University of Applied Sciences Technikum Wien

Advanced Software Engineering

Toolchain for Game Development

Master Game Engineering and Simulation

supervised by
FH-Prof. DI (FH) Alexander Hofmann

January 26, 2016

Contents

Part I
Software Maintenance in Game Engineering

Michael Knett and David Portisch

Abstract — Software Maintenance is an integral part of today's software life cycles, which also accounts for the development of video games. Even though maintenance has not historically received the same amount of attention like the other phases of the software life cycle, it forms a crucial part of software development as well. This paper is about the basics of Software Maintenance with a focus on how these are used in game development. In order to understand how maintenance is used in the development of video games, the general field of Software Maintenance is examined in this paper first. The term of Software Maintenance is elaborated and the different types of maintenance are discussed. Then the general maintenance process as well as some techniques for maintenance are described. The next part of this paper will focus on the most common forms of maintenance used in the gaming industry. This part will also show how the maintenance techniques in the gaming industry evolved over time. Finally, the topic of Community Feedback in video games is discussed. A case study of the handling of Community Feedback in the game Asterix & Friends shows how Community Feedback can be managed in the development of a browsergame.

Keywords — Software, Maintenance, Game Engineering, Patching, Updates, Community Feedback

1 Definition

Included in the life cycle paradigm for software, Software Maintenance forms an integral part of the software life cycle. Historically, the development part was categorized with more importance than maintenance. In general the other phases of the life cycle have received more attention than the maintenance part. However, this is now changing, as organizations try to keep the software operating as long as possible in order to gain the most out of their development investments.[1]

The result of software development is typically a program which satisfies the user requirements. However, the software product also needs to adapt and evolve when in operation in response to changes. This can be due to environmental changes, new user requirements or anomalies which are uncovered during the operation of the software. The maintenance phase in the life cycle of the software starts with the delivery of the product, but the maintenance activities start much earlier.[1]

Software Maintenance and maintenance in general are also formally described by ISO and IEEE Standards. The process of Software Maintenance is described in the IEEE 1219 Standard for Software Maintenance. It is described as the modification of a software product after its delivery in order to correct faults, improve its performance or adapt the software product in response to a changed environment. Maintenance activities prior to the delivery of the software product are also described in the standard, but only as an information annex. Furthermore, maintenance is described as one of the primary life cycle processes in the ISO/IEC 12207 Standard for Life Cycle Processes. The ISO/IEC 14764 International Standard for Software Maintenance describes the term similar to ISO/IEC 12207, but in this standard the focus is more on the aspects of maintenance before the delivery of the software product.[1]

Studies and surveys show that the major part of maintenance, over 80%, is used for non-corrective actions. This affirms that software evolves over its life cycle and thus maintenance is similar to software development, although it has its own unique processes. However, in contrast there is the common perception that maintenance is merely fixing bugs. This misconception originates from users who submit problem reports, which are in reality enhancement reports.[1]

Software Maintenance is needed to ensure that the

software product satisfies the user requirements over time. *Pigoski* lists the following reasons why maintenance has to be performed:[1]

- Correcting errors
- Correcting requirements and design flaws
- Improving the design
- Making enhancements
- Interfacing with other systems
- Converting programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrating legacy systems
- Retiring systems

Even though Software Maintenance activities are similar to those of software development, there are unique activities bound to Software Maintenance as well. Analysis, design, coding, testing and documentation are done by maintainers as well. Requirements are tracked and worked on just as in traditional software development. *Pigoski* mentions in [1] the following Software Maintenance activities: [1]

- **Unique Activities**
  Problem solving skills are very important for software maintainers. They have to analyze change requests, translate these into software terms and then identify the affected components. This gets even more complicated, if the maintainer has to maintain a software written by a third party. Furthermore, the maintainer needs intimate knowledge of the softwares code and its structure. This is used to perform an impact analysis, which identifies all the systems and system products affected by a change request. Finally the risk of actually implementig the change is determined, several potential solutions are provided and a recommendation for the best possible course of action is made.[1]
- **Supporting Activities**
  Supporting Activities like Configuration Management or Quality Assurance can also be performed by a maintainer.[1]

In IEEE 1219, the IEEE Standard for Software Maintenance, Configuration Management is listed as a critical part of the maintenance process. The modification request and problem reports need not only to be tracked. This is not sufficient. It is also important to control the software product and any changes made to it. This can be done by enforcing a Software Configuration Management process. [1]

Concerning quality it is more than just hoping that maintenance will increase the quality of the product. This needs to be planned and processes must be set up in order to support the maintenance of the software. [1]

Another important point is the Maintenance Planning Activity. The maintenance phase typically lasts for several years and thus accurate planning of resource needs is a key factor in maintenance planning. It is very important to include these resources, which include cost, into the project planning budget. Thus Maintenance Planning and the development of a maintenance plan should begin with the start of the project. This maintenance plan should be prepared during the development of the software product and address how the users will request modifications or report problems. [1]

1.1 Role of a Maintenance Programmer

As indicated before, problem solving skills are crucial for maintenance programmers as they need to analyze change requests, then translate them into software terms and finally identify the affected components. The maintenance programmer also needs to keep track of the changes made to the software through a Configuration Management Process.[1]

A challenge faced by a Maintenance Programmer can be a limited understanding of the maintained software. 40% to 60% of the time, as indicated by practitioners and researchers, is used for understanding the software to be modified/maintained. So program comprehension should be one of the key interests of a Maintenance Programmer. Often maintainers have a limited understanding of

the software they maintain and therefore need to acquire the knowledge about the software on their own, which can be even more difficult, if the software is not documented well.[1]

2 Types of maintenance

Maintenance is commonly divided into three parts. [2]
Perfective maintenance, which is about improving the product, takes up about 60% of the total time spent on maintenance. Corrective maintenance takes up about 18% and is about fixing bugs. Adaptive maintenance corresponds to adapting the application to a changed environment and takes up 18%. The remaining 4% are used on other forms of maintenance.[3]

These three different categories of maintenance (corrective, adaptive and perfective maintenance) were originally created by E.B. Swanson of UCLA. By using empirical data from the industry Swanson was one of the first to be able to examine what really happens in software evolution and maintenance.[1]

2.1 Perfective Maintenance

Perfective maintenance is defined by the Institute of Electrical and Electronic Engineers(IEEE) as "Modification of a software product after delivery to improve performance or maintainability". [4] This definition is very important for the game industry. Due to this every performance improvement, added feature or downloadable content(DLC) is considered to be perfective maintenance.

Thus the two goals of perfective maintenance are to improve *performance* and *other attributes* of the product. Both goals are achieved by applying tuning measures. Improving *performance*, also called *optimizing*, describes for example reducing the response time of the application or reducing the memory usage. The *improvement of other attributes* of the product is also called *Re-engineering*. In this context Re-engineering describes a technical improvement of the system. It shall be easier to operate, maintain and extend the software.[5]

The most widely used way for perfective maintenance in the game industry is for the use of DLC.[3] The strategy of developers is to release some downloadable content a few months after the release of their game.[3] This DLC can either be free or paid. The reason behind this is to extend the attention a game has for a few months more in order to get additional customers and keep existing ones happy.[3] With the use of this strategy developers can get an added income to keep maintaining the game after its release or divert the funds towards their next game.[3] Maintenance developers also get an additional feedback loop for their work. [3]

The Internet has revolutionized the perfective maintenance in the game industry, due to its easy ability to transmit patches and downloadable content. [3] Before this, when a bug or glitch was discovered, the only way to patch a game was to release an expansion, which in turn could introduce even more bugs. [3]

However, when optimizing or re-engineering the software, it implies that this is a planned change of the system in order to improve it. Thus it needs to be defined, what is actually an improvement of the system, how high the effort to achieve this is and whether this is cost-effective.[5]

2.2 Corrective Maintenance

Corrective maintenance is used after an applications initial release. It is the process to find and fix any bugs that have made it through testing.[3] Thus corrective maintenance is a reactive modification of the software product after its delivery.[1] The corrective maintenance then produces a patch which can be distributed via a network. These can be manually distributed or applied automatically via a download service like Steam or PSN. [3]

Corrective maintenance may also be needed after a perfective maintenance, because adding code and features increases the chances for more programming errors. [3]
For example, game developers may find players actively seeking and exploiting features in their game. Therefore glitching or breaking the game by accessing areas that were never meant to be explored or simply using a feature in a clever way and making

it unbalanced.[6]

Developers can fix issues as they arise, but by doing so, this can lead to more problems. To be able to correct code one needs to change code which, by it's nature, can lead to even more bugs. This type of error is named regression fault. In order to fix these issues even more corrective maintenance needs to be done, until all bugs have been erased. [6]

The defects which are addressed by corrective maintenance range from minor to serious major defects. The level of the defect depends on how much the defect affects the usability of the program and its environment. E.g. if a memory access error overwrites some data and the program continues running, the following results of the program are corrupted. The system is not usable anymore in this state and thus this is a serious defect. In contrast, it may be irritating if some text in the user interface is not correctly written, but this does not prevent using the program in most cases. Therefore this defect falls into the category of minor defects. The circumstances and the financial consequences of these two defects are entirely different and thus a major defect can weight 100 to 1000 times more than a minor defect.[5]

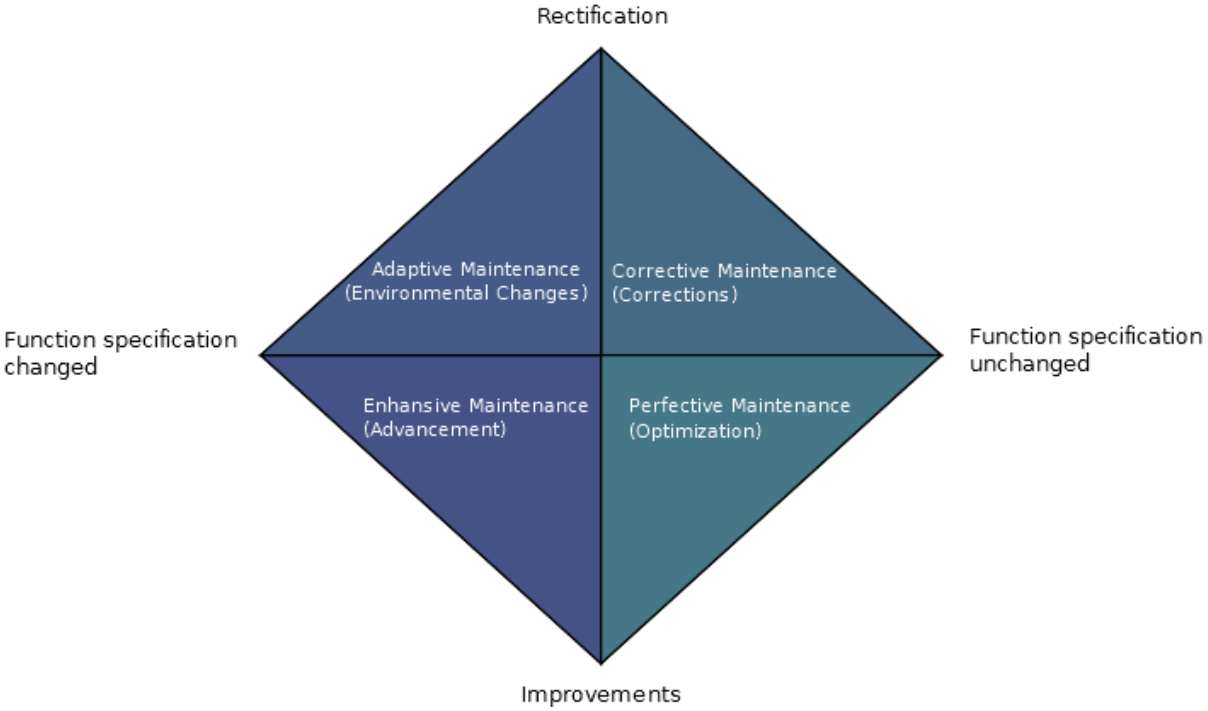

Figure 1: This figure which was adapted from the figure *Abb. 1-5* from [1] shows the four main activities of Software Maintenance and enhancement. Adaptive maintenance, corrective maintenance and perfective maintenance belong to the field of Software Maintenance. In contrast enhansive maintenance belongs to the field of Software enhancement, because it represents an addition to the products' features through enhancing the software. Thus it is different to adaptive maintenance where the products' features are not increased, but rather adapted to a changed environment.[1]

Therefore the corrective maintenance is divided into these categories [5]:

- **Maintenance and repair** This category deals with repair of the major defects. It is event driven and needs to react as soon as a major defect occurs. Thus an emergency service needs to be setup.
- **Defect Correction** Correcting minor defects is addressed by this category. The correction of these defects can be planned, because they do not prevent the operation of the software. Normally they are corrected all at once with the new release of the product.

2.3 Adaptive Maintenance

When an environment, where a game is running, is changed adaptive maintenance is used.[3] The software is modified after its delivery in order to keep it still usable in a changed or changing environment.[1] For example, if a new firmware update for a console is being released and a developer needs to change some code, otherwise it wouldn't work, this is called adaptive maintenance. Games are usually not effected by changes to the environment, especially on consoles.[3]

For software systems there are two types of environments which can change: First there is the functional environment where for example laws, specifications or business rules can change. On the other side there is the technical environment where for example Database-Systems, Operating Systems, the hardware or as mentioned before the firmware can change. Adaption is considered as the most important type of maintenance, because without advancement the software will *die*.[5]

2.4 Preventive Maintenance

Preventive maintenance, which was introduced after perfective, corrective and adaptive maintenance, is performed in order to prevent problems from happening. It is defined as a modification of the software after its delivery in order to detect and correct hidden faults, before they actually become active faults. Thus this type of maintenance is most often used in software systems where safety is a crucial concern.[1]

3 Techniques for Maintenance

Techniques specific to maintenance can be used in order to achieve effective Software Maintenance. *Pigoski* lists in [1] the following maintenance techniques: [1]

- **Program Comprehension**
  Programmers spend a considerable amount of time in reading and comprehending source code in order to implement changes. Thus it is crucial to aid this process with tools, techniques and processes. A key tool for the program comprehension are code browsers and an understandable and concise documentation can aid greatly as well. [1]
- **Re-engineering**
  In re-engineering a system is first examined and then changed in order to recreate it in a new form. This also includes the subsequent implementation of the new form of the system. Often re-engineering is not used in order to increase maintainability, but rather to replace aging legacy systems. [1]
- **Reverse engineering**
  Reverse engineering is the process of analyzing a subject system in order to get a higher level abstraction of it which is easier to comprehend. Thereby the system's components and their interconnection are identified to create this high level representation of the system. Reverse engineering is a passive process, because it does not change the system in any way nor does it yield a new system. There are different types of reverse engineering like re-documentation, design recovery and data reverse engineering. [1]
- **Impact Analysis**
  Impact analysis has two major tasks. First it checks which systems and system products are affected by a change request and second it estimates the resources which are needed in order to implement this change. This process

is performed after a change request enters the configuration management. [1]

4 Maintenance in Game Engineering

This chapter shall give an overview over terminology and techniques concerning maintenance in the area of Game Engineering.

4.1 Maintenance Terminology in Game Development

- **Hotfix**
  When a critical bug is found in the live-environment it needs to be fixed as soon as possible. If this can be done, without taking down the live-environment, then this is called a Hotfix. For example in a server-client environment this could only affect the server-side. That means the user would not need to restart and update their client. A Hotfix is generally made outside of normal testing-procedures, which means it is more error prone and does not guarantee that it will fix the problem. This is corrective maintenance.[7]
- **Bugfix**
  When a critical bug is found, and the environment needs to be taken down in order to fix it, then this is called a Bugfix. A Bugfix should be tested beforehand to confirm that it really fixed the issue. Bugfixes are usually grouped together before they are deployed, so the user only needs to update their client once. This is corrective maintenance.[7]
- **Tweak**
  A deployed/existing feature needs to be adjusted. This can be a combination of several hotfixes on the server and bugfixes on the client side. Depending on the situation, this can be adaptive or corrective maintenance.[7]
- **Content/Feature - Update**
  New Content and Features are a type of perfective maintenance.
- **Technical Maintenance**
  When the server-infrastructure needs to be updated, or another server needs to be added, that is called technical maintenance. Theses things are usually met with a small downtime of the system and is done at a time where the least amount of users are active. This is adaptive maintenance.[7]

4.2 Maintenance techniques in the early game industry

In the early game industry global Internet access for the general public was a far of dream. So games that were developed needed to be as bug-free as possible because released games had only a very limited amount of methods available to deploy patches.[3]

- **Addon/Sequel**
  One way to deliver a patch to the user is via addons. Addons are expansions of the original game content. Additional content however, can introduce new bugs as well.[3]
- **Re-Release**
  Another form of delivering a patched version to the user is with a re-release of the game. This was also one of the earliest form of DLC as developers could attach new content to the re-released version as well. For the user however, this meant they had to buy the same game again if they wanted a patched version or the new content. [3]
- **Manual Patching**
  With the rise of the Internet game developers were creating patches for their games more frequently. They then made them available on their (or other) websites for users to download and apply manually. Care needs to be taken when installing such patches as the version prior to the installation needs to match the patch as well.[3]

4.3 Today's maintenance techniques in the game industry

In modern times most users have access to the Internet. Therefore delivering a patch to the user is easier than before. This however can lead developers to code sloppier because they can just patch it afterwards[3]. Companies will try to release their games when no other major game is being released, as to not split the customer attention to multiple games and therefore potentially splitting profits for both games [3]. When a critical bug is found, that for example prevents the game to be finished under certain circumstances, then usually a game is delayed. On the other hand, the game could be released and later be patched as soon as possible. This would give the image that the company is lazy and unprofessional, which can lessen the profit made[3].

An obvious advancement of the earlier manual patching is an automated patching process. This can either be done via a download platform, like Steam or PSN, or via the games own launcher.

It is inevitable that software, especially games, have a need for change. These changes happen mostly because of a changing environment.[8]
One possible approach is to design, develop and maintain a system that is easy to change and each change should have as few impact as possible on the whole system. [8] This is known as Change Isolation. Methods that can utilize this can range from code level construction of classes, memory management or even up to a business level purchase of new servers and how to change or integrate them into an existing server cluster.[8]
Change Isolation can give us:

- Break a complex implementation into a modular design that can be easier understood and maintained.
- Extend a system/modules lifespan.
- Breaking down code results in modules/components that can be reused.

For example in *Bungie*'s old *blam* engine, which was used for e.g. the *Halo* games, the game code was accessible from any layer of the engine. This may sound great, because it simplifies the development of the game a bit, but the true implication was that it was hard to share code between their games. The reason for this was that the engine was tainted with game logic, which of course was not used in the next games. This is a point which the development team at *Bungie* took into account, when they developed their new engine for the game *Destiny*.[9]

However, *Change Isolation* is not always fully applicable. *Butcher* from the company *Bungie* explains in his talk [9] about the development of their new engine for the game *Destiny* that the dependencies of modules in different levels of the engines are stronger or weaker according to the layer they are in. He differentiates between the *Core-Engine* and the *Feature Components*. The *Core-Engine* are features which other engine features are built upon like Application Lifecycle, Resource Management, Streaming System, Core Render Architecture or the Core Content Pipeline. *Butcher* mentions that these *Core-Engine* features tend to be tightly coupled into on another because they have shared assumptions about data formats, object lifetime, state management and multithreading. Every component needs to follow these *rules* or else the game is buggy and/or unstable. This is one of the reasons why the development team decided to develop a new engine. The old *blam* engine had shared assumptions like single-threading, one platform, and a simple content pipeline which had no system for patching content post-release. The *Feature Components* in contrast build up upon the Core-Engine and tend to be more loosely coupled and talk to each other through interfaces.[9]

5 Software Maintenance Guide

Event though the maintenance phase of a software starts with the delivery of the product, the maintenance activities start much earlier.[1] This section shall give an overview over how maintenance can be integrated into the development of software.

The development of a software can be categorized into the five different phases of product life-cycle. At each stage a different amount of maintenance happens:

- **Planning Phase**

  Even in the planning phase maintenance should be considered. Creating coding guidelines can be used to structure code in a unified way, making maintenance work easier later on. The scalability of the software/game should be taking into consideration as well. Adaptive maintenance is, for example, used when upgrading a server infrastructure to take a heavier load during peak hours. It can therefore be planned that the servers can be swapped out for a newer generation or simply upgraded in their number. Of course the software needs to be prepared for this scenario.

- **Development Phase**

  During the development phase of a software product the software is usually developed in the three sub phases of *Design*, *Programming* and *Testing*. These phases are repeated over and over and can lead to a change in the technical concept of the product. Thus correction, adaption and extension of the technical concept is performed during the development phase and can be seen as the first work on System Maintenance.[1]

- **Evolution Phase**

  The first big Software Maintenance occurs during the *Evolution Phase*. The Software Product is still extended, but the main focus lies on the maintenance of the software version which is used in production. This is a critical task and has the highest priority. Corrective maintenance and adaptive maintenance is now used to correct the occurring defects and perform adaptions. The least priority has the task of enhancement of the software product. In the Evolution Phase re-engineering can be performed as well, to achieve technical perfection. However, most times there is not much time left from the three previously described fields, so that in practice re-engineering is only performed in emergencies.[1]

- **Maintenance Phase**

  During the Maintenance Phase only maintenance and correction are performed with a limited capacity. This phase starts, when the software product can't be enhanced anymore and thus only needs to be kept running anymore.[1]

- **Retirement Phase**

  If for example the functionality of the software does not suit its usage, it is retired. It may be that the system is still used by some users and thus it can still stay in production for a while. However, in this phase no maintenance happens anymore.[5]

Besides the software life-cycle according to [5] the software maintenance activity in the development can be divided into three parts[5]:

- **Maintenance and repair service** For the immediate correction of major defects.
- **Rectification service** For the periodic adaption of the system to a changed environment and the correction of minor defects.
- **Re-engineering service** Responsible for the occasional revision of the system (perfective maintenance)

As soon as the maintenance activities for a project are set up, these three parts can be found in all the phases of the software development life-cycle.

When maintaining the project the *process of change* needs to be taken into account. As iterations of changes happen, mostly through adaptive or perfective maintenance, the system will slowly drift apart from the original design. The system may still give the originally intended output, but it can use different algorithms than it's creators intended. This is called *structural decay*.[8]
Structural decay needs to be kept in check with a design that is focused on change and documentation.[8] Otherwise a system slowly becomes unmaintainable because future maintenance developers will have no idea how much of the original documentation is still valid. Examples can be found in big companies where undocumented legacy systems are still used because of the fear that some small change could break everything.

6 Community Feedback

6.1 Maintenance and Community Feedback

Some types of maintenance have to do with improving the product (perfective maintenance) or fixing bugs (corrective maintenance). In the development of games Community Feedback can be used as a tool to support these maintenance types, but also to be a part of the whole development process. This accounts for MMOs, browsergames, but also for games which are in *early access* and/or funded via crowd funding.

In his talk [10] at GDC about *The Long Dark & Community Influenced Development* Lierop depicts the way of *Community informed Development*. He describes it as "A process by which you take feedback from your community and incorporate it into your decision-making"[10]. It is important that this does not mean that they make the game the community wants. They are developing the game they want to make with the community as a sounding board of ideas.[10] This is quite similar to the approach *Sproing Interactive Media GmbH* is following with the development of the Browser Game *Asterix & Friends* concerning Community Feedback. They gather feedback from the community, listen to them, but finally the development team themselves decides what is implemented and what not.[7]

So there is a huge difference between *asking the community to give you feedback* and *asking the community to tell you what to make*. Asking the community for feedback involves them into the process of refinement (like *perfective maintenance*) and the developers have the chance to test out ideas. On the other hand, asking the community to tell the developers what to make puts the players into the *driver's seat*. However, the gamers are not equipped to do this.[10] Some gamers may also have a very strong interest into video games as a medium and have informed themselves about the development process, but they don't have the same knowledge about the game as a game designer of the game has. Thus they may have a great idea, but are not aware of the implications the idea has on the game as a whole, which a game designer of the development team is aware of in contrast.[7] If the developers listen to the opinions and suggestions of the players all the time, it is very likely that the finished game does not represent the team's strong vision of the game as it would have otherwise. The community can rather be seen as another voice which the developers can listen to in order to improve the game.[10]

This way of game development is quite different from the traditional way. With this way of development early on from the beginning of the development of the game, the developers can test out ideas in the concept development phase. This means that feedback is gathered early on from the community and the developers try to understand what they like. It is a *live* development similar to the development of an MMO and browser-games. There is a direct connection to the community and feedback is flowing in both directions. Thus this is a much more transparent process. The developers have direct access to the community and vice versa.[10]

In comparison the traditional way of game development is mostly performed in secret, until the project is done and released. There is very little input from outside and thus also very little validation. Often the result is then misaligned with the expectations of the audience. Furthermore, the communication is mostly done by the PR and marketing.[10]

Besides all the Community Feedback it is important to perform analytics as well. This is essential, because the people who comment and give feedback the most do not need to be the majority of the people who play the game. A good example of this happened during the development of the game *The Long Dark*. With the feedback from the community the developers of *The Long Dark* could identify two types of players who play their game. On the one hand they had players who wanted an emotional experience in the game and enjoy the atmosphere. But on the other hand there were players who wanted a hardcore survival experience with the game. So the team introduced three different experience modes in the game which differ for example in difficulty: First there is the *Pilgrim* mode, which is for the players who want an emotional experience from the game and enjoy the atmosphere. The *Hardcore* mode offers a hardcore simulation and the *Voyageur* mode lies in the middle of the other

two experience modes. Interestingly, the hardcore gamers tended to be more engaged with the game, commented a lot more and gave a much feedback. If the developers had only read the comments and feedback about their game, they may would have come to the conclusion that the majority of their player-base are hardcore players. However, according to the metrics the developers collected with the analytics, about 60% of their players played the *Voyageur* mode, about 25% the *Pilgrim* and only about 12% picked the *Hardcore* mode. So in reality the majority of the comments and feedback did not represent the majority of the player base. Thus it is important to perform analytics and collect metrics as well in order get objective data. The developers should understand how the players of their game actually break down in categories, or they could think that the players who give the most comments or feedback form the whole of their community, which does not have to be true.[10]

According to *Lierop* in [10] it is important that the developers stay to their vision. Community feedback can be a valuable tool, but it is also important to consider that this also implies risks to the direction and the vision of the game.[10]

6.2 Community Feedback in Asterix & Friends

This chapter is a Case Study about the role of Community Feedback in the development of Asterix & Friends. It is based on an interview via e-mail with Felix Seibert from Sproing Interactive Media GmbH which took place from 2015-12-16 to 2016-01-07.

Asterix & Friends is a Free-2-Play browsergame based on the brand of *Asterix and Obelix*. The player has to build up a gaul village and defend it against the Roman legionaries. Thereby the player needs to gather resources, refine them and construct buildings. Various tasks and missions need to be accomplished as well.[11] The game is developed by Sproing Interactive Media GmbH and published by Sproing Publishing GmbH. It was released in April 2013 for PC.[12]

How Community Feedback is given depends a lot on how much a player knows about the development process of a video game in general. It is important that the game developer knows about that and it can help to be aware how much the players who give the Community Feedback know about the development process. In general there can be two types of players who give feedback:[7]

- Some players have a solid knowledge about the reality of game development. With a distinctive interest into the media of video games, they may inform themselves a lot about the development process of them as well. This has an impact on the feedback they give.
- For the other players this kind of knowledge is missing. They may like to play video games a lot, but don't deal with the development process of a game. Of course this has an influence on the feedback they give and can lead to utopistic requests which are not possible for the development team to implement.

Of course both types of players can give good suggestions, but it can help to keep in mind which types of these players mainly give feedback for the game.[7]

In general Community Feedback for Asterix & Friends is gathered via these four platforms:[7]

- **A forum**

  The *forum* for the game also provides the possibility for the players to provide feedback.

- **A Support-form**

  Even though the primary purpose of the *support form* is to report bugs, it can also be used to provide feedback about the game.

- **A Support-Address**

  The *support address* works like the *support form*, but it does not have the predetermined structure like the *support form*.

- **Facebook**

  Feedback can be provided through the comment function of *Facebook*.

With Community Feedback there is no guarantee of qualitative and elaborate feedback. Mostly the spontaneous feedback by the players, which was not explicitly asked for by the developers, can be a bit

chaotic. For example Facebook comments are informally written and often only contain rough ideas than elaborated suggestions. In the forum *idea threads* are also started where the players discuss their ideas on how the game could be improved or changed. The feedback through the support forms is a bit different. Most times these are more elaborate.[7]

The development team of Asterix & Friends has made better experiences, when they guided the process of the Community Feedback. This way they get feedback about topic which are relevant for the team at the moment. After an update a *Feedback-Thread* is created in the forum or a link to a survey is posted on Facebook. However, spontaneous feedback is still very important for the team, because it can contain ideas and approaches which the development team didn't think of before.[7]

In the development of *Asterix & Friends* all the essential design decisions are made by the development team. Concerning the Community Feedback this means that the team acts as a *gatekeeper* who decides which changes or features are implemented and which are not. Thus the Community Feedback which is received by the development team is read and reviewed. Most times a member of the development team can evaluate right away, whether a suggestion can be implemented or not, because they know the restraints and constraints of their project. The development team of *Asterix & Friends* uses 4 aspects to review a suggestion from Community Feedback:[7]

- **Technical Constraints**

  Sometimes the technology which is used to build the game prevents a feature or a suggestion, because it is simply not possible to be implemented with it. In *Asterix & Friends* for example they got the suggestions from multiple players to change the chat, but that was not possible with the used technology. Changing the technology would be possible, but it would cost much time and effort and thus slow down the development of other features.

- **Good design**

  The more people are involved into the design process of game, the more difficult it is to maintain a clear vision of the game. Even though an idea is technically possible and well-thought-out, it may has implications on the game which are not good for the game as a whole. Thus the ideas also need to be checked by a designer who knows the game very well. *Design by the fans* does not have to be a good thing, because even though a person likes to play the game very much, this does not mean that they know what is best for the game.

- **Resources**

  Every project in a company has to work with a contingent of resources they get assigned. Thus it is not possible to implement some suggestions, because there are not enough resources to do so. The implementation of some suggestions would require to increase the team size of the development team significantly which is very often no option.

- **Publisher and Licence holder**

  Ideas and suggestions can also be rejected by partners like publishers. *Asterix & Friends* is the game for a worldwide well-known brand and thus the development team has to follow strict licence specifications. Quite often this is the reason, why ideas had to be discarded. Even if lots of players wish a feature to be implemented, it may is impossible due to licence constraints.

According to these points the development team of *Asterix & Friends* decides whether a suggestion can be implemented or not. However, even if an idea is categorized as implementable, this does not mean that this will happen immediately. The idea or suggestion needs to be assigned to a developer and may also needs to be elaborated further. If this task then has a low priority it can take some time until it is finally developed and implemented. When the task is finally done it is tested on the development server and afterwards included in the next update for the live server.[7]

The changes to the game are then documented, so that it is possible to retrace who, where and when changed what. This is essentially important if new people join the development team or existing members leave the team. For *Asterix & Friends* an internal wiki is used in the company to document the changes. All the changes on the live server are

also documented in the patch notes which are also distributed to the players. However, for the patch notes it can be possible, that some changes are not included, because they are not relevant and visible for the players or the development team does not want share this piece of Information. Most times, the patch notes are published at the day of the update, or shortly before. This way the players know for sure what changed in the game and thus can give adequate feedback.[7]

When a new feature or change is implemented the live server needs to be updated. For *Asterix & Friends* it was clear from the beginning of the development that there will be down times for the updates and the maintenance of the game and server. The development team follows the principle that there are down times when they are needed. Thus the server is updated, whenever there are enough new features and changes for an update, or if it is absolutely needed due to unavoidable technical changes and/or bugfixes. Down times are announced through all the communication channels in advance for the players. A good communication is needed so that the players are aware of the downtime. Furthermore, 30 minutes before the down time a timer is started in the game to make the players which are still in the game aware that the game won't be playable for a short time. The team tries to keep these down times as short as possible and tries to perform them when only a few players are online. Most times the down times for *Asterix & Friends* are performed in the late morning and the team tries to limit them to 3 - 4 hours.[7]

It didn't happen that often that Community Feedback resulted in a completely new feature. Most times it lead to the improvement and adaption of existing features and systems of the game.[7]

A good example for a bigger change based on Community Feedback is the implementation of the chat in the game. Many players asked for a chat to be integrated into the game and so it was implemented by the development team. Interestingly, this also resulted in a new feature, because after a short time there were many problems with trading frauds. These players didn't respect an arranged trade and thus stole the resources from the other party of the trade. The players then asked for a safe possibility to trade and so the in-game message system was extended with a safe trading option by the development team. However, most of the biggest improvements of the game originated from the development team themselves. One main reason for this is, that as mentioned before, the development of *Asterix & Friends* needs to follow strict licence constraints. These changes need to be discussed with the holder of the licence and to be adapted where required.[7]

7 Open Topics

There are some topics which are not dealt with in this paper, because it would go beyond the scope of it:

- Tools which can be used in order to support maintenance activities. Such tools include code browsers and documentation tools.
- Community Feedback tools like social networks and forums.
- Tools which are integrated into a game in order to support maintenance and the gathering of Community Feedback. These tools include analytics tools and in-game feedback tools.

8 Acknowledgements

We would like to thank Sproing Interactive Media GmbH for their cooperation concerning the Case Study and especially Felix Seibert from Sproing for his detailed and elaborate answers to all our questions. We could get a very interesting insight into how Community Feedback is handled in the development of *Asterix & Friends*.

Part II
Self Publishing

Lukas Frühstück and Jakob Schade

Abstract — As an independent game developer a successful launch of a game it especially crucial. The task of self-publishing may sound easy but there can be significant challenges. Different platforms and manufacturers have different rules and requirements which have to be met when releasing a game for their system. The following paper explores which options are available for an indie-developer to release their game, how big the challenges for different gaming platforms are and what general strategies should be kept in mind.

Keywords — self-publishing, indie-developer, PC, Steam, iOS, Android, consoles

1 Introduction

The classical way of releasing a newly developed game involves a publisher who has the resources and necessary industry contacts to quickly distribute a game over many platforms using different distribution channels. However, as an independent developer without any publisher it can be significantly harder to successfully release a game on many different platforms. On the other hand, without a contract with a publisher the developer retains complete control over his game and is free to decide about its fate.

Over the last years more and more indie-game studios started self-publishing their games. On the PC it is comparatively simple to achieve this. On consoles on the other hand it can be more difficult as the manufacturer enforces certain requirements and has to agree that your game can be published. The rules imposed on developers vary between different target platforms. Some are very restrictive whereas others allow the developer more freedom.

With the ongoing trend of indie-games most console manufactures have created special programs for developers to allow self-publication. This significantly simplified the release process but it is still more difficult compared to self-publication on the PC.

Self-publishing on mobile platforms is, compared to gaming consoles, also comparatively simple. One of the big difficulties here is that especially for the main platforms, Apple and Android, the market is very oversaturated and it can be hard to get the game recognized and discovered by players

2 PC

The PC is a very versatile platform with many different ways to distribute a finished game. Nowadays the primary way for indie-developers to release a game on PC is to use the various available online distribution platforms that are available. Distribution with physical media is still common for games but it is generally not feasibly for small developers due the required financial expenses.

2.1 Steam

When someone thinks about purchasing and downloading a game for pc, steam is probably the number one store where most of the gamers would start to search for their favored game. On steam you can find nearly any game, but what is the reason that so many publisher or game developer decide to release their games on Valve Corporation's platform?

When steam started in 2003 it was a platform to play Valve's own games on Windows PCs. Since then it grows and grows. Over the years more and more publisher decided to release their games with steam which currently has a userbase of more than 125 million users.

While this sounds nice it may be a problem to get the attention for the game. Steam currently offers more than 4500 games as the announced on GDC 2015. The get some attention for a game or to get it in steam store primarily it has to go through the Greenlight process, where gamer can decide if they would buy a game if it would be released in the store. But more on this later. [13]

Publishing Guidelines

To publish a game on Steam you have to own the rights to sell the game or have a specific authorization to represent the developers. Porn, inappropriate or offensive content, warez or leaked content is not allowed. The game must not allow cheating or hacking. Also games using copyright materials such as assets without permission from the owner are not allowed. In Greenlight process soliciting, begging, auctioning, selling, advertising, referrals racism, discrimination as well as threats of violence or harassment, even as a joke, are strictly forbidden. [14]

Costs

To get access to submit a game to steams Greenlight you have to pay a one-time Greenlight submission fee. The amount of the fee is 90 euro and is fully donated to Child's Play, a charity dedicated to improving the lives of children in over 70 hospitals worldwide. Once paid, this steam account is allowed to post and update an infinite amount of games or software within the Greenlight process. [14]

Figure 2: Steam Logo

Steam does not officially talk about additional costs. In the official announcement they write that they will get in contact with the developer to talk about revenue. Some unofficial reports say that steam receives about 40% of the prize of the final game.

Pricing Model

Steam offers many different pricing models. In the official Steamworks documentation valve writes that they will get in contact with the developer to find the best price for the game.

If a game is in store the publisher has the possibility to set a discount for the game. Mostly they are only for a limited time available. It is also possible to combine games and release them as packs with a little discount. Also an old game as additional content is part of the possibilities. To promote the game the developer has also the right to offer it for free.

Steamworks

Steamworks is a platform inside Steam. You can provide additional possibilities for your customers when they connect your game with their steam account. [15]

Steamworks supports matchmaking, achievements, anti-cheat technology, micro-transaction and is available for free for download and for retail games.

Steamworks also says that it is very easy to be implemented into a nearly ready game and brings not much overhead with. More information about Steamworks can be found here: https://www.steampowered.com/steamworks/

Greenlight

Greenlight was started in summer of 2012. It is a possibility for small and new developers to present their games for a small fee. Than Steam community members have the chance to support game ideas they like and make it possible for the developers to release it in store.

2.2 GOG

GOG.com, formerly Good old Games, is a distribution service for DRM free games and movies. It was originally founded in 2008 to sell classic games which are not further available in retail stores. In 2012 they changed their market and started to sell more recent title too.

Figure 3: GOG.com Logo

Comparing GOG.com with different online distributer is a little bit difficult, because main stream titles are not their main selling point. But on some specific games they are the second best behind Steam if they are not event better than Valve's platform.

Until today GOG.com has more than 1300 games in its library and they become more every week.

Publishing Guidelines

Every game will be reviewed by itself. The GOG.com Team will get in contact with every developer who submits a game and will tell them, what they think about their game.

Costs

There are no one-time costs at GOG.com when a new game is submitted to their indie platform. The revenue of all sold Games is split between GOG.com and the developer. GOG.com receives 30% of the earnings.

Get in Contact

To get a game into the GOG.com store the developers have to submit it on the indie website of GOG.com (http://www.gog.com/indie). GOG.com will get in touch with the developers to discuss if the game has a change to be released. They say from themselves that they will respond to every request and if they reject a game, they will tell the developers exactly why they decided so.

Important for GOG.com is, that the game is DRM free. Else there is no chance that the game will be released on their website. Also all contents of the game have to be originally made by the developers themselves or at least the developers have to hold all rights for assets etc.

Each and every game will be rated for itself if it is worthwhile to get a place in GOG.com store. The distributer says "We're not machines. We talk." Every game will get its individual evaluation. [16]

2.3 Humble

The Humble Store is well known because of the Humble Bundle. But aside from this cheap bundles they also provide download games or codes for steam. Humble Bundle was launched in 2010 but it took three additional years until the Humble Store itself went online. According to the Humble Bundle team, they "wanted to create something that would allow developers to easily sell their games through their own web site as well as provide a painless buying experience for purchasers". [17]

Store or Widget?

There are two different ways for selling a game via Humble Bundle. The developers can choose to offer the game in the humble store, where many different games are available. But they can also decide to sell the game from an own website. Therefore Humble Bundle provides a widget which can be used to be implemented in the games online appearance. [18]

Store

The Humble Store is the "normal" distribution medium. The game will be available alongside other games. The games can be provided either as steam keys or as DRM free downloads. To submit a game the developers have to fill out to form on this site (https://www.humblebundle.com/developer/store/application). The Humble Bundle Team will get in touch with the developers to discuss details.

Costs

The developers receive 75% of the revenue, which does not mean that Humble Bundle gets 25%. They just keep 15% and the other 10% go to charity. There are no one-time costs at Humble Bundle. [18]

Widget

The Humble Widget on the other side is more or less a possibility for the developers to sell their games themselves. After a registration on the Humble Bundle website the developers get some code to sell the game directly from its website. [19]

Costs

The Costs for this options are very low. Humble Bundle only wants 5% of the revenue if the developers sell the game outside of the Humble Store.

Figure 4: Humble Store Logo

Humble Bundle says about itself that the want to support the developers. They don't want much revenue and offer many different methods to distribute the game. Downside is, that not every distribution model is

Pricing Model

Humble Bundle offers Humble Pricing. The developers can choose if they want to use it. It automatically calculates different currency values and rounds them up to get more beautiful prices.

In the backend they also have full control over possible discounts for their games.

The Humble Bundle Team says itself that they want to support the developers. Humble store as well as Humble Widget offer really good revenues for the developer. To use all possibilities like sell Steam keys the game has to be submitted on the other distribution platform to, which could be a little bit more difficult than releasing on Humble Bundle.

2.4 Windows Store

Windows Store is Microsoft's answer to Apple's App Store or Google's Play Store. But while the second and the third primarily offer mobile apps, the Windows Store shall also distribute PC Games especially from indie developers. Windows Store was first published with the Consumer Preview of Windows 8 in February 2012. Since then every new Microsoft operating system received a version of it. In 2015, together with the launch of Windows 10, the Xbox and the Windows Phone stores were merged together into the Windows Store. [20]

There are currently more than 669.000 apps available in the Store and because every new Windows Version is delivered with a preinstalled version of it, there is a gigantic userbase.

Costs

To sign up, new developers have to pay a one-time fee to grand access. For indie developers the amount is 19$. A company account has to pay 99$.

These allow to publish apps on Windows NT and Windows Mobile Store.

Students are supported through the Dreamspark program. Microsoft also splits the revenue of the sold apps. They keep 30% of the income and pay the other 70% to the developers.

Pricing Models

The developers can choose if they want to offer their app for free or for a specific price. If they decide for the second one, there are many other options to choose from.

For example it is possible to offer a free trial of the game or application. Also the developers are allowed to choose a market where they want to sell their game and decide on own prices for them.

Offering a discount is also a possibility.

Analytics

Aside from the frontend of the Windows Store, Microsoft offers a developer portal, where the developers can watch over their published apps. There they can find a summary and a detailed report about downloads, sales, finances, ratings, etc.

Figure 5: Windows Store Logo

It is also possible to lay filter over this information. For example it is possible to get specific information for a single region.

Format Windows Store supports Win32 Application or .NET-Framework Applications. These can be packed for distribution in Windows store. For this is Microsoft Application Virtualization (short App-V) used to allow sandboxing.

Windows Store is a little bit different than the other three presented stores. It is more similar to the mobile platforms, that will be shown further in this paper.

3 Mobile

Publishing apps and games on the common mobile platforms Android and Apple iOS is comparatively easy. Both platforms provide app stores which allow developers to submit their apps. The biggest challenge when releasing a mobile game is not publishing it. The difficult part is to stand out and gain public exposure in a market with millions of other apps and games.

3.1 Apple App Store

The Apple App Store is the apps and games marketplace for Apple's mobile devices, primarily the iPhone and iPad. The store hosts over 1,2 million apps [21] and counts more than 100 billion app downloads. [22]

Apple's App Store is one of the key factors that caused the boom of mobile applications and mobile games. Founded in 2008 it gained enormous popularity with the success of the iPhone.

Nowadays the market is oversaturated with many different games and it can be very hard to get attention. Even if the odds of achieving financial success on the App Store are very small, it can be well worth it, should a game be included in one of the store's featured lists. At this point it is easily possible to get many downloads and to gain huge earnings.

Publishing Guidelines

To publish a game on iOS an Apple Developer Account is necessary. With this account it is possible to submit any app to the App Store. However, before it will be released the game is reviewed by Apple and if it does not meet their criteria it will not be published. The process of reviewing a new App can take multiple days and has to be considered when planning a release.

Apple is very strict about their rules and does not allow certain types of content on their store. The exact content guidelines are rather loosely defined in "We will reject Apps for any content or behavior that we believe is over the line." [23], but in general it can be said that games containing pornography, racism, certain types of violence, certain religious themes or personal attacks are likely to being rejected.

Additionally, if the game looks like "it was cobbled together in a few days" [23] or does not do "something useful, unique or provides some form of lasting entertainment" [23] it will be rejected too.

Costs

Developing apps and games for iOS is free but publishing them on the App Store requires joining the Apple Developer Program which incurs an annual fee of $99 for either a private person or an organization.[24]

Additionally, Apple keeps 30% of the revenue a game or an app generates – either through direct sales of In-App purchases. [25]

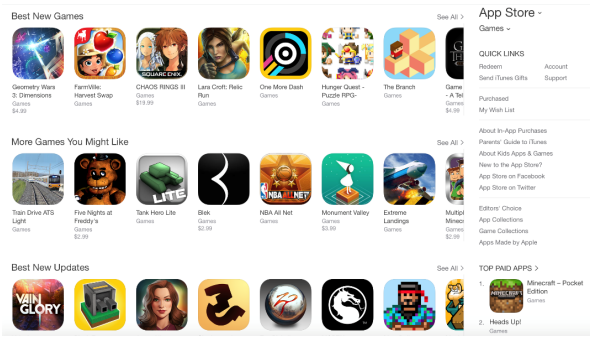

Figure 6: Apple App Store overview

Pricing Models

The Apple App Store allows for different business and pricing models to monetize a game. The most straightforward one is to publish a game as a premium app which requires the customer to pay for it once when downloading it. After that he can play the game without restrictions on his devices.

It is also possible to release a free app. This can be used for example to create a demo version which gives the customer the option to try the game an encourage him to buy the full premium version.

Another option is In-App purchases which can be used to sell premium upgrades, special items, consumables and many other digital goods directly in the app. This pricing model can be hugely successful. For example, in 2015 the game developer Bethesda generated profits of more than $5 million with their free game "Fallout Shelter" which contained one single item as an In-App purchase. [26]

The final pricing option is to publish a free app which contains advertisements and generates revenue this way. There are many different types of ads that can be integrated into an app – from simple banners to big full screen popups and even video clips. These ads are provided by different advertising programs. Notable examples for this are Apple's own iAd and Google AdMob. The revenue generated by in-app advertisements can vary drastically and depends on the advertising provider, on the number of users an app has and on the number of users that click on the advertisements.

Promotions

Apple will choose and promote games that they like by putting them on the front page of the App Store, by integrating them in lists of featured apps in certain categories of by giving out awards like the "Apple Design Award". The process of getting an app promoted cannot be influenced by the developer but it can be crucial for the success of a game on the platform.

As an example: The game "Blek", developed by the Austrian brothers Denis and Davor Mikan, was released in December 2013. It received general favorable reviews and sold about 30.000 copies in the

first 2 months. Then it received the "Apple Design Award 2014" and got featured on the store's front page. In the following two months additional 500.000 copies were sold. [27]

Promo Codes

Apple provides also the capability to generate promo codes for an app or for in-app items which can be used in custom marketing campaigns to promote a game.

Apple Game Center

As an additional feature for games published on the App Store, Apple provides the "Game Center". It is a social gaming platform with features like multiplayer, friend invites, leaderboards and achievements. The Game Center can be used by any game on the App Store free of charge. [28]

Apple GameKit

The GameKit framework works in conjunction with the Game Center. It provides peer-to-peer network multiplayer functionality for phones connected via Bluetooth or WiFi in the same local area. Furthermore, it also contains functionality for an in-game voice chat system which can be implemented in games. [29]

3.2 Android

When publishing games for devices running Android different things have to be considered in comparison to Apple iOS. Android is a open ecosystem with more than 24.000 [30] different device models. The primary publishing platform for most of these device is the official Android Play Store from Google. But as Android is an open platform, other distribution channels are allowed too and there are devices out there that have only access to third part distribution platforms and not the official store. To reach as many Android users as possible, these options have to be considered.

Play Store

The Play Store was founded in 2008 and is the official distribution channel for apps, games and other digital goods on the Android platform. It is developed and hosted by Google and contains about 1,4 million apps [21] with more than 200 billion total downloads. [31]

As with the Apple App Store these numbers show that the market is heavily oversaturated and it is difficult to gain enough attention to become popular and financially successful.

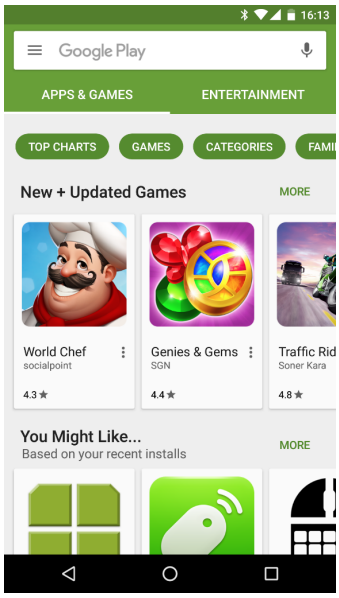

Figure 7: Android Play Store overview

Publishing Guidelines

To publish a game on the Play Store a Google Developer Account is necessary. With this account it is possible to submit any app to the Play Store.

The general criteria for apps and games that are allowed on the Play Store are similar to those on the Apple App Store. No explicit sexual content, violence, bullying and hate speech are allowed.

However, the process of submitting a new app is generally much faster compared to the Apple App Store as the Play Store does not have a lengthy review process. But this does not mean that apps are not reviewed before being published on the Play

Store. Before 2015 all new apps were immediately accepted but since then a half-automated process is in place that checks new apps before making them available on the store. [32]

Costs

To create an Android Developer Account an initial, one-time fee of $25 is required. Additionally, 30% of the revenue generated by app sales and in-app products go to Google.

Pricing Models

Similar to the Apple App Store, the Play Store supports four different pricing models.

Premium apps are sold for a price determined by the developer. Once bought they are available on all devices of the customer.

Free apps can be released too. These could be used as demo app or combined with an ad provider like Google AdMob to create revenue by displaying advertisements. In-App payments for items, consumables or other game related products are supported too.

Promotions

The Play Store provides auto generated charts and lists of apps which are currently trending. Additionally, Google provides high quality apps and games with an “Editor’s Choice” award which causes an app to be included in a separate section of the Play Store.

As with the App Store, it can be crucial for the success of a game to be included in one of the lists of features games. This highly increases the visibility of the game in the otherwise extremely oversaturated Play Store.

Generating promo codes is also supported and allows for custom promotional campaigns.

Google Play Games Services

The Google Play Games Services is a framework of tools and libraries that can be added to a mobile game. While created by Google and thus mainly focused on Android and the Play store, it is also available for iOS, C++ and Unity.

The framework is provided free of charge and consists of many different features which can be integrated into a game and are focused primarily around social gaming. [33]

Social Features

Social features included in the library are for example leaderboards, achievements, player stats, events, game invites and sending gifts to friends.

Multiplayer

The Play Games Services include libraries that enable multiplayer capabilities for games. Real time and turn based multiplayer games modes are supported without the requirement for a special server or connection management. [34]

Other

Additional features include a piracy protection and license check to ensure the user is playing a legitimate copy of the game and the ability to store save games online to allow the user to easily move between devices without losing progress.

One of the newest features added in October 2015 is the ability for users to quickly create Let’s Play videos of their mobile games and share them on social media. [35]

Amazon App Store

The Amazon App Store is an alternative distribution platform for android apps and games. With over 400.000 apps it is the second largest store for Android behind the official Play Store. [36] It comes preinstalled on Amazon’s own Android devices like the Kindle Fire tablet, where it is the only available app store, and can be installed on most other Android devices. As a special future Amazon offers promotional campaigns like "Free App of the Day" which provides users with a different free app every day.

The registration for the Amazon App Store is free and the basic revenue share is similar to other stores

where 30% of the income goes to Amazon. New apps on the Amazon App Store have to undergo a manual review process similar to the one on Apple's App Store.

Concerns

While the Amazon App Store has gained significant market share, it might not be the right distribution platform for a mobile game. The terms of services of Amazon contain certain wordings which caused the International Game Developers Association to call it "...a threat to game developers". [37]

Clauses in the developer agreement include the ability for Amazon to temporarily reduce the price of an app up to 80% of the minimum list price during promotions without asking or notifying the developer. Additionally, developers have to agree to not make an exclusive promotional deal with any other app store. [37]

These terms and conditions might change over time and it is important for a developer to carefully check them before releasing their game on this store.

SlideMe

SlideMe is another alternative app distribution platform for Android. The SlideMe store is preinstalled on devices of over 140 manufacturers and is focused on niche markets where Google Play is not available. [38]

The registration as a developer on SlideMe is free and their business model is also based on the common 70/30% revenue share system.

SlideMe focuses on high quality apps without an oversaturation of the market and without useless apps. Due to this, SlideMe uses a strict review process for new apps.

Being listed on the SlideMe store can help to boost the sales of a new app and especially small developers report better sale than on Google Play. [39]

Android Game Consoles

Android as an open platform is also used in other devices than phones and tablets. There are some Android based game consoles out there with their own stores. While they are not as widespread as smartphones and tablets, their customer base consists primarily of gamers and thus publishing for these systems might be worthwhile.

Razer Forge TV

In 2012 the Android-based gaming console called

Figure 8: Razer Forge TV

"Ouya" was funded on Kickstarter. [40] While it was one of the first of its kind, it did not achieve long term success and was discontinued and sold to Razer in 2015. Razer rebranded it and build a new hardware platform called the "Razer Forge TV" and added a new App Store called the "Cortex Store". [41]

Creating a developer account for this platform is free and all sales use the typical revenue share where Razer receives 30%.

NVidia SHIELD

NVidia SHIELD is a line of Android devices fo-

Figure 9: NVidia SHIELD lineup

cused on gaming. They are developed and manufactured by NVidia and possess a higher that average gaming performance.

Currently there are three SHIELD devices available: A handheld console, a tablet with wireless controller and a TV console and streaming box. [42]

Games for the NVidia SHIELD devices are sold through the Google Play Store and thus the same rules and regulations apply. However, the SHIELD devices also contain a separate app which features and highlights specific games and links to their Play Store pages. The list of games in this application is curated by NVidia and focuses on high quality games, especially those which use the hardware capabilities available on SHIELD devices. To be included in this list an application has to be sent to NVidia where the game will be reviewed.

The process of releasing for SHIELD does not incur any other costs than the registration fee and the revenue share of the Google Play Store.

4 Consoles

Developing for a gaming console is much more difficult than for PC. But the reason is not that the developer has to learn a new programming language or work with difficult systems, it is because of the guidelines provided by the console manufactures which have to be followed

These guidelines are several hundred pages long and contain many information how a game for this specific console has to look like, how the game should react on specific user input and how not. The game developer is not allowed to reinvent the wheel, he must follow the guidelines strictly else the game would not be released on the console. This process can take several months up to half of a year and many games fail to be released on a console. Mostly the developers were not able to implement them into their games or were not able to finance the additional time which was needed to do this.

One easy to understand example for this guidelines is the button layout. On a Nintendo console the A-Button is used to accept an action and the B-Button to decline it. On Microsoft's console, the Xbox, it is similar. The difference is that the buttons are on different positions. Nintendo A-Button is on the right side while Microsoft's is on the bottom side of the layout. That would not be a problem if there was not Sony. Son's guidelines do not just say this button is for action and this is for decline, it differs depending on the region where the game will be released. So the developer has to implement a switch in the code to change the button layout if it is released in Japan or in Europe or America. In detail this means that in Europa and America the gamers accept actions with the X-Button on the PlayStation controller. This works with the same layout like Microsoft's Xbox controller. In Japan actions are accepted with the Circle-Button on the right side of the controller. So it is similar to the Nintendo controller where the "accept"-Button is also on the right side.

These guidelines go quite into detail and the developers have to follow all and everything of them to be able to release a game on a console. But the console manufactures have started to think about independent developers and to support them a little bit.

In the following part it can be read about what it takes to release a game on a specific console and where information and help can be found to reach this goal.

4.1 Nintendo

Nintendo is a very traditional Japanese corporation. They have strictly hierarchical order and work very conservative. But in the last few years they began to open up for more and more developer and after the launch of their online shops they also started to support independent developer to release their games on Nintendo's consoles.

With the Nintendo Wii U they created a possibility for easy entry into development for their console. Every licensed Nintendo developer is allowed to use either the Nintendo Web Framework or Unity for Wii U to develop and publish their games but more on this later.

If a developer wants to develop a game for Nintendo's consoles he has to register on

www.warioworld.com. This is the main developer platform of Nintendo. If he wants to use the aforementioned Nintendo Web Framework or Unity for Wii U, he also has to sign up on wiiu-developers.nintendo.com to get access to the needed software.

Before developers are allowed to release a game on Nintendos online shops they must be rated with an age rating certificate. They will need the ESRB for North American releases and PEGI as well as USK for the European ones. The game itself must not contain many different languages but the electronic manual, which is attended to all releases, must contain English, French, Italian, German, Spanish and Dutch.

The trickiest part is the QA check, which we have already addressed before. Nintendo tests quite fast, but after they find a few bugs in the game or the documentation, they send it back to the developers and they have to start the deploying process all over again. And they are not allowed to forget to update the version number of all their papers documentaries.

30% of the income goes directly to Nintendo. But since the eShop launched on Wii U and 3DS the developer or publisher have the free choice to choose how much the game will cost. Also they are allowed to create temporary price drops and develop a good price strategy. [39]

4.2 Playstation

Sony was the first one of the "big" three companies which started to support independent developer with their program called PSP Mini. That program was built especially for small developer to help them with their first releases and lower the financial risks with offering a monthly revenue. Today still the company supports startups and university to produce games for PlayStation.

Figure 10: Nintendo Logo

Figure 11: Sony Playstation 4 Logo

For European developer there is one specific website, where they have to register to get in touch with the Sony Team which is responsible for Europe, Australia and a few small countries. On http://develop.scee.net/ everything can be found, what it takes to be a Sony PlayStation developer. [39]

If the game will be approved for Sony's online store depends on their scope and the implemented platform features. Sony exclusive game titles must include at least one platform feature. All non-exclusive title must maintain feature and content parity for at least three months.

4.3 Xbox

Figure 12: Microsoft XBox One Logo

At the beginning of Xbox One, Microsoft decided to lock indie developers out from their console. But only a short time after launch, they made a U-turn and support them now too. [39]

Microsoft started ID@Xbox to allow qualified game developer to release their games in Xbox store. ID also allows developer to use the Xbox Live features. These connect all gamers together over all Microsoft platforms. Xbox Live is also supported by Windows Store applications. So games which are released on Xbox and Windows Store can be connected and are able to exchange data.

With ID@Xbox it is possible for the developers to get all the benefits out of the console, including the full power of the console, cloud services, Kinect and Xbox Live toolset such as Xbox SmartGlass, multiplayer, Achievements, Gamerscore and more.

Register to the program is easy, just fill out a form on http://www.xbox.com/en-us/Developers/id. Everyone can register, but developers who can show some previous work on PC or other consoles have a far higher chance to be accepted. But once in there is much benefit the developers can gain from Microsoft.

Part III
GPU and CPU profiling of C++ and C# programs

Kernjak Martin and Kucher Josua

Abstract — Profiling is a integral part of developing any performance critical programs. Various IDEs, Editors and other third party programs offer profilers. This paper introduces and evaluates the profilers of some of the industries biggest IDEs, Engines and Hardware Manufacturers.

Keywords — profiling, cup-profiling, gpu-profiling, gprof, unity engine, unreal engine

1 Introduction

Profiling is an essential part of performance optimization in the development of software applications. In executable Programs, increases in speed of execution are very often desirable and profiling helps to achieve this especially if the implementation has routines that take up a large portion of the execution time. Finding these can help the programmer optimize these portions of code to achieve better performance and decrease the execution time significantly.
In most cases the software that needs to be optimized consists of a lot of small routines and a variety of abstraction levels. In some cases programs have been written by single individuals who comprehend all these abstractions but more often then not software is a group effort that evolved over time, this leads to shifting demands on abstraction levels. With the addition of external libraries which introduce additional abstraction levels into the program which are usually not examined further.[43] [44] describe three kinds of program analysis tools, the first using program counters, the second kind are tracing data generation addresses and instructions and lastly simulators. The purpose of this work however is to give an introduction to various profilers and to show their functionalities so the inexperienced can use it as a guide to set up profiling in their project.

2 Profiling

In essence profiling can be achieved in two different ways, execution counts and execution time, with both of them having respective benefits and problems.
Execution counts contain the exact amount of times a statement or routine is called. This can be useful when determining if the implemented algorithm is functioning correctly. In depth examination of such counters which are usually presented tabularily besides the source code, can help find error and overcomplexity in the program. In some ways this works similar to debugging as it helps to identify dead code. Execution counts ave a major downside though as they do not accurately represent the amount of time it took to execute a certain piece of code. This holds especially true if their are multiple abstraction levels.

Listing 1: Call Count example

```
/*line Number---------------- count*/
0001        if(x=foo)            0055
0002        functionX(foo)       0032
0003        if(x=baa) =          0055
0004        functionY(baa,foo)   0017
```

Fig. 7.2 is an example of how call counts may be displayed inside code.

To determine execution times, two ways can be implemented. The first Method is sampling times-

tamps when entering and exiting a routine and calculating the time difference to gain the elapsed time. This become inherently complicated if the program uses time sharing due to the time slicing performed by the program. [43] describe another method as well which uses the sampling of the program counter at an arbitrary interval and use the distribution of these samples to determine the execution time of a program. Apparently this method works better for time sharing systems where the sampling interval can correspond with the time slicing of the program. Sampling also is independent of the operating system and only needs to be able to use interrupts. It is crucial that the sample intervals are uniform and are chosen sensibly. To high sample rates lead to problems in the profiled programs and to low sample rates are not accurately representing the time distribution. The difference between method one and two is their output, Method one can provide exact timings and method two is a statistical approximation.[43]

3 Evaluation and Instruction

In the following sections various profilers will be discussed and introduced in a manner that will facilitate the understanding both in setup and usage of each profiler respectively. Furthermore multiple criteria (these are discussed at a later point) will be evaluated for each profiler to provide a fast way to asses usefulness for a specific project/use case. In this work only profilers used for C++ and C# as well as game engineering related products, such as game engines are discussed, specifically the following:

- Visual Studio Profiler
- Unity Profiler
- Unreal Profiler
- Nvidia Nsight
- AMD CodeXL

Criteria for the evaluation include the following and will be marked as ☑ if met by the respective profiler:

1. CPU Sampling: The percentage of time spent in the function over the whole runtime of application.
    - Call Counts: Number of Calls to a function over the whole program runtime.
    - Sampling Percentages: Sampling percentage of inclusive and exclusive percentages.
2. Time Measurement: The actual time in seconds spent inside a function
    - Function Level Measurement: Time Measurements of individual function in seconds.
    - Line Level Measurement: Time Measurements of individual lines in seconds.
3. Memory Usage: Memory allocated and used by the application
    3.1. Memory Consumption: Combined stack and heap memory usage
        - Heap Allocation: Memory allocated on the heap.
        - Stack Allocation: Memory allocated on the stack.
    3.2. Memory Allocation/Deallocation Time: Timestamps for memory allocation and deallocation.
4. Multithread support: Profiling over multiple threads

4 Visual Studio Profiler

Visual Studio offers its own profiler. This profiler has been available in the professional version for some time but is now also available in the community edition since Visual Studio 2015. It offers a simple real time statistic whenever the debugger is used. This shows the run-time, the used memory and the CPU utilization. The more in depth profiler can be used for multiple different profiling

methods. These can then be analyzed in the performance explorer.[45]

4.1 Evaluation

Criteria	Availability
CPU Sampling	☑
Call Counts	☑
Sampling Percentages	☑
Time Measurement	☑
Function Level Measurement	☑
Line Level Measurement	☑
Memory Usage	☑
Memory Consumption	☑
Heap Allocation	☒
Stack Allocation	☒
Memory Allocation/Deallocation Time	☒
Multithread support	☑

Table 1: Results Visual Studio Profiler

The Visual Studio profiler supports almost every evaluated feature except for the exact Heap and Stack allocations. It only offers the current memory usage for a specific point in time, but does not show where in the code these allocations occurred.

4.2 Profiling Methods

The Performance Explorer offers 4 different profiling methods.

1. CPU Sampling
2. Instrumentation
3. .Net memory allocation
4. Resource Contention Data (concurrency)

CPU sampling checks in which function the process currently resides in at regular intervals. The performance explorer can then be used to show the absolute and relative inclusive and exclusive sample count. This is a low overhead profiling method and is useful for an initial analysis.

Instrumentation is a more in depth profiling method that collects detailed timing data. As such the overhead is also considerably larger and can result in significant slow downs. Analyzing the profiling report is also much more time intensive.

The memory allocation profiling is only available in .Net and shows the functions which led to memory allocations and how many bytes were allocated.

Resource contention Data detects when one thread has to wait for another. More general data about how the threads are handled by the application is also collected. This lists the number of contentions and also how long they lasted.

These are the more in depth profiling methods but visual studio also offers CPU, GPU and memory tracking.

[46] [47]

4.3 Starting a Profiling Session

The solution should be loaded to start the profiling session. Go in the top menu to Debug->Start Diagnostics Tools Without Debugging. A new window opens where the target and the profiling tool can be selected. The target defaults to the currently active solution but any project within the solution can also be selected among other targets. Select ”Performance Wizard” and click start. Another window will open where the profiling method can be selected. The concrete project can also be selected here. Afterwards the program will be recompiled and started. The profiling session ends when the main function finishes. Finally a report will be generated from the profiling data.

The CPU, GPU and memory tracking tools work in a similar way but the user has to take explicit snapshots for the memory profiling.

4.4 Using the Performance Explorer

The performance explorer is automatically opened after a profiling session has finished. Every Profiling session is listed on the left and the last report is opened automatically. These reports can also be saved and loaded. The report section can be selected at the top under ”Current View”.

The actual views are dependant upon the profiling method.

4.5 CPU Sampling

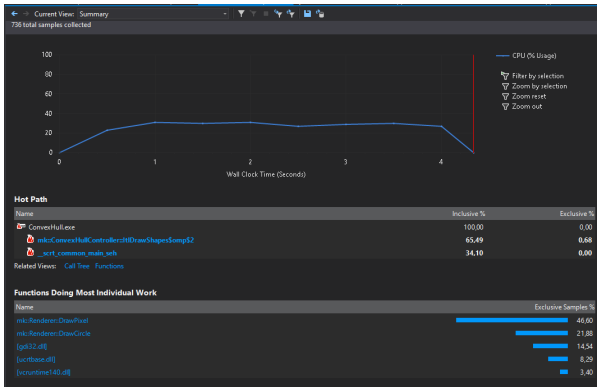

Figure 13: Visual Studio Profiler CPU Sampling Report Summary

This is the summary page. The graph at the top shows the CPU usage during the execution time. Below is the "Hot Path" which shows the branch of the call tree with the highest inclusive samples. Below that are the functions with the highest amount of exclusive samples. Meaning that these function where covered by the most samples while disregarding samples in their child functions. Both of these offer a good starting point for optimization. The function can also be selected for a per function and line breakdown.

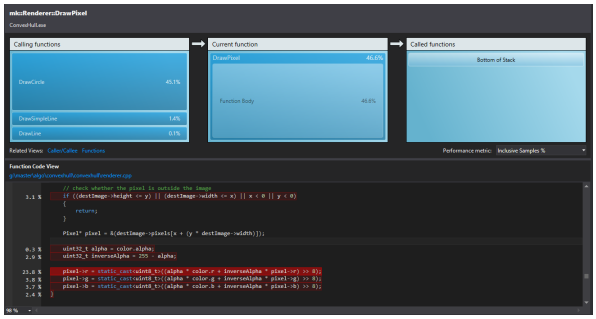

Figure 14: Visual Studio Profiler CPU Sampling Function Breakdown

The function breakdown shows the sample percentages for the caller functions and any function called by the current one. The actual lines with the highest sample count are also highlighted below.

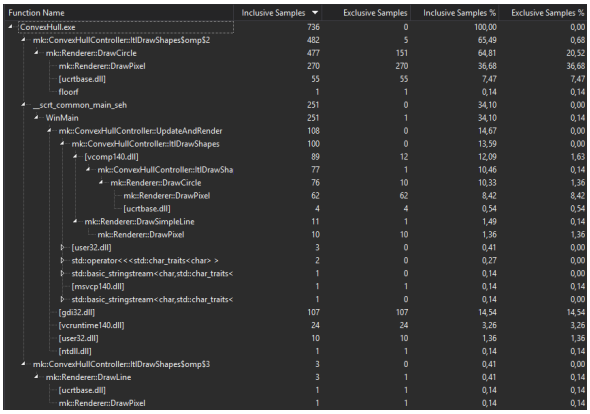

Figure 15: Visual Studio Profiler CPU Sampling Call Tree

There are also other views such as the aforementioned call tree.

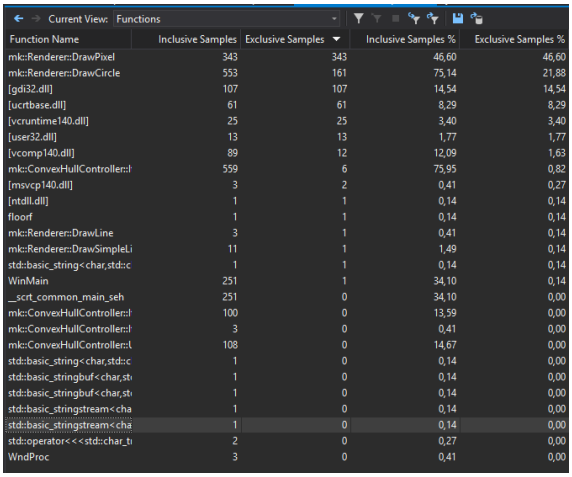

Figure 16: Visual Studio Profiler CPU Sampling Functions

And a complete function breakdown.

4.6 Instrumentation

The instrumentation report also includes real time measurements. The report structure is the same as for the CPU sampling but the values are based on the actual execution time instead of the more in-precise samples.

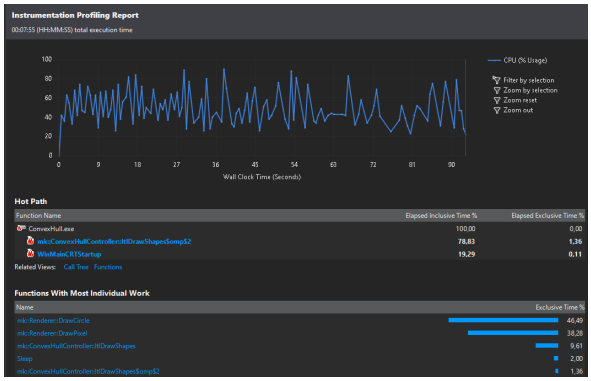

Figure 17: Visual Studio Profiler CPU Instrumentation Summary

4.7 Concurrency

The concurrency report follows the same structure again but is based on the thread contentions.

Figure 18: Visual Studio Profiler CPU Concurrency Summary

4.8 CPU/GPU Capture

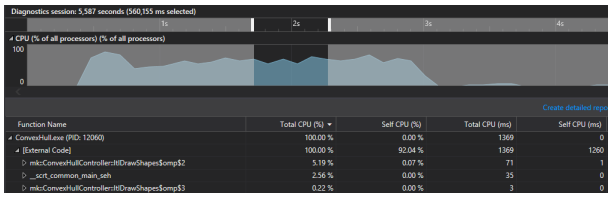

Figure 19: Visual Studio Profiler CPU Capture

The simple CPU/GPU capture tools don not offer a detailed report but a simple CPU/GPU Usage and function breakdown. Certain section of the execution time can also be selected for a function level breakdown of the selected time frame.

4.9 Memory Capture

The memory capture tools works a bit different because the snapshots have to be taken by the user during the profiling session. A "Take Snapshot" button will appear in the performance explorer window during the profiling session.

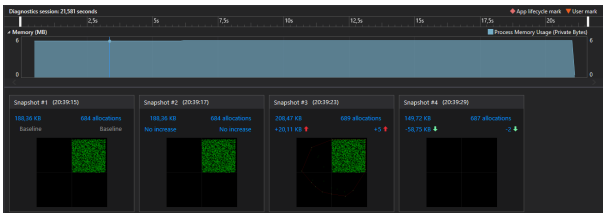

Figure 20: Visual Studio Profiler Memory Snapshots

The general memory usage is shown at the top with more detailed information for each snapshot.

5 Unity Profiler

The Unity Profiler is the profiler shipped with the Unity Engine, as of the current version (Unity v.5.2.3). It is found inside the Unity Editor tool under Window->Profiler. It can be run while testing the application in Unity and records performance Data later output as a timeline for analysis. It is frame accurate and can be used to see where spikes in eg. frametime occur during runtime and according to the Unity Documentation should be used to compare performance after code changes while providing minimal impact on performance. Due to Unitys multiplatform compatibility some platforms (Android,iOS,Webplayer) require special steps in profiling which will not be discusses in this paper but can be found in the Unity documentation.

5.1 Evaluation

In accordance with the evaluation criteria provided in Section 3 the unity profiler was analysed. This analysis of the profilers capabilities yielded the following results.

*Time measurements are not provided for Functions of the actual code, but for calls to Unity functions that occur inside the code. Conclusions to the

Criteria	Availability
CPU Sampling	☑
Call Counts	☑
Sampling Percentages	☑
Time Measurement	☑
Function Level Measurement*	☒
Line Level Measurement	☒
Memory Usage**	☑
Memory Consumption	☑
Heap Allocation	☒
Stack Allocation	☒
Memory Allocation/ Deallocation Time	☑
Multithread support	☑

Table 2: Results Unity Profiler

actual code can be drawn from these.
**Although there is not Heap and or Stack Allocation, Allocation for different relevant Objects is kept track of eg.Textures, Meshes, Materials.

The Unity Profiler is highly specialized to the fit the engines needs. In contrast to profilers for raw code it provides a lot more information that are relevant in the context of a game. All information is presented in a timeline and on a frame by frame basis. It contains information about CPU and GPU usage as audio sources, physics calls and network traffic. It does not provide a direct link to the program code, which requires the programmer to have a certain amount of insight into unity functions, since only those are represented in the profiler. Function calls implemented in will also show but most information needs to be deducted from the actual unity function calls.

5.2 Using the Unity Profiler

To start profiling with the Unity Profiler you need to open the project you want to profile and select the entry point for your game (eg. Main Menu) and open the profile which can be found inside the "Window" menu under "Profiler". The profiler window itself is composed of several sub-windows each representing a different timeline (eg. CPU Usage, GPU Usage, Memory Usage etc.). In the top of the profiling window are several options such as the record button, which enables you to record data while playing the game in the editor, the deep profiler, which allows you to profile all mono calls so you can debug your scripts and a Clear button. Furthermore it contains information about how much frames where recorded in total and which frame you are currently profiling, since all profiling is done in a frame by frame basis.

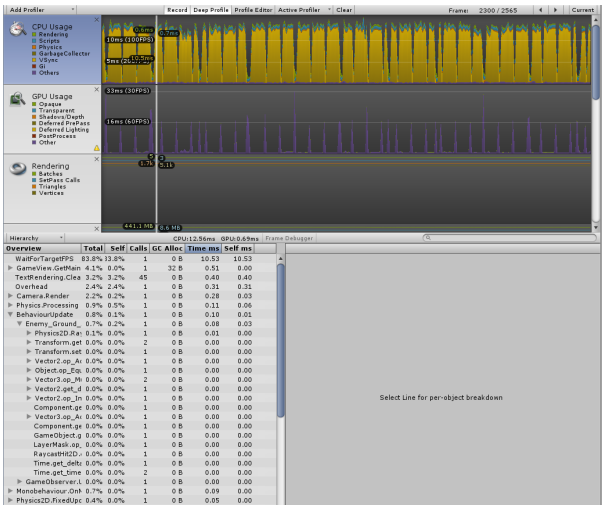

Figure 21: Unity Profiler Window

The mouse cursor can be used to select the current frame inside the timeline and additional information is shown based on the context of the subprofiler. Additionally the bottom of the window shows more information about the currently selected subprofiler shown in blue in Fig. 21.

CPU Usage:

In case of the CPU sub-profiler it shows either the Hierarchy of the calls as in Fig. 21 or the timeline. Either way the information is ordered by usage, highest usage first. Using this information the most impacting unity behaviours can be retrieved and further analysed. As seen in Fig. 21 the hierarchy is split up in parent and child calls. Parent Calls consist of high level Unity calls as for example the Behaviour.Update function. Inside these parents all children are shown and inside the children the individual Unity calls can be analysed such as Vector additions, transformation calls and physics calls.

Memory:

The memory sub-profiler, uses 2 different modes of displaying, the simple mode which shows high level memory usage and the detailed view, in context of this work the more interesting one. To get the detailed view of a frame the "Take Sample" button needs to be pressed. The result is a tabular representation of used memory, which is split up in different sections, with the Scene Memory part being the most relevant one, showing the memory usage of all objects inside the scene including their reference counts. This information can be used to profile memory restricted applications.

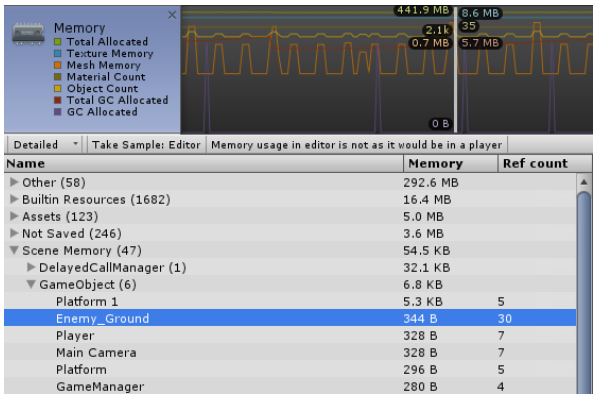

Figure 22: Unity Profiler Memory Area

Rendering:

In the Rendering sub-profiler shows information such as Batches, Triangles, Vertices and SetPassCalls for the current frame. These are very similar to the rendering statistics window that can be enabled in the Editor during a running the game.

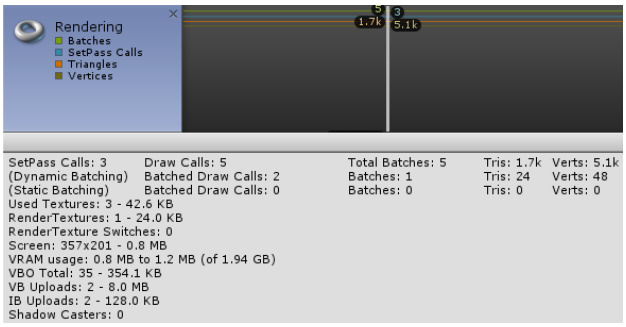

Figure 23: Unity Profiler Rendering Area

GPU:

The GPU sub-profiler works like the CPU profiler and shows the number of DrawCalls and time spent inside each DrawCall. These are shown in a hierarchical way like in the CPU view. If the GPU profiler does not work for the current project it might be due to lack of the appropriate driver.

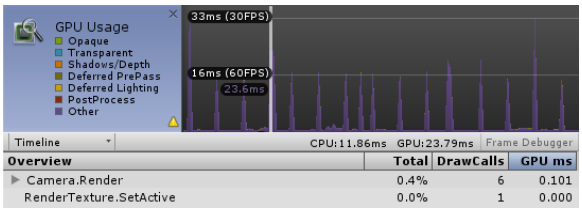

Figure 24: Unity Profiler GPU Area

Physics:

Inside the Physics sub-profiler information about 2D and 3D physics can be found and is split up into five different kinds of Information.
Active Rigidbodies: These are currently moving physics objects inside of the inspected Frame
Sleeping Rigidbodies: Show the physics objects that are at rest and are not updated in the current frame.
Number of Contacts: Number of contacts of all colliders in the scene.
Static Colliders: Number of colliders not attached to rigidbodies.
Active Colliders: Number of colliders not attached

to rigidbodies.

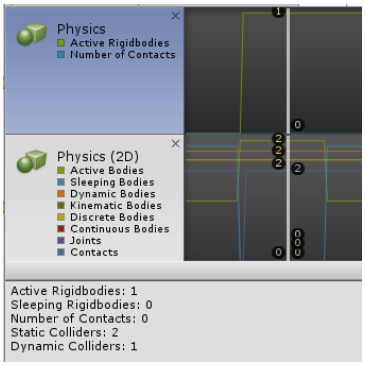

Figure 25: Unity Profiler Physics Area

6 Unreal Engine Profiler

The Unreal engine 4.9 offers its own profiler for CPU as well as GPU profiling. The profiler can be run in real time during the game for an immediate performance representation. Performance reports can also be saved and loaded later for a more in depth analysis. Unreal also offers a simple fps breakdown of the workload. This breaks the performance down into multiple categories.

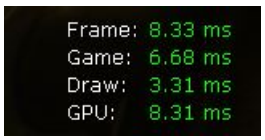

Figure 26: Unreal Stat Unit https://docs.unrealengine.com /latest/images/Engine/Performance/stat_unit.jpg

First is the total frame time, Game represents the CPU game thread, Draw is the CPU render thread and GPU is the GPU time. This helps narrow down whether the CPU or the GPU should be profiled. The GPU is the bottleneck in the provided example. These frame rates can also be continuously captured over a period of time and combined into a graph. [48]

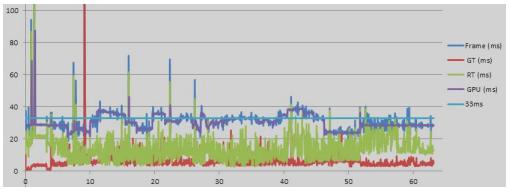

Figure 27: Unreal FPS Chart https://docs.unrealengine.com /latest/images/Engine/Performance/fpschart.jpg

7 Evaluation

The Unreal profiler supports many of the evaluated features.

Criteria	Availability
CPU Sampling	☑
Call Counts	☑
Sampling Percentages	☑
Time Measurement	☑
Function Level Measurement	☑
Line Level Measurement	☒
Memory Usage	☑
Memory Consumption	☑
Heap Allocation	☑
Stack Allocation	☒
Memory Allocation/Deallocation Time	☒
Multithread support	☑

Table 3: Results Unreal engine Profiler

The Unreal profiler provides most of the evaluated features.

7.1 CPU Profiling

The Unreal profiler is built into the editor of the Unreal engine. The profiler can be opened be selecting the ”Window” tab in the top menu of the editor then ”Developer Tools” and ”Session Front end”. This opens the ”Session Front end” window where the profiler is located.

This is the complete ”Session Front end” window. It is separated into multiple sub-windows.

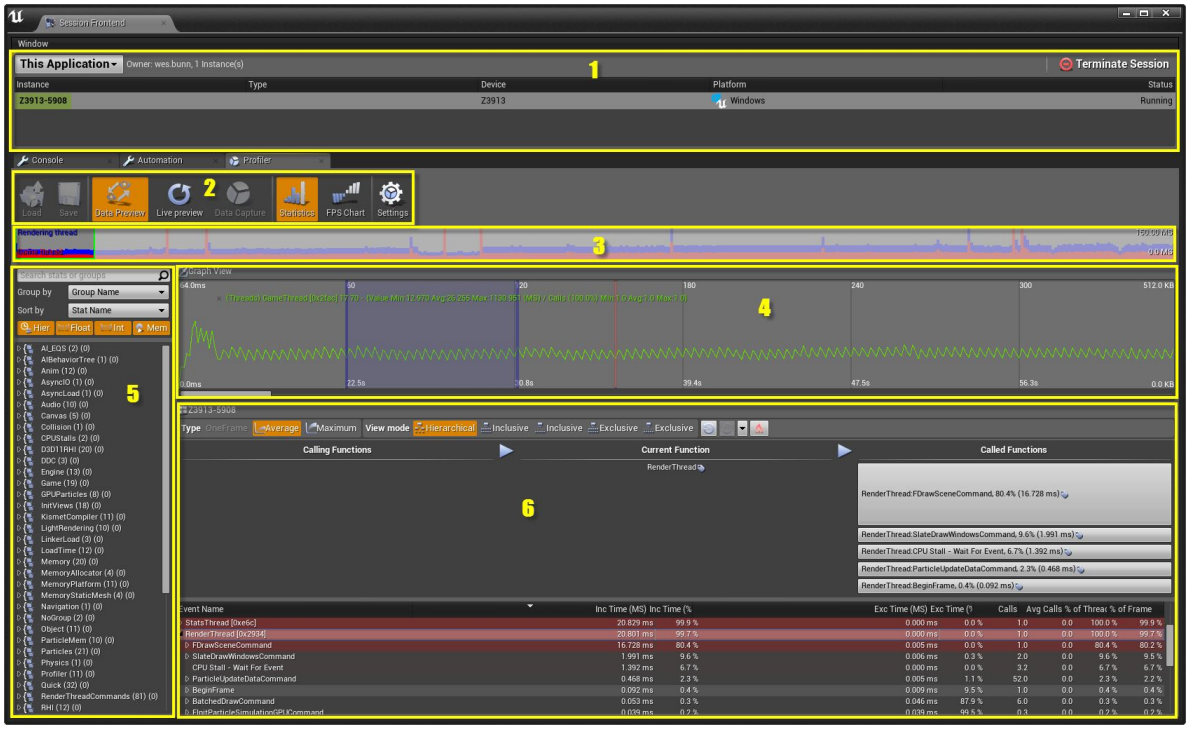

Figure 28: Unreal Session Frontend menu
https://docs.unrealengine.com/latest/images/Engine/Performance/Profiler/ProfilierUI.jpg

1 Session Window

The profiled session can be selected here and various session information is also displayed. Below is various environment information such as the platform (Windows, PS4, ...).

2 Main Toolbar

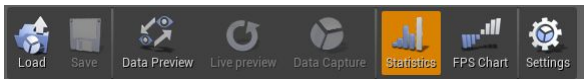

Figure 29: Unreal Main Toolbar
https://docs.unrealengine.com/latest/images/Engine/Performance

The main toolbar offers various profiling options. The profiling data can be loaded and saved. But the save option is currently not functional. "Data Preview" starts a continuous capture of the profiling data. "Live Preview" sets the currently displayed data automatically to the last captured frame. "Capture Data" stores the captured profiling data which can be saved at the end. The last option is the "Fps Chart". Note that this FPS chart is different from the one mentioned in at the beginning of the Unreal profiler. This creates a histogram of the FPS but does not include a breakdown of the FPS per pipeline stage. [49]

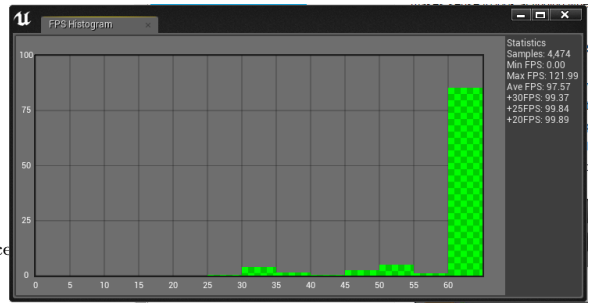

Figure 30: Unreal FPS Histogram

This shows how many samples fall in a certain FPS bucket. Additional global information is also displayed on the right, such as the maximum, minimum as well as the sample percentages above certain FPS thresholds.

3 Full Data Graph

Below that is the full data graph. It provides a graphical overview of the entire profiling data. The bar chart shows the total time the render and game thread took. The highlighted section is used for the more detailed data graph below and can be moved in the full data graph. [49]

4 Data Graph

Bellow are the main data evaluation tools.

5 Filter and Preset Window

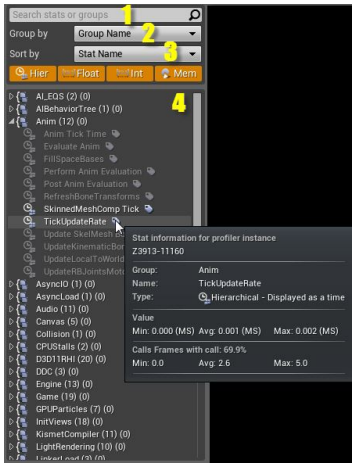

Figure 31: Unreal Filter and Presets Window

To the left is the filter and presets window. The information that is to be displayed in the data graph is selected here. There is hierarchical structure view for the stats. These are sorted into integer, float and memory stats. The stats can be filtered by their group, name and type (1, 2, 3). Every stat that is hovered in the stats window also displays the stat information such as the minimum, maximum and average values. The stats can be selected by clicking or dragging them to the data graph (4). Every selected stat is then tracked in the data graph. [49]

6 Event Graph Window

The event graph window is separated into two sub windows. The upper part is the data graph.

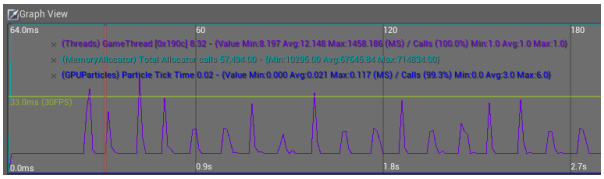

Figure 32: Unreal Data Graph

Every selected stat has its information displayed as a text overlay. The time segment is the highlighted section on the full data graph window. The stats displayed for each stat are based on the entire profiling data and are not only based on the currently highlighted selection. The displayed information contains the current, minimum, maximum and average values. Hierarchical stats also show additional call information. They also show the percentage of frames with a call to the stat as well as the minimum, maximum and average calls to the stat. These values are again based on the entire profiling data. [49]

The lower part is the main event graph and the toolbar.

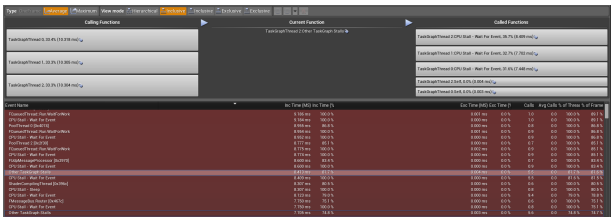

Figure 33: Unreal Event Graph

The toolbar has multiple options to decide which information should be displayed. The information is based on the frames that have been selected in the data graph. The first section in the toolbar decides what type of information should be displayed

1. One Frame: Can be enabled if only one frame has been selected in the data graph. This only shows the information from a single frame.
2. Average: This can be enabled if multiple frames have been selected. This shows a per frame average graph.

3. Maximum: This can be enabled if multiple frames have been selected. This shows a per frame maximum graph.

[49]

The second section can be used to choose how the events are displayed and sorted in the main event graph.

1. Hierachical: This displays the events in a hierarchical tree view.
2. Inclusive: This displays the events as flat list sorted by the inclusive time.
3. Inclusive (different icon): This displays the events as flat list coalesced by the event name and sorted by the inclusive time.
4. Exclusive: This displays the events as flat list sorted by the exclusive time.
5. Exclusive (different icon): This displays the events as flat list coalesced by the event name and sorted by the exclusive time.

[49]

And the last section contains step back and step forward buttons and an option to display the hot path for the selected events.

The next part is are the function details. This shows the currently selected event and the functions or events that called the selected event. These are displayed as buttons and are scaled in relation to their performance impact. The windows is separated into three parts for the calling functions, the current function and the called functions respectively. The inclusive time in milliseconds and as a percentage are also shown for every function. [49]

The last part is the main event graph. This is a table with every event. The following information is displayed for every event. This information can also be shown in a separate window by hovering the event.

1. Event Name
2. Inclusive Time in milliseconds
3. Inclusive time as a percentage
4. Exclusive Time in milliseconds
5. Exclusive time as a percentage
6. The number of times the event has been called
7. The average number of times the event has been called
8. The inclusive time epercentage per thread.
9. The inclusive time epercentage per frame.

[49]

7.2 GPU Profiling

The GPU profiling is a bit more limited than the CPU profiler. The GPU profiler can be accessed by opening the unreal console and entering ”profilegpu”. This opens a new window with the GPU data that has been captured up to this point. The data itself is based on GPU timestamps. It shows a timing breakdown of how long individual passes took. The timing breakdown is hierarchical and each pass can be broken down further. Sometimes down the draw calls. [50]

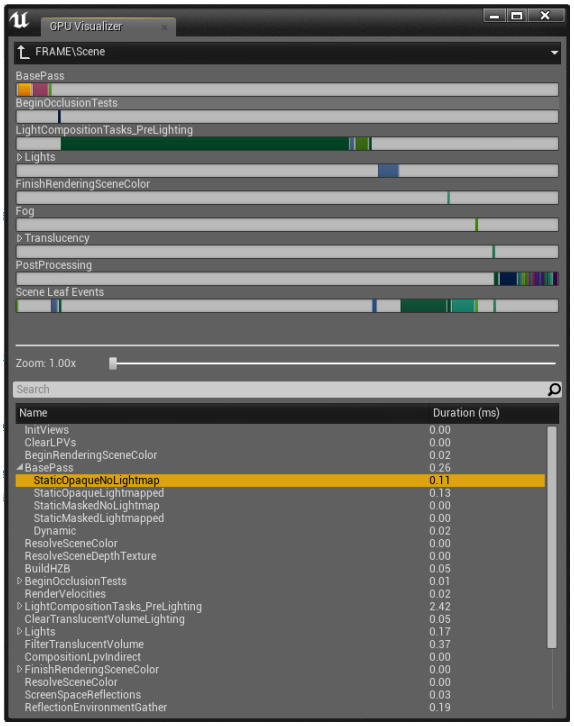

Figure 34: Unreal GPU Visualizer

The profiler shows a graphical breakdown of the passes at the top with a multi level bar chart. Be-

low is the hierarchical breakdown of the passes and their respective times.

8 Nvidia Nsight

Nvidia Nsight is a debugger designed to debug programs running on Nvidia GPUs. In this work, we will focus on the NVIDIA Nsight Visual Studio 5.0 implementation, a additional software package for the Visual Studio IDE, providing all Nsight functionality inside Visual Studio. To use them an additional installation of the Nvidia Display Driver which supports Nsight is necessary. [51] lists the following drivers as compatible: Release 355.85 and Release 355 or newer.
Nvidia Nsight generally consists of three different functionalities. The CUDA Debugger, allows to debug applications using Compute Unified Device Architecture(CUDA), according to [51] it is able to inspect memory usage and perform memory checks in addition to normal debugging features such as breakpoints. Though the debugger only works on programs created with either the CUDA Runtime API or the CUDA Driver API. The normal Graphics Debugger, can be utilised to debug a frame by each draw call, allowing you to inspect the shaders and pipeline. Furthermore this allows the profiling of graphics code together with the Analysis and Profiling tools. These show the workload of your system and shows API Calls (eg. OpenGl, CUDA) as well as memory management, draw calls and Events on the GPU and CPU. Nsight can be configured in two different ways a pure client side version running the IDE and Nsight on the same machine, and a target and Host setup using an additional machine to debug code. This work will focus on the pure client side installation and addition information for the Target Host setup can be found on the Nvidia Nsight page.[51]

8.1 Local Debugging

To start using the local debugging feature of Nsight, the user first needs to check if the debugging machine fulfills all the requirements needed, including a supported Graphics card (the list can be found in the Nsight User Guide). After that the Nsight Monitor needs to be started, which is found under:

Criteria	Availability
CPU Sampling	☑
Call Counts	☒
Sampling Percentages	☒
Time Measurement	☒
Function Level Measurement	☒
Line Level Measurement	☒
Memory Usage	☑
Memory Consumption	☑
Heap Allocation	☑
Stack Allocation	☒
Memory Allocation/Deallocation Time	☒
Multithread support	☑

Table 4: Results Nsight Profiler

\ProgramData\Microsoft\Windows\Start Menu\Programs\NVIDIA Corporation\Nsight Visual Studio Edition 5.0

After starting Nsight it will appear in the taskbar and can be opened with a double click on the icon. This needs to be started in order to use Nsight, and will automatically be started if Nsight is used inside the Visual Studio IDE.

The Frame Debugger

Using the Graphics Debugger enables the user to utilise the Frame debugger to debug frames in realtime. Rendering calls can be examined as well as the the pipeline state, it provides an visualisation to help debug as well. Captures Frames can be saved and analysed offline as well as profiled. It additionally provides a Heads up Display which overlays over the debugged application and can be customized with additional graphs. It also enables four additional modes:

1. Depth Complexity View: This shows how much geometry is overdrawn every drawcall.
2. 2X2 textures: Provides insight into the application in term of it being texture bound

3. Null Scissor Rectangle: Provides insight into the application in term of it being pixel bound
4. Minimum Grometry: Shows whether the application is CPU or GPU bound by drawing only the first triangle of each draw call. Rising framerates indicate a GPU bound application, while no change indicates a CPU bound application.

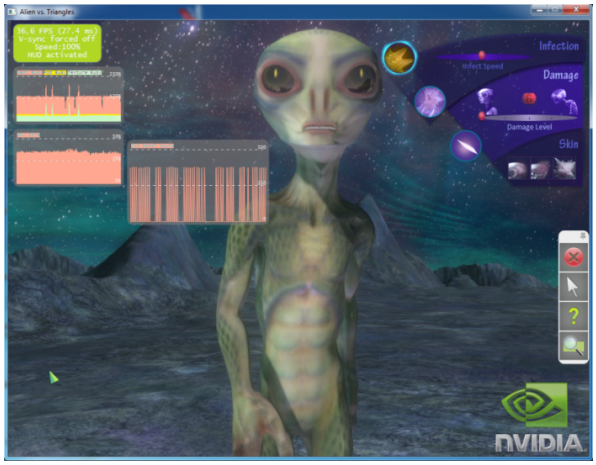

Figure 35: Nsight Frame Debugger HUD
http://docs.nvidia.com/nsight-visual-studio-edition/5.0/Content/Images/HUD.Main.RunMode_700x543.png

You can enter the Debugger directly form the HUD by either pressing CTRL+Z or in the Visual Studio Menu under Nsight, by pressing the Pause and Capture Frame button. Doing this enables you to use a set of options, like rendering the Depth buffer or the stencil buffer instead of the image or displaying the Scene in Wireframe mode.
When Debugging a Frame there are various tools available to the user. One of them are Performance Markers which can be set in four different ways, via the NVTX tool extension, OpenGL command such as KHR_debug group, glPushDebugGroup and glPopDebugGroup, Direct3D 11.1 Commands like ID3DUserDefinedAnnotation and the Direct3D 9 commands D3DPERF_BeginEvent and D3DPERF_EndEvent. These can be used to mark a section of events for time measurement. Time spent in Sections between the markers will be measured and can be analyzed in either the Scrubber or the HUD. The Sections will be highlighted in the defined marker color in graphical representations respectively.

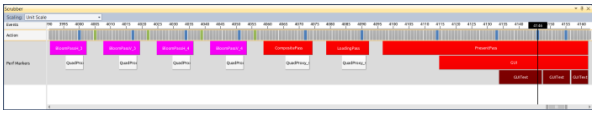

Figure 36: Nsight Scrubber
http://docs.nvidia.com/nsight-visual-studio-edition/5.0/Content/Images/Graphics_Perf_Markers.001_700x125.png

Events can also be inspected individually in the Events Page of the Frame Debugger, nested events will also be displayed in dropdown menus. The Event lost shows all API Calls and highlights performance marked Segments in their respective color. An additional Page of information is the API Statistics Page which provides information on the API Calls as well as the amount of times they have been called. To open this windows the user needs to access the Nsight menu in visual studio, select windows and then the API Statistics window.

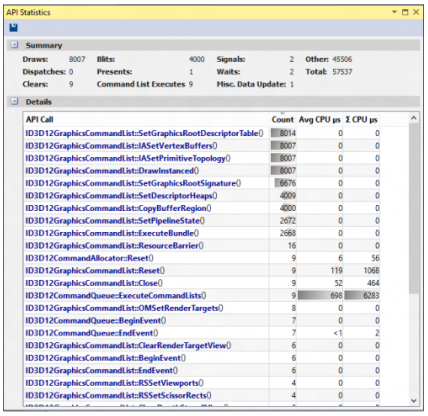

Figure 37: Nsight API Statistics
http://docs.nvidia.com/nsight-visual-studio-edition/5.0/Content/Images/graphics_d3d12_api_statistics.002_500x485.png

Additional information about the current Frame includes Heap information, which shows all created heaps with their respective resources and resource Types. Some additional features are Graphics API specific and can be read up on in the Nsight User Guide on Nvidias page.

Analysis Tools

The analysis Tools allows the user to run the program and attach a trace to it which at then end of recording or at the end of the programs execution creates a detailed report showing all activity during

the program including events from other programs and a lot more depending on the selected settings.

To start using the Analysis Tools the user needs to click on the Nsight menu in Visual Studio 2015 and select the Start Performance Analysis option. Which opens the Activity Document. Inside this Form the user can select what and how he wants to profile his application.
First part is the Application Settings which contains information like the debug mode (local or remote) and where the application an its working directory are located.
The Triggers and Actions tab governs when the information gathering is started and stopped, this can be set to automatic(Program start, and End) or to manual respectively.
The Activity Type sets the kind of analysis that gets performed. The most important here are Trace Application, which traces data from a single target application and Trace Process Tree, which also includes all child processes in the trace.
The Trace and Profiler Settings contain the data you want to trace and is split in System,Tools Extension, CUDA, OpenCL, DirectX and OpenGl. Each of them also contain subtraces which can be switched on and off individually but for the sake of this work the most important are the following:

1. System: Traces CPU usage
2. Tools Extension: In case Performance Markers ares used these are also taken into consideration during the tracing process
3. DirectX: Provides an API Trace as well as GPU information
4. OpenGL: Provides an API Trace as well as GPU information

Advanced Options typically do not need and should not to be adjusted.
The last Section is the Launch an capture control which is split into three parts. The connection status, which shows whether Nsight is connected to a Nvidia device, the Application control which lets the user start and terminate the application and the capture control, that is used to start and stop the capture process.

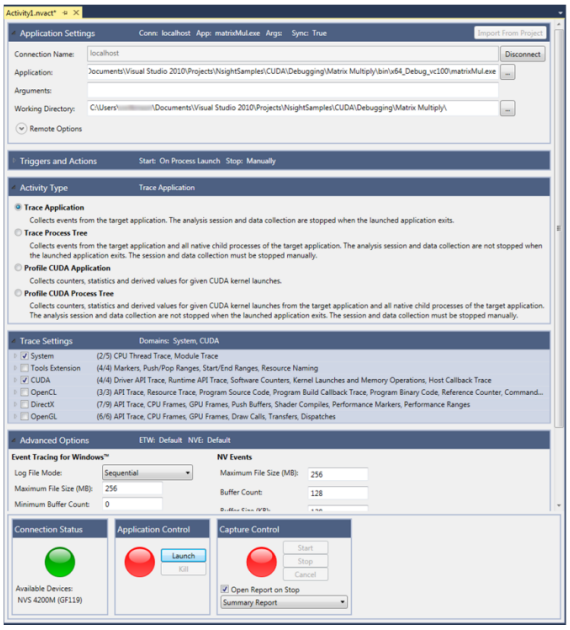

Figure 38: Nsight Activity Document

9 Analyze the collected Data. Step by Step

1. Start the Nvidia Nsight Monitor
2. Start Visual Studio
3. Select the project you want to analyze and create a build of the project
4. Select ”Start Performance Analysis” form the Nsight menu
5. Select the data you want to trace see section 7.1.2
6. Click the Launch button to start the application
7. If manual capture is selected click the start button under the capture section
8. Click the Kill Button to stop the application and the Capture

These steps lead to the automatic opening of the NVReport.

Fig. 39 shows the ”Summary Report”. The top left corner shows a dropdown menu which can be

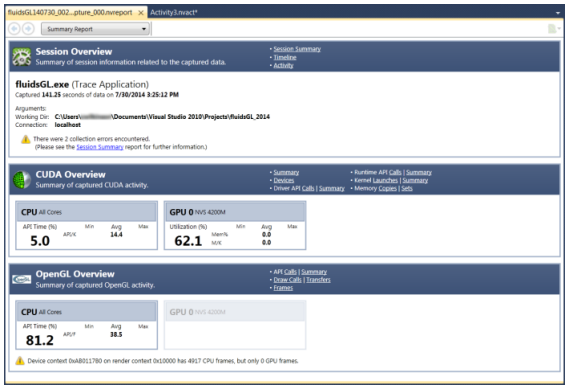

Figure 39: Nsight Summary Report http://docs.nvidia.com/nsight-visual-studio-edition/5.0/Content/Images/Analysis_SummaryReport.003_700x474.png

opened to see each segment of the report that corresponds with your selected trace targets. Basic information is always displayed under the "Summary Report" tab.

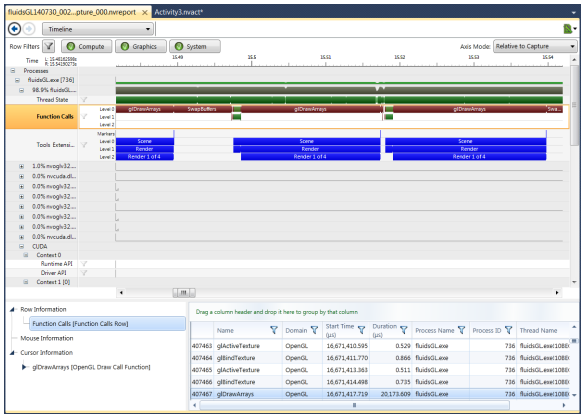

Figure 40: Nsight Timeline http://docs.nvidia.com/nsight-visual-studio-edition/5.0/Content/Images/Analysis_TimelinePage.003.png

Fig. 40 shows the timeline. This Tab will also always be present in every report and enables in depth analysis of the System during the captured timeframe. These include API calls, CPU usage by other programs during the runtime as well as GPU usage. Using the Timeframe information such as how well the GPU is fed can be deduced to further optimize the Application. For Example if the GPU is idle a lot of the time maybe it is due to to much communication between GPU and CPU and calls need to batched.

10 Additional Features

10.1 Frame Serializer

During the use of the Frame Debugger, a frame can be captured and serialized.

Figure 41: Nsight Frame Debugger Overlay save Button

The Button displayed in Fig. 41 will be prominent in the top right of the screen during debugging and enables the serialization oft the current frame. This data can then be sent to other users of Nsight to analyzed there. This enables the possibility to share bugs with other members of the development team as well as Nvidia for optimizing their drivers.

11 AMD CodeXL

AMD offers its own profiler and performance analyser CodeXL. This a a new program that succeeds the previous AMD CodeAnalyst as AMDs profiling and debugging tool. This profiler was specially built to profile AMDs product line of CPUs, GPUs and APUs. CodeXL is available as a standalone application for Windows and Linus as well as a Visual Studio extension. A more detailed tutorial for CodeXL is omitted a this point as the authors do not have access to the AMD products necessary to properly familiarize themselves with the program. [52]

12 Evaluation

The CodeXL CPU profiler relies heavily on CPU Sampling and omits memory tracking and time measurements.

Criteria	Availability
CPU Sampling	☑
Call Counts	☒
Sampling Percentages	☑
Time Measurement	☒
Function Level Measurement	☑
Line Level Measurement	☒
Memory Usage	☒
Memory Consumption	☒
Heap Allocation	☒
Stack Allocation	☒
Memory Allocation/Deallocation Time	☒
Multithread support	☑

Table 5: Results AMD CodeXL

13 CPU Profiling

The CPU profiler allows for time or instruction based sampling. The sample counts and relative sample measurements are then collected and displayed on a per function level. This allows for relative performance analysis. The sampling is realised through hardware level performance counters and should as such have a minimal performance impact and offer accurate readings. [53]

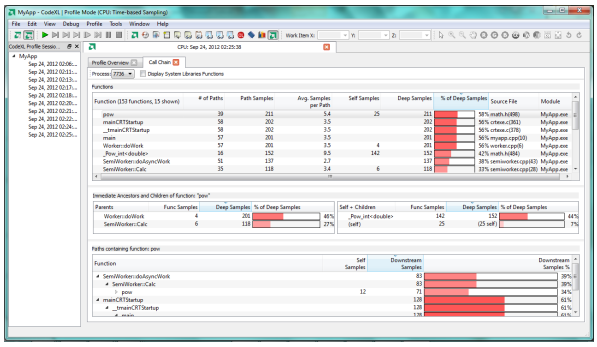

Figure 42: CodeXL Time-based Sampling
http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/2_TBP.png

CodeXL also offers call chain relationship analysis. This shows the functions in relation to their caller and callee. All CPU profiling features also work with with multiple threads. CodeXL also measures various CPU metrics such as CPU frequency, and CPU core thermal data. [52]

14 GPU Profiling

CodeXL offers real time API level debugging for OpenCL and OpenGL which shows the API function calls and which code paths led to them. Shader code can also be debugged by exporting the compiled file during runtiume and then loading it into CodeXL for debugging. Performance data is also collected through GPU counters, application trace, kernel occupancy and hotspot analysis. The data is gathered during run time and can then be evaluated. It also offers various GPU metrics such as the power consumption of discrete GPU components as well as thermal data and the GPU frequency. The GPU memory access characteristics and the actual execution time of the kernels are also tracked. [52, 53]

Part IV
Software Testing in Unity and C++

Jakob Maier and Simon Dimitriadis

Abstract — Bugs are hard to catch. A tool that allows for more efficiency is testing. The games industry or rather the whole software industry has come a long way from manual testing to fully automatic testing with automatic builds, deployments and reports. The problem with TDD is that it is hard to sell. Companies that are new to testing, may have a high inhibition level to start using it. But the benefits outweigh the doubts by far.

Keywords — Game Development, Test Patterns, Testing, Continuous Integration, TDD, gTest, Unity Test Tools

1 Introduction

The focus of this paper is to describe testing in different ways. Nowadays testing plays an important role in most software companies. There are many different ways of testing because there are various requirements for tests. This paper gives a short overview about the most important testing methods, common terminology of testing and useful patterns. Furthermore different fixture setups and assertion method styles are described. In the following chapters, two different methods of testing, F.I.R.S.T. and TDGD, are analyzed in greater detail. Afterwards, Google's approach to testing is discussed. Right after this chapter, Google's testing framework gTest as well as the unity test tools are presented.

2 Test Code Organization

This section describes how tests are usually organized in a project, independently of the used language.

2.1 Test Methods

Each test (-method) consists of four different parts that are executed consecutively and should not be mixed up (see Fig. 43). This is also called the four-phase model [54].

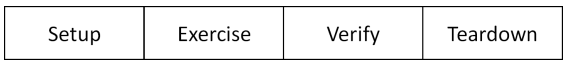

Figure 43: Code organization within a single test method

The **setup** component (also called fixture) sets up the required state to run the test. It instantiates objects (including the SUT and DOCs), initializes their internal state and creates the required connections between different objects. However, it only performs the minimum work needed to run the test.

The **exercise** part executes the code that should be tested. In case of unit tests, this is often just a single line of code that only calls one method on the SUT.

Afterwards, the **verify** code checks the test result and if everything worked correctly. This section decides it the test succeeded or failed using assertions.

The **teardown** component cleans up behind the test. It closes open handles (file handles and database connections) and frees previously created memory if the language does not provide a garbage collector.

2.2 Test Classes and Suites

Each test method only tests one very small aspect of the SUT. The whole SUT functionality should be split up in as many test methods as possible to avoid big, confusing test methods that have to be refactored in every code change of the SUT. It also

ensures easy defect localization [54, p. 154]. This leads to many test methods even in small projects and requires further structuring into test classes and so called test suites (see Fig. 44).

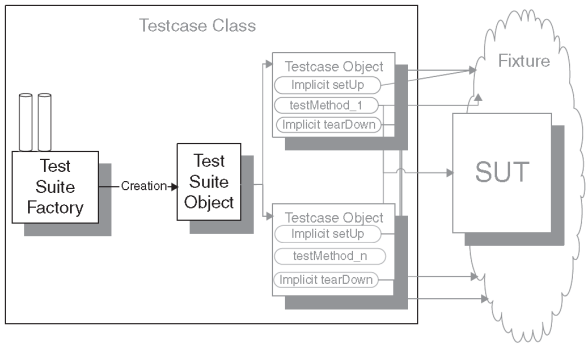

Figure 44: Code organization: suites and classes [54, Chapter 24]

A test class consists of multiple test methods, each with its individual setup and teardown components. There are different approaches to structuring tests into classes:

- **Test case Class per Class** is a very common approach that ensures that test case classes do not grow too big and that tests and their associated SUT can be located easily.
- **Test case Class per Feature** makes it easy to identify broken features.
- **Test case Class per Fixture** reduces the test code and makes the test classes more readable (only a single `setup()` method is needed). When using this technique, methods that require the same fixture (the same state before executing) are composited. However, this approach can easily result in scattering test conditions across many test classes.

Testing frameworks automatically create a test suite that contains all existing test classes. In order to run the tests, the all-tests test suite can be executed. In smaller projects, dividing all test methods into test classes is usually enough. However, as soon as the project grows, the execution time of test methods starts playing an important role. Each code change (even a small one) requires the developer to execute the complete test suite, including all test methods ever written. To avoid this, test classes can be further divided into so called test suites.

A test suite has a speaking name and is a collection of tests that belong together. Testing frameworks which support suites (like the xUnit family) allow classes to provide a test suite factory which returns a test suite. These factory methods create a new test suite object and attach other test suites, test classes or only single test methods to it. This allows developers to structure tests arbitrarily and only execute parts of the tests.

3 Terminology and Common Patterns

3.1 Fixture Setup Patterns

Test fixtures are used to set up the required state for running a test. Test fixtures instantiate and initialize the SUT as well as DOCs and are also responsible for preparing test doubles *(see chapter 3.4, p. 48)*.

Two kinds of fixtures exist:

- **Fresh fixtures** are part of the test method itself. They are not reused in other test methods and are teared down at the end of the test.
- **Shared fixtures** are constructed to be used in multiple test methods. These methods can either be called once for all tests (the test state is reused), or are automatically called before each test method (test code reuse). The former is especially useful for building expensive objects like in-memory databases.

Fig. 45 illustrates the different locations where fixtures and their corresponding teardown routines are located.

Fresh Fixture Setup

- **In-line setup** means that the constructor of the required objects (for example the SUT) is called directly within the test method.

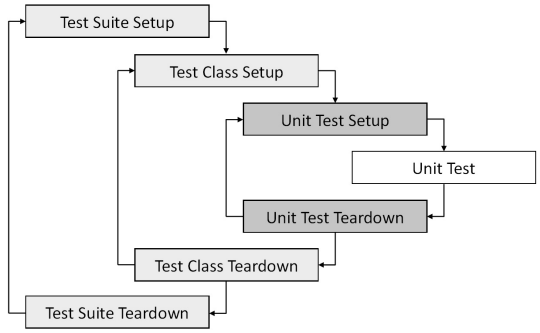

Figure 45: Location of fixtures and their corresponding teardown routines. The dark gray components are fresh fixtures, while the light gray routines are shared fixtures and therefore reused between multiple unit tests.

- **Delegated setup** facilitates creation methods that are responsible for returning usable objects.
- **Creation methods** are used if the construction of objects is more complex. For the test method itself it is unimportant how to construct such objects. To avoid unnecessary complexity, the code is moved to a separate method.
- **Implicit setup** are `setUp()` routines in the test class (or test suite) that are automatically called by the testing framework.

Shared Fixture Setup

- **Prebuild fixtures** are created implicitly by the testing framework before the first test is executed. They are usually located in the `setUp()` code of the test suite. This is useful for objects that are expensive to build (time consuming) like in-memory databases. Such objects are built only once and then reused by all tests. To avoid building the object for each test case, a singleton pattern can be utilized.
- **Lazy setup** is similar to the prebuild fixture with one key difference: the object is not build in advance, but instead when the first test method tries to access it. This can easily be implemented using a lazy initialized singleton but can lead to problems regarding the teardown execution, in which case the next strategy can be used:
- **Setup decorators** are wrapped around the test suite, containing a decorator that executes code before and after the execution of all contained tests.
- **Suite fixture setup** are the `setUp()` and `teadDown()` routines within the test suite.
- **Chained tests** reuse the fixtures that are left over from the previous test method. This is very similar to how a human tester tests code: while performing a long series of actions, the outcome of each intermediate step is verified.

3.2 Result Verification Patterns

There are two different kinds of verification strategies:

- **State verification** is the phrase if the SUT's internal state or the state of a DOC is verified after execution.
- **Behavior verification** is the solution if there exists no state that can be checked. In such a situation, the behavior on how the SUT interacts with its environment has to be evaluated. Usually this requires the use of test doubles as described in chapter 3.4, p. 48.

Assertion Method Styles

Depending on how assertions are used in test methods, different terminology is used to describe them. The following list contains some well-known terms for different assertion types.

- **Unfinished test assertions**
  A common approach in writing tests is to first think about all required test cases and writing down the methods without implementation code. Until the methods have been fully implemented, an *unfinished test assertion* is used to force a test failure. This kind of assertion is used to ensure that the developer does not forget to finish all test cases.
- **Custom assertions**
  There is often a need for test-specific object comparisons that are reused in multiple test methods. In such cases, assertions are moved

into a separate method to reduce duplicate code. Therefore, *custom assertions* are needed if multiple assertions should be condensed into a single method (assertion) call.

- **Delta assertions**
  These kind of assertions are often needed in case of shared fixtures (and therefore shared objects) that lead to tests influencing other tests. In such scenarios, the initial state of a test case can be different depending on which tests were executed beforehand.

  Delta assertions take two separate state snapshots, one before and one after test execution, and compare them afterwards. An example can be the number of database rows that have been inserted into a table.

- **Guard assertions**
  An important principle of writing tests is to avoid conditional test logic. Conditional test logic are `if`s which are used to skip assertions, a scenario that should never happen in well-written test methods. Conditional test logic is a great sign of code smell which shows that something is wrong.

  Sometimes, assertions must not be called since they would cause an error, like accessing an invalid object that solely exists because the SUT already failed the test. In these situations, *guard assertions* are used instead of `if`s to detect test failures and to abort test method execution before the dangerous assertions can be executed. Guard assertions can usually be found between the SUT execution phase and the verify phase.

- **Shared fixture state assertions**
  In case of shared fixtures, guard assertions can also be used at the beginning of a test to ensure that the fixture satisfies the test's needs. These guard assertions are called *shared fixture state assertions*.

3.3 Fixture Teardown Patterns

Fixture teardown is the process of closing open handles, unregistering events and freeing allocated memory. In languages like Java, the **garbage collected teardown** is able to do most of the work.

Other languages like C++ use **automated teardowns** where the allocated memory has to be freed manually. This can be difficult because the developer has to provide a clean teardown for all possible outcomes, even if the test failed and the SUT is in an invalid state.

Similar to the fixture setup *(see chapter 3.1, p. 46)*, the code organization of teardown logic distinguishes between **in-line teardown**, **delegated teardown**, **teardown methods** and **implicit teardown**.

3.4 Test Double Patterns

Test doubles are needed if the SUT depends on code that is cannot be included in the test. In these situations, the DOC has to be replaced by a so called *test double* [55].

- **Dummy objects** are the simplest approach and just act as a simple placeholder without containing any logic. The most basic forms of dummy objects are a null-object or passing a `new Object()`.
- **Fake objects** are very similar to stubs and often hard to distinguish. They implement the real object in the simplest way. An example for a fake object is a database access double that always returns the same, hard-coded row and does nothing if new data should be inserted.
- **Stubs** are influencing the SUTs to force specific code paths to be executed. It is often possible to tell the stub in the fixture which code paths the SUT should execute. In contrast to fake objects, stubs are trying to challenge the SUT by influencing it.
- **Spy objects** provide values for the SUT like test stubs. But in addition to that, they also capture and save the input for later verification of the test. Test spies are therefore very usable for behavior verifications *(see chapter 3.2, p. 47)*.
- **Mock** objects are similar to test spies, with the difference that they are verifying the ob-

served input while being executed instead of saving it for later.

Another distinction of test doubles is their construction:

- **Hard coded test doubles** are created once and behave the same way in every situation.
- **Configurable test doubles** are set up and configured in the fixture and can behave differently for different test methods which makes them more versatile and helps to reduce test code.

Fig. 46 illustrates the connections between the different test double types.

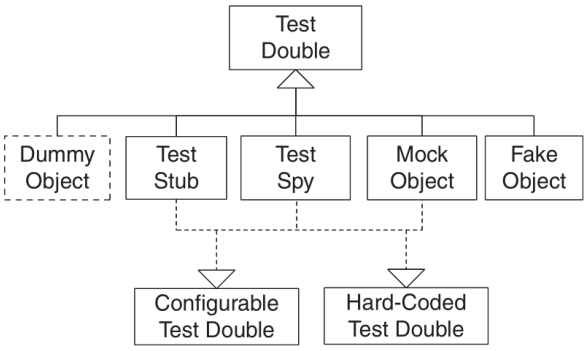

Figure 46: Relations between different kinds of test doubles [54, p. 527].

4 F.I.R.S.T.

In order to have useful and maintainable code, test code should follow the same guidelines as production code and a dual standard should be avoided. For more information about clean test code and clean code in general, see [56].

In addition to general coding guidelines, test code has five additional rules (F.I.R.S.T) to be considered as clean:

- **Fast**
  Tests should be run frequently to find problems as soon as possible, which requires them to run slowly. Tests that run slow cannot be executed after each change and problems are therefore detected late.
- **Independent**
  Test cases should be independent from each other. It must be possible to run the tests in an arbitrary order or even simultaneously using multi-threading. If tests depend on other tests, the first failing test will cause a cascade of further failures, hiding actual problems in all those dependent unit tests.
- **Repeatable**
  Tests should work the same way on every environment and must always be deterministic. If a test requires a specific testing environment to run (for example an active network connection or stable database), there will always be an excuse for failing tests, making them pointless.
- **Self-Validating**
  A test always returns a boolean: Either the test failed, or it succeeded. It should never be necessary to manually check the result for its success, for example by examining the console output for specific log statements.
- **Timely**
  Unit tests should be written just before the production code (TDD: test driven development). If tests are written after the production code has been finished, chances are high that they will never be written. Furthermore, if the production code is written first, it is possible that the written code is impossible to test, for example due to non-replaceable dependencies like system calls. This cannot happen if the tests are written beforehand.

5 Test driven Game Development (TDGD)

Fig. 47 shows the life cycle of a game development project which usually consists of four different parts.

Historically, game testing is performed in the last phase, the “post-production” phase, where both professional testers as well as beta testers are performing manual game testing by playing the game for many hours. Since automated testing got more

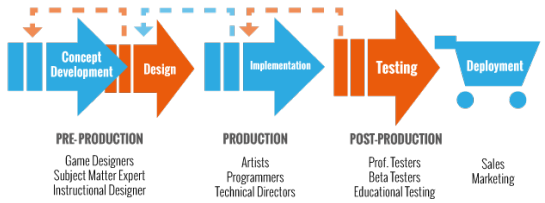

Figure 47: Game development life cycle

and more popular in the last few years, the game engineering sector also experienced major changes in regard to automated software testing and started to adapt agile ideas.

Thanks to many blogs like Noel Llopis's [57] there are indicators on how well TDD is acknowledged in the games industry. He addresses the fears and misconceptions that teams have about TDD. It can definitely be applied to games too. He points out numerous times that TDD is a guide to design the code base and that it is a development methodology. Francesco Carucci's presentation [58] on Testing in AAA games goes in the same direction. Actually it goes one step further and states that using TDD lowers production costs because the time spent on debugging and maintaining the code can be reduced drastically. He points to a chart published at gamasutra [59]. It shows the development phase of the Vision engine. They aggregated the bug reports and put the numbers of unit tests they had in their code base in the chart to put it into perspective. They stated that in the beginning they relied completely on manual test and even though every new version had been checked, the bug reports kept coming in. Many articles emphasize the fact that a fully automated continuous integration system is a key part to success, when quality code matters. These systems have to fulfill every task from build, deploy to automated testing with bug reports on errors.

Although many articles argue that TDD should be used, they are not in complete agreement on what can and should be tested. First of all it needs to be pointed out that many articles are quite old and the more recent ones claim that almost everything can be tested. They also provide some guidelines on how to write the code testable:

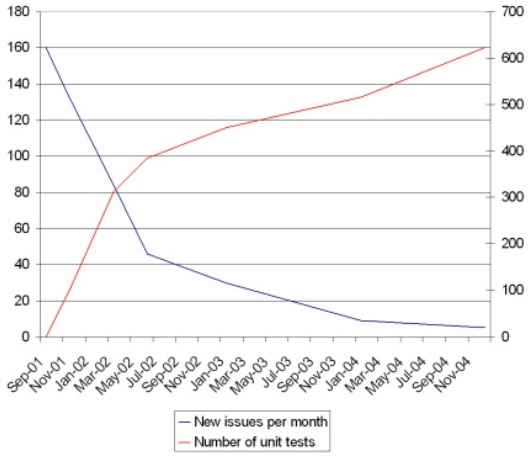

Figure 48: Describes the bug report / unit test curve for the Vision engine.

- Keep the code as simple as possible
- Look for modularity
- Have low dependencies
- Avoid complicated states in objects
- Keep tests independent from each other
- Keep tests very fast
- Remove randomness from tests

Some side effects are that you can use the unit tests as documentation, because every comment needs to be maintained and the compiler cannot check if you updated your comments according to your refactoring. But with unit tests, you have at least one example of how to use the API in a correct way. Another benefit is that you always have a check that what you implemented is what you meant the code to do. There is almost instant feedback. If you fail a test you know where to fix the broken code. You also have the advantage to test your code on different platforms as your build server checks them for you.

While it was discouraged to test things like graphic API calls or high level code in the past, it is now a normal thing to do. Llopis describes the way they approached it. For him there are three possible ways:

- Insert a layer between the graphics renderer

and the graphics API. That way every call can be intercepted, checked for errors and - most important - unit tested.

- Check for the API state. After every draw call the API is queried for the correct state. This approach has proven to be difficult because everything that has been altered needs to be cleaned up again. It also depends on the graphic API you want to use.
- Isolate graphic calls. This can be done if the other two options fail. That gives the advantage that every call to that external API goes through one single point and can be tested to some extent, but does not give a lot more information about the call. That shouldn't be a problem since the own code should be tested and not the API.

5.1 Tests during Game Development

The following section gives some examples for tests that can be easily utilized during game development.

- **Smoke testing**
  These kinds of tests can be easily performed using an automatic test exerciser. They usually focus on user interface logic (button clicking, menu opening, touch gestures) and don't require very sophisticated test logic. Although smoke tests don't provide very detailed results, they are still very powerful in finding simple flaws and providing quick feedback after code changes.
- **Functional testing**
  This is probably the single most important test group. Functional tests check that the game behaves correctly and are usually associated with manual testing and playing the "game through". A common approach to deal with this kind of testing is to write an AI that automatically plays the game at a much higher speed than any human player. This AI performs predefined actions and checks if the game responds as expected. With such a test, developers can avoid game-breaking bugs like unsolvable levels and unbeatable enemies due to malfunctioning game elements. If the AI is able to finish the level with a set of predefined actions, a human player should be able to do so too.
- **Regression testing**
  Regression tests are usually run after every code change and therefore before any commit. They ensure that existing functionality is still working correctly after new features have been added. It is important to note that regression tests should be performed regularly on all supported platforms to ensure a certain level of confidence. The tests should also be extended after new functionality has been added.
- **Performance testing**
  Most game engines already provide tools for performance testing and debugging, so it should be easy to utilize this information and verify the differences in performance after a new feature has been added. Performance testing can also give hints about other aspects of the game like battery consumption - an important aspect for mobile games.
- **Connectivity testing**
  Most games today utilize some sort of network connectivity. They have server-client interaction, login checks, social media features or just the ability to download additional game content while playing. Connectivity tests ensure that the network communication is working correctly and not only involve game code, but also server-side code that can be written in a completely different language. This kind of tests is also very important for anti-cheating functionality and security concerns.
- **Localisation tests**
  These kinds of tests become important as soon as the game should be released on a global basis. Localisation tests mainly test different translations on different platforms and ensure that GUI-layouts don't brake by comparing screen shots.

[60].

Last but not least the TDD way allows a company to stay agile and cope with change requests much more easy than without. Fig. 49 shows how difficult it would be if everything were implemented

the traditional way and testing would take place in the post-production phase only.

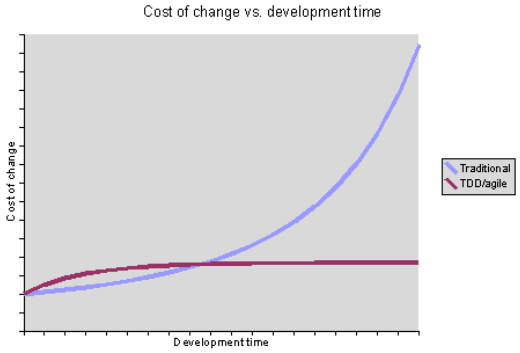

Figure 49: Cost of change over development time.

6 Case Example: Google

This chapter describes how testing is done at Google. For this paper the book "How Google tests software" [61] has been chosen. It may be less important for testing in games itself, but could become a handy tool for testing server applications. Also, the book that was chosen for reference describes not only how it is done technically but also how the whole company utilizes testers to ship quality code in quick iterations. The approach Google takes is called the *quality first approach*. The author says that the statement "quality cannot be tested" is wrong. He insists that the work flow should always be to write some code and then test this code with its according test. It is much more important to have a feature implemented rock solid rather than writing new ones. Google has also put some thought into the company's structure and how to combine it with testers that are not assigned to a certain division. First of all tests have to be implemented by the developers that write the code in the first place, simply for the reason that the author of the code should most likely spot the bug or take the least time to debug the code.

Quality can only be achieved if testing is an unavoidable aspect of the development. That is enforced not only through the company's structure but also through its code repository. Google defines different roles for different aspects of testing:

- **The software engineer (SWE)** does the usual coding in TDD but with a focus on features. He also does the majority of feature implementations.
- **The software engineer in test (SET)** is also a developer but is more the one that has to keep an eye on quality. He reviews code and improves it to be easier to maintain and to test. Not only does he improve the testability, he also provides the test infrastructure.
- **The test engineer (TE)** is related to the SET role but has a different focus. He tests from the users perspective and not the developers perspective. His job is to write automation scripts that mimic users, he provides scenarios and he interprets the executed test and the result.

These testing roles are part of a horizontally structured department. Its members work on all the projects and report to their own executive. Every tester can raise any kind of concern, whether it is that there are too many bugs or a security problem. If a development team wants to take any kind of shortcuts like releasing a new version with fewer tests or the like, this has to be negotiated with the testing department.

Google always aims to release a new build as quickly as possible with some limitations. The reason they want to get to the costumers as quickly as possible is that they need the feedback quickly. If a brand new product is developed, they have a rule of thumb that the core product should be built and finished as fast as possible. As soon as the core is stable, it will be released so that a large group of people can be reached very early. After that, the team has to iterate as fast as possible again and consider user feedback. A thing to keep in mind though is that, just because it is an early version, it does not mean that it is bad or halfheartedly made.

Google does not distinguish between **code**, **integration** and **system** testing. They name the tests after the scope size. Scopes can be:

- **small**

- **medium**
- **large**

The smaller a test, the more likely it is to be automated. Small tests cover only units of code, medium tests cover multiple units in a faked environment and large tests cover any number of units in production environment and without (!) any faked resources. The coverage for each category has to be within acceptable limits. For example, if medium and large tests cover only a low percentage and small tests cover most of the system, the system probably has end to end user problems.

It is also important to note that tests should fulfill the following:

- They are independent so they can be executed in any order.
- They must not have any side effects.
- They must not alter the environment or require any environment modifications.

A general rule of thumb to distinguish what should be tested automatically is whether it requires human intuition to test. If it cannot be automated in the first place, the tests are recorded whenever possible and repeated by scripts or similar methods. Very important in the whole testing process is that the section that leads to a test failure is pinned down and the authors of the affected code are notified automatically. The system should also file a bug report with the authors tagged to it.

It has already been mentioned that testing is a horizontally structured department at Google. Therefore, it is not guaranteed that every project has testers of said department on board. Every project team has to request staff form this department. They have to convince the testers that their project is exciting and full of potential.

The basic work flow at Google is one of the following:

1. **Test planning**
   It is done with Google Test Analytics. It is risk-based, quick and automatically updated.
2. **Test coverage**
   New versions are checked by bots for differences. If they find any differences, they are reported to humans for quick evaluation.
3. **Bug evaluation**
   It is determined whether these differences are bugs or new features.
4. **Exploratory testing**
   This is done by crowd testers and early adopters. These bugs are mostly related to configuration and context. These bugs are hard to spot and require human intelligence to induce and report.
5. **Triage and debugging**
   Realtime dashboards provide an overview of bug trends. Bugs are reported with the data needed to investigate the problem. Usually there is also a recording on how to reproduce the bug in front of the developers' eyes.
6. **Deploy and return to step 1.**
   Rinse and repeat.

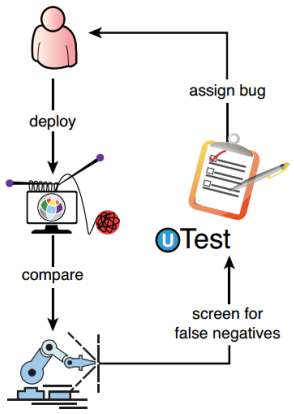

Figure 50: Workflow at Google

Every engineer at Google can change his projects after a period of 18 months - if he wants to. Therefore, it is a policy that dependencies on people should be avoided. This means that if there are rock star like developers in a team, everything should be done to embody whatever this person is so good at into some tool or packaged in a way to be useful for others.

One problem that arises is the following: When testing becomes a service that everyone can outsource to the testing department, developers

won't write the code to be easily tested. - Simply because they don't think about it while they are writing it. Therefore, quality and TDD has to become a mindset of any developer in the company.

Google's testing process in a quick overview:

- Put quality in the work flow of every engineer.
- When done earnestly and honestly, quality increases.
- Freshly written code is better.
- Early builds are better.
- Integrations are unnecessary. System testing can concentrate on real user-oriented issues.
- Keep the projects and all the engineers free of accumulated bug debt.

6.1 Applying Google's Approach to the Games Industry

It has been pointed out in the chapters before that TDD is entering the games industry. It has definitely proven to be very important in other parts of the industry. Many systems could not be created or operated without properly tested code. As systems grow bigger and bigger, it is hard to keep track of everything necessary. Even if it may be difficult to unit test games, in particular everything that appears only visual, it might be a good advice that doing at least some testing is better than no testing.

What we can learn from Google's approach would be: Test whatever can be tested automatically and then go back to the tests where human interaction is necessary. Then you have to decide what can be scripted after recording and what has to be manually tested all the time. It may turn out to be very little that cannot be tested automatically after all.

7 Testing C++ with Google Test

A straight forward tool to write unit tests in C++ is gTest by Google. With its current version 1.7.1 some examples are implemented to be shown in this paper. There are different ways to setup a project but one thing that always has to be kept in mind is that test code should not be shipped. For that reason the project (VS: "Solution") is split in different parts.

1. A DLL that houses all the code under test.
2. The Google test project that uses the DLL and tests it. The test code is in this project.
3. A thin client that produces the end result and contains code that will not be tested. This is usually built to an exe, utilizing functionality from the DLL.

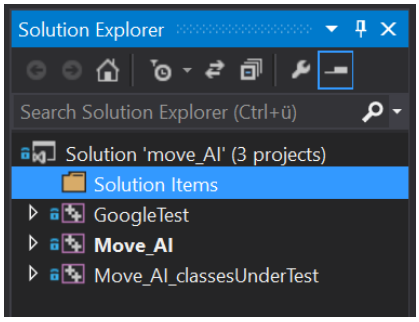

Figure 51: Solution setup in Visual Studio

Fig. 51 shows how the solution setup could look like. It is also possible to have the main project house the test code and to stripe everything out on a Release Build with custom scripts. To use gTest, it is only necessary to include it via Nuget or set it up the usual way via include paths in Visual Studio's project properties. The main function for gTest can be seen in Fig. 52. It only takes one include and one call to get everything running.

```
1 #include <stdio.h>
2 #include "gtest/gtest.h"
3
4 GTEST_API_ int main(int argc, char **argv) {
5     std::cout << "Running main() from gtest_main.cc\n";
6     testing::InitGoogleTest(&argc, argv);
7     return RUN_ALL_TESTS();
8 }
9
```

Figure 52: Google test main

After that, every test in the gTest project is found automatically and will be called sequentially. There are also different ways to execute tests, one is building the gTest project into an exe and start it seperatly. This is also useful for continuous integration

because it only requires MSBuild to be installed. An output example of this test runner can be seen in Fig. 53. Another option is to use Resharper as seen in Fig. 54.

```
Running main() from gtest_main.cc
[==========] Running 8 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from RandomTest
[ RUN      ] RandomTest.RangeTest01
Setup test:
[       OK ] RandomTest.RangeTest01 (4 ms)
[ RUN      ] RandomTest.RangeTest01_Binomial
Setup test:
[       OK ] RandomTest.RangeTest01_Binomial (5 ms)
[----------] 2 tests from RandomTest (9 ms total)

[----------] 6 tests from InstantiationName/RandomTestKeepFixture
[ RUN      ] InstantiationName/RandomTestKeepFixture.RangeTest01/0
setup test:
[       OK ] InstantiationName/RandomTestKeepFixture.RangeTest01/0 (3 ms)
[ RUN      ] InstantiationName/RandomTestKeepFixture.RangeTest01/1
setup test:
[       OK ] InstantiationName/RandomTestKeepFixture.RangeTest01/1 (3 ms)
[ RUN      ] InstantiationName/RandomTestKeepFixture.RangeTest01/2
setup test:
[       OK ] InstantiationName/RandomTestKeepFixture.RangeTest01/2 (3 ms)
[ RUN      ] InstantiationName/RandomTestKeepFixture.RangeTest01_Binomial/0
setup test:
[       OK ] InstantiationName/RandomTestKeepFixture.RangeTest01_Binomial/0 (4 ms)
[ RUN      ] InstantiationName/RandomTestKeepFixture.RangeTest01_Binomial/1
setup test:
[       OK ] InstantiationName/RandomTestKeepFixture.RangeTest01_Binomial/1 (4 ms)
[ RUN      ] InstantiationName/RandomTestKeepFixture.RangeTest01_Binomial/2
setup test:
[       OK ] InstantiationName/RandomTestKeepFixture.RangeTest01_Binomial/2 (4 ms)
[----------] 6 tests from InstantiationName/RandomTestKeepFixture (26 ms total)

[----------] Global test environment tear-down
[==========] 8 tests from 2 test cases ran. (38 ms total)
[  PASSED  ] 8 tests.
```

Figure 53: Google test exe

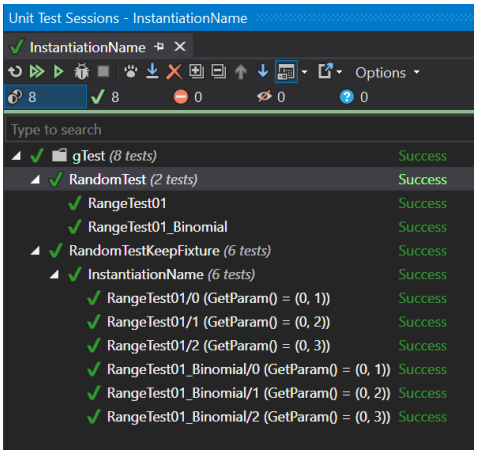

Figure 54: Google test resharper

Resharper allows the execution of every test individually. It can also run tests grouped together and is capable of debugging test code really easily. Tests are written with some very basic defines. The most simple example might be shown in Fig. 55. Tests are defined with the `TEST`-define, a test group name as well as a test name. Assertions are inserted with different variants of the `EXPECT` macro.

As soon as some state has to be set up for more that one test, fixtures become a handy tool. To create one, the `Test` class has to be extended and is then provided as the first parameter for the `TEST_F`

```
TEST(SimpleGroup, SimpleTest) {
  auto value = 9;
  EXPECT_TRUE(value == 9);
}
```

Figure 55: Assertions in gTest

defines. Fig. 56 shows an example.

```
class RandomTestFixture : public ::testing::Test {
protected: // protected or public
  virtual void SetUp() override {
    std::cout << "Setup test: \n";
    sd::Seed();
  }
  virtual void TearDown() override {}
};

TEST_F(RandomTestFixture, RangeTest01) {
  for (auto i = 0; i < 100000; ++i) {
    auto rand = sd::Random();
    EXPECT_GE(rand, 0);
    EXPECT_LE(rand, 1);
  }
}

TEST_F(RandomTestFixture, RangeTest01_Binomial) {
  auto hasValBelowZero = false;
  auto hasValGreaterZero = false;
  for (auto i = 0; i < 100000; ++i) {
    auto rand = sd::RandomBinomial();
    //std::cout << rand << "\n";
    EXPECT_GE(rand, -1);
    EXPECT_LE(rand, 1);
    if (rand < 0) {
      hasValBelowZero = true;
    } else if (rand > 0) {
      hasValGreaterZero = true;
    }
  }
  EXPECT_TRUE(hasValBelowZero);
  EXPECT_TRUE(hasValGreaterZero);
}
```

Figure 56: Google test tests with a fixture

Parameterized functions become useful as soon as the same functionality should be called with lots of different data. Parameterized tests are written like conventional tests with a fixture, with the difference that they are instantiated using the `INSTANTIATE_TEST_CASE_P` macro.

Fig. 57 shows a simple example how parameterized tests look like.

8 Testing with Unity

As soon as Unity is used to develop a game project, “Unity Test Tools” is the way to go for implementing test cases.

“Unity Test Tools is a test framework for the games

```
class RandomTestKeepFixture
    : public testing::TestWithParam<std::pair<int, int>> {
public:
  virtual void SetUp() override {
    std::cout << "setup test: \n";
    sd::Seed();
  }
  virtual void TearDown() override {}
};

INSTANTIATE_TEST_CASE_P(InstantiationName, RandomTestKeepFixture,
                        ::testing::Values(std::pair<int, int>{0, 1},
                                          std::pair<int, int>{0, 2},
                                          std::pair<int, int>{0, 3}));

TEST_P(RandomTestKeepFixture, RangeTest01) {
  auto test = GetParam();
  for (auto i = 0; i < 100000; ++i) {
    auto rand = sd::Random();
    EXPECT_GE(rand, test.first);
    EXPECT_LE(rand, test.second);
  }
}

TEST_P(RandomTestKeepFixture, RangeTest01_Binomial) {
  for (auto i = 0; i < 100000; ++i) {
    auto rand = sd::RandomBinomial();
    EXPECT_GE(rand, -1);
    EXPECT_LE(rand, 2);
  }
}
```

Figure 57: Google test parametrized tests

and interactive content you make in Unity. By using these tools you can build quality products from the ground up. Unity Test Tools are set up to feel like a natural extension of the Editor, efficient and intuitive to use and they are free on the Asset Store." [62]

8.1 Integration Test Framework

Integration tests are designed to use a completely separate scene only for testing. This scene can be considered as a test suite *(see chapter 2.2, p. 45)* and can continue multiple tests. Tests are executed one after another (not in parallel) to avoid mutual influences.

Test Execution Procedure

1. Play mode is enabled
2. The first (or next) test object is activated
3. Wait until test has finished or a timeout occurred
4. Current active test gets deactivated
5. If there are more tests, go to step 2
6. Report results and finish test run

Test objects are conventional GameObjects that have a TestComponent attached. Every GameObject that is under the hierarchy of the test object is considered to belong to this test only. Every other GameObject is common for every test on the scene.

Note that test objects can also be nested to form test groups, in which case the parent test object is not executed as a discrete test. Fig. 58 shows an example of two test groups with seven tests, three of them failing and one configured to be ignored.

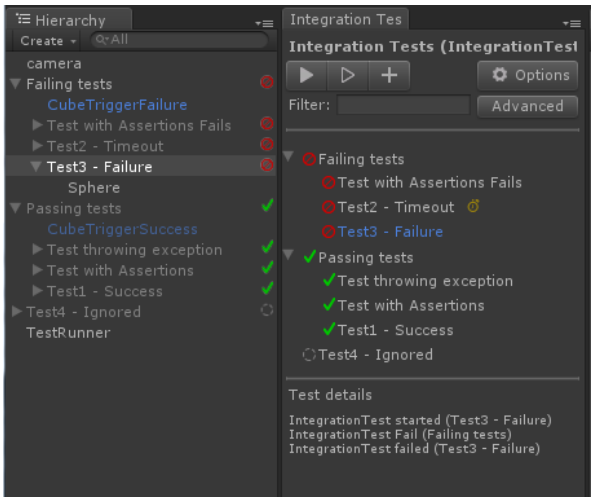

Figure 58: Integration Tests with Unity Test Tools

Manual Integration Tests

The easiest way to add tests to a Test Scene is by adding a Test Component to the desired GameObject. Every test component gets a reference to the associated test script and holds settings like the timeout or expected exceptions. An alternative is to use the Test Runner for adding new tests *(see chapter 8.1, p. 57)*. However, these test objects can also be created automatically by using so called "Dynamic Integration Tests".

Dynamic Integration Tests

A more convenient way to add tests is to annotate the test scripts themselves from within the code. In order to mark a class as a test class, it has to derive from `MonoBehaviour` and contain the attribute `[IntegrationTest.DynamicTest (testSceneName)]` on top of it. The runner automatically discovers the new test class by using reflection and displays it on the list.

Other testing attributes can be added as well:

- `[IntegrationTest.Ignore]`
- `[IntegrationTest.ExpectExceptions (false, typeof(ArgumentException))]`
- `[IntegrationTest.SucceedWithAssertions]`
- `[IntegrationTest.Timeout(1)]`
- `[IntegrationTest.ExcludePlatform()]`

Tests are executed automatically as soon as the test object gets enabled *(see chapter 8.1, p. 56)* and can have different conditions to be stopped:

- `Testing.Pass()` gets called, the test succeeds.
- `Testing.Fail()` gets called, the test fails.
- The timeout is reached and the test gets aborted. It fails.
- An unknown exception gets thrown, the test fails.
- A known exception is thrown and the test passes.
- Every assertion component has been checked at least once.

Integration Test Runner

Since Unity 5.3, the Test Runner became part of the Editor itself and was removed from the Unity Test Tools package.

Fig. 59 shows the Controls of the Test Runner within the Unity IDE:

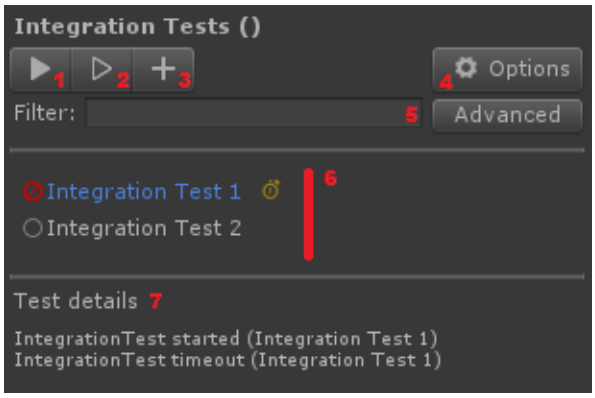

Figure 59: The Unity Integration Tests Runner

1. Run all tests in the scene
2. Run selected tests only
3. Create a new test object
4. Various test runner options
5. Filter tests by name or state (succeeded, failed, ignored, ...)
6. List all tests that are part of the scene
7. Log messages and exceptions (test console)

Platform Runner

Unity also comes bundled with the "Platform Runner" which can be used to run the test scene(s) automatically on different platforms like Android.

For more information on how to use Unity Test Tools, refer to their wiki at [63].

8.2 Assertion Components

Assertion components are the second most important part of Unity Test Tools. They are used to assert desired states of game objects and decide if tests should pass or fail. Assertion components can be fully set up and configured using the GUI and consist out of a condition that must evaluate to true, as well as a timing component that defines when the condition should be checked. If a condition is not met, an exception is thrown in order to allow the investigation of the issue. It is also possible to enable "Error on pause", which pauses the execution whenever an error occurs.

Fig. 60 shows an example for such an Assertion Component. Note that is it also possible to write custom Assertion Components.

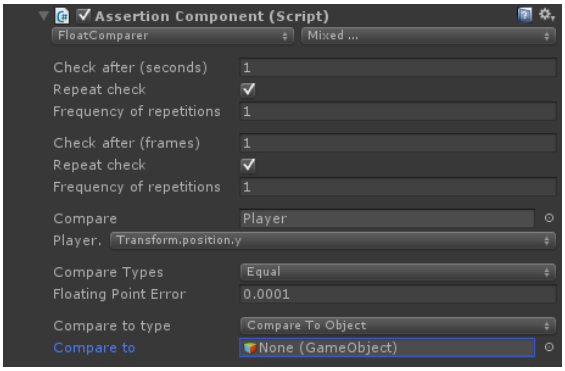

Figure 60: Unity Assertion Components

8.3 Continuous Integration

It is also possible to run the integration tests from within a command line without opening the Unity IDE:

```
Unity.exe -batchmode -projectpath ...
    -executeMethod UnityTest.Batch.↩
        RunIntegrationTests
    -testscenes=TheScene
    -targetPlatform=StandaloneWindows
    -resultsFileDirectorty=Results
```

9 Conclusion

As we have shown in this paper, testing is not only an important part in the software industry but also in the games industry. Certain rules and guidelines for testing are advantageous. These are described in the first part of the paper. The second part focuses on testing in real projects. Google's testing culture is described in greater detail. Also, their testing framework gTest is introduced. As a comparison we have also analyzed Unity's testing possibilities. This paper has shown that testing in games is definitely possible and should be done regardless of the size of the project.

10 Index of Abbreviations

DOC dependent-on component

GUI graphical user interface

SET software engineer in test

SUT system under test

SWE software engineer

TDD test driven development

TDGD test driven game development

TE test engineer

Part V
Software Architecture

Florian Krauß and Dino Madach

Abstract — Software architecture is able to grant a lot of benefits in the process of software engineering. Furthermore it supports engineers in avoiding problems occurring during this process. This article will show why software architecture becomes more and more important to the modern world and how it can be presented to any kind of stakeholders. That includes a short description of the most important UML basics and the evaluation of an UML visualisation tool: the Enterprise Architect. Additionally there is an introduction to model driven architecture to present its purposes.

Keywords — Software Architecture, UML, Model Driven Architecture, Enterprise Architect

1 Definition

"Software architecture is about dismantling and assembly with respect to style and aesthetics." [64] There are a lot of different opinions about what software architecture exactly is. In a historical view the word "architecture" is build from the greek words "arché" (begin, source) and "téchne" (art, trade). So literally translated it means something like "the first art". [65]

In connection with buildings you could say, the first art, the architecture, is what has to be done before constructions can be made. Let take this for software as well. According to [66] in the classic waterfall model there are two phases in software engineering before you start with any implementations. These are analysis and design. "From this perspective it can be handled as a bridge between the initial idea and the beginning of its realization. Specifically software architecture of a software system has to define its components, their interfaces and relations and their externally visible features."[67]

There are two kinds of requirements for software: functional and non-functional. The functional requirements are mostly mandatory and, if you take a closer look at it, often not the reason for project disrupting issues. On the other hand, non-functional requirements like performance, compatibility, extensibility, etc. need far more effort to be achieved. That is also what [67] meant with "architecture creates quality". In conclusion software architecture has three greater purposes:

1. It grants a mutual understanding of the problem and provides a common platform for further discussions.

2. If gives an structural overview over the system divided into its components, interfaces and the relations between them.

3. For later maintenance work or extensions it serves as documentation to the engineers.

You are very lucky if your current project (or maybe the one your looking forward to) is settled "in the open countryside". But be aware: in software engineering there is an anti-pattern, called "reinventing the wheel". In this case anti-pattern just means: something that often happens in practice, you should really avoid to do. Don't try to reinvent the wheel with your software architecture! If you run across a problem, don't think you are the first engineer that ever had to solve it. There are a whole lot of solutions to well-known and repeatedly arising problems in this field of research. The next section will introduce the most common solutions to you, the so called patterns.

2 Architectural Patterns

"The idea of patterns as a way of presenting, sharing, and reusing knowledge about software systems is now widely used. [...] In this section, I introduce architectural patterns and briefly describe a selection of architectural patterns that are commonly used in different types of systems. For more information about patterns and their use, you should refer to published pattern handbooks" [66]

2.1 Model View Controller

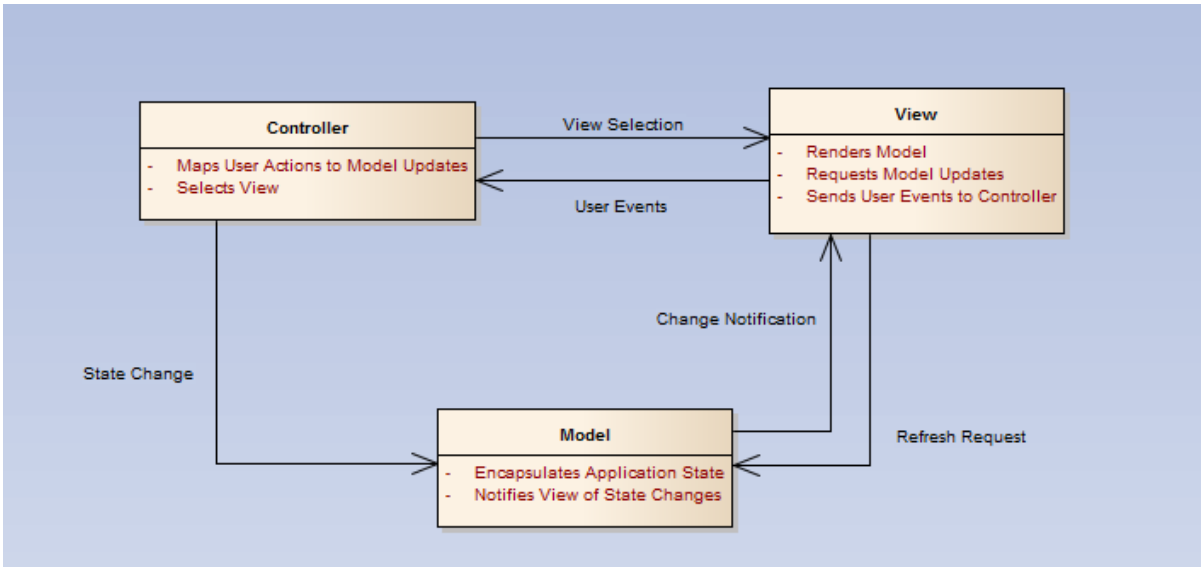

Figure 61: [66]

"This pattern is the basis of interaction management in many web-based systems. The stylized pattern description includes the pattern name, a brief description (with an associated graphical model), and an example of the type of system where the pattern is used (again, perhaps with a graphical model). You should also include information about when the pattern should be used and its advantages and disadvantages. A Graphical model of the architecture associated with the MVC pattern is shown in Fig. 61 which is a conceptual view. A possible run-time architecture when this pattern is used for interaction management in a web-based system would require a browser element in addition." [66] "Porting an application to another platform shouldn't lead to a revised version of the whole application. Simple changes or extensions and reusability of single components is the goal. User interfaces changes very often. The same information is to display in different windows in various forms against the complexity of the required frameworks. Different user-groups need different processing. The difficulty here is to weigh up a consistent view to a model to performance issues, caused by an excessive amount of updates" [68]

Advantages

"Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them."[66]

Disadvantages

"Can involve additional code and code complexity when the data model and interactions are simple."[66]

2.2 Repository

"[...]the repository pattern Fig. 62 describes how a set of interacting components can share data. The majority of systems that use large amounts of data are organized around a shared database or repository. This model is therefore suited to applications in which data is generated by one component and used by another. Examples of this type of system include command and control systems, management information systems, CAD systems, and interactive development environments for software. Fig. 62 is an illustration of a situation in which a repository might be used. This diagram shows an IDE that includes different tools to support model-driven development. The repository in this case might be a version-controlled environment [...] that keeps track of changes to software and allows roll-back to earlier versions. Organizing tools around a repository is an efficient way to share large amounts of data. There is no need to transmit data explicitly from one component to another. However, components must operate around an agreed repository data model. Inevitably, this is a compromise between the specific needs of each tool and it may be difficult or impossible to integrate new components if their data models do not fit the agreed schema. In practice, it may be difficult to distribute the repository over a number of machines. Although it is possible to distribute a logically centralized repository, there may be problems with data redundancy and inconsistency. In the example shown in Fig. 62, the repository is passive and control is the responsibility of the components using the repository. An alternative approach, which has been derived for AI systems, uses a 'blackboard' model that triggers components when particular data become available. This is appropriate when the form of the repository data is less well structured. Decisions about which tool to activate can only be made when the data has been anylyzed." [66]

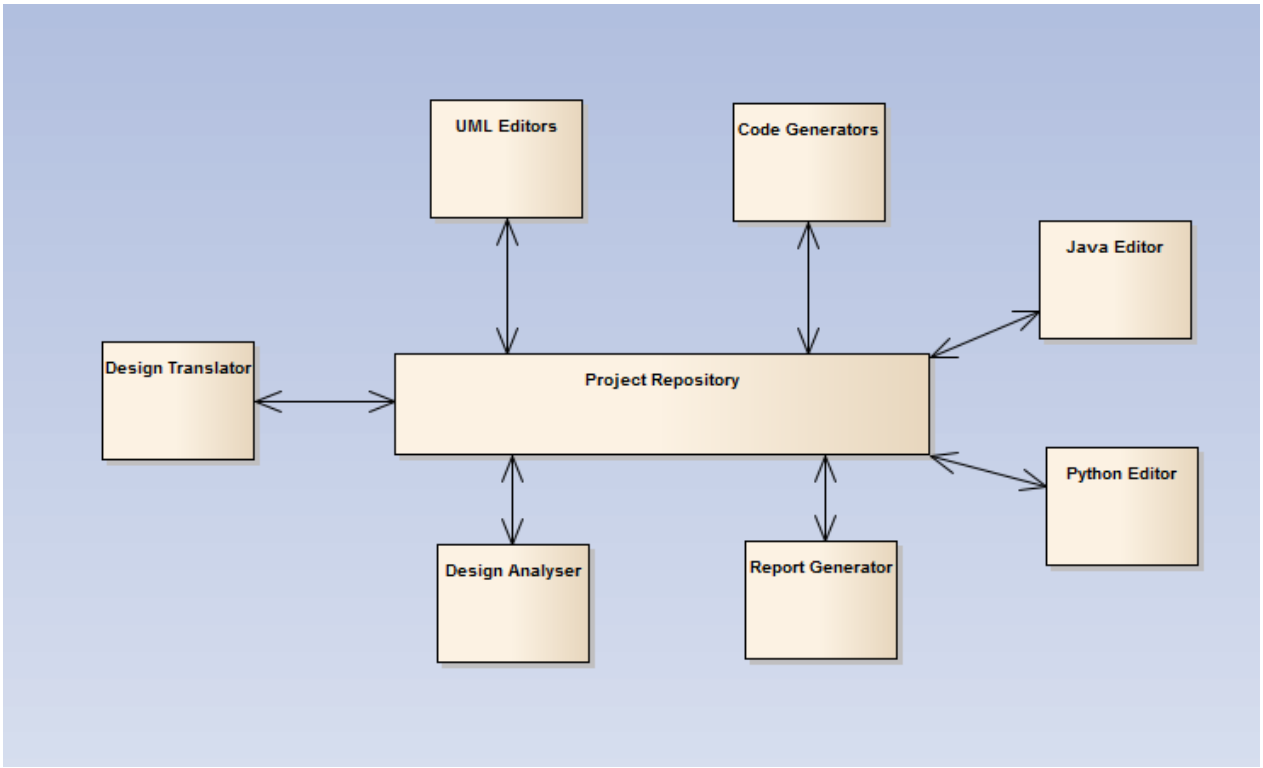

Figure 62: [66]

This pattern is highly recommended when you have large volume of data over a long period of time. In specific, if generated data is able to trigger actions (activate a security tool, start backup, etc.)

Advantages

"Components can be independent - they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups don at the same time) as it is all in one place."[66]

Disadvantages

"The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult."[66]

Blackboard

The above mentioned blackboard is defined as the following: "Several specialised part-systems provide their knowledge to create a possibly incomplete or just approximated solution." [68] "Parts of a blackboard are: - one or multiple independent knowledge sources investigate a problem under specific criterias and send suggested solutions to the blackboard. - The central blackboard manages the suggested solutions or parts of them sent by the knowledge source. - The control-component observes the blackboard and controls at needs the execution of the knowledge source." [68] The blackboard would be an additional element here. Take the knowledge source as the repository and the control class in Fig. 63 as one component. As we already know, components can be reused on other repositories which is a quality criteria for software. Furthermore, the model can be extended as much as you want until it fits your personal needs.

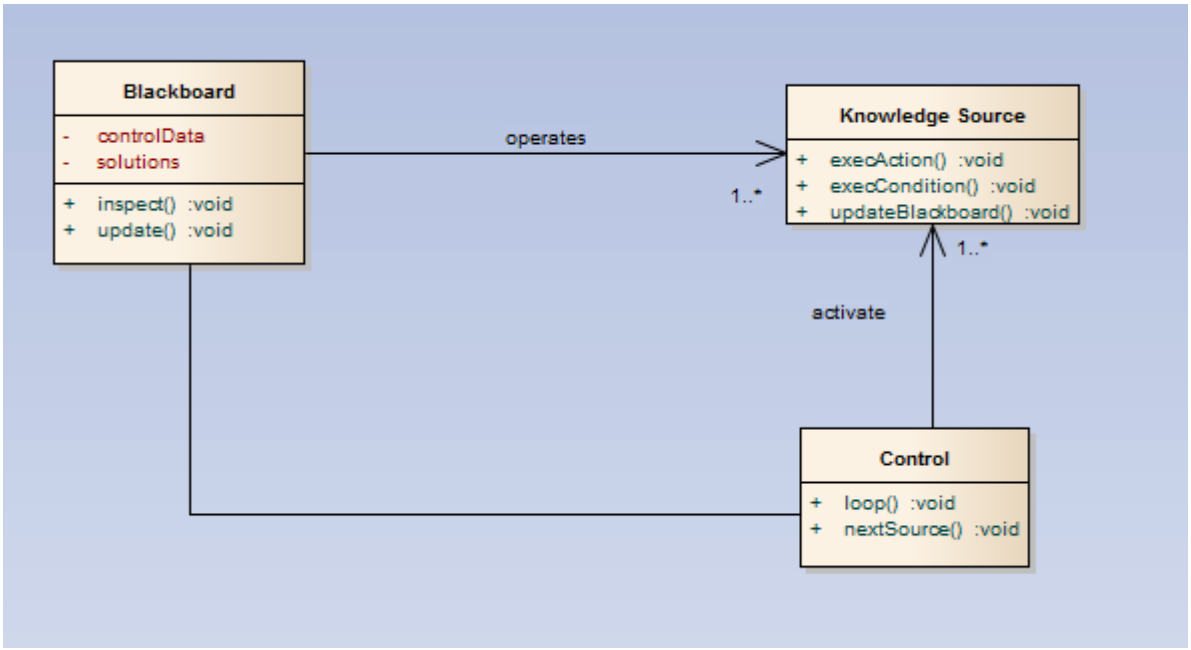

Figure 63: [68]

2.3 Client Server

"The repository pattern is concerned with the static structure of a system and does not show its run-time organization. My next example illustrates a very commonly used run-time organization for distributed systems. The Client-server pattern is described in Fig. 64.[...] A system that follows the client-server pattern is organized as a set of services and associated servers, and clients that access and use the services. The major components ofthis model are:

1. A set of servers that offer services to other components. Examples of servers include print servers that offer printing services, file servers that offer file management services, and a compile server, which offers programming language compilation services.

2. A set of clients that call on the services offered by servers. There will normally be several instances of a client program executing concurrently on different computers.

3. A network that allows the clients to access these services. Most client-server systems are implemented as distributed systems, connected using Internet protocols.

Client-server architectures are usually thought of as distributed systems architectures but the logical model of independent services running on separate servers can be implemented on a single computer. Again, an important benefit is separation and independence. Services and servers can be changed without affecting other parts of the system. Clients may have to know the names of the available servers and the services that they provide. However, servers do not need to know the identity of clients or how many clients are accessing their services. Clients access the services provided by a server through remote procedure calls using a request-reply protocol such as the HTTP protocol used in the WWW. Essentially, a client makes a request to a server and waits until it receives a reply. Fig. 64 is an example of a system that is based on the client-server model. This is a multi-user, web-based system for providing a film and photograph library. In this system, several servers manage and display the different types of media. Video frames need to be transmitted quickly and in synchrony but at relatively low resolution. They bay be compressed in a store, so the video server can handle video compression and decompression in different formats. Still pictures, however, must be maintained at a high resolution, so it is appropriate to maintain them on a separate server." [66]

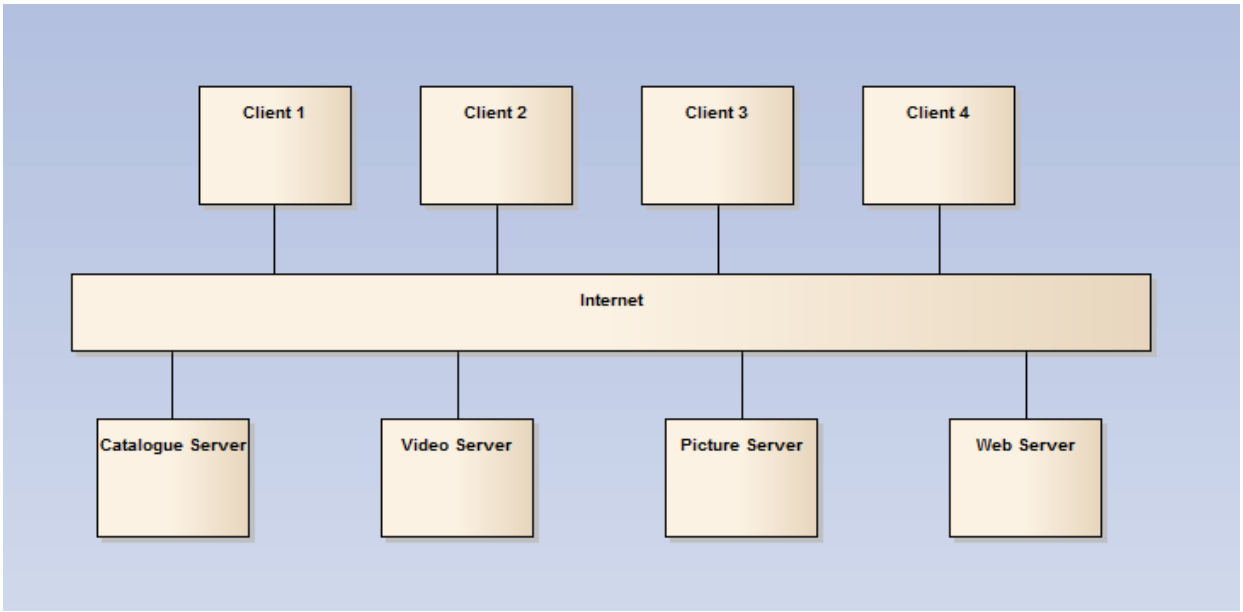

Figure 64: [66]

You can find a more simple representation of this model in [68] who describes it more general as "service oriented architectures" "In general services pro-

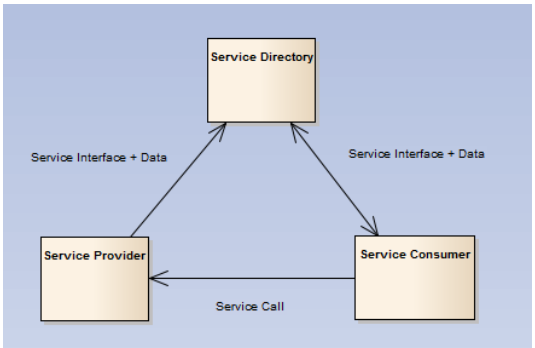

Figure 65: [68]

vide rough interfaces. You talk about rough, if the services enable complex functionality with few calls. [...] Services are locally independent and can be activated anytime from any location, if user and application have proper rights" [68]

Advantages

"The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services."[66]

Disadvantages

"Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations."[66]

3 UML

This chapter will give a brief overview of the history of origins, what UML essentially is and presents a few design patterns.

3.1 History of origins

In the late 1980s, many object-oriented modeling methods were being developed. Most of those methods used different visual modeling techniques and notations. About 50 object-oriented methods and as many design formats have been developed and were used in the mid 1990s. The most widespread methods used in the industry were the Object Modeling Technique (OMT) developed by Jim Rumbaugh, the Object-Oriented Software Engineering by Ivar Jacobson and the Booch Method by Grady Booch. The Rational Software Corporation began in the mid 1990s the development of a unified modeling language. They hired Jim Rumbaugh to join Grady Booch in the development. They combined their methods which became the version 0.8 of the Unified Method. 1995, Ivar Jacobsen joined them at Rational software and together they developed version 0.9 a year later in 1996. In 1997, the version 1.0 of the Unified Method was finished and they submitted it to the Object Management Group (OMG) under the name Unified Modeling Language (UML). The OMG took over the development and release new version of the UML. The current version in 2015 is 2.5.

3.2 What is UML

UML stands for the Unified Modeling Language and is a standard for modeling businesses, software applications, and system architectures. It uses a variety of diagrams for the visual representation and to communicate in a standardized way the overall design of a system. The UML is a standard of the OMG, but due to its flexibility and the possibility to customize it, the language is not only a well adjusted tool to model object-oriented (OO) software applications. This graphical language allows to model almost anything like businesses, data, organizations, theoretical political systems, legal

contracts, biological systems, languages, hardware, non-object-oriented applications and many more. UML is an open modeling standard, so it can be used by anyone. Corporations, consultants, software companies and even governments who need and rely on UML as a standard, further support its development and improvement.

3.3 Common diagrams

As a graphical modeling language, UML makes heavy use of diagrams to model a system. It differentiates between two types of diagrams, the structure diagrams and the behavior diagrams. "Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts." [69]

"Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time." [69]

Structure diagrams

- Class diagram
  - Displays the static structure of classes and their relationships towards each other. (associations, aggregations, as well specialization and generalization)
- Composite structure diagram
  - Displays the inner composition/structure of a classifier
- Component diagram
  - Displays components, their interfaces, ports and the relationships between them
- Deployment diagram
  - Displays the infrastructure and their dependencies as well as the distribution of runtime elements on the hardware
- Object diagram
  - Displays a snapshot of the class diagram. It shows a structure of the instances and their connections
- Package diagram
  - Displays an overview of the complete system through packages of subsystems. It contains logical summaries of the system parts.
- Profile diagram
  - "Describes lightweight extension mechanism to the UML by defining custom stereotypes, tagged values, and constraints. Profiles allow adaptation of the UML metamodel for different platforms and domains"[70]

Behavior diagrams

- Activity Diagram
  - Displays the flow of control or object flow. It shows the sequence and conditions of an activity and can describe e.g. a Use-Case in detail
- Use-Case Diagram
  - Displays a overview of all processes the system uses to react to the input of actors or extern systems
- State machine diagram
  - Displays the discrete behavior of a part of the system with a finite state machine

Interaction diagrams

- Interaction overview diagram
  - Shows the interplay of interactions and consists usually of references to other interaction diagrams
- Communication diagram
  - Displays the interaction through messages between objects

- Sequence diagram
  - Shows for a specific scenario the communication between objects and/or system parts
- Timing diagram
  - Displays state changes dependent on time

3.4 Design Patterns

3.5 Command Pattern

The command pattern belongs to the behavioral design patterns and is concerned with the communication between objects. It addresses the problem to be able to send requests, without knowing neither the requested operation nor the receiver.

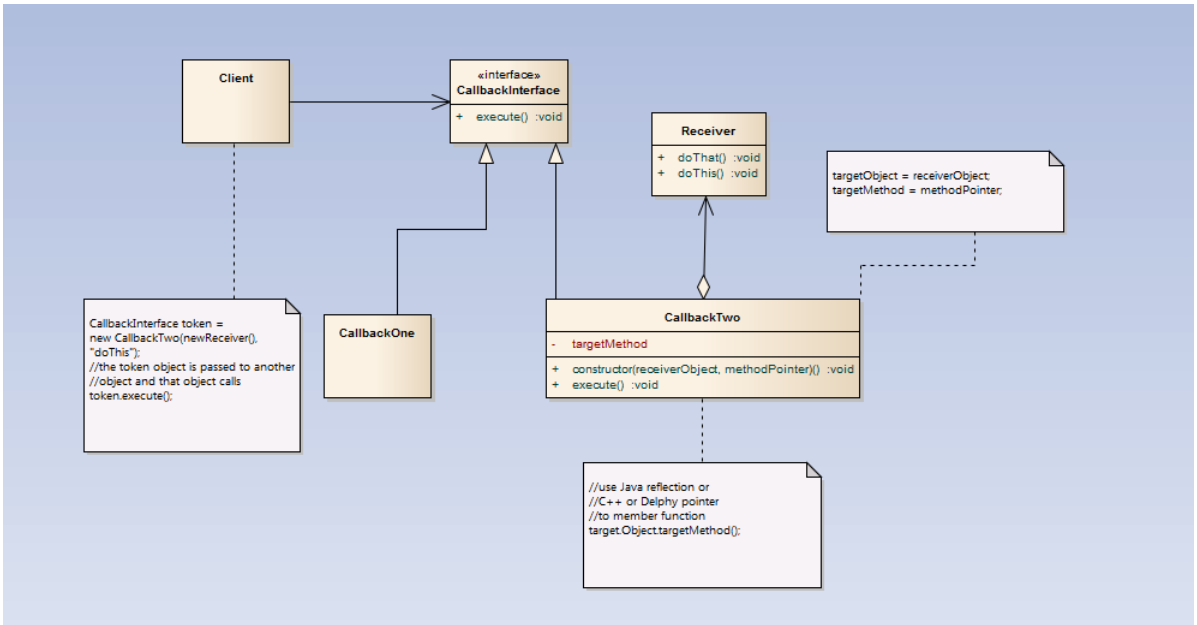

Figure 66: [71]

Intent

- Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations
- Promote "invocation of a method on an object" to full object status
- An object-oriented callback

[71]

Description

"Command decouples the object that invokes the operation from the one that knows how to perform it. To achieve this separation, the designer creates an abstract base class that maps a receiver (an object) with an action (a pointer to a member function). The base class contains an execute method that simply calls the action on the receiver. All clients of Command objects treat each object as a "black box" by simply invoking the object's virtual execute method whenever the client requires the object's "service".

A Command class holds some subset of the following: an object, a method to be applied to the object, and the arguments to be passed when the method is applied. The Command's "execute" method then causes the pieces to come together. Sequences of Command objects can be assembled into composite (or macro) commands."[71]

Practical usage in game development

One possible usage for the command pattern in game development is the way how user input is handled. A simple and fast implementation to handle the user input is to hardcode certain functions or actions to fixed buttons. This works as long as there is no need to configure how the buttons are mapped. Otherwise the pattern provides a nice solution to encapsulate the available buttons from their supposed function, by simple mapping the buttons to a command object. In this case, the button calls the execute() function of the command base class and doesn't know or care who the receiver is.

Handling actions within the game with commands opens up the option for the AI to use this too. Instead of hardwiring the AI to the actors, the system could simply send out commands to control an actor. "The decoupling here between the AI that selects commands and the actor code that performs them gives us a lot of flexibility. We can use different AI modules for different actors. Or we can mix and match AI for different kinds of behavior."[72]

Another useful and common application for the command pattern is the implementation of undo and redo. By storing commands in a list and extending the command object with a undo() function it is easy to implement this functionality.

3.6 Flyweight Pattern

The Flyweight pattern is one of the structural design patterns. It provides a solution for "designing objects down to the lowest levels of system "granularity"[...]" [71]

Intent

- Use sharing to support large numbers of fine-grained objects efficiently

[71]

Description

"The Flyweight pattern describes how to share objects to allow their use at fine granularities without prohibitive cost. Each "flyweight" object is divided into two pieces: the state-dependent (extrinsic) part, and the state-independent (intrinsic) part. Intrinsic state is stored (shared) in the Flyweight object. Extrinsic state is stored or computed by client objects, and passed to the Flyweight when its operations are invoked."[71] "Flyweights are stored in a Factory's repository. The client restrains herself from creating Flyweights directly, and requests them from the Factory. Each Flyweight cannot stand on its own. Any attributes that would make sharing impossible must be supplied by the client whenever a request is made of the Flyweight. If the context lends itself to "economy of scale" (i.e. the client can easily compute or look-up the necessary attributes), then the Flyweight pattern offers appropriate leverage." [71]

Practical usage in game development

A case where this pattern can be of use is, when a large amount of objects that use the same intrinsic values needs to be displayed. E.g. render a large forest. By gathering the intrinsic values in one spot, it is possible to pass this information to the GPU once and use it to render all similar objects in the scene, instead of passing it multiple times.

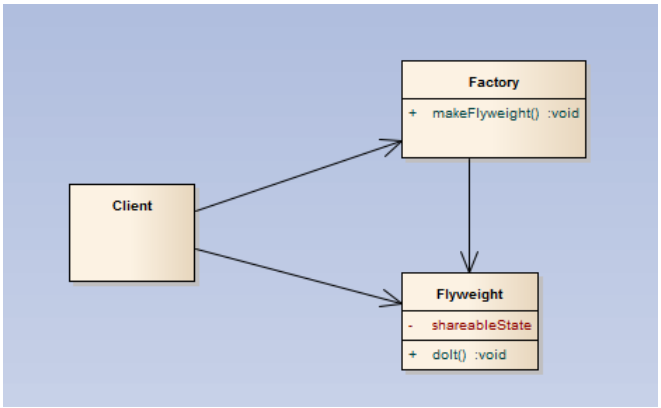

Figure 67: [71]

3.7 Observer Pattern

Also a behavior pattern and it allows components to notify other components without knowing which components are affected or how many there are.

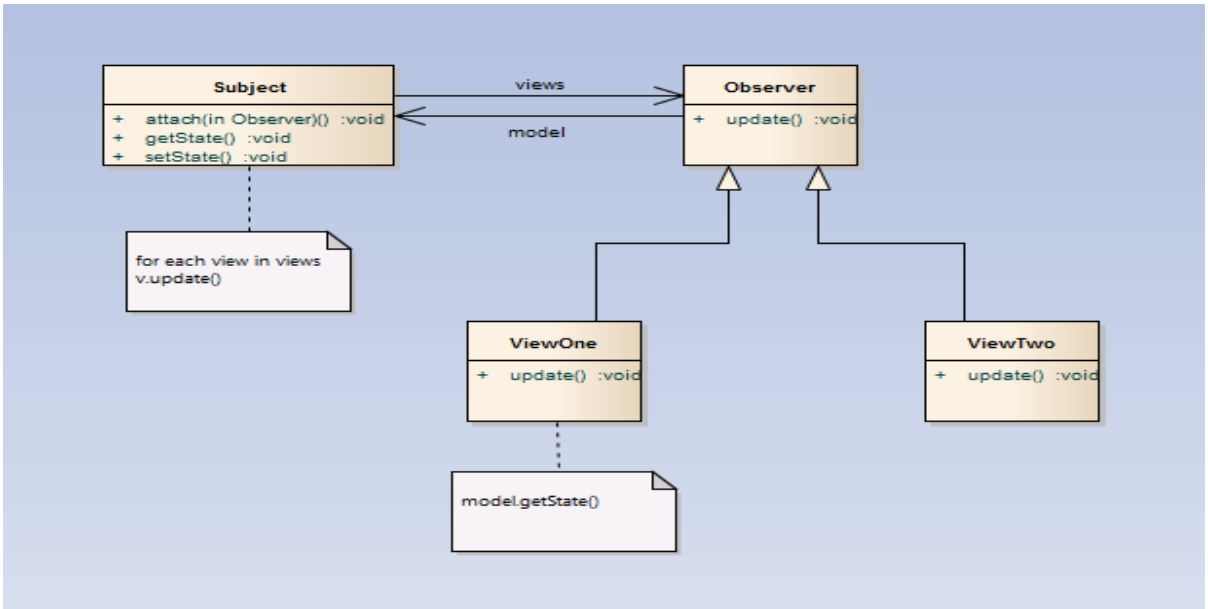

Figure 68: [71]

Intent

- Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Encapsulate the core (or common or engine) components in a Subject abstraction, and the variable (or optional or user interface) components in an Observer hierarchy.
- The "View" part of Model-View-Controller.

[71]

Description

“Observers register themselves with the Subject as they are created. Whenever the Subject changes, it broadcasts to all registered Observers that it has changed, and each Observer queries the Subject for that subset of the Subject’s state that it is responsible for monitoring. The protocol described above specifies a "pull" interaction model. Instead of the Subject "pushing" what has changed to all Observers, each Observer is responsible for "pulling" its particular "window of interest" from the Subject. The "push" model compromises reuse, while the "pull" model is less efficient. “[71]

Practical usage in game development

One way to use the observer pattern in a game is to implement an achievement system with it. Achievements can vary wildly in their complexity to unlock them, from simply completing a level to carrying out a series of specific task controlled by different parts of the system. For example, instead of writing checks for achievements directly into the physics code, we extend this part with the subject interface and turn it into a subject. Now the physics code only has to send a notification if something happened that might be of interest for the achievement system.

3.8 Decorator/Component Pattern

A structual pattern which allows to add and remove functionality to an object.

Intent

- You have a class that touches multiple domains which you want to keep decoupled from each other.
- A class is getting massive and hard to work with.
- You want to be able to define a variety of objects that share different capabilities, but using inheritance doesn't let you pick the parts you want to reuse precisely enough.
- Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

[71] [72]

Description

The Decorator or Component pattern encapsulates different domains and responsibilities which can be attached or removed dynamically. The concrete object with the desired functionality is simple a container, holding the components. By adding and removing components from the holding container, the behavior and functionality of an object can be altered. It allows building different objects through its flexibility to combine components in an arbitrary manner. Since the actual behavior for the created object is inside the components and is accessed through pointers, the added level of indirection could result in a poorer performance.

Practical usage in game development

A very good opportunity to use this pattern while developing a game is to use it for the backbone functionalities of a game. Physics, graphics, input reading, sound, collision detection, rendering etc. One or more of those parts are probably used by every game object inside a game. To avoid coupling and rewriting the same code parts over and over again, the component pattern is a good solution to this problem. An example of this pattern would be the Unity3D GameObject and its components. In this engine, all the major parts an object might need to use are separated into their own component which can be simply attached to the container to add its functionality.

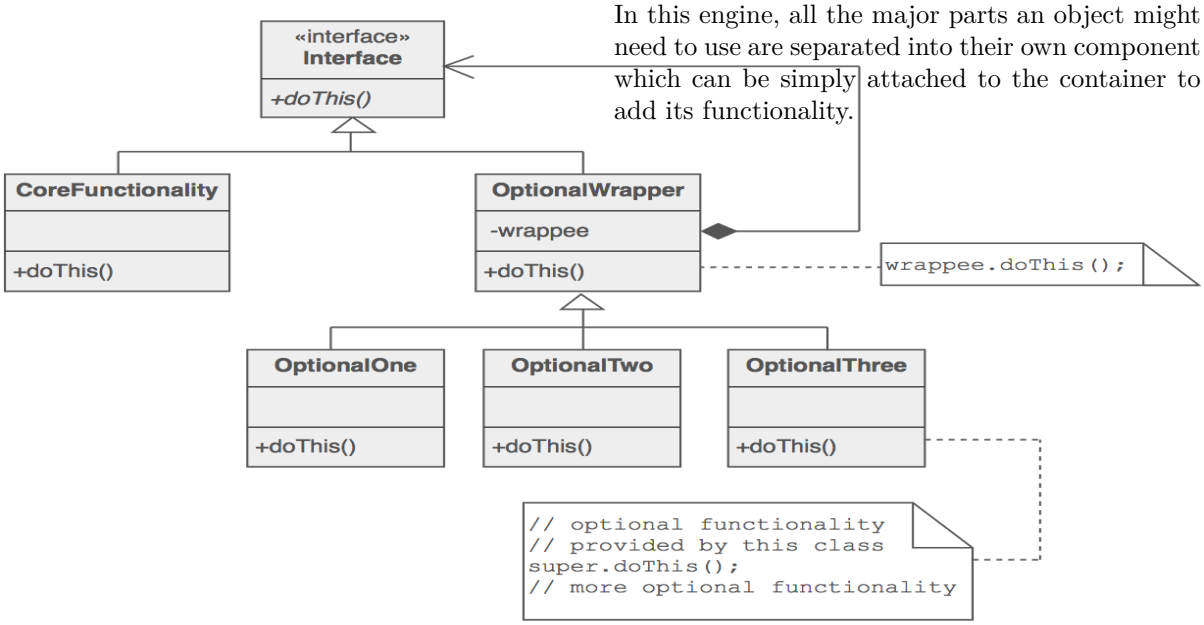

Figure 69: [71]

4 Evaluation of UML Tools

In this section, I will give you a little overview over some UML tools, I tested.

4.1 Enterprise Architect

The EA is the tool I got the most experience with. If you are just searching for anything to display your UML diagrams, take this one. It is relatively easy to learn and provides anything you need for all types of UML visualization. With the EA it is really easy to create UML diagrams in very short time. If you are more experienced with modelling, you can use it for model-driven source code generation as well. It works out for Java and C#. Further you can reuse once created classes in multiple diagrams. The project browser allows you to link classes several times and you can follow these links back to the origin. Also useful is the integrated UML compliance guard. If you for example try to connect two use-cases with an inheritance it wont let you do that. So it is also a good support for less experienced UML designers.

Criteria	Result
Name	Enterprise Architect
Version	10.0.1010
Vendor	Sparx Systems
License	Ultimate
Model Import/Export	XMI, CSV
Diagram Export	PDF
class diagram	yes
use-case diagram	yes
activity diagram	yes
state diagram	yes
component diagram	yes
requirements diagram	yes
Language	english
OS	Win7/8/10

Table 6: Test: Enterprise Architect

4.2 Astah

Astah is a very potent UML tool, with lots of available diagrams, export formats and customizations in form of plug-ins. Unfortunately it is not free, except for students. Astah is a fully fledged UML Tool. It supports the common UML diagrams and a lot of export types. Importing works via the XML format but it can export various image types, HTML, CSV, RTF and can also export to Java, C# and C++. Furthermore Astah can natively convert java files into diagrams. The vendor offers free plug-ins to further customize the application. For the diagram creation from code files, the vendor already offers the plug-ins for C# and C++.The User interface is simple and well structured, but it needs some time to really get used to it.

Criteria	Result
Name	Astah Professional
Version	7.0.0/846701
Vendor	Astah
License	Trial
Model Import/Export	XMI
Diagram Export	Image, XML, CSV, HTML
class diagram	yes
use-case diagram	yes
activity diagram	yes
state diagram	yes
component diagram	yes
requirements diagram	yes
Language	english, japanese
OS	Win7/8/10, Linux, MAC OS

Table 7: Test: Astah Professional

5 Model Driven Development

> ”models are used for designing systems, understanding them better, specifying required functionality, and creating documentation. Code is then written to implement the designs.”[73]

The application of models to software development is a well-known approach and has become even more popular with the introduction of the Unified Modeling Language (UML). In Model-Driven Software Development (MDSD or MDD) an entirely different approach is adopted with respect to the usage of models. MDSD is considered as the natural continuation of the current programming languages and software development methods. In MDSD models do not constitute documentation but are considered equal to code. MDSD aims to utilize domain-specific languages to create models that express application structure and behavior in a more efficient way. The models are then (semi)automatically transformed into executable code by model transformations.

MDD can be defined as a software development methodology with following characteristics:

1. MDD focuses on the models rather than the code, and the models are the major artifacts of the software development.
2. Models in MDD are formal thus can be transformed into the software automatically, or the models are executable.
3. Models are created with a modeling language at a higher abstraction level than the programming language.

Models in MDD can be created with either General Purpose Languages (GPLs), or Domain-Specific Languages (DSLs). UML is the most popular modeling GPL standardized by Object Management Group, which was developed to be able to model all kinds of application domains. Fig. 70 shows the concepts of MDD, and their relationships:

A typical example is the level model created with the level editor, which provides a visual environment for game world modeling. However, the game elements that engine tools can model are restricted to a narrow scope of game software, which are

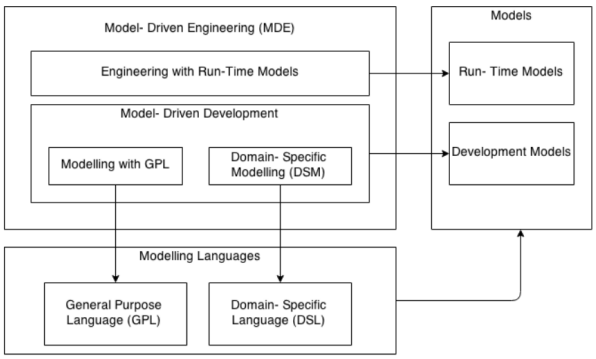

Figure 70: MDD concepts and their relationships

mainly assets like level data and presentation data (Fig. 71). A lot more elements such as AI, control, rules still have to be handcoded, either with a native language or with a scripting language. Some state-of-art game engines do provided visual modeling tools for creating script code, for example, Unreal Kismet, but these tools are not taking full advantages of MDD, such as use of meta-models and language workbenches, which makes it difficult to be adapted to a new game domain [74].

An essential value of MDD is the raised abstraction level, which allows specifying solution with problem domain concepts. To be successful in practice, MDD approaches must find the balance among the abstraction level, the complexity of the language, and the broadness of the domain [73].

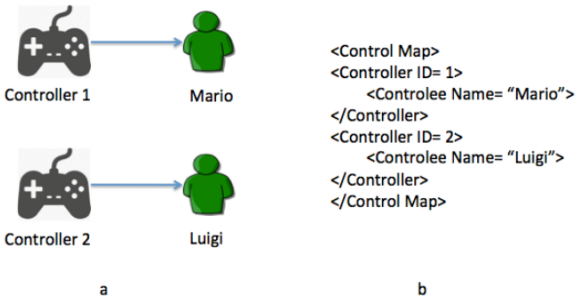

Figure 71: Graphical Syntax V.S. Textual Syntax

5.1 The Goals of MDSD

1. Runnable code can be generated from formal models using one or more transformation steps.
2. The use of automated transformations and formally-defined modeling languages lets you enhance software quality.
3. Better maintainability of software systems through redundancy avoidance and manageability of technological changes.
4. Architectures, modeling languages and transformations can be used in the sense of a software production line for the manufacture of diverse software systems. This leads to a higher level of reusability and makes expert knowledge widely available in software form.
5. Another significant potential is the improved manageability of complexity through abstraction.
6. MDSD offers a productive environment in the technology, engineering, and management fields through its use of process building blocks and best practices.

5.2 Basic Concept

Domain-related specifications are defined in Platform-Independent Models (PIMs). A formal modeling language is used that is specific to the concepts of the domain to be modeled. In most cases, one would use UML that has been adapted via profiles to the respective domain, not least because of its tool support. These domain-specific descriptions are completely independent of the later implementation on the target platform. Fig. 72 illustrates this basic principle.

Via model transformation, usually automated with tools, Platform-Specific Models (PSMs) are created from the Platform-Independent Models. These Platform-Specific Models contain the target platform's specific concepts. The implementation for a concrete target platform is then generated with another tool-supported transformation based on one or more PSMs.

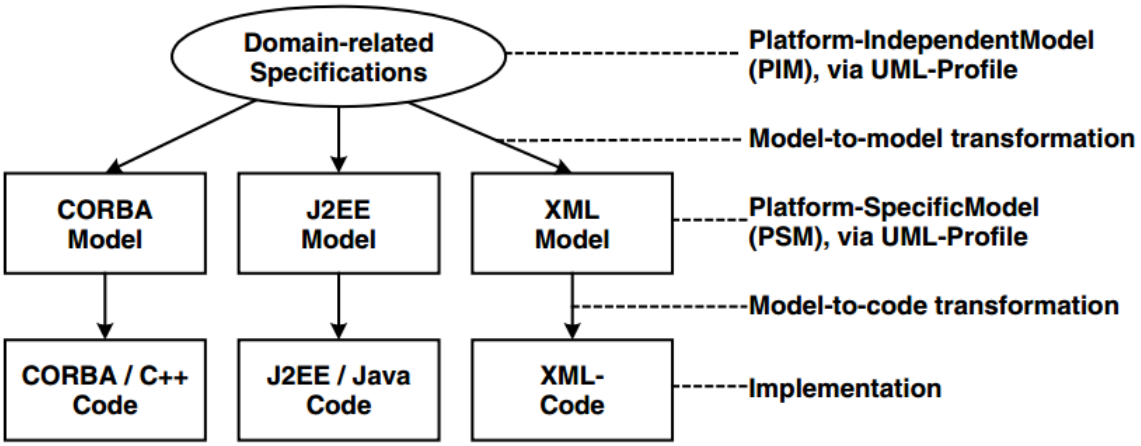

Figure 72: Basic concept of MDA Transformation

Part VI
Style Guide

Sebastian Szuzskiewicz and Sascha Kalab

Keywords — ***Style, C++, Clean Code, Naming Conventions***

1 Why Style Guides

A style guide is a set of standards for the writing and design of documents, either for general use or for a specific publication, organization, or field. A style guide establishes and enforces style to improve communication.

In the case of Computer Science, style guides help programmers to communicate with each other. If someone wants to use a specific library or function for example they need to understand what they want to include in their project. To accomplish this, style guides give programmers directive they can use to unify code.

But why do we want to do that?

An example story from Robert C. Martins: I know of one company that, in the late 80s, wrote a killer app. It was very popular, and lots of professionals bought and used it. But then the release cycles began to stretch. Bugs were not repaired from one release to the next. Load times grew and crashes increased. I remember the day I shut the product down in frustration and never used it again. The company went out of business a short time after that. Two decades later I met one of the early employees of that company and asked him what had happened. The answer confirmed my fears. They had rushed the product to market and had made a huge mess in the code. As they added more and more features, the code got worse and worse until they simply could not manage it any longer. It was the bad code that brought the company down. [75] This story shows that just bad written code brought down a company with huge potential. So the conclusion why code styles and style guides are needed is not only they help to communicate with other programmers but also you can spare time producing and maintaining code just with a good structured code style and codebase where everyone knows how to read through it.

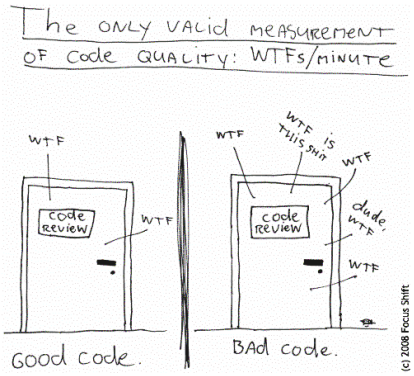

Figure 73: `http://www.osnews.com/images/comics/wtfm.jpg`

As part of writing good code it has to be kept clean over time. Starting with the best intentions and messing up over periods of time still releases bad code. In order to accomplish this the coding style has to be used and respected all the time.

2 Clean Code Style

2.1 Meaningful Names

Programmers name various kinds of things like variables, functions or arguments. Naming is a huge part of good code, because every programmer has to do so much of it and because of that they should do it well.

2.2 Intention-Revealing Names

Choosing good names takes time but saves more than it takes. The name of a variable, function, or class, should answer all the big questions. It should tell you why it exists, what it does, and how it is used. If a name requires a comment, then the name does not reveal its intent.

```
int d; // elapsed time in days
```

The name d reveals nothing. It does not evoke a sense of elapsed time, nor of days. Names should be chosen to specify what is being measured and the unit of that measurement:

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

2.3 Avoiding Disinformation

Programmers must avoid leaving false clues that obscure the meaning of code. Words whose entrenched meanings vary from our intended meaning should be avoided. For example, hp, aix, and sco would be poor variable names because they are the names of UNIX platforms or variants. Even if hypotenuse and hp looks like a good abbreviation for hypotenuse, it could be disinformative. [75]

Refering to a grouping of accounts as an accountList is misinforming unless it is actually a List. The word list means something specific to programmers. If the container holding the accounts is not actually a List, it may lead to false conclusions. So accountGroup or bunchOfAccounts or just plain accounts would be better. [75]

Using names which vary in small ways can also conclude in wrong information. How long does it take to spot the subtle difference between a XYZControllerForEfficientHandlingOfStrings in one module and, somewhere a little more distant, XYZControllerForEfficientStorageOfStrings? The words have frightfully similar shapes. [75]

2.4 Pronounceable Names

Humans are good at words. A significant part of brains is dedicated to the concept of words. And words are, by definition, pronounceable. It would be a shame not to take advantage of that huge portion of our brains that has evolved to deal with spoken language. So make your names pronounceable. [75] Example from Robert C. Martins: A company I know has genymdhms (generation date, year, month, day, hour, minute, and second) so they walked around saying "gen why emm dee aich emm ess". I have an annoying habit of pronouncing everything as written, so I started saying "gen-yah-muddahims." It later was being called this by a host of designers and analysts, and we still sounded silly. But we were in on the joke, so it was fun. Fun or not, we were tolerating poor naming. New developers had to have the variables explained to them, and then they spoke about it in silly made-up words instead of using proper English terms. Compare

```
class DtaRcrd102
{
    private Date genymdhms;
    private Date modymdhms;
    private final String pszqint = "↩
        102";
    /* ... */
};

to

class Customer
{
    private Date generationTimestamp;
    private Date ↩
        modificationTimestamp;;
    private final String recordId = "↩
        102";
    /* ... */
};
```

Intelligent conversation is now possible: "Hey, Mikey, take a look at this record! The generation timestamp is set to tomorrow's date! How can that be?"

2.5 Searchable Names

Single-letter names and numeric constants have a particular problem in that they are not easy to locate across a body of text. One might easily grep for MAX_CLASSES_PER_STUDENT, but the number 7 could be more troublesome. Searches may turn up the digit as part of file names, other constant definitions, and in various expressions where the value is used with different intent. It is even worse when a constant is a long number and someone might have transposed digits, thereby creating a bug while simultaneously evading the programmer's search. Likewise, the name e is a poor choice for any variable for which a programmer might need to search. It is the most common letter in the English language and likely to show up in every passage of text in every program. In this regard, longer names trump shorter names, and any searchable name trumps a constant in code. [75]

3 PascalCase and camelCase

CamelCase is the practice of writing compound words or phrases such that each word or abbreviation begins with a capital letter. Camel case may start with a capital or, especially in programming languages, with a lowercase letter. Common examples are LibreOffice, PowerPoint, iPhone or in online usernames such as UserName. In Microsoft documentation, camel case always starts with a lower case letter (e.g. backColor), and it is contrasted with PascalCase which always begins with a capital letter (e.g. BackColor). The use of medial caps for compound identifiers is recommended by the coding style guidelines of many organizations or software projects. For some languages (such as Mesa, Pascal, Modula, Java and Microsoft's .NET) this practice is recommended by the language developers or by authoritative manuals and has therefore become part of the language's "culture". Style guidelines often distinguish between upper and lower camel case, typically specifying which variety should be used for specific kinds of entities: variables, record fields, methods, procedures, types, etc. These rules are sometimes supported by static analysis tools that check source code for adherence. The original Hungarian notation for programming, for example, specifies that a lowercase abbreviation for the "usage type" (not data type) should prefix all variable names, with the remainder of the name in upper camel case; as such it is a form of lower camel case. Programming identifiers often need to contain acronyms and initialisms that are already in upper case, such as "old HTML file". By analogy with the title case rules, the natural camel case rendering would have the abbreviation all in upper case, namely "oldHTMLFile". However, this approach is problematic when two acronyms occur together (e.g., "parse DBM XML" would become "parseDBMXML") or when the standard mandates lower camel case but the name begins with an abbreviation (e.g. "SQL server" would become "sQLServer"). For this reason, some programmers prefer to treat abbreviations as if they were lower case words and write "oldHtmlFile", "parseDbmXml" or "sqlServer". A proper definition of the usage of Camel- and PascalCase is given in the following chapters. `https://en.wikipedia.org/wiki/CamelCase`

4 How to write comments

Comments are an essential part of your code as they aid the reader in understanding it. This does not mean comments are a cure for poorly written code in terms of formatting, naming and style. They should be aimed to assist clarifying certain parts, e.g. complex operations or the meaning of hard coded integer values. Writing good comments is a bit of an art itself and the rule of thumb is "Quality over quantity".

The idea is to write as less comments as possible with the best result possible, meaning that code should be self-explanatory as far as the language allows to. Also, comments should explain the "why" rather than the "what" because the ability of reading code implicitly clarifies the "what" most of the time. Another important thing to keep in mind is that comments cannot replace code, so writing a comment that states a certain header file needs to be included or the value of a variable should be in a certain range is not a good habit. This is the responsibility of code and should not be in a comment.

At best a comment shortly describes something and does not distract the reader from the code. It is important to keep comments from being ambiguous and they need to always be updated when the code their assigned to is. The worst thing comments can do to a reader, which could be the author of a piece of code himself, is that they do not tell the truth anymore or just repeat what the code can say itself.

5 Formatting

Formatting makes code readable to the human eye. A compiler does not care about that since it only processes bytes of information. Keeping code in a nice and readable format is one of the most important things to do besides making it work correctly.

> "You should take care that your code is nicely formatted. You should choose a set of simple rules that govern the format of your code, and then you should consistently apply those rules." [75]

Good and consistent formatting is more important than just getting code to work. It makes code more maintainable as well as extensible and helps keeping track of the big picture. When working in a team, tools should be used with common rulesets for all team members. Such tools and rulesets will be discussed in later sections. Main parts of formatting are:

1. Indentation
2. Bracketing
3. Ordering of Sections
4. Placement of Blanks

5.1 Indentation

Indentation is about horizontal placement of code and where a line starts. It helps in keeping track of which section a line of code belongs to and makes reading a lot easier. Wrong indentation is disastrous for reading and can easily lead to misunderstandings. Whether indentation is achieved through blanks or tabulators is not that important as to keep it consistent. Furthermore it is a matter of taste whether the body of a section should be indented or kept in the same vertical line as the beginning, e.g. the body of a switch-case statement.

5.2 Bracketing

There is a lot that could be said about setting brackets and what is too much. First of all, there is more to it than just where to set the opening bracket of a function or if-statement body. For example, most Java programmers prefer to place the opening bracket in the same line as the function header whereas most C# programmers will set it in the next line. Truth is, overusing of brackets can make code unreadable for humans but not for machines. Similar to comments the compiler has no problem resolving and respecting all used brackets as long as they are syntactically correct. A human on the other hand is easily overwhelmed even with a small number of brackets. Imagine the following line as an example:

```
int a = (b + (c * d));
```

There is actually no need of any brackets at all in this line of code. The operator precedence will make sure that the compiler knows what to do in this case, still the brackets are not wrong syntactically. In some situations though, it is necessary to set brackets to ensure the wanted behaviour. As barley any programmer knows the exact order of the operator precedence table brackets are useful but they should be kept to a minimum.

5.3 Ordering of Sections

Ordering is actually not just about sections but also about functions. Functions should be grouped in a way so that is easy to find functions that belong and are used together. For sections it has the meaning of sections in a class header layout, if the programming language uses that concept. Member variables of a class should be grouped together as well as so called "getter" and "setter" functions. It is common practise to keep member variables either at the top or the bottom of a class definition.

Either way, constructors and operators should be first followed by possible other functions.

5.4 Placement of Blanks

This topic is bit about nit-picking since it does not improve readability of code this much. Still it something to discuss as it is part of a formatting rule set. Besides a syntactic rules of the used language, blanks should be placed where it helps making the code more readable, e.g. to indicate operator precedence without using brackets. Generally it is "Quality over quantity" again.

6 Don'ts of Code Style

The following section describes some of the most done mistakes in code styling that affect readability and maintainability of code. Some of these "Don'ts" may have funny names but horrendous impacts.

6.1 Yoda Conditions

Seen in the code of some programmers the so called "Yoda Conditions", which got their name from the famous Star Wars character, are over read easily but leave the feeling that something is wrong with the code. They appear when a programmer writes an if-statement and puts the constant before the variable:

```
if (1 == myInt)
```

The correct way to do this is to simply put the variable before the constant which makes a lot more sense and prevents a possible read from scratching his head.

6.2 Fake Comments

When comments without any useful information are placed somewhere in code just with the purpose of silencing a tool like "FxCop" they are wrong. The right way is to actually give information about why a piece of code was written.

6.3 Monolithic Functions and schizophrenic Classes

Functions that are way too long and complex usually do too much work on their own. They are monolithic and combine logic which should actually be split up into several functions to make the code reusable, more readable and most important more maintainable.

The term "schizophrenic class" indicates that a class has more than one purpose which is plain wrong and should be avoided. One class has only one purpose and consequently ignoring this paradigm can lead to various problems with more or less impact on code quality.

7 Code Style Guide Example for C++

In the following a style guide for C++ code is presented. It is meant to be a guideline to keep in mind when writing code. These guidelines are mainly focused on C++ code in general but certain points (e.g. exception handling) address game related code. Further reading: `https://google.github.io/styleguide/cppguide.html`

7.1 Naming Conventions

Classes / Interfaces / Base Classes

Class names in general, as well as interfaces or base classes, should be written in PascalCase style and should be a noun or a combination of nouns.

Example: WindowController

Interfaces, which have the condition not to contain any members but functions, should start with the letter "I" followed by one or more nouns.

Example: IObserver

Base classes should be named like normal classes with the extension ”Base” at the end.

Example: BehaviourBase

Naming classes in that scheme makes it obvious to other programmers what kind of class they are dealing with and how to use it. Additionally it makes searching for a specific type of class a lot easier.

Variables

Variables do not have a general rule applying to all of them. They have to be differentiated depending on the scope they exist in. Global scope is not listed below as global variables should be avoided but in case a global variable is necessary or of specific benefit it should be written in either PascalCase or camelCase and prefixed with ”g_”.

Example: g_Count

Local Variables / Parameters

Variables local to a function as well as parameter variables should be written in camelCase without any prefix at all.

Example: distanceToGoal

Member Variables

Variables in class scope (member variables) should be written in either PascalCase or camelCase but either way prefixed with ”m_”.

Example: m_Position – or – m_orientation

Functions

Function names in general, independent of their scope, should simply be written in PascalCase and should usually consist of verbs.

Example: CalculateSteering

Getter / Setter Functions

So called ”Getter” and ”Setter” functions are used to get access to member variables that are not directly accessible. They should be named similar to the member variable they are assigned to but with the prefix ”Get” or ”Set”. That way it is easy to see which data of a class can be retrieved and which can be set.

Example: GetPosition – or - SetOrientation

Templates

This section only applies to type template parameters. Since C++ allows two keywords to declare a type template parameter it should be differentiated between a parameter which is meant to be a simple type and a parameter which is meant to be a custom type. For simply types the template parameter should be declared using ”typename” and for a complex type ”class” should be used.

Example: template<typename T, class U> Select(...)

7.2 General Rules

Include Guard

Every header file has to have a so called ”include guard” which usually is a pre-processor statement in the very first line(s). Most modern compilers support the ”#pragma once” command but for compatibility and standard conformity reasons an ”#ifndef” or ”#if” command should also be used. Notice that the last two command need to be closed at the very end of the header file placing a ”#endif” command. Example:

```
#pragma once
#ifndef HEADER_FILE_NAME_H_
<code>
#endif // HEADER_FILE_NAME_H_
```

Exceptions

In game related code exceptions are a bad practise and should not be used at all. The reason for that is they are slow and not very well implemented in C++.

Bracket Placement

In general brackets should be used even when they can be left out without creating syntactic errors. For example an if-statement with only on line in its scope. It makes code much more readable and avoids painful errors.

On the other hand parenthesis should not be overused but every time it supports readability. Common sense is helpful for this kind of brackets.

8 CppCheck

8.1 Description

Cppcheck is a static analysis tool for C/C++ code. Unlike C/C++ compilers and many other analysis tools it does not detect syntax errors in the code. Cppcheck primarily detects the types of bugs that the compilers normally do not detect. The goal is to detect only real errors in the code (i.e. have zero false positives).

8.2 Integration in Development Tools

Cppcheck is integrated with many popular development tools. For instance:

1. Clion - Cppcheck plugin
2. Code::Blocks - integrated
3. CodeDX (software assurance tool) - integrated
4. CodeLite - integrated
5. CppDepend 5 - integrated
6. Eclipse - Cppcheclipse
7. gedit - gedit plugin
8. Hudson - Cppcheck Plugin
9. Jenkins - Cppcheck Plugin
10. Mercurial (Linux) - pre-commit hook - Check for new errors on commit (requires interactive terminal)
11. Tortoise SVN - Adding a pre-commit hook script
12. Git (Linux) - pre-commit hook - Check for errors in files going into commit (requires interactive terminal)
13. Visual Studio - Visual Studio plugin

8.3 Features

1. Out of bounds checking
2. Memory leaks checking
3. Detect possible null pointer dereferences
4. Check for uninitialized variables
5. Check for invalid usage of STL
6. Checking exception safety
7. Warn if obsolete or unsafe functions are used
8. Warn about unused or redundant code
9. Detect various suspicious code indicating bugs

For a full description of all features look at: `http://sourceforge.net/p/cppcheck/wiki/ListOfChecks` Both command line interface and graphical user interface are available.

8.4 Usage

As with many analysis programs, there are many unusual cases of programming idioms which may be acceptable in particular target cases, or outside of the programmer's scope for source code correction. A study conducted in March 2009 identified several areas where false positives were found by cppcheck, but did not specify the program version examined. Cppcheck has been identified for use in systems such as CERNs 4DSOFT meta analysis package, for code verification in high energy particle detector readout devices, system monitoring software for radio telescopes as well as in error analysis of large projects, such as OpenOffice.org and the Debian archive.

The idioms and terms, as described in this styleguide or from various other resources, need to be entered in cppcheck.

8.5 Where to get from?

Cppcheck can either be downloaded directly from: `http://sourceforge.net/projects/cppcheck/` or integrated in various development tools via specific options (see Features).

8.6 Support

The IRC channel can be access with a web browser: `http://webchat.freenode.net` Forum: `http://sourceforge.net/p/cppcheck/discussion/`

References

[1] Thomas M. Pigoski. Software maintenance. May 2001. URL: http://sce.uhcl.edu/helm/SWEBOK_IEEE/data/swebok_chapter_06.pdf.

[2] Vianney Cote and Denise St-Pierre. A model for estimating perfective software maintenance projects. pages 328–334, 1990. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=131382.

[3] Anthony Jarman. Testing and maintenance in the video game industry today. 2010. URL: http://digitalcommons.liberty.edu/cgi/viewcontent.cgi?article=1136&context=honors.

[4] Software Engineering Standards Committee et al. Ieee standard for software maintenance. *IEEE Std*, page 5, 1998. URL: http://www.cs.uah.edu/~rcoleman/CS499/CourseTopics/IEEE_Std_1219-1998.pdf.

[5] Harry M. Sneed, Martin Hasitschka, and Maria-Therese Teichmann. *Software-Produktmanagement: Wartung und Weiterentwicklung bestehender Anwendungssysteme*. dpunkt-Verlag, 1 edition, 2005.

[6] Michael E Moore and Jeannie Novak. *Game development essentials: Game industry career guide*. Delmar Learning, 2009. URL: http://dl.acm.org/citation.cfm?id=1816548.

[7] Felix Seibert. Interview with felix seibert about community feedback in asterix and friends. personal communication, 2016.

[8] BCS. Software maintenance - an overview. *bcs - The Chartered Institute for IT*, 2015. URL: http://www.bcs.org/content/ConWebDoc/3063.

[9] Chris Butcher. Destiny: Six years in the making - gdc 2015. 2015. URL: https://www.youtube.com/watch?v=UU92a5pKV6k.

[10] Raphael van Lierop. The long dark & community influenced development. 2015. URL: https://www.youtube.com/watch?v=xeCN8JJ7L9M.

[11] Sproing-Publishing-GmbH. Asterix and friends faq. 2014. URL: https://www.asterix-friends.com/de/faq/.

[12] Sproing-Interactive-Media-GmbH. Sproing website - asterix and friends. 2015. URL: https://www.sproing.com/game/asterix-friends/.

[13] Ryan Smith. Valve to Showcase SteamVR Hardware, Steam Machines, & More at GDC 2015, February 2015. URL: http://www.anandtech.com/show/9003/valve-to-showcase-steamvr-hardware-steam-machines-more-at-gdc-2015.

[14] Valve. Steam Greenlight, 2016. URL: https://steamcommunity.com/sharedfiles/editgame/?appid=765.

[15] Valve. STEAMWORKS, 2016. URL: https://www.steampowered.com/steamworks/.

[16] GOG. GOG.com, 2016. URL: https://www.gog.com/indie.

[17] Ben Kuchera. Penny Arcade - How the Humble Store may challenge Steam the business of helping indie developers, January 2013. URL: https://www.penny-arcade.com/report/article/how-the-humble-store-may-challenge-steam-the-business-of-helping-indie.

[18] Humble Bundle. Humble Bundle Developer Resources, 2016. URL: https://www.humblebundle.com/developer.

[19] Humble Bundle. The Humble Widget, 2016. URL: https://www.humblebundle.com/developer/widget.

[20] Microsoft. Publish Windows apps - Windows app development, 2016. URL: https://dev.windows.com/en-us/publish.

[21] Steve Ranger. iOS versus Android. Apple App Store versus Google Play: Here comes the next battle in the app wars, January 2015. URL: http://www.zdnet.com/article/ios-versus-android-apple-app-store-versus-google-play-here-comes-the-next-battle-in-the-app-war

[22] Nathan Ingraham. Apple's App Store has passed 100 billion app downloads, June 2015. URL: http://www.theverge.com/2015/6/8/8739611/apple-wwdc-2015-stats-update.

[23] Apple. App Store Review Guidelines, 2016. URL: https://developer.apple.com/app-store/review/guidelines/.

[24] Apple. Choosing a Membership, 2016. URL: https://developer.apple.com/support/compare-memberships/.

[25] Apple. Apple Developer Program, 2016. URL: https://developer.apple.com/programs/.

[26] Paul Tassi. 'Fallout Shelter' Brings In $5.1m For Bethesda, But Is About To Hit A Wall, July 2015. URL: http://www.forbes.com/sites/insertcoin/2015/07/17/fallout-shelter-brings-in-5-1m-for-bethesda-but-is-about-to-hit-a-wall/.

[27] Rebecca Borison. Blek Founder Tells Us How He Got To No.1 In The App Store For 20 Straight Days, May 2014. URL: http://www.businessinsider.com/blek-cofounder-explains-the-apps-success-2014-5.

[28] Apple. Game Center for Developers, 2016. URL: https://developer.apple.com/game-center/.

[29] Apple. GameKit Framework Reference, 2016. URL: https://developer.apple.com/library/ios/documentation/GameKit/Reference/GameKit_Collection/.

[30] OpenSignal.com. Android Fragmentation Report August 2015, 2015. URL: http://opensignal.com/reports/2015/08/android-fragmentation/.

[31] Emil Protalinski. App Annie 2015: Google Play saw 100% more downloads than the iOS App Store, but Apple generated 75% more revenue, January 2016. URL: http://venturebeat.com/2016/01/20/app-annie-2015-google-play-saw-100-more-downloads-than-the-ios-app-store-but-apple-generated-75

[32] Sarah Perez. App Submissions On Google Play Now Reviewed By Staff, Will Include Age-Based Ratings, March 2015. URL: http://social.techcrunch.com/2015/03/17/app-submissions-on-google-play-now-reviewed-by-staff-will-include-age-based-ratings/.

[33] Greg Hartrell. Unlocking the Power of Google for Your Games, at GDC | Android Developers Blog, March 2014. URL: http://android-developers.blogspot.co.at/2014/03/unlocking-power-of-google-for-your_17.html.

[34] Google. Play Games Services, 2016. URL: https://developers.google.com/games/services/.

[35] Sarah Perez. Android Users Can Now Record And Publish Their Video Gameplay From The Google Play Games App, October 2015. URL: http://social.techcrunch.com/2015/10/28/android-users-can-now-record-publish-their-video-gameplay-from-the-google-play-games-app/.

[36] Ben Fox Rubin. Amazon Appstore nears 400k apps on 'huge progress', March 2015. URL: http://www.cnet.com/news/amazon-appstore-nears-400k-apps-on-huge-progress/.

[37] IGDA. Amazon's clarification fails to address game developer concerns, April 2011. URL: https://igdaboard.wordpress.com/2011/04/19/amazon%e2%80%99s-clarification-fails-to-address-game-developer-concerns/.

[38] SlideME. About SlideME, 2016. URL: http://slideme.org/about-slideme.

[39] Richard Hill-Whittall. *The Indie Game Developer Handbook*. Focal Press, Burlington, MA, February 2015.

[40] OUYA. OUYA: A New Kind of Video Game Console, 2016. URL: https://www.kickstarter.com/projects/ouya/ouya-a-new-kind-of-video-game-console.

[41] Razer. Razer Developers, 2016. URL: https://gamers.ouya.tv/developers.

[42] NVidia. Next-Generation Android Games on NVIDIA SHIELD, 2016. URL: http://shield.nvidia.com/games/android.

[43] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. Gprof: A Call Graph Execution Profiler. In *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*, SIGPLAN '82, pages 120–126, New York, NY, USA, 1982. ACM. URL: http://doi.acm.org/10.1145/800230.806987, doi:10.1145/800230.806987.

[44] Amitabh Srivastava and Alan Eustace. ATOM: A System for Building Customized Program Analysis Tools. *SIGPLAN Not.*, 39(4):528–539, April 2004. URL: http://doi.acm.org/10.1145/989393.989446, doi:10.1145/989393.989446.

[45] Microsoft. Beginners Guide to Performance Profiling, 2015. URL: https://msdn.microsoft.com/en-us/library/ms182372.aspx.

[46] Microsoft. How to: Choose Collection Methods, 2015. URL: https://msdn.microsoft.com/en-us/library/ms182374(v=vs.140).aspx.

[47] Microsoft. Understanding Profiling Methods, 2015. URL: https://msdn.microsoft.com/en-us/library/dd264994.aspx.

[48] Unreal. Performance and Profiling, 2016. URL: https://docs.unrealengine.com/latest/INT/Engine/Performance/.

[49] Unreal. Profiler, 2016. URL: https://docs.unrealengine.com/latest/INT/Engine/Performance/Profiler/index.html.

[50] Unreal. GPU Profiling, 2016. URL: https://docs.unrealengine.com/latest/INT/Engine/Performance/GPU/.

[51] Nvidia. NVIDIA Nsight Visual Studio Edition User Guide - NVIDIA Nsight Visual Studio Edition 5.0 User Guide, 2015. URL: http://docs.nvidia.com/nsight-visual-studio-edition/5.0/Nsight_Visual_Studio_Edition_User_Guide.htm.

[52] AMD. CodeXL - powerful Debugging, Profiling and analysis, 2015. URL: http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/.

[53] AMD. CodeXL benefits in detail, 2015. URL: http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/codexl-benefits-detail/.

[54] Gerard Meszaros. *XUnit Test Patterns: Refactoring Test Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[55] Niraj bhatt – architect's blog. https://nirajrules.wordpress.com/2011/08/27/dummy-vs-stub-vs-spy-vs-fake-vs-mock/ (2011-08-27).

[56] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008.

[57] Noel Llopis. Backwards Is Forward: Making Better Games with Test-Driven Development. `http://gamesfromwithin.com/backwards-is-forward-making-better-games-with-test-driven-development` (2006-03-12).

[58] Francesco Carucci. AAA Automated Testing for AAA Games. `http://www.crytek.com/cryengine/cryengine3/presentations/aaa-automated-testing-for-aaa-games` (2009-08-16).

[59] Dag Frommhold Fabian Röken. Automated Tests and Continuous Integration in Game Projects. `http://www.gamasutra.com/view/feature/2269/automated_tests_and_continuous_.php` (2005-03-29).

[60] Ville-Veikko Helppi. The Agile Process for Mobile Game Development and Testing. `http://testdroid.com/tech/the-agile-process-for-mobile-game-development-and-testing` (2015-01-14).

[61] James A Whittaker, Jason Arbon, and Jeff Carollo. *How Google tests software*. Addison-Wesley, 2012.

[62] Unity QA: Test Tools. `https://unity3d.com/unity/qa/test-tools`.

[63] Unity Test Tools Wiki. `https://bitbucket.org/Unity-Technologies/unitytesttools/wiki/` (2015-12-14).

[64] Philippe Kruchten. Architectural blueprints—the "4+ 1" view model of software architecture. *Tutorial Proceedings of Tri-Ada*, 95:540–555, 1995.

[65] Wolfgang Pfeifer and Wilhelm Braun. *Etymologisches wörterbuch des deutschen*. Akademie-Verlag, 1989.

[66] Ian Sommerville. *Software Engineering*. Pearson, 2011.

[67] Gernot Starke. *Effektive Software-Architekturen*. Hanser Verlag, 2011.

[68] Gernot Starke, Andreas Rausch, Arne Koschel, and Mahbouba Gharbi. *Basiswissen für Softwarearchitekten*. dpunkt, 2013.

[69] uml diagrams.org. Uml 2.5 diagrams overview, nov 2015. URL: `http://www.uml-diagrams.org/uml-25-diagrams.html`.

[70] uml diagrams.org. Uml profile diagrams, nov 2015. URL: `http://www.uml-diagrams.org/profile-diagrams.html`.

[71] Alexander Shvets. *Design Patterns Explained Simply*. Sourcemaking.com, 2014.

[72] Robert Nystrom. *Game programming patterns*. Genever Benning, 2014.

[73] Krzysztof Czarnecki Thomas Stahl, Markus Voelter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 1 edition, 2006.

[74] Zhu Meng. *Model-Driven Game Development Addressing Architectural Diversity and Game Engine-Integration*, volume 95. Norwegian University of Science and Technology, 2014.

[75] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 1 edition, 2008.