A Review of Virtual and Augmented Reality in Mobile Game Development

**Type:** Learning-oriented

Megan Smith, Shakeel Osmani

CSC 8350 Advanced Software Engineering

Computer Science Department, Georgia State University   
Atlanta, Georgia 30302 USA

Spring 2016

[megan.n.smith729@gmail.com](mailto:megan.n.smith729@gmail.com), [shakeel.osmani@gmail.com](mailto:shakeel.osmani@gmail.com)

Abstract— **Game development has recently gained a lot of momentum and is now a major software development industry amounting to 22 Billion USDs a year in revenue. With the rise of more powerful and less expensive mobile devices, conforming mobile game development to include applications utilizing virtual reality (VR) and augmented reality (AR) has become increasingly popular. Hence, this paper will focus on the aspects of software engineering and development of VR and AR mobile gaming applications. We will explore how to apply design patterns to handle issues with first person shooting game development, as well as the options and the best practices of the Game Development Life Cycle (GDLC). We will also investigate the existing agile game development methods and how to apply these methodologies into virtual and augmented reality systems. In this paper, the Unity3D game engine is used to represent the mobile game architecture as it is able to control features and components for mobile game application development. The Unity3D component diagram is created to visualize the requirements of common mobile games and to capture the embedded functional processes. We will also discuss how to use Unity 3D to achieve mobile augmented reality. And we will use this as a guideline to make Google cardboard augmented reality and virtual reality applications running on android.**

Keywords— Game development, Virtual reality, Augmented reality, Unity3D, Game Development Life Cycle, Mobile game development, Software engineering, Google Cardboard

1. introduction

Virtual and augmented reality projects are closely tied to rapid evolution of technology, and to the need for client’s constant feedback during the whole projects lifecycle. The gaming industry as a whole evolves constantly. There is a great need to increase reusability, maintainability and quality of game development in order to adapt to evolutionary game development processes. This can be done by using design patterns, following the game development life cycle model and guidelines and by applying iterative and incremental agile game development methodologies. All of these tactics will be further discussed in this paper.

1. design patterns

Using design patterns allows for a well-designed game program that extends easily, that is easily portable to other platforms without deep revision of source code, has better scalability, maintainability, flexibility and extensibility, allows for code reusability, and has robustness to changes. This results in that fact that a programmer will not have to completely discard their work to accommodate changes in algorithm, evolution of game level or changes to game mechanics.

There are two categories of design patterns in game development. One type of design pattern is used to describe the game mechanics during game development. These design patterns address the game play and game rules. The other type of design pattern is the object orientated design pattern. We will be mainly focusing on object orientated design patterns for this review.

Object orientated design patterns can be applied to first person shooter games. Where, first person shooter games involve an avatar, weapon and enemies/interaction objects. Design patterns can be applied in the implementation of expressive play, game and character state management, rendering scene, maps in the game world, online and team communication, weapon management and multi-enemy rendering. We will focus on those implementations that pertain closely with virtual reality systems on mobile devices, such as game state management, character state management and weapons management.

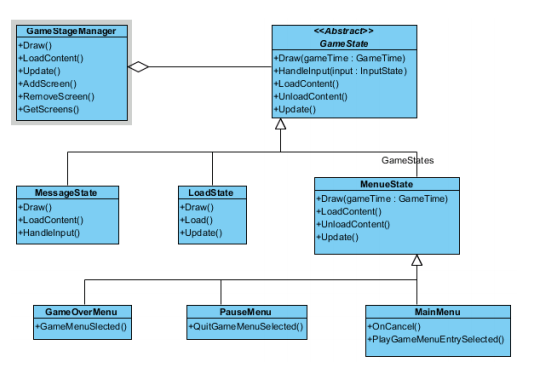


Figure 1. State Pattern for Game State Management

The state design pattern avoids tightly coupled code handled by if or switch statements. Therefore, the state design pattern allows an object to alter its behavior when it’s internal state changes. Each state is a subclass, so if more states are required during game development, the programmer simple adds another subclass. The classes are well encapsulated, and the change of state is implemented within each class. Above in figure 1, you can see the state design pattern for game state management. Here, it is shown that the GameStateManager maintains a concrete state at any given time. The abstract GameState class encapsulates the behaviour associated with a particular state of game. The concrete states of game such as LoadState, MenuState, Pause, EndState, and Main implement the behaviours associated with each state in regarding Draw() and Update() respectively.

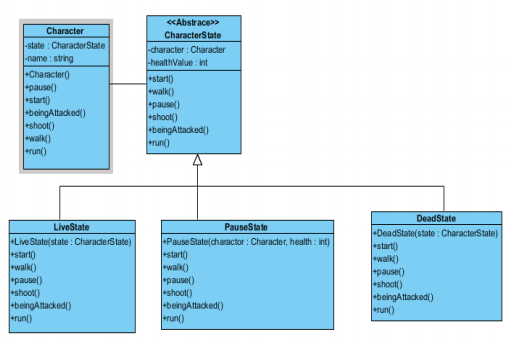


Figure 2. Character State Management with State Pattern

The character in a first person shooter game is designed with different states: normal, killed, damaged, etc. Where there are different behaviors for different states. Above you can see the character state management with a state design pattern diagram. Here, CharacterState is an abstract class. The subclasses, LiveState, PauseState and DeadState, are concrete classes that implement the transition between different states with the pause(), start() and beingAttacked() methods. The game state management design pattern and the character state management design pattern are very similar in functionality and design.

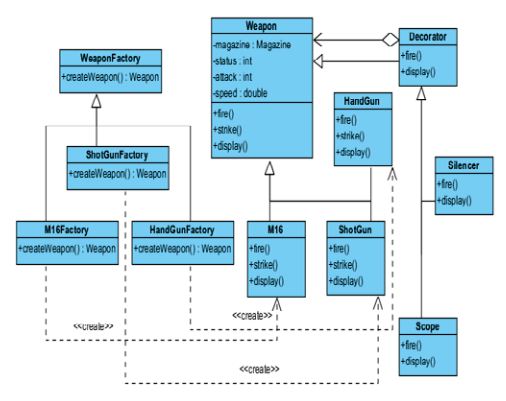
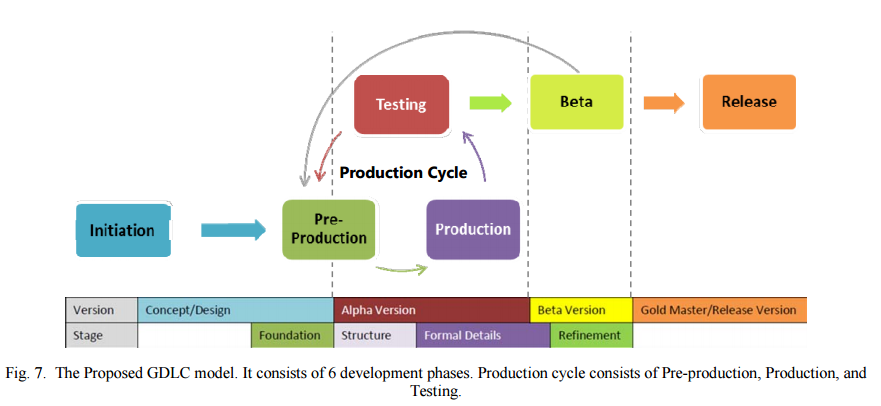


Figure 3. Factory Methods and Decorator Patterns in Weapon Management of FSP

First person shooter games, such as virtual reality games, involve multiple weapons, which have multiple accessories, such as a silencer or scope, and the weapons can have different appearance, usage and power as well. Here, the programmer wants to build a system that will be able to add new weapons without modifying the existing system. In order to accomplish this, the programmer uses the factory method, decorator and iterator design patterns. Factory method pattern defines an interface for creating an object, but let’s subclasses decide which class to instantiate. Decorator pattern provides a flexible way to extend functionality by attaching additional functions to a weapon dynamically. The weapon management design pattern can be seen above in figure 3, where the chosen design pattern will utilizes factory methods and decorator patterns in regards to a first person shooter gaming environment. In the design, the WeaponFactory is an abstract class and can be extended with different concrete class such as ShotGunFactory, M16Factory etc. to create different concrete weapons. The Weapon class is an abstract and is extended with different concrete weapons to realize different functions such as display(), fire() etc. The Decorator class maintains a reference to a Weapon object and defines an interface that conforms to Weapon's interface [1].

1. game development lifecycle and guidelines

Video games have recently become an overwhelming profitable subset of software engineering. Whereas software development has its own Software Development Lifecycle in order to ensure a systematic process to developing software, game software development requires different guidelines to govern its development process. This such process is the Game Development Lifecycle.

Software development for games requires a tailored development process because the primary purpose of games is to provide entertainment, where other forms of software are purely a product of engineering. Games are a mixture of programming and art, where the creative and imaginative aspects of the final product are just as important as the code behind the game itself.

There are many forms of the Game Development Life Cycle and many practices on how a Game Development Life Cycle is applied in a project. No single Game Development Life Cycle is perfect, but there are three main questions that arise when constructing a Game Development Life Cycle. Question one: What are the steps and the phases of a game development process? Question two: What are the quality criteria’s that must be considered during each phase? Question 3: What kind of Game Development Life Cycle which can be the best practice in a proper game development and deliver a good quality product?

The goal of the research summarized in this review is to propose a new Game Development Life Cycle and guidelines to properly conduct the outputted Game Development Life Cycle and successfully deliver a good quality game.

*Game Prototype Usability Quality Criteria:*

The criteria used to determine the game quality is based on Fullerton’s game prototype usability quality criteria. A prototype can be summarized into four main stages, where those stages are judged based on five main quality criterias. The first stage is Foundations, which represents the gameplays basic concepts in the form of either low fidelity prototype or incomplete game. The second stage is Structure, which is a refined version of Foundations which already has the core gameplay logic, mechanics and rules. The third stage is Formal Detail, which is a refinement of Structure that includes necessary rules and procedures to make the game fully functional. The final stage is Refinement, which is the refined and almost finished game.

The first quality criteria is Functional, which means the game’s feature is playable and operating well. The next quality criteria is Internally Complete, which indicates all rules, branches and conditions have been properly addressed. The third quality criteria is Balanced, which ensures the game difficulty is just right. The fourth quality criteria is Fun, which means the game is engaging, challenging and entertaining. The last quality criteria is Accessible, which means the game is easy to understand, easy to learn and intuitive.

*The Proposed Game Development Life Cycle:*

The proposed Game Development Life Cycle consists of six development phases. The first phase is initiation. This where the game concept and a simple game description will be developed.

The second phase of the Game Development Life Cycle is pre-production. This phase involves the creation and revision of the game design and the game prototype. This is also where the first prototype iteration is constructed. The first prototype focuses on the Foundation and Structure stages. Here the Foundation stage centers of the fun quality criteria and the Structure stages centers on the core gameplay and mechanisms, such as logic and game rules.

The third phase is the production phase. Here the developers focus on asset creation, source code creation and the integration of both elements into the prototype. This prototype iteration deals with formal details and refinement. Where formal details adds new features, improving overall performance and performs bug fixing and refinement deals with making the game more fun, challenging and easier to understand.

The fourth phase is testing. The type of testing performed here is internal testing, which tests the games usability and playability. Each phase in the Game Development Life Cycle has a testing method specific to that phase. For instance you can have Formal Details Testing, a testing method to test the internally complete quality criteria or you can have Refinement Testing. The output of the testing phase is a bug report, change request and general development decisions.

If the results of the testing phase are appropriate, the next phase in the Game Development Life Cycle is Beta. The Beta phase conducts third-party or external testing, called beta testing. The Beta phase tests both Formal Details and Refinement. There are two main types of Beta testing: closed beta and open beta. Closed Beta testing only allows invited individuals to be the testing participates. While open Beta testing allows anyone who registered to be the testing participates. The output of Beta testing are bug reports and user feedback.

The last phase is Release. The release phase occurs once all Beta testing is finished and has returns positive results. In the release phase, the product is launched, the project documentation is written, knowledge of the game development is shared, post-mortems are performed and planning for the game maintenance and game expansion is addressed.

The Game Development Life Cycle can be viewed as a whole in the image provided at the top of the previous page.

*Game Development Guidelines*:

In order to successfully apply the proposed Game Development Life Cycle, the game development guidelines should be followed. These guidelines closely follow the phases of the Game Development Life Cycle. The first guideline is role management, which provides explanation, importance and responsibilities of each role. The next guideline is initiation, which provides methods on how to generate ideas and game concepts. The following guideline is pre-production, which provides explanation of game design elements, such as game description, characters, storyline and concept arts. The next guideline is production which focuses on programming and asset creation. This guideline is followed by the testing, beta and release guidelines which develop testing methods and explains how to release the game package.

The proposed Game Development Life Cycle was applied to a project called Feline Project, where this project was completed in four production cycles. The Feline Project succeeded in producing a good quality game. This application shows that applying the proposed Game Development Life Cycle in a real life game development process is successful in creating and delivering a good quality game [3].

1. agile development process for virtual reality

The proposed model consists of a hybrid model, gathering elements from both XP and Scrum, adapted to the context of Virtual Reality systems development. Development is executed iteratively, and feedback received in past iterations is used to help planning the next ones. Some key features of VRS development include evolutionary nature, iterative building of high-fidelity models, need for clients feedback, need for interaction and usability testing and need for system modularization.

Figure 4 represents the virtual reality development process from a basic, standard perspective where the process is based on Software Engineering models, adapted to the particularities of Virtual Reality Systems. The proposed process is composed of 5 stages, executed in each iteration: Requirement Analysis, Design, Implementation, Evaluation and Deployment. This simple model does a good job presenting a development process that aggregates prototyping with iterative and evolutionary software development.

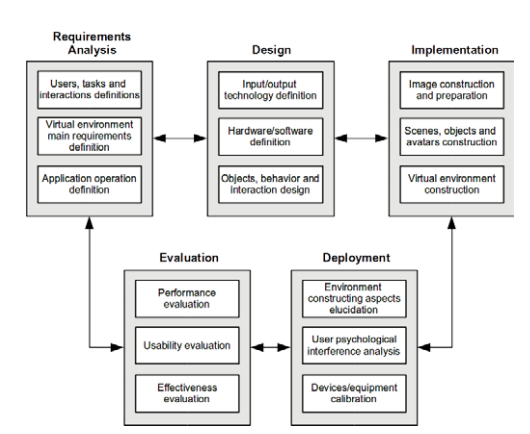
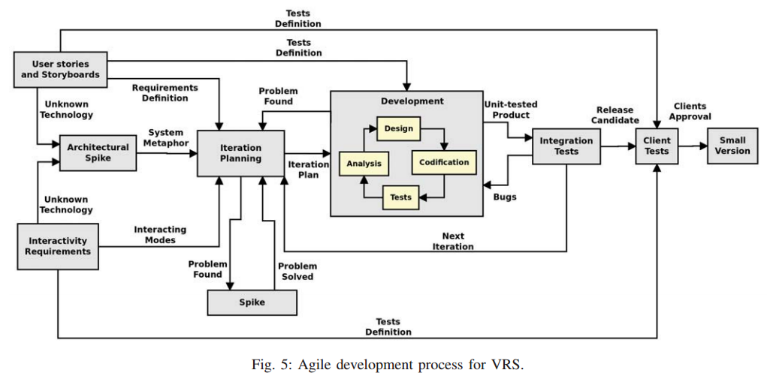


Figure 4. VRS development process.

The figure at the top of the following page shows the agile development process for virtual reality systems. The process beings with the user stories/storyboards, which give a description of system functionality from the user’s point of view. User stories are very helpful on requirement analysis because they provide developers with users’ real expectations about the system. The next step in the process is the interactivity requirements. Here, the programmer develops test applications for implementing the desired interaction methods. These applications should help developers and clients in the evaluation and viability analysis of the required interaction methods.

During the user stories or the interactivity requirements phases of the agile development process, an architectural spike can occur. The architectural spike is where viability analysis of the new requirements is conducted. Here, resources available for requirements implementation are investigated. This activity has the goal of reducing risks related to unknown technology, such as third-party libraries. The architectural spike results in the metaphor definition. Where a metaphor is a subset of the requirements in focus.

Iteration planning, the next phase in the development process, allows for adaptive behavior to embrace changes on application requirements and provide solutions found inside the iteration. Iteration planning results in an iteration plan addressing requirements, user stories and interactivity requirements analysis. During iteration planning, each time a problem is detected, a spike is conducted, in order to investigate and propose possible solutions. A spike is a small development cycle, which provides developers with possible solutions to a problem. Inside the spike, test applications or prototypes can be built to help developers on testing and discussing proposed solutions. If possible, clients’ feedback can be used to guide the development team in the right direction, according to users’ needs.

The next phase in the agile development process for virtual reality systems is development, which consists of four main tasks: analysis, design, codification and tests. To begin, analysis and design share the common goal of structuring the implementation of the iteration plan’s requirements. Codification is the production of source code. Tests are guided by the clients and are used to guide implementation of the most important requirements.

Once the development phase is completed, the process beings with integration tests. Modifications performed in the current iteration will be added to the main system after passing integration tests. If any problem is found during integration tests, development activity is restarted. Developers will then propose and test possible corrections to the problems found. Integration tests output a release candidate, which is a significant number of modifications integrated into the main stream.

The final phases of the development process includes client tests and small version. Client’s tests is where clients perform functional usability and integration tests on the release candidate. The client tests output a small version on approval from clients, where a small version is a subset of the proposed requirements [5].

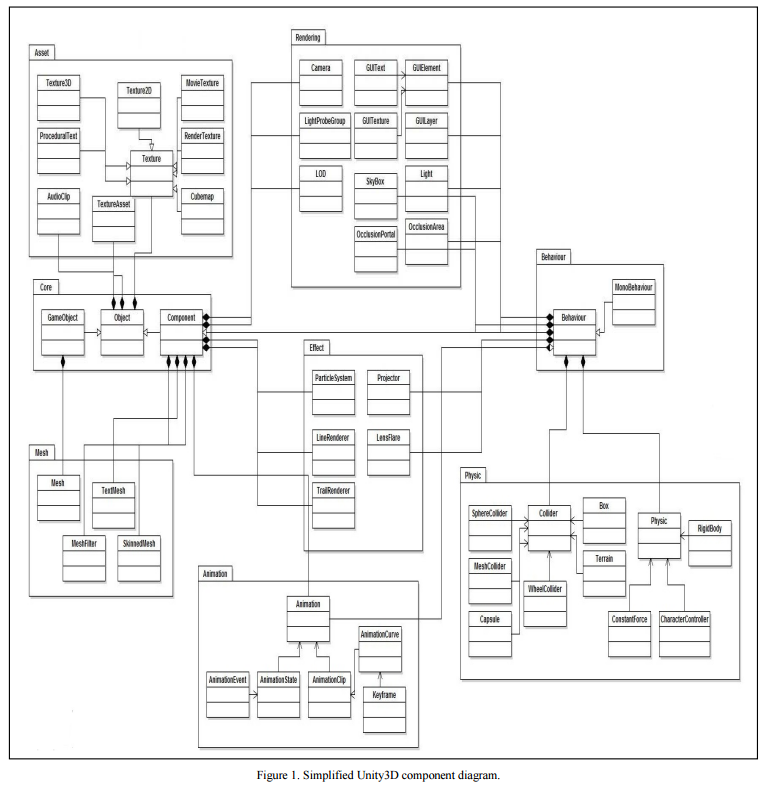
1. unity 3d and blender

*Unity 3D:*

In order to build virtual reality gaming worlds, developers need a powerful gaming engine. The engine discussed in this paper is Unity 3D. Unity 3D is a multi-platform development tool that was designed for creating games from the easiest phase. Unity comes with a fully integrated set of tools and allows for rapid flows to create 3D content. The Unity 3D gaming engine has high rendering power, and highly optimized physically-based shading.

Unity also provides an intuitive workspace with testing and editing features. The Unity environment is set up in such a way that developers can see the relations between programming code and game objects visibly. Unity is also considered one of the most widely referred gaming engine in most research and mobile development groups [4].

A simplified Unity component diagram can be seen on the following page. These components are the building blocks for any virtual or augmented reality gaming application. The Core component consists of three class diagrams namely Object, Component and GameObject class diagrams. The Object class is used to create, use or destroy the model for the current scene. The Behaviour component consists of Behaviour and MonoBehavior class diagrams. This component is used to enable and disable the model in the Object class diagram in the Core component and also associates with other components such as Rendering, Physic, Effect, and Animation. The Animation component consists of Animation, AnimationState, AnimationEvent, AnimationClip, AnimationCurve and Keyframe class diagrams. This component allows the modification of speed, time and scripting to playback the animations. The Physic component consists of SphereCollider, Box, WheelCollider, MeshCollider, Capsule, Terrain, Collider, RigidBody, CharacterController and ConstantForce. This class provides functions for collider processes and also the application of continuous forces on the objects. The Rendering component consists of Camera, LightProbeGroup, and LOD classes that are associated with the Component class in the Core component. The Effect component consists of ParticleSystem, LineRenderer and TrailRenderer and is associated with the Component class. The Mesh component includes Mesh class, MeshFilter class, SkinnedMesh class and TextMesh class and allows the modification of mesh scripts and filters all the mesh components through association with the Component and GameObject class diagrams in the Core component [2].



*Augmented Reality:*

The software engineering concepts covered in this paper can also be applied to augmented reality applications, not just virtual reality applications. Augmented Reality is am application where virtual content is placed on top of a real world camera view. Unity 3D and the Vuforia augmented reality extension enables vision detection and tracking functionality within Unity and allows developers to create augmented reality applications easily. Unity 3Ds built in shaders and physics engine allows for realistic model appearance and manipulation effects. Augmented reality applications built in Unity 3D also allows for virtual buttons. Where a virtual button is a virtual object that is drawn on and can be pressed by a user directly. Using virtual buttons makes for a more intuitive and more user friendly interaction within the augmented reality application. .

*Blender:*

To create detailed 3D gaming objects to be used in virtual and augmented reality applications, modeling software should be utilized. In this paper we will discuss the Blender modeling software. Blender is a free open-source 3D creation suite that offers cross platform support. It supports the entire 3D pipeline, which consists of modeling, rigging, animation, simulation, rendering, compositing, etc. Blender also allows for customize applications with Python scripting.

1. android and google cardboard

The virtual or augmented reality applications created in the Unity environments can easily be built onto an Android device. Where Android devices provides the generic platform for the mobile game. Android also has a tight integration with Unity via an SDK. Once on the mobile phone the game can be played by using Google Cardboard.

To begin, a programmer only needs to use the Google Cardboard plugin within the Unity environment before building the game to the Android device. The Google Cardboard establishes a virtual reality appearance for the user through the use of two 45 mm focal length lenses. The player can also interact with the game by using the magnets on the side of the Cardboard viewing device. The magnets are used for clicking inside of games. When you pull and release the ring, your phone’s magnetometer detects changes in the magnetic field. The magnets are essentially using the compass sensor in the phone.

The Goggle Cardboard also provides a NFC (near field communication) tag. The tag is used to trigger the launch of the Cardboard Android Application automatically. Cardboard applications split the smartphone display image into two, one for each eye. Then applies barrel distortion to each image to counter pincushion distortion (towards the center of the image) from the lenses. Where barrel distortion gives the apparent effect of an image which has been mapped around a sphere or barrel. This ultimately results is a stereoscopic ("3D") image with a wide field of view.

1. conclusion

In conclusion, in this paper we have discussed the various software development techniques that can be applied to virtual and augmented reality systems. In order to achieve a sound procedure for software development, programmers can use different design patterns for different aspects of their virtual reality mobile game, such as the state design pattern or the decorator design pattern.

The Game Development Lifecycle can also be a very helpful tool while developing video games, whether they be virtual reality or not. More specifically, the agile development process for virtual reality systems allows for the flexibility and constant iterations and prototyping needed for virtual reality gaming development.

The virtual reality mobile game can be developed using the Unity 3D environment, along with Blender to allow for more complex 3D modeling for game objects. Mobile games developed in Unity can easily be deployed onto any Android phone, and with the help of the Google Cardboard viewing device, developers can make the mobile game into a virtual reality mobile game.

References

1. Qu Jufeng, Wei Young, Song Yinglei, “Design patterns applied for networked first person shooting game” IEEE June 30 2014
2. Abdullah N.A.S, Rusli N.I.A, Ibrahim M.F., “Mobile Game Size Estimation: COSMIC FSM Rules UML Mapping Model and Unity 3D Game engine”, IEEE, pp 42-47, 26 Oct 2014
3. Ramadan R and Y Widyani, “Game Development Life Cycle Guidelines,” IEEE, 28 September 2013, pp.95-100
4. Sung Lae Kim, Hae Jung Suk, Jeong Hwa Kang, and Jun Mo Jung, “Using Unity 3D to Facilitate Mobile Augmented Reality Game Development,” IEEE, 08 March 2014, pp. 21–26
5. F. Mattioli, D. Caetano, A. Cardoso, and E. Lamonier, “On the Agile Development of Virtual Reality Systems,” Int'l Conf. Software Eng. Research and Practice, SERP 2015, pp. 10–12