BEYOND VERSION CONTROL - INSPIRATIONS FOR NEW GAME DEVELOPMENT PIPELINES AND FUTURE GAME ENGINES

Oliver Engels and Robert Grigg
International Game Architecture and Design (IGAD)
NHTV Breda University of Applied Sciences
4800 DX Breda, The Netherlands
E-mail: `120010@nhtv.nl`, `grigg.r@nhtv.nl`

KEYWORDS

Game Development Methodology, Game Engines, Game Development Tools, Version Control Systems

ABSTRACT

Collaboratively editing game worlds and the underlying asset version management techniques present us with many challenges when using a traditional Version Control System (VCS). This paper suggests a new method that will aid in the tracking, branching, and selection of asset versions within a game development pipeline. A prototype has been developed which demonstrated improvements in both productivity and usability providing greater visibility of incremental asset changes and alterations to their relationships which equated to approximately a 30% better user experience when compared to an existing industry VCS.

INTRODUCTION

In our last publication a focused prototype was developed around the Quality Assurance (QA) processes of a game development pipeline (Engels and Grigg, 2015) where the process of finding, recording and communicating product issues was done all within the game engine environment. The aim was to evaluate the improvements gained when the approach is to remain as much as possible within the game engine. This experiment was part of a bigger vision of how game development pipelines (see Figure 1) and future game engines, that may live in the cloud, will function for improved productivity.

Previously we have shown that it is not only possible, but allows developers to stay within the game engine to quickly find and resolve problems leading to substantial productivity gains. This paper continues on that concept of what a game production environment of tomorrow would look like given a collaborative editing platform and how the underlying versioning of assets and product releases could be handled. The research question posed: *Would a new innovative approach to asset versioning, review, comparison and selection provide improvements in a game development production pipeline?*

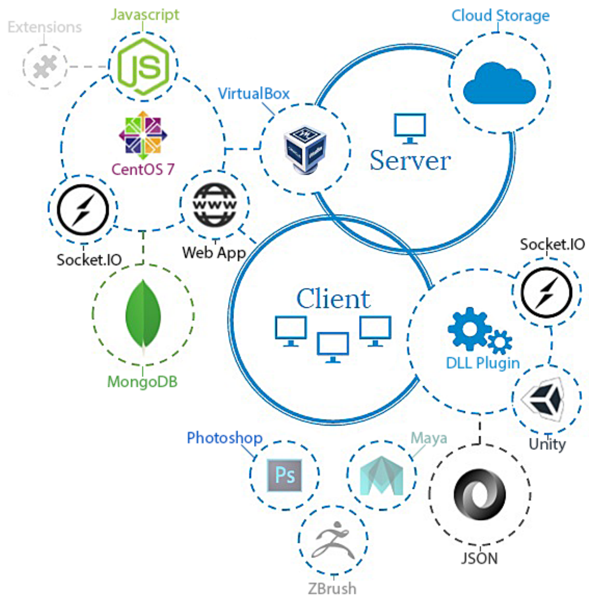


Figure 1: Architecture of the prototype system.

VERSION CONTROL SYSTEMS

Version Control Systems (VCSs) are widely used in the games development industry and there is an abundant amount of resources for the use, operation and management (Loeliger and McCullough, 2012; Wingerd, 2005; Vesperman, 2006; Pilato et al., 2008). There are also similar resources for supporting tools that gives these systems more features or makes them more user friendly, an example of this is the version management client Tortoise (Harrison, 2011). Currently there is little research into the underlying mechanisms, models and methodologies behind a VCS.

We see the game industry as an iterative development environment, however, this is not something that permeates all aspects of the game development pipeline. Game developers always look to move forward, where the old is replaced with the new, and the broken replaced with the repaired. In situations where the latest version of an asset is found to be inadequate, or technically create a problem, it is then *rolled-back* to be replaced by a prior version or the problem is addressed and the new version of the asset is *added.*

VCS PRACTICALLY

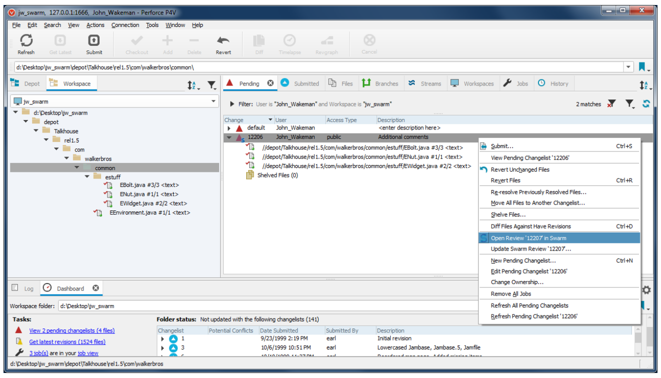
VCS software is used to store different versions of an asset to a shared central repository that is remote and secure to that of the end-user's computer. Originally tools from the software development industry focused on the effective storage of text files that contained things such as source-code, meta-information, or configuration details. A lot of these tools faced challenges in moving from a source control model to a larger asset management function where assets in game development included binary files and extremely large files.

The operations of a VCS includes *adding, removing, replacing,* and *obliterating* (the permanent removal) at an asset or file level. At the sub-asset level then *updating* parts of the asset may include changes to lines of a text file or positions of vertices in a model. Actions can also include *reverting* where this is the *replacing* of an older version of the asset. Tools that accompany or are apart of the VCS may allow asset *locking* to prevent changes to the same file at the same time and *comparison* to review their internal changes or visual differences of an asset. To help manage a group of assets that are associated, in a release for example, then *labeling* or *tagging* is used. This can be viewed more generally as meta-data that is kept on assets within the repository that for some VCSs is often customisable.

More complex operations to assets such as *renaming*, *copying*, *moving*, and *restructuring* (altering the underlying folder structure) lead to further challenges and complications such as the creation of large numbers of duplicate assets or assets that are quite similar. A recent example of this was from the developer CD Projekt RED's Witcher 3 project where an example of 2600 texture assets had found 300 potential duplicates and 13 identical assets (Krzyscin, 2016).

Finally these systems need most of these operations to be performed manually to ensure that the assets make it into the VCS but can also lead to missing or lost assets if not performed properly. Loh worked to devise a formal model to address some of these complicated VCS issues, for example, conflicting assets where both a developer's local asset has changed and also the remote version by another user (Loh et al., 2007). Others have formulated models based on a patching perspective (Angiuli et al., 2014) in work connected to open-source projects such as Darc (Hoffmann, 2004).

Perforce (Cowham and Firth, 2013) is a leading Distributed Version Control System (DVCS) in the game industry (see Figure 2) and in 2008 was then used by 18 of the 20 top game developers (Aguirre, 2008). Perforce is used at game companies such as EA (Electronic Arts, 1982), Ubisoft (Ubisoft, 1986) and Nintendo (Nintendo, 1889). The company describes their tool as *"an enterprise version management system in which users connect to a shared file repository. Perforce applications are*


jw_swarm, 127.0.0.1:1666, John_Wakeman - Perforce P4V
File
Edit
Search
View
Actions
Connection
Tools
Window
Help
Refresh
Get Latest
Submit
Checkout
Add
Delete
Revert
Diff
Timelapse
Revgraph
Cancel
Depot
Workspace
Pending
Submitted
Files
Branches
Streams
Workspaces
Jobs
History
2 matches
Submit... Ctrl+S
View Pending Changelist '12206'
Revert Unchanged Files
Revert Files Ctrl+R
Re-resolve Previously Resolved Files...
Move All Files to Another Changelist...
Shelve Files...
Diff Files Against Have Revisions Ctrl+D
Open Review '12207' in Swarm
Update Swarm Review '12207'...
New Pending Changelist... Ctrl+N
Edit Pending Changelist '12206'
Change Ownership...
Remove All Jobs
Refresh All Pending Changelists
Refresh Pending Changelist '12206'
Log
Dashboard

d:\Desktop\jw_swarm\depot\Talkhouse\rel1.5\com\walkerbros\common\

```
jw_swarm
d:\Desktop\jw_swarm
  depot
    Talkhouse
      rel1.5
        com
          walkerbros
            common
              estuff
                EBolt.java #3/3 <text>
                ENut.java #1/1 <text>
                EWidget.java #2/2 <text>
              EEnvironment.java #1/1 <text>
```

Filter: User is "John_Wakeman" and Workspace is "jw_swarm"

Change	User	Access Type	Description
default	John_Wakeman		<enter description here>
12206	John_Wakeman	public	Additional comments

```
//depot/Talkhouse/rel1.5/com/walkerbros/common/estuff/EBolt.java #3/3 <text>
//depot/Talkhouse/rel1.5/com/walkerbros/common/estuff/ENut.java #1/1 <text>
//depot/Talkhouse/rel1.5/com/walkerbros/common/estuff/EWidget.java #2/2 <text>
Shelved Files (0)
```

Workspace folder: d:\Desktop\jw_swarm

Tasks:
View 2 pending changelists (4 files)
Get latest revisions (1524 files)
3 job(s) are in your job view

Folder status: Not updated with the following changelists (141)

Changelist	Potential Conflicts	Date Submitted	Submitted By	Description
1		9/23/1999 2:19 PM	earl	Initial revision
3		10/6/1999 10:51 PM	earl	Lowercased Jambase, Jambase.5, Jamfile

d:\Desktop\jw_swarm\depot\Talkhouse\rel1.5\com\walkerbros

Figure 2: Perforce P4V - client-side VCS application.

used to transfer files between the file repository and individual user workstations" (Wingerd, 2005). With the use of meta-data Perforce is able to store information on different asset versions, permissions and change-logs. Perforce, SVN (Pilato et al., 2008), and GitHub (GitHub, 2008) are VCSs that allow developers to use *branching* where another version of the project can be developed and tested in parallel with one or more other branches. Applying the use of branching for the first time often requires training for the concept to be properly used. Perforce takes this a step further with *streaming* where there are rules and relationships between branches that aims to reduce the merging between branches by half due to a straight copy relationship in one direction (Perforce, 2012). The example of *streaming* is a step closer to automating some aspects of a VCS.

Finally a VCS does not automatically capture all changes made but this is something that a collaborative development environment incorporates at the core. Examples today are services such as Google Docs (Google, 2007) aim to capture all changes while also being able to show the history of changes and revert if required. We propose a different way of doing this with version control automation to review, compare and select assets to be combined and used in a project.

VCS AUTOMATION

Automatic versioning allows everybody in the development team to use branches to its full potential. The branching in the prototype is on a per asset basis and will automatically determine the number of the assets from the server side. Developers would not need to lock assets as new versions are saved to the server, and assets will not be overwritten, but start a new branch.

A project build consists of asset versions and these can be *swapped*, *updated* and *replaced* throughout the development process. Auto-branching will allow a developer to branch without thinking of the concept of branching. A revision of a file will also not state that this is the latest file, it will only give its position within a particular branch. An older file can have a higher revision than a younger file and vice versa, this happens as versions and branches are established to the files that are already on the server. Versions of assets on the server can always be retrieved from the server to be used in other parts of a project.

When an asset is saved a function is called to hash the byte data of that asset into a MD5 encrypted key. The file will then be copied to an upload buffer folder where it will be temporarily stored to receive a callback from the server for *uploading* or *deleting*. The client will first send the MD5 hash to the server and check if the file already exists there. If it does then the server will send the delete action back to the client. If the opposite is true then the server will add the hash to the database and check what version the server needs to receive, which can be determined by looking at the last uploaded version from the developer. All server interaction of the developer are stored in the database which also includes the last upload and download requests for assets. From here we can determine the version that the new asset needs to get by looking at these assets (see Figure 3).

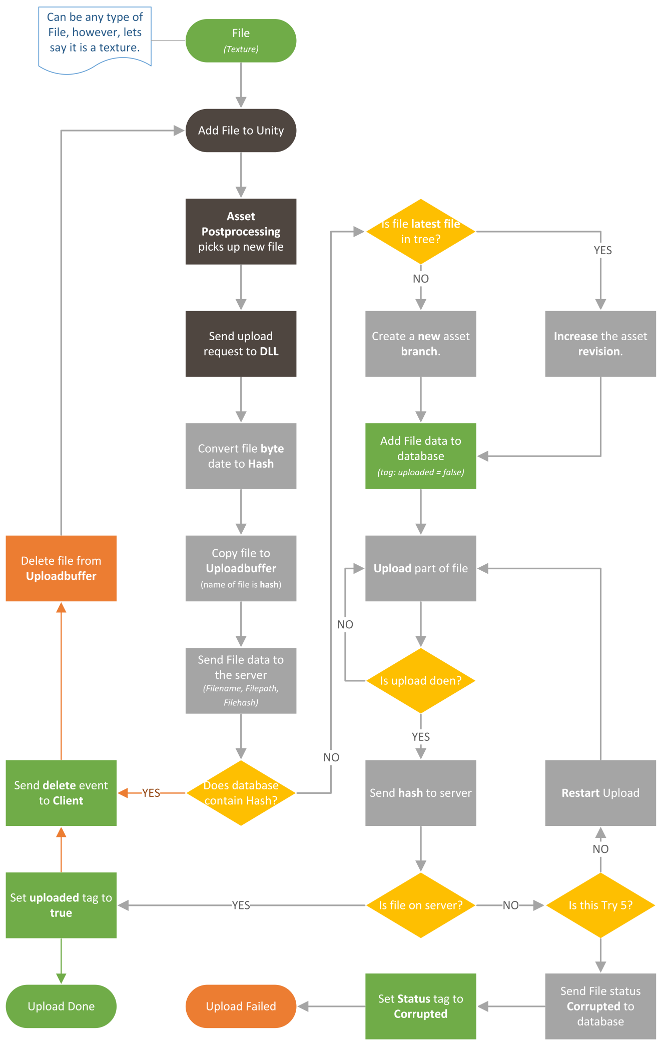

Can be any type of File, however, lets say it is a texture.
File (Texture)
Add File to Unity
Asset Postprocessing picks up new file
Send upload request to DLL
Convert file byte date to Hash
Copy file to Uploadbuffer (name of file is hash)
Send File data to the server (Filename, Filepath, Filehash)
Does database contain Hash?
YES
NO
Send delete event to Client
Delete file from Uploadbuffer
Is file latest file in tree?
YES
NO
Create a new asset branch.
Increase the asset revision.
Add File data to database (tag: uploaded = false)
Upload part of file
Is upload doen?
NO
YES
Send hash to server
Restart Upload
Is file on server?
YES
NO
Is this Try 5?
NO
Set uploaded tag to true
Upload Done
Upload Failed
Set Status tag to Corrupted
Send File status Corrupted to database

Figure 3: FTP file upload handling.

For this prototype automatic branching is to be handled by using four numbers, an example of a version given to an asset could be (1.0.3.5). The first number indicates the *Trunk* that the given asset is in which is (1) for (1.0.3.5). The next number is the *Parent Trunk* which is (0) for (1.0.3.5). The third number is the *Branch* that the asset is in which is (3) for (1.0.3.5). The last number is the *Revision* number of the asset which is (5) for (1.0.3.5). From these numbers we can determine the origin and direction of the asset, for asset number (1.0.3.5) we can determine that its origin is (1.0.3.4) if *Revision* (5) of (1.0.3.5) is the first revision of the branch then the origin would be (1.0.?.4) where the question-mark is not required to indicate the origin of the given asset. For the direction we simply increase the *Revision* number, for example (1.0.3.5) to (1.0.3.6). If the system notices that there is already a version (1.0.3.6) then the automatic branching operates and give the asset the number (1.3.?.6), where the question-mark will be determined by the amount of branches in the (1) *Trunk*. Figure 4 shows the version numbers that an asset will receive and how the branching could look for an asset.

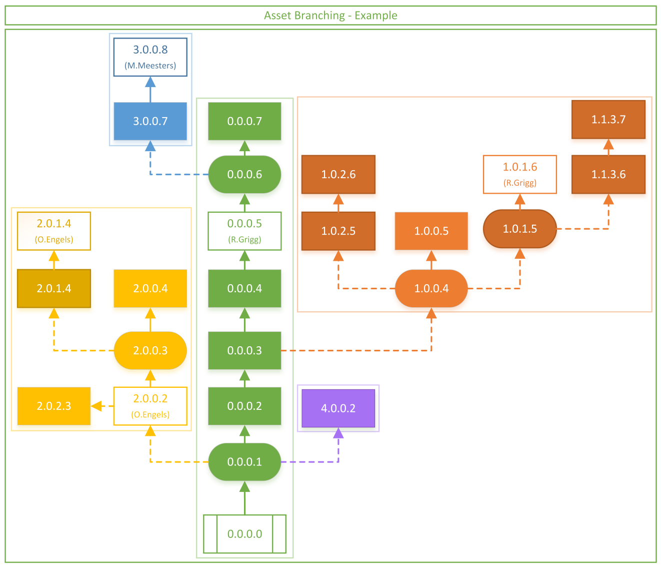
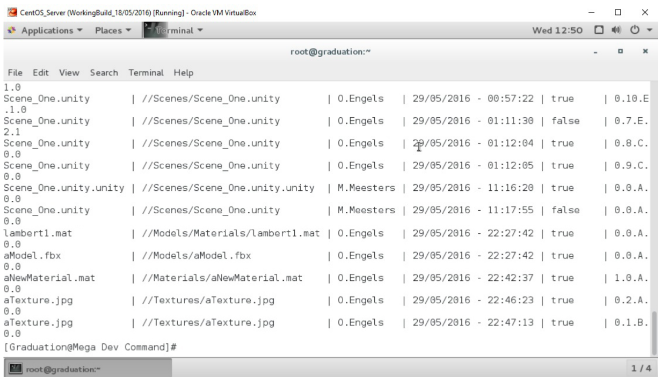


Figure 4: Asset version branching diagram.

EXPERIMENT

For the server-side of the prototype a Virtual Machine (VM) using VirtualBox (Oracle, 2007) was used to run a CentOS 7 (CentOS Project, 2004) operating system (see Figure 5) for an image that can be run locally on a server or deployed to the cloud. The server also had Perforce installed for obtaining a comparison for the experiment. Unity (Unity Engine, 2005) was the game engine used, although the system is designed to be engine agnostic, as this was the most suitable solution for the development of the prototype (see Figure 6). The server-side of the prototype was developed with NodeJS (Node.js, 2009) and uses Socket.IO (Socket.IO, 2014) for the communication between client and server. A MongoDB (MongoDB, 2009) database was setup for data storage. The extendability of MongoDB was ideal for use as the data within the database can grow rather quickly. The FTP was used for uploading and downloading files to and from the server and on the client-side a FTP client library (Otom, 2012) was used to accomplish this. Connection to the Unity Engine was done by using a Dynamic Linking Library (DLL) file written in C++ that would connect to the server and transfer data.



```
1.0
Scene_One.unity       | //Scenes/Scene_One.unity         | O.Engels    | 29/05/2016 - 00:57:22 | true     | 0.10.E
.1.0
Scene_One.unity       | //Scenes/Scene_One.unity         | O.Engels    | 29/05/2016 - 01:11:30 | false    | 0.7.E.
2.1
Scene_One.unity       | //Scenes/Scene_One.unity         | O.Engels    | 29/05/2016 - 01:12:04 | true     | 0.8.C.
0.0
Scene_One.unity       | //Scenes/Scene_One.unity         | O.Engels    | 29/05/2016 - 01:12:05 | true     | 0.9.C.
0.0
Scene_One.unity.unity | //Scenes/Scene_One.unity.unity   | M.Meesters | 29/05/2016 - 11:16:20 | true     | 0.0.A.
0.0
Scene_One.unity       | //Scenes/Scene_One.unity         | M.Meesters | 29/05/2016 - 11:17:55 | false    | 0.0.A.
0.0
lambert1.mat          | //Models/Materials/lambert1.mat | O.Engels    | 29/05/2016 - 22:27:42 | true     | 0.0.A.
0.0
aModel.fbx            | //Models/aModel.fbx              | O.Engels    | 29/05/2016 - 22:27:42 | true     | 0.0.A.
0.0
aNewMaterial.mat      | //Materials/aNewMaterial.mat     | O.Engels    | 29/05/2016 - 22:42:37 | true     | 1.0.A.
0.0
aTexture.jpg          | //Textures/aTexture.jpg          | O.Engels    | 29/05/2016 - 22:46:23 | true     | 0.2.A.
0.0
aTexture.jpg          | //Textures/aTexture.jpg          | O.Engels    | 29/05/2016 - 22:47:13 | true     | 0.1.B.
0.0
[Graduation@Mega Dev Command]#
```

Figure 5: VM running CentOS 7 server.

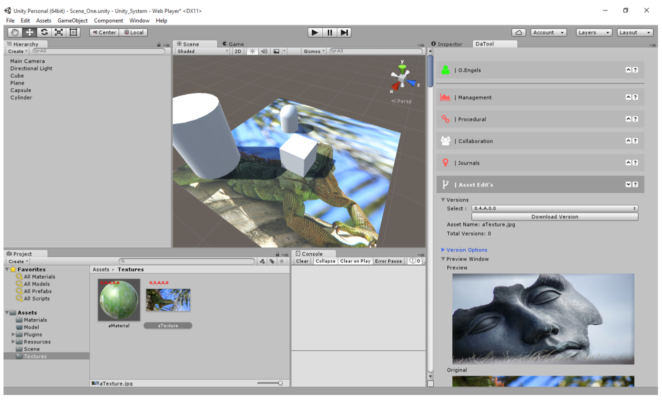


Figure 6: Unity 5 auto-versioning prototype.

For the experiment users were asked to find and review pictures from both Perforce and the prototype, after which a questionnaire was completed. The questionnaire was based on the System Usability Scale (SUS) (Patrick et al., 1996) and consisted of twelve questions of which two are open. The other ten questions are on a scale of one to five with five being the most positive.

RESULTS

Participants were excited by the new approach that was handled by the prototype within their production environment. To keep the test as accurate as possible we focused on the asset revisioning and retrieving feature. The results from this show that participants would

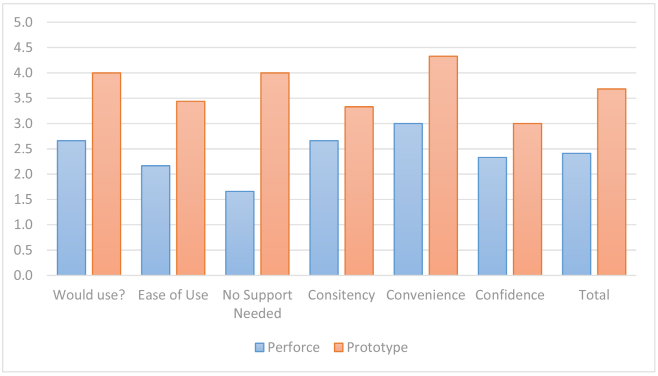


Figure 7: User results from the prototype use.

rather use the new improved way of working embodied in the prototype and that there was little need for support and explanation for its operation. Perforce was perceived as much more complex and some participants were having a hard time understanding it. All participants agreed that both tools were very useful within the game industry, however, the results showed that the prototype scored an average of 30% higher than Perforce, which is a significant increase.
The findings show that an automatic versioning approach to game assets has a positive influence on the way developers work. Assets within the game development pipeline do not need to be manually committed, or assets locked, which avoided a number of problems. For large milestones developers no longer needed to be reminded to commit missing or updated assets. The research in this document has shown that developers embraced this new way of working that removes a lot of the overheads of a VCS.

DISCUSSION

The branching functions in a traditional VCS was particularly challenging for certain developers as training or support was required. The automated prototype allowed developers to not worry about this while still having the benefit of using a branching system inherently. This approach opens up the option for Producers, Team-Leads, and even the possibility for an Editor role in being able to *review*, *compare* and *select* different assets for a given release.
Additionally people in more of a management or directing role could join the collaborative environment and not only annotate the end result but also the steps involved in the alteration of assets in, for example, the structuring of a game level. For certain assets the dependencies on other assets can lead to errors when retrieving and combining these resources, for example, when an asset is linked to assets that do not exist in the client project. These problems are in part handled when the transfer of the missing file is the solution but when deeper dependencies are involved, such as the number of required parameters in a script function, then a smarter VCS ap-

proach in combination with smarter engine architecture may be the solution.

CONCLUSION

We conclude that the use of assets versions that can be *reviewed*, *compared* and *selected* is not only feasible but beneficial for the development process of games. It allows for quicker iterations and multiple development branches where different developers can use and edit the same assets. They do not have to worry about branching and corrupting assets for other developers, which provides a more relaxed way of working. Developers can get together within the software they are using and try out things without interfering with each other, show the new edits to team members and obtain feedback within the software that they are most familiar with. These edits can then be added to the already working build of the game and be reviewed by a producer or lead within the company. All versions of assets have the incremental steps of the asset changes recorded which can be retrieved from the server at any time. Existing VCSs may record incremental changes to non-binary formats but fail to capture useful incremental changes to binary assets of operations that relate to the organisation of assets together.

FURTHER RESEARCH

Some assets are sensitive to changes and have dependencies that if not met may break a build. One approach could be the improvement of the underlying model where these dependencies are incorporated into the VCS. These dependencies, in combination with an automated testing system, could be refined to be more sensitive to build and play errors and may possibly feedback to the version and branch information in defining *operational subsets* of assets.
In creating a more engine agnostic approach the development of an Unreal Engine (Epic Games, 1991) or CryEngine (Crytek, 1999) version may help test and improve the existing prototype.

REFERENCES

Angiuli, C., Morehouse, E., R Licata, D., & Harper, R. (2014). *Homotopical Patch Theory.* New York, NY: ACM.

Cowham, R. & Firth, N. R. (2013). *Learning Perforce SCM* (1st ed.). Birmingham, UK: Packt Publishing.

Engels, O. & Grigg, R. (2015). *Translating a modern film and TV screening room to an integrated game engine production environment.* Ostend, Belgium: GameOn 2015.

Harrison, L. A. (2011). *TortoiseSVN 1.7 Beginner's Guide* (2nd ed.). Birmingham, UK: Packt Publishing.

Krzyscin, K. (2016). *CD Projeckt Red: Evolving REDengine.* Mgr Hopmansstraat, Breda: CD Projekt Red.

Loeliger, J. & McCullough, M. (2012). *Version Control with Git* (2nd ed.). Sebastopol, CA: O'Reilly Media.

Loh, A., Swierstra, W., & Swierstra, D. (2007). *A Principled Approach to Version Control.* Microsoft Research.

Patrick, J., B, T., Ian Lyall, M., & Bernard, W. (1996). *Usability Evaluation in Industry* (1st ed.). Boca Raton, FL: CRC Press.

Pilato, M., Collins-Sussman, B., & W Fitzpatrick, B. (2008). *Version Control with Subversion* (2nd ed.). Sebastopol, CA: O'Reilly Media.

Vesperman, J. (2006). *Essential CVS* (2nd ed.). Sebastopol, CA: O'Reilly Media.

Wingerd, L. (2005). *Practical Perforce* (1st ed.). Sebastopol, CA: O'Reilly Media.

WEB REFERENCES

Aguirre, S. (2008). 18 of Top 20 Games Publishers Use Perforce to Manage Digital Assets. http://www.prweb.com/releases/Game_Development/Perforce_Software/prweb1557184.htm. Alameda, CA, U.S.

CentOS Project. (2004). CentOS. https://www.centos.org.

Crytek. (1999). CryEngine. http://www.crytek.com/. Frankfurt, Germany.

Electronic Arts. (1982). EA Games. http://www.ea.com. Electronic Arts, Inc.

Epic Games. (1991). Unreal Engine. https://epicgames.com/. Crossroads Blvd Cary, NC.

GitHub. (2008). GitHub. https://github.com. San Francisco, CA: GitHub Inc.

Google. (2007). Google Docs. Mountain View, CA.

Hoffmann, G. (2004). Darcs. http://darcs.net/. Brooklyn, NY.

MongoDB. (2009). MongoDB. https://www.mongodb.org. MongoDB Inc.

Nintendo. (1889). Nintendo. http://www.nintendo.com. Kyoto, Japan.

Node.js. (2009). Node.js Foundation. https://nodejs.org.

Oracle. (2007). Virtual Box. https://www.virtualbox.org. Oracle Corporation.

Otom. (2012). FTP Client Class. Germany.

Perforce. (2012). Perforce Streams Adoption Guide. https://www.perforce.com/sites/default/files/pdf/streams-adoption-guide.pdf. Perforce.

Socket.IO. (2014). Socket.IO. http://socket.IO.

Ubisoft. (1986). Ubisoft. https://www.ubisoft.com. Rennes, France.

Unity Engine. (2005). Unity 5. https://unity3d.com.