An Overview of Game Testing Techniques

Claudio Redavid
GSEEM Student at Mälardalen University
Västerås, Sweden
crd11001@student.mdh.se

Adil Farid
GSEEM Student at Mälardalen University
Västerås, Sweden
afd11003@student.mdh.se

ABSTRACT

The purpose of this paper is to analyze in general the process of testing, developing video games, and describe in more detail all the aspects of the test phases in a game development process. The game industry has grown significantly in the last decade and many companies have started to use software engineering techniques in the game development process. In game development process the development team is divided into small teams that manage separate aspects of a game. All developing processes and products are subjected to test, for validation and verification. Modern software testing techniques can be easily used in game development as well, however it is not easy to test a game due to its complexity, but customers and managers are putting hard constraints on a game quality more and more frequently. Effective testing techniques can reduce the testing time and improve game quality and production time.

Keywords

Software Engineering, VideoGame, Software testing, Alpha and Beta test, Game prototype, Game testing techniques, Game test phases

1. INTRODUCTION

Software testing can be stated as the process of validating and verifying software products, which meets the requirements that guided its design and development, works as expected, and can be implemented with the same characteristics [26]. Software Testing is one of the most important phases of the software development process. As shown in Figure 1 software engineering paradigm has evolve significantly from procedural code to object oriented software, to component based software; from thousand to million lines of code with reliability, security, scalability, real time, safety and performance requirements [19]. With the improvement in complexity of software development process, testing is getting more and more complex and critical. The simple testing techniques used for procedural code were need to be extended for object oriented coding techniques like polymorphism, binding and other new object oriented features. New testing techniques were introduced for component based programming. In traditional procedural coding, testing was the last step of the software development process (i.e. test cases are selected based on the source code), because of which different factors like schedule slippage, cost constraints and time-to-market pressure results in neglected testing. Nowadays testing techniques are applied throughout the software development process based on different stages of software development, increases the portion of testing before it actually reach the testing team. New development methods like Agile often uses test driven development (TDD). The primary purpose of the testing is to identify and correct failures in the system, but testing can never completely identify the defects with in software. That's why, often diverse testing is perform on different part of the software. A very famous Murphy law states "The subtlest bugs cause the greatest damage and problems" [5]. Some of the famous software failures are:

- Ariane 5 rocket, a European Space Agency rocket explodes after 39 seconds at the altitude of 2.5 km because of a small bug in numbers conversion. It was built in 10 years with the cost of $7 billion. Four expensive and uninsured scientific satellites also finishes off with it [14].
- Software errors in $193 million baggage-handling system of the Denver International Airport not only cause the delay in the opening of the airport but costing $1.1 million per day in interest [23].
- A Patriot Missile Failure on February 25, 1991 in Dharan, Saudi Arabia results in 28 deaths. It was cause of poor handling of rounding errors [4].
- The sinking of the sleipner A offshore platform in Gandsfjorden near Stavanger, Norway, on August 23, 1991, resulted in a loss of nearly one billion dollars. It was found to be the result of inaccurate finite element analysis [4].

Videogames holds a big share of software products in actual market. Video game industry is growing with an enormous rate. According to [7], in June 2011, the global video game industry was valued $65 billion, which was $62.7 billion in 2010. Video game industry has evolved significantly, from simple 2D to complex 3D games, simple colors to real time colors, simple GUI to interactive environment. With the increasing demand of complex and real time games, different game development and testing techniques have been developed to build effective and complex games. Game development and testing techniques are also important because nowadays hundreds of people work together for developing a new game. Video Game is a good teaching method, because sectors like economic, research and social needs more effective and motivating tools for learning [21].

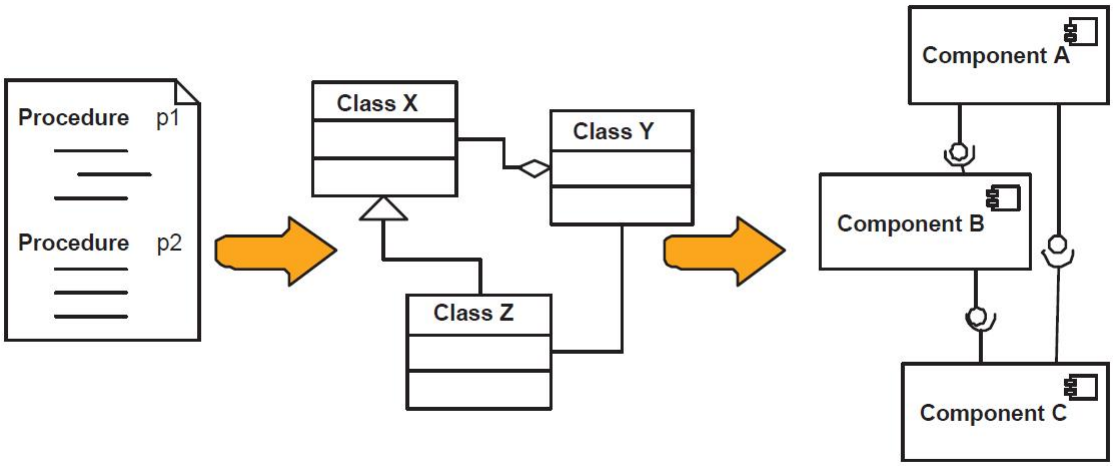

Figure 1: From Procedural Programming to Component based Development

The remaining sections of the paper are distributed as follows. Section 2 discusses common testing techniques used in software development process, section 3 discuss the general process of Game development. Section 4 discusses the testing techniques used commonly in game development. Section 5 provides the summary of the article.

2. TESTING

According to Miller [18], "the general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances". Software testing verify and validate the quality of the software produced, so as to make sure that the system is bug free and the results produced are according to the expectation.Studies have shown that different testing techniques targets different types of faults, therefore usually testing teams apply different tests to ensure the software quality. According to [19] we can divide software testing into different methods.

2.1 Code Based testing

Code-based testing (also known as white box testing or structural testing) refers to the use of source code for planning the test cases. Mostly developers perform Code-based testing. Generally code based testing is represented through a graph; either data flow or control flow graph. Code-based testing allow the developers to test the most parts of software that are rarely tested and ensures the most important function point have been tested [26]. Code-based testing is unit testing. A wide range of open source testing tools are available for automated code-based testing. All tools cover some sort of coverage criteria. There are different types of coverage criteria for code-base testing i.e. statement coverage, branch coverage, condition coverage and path coverage.

2.2 Object Oriented testing

In Object Oriented testing we focus more on integration and system testing rather than unit testing. Object oriented testing is like procedural testing with some specific characteristics, which makes it more diverse. All methods in the class are tested with unit testing, messages and events between classes are tested with integration testing where as thread interaction and thread testing is performed by system testing. JUnit [13] is the most popular testing framework for Java code, NUnit and many other tools have been developed for testing in other languages.

2.3 Component Based testing

A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Components are built for the purpose to be reused. Component based testing is assembly of reusable components, designed to meet the quality attributes identified during architecting phase [19]. Different components in component-based testing can be white box or black box because they can be developed in-house or brought from external vendors.

2.4 Specification Based testing

Specification-based testing (or Black box testing) is a testing method in which the tester did not know anything about the internal structure , design or implementation [3]. Black box test cases are derived from the design documents. Black box testing mainly tests the functionality of the software. The tester inputs the data and gets the output then the tester check the produced output with the results specified in the test case. Specification based testing is usually required due to the following reasons; code unavailability, scalability, accuracy and effectiveness. Some forms of specification based testing are:

Model based testing based on formal specifications: Formal specification are used to generate models of a system, to test the system guaranteeing high accuracy, objectivity and repeatability then ad hoc based test derivation for informal specifications. Many specification based testing approaches has been proposed for formal languages like, Z, CSP, CSS and Petri Nets. TorX, TGV and TVEDA are well known tool for formal tests.

Model based testing based on grammatical specifi-

cations: Mostly refer to as UML[1] based testing, and testing based on model design through grammatical tools. AsmL, TESTOR, UMLAUT, UML Test, AGEDIS and SCENTOR are some of the automating tools work on different coverage criteria.

Software Architecture based Testing: Software Architecture is the first document produced in software development process. It breaks down the system in a number of ways (component, connector, data elements). Software architecture is the study of how these components can be integrated in order to satisfy the desire functional and non functional properties.

Other than conventional testing methods, there are some non functional testing techniques which have an important impact on the final product quality:

Risk Based testing: Risk base testing is the probability of failure of a portion of code, as determined by its complexity [16]. In software testing we think of risk failures in three different dimensions. Way the software could fail, how likely the software can fail, and what are the consequences of the failure [10]. Risk based testing take care of the three possible risk failures, and design test cases to trigger these failures.

Performance testing: The customer major concern about software is its ability to meet performance requirement [6]. Performance tests determine how fast system or some parts of the system can work under the particular workload. Load testing, stress testing, configuration testing, spike testing are some of the sub categorization of performance testing.

Recovery testing: In software, recovery testing is how well application can recover from crashes, hardware failures and other similar failures [24]. In recovery testing the application is forced to fail, and then see how it recovers from the failure conditions and environment. Requirement specification specifies the types and extent of the recovery.

Security testing: It is the process of determining how safe the software works from external threats. The Information system should protect data from external threats, and maintain functionality as intended [9, 1]. The system should be able to stop uncontrolled system access, operating system flaws, communication system flaws and weak encryption algorithms. According to [9], a software security practitioner should perform the following to manage system security risks:

- creating security abuse/misuse cases,
- listing normative security requirements,
- performing architectural risk analysis,
- building risk-based security test plans,
- wielding static analysis tools,
- performing security tests,
- performing penetration testing in the final environment,
- cleaning up after security breaches.

[1]Unified Modeling Language (http://www.uml.org/)

3. GAME DEVELOPMENT PROCESS

Creating a game is more complex as it is believed. In 1980's only one or two person were involved in creation of a game, while in 1990's teams already included 10 people and later in 2003's the number was tripled. Today, typically for a big game, in the entire process participate hundreds of people [17]. For example in 2003, a team of 50 members work together to develop a game called Tony Hawk Underground. In 2009, Assassins's Creed II has a team of 450 developers. Everyone has a specific duty in the development process. Not only the team size has increased during this period, even the number of lines of code has passed from 100-200 to thousands and, some times, millions of lines. Therefore a kind of "organization" has become necessary. Software engineering can came to help for solving this problem.

3.1 Game development roles

There are eight principal sectors in which the game development teams can be divided:

- Design parts
- Coding parts
- Art parts
- Audio parts
- Management parts
- Quality Assurance parts
- Business parts
- Manufacturing parts

Designers have the important responsibility to invent an original and funny game idea. The designer is the only person to modify the original concept according to money and time constraints. There are also levels and dialogues designers for creating more exciting stuff for the game. A programmer job is to implement all the code necessary for running the game (3D engine, AI[2] programming and tools), respecting at the same time all the design artifacts. Artists and audio experts put significant effort to create more realistic 3D models, textures and sound effects. Management of a game project is the most critical component. Managers have to coordinate all the process steps and resources, resolve conflicts between artifacts and team members. This sector includes the executive producers that have to solve all the strategic company issues. Testers held the other critical part in a videogame life. Before the beta testing the software has to be entirely tested, all its part must work as expected. Beta phase is composed by voluntaries that play the game again and again and report all the issue that they have found. The non-fun part is played by business man. They have to cut and squeeze the game to make ends meet. Manufacturers serve as communication channel among the development studio and the shops. An example of how to compose a typical videogame development process is shown in Figure 2.

[2]Artificial Intelligence

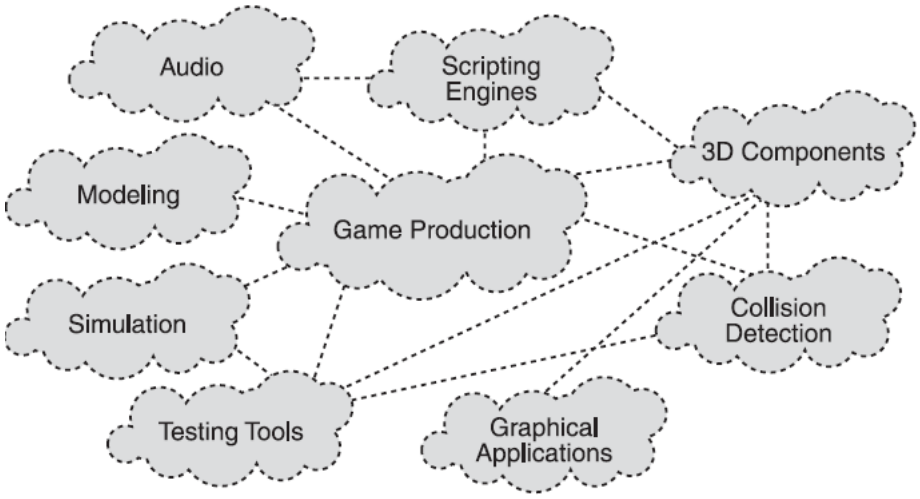

Figure 2: Typical connection between different field of game development process

3.2 Game development life cycle

Like a living organism, a game is born, lives and dies. Life cycle of a videogame is composed by the following steps, as explained in [11].

Concept Development. In this phase a small group of people brainstorming for getting out a new idea for a game. After the meeting a concept document is created, inside this document there is a short but detailed description of the videogame.

Preproduction. In this phase four artifacts are produced: the game design document, the technical design document, the project plan and a prototype of the game. In a game project, it is very important to put effort to create technical design document and game design document. Only when they are 100 percent complete, the production of the game can start formally. Technical design document should specify what programming languages will be used, such as C#, C++, SQL[3], UnrealScript[4], and should decide which components for the game may be reused from existing software or be licensed from third parties [8]. The game design document and/or the technical design document may identify areas of the game that are likely to undergo significant change during the development of the game. In this phase managers or team leaders have to take a very critical decision: they can focus on the fun part and sacrifice the business part or instead choose the business part and sacrifice the customer wish/request/demand. In the preproduction step software engineers try to identify, address, and reduce or eliminate problems in the software development effort before they cause the development effort to fail. When a problem appears in the development process, a well formed risks analysis can help to reduce and control damage.

Development. At the beginning it's a good practices to nominate functional leads for each subsystem of the game [8]. This permits an efficient division of the work and improves experience of programmers. Therefore, it is necessary to upgrade the level of the development group to produce better games. The leads parcel out the game requirements and game design document to each of the technical fields. Frequently, UML schemas are used into the initial steps. Use case diagrams, for example, illustrate the game design document and the behavior of the various subsystems in the game (Figure 3) [17]. Use case diagrams should be used as the raw material to develop the static design of the software. There are many requirements that games need to meet that are not interacting with the player, these are called non visible requirements. In a typical game basic non-visible requirements are: localization, Security, portability, database management, concurrency among different threads. Software engineering could help developers to write code faster and correctly. Design patterns and documentation are good practices, but for small project software engineering can be dangerous [27]. If the small team spends a lot of time for documenting, analyzing and planning, after there will be no more time for coding, so all release dates will be postponed resulting in customer dissatisfaction.

Alpha/Beta/Code Freeze. Planning a quality assurance plan for tracking defects and bugs is a good technique during all the development process. The test team must estimate the number of bugs, they don't have to waste time testing the areas of the game they want to test. The most critical element of the QA[5] plan must be to articulate very clearly what objective, measurable quality goal the game must achieve before it is ready to be released. All software and all games ship with bugs; knowing this, QA plan cannot be to simply test and fix the game forever until someone feels enough pressure to ship the game. After alpha and beta fixing, the code is blocked, so no more modifications are permitted.

[3]Structured Query Language (http://en.wikipedia.org/wiki/SQL)

[4]A programming language for Unreal Game Engine (http://wiki.beyondunreal.com/Legacy:UnrealScript)

[5]Quality Assurance

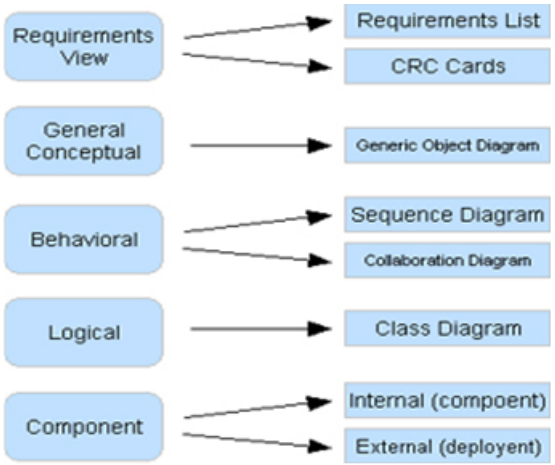

Figure 3: Different views in the design of the game

Release to Manufacture. Now is the time for truth. If all things have been done right when the game is released some gains should come early. This phase is not strictly related to the development, but is important because even a videogame is a business, someone spend money for realize it. Profits should at least cover the outgoings. Make a game is not a cheap adventure. In 2001 3,000 games were released for the PC platform, only 100 or so of those titles turned a profit, and of those only the top 50 made significant money for the developers and publishers [8]. The principle problem of the designers is that they fail to meet their financial expectations because the developers fail to clearly articulate and implicate their expectations. Knowing financial expectations it is the only way for your game project to achieving success. It is not necessary to point high gains, make a good plan for the project, even if it is not a big game yet, can help a lot for immediately earning something in the beginning and for future improvements.

Patch/Upgrade. Finally, after the game is released the work is not over, there is a dedicated team whose sole job is to maintain the game. Maintaining a game means release patch for fix some bugs, introduce new staff in the game or give help to customers.

4. TESTING IN GAMES

Testing plays a very important role in Games. A game is tested at different level of its development process. Generally in software engineering practice, Software test plan document contains all the information about testing the software. But testing game is different from testing software. There are many steps involved other than test cases for a game mostly because almost all game testing is black box testing. Game programmers usually don't test their own games, neither have time to fully test it, nor is it a good idea to test by themselves. Game programmers only test small pieces of their code before they submit it for integration with the rest of the game or as third party code to be used by other game programmers. Figure 4 shows the basic cycle of game testing. According to [11] game testing is performed in the following six step order.

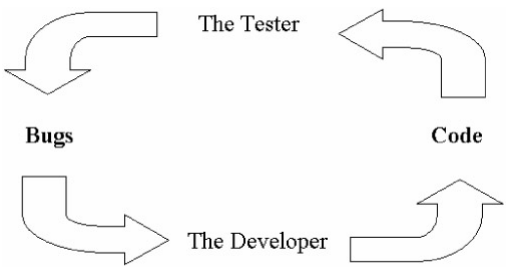

Figure 4: Testing Life Cycle

Plan and design the test: Although much of this plan and done earlier in software test plan document, but with every new prototype of the game, this documents need to be revisited to update any change in the specifications, new test cases, new configuration support. The tester should insure that no new issues were introduced.

Prepare the test: All the teams should update their code, tests, documents and test environment and align it with one another. The test development team should mark the bugs fixed, and the test time should verify them.

Perform the test: Run the test suit again. If any defect is found, test around the defect to make sure that the bug is certain.

Report the results: Complete details about the bugs are reported.

Repair the Bug: The test team participates in this step by explaining the bug to the developing team and provides direct testing to track the bug.

Return to step 1 and retest: A new build is produced after one cycle.

Testing games is a repetitive process because new build may have its own bugs and failed test.

Game testing is performed in a structured manner. Irrespective of the size of the game, and time required for producing game, all game testing should follow the basic structure as explained in [11, 25]:

Pre-production. Testing begins when the project begins. Usually there is no tester in the beginning but project scope, project design and assets are being produced in the starts and need to be evaluated, reviewed and corrected. Software test plan document is also produced in this early phases. A good testing plan document set standards for the software. There is not quality test in games. The quality of the game is evaluated by code, graphics and sounds that are produced and compiled into the game code. Proper documentation testing (project plan, design and test plan) can help in fixing problems sooner and cheaper. Delaying testing till the end may cause some serious problems including project failure.

Kickoff. "The test kickoff illustrates the principle that increasing an organization's speed results from an iterative process of identifying obstacles, designing a new process that eliminates them, and ensuring that the new way is implemented" [11]. Test kickoff has a very good impact on software development, it is an opportunity to improve test understanding, test quality and test execution. Test kickoff's address critical issues related to software under consideration, like special test instructions, test execution questions or issues, test improvement suggestions. Kickoffs are very beneficial at every lowest level of testing.

Alpha. Actual testing begins in this step. The Alpha prototype is tested against Alpha criteria established in planning phase. During the course of Alpha tested, software developed by different programmers is linked together and game design is fine tuned. Major feature of Alpha test is to play the game from start to finish along some path and revised it. All modules of the game are tested at least for once. It is very critical to stick to the test plan, and carefully perform the entire tests defined in the plan.

Beta. Beta Testing begins right after the end of Alpha test, generally begins when game is feature complete. In beta testing the focus is mostly on perfecting the game, the developers has already created. Beta test identifies and fixes the remaining bugs. Beta testing is usually performed by testers from outside the developing team. Main features of beta testing are that tester can play game along all possible paths, the entire user interface is final, game logic and AI is final, all controllers work and final artwork & audio is implemented.

Gold. By the end of beta test the game is declared to be "code freeze". A brief intense period of testing is performed on what is considered to be the final build of the game. All the test suits (or as many as time permits) are rerun on the game, to ensure that no old bug is reproduced. Any bug in this stage delays the release of the game, since a new version of the game is to be build after removing the bugs. In Gold test all critical bugs (crashes, hangs, and function failures) are solved. 90% of all major bugs (functionality and performance) are fixed. 85% of all minor bugs (system performance issues which affects some of the user) are fixed. And release level performance (i.e. 60-fps frame rate) is achieved.

Release. Once the game passed the Gold test, the game is delivered to the customer for installation or distribution to the end users. The game quality in this phase is considered to be sufficient for mass distribution.

Post-release. During its life cycle, if the game was shipped with even one or two bad defects, it's the time for the patch or update. For each patch, the development team has to revisit the entire list of bugs and incorporate some new polish features. Each individual bug fix or polish feature means more testing (but should be planned). Each new patch must be tested to see whether it works with the both the base game and earlier patches version.

4.1 Game Testing Techniques

There are some testing techniques, which are more effective in game testing with respect to software testing. Some of the important techniques are:

Combinatorial Testing: It's always an important issue for testers and software managers to decide how much testing should be enough [11] Game quality has to be good enough for customer, but testing has to stop before the release date. It is neither practical nor economical, to test every possible combination of game event, configuration, function and options. Skipping some testing or taking shortcuts is always risky. Combinatorial testing is a quick way to find defects earlier in the game while keeping the test sets small, covering as much area as possible. According to [12], the key idea of combinatorial Testing is to detect the interaction failures. Since not every parameter contributes to every failure, and most failures are triggered by a single parameter value or interactions between two or more parameters. In combinatorial testing, parameters (individual elements) are selected from game elements, functions and choices such as game events, game settings, game play options, hardware setting, character attributes, customization choices etc. The test then created can be homogenous (test parameters of same type) or heterogeneous (test parameters of different types). According to [2], combinatorial testing is very effective and beneficial such as it increases test execution efficiency (more than twice defects found per test hour), provides better quality (more defects found overall), better phase containment, reduce cost for both testing and bug fixing and increase speed to market. Pairwise combinatorial tests provide a good balance of breadth and depth of coverage, which allows tester to test more areas of the game than concentrating resources on just a few areas. [15] is an example of automation tool available for combinatorial testing.

Test Flow diagrams: Test flow diagrams (TDF) are used to create models representing game behavior from player's perspective. Testing takes place by exploring the model, along all possible paths to explore the unexpected game states. TDF is a formal approach to test design. Because TDF is graphical in nature, it is easy to review, analyze and provide feedback on test designs. Complex features can be represented by complex TDF's, but usually small TDF are preferable.

Cleanroom testing: Cleanroom testing is derived from Cleanroom Software Engineering. Cleanroom software engineering process is the software development process intended to develop software with a certifiable level of reliability [22]. Cleanroom testing techniques improves the reliability of the game. This technique is applied to the problem of why the user find bugs in the game after it has been through hundreds of hours of testing before being released. The game team's test strategy in cleanroom test is to produce tests that play the game the way user will play it.

Test Trees: Test tree is a usability technique for organizing

test cases, which helps in selection of proper set of tests for a given set of code change. Test Tree improves the overall understanding of complex game features and take care of potentially chaotic function, especially when these functions interact with other game rules, functions and elements. The test tree is constructed by decomposing the feature into subset until the bottom nodes identify elements to use or specification to perform during testing.

Play Testing: Play testing describes playing the game for non functional features like balance, difficulty and most importantly for "fun factor". Unlike other game testing techniques, play testing answers a very important question: "Does the Game work well". Play testing is more about judgment than facts.

Adhoc Testing: Ad hoc testing is also sometime refers to as general testing. It is less structured test. Ad hoc testing allows tester to explore paths based on their intuition. There are two kinds of ad hoc testing. First is free testing, which is testing without any planning or documentation. The second is direct testing, which is a single test, improvised to answer a specific problem.

5. CONCLUSION

Software testing has become a very broad area over the last decade. It is included in almost all the software engineering development steps such as specification, design, implementation, maintenance, process and management issues. In this work we have tried to give an over view of the testing state of the art, by briefly introducing different methods of testing.

We also introduce game development briefly. Game development process is very complex, and may involve hundreds of people. Many small teams work on different aspects of software, which are then combined together. Game development process is very well defined, with clear division of responsibilities.

Game Developing team on average, spends more time on testing than any other application development team. In fact game development process is mostly based on black box testing. Game testing sometimes became more complex, because of the diverse development of different components of the game. Testing plays an important role in every development steps, and the developing team only moves to the next level after approval from the testing team. Over the years, software testing techniques have matured and are fruitful, but still need to improve adequately. Only in US, more than $22.6 billion can be saved by implementing an improved testing infrastructure to enable and more effective identification and removal of software defects [20].

Some area of software testing like testing automation, self testing, anti-model based testing are still not fully matured. Significant amount of work is going on in software testing and developing efficient test processes and techniques and tools that assist in the creation of quality software has become the one of the most important research area of this field.

6. REFERENCES

[1] Testing experience. *The Magazine for professional Testers*, 2009.

[2] Combinatorial testing web site. http://www.combinatorialtesting.com, October 2011.

[3] Differences between black box testing and white box testing. http://softwaretestingfundamentals.com/differences-between-black-box-testing-and-white-box-testing/, October 2011.

[4] D. N. Arnold. Some disasters attributable to bad numerical computing. www.ima.umn.edu/ arnold/disasters/disasters.html, October 2011.

[5] R. Avidor. Murphy laws site. http://www.murphys-laws.com/murphy/murphy-computer.html, October 2011.

[6] A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker. Software performance testing based on workload characterization. In *Proceedings of the 3rd international workshop on Software and performance*, pages 17–24, New York, NY, USA, 2002. ACM.

[7] L. B. Baker. Factbox: A look at the $65 billion video games industry. http://uk.reuters.com/article/2011/06/06/us-videogames-factbox-idUKTRE75552I20110606, October 2011.

[8] E. Bethke. *Game Development and Production*. Plano, Texas, 2003.

[9] G. M. BRUCE POTTER. Software security testing. *IEEE COMPUTER SOCIETY*, 2004.

[10] J. B. Cem Kaner. Risk-based testing.

[11] R. B. Charles P. Schultz and T. Langdell. *Game Testing All in One*. Thomson Course Technology, Boston, 2005.

[12] Y. L. D. Richard Kuhn, Raghu N. Kacker. Practical combinatorial testing. *NIST Special Publication 800-142*, October 2010.

[13] S. L. Erik Hatcher. *Java Development with Ants*, chapter Chapter 4 testing with JUnit. October 2002.

[14] J. Gleick. A bug and crash. http://www.around.com/ariane.html, October 2011.

[15] R. Kuhn. *Combinatorial Coverage Measurement Tool*, January 2010.

[16] A. G. Linda H. Rosenberg, Ruth Stapko. Risk-based object oriented testing.

[17] P. McCarthy. Software engineering and game development. *University of Wisconsin-Platteville*.

[18] E. F. Miller. *Introduction to Software Testing Technology*. Second edition.

[19] H. Muccini. Software testing: Testing new software paradigms and new artefacts. *The Wiley Encyclopedia of Computer Science and Engineering*, pages 2716–2732, January 2009.

[20] B. Schechner. Software testing techniques. *Getting started with software testing*.

[21] U. Wechselberger. Teaching me softly: Experiences and reflections on informal educational game design. *University of Koblenz-Landau*.

[22] Wikipedia. Cleanroom software engineering. http://en.wikipedia.org/wiki/Cleanroom_Software_Engineering,

October 2011.

[23] Wikipedia. Denver international airport: Accidents and incidents. http://en.wikipedia.org/wiki/Denver_International_Airport, October 2011.

[24] Wikipedia. Recovery testing. http://en.wikipedia.org/wiki/Recovery_testing, October 2011.

[25] Wikipedia. Software release life cycle. http://en.wikipedia.org/wiki/Software_release_life_cycle#Alpha, October 2011.

[26] Wikipedia. Software testing. http://en.wikipedia.org/wiki/Software_testing, October 2011.

[27] J. P. F. with Omar Salem. *Software Engineering for Game Developers*. Stacy L. Hiquet, Boston, 2005.