the terminology surrounding it, which leaves room for opinionated viewpoints around the subject that often contradict with each other. It has proved challenging to find peer reviewed research data that would cover some of the more detailed practical aspects of DevOps, and in absence of hard-boiled research data a lot of reliance on the research has been put to expert opinions and practical guides. However, since this kind of practical DevOps-related material is widely available online it is still possible to gather a somewhat cohesive idea what the typical aspects of DevOps look like in practice for many organizations practicing it globally. Yet, the theoretical basis could have benefitted from a systematic review of a wider scope of data around individual themes. Since the research was an action research in nature, a very practically oriented research, the pragmatic approach may be seen as justifiable since the scope of the research was to focus on practicalities of DevOps and mobile game development in a real-world setting instead of focusing on moot points about the theoretical discrepancies surrounding the subject.

6.1.2 Reliability of methods and materials

The research was conducted using the action research strategy, where the aim was to gain insight on the research subject while actively attempting to inflict change on it. Since the researcher is closely associated with the research subjects in this type of research and devoted to the instillment of the change personally, it may reduce some of the observational objectiveness from the research. However, for this exact reason interviews were used to support the factual validity of the research, not relying on the observational data alone. The repetitive nature of action research allowed iterative development, which meant that the changes introduced to the project were able

to be studied and improved over the course of the research, which improves the verifiability of the data. The structured observation technique used was able to produce a good variety samples throughout the research period, but since the research was conducted during a time when the case company office was closed, and the development team was working remotely, the observation was limited to online meetings with the team and the use of instant messaging on the project team communication channel. This fact might have reduced the ability to gather especially cultural observational insight that might have been recovered more easily from a more natural setting if the team was working in the same physical office space. Since the research methods and materials were highly qualitive in nature, the results cannot be automatically applied to other projects especially in other organizations, but in the context of this research they would seem to form a credible foundation even to support further organizational developmental needs.

6.1.3 Research ethics

The research was conducted in accordance with the responsible conduct of research, and in respect to the ethical principles enforced by JAMK University of Applied Sciences ("Ethical principles and data protection", n.d.). To ensure the appropriate data management throughout the research, a data management plan was formed so that the inescapable consistency of the data management could be ensured. The contents of the data management plan were presented to the individuals that were being researched, and the extent of the research was openly disclosed with all the research participants. No personal data was collected in the perimeter of this research.

6.2 Analysis of the research results in respect of the theoretical framework

In this chapter the fundamental research results are discussed and analyzed in respect of the theoretical framework. Each theme of the research: practices, tools and culture are analyzed separately.

6.2.1 Practices

The DevOps practices that were deemed as worth pursuing in the context of the first research question and in answering of the first research question, were: continuous integration, continuous delivery, continuous deployment, continuous monitoring and continuous improvement, all of which have been seen as key components of DevOps according to Hall (n.d.). Additionally, automatic versioning and automatic testing were found to be important practices in the results. However, whether these should have been seen as part of the previously mentioned continuous practices or as separate practices is debatable. According to Pittet (n.d.) the automated testing is usually built into the CI/CD practices. Similarly, Forsgren and colleagues (2018, 41-58) suggest that continuous testing should be seen as something that enables the continuous delivery. Nevertheless, since these practices were explicitly raised as separate practices in the research results, it must mean that they were prominent to the development team, so addressing them separately was justifiable to support the aspirations and necessities of the development team. And as was evident from the second research question related to tooling, the automatic versioning and automatic testing were in fact closely aligned with the continuous practices when it came to forming practical solutions such as the automation pipelines.

6.2.2 Tools

The tooling was conceptualized in the results using DevOps toolchain as the comprehensive aid for breaking down the monumental matter that is DevOps. As Krohn (n.d.) had stated, the compilation of a unified toolchain is important so that the integrity of the toolchain would be secured, and so that the chosen tools would work well together. This idea was kept in mind when the tooling choices were introduced to the project. It is also apparent from the research results that tools which were closely aligned together were favored in the tool selection process. For example Bitbucket Pipelines was used for supporting continuous integration, because Bitbucket was already

being used in the project, and Unity Cloud Diagnostics was chosen for game client monitoring because it was well-matched with the Unity game engine. Similarly on the backend side Azure Monitor was used for monitoring because the backend was hosted in Azure. Other tooling choices could have been also made, but the aspiration was to keep things as simple as possible, and the flow of work as smooth as possible.

6.2.3 Culture

The cultural impacts of DevOps in the research results seemed to be in line with the findings of the theoretical framework as Forsgren and colleagues (2018, 29-40) have pointed out that if properly adapted, DevOps feeds built-in trust and psychological safety that leads to many of the fine things which were also witnessed in the research results. Generally, the DevOps mindset seemed to lead to not one, but many positive causations in the development team.

For the cultural foundation in the research setting, it should be mentioned that in the action research setting the researcher was in fact actively responsible for bringing forth change and transformation which was characteristic both to DevOps adoption and to the action research strategy (Forsgren et al., 2018, 29-40; Kananen, 2014, 11-12). Hence, it could be argued that the chosen research strategy matched well with the subject at hand.

6.3 Conclusion

6.3.1 Extensibility

The research results are expected to be applicable in the use of other development projects inside the employer organization. Many of the tooling choices in the research were made specifically for the requirements of this mobile game development project in mind, so some adaptation will probably be needed when attempting to extend the results of this research into other projects. However, the foundation that has been laid and the experiences that have been gained are alone very useful in supporting similar undertakings in other projects.

6.3.2 DevOps is applicable in mobile game development

The research results have showed that DevOps can also be highly useful in the use of mobile game development, and inside relatively small teams. There might be slightly different problems present in mobile game development than in software development, but mostly game projects are closely under similar constraints that any software project is.

6.3.3 There is no one way to execute DevOps

DevOps has proved to be very adaptable in nature. There is no standardized way to execute DevOps, which means that it can be tailored to the organizational and project-based needs in order to be used most efficiently. The DevOps model that was applied in this project could be taken as far as to form a new concept: “GameDevOps”, where the aim is to focus on around the impediments that mobile game development environment is causing to the developmental processes, such as challenging testability and publishing platform related constraints.

6.3.4 DevOps is a journey not a destination

While the statement might sound like a cliché it could be argued that DevOps should be considered a journey, not a destination. The reason for this is that DevOps is a not band-aid that can be thrown at a project and then be forgotten. DevOps is not something that is ever ready or completed. DevOps is culture and culture needs nourishment to thrive. Adapting DevOps requires that the DevOps transformation is properly initiated, and that enough information about it is shared inside the organization for DevOps to become a realistic choice for adoption.

6.3.5 Appropriate automation

One of the most meaningful takeaways has been the importance of understanding that automation is not in itself the purpose of DevOps. The purpose of automation is to serve the team and advance the goals of the team in the best way possible with the ability to automate work that is less meaningful and enable the team to spend their precious time on work that is truly meaningful and work that most benefits from collaboration.

Automating everything is not the goal of DevOps, and this should be understood both by the team and by the enthusiasts that are driving the DevOps transformation. The goal is to identify the appropriate targets for automation and plan for automating these before anything else.

6.3.6 Understanding the benefits of Live Ops

Many of the problems presented with fat clients, such as game clients, and long platform review times may be possible to solve with Live Ops tools and services. The presumption here is that Live Ops means the ability to provide updates to the game without separate client updates so that instead of client updates we are talking about live updates. The use of Live Ops in mobile game development could be seen as well-matched with the DevOps objectives.

6.3.7 QA testing is still the backbone of game testing

QA testing is still the most important way to test that a game is functional and that all the new features and existing features are working as expected. Automating all the tedious tasks around building, signing and releasing should be definitely automated so that the QA processes may be supported in the best way possible. Enabling the QA team to do their job in the best possible way is also a responsibility of the development team since all parties benefit from empowered quality assurance.

6.4 Suggestions for further research

DevOps is still evolving and new aspects for DevOps are emerging all the time. Service providers are making DevOps-related tools available in their services which might open new opportunities for automating the use of their tools and services. DevOps as a Service platforms that offer all the necessities in one big toolchain are worthwhile to study further, but it is also recommendable to carefully look at the competition of the forerunners since many of the providers out there have just recently started offering their corresponding DevOps services for their customers. Locking into a one vendor offering just about everything in one package arguably sounds tempting, but before blindly following the ongoing trends one should carefully examine what are the benefits and drawbacks of given services and make an educated evaluation only after analyzing the current offerings and pricing models that are out there.

For further research inside the employer organization, it could be seen that further standardizing the use of DevOps tools and building versatile automation tools that can be used across many projects might be the next logical step. While the DevOps tooling solutions that were turned up during the research period turned out to be quite flexible, there are still many more tools out there that might suit the needs of different projects in a better way and offer even more flexibility depending on project-based requirements.

References

App Store App Review. (n.d.). App Review guidelines at Apple Developer web page. https://developer.apple.com/app-store/review

Atlassian Bitbucket. (n.d.). Introduction to Bitbucket features at Atlassian web page. https://www.atlassian.com/software/bitbucket

Atlassian DevOps. (n.d.). Introduction to DevOps at Atlassian web page. https://www.atlassian.com/devops

Azure Monitor alerts. (2021). Overview of alerts in Microsoft Azure. Microsoft Docs. https://docs.microsoft.com/en-us/azure/azure-monitor/alerts/alerts-overview

Bitbucket Pipelines. (n.d.). Introduction to Bitbucket Pipelines at Bitbucket web page. https://bitbucket.org/product/features/pipelines

Brown, A., Stahnke, M., & Kersten, N. (2020). 2020 State of DevOps Report. Puppet & CircleCI. https://puppet.com/resources/report/2020-state-of-devops-report

Brown, T. (2020). DevOps vs Agile: What's the difference? Article on opensource.com web page. https://opensource.com/article/20/2/devops-vs-agile

Build and run your app. (2021). Android Studio build instructions at Google Developers web page. https://developer.android.com/studio/run

Chandler, H. M. (2014). The game production handbook (3rd ed.). Jones & Bartlett Learning.

Davis, J., & Daniels, R. (2016). Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale. O'Reilly Media.

DevOps Automation. (2019). Article at Tiempo Development web page. https://www.tiempodev.com/blog/devops-automation

Dogtiev, A. (2021). App Stores List. Article at Business of Apps web page. https://www.businessofapps.com/guide/app-stores-list

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. IEEE Software 33, 3, 94–100. https://doi.org/10.1109/MS.2016.68

Ethical principles and data protection. (n.d.). Thesis ethical principles and data protection. JAMK University of Applied Sciences. https://oppimateriaalit.jamk.fi/opinnaytetyo/en/thesis/ethical-principles-and-data-protection

Fastlane Docs. (n.d.). Fastlane app automation tool documentation. https://docs.fastlane.tools

Filipova, O., & Vilão, R. (2018). Software Development From A to Z: A Deep Dive into all the Roles Involved in the Creation of Software. Apress.

Firebase Remote Config. (n.d.). Firebase documentation at Google Developers web page. https://firebase.google.com/docs/remote-config

Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. IT Revolution Press.

Forsgren, N., Smith, D., Humble, J., & Frazelle, J. (2019). 2019 Accelerate State of DevOps Report. DORA & Google Cloud. https://cloud.google.com/devops/state-of-devops

GitHub DevOps. (n.d.). Introduction to DevOps tools at GitHub web page. https://github.com/learn/devops

GitLab DevOps. (n.d.). Introduction to DevOps solutions at GitLab web page. https://about.gitlab.com/stages-devops-lifecycle

Goasduff, L. (2021). Gartner Says Worldwide Smartphone Sales Declined 5% in Fourth Quarter of 2020. https://www.gartner.com/en/newsroom/press-releases/2021-02-22-4q20-smartphone-market-share-release

Gokarna, M. (2020). DevOps phases across Software Development Lifecycle. TechRxiv. https://doi.org/10.36227/techrxiv.13207796.v2

Hall, T. (n.d.). DevOps Best Practices. Article at the Atlassian web page. https://www.atlassian.com/devops/what-is-devops/devops-best-practices

Hargreaves, T. (2016). DevOps for Mobile: It's Not Just for Enterprise Anymore. Recording of a conference speech at GDC Vault web page. https://www.gdcvault.com/play/1023436/DevOps-for-Mobile-It-s

Hiremath, O. (2021). DevOps Toolchain, Clearly Explained: The What, Why, and How. https://www.plutora.com/blog/devops-toolchain

Hüttermann, M. (2012). DevOps for Developers. Berkeley, CA: Apress.

IDC Smartphone Market Share. (2021). Smartphone Market Share. IDC. https://www.idc.com/promo/smartphone-market-share

Jowsey, B. (2018). What is live ops and why should you care? https://www.pocketgamer.biz/comment-and-opinion/69071/what-is-live-ops-and-why-should-you-care

Kaasila, J. (2015). GTAC 2015: Mobile Game Test Automation Using Real Devices. Google TechTalks. https://www.youtube.com/watch?v=WFBfRk-GLRo

Katz, D. (2018). Continuous delivery for mobile with fastlane: Automating mobile application development and deployment for iOS and Android. Packt.

Kananen, J. (2009). Toimintatutkimus yritysten kehittämisessä. JAMK University of Applied Sciences.

Kananen, J. (2014). Toimintatutkimus kehittämistutkimuksen muotona. Miten kirjoitan toimintatutkimuksen opinnäytetyönä. JAMK University of Applied Sciences.

Krohn, R. (n.d.). Considerations for your DevOps toolchain. Article on the Atlassian web page. https://www.atlassian.com/devops/devops-tools/choose-devops-tools

Lancaric, M. (2018). A guide to soft-launching your mobile game. https://www.pocketgamer.biz/comment-and-opinion/67713/soft-launch-guide

Leswing, K. (2018). Apple will cut App Store commissions by half to 15% for small app makers. CNBC. https://www.cnbc.com/2020/11/18/apple-will-cut-app-store-fees-by-half-to-15percent-for-small-developers.html

Leung, T. (2017). Quick Guide to Live-Ops in Free-to-Play Games. https://medium.com/@supertommy/quick-guide-to-live-ops-in-free-to-play-games-2e3080ad4c47

Lewis, R. (2020). Mobile gaming trends 2021. Blog entry on ironSource web page. https://www.ironsrc.com/blog/mobile-gaming-trends

Little, C. (2019). DevOps toolchain figure at Gartner's blog page. https://blogs.gartner.com/christopher-little/2019/01/09/jan-2019-devops-agenda

Mala, D. (2019). Integrating the Internet of Things Into Software Engineering Practices. IGI Global. http://doi.org/10.4018/978-1-5225-7790-4

Mobile Live Operations Best Practices. (n.d.). The Free to Play Game Design Bible: Mobile Live Operations Best Practices. https://mobilefreetoplay.com/bible/mobile-live-operations-best-practices

Mäkelä, K. (2019). Agile vs. DevOps: the grand debate. Blog entry on Eficode web page. https://www.eficode.com/blog/agile-vs-devops

Ojasalo, K., Moilanen, T., & Ritalahti, J. (2009). Kehittämistyön menetelmät. Uudenlaista osaamista liiketoimintaan.

Okes, D. (2013). Performance metrics: The levers for process management. ASQ Quality Press.

Paul, F. (2015). 10 Deep DevOps Thoughts From Chef's Jez Humble. https://newrelic.com/blog/best-practices/devops-jez-humble

Phulare, A. (2020). DevOps Life cycle: Everything You Need To Know About DevOps Life cycle Phases. Edureka. https://www.edureka.co/blog/devops-lifecycle

Pittet, S. (n.d.). Continuous integration vs. continuous delivery vs. continuous deployment. Article on Atlassian web page. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Priority Alerts. (n.d.). Priority Alerts app at F-Droid web page. https://www.f-droid.org/en/packages/com.saiga.find.messagefinder

Raeburn, J. (2017). Lean Live Ops: Free Your Devs! Recording of a conference speech at GDC Vault web page. https://www.gdcvault.com/play/1024185/Lean-Live-Ops-Free-Your

Rajkumar, M., Pole, A. K., Adige, V. S., & Mahanta, P. (2016). DevOps culture and its impact on cloud delivery and software development. https://doi.org/10.1109/ICACCA.2016.7578902

Riti, P. (2018). Pro DevOps with Google Cloud Platform: With Docker, Jenkins, and Kubernetes. Apress.

Rothwell, W. (1999). The Action Learning Guidebook: A real-time strategy for problem solving, training design, and employee development. Jossey-Bass/Pfeiffer.

Schaub, W. (2019). DevOps transformation: Key differences in small, midsize, and large organizations. https://opensource.com/article/19/1/devops-small-medium-large-organizations

Schultz, C. P., & Bryant, R. D. (2017). Game Testing: All in One, Third Edition. Mercury Learning.

Singh, M. (2021). Google Play drops commissions to 15% from 30%, following Apple's move last year. TechCrunch. https://techcrunch.com/2021/03/16/google-play-drops-commissions-to-15-from-30-following-apples-move-last-year

StatCounter Mobile OS Market Share In India. (2020). Mobile Operating System Market Share India. StatCounter. https://gs.statcounter.com/os-market-share/mobile/india/2020

StatCounter Mobile OS Market Share In USA. (2020). Mobile Operating System Market Share United States of America. StatCounter. https://gs.statcounter.com/os-market-share/mobile/united-states-of-america/2020

StatCounter Mobile OS Market Share Worldwide. (2020). Mobile Operating System Market Share Worldwide. StatCounter. https://gs.statcounter.com/os-market-share/mobile/worldwide/2020

Storbakken, M. (2020). DevOps: Where Are We and How Did We Get Here? https://blogs.vmware.com/management/2020/05/devops-where-are-we-and-how-did-we-get-here.html

Swartout, P. (2012). Continuous delivery and DevOps: A quickstart guide. Packt Publishing.

Telfer, A. (2016). What to Expect When You're Expecting a Soft Launch. Recording of a conference speech at GDC Vault web page.
https://www.gdcvault.com/play/1023238/What-to-Expect-When-You

Understand Play policies. (2020). Understand and apply the Google Play policies to create a trusted app. Google Developers.
https://developer.android.com/distribute/best-practices/develop/understand-play-policies

Unity Test Framework. (n.d.). Overview of the Unity Test Framework at Unity web page.
https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html

Ultima, J. (2021). Interview with the Zaibatsu Interactive Oy's CEO. Interviewed on 21 May 2021.

Unity Cloud Build. (n.d.). Introduction to Cloud Build service at Unity web page. Unity.
https://unity3d.com/unity/features/cloud-build

Unity Cloud Diagnostics. (n.d.). Introduction to Unity Cloud Diagnostics at Unity web page.
https://unity3d.com/unity/features/cloud-diagnostics

Unity company presentation. (n.d.). Introduction to the Unity company at their web page.
https://unity.com/our-company

Unity solutions. (n.d.). Overview of the use cases of the Unity engine.
https://unity.com/solutions

UptimeRobot FAQ. (n.d.). Frequently asked questions section at UptimeRobot web page.
https://uptimerobot.com/faq

Watts, S., & Kidd, C. (2017). DevOps vs Agile: A Complete Introduction. BMC Blogs.
https://www.bmc.com/blogs/devops-vs-agile-whats-the-difference-and-how-are-they-related

Webhook Relay Introduction. (n.d.). Introduction to Webhook Relay on its web page.
https://webhookrelay.com/intro

Wijman, T. (2021). Global Games Market to Generate $175.8 Billion in 2021; Despite a Slight Decline, the Market Is on Track to Surpass $200 Billion in 2023. https://newzoo.com/insights/articles/global-games-market-to-generate-175-8-billion-in-2021-despite-a-slight-decline-the-market-is-on-track-to-surpass-200-billion-in-2023

Appendices

Appendix 1. Initial assessment interview with the development group

Theme 1: Development viewpoint

- What recommendable practices have you used in previous projects?
- What recommendable tools have you used in previous projects?
- What expectations do you have for the potential cultural change in the project?
- Which practices are needed for implementing DevOps in the mobile game development project?
- Which tools are needed for implementing DevOps in the mobile game development project?

Theme 2: Operations viewpoint

- What recommendable practices have you used in previous projects?
- What recommendable tools have you used in previous projects?
- What expectations do you have for the potential cultural change in the project?
- Which practices are needed for implementing DevOps in the mobile game development project?
- Which tools are needed for implementing DevOps in the mobile game development project?

Appendix 2. Initial assessment interview with the steering group

Theme 1: Present state and target state

- Have you ever used DevOps practices in previous projects?
- What expectations do you have for the potential cultural change in the project?
- What kind of strengths would you expect to perceive in the DevOps transformation?
- What kind of weaknesses would you expect to perceive in the DevOps transformation?
- What kind of opportunities would you expect to perceive in the DevOps transformation?
- What kind of threats would you expect to perceive in the DevOps transformation?

Theme 2: Implementing DevOps culture

- How would you expect the DevOps culture to impact the development team activities?
- How would you expect the DevOps culture to impact the end-user i.e. the player?
- How would you expect the DevOps culture to impact project management?
- In what ways would you attempt to support the DevOps transformation and adaptation?

Appendix 3. Conclusive interview with the development group

Theme: DevOps lifecycle and toolchain

- What kind of changes have you perceived during the research period in general?
- What kind of cultural changes have you perceived during the research period?
- What kind of changes have you perceived in the planning phase?
- What kind of changes have you perceived in the coding (developing) phase?
- What kind of changes have you perceived in the building phase?
- What kind of changes have you perceived in the testing phase?
- What kind of changes have you perceived in the releasing phase?
- What kind of changes have you perceived in the deploying phase?
- What kind of changes have you perceived in the operating phase?
- What kind of changes have you perceived in the monitoring phase?
- What kind of further improvements would you hope for?

Appendix 4. Conclusive interview with the steering group

Theme 1: DevOps lifecycle and toolchain

- What kind of changes have you perceived during the research period in general?
- What kind of changes have you perceived in the planning phase?
- What kind of changes have you perceived in the coding (developing) phase?
- What kind of changes have you perceived in the building phase?
- What kind of changes have you perceived in the testing phase?
- What kind of changes have you perceived in the releasing phase?
- What kind of changes have you perceived in the deploying phase?
- What kind of changes have you perceived in the operating phase?
- What kind of changes have you perceived in the monitoring phase?
- What kind of further improvements would you hope for?

Theme 2: Impact of DevOps culture

- Have you perceived that DevOps culture has had an impact on development team activities?
- Have you perceived that DevOps culture has had an impact on the end-user i.e. the player?
- Have you perceived that DevOps culture has had an impact on project management?
- In what ways have you attempted to support the DevOps transformation and adaptation?
- What kind of further improvements would you hope for?

Appendix 5. Observation log sheet template

Observation log sheet	
Name of the research	DevOps in mobile game development
Name of the researcher	Tarmo Jussila
Consecutive number of observation	
Date and time of observation	
Subject(s) of observation	
Description of observation	

Appendix 6. Keywords and search expressions

Keywords

- DevOps
- DevOps culture
- DevOps toolchain
- Mobile game development
- Continuous integration
- Continuous delivery
- Continuous deployment
- Unity
- Azure
- Bitbucket

Search expressions

- ➔ DevOps
- ➔ “DevOps culture”
- ➔ “DevOps toolchain”
- ➔ DevOps AND culture
- ➔ DevOps AND toolchain
- ➔ DevOps AND “continuous integration”
- ➔ DevOps AND “continuous delivery”
- ➔ DevOps AND “continuous deployment”
- ➔ DevOps AND CI/CD
- ➔ DevOps AND “game development”
- ➔ DevOps AND “mobile development”
- ➔ DevOps AND Unity
- ➔ DevOps AND Azure
- ➔ DevOps AND Bitbucket
- ➔ “continuous integration” AND “game development”
- ➔ “continuous delivery” AND “game development”
- ➔ “continuous deployment” AND “game development”
- ➔ “continuous integration” AND “mobile development”
- ➔ “continuous delivery” AND “mobile development”
- ➔ “continuous deployment” AND “mobile development”

Appendix 7. Data management plan

1. General description of data

What kinds of data is your research based on? What data will be collected, produced, or reused? What file formats will the data be in?

The research is based on data collected from interviews and through observation. Interview data is recorded and then transcribed into text format. Observation data is collected in text format in a log sheet. Used text file format is .docx. Used recording file format is .mkv.

How will the consistency and quality of data be controlled?

The consistency and quality of data will be controlled by maintaining and backing up the original data files until they have been converted or processed into another format. For recordings, the original data files are maintained until the data has been fully transcribed into text.

2. Ethical and Legal Compliance

What ethical and legal issues are related to your data management, for example, the Data Protection Act and other legislation related to the processing of the data)?

No personal data is collected in the research. All the recorded interview data is anonymized when it is transcribed into text format from the recording format. All the interviewees have been notified of the research and the data management plan has been presented to them.

How do you manage the rights to the material you use, produce and share? Is the material confidential, are there any copyrights, licenses, or other restrictions?

The commissioner of the research holds the rights to the research material and the material that is produced during the research. A thesis report covering the material will be made publicly available online.

3. Documentation and metadata

How will you document your data in order to make it findable, accessible, interoperable, and re-usable for you and others?

The data collected in the research will be presented in a thesis report by the researcher. The report will be made publicly available online after the research has ended.

4. Storage and backup during the thesis project

Where will your data be stored, and how will it be backed up?

The research data will be stored both locally by the researcher, and in a cloud-based storage solution provided by the commissioner of the research. The locally stored data will be backed up in the cloud storage during the research but will be disposed of after the research has ended.

Who will be responsible for controlling access to your data, and how will secured access be controlled?

The principal researcher is responsible for controlling the access to the data and will be the only one with access to the data during the research. The secured access to the data will be ensured by using exclusive access rights to the data files.

5. Opening, publishing, and archiving the data after the thesis project

What part of the data can be made openly available or published? Where and when will the data, or its metadata, be made available?

The data collected in the research will be presented in a thesis report by the researcher, and the published report will contain the only form of the data which will be made public.

Where will data with long-term value be archived, and for how long?

All the original data files will be systematically destroyed after the research has ended, including the backups. All the processed data files, outside the published thesis report, will be disposed of after the research has ended.

6. Data management responsibilities and resources

Who will be responsible for specific tasks of data management during the life cycle of the research project?

The principal researcher is solely responsible for data management during the life cycle of the research. The data will be temporarily stored, for backup purposes, in a cloud-based storage provided by the commissioner, but the principal researcher is the only one with access to the data.