HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Bearbeitungsbeginn: 01.09.2014

Vorgelegt am: TBA

Thesis

zur Erlangung des Grades

Master of Science

im Studiengang Medieninformatik

an der Fakultät Digitale Medien

Dominik Steffen

Matrikelnummer: 245857

Technical game-authoring process and tool development

Erstbetreuer: *Prof. Christoph Müller*

Zweitbetreuer: *Prof. Dr. Wolfgang Taube*

Abstract

Arbeitsprozesse mit heutigen Game Engines verlangen von Entwicklern meist das Erlernen neuer Toolsets und das während eines Zeitlich knapp bemessenen Projekt Zeitraums. Es wäre für Entwickler einfacher sich mit den bereits bekannten Tools, wie Modeling Editoren, zu beschäftigen und mit diesen großartige Ergebnisse zu erreichen. Designer müssen sich oft in unbekannte Editoren und SDKs einarbeiten während Entwickler sich in Grafische Editoren einarbeiten sollen um ihren Code an der richtigen Stelle des Projekts einzubinden. Diese Arbeit baut eine Brücke zwischen beiden Welten. Durch die Konzeption eines Software Tools und Entwicklungsprozesses zum erstellen von Game Authoring Tools, wird gezeigt wie mit verschiedenen Frameworks bestehende Software erweitert werden kann um sie als Authoring Tool zu benutzen. Mit Hilfe eines für Cinema 4D konzipierten Plugins ist es möglich, dass Designer oder Entwickler jederzeit mit ihren eigenen Tools in die Entwicklung eines Projektes einsteigen. Das während der Arbeit konzipierte und in Teilen umgesetzte Plugin und Framework bietet grundlegenden Funktionen um an einem Projekt mit der Fusee Engine zu arbeiten. Das zuerst während dieser Arbeit als Prototyp umgesetzte Plugin bietet ausreichende Funktionalität um ein Projekt als Entwickler als auch als Artist zu erstellen und zu bearbeiten. Hierzu wurden verschiedene weitere Game Engine Editoren analysiert. Das C# Plugin Projekt Fusee Uniplug wurde analysiert und in seinem Funktionsumfang erweitert. Das Ergebnis dieser Arbeit ist eine grundlegende Software Bibliothek in C# die nicht nur für Cinema 4D eingesetzt werden könnte sondern Editorunabhängig als Authoring Tool Framework und Tool weiterentwickelt werden kann.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Masterthesis selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel die sowohl zum schreiben dieser Arbeit als auch zum Entwickeln des dazugehörigen Sourcecodes benutzt wurden, habe ich angegeben.

Dominik Steffen, Küssaberg den 20. Mai 2015

"Hier steht ein wichtiges Zitat zur Entstehung dieser Arbeit."

- TBD.

Dominik Steffen
Matr.-Nr.: 245857
Hochschule Furtwangen

E-Mail:
dominik.steffen@hs-furtwangen.de
stik@hs-furtwangen.de
dominik.steffen@gmail.com

Inhaltsverzeichnis

1 Einführung

Arbeitsprozesse in heutigen Game Engines verlangen von Entwicklern häufig das Erlernen neuer Tools und Toolsets und dies während eines zeitlich knapp bemessenen Projektzeitraums. Es wäre für Spieleentwickler möglicherweise einfacher sich nur mit bereits aus dem Arbeitsalltag bekannten Tools zu beschäftigen und die Anzahl der verwendeten Tools auf ein Minium zu reduzieren. Dies reduziert möglicherweise auch den Arbeitsaufwand der Tool Developer selbst. Sie könnten so bestehende Tools erweitert ohne die Notwendigkeit jegliche mögliche Funktion in das Tool zu integrieren. Welche Arten von Tools verwendet werden unterscheidet sich je nach Entwicklergruppe. Für Artists und Designer handelt es sich hierbei meist um Welt-Editoren sowie Grafik und 3D-Modeling Applikationen oft auch Digital Content Creation Applikationen (kurz DCC) genannt. Programmierer in der Spieleentwicklung arbeiten meist mit Entwicklungsumgebungen, Debuggern und Analyse Tools der Game Engine.

Spezielle Game Authoring Tools[1] und Editoren ermöglichen aber auch Teammitgliedern ohne tiefgehendes technisches Verständnis für die Programmierung von und mit Engines, an Projekten mitzuarbeiten. So können Designer und Artists Skripte und Funktionalität für die Spiellogik entwerfen oder die Usability und Performance der Applikation testen. Authoring Tools können das Entwickeln von Anwendungen erheblich beschleunigen und Entwickler bei alltäglichen Aufgaben unterstützen. Vgl. Stefan, Göbel Ralf, Steinmetz Florian, Mehm Christian,Reuter. »Future Trens in Game Authoring Tools«. In: (2014). S. 536-541

Diese Editoren wie z.B. der Unreal Engine 4 Editor und der Unity3D Editor bieten oft Möglichkeiten ein Projekt visuell zu editieren. Oft gibt es auch Möglichkeiten visuelles Scripting oder visuelles Programming[2] in diesen Tools zu nutzen. Authoring Tools verbreiten sich heute selbst auf dem Consumer Markt. Sei es hier durch so genannte Mod Kits (Tools die Entwickler zur Erweiterung

[1] Werkzeuge zum Erstellen von Interaktiven Softwareapplikationen

[2] Programmierung mit Hilfe eines oft auf Graphen basierenden User Interfaces

durch Fans für ihre veröffentlichten Spiele bereitstellen) oder durch Game Engines die auf User Generated Content setzten wie die V-Play Game Engine siehe [Feldbacher, Christian Klinge, Heiko, Chefredakteur. »User-Generated Content in the V-Play Game Engine«. In: *Making Games* 3 (2015), S. 24–27]. Hier beschreibt die Making Games in der Ausgabe 03/2015 die Nutzung der V-Play Engine um einerseits Interaktive Software wie Spiele für den Markt zu entwickeln, als auch die Möglichkeit das Tool dazu zu verwenden User-Generated Content zu erzeugen und Software so längerfristig durch Kundenbindung am Markt zu etablieren.
Im professionellen Sektor ist das Entwickeln interner Tools eine anspruchsvolle Aufgabe und dieser Aufwand wird meist bei größeren Projekten wie z.B. dem im Jahr 2014 erschienenen Action-Rollenspiel Deck13. *Lords of the Fallen*. http://www.deck13.de/. 2014. (Geprüft am 18. 05. 2015) des Entwicklerteams Deck13 aus Frankfurt praktiziert. Das Team hat hierfür einen WYSIWYP (What you see is what you play) Editor für das Produktionsteam des Titels entwickelt. Der Editor bietet die Möglichkeit das Spiel genau so zu bearbeiten wie es der Spieler nach dem Kauf zu sehen bekommt was aus einem Interview mit den Entwicklern in der Zeitschrift Making Games hervor geht:

> "We needed an editor that could display the game in the same way that a player would experience it, as we couldn't allow for differences in the experience of, say, a level artist and the gamer, or a game designer and a gamer. [...] Basically you can start the game in »game mode« and »editor mode«. [...] Also, it meant that artists ang game/level-designers used the same view on all objects."
>
> Klose, Jan Lange, Thorsten. »Lords of the Fallen Tackling a new gen game with an emerging studio«. In: *Making Games* 3 (2015), S. 44–45

Diese Entwicklung birgt einen interessanten Gedanken und zeigt, dass Tool Development in der Spiele-Industrie in den letzten Jahren durch komplexe Projekte und immer aufwendigere Features einen wichtigen Stellenwert innerhalb der Spielebranche erreicht hat. Es zeigt aber außerdem, dass Tool Development einen immer größeren Aufwand bei Planung und Umsetzung benötigt. Die Monumentalen Werkzeuge der heutigen Game Engines unterstützen jegliche Funktionalität in einem Programm und sind aus diesen Gründen oft schwerfällig in Bedienung und Weiterentwicklung. Möglicherweise wäre es interessant die Funktionalität auf verschiedene kleine Tools aufzuteilen um so jeder Entwicklergruppe ein eigenes Paket an die Hand zu geben das nur die benötigte Funktionalität enthält. Wird dieser Ansatz nun kombiniert mit der Überlegung

keine komplett neuen Tools zu schaffen, sondern bestehende Software durch Pluginfunktionalität zu erweitern könnte eventuell einiges an Entwicklungszeit eingespart werden.

1.1 Fragestellung

Diese Arbeit untersucht ob ein nach Entwicklergruppen getrenntes Tool Design den Anforderungen von alltäglichen Taks ausreichen könnte reflektiert den Entwicklungsprozess von Game Authoring Tools, konzipiert ein solches Editor Tool auf Basis der Fusee[3] Engine und setzt Module davon in die Praxis um. Nach Entwicklergruppen getrennte Tools zeichnen sich durch eine geringe Kopplung untereinander aus. Sie überschneiden sich in ihrer Funktionalität aber stark. So können Projekte mit der Fusee Engine aktuell nur in Visual Studio erstellt werden. Aufgrund der fehlenden GUI Implementierung muss hierfür aber der Code der Engine und das Prozedere der Fusee Programmierung vorerst durchdrungen werden. Ein Modeling Editor wie Cinema 4d bietet darüber hinaus GUI Funktionalität und könnte so eine Schnittstelle für all diejenigen Mitarbeiter sein, deren tägliche Arbeit sich nicht auf das Programmieren beschränkt sondern auch das Erstellen von Modellen, Grafiken und weiteren Asset Dateien umfasst. Hierzu zählen in etwa Designer und Artists. Ihnen könnte eine GUI Oberfläche die Entwicklung mit Fusee erleichtern oder gar erst ermöglichen, während sich für Software Ingenieure (Engineers) nichts an der gewohnten Arbeit in Visual Studio verändern würde was die Effizienz der Entwickler möglichweise steigern kann. Das konzipierte Authoring Tool kann beide Welten der Entwicklung (Programmierung und Gestaltung) getrennt von einander bedienen ohne dass eine Partei die gewohnte Anwendungswelt verlassen muss. Dies steht im Gegensatz zu den Monumentale Editoren der Unreal Engine 4[4] und der Unity 3D[5] Engine. Dieses Konzept ist an Jason Gregorys These, dass Authoring Tools verlässlich und einfach zu nutzen sein müssen, angelehnt. Vgl. Jason Gergory. *Game Engine Architecture*. Taylor & Francis Ltd., 2009. ISBN: 978-1568814131, S. 49.
Beide anderen Engines (Unreal Engine 4 und Unity3D Version 5) werden in einem gesonderten Abschnitt behandelt. Diese zwei Applikationen bieten die komplette Welt des Game Engine Authoring in einer Applikation an. Sicherlich hat dieser Ansatz gewisse Vorteile aber vor allem den Nachteil, dass sich Entwickler vor dem Beginn eines Projektes in das jeweilige Tool einarbeiten

[3] Fusee (Furtwangen Simulationd and Entertainment Engine - http://www.Fusee3d.org)
[4] EPIC GAMES, INC. *Unreal Engine 4*. http://www.unrealengine.com. 2004-2015. (Geprüft am 18. 05. 2015)
[5] Unity Technologies. *Unity 5*. http://www.unity3d.com. 2015. (Geprüft am 16. 05. 2015)

müssen. Damit verlängert sich die Asset Pipeline des Projekts um mindestens einen weiteren Schritt was wiederum Einfluss auf die Qualität der Assets haben kann. Durch das getrennte Tool, soll keine der Parteien einen Nachteil betrefflich der Funktionalität erlangen. Es soll hier weiterhin in einem praktischen Ansatz untersucht werden, wie es möglich ist ein solches Authoring Tool zu konzipieren wenn es in einen bereits bestehenden proprietären Modeling Editor (hier Cinema 4D von Maxon[6]) mit Hilfe dessen API integriert werden soll. Hierfür gelten einige Voraussetzungen. Das konzipierte Tool und die Anbindung an Cinema 4D müssen in der Programmiersprache C# implementiert werden. Das Modul Uniplug, ein Teilprojekt der Fusee Engine, muss verwendet und das Authoring Tool trotzdem nicht an den Cinema 4D Editor gebunden sein. Des Weiteren soll das Tool durch andere Entwickler leicht auf andere Editoren anpassbar sein.
Somit wird auf eine enge Bindung zwischen dem Authoring Tool und dem Modeling Editor Cinema 4D verzichtet. Dies hat weiterhin den Vorteil, dass bei crashes der Spiele Engine die Tools nicht beeinträchtigt werden. Eine geringe Verlässlichkeit der Tools durch diese vermeidbaren Probleme könnte den Produktionsprozess negativ beeinflussen. Vgl. Jason Gergory. *Game Engine Architecture*. Taylor & Francis Ltd., 2009. ISBN: 978-1568814131, S.55.
Den Authoring Tool Entwicklern kann durch die Verwendung von bereits erhältlicher Software als Benutzerschnittstelle einiges an Arbeit abgenommen werden. Die wiederverwendung von Funktionalität beschleunigt Implementierungsphasen und führt zu schneller nutzbarer Software. Dies kann beispielsweise durch die Verwendung von APIs erfolgen. Weiterhin sind viele Funktionen dieser Software bereits getestet und somit entfällt auch hier ein gewisser Anteil an Arbeitsaufwand. Das Open Source ATF Framework[7] von Sony zielt auf genau diese Entwicklung ab und wird in dieser Arbeit auch weiter untersucht. Ben Carter schreibt in Ben Carter. *The Game Asset Pipeline*. Delmar Cengage Learning, 2004. ISBN: 1-58450-342-4:

> "The users of the tool [...] should spend as much of their time as possible in that tool. Every time they have to switch to another application to perform some task [...] they are losing time and potentially, breaking their concentration. [Car04, S. 18]"

Seine Theorie spricht also für die Integration von Tool Funktionalität in die meist genutzte Software der Entwickler. Carter regt hier an, dass der Nutzer so

[6] MAXON Computer GmbH. *Cinema 4D R16*. http://www.maxon.net/de/products/cinema-4d-studio.html. 2014-2015. (Geprüft am 18. 05. 2015).
[7] AuthoringToolFramework-Abrufbarunterhttps://github.com/SonyWWS/ATF/wiki letzter Zugriff am 15.05.2015

viel Zeit wie möglich in einem Tool verbringen sollte. So kann die Konzentration des Entwicklers länger aufrecht erhalten werden und er wird nicht durch Veränderungen der Arbeitsumgebung gestört. Dies ließe sich durch die bereits erwähnte Trennung von Tools nach Entwicklergruppen realisieren.

1.2 Struktur

Vor der Konzeption wird der Arbeitsprozess des Authoring Tools Development anhand von Fachliteratur und Interviews aus Fachzeitschriften untersucht. Verschiedene Projektmanagement Modelle zur Software Entwicklung werden geprüft und eine Empfehlung für die interne Entwicklung eines Authoring Tools unter dem Zeitdruck eines laufenden Projektes wird aufgrund der gewonnenen Erkenntnisse formuliert. Es folgt ein Abschnitt der sich dem Requirements Engineering für das konzipierte Tool widmet. Auf dem Requirements Engineering basierend wird ein Softwarekonzept inklusive System Design entwickelt. Nach der Konzeption wird versucht die Basis Funktionalität des Frameworks und des C4D Plugins in Visual Studio mit Hilfe von C# Code und der nach C# gewrappten[8] Cinema 4D API als Programmunabhängige Softwarebibliothek zu implementieren. Die gewrappte Cinema 4D API basiert auf einem ehemaligen Projekt der Hochschule Furtwangen. Dieses wird als Grundlage für die hier umgesetzte Implementierung genutzt und bot zum Zeitpunkt dieser Arbeit einen marginalen Umfang an Basisfunktionalität zur Kommunikation mit der Applikation Cinema 4D. Das Uniplug Projekt bietet somit die Möglichkeit Plugins für Cinema 4D in der Programmiersprache C# zu entwickeln. Mit den Maxon eigenen Schnittstellen ist nur das Erstellen von Plugins nur in C++, Python und Coffee (einer von Maxon selbst entwickelten Skriptsprache) möglich.
Diese Arbeit zielt nicht darauf ab ein komplett entwickeltes Tool für das Erstellen von Spielen in der Fusee Engine zu erschaffen. Es wird eher versucht einen Grundstein für weitere Forschung und Entwicklung auf dem Gebiet des Game Authoring Toolkit Developments im Bezug auf die Fusee Engine zu legen. Das Kernziel ist die Analyse und Darstellung eines Entwicklungsprozesses, das Erstellen eines Konzeptes für das Fusee Authoring Tool und die Erläuterung der einzelnen Module eines solchen Software Projektes.

1.3 Verwendete Methodik

Durch die Analyse verschiedener Referenzmodelle sowohl im Prozess und Projektmanagement als auch durch Analyse von Referenzsoftware werden ver-

[8] Eine Software welche von einem anderen Stück Software umgeben wird.

schiedene Aspekte des Game Authoring herausgearbeitet und in einer an die Zielsetzung dieser Arbeit angepassten Reflektion für den Leser wieder gegeben. Durch Konzeption mit anerkannten Methoden aus dem Software Engineering (Verwendung der UML Notation und verschiedener Kreativitätstechniken) entsteht ein Konzept für eine Software Bibliothek deren Ziel das Abbilden von Authoring Software Funktionalität darstellt. Das im Anschluss durchgeführte Prototyping prüft die Umsetzbarkeit der erstellten Konzeption und schafft eine Basis um zukünftig auf diese Arbeit aufbauende Projekte zu vereinfachen.

1.4 Forschungsstand

Ein wichtiges Literarisches Werk für diese Arbeit ist Game Engine Toolset Development von Graham Wihlidal [Wih06]. Wihlidal beschreibt hier verschiedenste Konzepte des Tool Developments. Er beschäftigt sich mit theoretischen und praktischen Aspekten. Er beschreibt die Prozesskette im Game und Tool Development und führt den Leser durch die verschiedenen Phasen des Tool Developments. Im weiteren Verlauf seines Buches beschreibt er technische Konzepte für die Entwicklung von Tools. Er erläutert einige Optimierungsmöglichkeiten und gibt dem Leser verschiedenste Werkzeuge Techniken zur Verbesserung der Performance eines Tools an die Hand. Jeglicher Beispielcode des Werks ist in C# geschrieben. Das Werk kann als Überblick des Tool Developments betrachtet werden welches in manche Bereiche etwas tiefer in technische Konzepte vorstößt. Wihlidal stützt die Analystischen Ergebnisse der Prozessentwicklung in dieser Arbeit und wurde für verschiedenste Aspekte des Systemdesigns herangezogen. Seine Ausführungen zu den best Practice Beispielen unterstützen die Konzeption des praktischen Teils.
Das zweit wichtigste Werk der Arbeit ist Heather Chandlers “The Game Production Handbook” [Cha06]. Sie behandelt in diesem Werk den Prozess der Game Production ausführlich. Das Buch bietet einen Überblick über die Struktur eines Entwicklerteams, der Rollenverteilung des Teams und Formaler Prozesse im Projektmanagement. Diese Ausführungen liegen dem analytischen Teil dieser Arbeit zu Grunde und stützten den Aufbau der Projektmanagement Prozesse und der Team Strukturen. Chandler gibt außerdem einen Überblick über die verschiedenen technischen Bereiche einer Produktion. Weiterhin wird der Produktionszyklus eines Spiels in seiner Gänze behandelt.
Das dritte wichtige Werk für diese Arbeit ist “The Game Asset Pipleine” von Ben Carter [Car04]. Seine Ausführungen zum Asset Management unterstützen den konzeptionellen Teil dieser Arbeit. Weiterhin wurde das Werk während der Analyse anderer Editoren zu Vergleichszwecken herangezogen. Carter bie-

tet einen wertvollen Einblick in das Management jeglicher Asset Datentypen. Weiterhin erläutert er die Wichtigkeit und verortung von Assetmanagement System im Tool Development.

1.5 Verwendete Software zur Erstellung dieser Arbeit

- Microsoft Visual Studio 2013,
  verwendet als Entwicklungsumgebung für das Softwareprojekt. Sowohl in der Professional als auch der Community Edition. Zu Beziehen unter https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx.
- Die Erweiterung ReSharper in Version 7.1 für Visual Studio 2010 http://www.jetbrains.com/resharper.
- Umlet http://www.umlet.com/
  Ein kostenloses Tool um Schaubilder und UML Diagramme zu erstellen.
- GitHub for Windows und die Git Shell www.github.com
  Verwendet als Versionskontrollsystem.
- TexWorks und MikTex www.tug.org/texworks/
  LaTeX- Editor und "Compiler".
- Photoshop von Adobe in der Version CC2014.
- Adobe Acrobat Reader in der Version 2015.007.20033. Kostenfrei herunterzuladen unter https://get2.adobe.com/reader/.

1.5.1 Die Fusee Engine

Bei der Furtwangen University Simulation and Entertainment Engine (kurz Fusee) handelt es sich um eine akdemische 3D Engine. Sie eignet sich vor allem für das Entwickeln von Interaktiven 3D Applikationen und wird in Forschung und Lehre eingesetzt. Die Engine wurde von Herrn Prof. C. Müller an der Fakultät DM in Zusammenarbeit mit einer Studentischen Projektgruppe entwickelt. Seit dem wird die Engine stetig als Open Source Engine von Studierenden in Projekten und Abschlussarbeiten weiter entwickelt.

Eine Besonderheit der Engine ist die Möglichkeit Applikationen für Web Browser zu bauen ohne auf die Nutzung von Plugins angewiesen zu sein. Mit Hilfe von Crosscompilern (hier der Compiler JSIL) kann der C# Code der Engine in Javascript übersetzt werden. Vgl. Gärtner, Fabian Müller, Christoph. »Student

Project Portable Real-Time 3D Engine«. In: *EUROGRAPHICS2014* (2014). Education Paper. URL: http://fusee3d.org/wp-content/uploads/2014/04/FUSEE_Educational.pdf (geprüft am 07. 04. 2015)

Fusee ist kompatibel zur Open Source .Net Variante Mono, kann auf Windows Systemen aber auch ohne Einschränkungen mit Microsofts .Net arbeiten. Auf die Fusee Engine wird in dieser Arbeit häufig Bezug genommen, da einige Bestandteile des Fusee Projektes essentiell für den praktischen Teil dieser Arbeit und die Fragestellung sind.

2 Tool Development in internen Entwicklerteams

2.1 Definition: Tools und Toolsets

Die Werkzeuge eines Entwickler Teams können übergreifend in Tools und Toolsets klassifiziert werden. Während es sich bei Tools um eigenständige Software handelt die meist zum Lösen einer spezifischen Aufgabe entwickelt oder eingekauft wurde, handelt es sich bei Toolsets um eine in sich abgestimmte Kollektion von Werkzeugen die zusammen genommen die Produktions Pipeline des Produktes darstellen. Die Komplexität von Tools reicht von einfach Text Editoren bis hin zu Modeling und Level Design Software. Tools können oft für eine spätere Aufgabe wiederverwendet werden. So genannte "Throw-away tools" (z. Dt. Wegwerf-Tools, meist durch schnelle Tool Hacks entwickelt) wurden meist speziell für eine Aufgabe angefertigt ohne dabei die Wiederverwendbarkeit im Auge zu behalten. Vgl. [Wih06, S. 3]
Oft ergibt sich aber aus ehemaligen Wegwerf-Tools oft auch die Chance ein Werkzeug zu entwickeln dass den Entwicklern längere Zeit treue Dienste leistet.

2.2 Internes Tool Developing oder Tool Licencing?

Internes Tool Development ist ein wichtiger Aspekt im Team eines Games und Software Entwicklerteams. Erich Bethke [Erik Bethke. *Game Development and Production*. Wordware Publishing Inc.; Auflage: Pap/Cdr (Februar 2003), 2003. ISBN: 978-1556229510] berichtet davon, dass Michael Abrash[1] ihm erläuterte, "dass 50% der Entwickler Arbeit bei idSoftware in das Tool Development fliesse." vgl. [Bet03, S. 44]. An der Relevanz des Themas hat sich trotz des zurückliegenden Zeitraums (Jahr 2003) kaum etwas getan. Sony hat für den Release der Playstation 4[2] ein Development Kit[3] für die internen Entwickler

[1] Ehemals idSoftware, ehemals Valve VR, aktuell Chief Scientist bei Oculus https://www.oculus.com/company/

[2] Playstation 4 - Erschienen im Herbst 2013

[3] Will Freeman. *Kingdom Come: Deliverance Video Update 9*. Available online http://www.develop-online.net/tools-and-tech/inside-the-system-sony-s-internal-console-tools-team/0191319. 2014. (Geprüft am 07. 04. 2015).

Studios erstellen lassen, welches bereits während der Planung und Entwicklung der Konsole entwickelt wurde. Sony hat diese Prozedur perfektioniert und lässt die eigenen Tools sogar in einem eigens dafür gegründeten Unternehmen für die eigenen Studios erstellen [4]. Sony hat im Herbst 2014 den für Playstation 3 Spiele eigens intern entwickelten Welt Editor "Level Editor"[5] als Open Source Software veröffentlicht und für Jedermann auf GitHub verfügbar gemacht. Der Editor kommt ohne direkten Enginebezug aus und lässt sich somit für verschiedenste Projekte der Sony Studios anpassen. Das ATF Framework, auf welchem viele interne Tools von Sony basieren[6], kann von Sound Editoren über Cinematic Editoren bis hin zu State Machine Visualisierungen genutzt werden.Eine Übersicht vonn Tools, welche das ATF Framework erfolgreich verwendet haben findet sich unter https://github.com/SonyWWS/ATF/wiki/ATF-Gallery. Natürlich ist hier trotzdem noch ein gewisser grad an Aufwand zu betreiben, aber durch das integrierte ATF Framework werden viele Bereiche mit wiederverwendbarem Code abgedeckt. Sony Hauseigene Entwicklerstudios haben ebenfalls ihre eigenen Tool Kits und Editoren auf dem von Sony bereitgestellten ATF Framework und "Level Editor" erstellt um Spiele wie Naughty Dogs Uncharted[7], Guerilla Games' Killzone Serie[8] oder Quantic Dreams Beyond:Two Souls[9] zu erstellen. Eine Übersicht der Studios welche das ATF Framework verwenden findet sich unter dieser Adresse https://github.com/SonyWWS/ATF/wiki/ATF-Adoption. Dieser große Einfluss des Frameworks zeigt, dass selbst in großen - und kleineren - Studios immernoch Bedarf nach einfach und schnell zu erweiternden Frameworks und Editoren besteht. Das ATF Framework bzw. der "Level Editor" von Sony waren auch eine Inspiration das das praktische Projekt zu dieser Arbeit zu konzipieren.

Durch Betrachtung der hier erwähnten Unternehmen und der Methoden der einzelnen Studios, lässt sich erkennen, dass die Größe des Studios bei der gewählten Methode der Entwicklung eine gewisse Rolle spielt. Große Unternehmen wie Sony Entertainment können es sich finanziell leisten ein gesondertes Studio für die Tool Entwicklung zu betreiben oder gleich Third Party Software

[4] SNSystems http://www.snsystems.com/

[5] Alex Wawro. *Sony releases level editor that's open source and engine-agnostic*. Available online http://www.gamasutra.com/view/news/224682/Sony_releases_level_editor_thats_open_source_and_engineagnostic.php. 2014. (Geprüft am 07. 04. 2015).

[6] Hier ein Einführungsvideo: https://www.youtube.com/watch?v=aU-9vzFELxc

[7] NaughtyDog. *Uncharted*. http://www.naughtydog.com/games/uncharted_drakes_fortune/. 2007, 2009, 2011. (Geprüft am 20. 05. 2015).

[8] Guerrilla Games. *Killzone 1, Killzone 2, Killzone 3, Killzone: Mercenary, Killzone: Shadow Fall*. http://www.guerrilla-games.com/. 2004, 2007, 2011, 2013, 2013. (Geprüft am 12. 05. 2015).

[9] Quantic Dream. *Beyond: Two Souls*. http://www.quanticdream.com/en/. 2013. (Geprüft am 15. 05. 2015).

zu lizenzieren. Kleinere Entwickler verteilen die Aufgaben meist an interne Mitarbeiter oder lizenzieren Third Party Tools. Im Akademischen Sektor an der Hochschule Furtwangen, in welchem Fusee eingesetzt wird, wäre es am effizientesten bei der Wahl von Tools auf Open Source Software oder selbst entwickelte Tools zurückzugreifen. Das Konzept welches im Rahmen dieser Arbeit erstellt wurde, kann zukünftige Projektgruppen beim Erstellen von Fusee Applikationen und Tools unterstützen und fördert somit das interne Tool Development. Somit haben auch Entwickler innerhalb eines Projektes, sollten sie gleichzeitig Tool Entwickler und Stakeholder (Nutzer) des Tools sein, einen doppelten Einfluss auf die Qualität und den Projektverlauf des Tool Development.

2.3 Organisationsstrukturen und Rollenverteilung

Wihlidahl [Wih06, S. 5] beschäftigt sich mit verschiedenen Organisationsmodellen des internen Tool Development. Hierbei geht es darum die Verantwortlichkeiten für Entwicklung und den Support von internen Tools festzulegen. Er klassifiziert folgende Modelle:

- Dedicated Tools Team
- Developer Ownership
- Game Team Develops - Tool Team Supports
- Engine Team Develops - Game Team Supports
- Content Team Develops and Supports

Für die Umsetzung der Konzeption dieser Arbeit und die Akademische Entwicklung mit der Fusee Engine empfehlen sich die folgenden zwei Strukturen. Mit ihnen können auch relativ kleine Teams erfolgreich im Bereich des Tool Development agieren, da kein großer Overhead im Bereich des Management oder Personals impliziert wird.

2.3.1 Dedicated Tools Team

Wihlidal grenzt die verschiedenen Strukturen der Tool Entwicklung ab und beschreibt hierbei verschiedene Vorgehensweisen zur Konzeption und Umsetzung von Tools durch interne Teams. Diese Art der Struktur vertraut einem Team die komplette Konzeption und Umsetzung des Tools an. Diese Art der Arbeit verlangt Skills im Bereich der Entwicklung aber auch im User Interface Design. Das Team bietet internen Support für das Tool. Vgl. [Wih06]. Diese Art der Teamorganisation bietet sich besonders für Projekte und Teams an

die es sich zum Ziel gesetzt haben ein Werkzeug für die ContentCreation oder eine ContentCreation Software zu entwickeln. Dieses Profil passt sehr gut auf die bisherigen Teams welche sich mit dem Fusee Projekt auseinander gesetzt haben und wird auch in Zukunft sicher nützlich sein.

2.3.2 Content Team Develops and Supports

Wihlidal beschreibt hiermit eine art der Organisation die vor allem kleine Teams unterstützt. Die Content Entwickler des Projekts entscheiden welche Tools sie für das effiziente Arbeiten am Projekt benötigen und designen und setzen diese diese selbstständig um. Diese Art der Entwicklung kann die Produktivität des Teams unterstützten. In kleinen Teams in welchen vor allem während der Entwicklung immer wieder Ressourcen frei werden (z.B. könnten Engine Entwickler zur Projektlaufzeit freie Ressourcen investieren so lange nicht an der Optimierung der Applikation entwickelt wird) lässt sich diese Variante gezielt einsetzen um den Overhead gering zu halten und die Entwickler stetig im Projekt einzubinden. Vgl. [Wih06].

2.4 Asset Pipelines

Artists produzieren Assets für das Projekt. Um diese in das Spiel zu übertragen bedarf es einer so genannten Asset Pipeline. Bei dieser Asset Pipeline handelt es sich um eine angepasst Zusammenstellung mehrerer Entwicklertools. Das Ziel der Asset Pipeline beschreibt [Car04] mit:

> "Quite simply, the term describes the sequence of processes that takes assets from their source form [...] to the final data that can be burned onto a disc or cardridge to form part of the finished game." [...] It is comparatively rare that fully working game disc with all the assets needs to be produiced. It is certainly an event which happens more frequently as the game gets closer to being completed [...]. Ben Carter. *The Game Asset Pipeline*. Delmar Cengage Learning, 2004. ISBN: 1-58450-342-4, S. 6

Somit ist nach Carter die Assetpipeline ein wichtiges Instrument und besteht meist aus einer Zusammenstellung von eigens entwickelten Tools und lizenzierter Software wie z.B. Game Einengens, Modeling Editoren wie 3DS Max, Cinema 4D, oder Maya und weiterer Software.
Diese Arbeit beschäftigt sich also mit einem Teilgebiet der Asset Pipeline. Genauer gesagt mit dem Ziel der Asset Pipeline, dem Welt Editor. In einem späteren Kapitel wird das Fusee Asset Management betrachtet und die Asset

Pipeline für das Entwickeln einer Fusee Anwendung mit Cinema 4D konzeptionell untersucht. Hierbei wird untersucht, ob es denn möglich ist die Wege von Content Creation Tools wir Cinema 4D derart zu verkürzen, dass eben nicht nicht die Assets in diesen Editoren erstellt werden, sondern auch die Asset Pipeline und der Welt Editor darin integriert sind.

2.4.1 Der Einfluss der Asset Pipeline auf die Tool Entwicklung

Um eine Produktion erfolgreich zu starten, bedarf es einer Asset Pipeline und der dazugehörigen Tools. Um zu untersuchen, welche Tools für das Projekt nötig sind, sollten sich Lead-Developer eines Spielesoftware Projektes zu Beginn des Arbeit folgende von Chandler in [Cha06, S 223-224] beschriebene Fragen stellen.

- Welche Werkzeuge und Software wird benötigt?
- Ist es möglich Assets in das Game Format und wieder zurück in das Ausgangsformat zu konvertieren? (Zwei Wege Funktionalität)
- Gibt es irgendwelche Engpässe? Kritische Konzepte im Game Design?
- Wann muss das Tool umgesetzt sein?
- Wie werden Assets im Tool verwaltet? (Entscheidung für eine "Version Control Software")
- Welche Teile eines Prozesses, einer Software können automatisiert werden?

Vgl. [Cha06, S. 224-225]
Durch das Beantworten dieser Fragen, kann sicher gegangen werden, dass die meisten wichtigen Aspekte berücksichtigt wurden. Dieser Prozess wird für das FuseeAT Beispielhaft durchgeführt. Die Analyse findet sich in Abschnitt 4.2 dieser Arbeit.
Carey Chico[10] (als Experte im Werk von [Cha06] vertreten) beschreibt die Relevanz einer guten Asset Pipeline und vor allem des dedizierten internen Tool Developments wie folgt:

> One of the necessities of game development is a solid tool stategy. You must have a core group of engineers who are dedicated to tools programming on your team. They can enhance the proprietary tools that are part of your pipeline by upgrading features [...]

[10] Aktuell Präsident bei Zero Mass Enery https://www.linkedin.com/in/careychico, ehemals Pandemic Studios

> and adding new features based on the game development needs. [...] Because efficient game production depends on creating assets quickly, the developers are constantly thinking of ways to use tools to speed up the asset production pipeline [...]. The longer it takes an artist to get an asset frim source art to an asset that can be seen in game, the less they want to deal with the process, and the lower the quality. [...] All pertinent people on the team must be able to access, manipulate, modify and change the content in the game simultaneously or equally. [Cha06, S. 224-225]

Hiermit zeigt Chico ein Kernthema dieser Arbeit auf: Einige auf Tools fokussierte Mitarbeiter unterstützen den Produktionsprozess einer Spiele Software positiv. Tools die auf bereits bekannter Software basieren sind wichtig. Ihrer Bedienung ist leicht zu verstehen und sie stehen jedem Mitglied des Teams zur Verfügung. Solche Authoring Tools könnten den Produktionsprozess zugänglicher und effizienter gestalten. Da eine Erweiterung eines intern entwickelten Tools durch die Entwickler dauerhaft möglich wäre, kann auch ein langfristiger Einsatz dieser Tools gewährleistet werden und rechtfertigt somit eine aufwändigere Entwicklungsarbeit und ein dediziertes Entwicklerteam. Die Eckpfeiler dieser Tools wiederum ergeben sich aus der Asset pipeline der Entwickler, Designer und Artists. Somit ergibt sich eine enge Beziehung zwischen Third Party Software und den eigenen Tools. Womit wieder der Kern dieser Arbeit angesprochen wird. Eine noch engere Beziehung zwischen den Third Party Content Creation Applikationen und eigenen Tools. Somit wäre es auch nach Chico sehr praktisch die Editoren der Game Engine direkt in die Content Creation Software wie Cinema 4D zu integrieren. Ein kürzerer Weg kann dann nicht mehr erreicht werden.

3 Produktionsprozess: Projektplanung und Requirements Analyse

3.1 Projektmanagement für Game Authoring und Tool Development

Um einen Entwicklungsprozess abzubilden und Tools für Entwickler, sogenannte Authoring Tools oder Developer Tools, zu entwickeln bedarf es einer gewissen Organisation. Im Bereich der modernen Spieleentwickelung in kleinen bis mittleren Unternehmen (seltener bei großen AAA Produktionen [1]) wird hierfür meist ein agiles Modell zur Softwareentwicklung eingesetzt. Das hier vorgestellte und analysierte Scrum Modell zeichnet sich besonders durch geringen management Overhead und eine sehr schnelle Reaktionszeit auf Probleme und Änderungen während des Projektes aus.
Das es sich bei dem Projektmanagement um einen wichtigen Teil des Entwicklungsprozesses handelt, soll hier nun eine Empfehlung für ein agiles Modell herausgearbeitet werden. Diese soll sich speziell an kleinen Teams orientieren. Ein wichtiger Aspekt ist der meist vorhandene Zeitdruck in der Tool Entwicklung.

3.1.1 Agiles Modell oder Wasserfall Modell?

Viele Entwickler (z.B. Ubisoft, siehe [Sch14][2]) setzen heute auf moderne Modelle zum Entwickeln von Software. Die so genannten agilen Modelle (wie Beispielsweise Scrum, Extreme Programming und Feature Driven Development) ermöglichen meist das schnelle (agile) reagieren auf plötzlich auftauchende problematische Situationen während einer Entwicklung. Schwerfälligere Modelle wie das Wasserfallmodell, Spiralmodell (weiter entwickeltes iterativ orientiertes Modell) haben hier meist Probleme durch ungleich höheren Mana-

[1] Allgemein: Hochqualitative Spiele Software mit großem Entwicklungsbudget und einer Breiten Zielgruppe. Vgl. [Cif05]

[2] Artikel der Zeitschrift Making Games, verfügbar unter http://www.makinggames.biz/features/projektmanagement-mit-scrum,2534.html, zuletzt geprüft am 12.05.2015.

gement Overhead, ihrer Komplexität und sie benötigen ein Zeitaufwändigeres re-iterieren im Falle von Updates und Umstrukturierungen durch unvorhergesehene Ereignisse und Probleme. Hochkomplexe Software Projekte die über längere Zeiträume entwickelt werden können meist nur durch stark strukturierte Modelle wie das Wasserfall Modell überblickt und erfasst werden. Allerdings bedeutet der zusätzliche Bürokratische Mehraufwand auch oftmals einen erhöhten Overhead im Personal-, Software- und Knowledge-Bereich. Es ist im Fall des schnell-lebigen Tool Developments also geschickter, sich mit einem ebenso schnelllebigen und agilen Projektmanagementmodell wie Scrum zu organisieren. Besonders im Falle von Scrum ist der Overhead der Management Komponente gering und das Modell lässt sich schnell adaptieren und in den Arbeitsprozess eines Teams integrieren.

3.1.2 Das Scrum Modell

Der Scrum Prozess ist ein empirisches Modell des Projektmanangements und tauchte das erste mal in der Veröffentlichung Ikujiro, Nonaka Hirotaka, Takeuchi. *The new product development game*. Available online https://hbr.org/1986/01/the-new-new-product-development-game. 1986. (Geprüft am 15. 04. 2015) auf - damals nicht unbedingt in der Software- sondern der allgemeinen Produktentwicklung angesiedelt. Seitdem hat sich das Modell weiter entwickelt und erfreut sich bei innovativen Softareprojekten im Games und Indie-Games Bereich (auch und meist wohl auch vor allem im Tool Development) sehr großer Beliebtheit. Die Entwickler CCP und Warhorse Studios hatten hierzu eigene Videos und Artikel veröffentlicht in welchen sie die Vorzüge des Systems und die Integration im eigenen Entwicklungsprozess präsentieren. Hierzu veröffentlichte das Studio CCP einen kompletten Talk der Hauseigenen Konferenz unter CCP Games. *Eve Online Fanfest 2009 - Scrum and Agile development processes*. Available online https://www.youtube.com/watch?v=GqsReCZD4hc. 2009. (Geprüft am 07. 04. 2015). Das Unternehmen Warhorse Studio veröffentlichte zwei Entwicklertagebücher. Beide thematisierten den Umstieg auf Scrum und die daraus gewonnene Effizienz für das Teams unter Christopher Schmitz. *The art of waiting*. Available online http://www.warhorsestudios.cz/index.php?page=blog&entry=blog_031. 2013. (Geprüft am 07. 04. 2015) und Daniel Vavra Martin Klekner Votja Nedved. *Kingdom Come: Deliverance Video Update 9*. Available online https://www.youtube.com/watch?v=LfklQR-36-8. 2014. (Geprüft am 07. 04. 2015).
Das Scrum Modell basiert nach Ken Schwaber [Ken Schwaber. *Agile Project Management with Scrum*. Micorosoft Press, 2004. ISBN: 073561993x, S. 11.] auf dem Prinzip der empirischen Prozess Kontrolle. Er beschreibt drei Standbeine

dieser empirischen Kontrolle:

- Visibility
- Inspection
- Adaptation

Visibility: Alle Aspekte der Prozesse welche den Erfolgreichen Ablauf des Projekts beeinflussen müssen von denjenigen verständlich sein, die in den Prozess eingebunden sind. Das bedeutet in erster Linie, dass verschiedene Stati eines Projektes genau definiert sind und für jeden einsehbar und verständlich formuliert.
Inspection: Jeglicher Fortschritt des Projekts muss immer wieder geprüft werden um Probleme und Abweichungen im Projekt frühzeitig zu erkennen und diesem entgegenwirken zu können. Hierbei ist es wichtig, dass der Inspektor der die Inspection durchführt die Sache durchdringen kann welche er versucht zu untersuchen.
Adaption: Sollten verschiedene Aspekte des Projekts das positive Endergebnis negativ beeinflussen so sind Änderungen im Prozess und Projekt nötig um dem entgegen zu wirken.
Erläuterungen entnommen, übersetzt und angepasst aus [Sch04, S. 11].

Scrum eignet sich besonders für unkomplizierte und schnelle Prozesse. [Cha06] beschreibt es wie folgt:

> "It is relatively easy to implement as it requires no formal training, only a commitment by the team to use the process." [Cha06, S. 45]

> "The basics of scrum involve creating subsets of self-directed teams within the larger project team [...] and work toghether to complete a set of tasks that will result in a tangible deliverable at the end of a set period of time." [Cha06, S. 45]

Diese Struktur der kleinen Teams im Team lassen sich auf das Prinzip des internen Tool Development Teams anwenden, da sich schon die Strukturen gleichen.
Ein Scrum Entwicklerteam ist mit folgenden Rollen besetzt:

- Product Owner - Verwaltung der Tasks, vertritt sämtliche Stakeholder (in diesem Fall meist das interne Developer Team, steht für Rückfragen und Kommunikation nach "außen" zur Verfügung.)

- Entwicklungsteam - Das tatsächliche Team.
- Scrum Master - Überblickt die Arbeit des Teams, koordiniert und räumt Hindernisse die den Entwicklungsprozess aufhalten aus dem Weg.

Scrum kann innerhalb eines Projektes und Teams beliebig heruntergebrochen werden, bis die gewünschte Größe eines Entwicklerteams erreicht wird. Externe Rollen wie Stakeholder etc. verlagern sich somit auf andere interne Projektleiter oder Teammitglieder. Aus diesem Grund ist das Model gut für die Entwicklung von Development Tools und Toolkits geeignet. Mit Hilfe des Models, können benötigte Toolkits während einer Projektlaufzeit schnell und effizient entwickelt werden ohne dass ein schwerfälliger Managementprozess die Entwicklung blockiert. Somit ergänzt sich dieses Modell gut mit dem doch eher agilen entwickeln von Developement Tools während der Projektlaufzeit - denn in den seltensten Fällen wurde vor dem Beginn des Projekts daran gedacht alle nötigen Tools bereitzustellen. Oftmals ergeben sich auch während der Entwicklung neue Herausforderungen für das Team welche nach neuen Tools oder Features für bestehende Software verlangen.
Hier soll nurn kurz ein Szenario aufgebaut werden, welches das Tool Developement Team eines aktiven Software Entwicklers beschreibt. Zuerst einmal sollen die Rollen verteilt werden. Dieses Szenario erläutert vor allem die Personalstruktur der Teams während einer Tool Entwicklung.

- Product Owner - Meist der leitende Entwickler des Software Projektes. In diesem Fall meist ein Producer und/oder Game Developer.
- Entwicklungsteam - Das Tool Development Team selbst.
- Scrum Master - Die leitende Person des Tool Development Teams, bzw. sollte sich das Team sehr nah am Scrum Modell bewegen, dann meist ein Entwickler außerhalb des Teams aber mit guten Kontakten zum Team selbst und erhöhter Erreichbarkeit.

Es kann hierbei auch von Vorteil sein, das Tool iterativ in den Arbeitsalltag des Teams zu integrieren um die Entwickler nicht durch einen Berg an neuen Features zu verunsichern und so die Einarbeitungszeit möglichst gering zu halten.
Es soll hier an einem kurzen Beispiel deutlich gemacht werden, wie ein solches Tool eingeführt werden könnte. Der Prozess der Entwickler kann mit Scrum wie unten in der Auflistung dargestellt ablaufen. Jeder Schritt wird in Sprints eingeteilt und enthält gleichzeitig tägliche Stand Up Meetings. Sollte der Sprint

eines Punktes erfolgreich abgeschlossen sein, so wird der nächste Schritt angegangen.
Szenario: Ein Team benötigt einen Textur-Editor / Tool um Texturen in das Format der Game-Engine zu transformieren.

- Der Antrag für ein spezifisches Tool vom Producer/Entwickler/oder anderen Personen wird gestellt.
- Das Tool wird bewilligt und das Tool Development Team wird beauftragt.
- Das Team entwickelt designed das Tool und implementiert Basisfunktionalität.
- Das Tool wird mit der Basisfunktionalität an das Produkt Team herausgegeben.
- Die fehlenden Funktionen werden implementiert.
- Das Tool wird mit der erweiterten Funktionalität herausgegeben.
- Es wird mit dem Produkt Team Rücksprache gehalten, welche Funktionen noch benötigt werden.

Zusammengefasst kann der Scrum Prozess besonders aufgrund seiner empirischen Natur und der schnellen Reaktionszeit des Teams auf negative Entwicklungen für das Produkt empfohlen werden. Die schnelle und leicht verständliche Struktur dieses Managementprozesses kann jeder Mitarbeiter schnell verstehen und in den Alltag seiner Tätigkeit als Entwickler integrieren. Der Prozess ist sehr transparent und bezieht schon alleine durch seine Struktur jederzeit alle notwendigen Stakeholder und Mitarbeiter in die Entwicklung mit ein. Dies alles soll nicht bedeuten, dass andere Prozesse in gewissen Situationen nicht auch zum Erfolg einer Entwicklung beitragen können. Der hier vorgestellte Prozess lässt sich aber ganz besonders gut in das Entwickeln mit der akademischen Fusee Engine an der Hochschule Furtwangen integrieren. So wurde in etwa das Fusee Projekt Uniplug, auf welches in einem späteren Kapitel noch genauer eingegangen wird, mit Hilfe eines Scrum Prozesses Entwickelt.

3.2 Mitglieder eines Entwicklerteams

Es soll hier ein kurzer Überblick über die gängigsten Mitglieder eines Entwicklerteams gegeben werden. Die Mitarbeiter eines Spieleentwicklers werden für

diese Arbeit in die folgenden Gruppen aufgeteilt werden - Artist, Designer, Engineer und Producer. Jede Gruppe arbeitet hierbei meist interdisziplinär mit anderen zusammen, kümmert sich aber doch um die ganz eigenen Bestandteile eines Produktes. Es ist durchaus so, dass jede Gruppe ihre eigenen Tools und Methoden verwendet. Dieser Ansatz wird in der Konzeptionierung dieser Arbeit aufgegriffen und weiter verfolgt.
Bei der Bezeichnung und Aufteilung der verschiedenen Teammitglieder in Fachbereiche orientiert sich diese Arbeit am Werk von Heather Chandler. *The Game Production Handbook*. Thomson Delmar Learning, 2006. ISBN: 1-58450-416-1, in welchem er die Produktionsprozesse eines Spiels sowohl in einer designorienten als auch technischen Herangehensweise diskutiert.
Diese Auswahl beschränkt sich auf Mitglieder des Teams welche mit dem Entwicklungs Prozess des Tool Authorings mehr oder weniger direkt in Verbindung treten.
Grundsätzlich ist eine solche Analyse auch für das Tool Developer Team sinnvoll. Während der Konzeption eines Tools muss darauf geachtet werden, welche Gruppen von Entwicklern mit welchem Grad an technischem und gestalterischem Verständnis mit dem Tool umgehen sollen. Daher ist es essentiell über die Strukturen des eigenen Entwickler Teams in Kenntnis zu sein. Vor allem sollte die topologische Verteilung aufgrund von Stakeholderanalysen während der Tool Entwicklung bekannt sein.

3.2.1 Producer

Der Producer (Produzent) ist für gewöhnlich bereits einige Jahre in der Industrie als Entwickler tätig gewesen bevor er diese Position einnahm. Er ist meist verantwortlich für ein Projekt und das management des Entwicklerteams. Zu seinen Aufgaben gehört die Überwachung des Projektverlaufs, das einhalten der Deadlines und des Budgets. Producer kümmern sich in erster Linie um den Ablauf des Alltagsgeschäftes (der Entwicklung) und nicht um die Kreativen Aspekte des Projekts. Produzenten können ihren Fokus auf viele Aspekte des Projktes legen. Meist treten sie als Developer Producer (DP) und Publisher Producer (PP) auf. Als DP sind sie meist in den gesamten Tagesablauf der Entwicklung eingebunden und beteiligen sich auch an den eigentlichen aufgaben der Entwicklung während sie als PP vor mit externen Entwicklern arbeiten und die Interessen des Publishers vertreten und nicht besonders stark in den Entwickleralltag eingebunden sind. Vgl. Heather Chandler. *The Game Production Handbook*. Thomson Delmar Learning, 2006. ISBN: 1-58450-416-1, S. 19-20
DP können hier auch in das Entwicklen von Toolsets eingreifen. Sie koordi-

nieren die Kommunikation des Produktentwicklerteams mit dem Tool Team und stellen sicher, dass alle Feates des Tools für die alltägliche Produktion enthalten sind.

3.2.2 Artists

Artists sind in einem Games Projekt für jegliche grafische Repräsentation des Spiels nach Außen zuständig. Sie erstellen Modelle von Spielfiguren und Umgebungen und kreieren Texturen und User Interfaces in DCC Applikationen. Bei den Artists handelt es sich um eine wichtige Kerngruppe für diese Arbeit da sie einen Großteil der Arbeitszeit in den Authoring Tools und Editoren des Spiels verbringt. Artists können in mehrere Untergruppen aufgeteilt werden. Dies bedeutet jedoch nicht, dass jedes Unternehmen jede Artists Rolle beschäftigt. Oftmals übernehmen einzelne Mitarbeiter mehrere Rollen je nach dem Entwicklungsstand des Projekts.

Modeling/Animation Artist

Ein Animation Artist verbringt die meiste Zeit damit Animationen und Modelle (3D, 2D), kurz: Assets [3], für die Verwendung im Spiel vorzubereiten. Programme wie Cinema 4D[4], 3DS Max [Aut15a], oder Modo[The15] sind Beispiele für Kernsoftware dieser Entwickler. Der Vollständigkeit halber sei hier noch das Open Source Projekt Blender [Ble15] erwähnt.

World Builder / Level Designer / Environment Artist

Diese Artists zeichnen sich für das erstellen und gestalten von Welten und Leveln verantwortlich. Sie sind sowohl in 2D als auch in 3D Softwareprogrammen bewandert und verstehen sich nicht nur auf das ausgestalten von Leveln und Welten sondern auch auf das entwickeln der Levelstrukturen. Es handelt sich hierbei nicht immer um reine Gestalter, diese Position kann auch von Gamedesigner besetzt werden. Vgl. [Cha06, S. 24].

3.2.3 Designer

Designer (Gamedesigner) arbeiten eng mit Artists und Engineers zusammen. Meist Entwicklen Game Designer das Spielprinzip, den narrativen Raum des Spiels und das Regelwerk. Sie schreiben oft Skripte und kleine Implementierungen oder verbessern Grafiken oder Spielfunktionen und entwickeln User

[3] Assets sind Bestandteile des Produktes welche eine Grafische oder logische Repräsentation im Produkt erfahren. Dazu zählen z.B. Modelle, Texturen und Code Dateien.
[4] MAX15.

Interfaces welche von den Artists ausgestaltet werden. Sie verwenden Assets aus der Designabteilung und fügen diese mit Skripten zusammen. Spieltests werden von Ihnen überwacht um den Spielfluss und das Erlebnis des Rezipienten beim Spielen zu optimieren.

Level/World Designer

Level bzw. World Designer erstellen aus den erschaffenen Assets eine oder mehrere zusammenhängende Spielwelten - sogenannte Level. Diese Welten werden durch sie und weitere Artists mit Inhalt nach den Plänen der Game Designer gefüllt. Oft haben diese Welten einen gewissen gestalterischen Anspruch und von den Designern erwünschten Artstyle welche die Atmosphähre des Spiels repräsentiert. Meistens werden diese Welten in einem extra dafür geschaffenen Editor angefertigt und können nich in einem Modeling Tool wie Cinema 4D entwickelt werden. Ein Beispiel für solche Level Editoren ist der GTKRadiant[5] Editor für Spiele basierend auf der idTech3 und idTech4 Engine [6], beide als Open Source auf der Platform GitHub verfügbar. Weitere Beispiele sind der Level Editor von Sony, auf welchen diese Arbeit später noch eingeht sowie der Unity3d Editor. Der Unity3d Editor beinhaltet eine gesamte Game Engine, jedoch wird direktes Modeling und das erstellen von Texturen von Grund auf nicht unterstützt. Alle Assets, außer primitiver Geometrischer Objekte wie Würfel und Kugeln etc. müssen in externen Programmen erstellt und importiert werden. Vgl. [Cha06, S. 31]
Der konzeptionelle Teil dieser Arbeit wird später die versuchen einen Ansatz zu entwickeln der es ermöglicht zumindest einen Teil der Level und Welteditor Tools zu beseitigen. Somit könnten Level und World Designer und Environment Artists ihre Arbeit in die bereits bekannten Modeling Tools verlagern und so eine verbesserte Projektivität erreichen.

Scripter

Scripter sind meist dafür zuständig verschiedene Ereignisse in einer für die Game Engine extra entwickelten Script Sprache (oftmals auch einfache Adaptionen von Javascript oder Actionscript) zu beschreiben und so die Welt des Spiels interaktiver zu gestalten. Diese Aufgaben unterstützen die Spiellogik oder aber beschreiben die Funktionen ganzer Systeme wie z.B. die eines Aufgabensystems (Quest Systems) welches dem Spieler während des Spiels

[5] Open Source Projekt GTKRadiant http://icculus.org/gtkradiant/

[6] Beide Engines und weiterer Source Code von idSoftare herunterzuladen auf dem Account des Unternehmens auf GitHub unter https://github.com/id-Software - geprüft am 08.04.2015

mitteilt, was er in der Spielwelt zu tun hat. Scripter kümmern sich also um die Umsetzung von Gameplay relevanten Funktionen und sind selten an der Entwicklung des Engine Codes oder Tool Codes beteiligt.

User Interface Designer

User Interface Designer kümmern sich um das Erstellen von grafischen Schnittstellen welche die Interaktion mit dem Benutzer ermöglichen. Hierfür verwenden sie oft Scriptsprachen wie Actionscript von Adobe (Zur programmierung von Adobe Flash Interfaces) oder gar fertige Middelware wie Scaleform[7] ein Cross Plattform UI Solution Tool[8] von Autodesk. Diese Gruppe der Entwickler wird durch diese Arbeit nur sehr gering beeinflusst. In der Fusee Engine werden Interfaces über Code Dateien eingebunden und daher in externen Grafikprogrammen und Visual Studio angefertigt. Vgl. [Cha06, S. 31]

3.2.4 Engineer

Engineers (z. Dt. Ingenieure) arbeiten meist am Kern der Applikation und entwickeln den Source Code für die Anwendung, Engine, Netzwerkfunktionen, KI, und Tools. Diese Entwickler arbeiten hauptsächlich in einer IDE [9] wie Visual Studio (auf welches sich das zu dieser Arbeit konzeptionierte Tool bezieht) oder XCode [10]. Der in der IDE geschriebene Code wird dann von den Engineers selbst oder von Game Designer in der Engine verwendet. Hierbei kann sich das Tätigkeitsfeld ausweiten bis hin zur Entwicklung von Gamelogic [11]. Vgl. [Cha06, S. 26]

Tool Engineer

Tool Entwickler sind ein Kernpunkt dieser Arbeit. Sie sind verantwortlich dafür Werkzeuge für den internen Gebrauch zu entwickeln durch welche die anderen Teammitglieder ihre Arbeit effizienter gestalten können. Hierbei entwickelt ein kleines Team - meist während oder vor der eigentlichen Arbeit an einem Projekt die Tools für die restlichen Entwickler des Projektes. Diese Tool Palette kann von Textur-Editoren bis hin zu kompletten Welteditoren fast alles vorstellbare enthalten. Verschiedene Studios haben eigene Tool Developer Teams, welche sich nur um diesen Bereich des Produktes kümmern. Diese Teams betreuen

[7] Aut15b.
[8] Ermöglicht das erstellen von 2D, 2.5D und 3D Ui Elementen. Wird z.B. von der Unreal Engine 4 verwendet.
[9] Integrated Developement Environment
[10] X-Code ist nur für MacOSX erhältlich
[11] Logik des Spiels, ermöglicht das interagieren etc. mit und in der Software

auch meist den Modding Support für ein fertiges veröffentlichtes Produkt. Beispiele für Modding Tools sind z.B. das RedKit von CDProject Red für das Spiel The Witcher 1 und 2, der LevelEditor von Sony der in einer Open Source Version vorliegt oder das Creation Kit von Bethesda Softworks welches einen Modding Support für die Spiele der The Elder Scrolls Reihe bereit stellt. Vgl. [Cha06, S. 27]

Grahics Engineer

Computer Graphics Ingenieure beschäftigen sich meist mit dem Entwickeln der eigentlichen Engine (und den angrenzenden Teilgebieten) des Produktes. Oft sind Graphics Engineers aber auch an der Tool Entwicklung beteiligt. Gerade in kleineren Unternehmen könnten die eigentlichen Strukturen schnell aufbrechen um Synergien im Team zu nutzen. [Cha06, S. 27]

3.2.5 Der Analyse und Entwicklungsprozess des Tool Developments

Nach Wihlidal unterscheidet sich die Planung eines Projektes zur Erstellung eines Developer Tools nicht sehr von der Planung für die allgemeine Entwicklung von Software. Der Software Development Life Cycle (SDLC) besteht aus vier Planungsphasen. Der erste Schritt wäre eine allgemeine Planung des Tools. Diese beinhaltet Funktionen und Umfang des Tools. Eine Beschreibung der Anforderungen bzw. der Ziele des Projektes wird mit den begünstigten Entwicklern abgesprochen. Die zweite Phase beschreibt eine Bedarfsanalyse der Stakeholder. Es werden Arbeitsabläufe skizziert und mit den Beteiligten durchgesprochen. Je nach Komplexität und Relevanz des Projekts wird Software anderer Hersteller oder anderer Arbeitsbereiche ebenfalls analysiert und das Ergebnis zur Gesamtanalyse hinzugezogen. So könnte eventuell der Einsatz einer Drittanbieter Software in gewissen Situationen vorteilhafter sein als eine komplette interne Neuentwicklung. Diese Entscheidung ist aber sehr Situationsabhängig. Daraufhin folgt die Designphase in welcher das Entwicklerteam des neuen Tools das Requirements Engineering abschliesst und mit dem Software-/Systemdesign beginnt. Hier wird das Produkt in Form von UML Diagrammen und Veranschaulichungen entwickelt. Die tatsächliche Implementierung folgt als letzte Phase des Projektablaufs. Während der Implementierung kann aufgrund der Verwendung des agilen Scrum Systems immer schnell auf Hindernisse und Wünsche der Stakeholder reagiert werden. Vgl. [Wih06, S. 37-40]
Auch wenn die hier erwähnte Methodik einige Aspekte des Wasserfall Projektmanagement Models nachzeichnet ist es möglich diese Schritte in ein agiles

Modell wie Scrum zu integrieren. So kann das Modell z.B. um Iterationen bzw. Sprints erweitert werden. Im folgenden Abschnitt wird genauer auf die jeweiligen Schritte der organisatorischen Planung eingegangen.

Planung

Begonnen wird mit der Planungs- und Analyephase. Hier wird zuerst eine Requirementsanalyse [12] durchgeführt. Mit ihr sollen alle wichtigen Kerneigenschaften der Software identifiziert und niedergeschrieben werden. Ein an diese Arbeit angehängtes Designdokument führt diese Requirementsanalyse weiter aus. Die Anforderungsanalyse ist in einer solchen Situation, in welcher ein Produkt unter Zeitdruck für den Produktivbetrieb entwickelt wird, ein wichtiger Bestandteil der Projektplanung. Ein Tool, welches nicht den Anforderungen der Teammitglieder entpsricht kann nicht im Betrieb eingesetzt werden und verzögert im schlimmsten Fall die weitere Entwicklung der gesamten Produktentwicklung.

Requirements Analyse

Dier hier aufgeführten Punkte sind Anforderungen welche das zu konzipierende Tool erfüllen sollte. Diese Punkte werden im Requirements Dokumentations Dokument genauer betrachtet. Dieses Dokument findet sich im Anhang zu dieser Arbeit.

- Eine Software soll es ermöglichen, dass Artists, Designer und Developer an ein und dem selben Projekt arbeiten können ohne die gewohnte Arbeitsumgebung (3D-Modellierungssoftware, IDE) zu verlassen und etwas komplett neues (Level-Editor) zu erlernen.
- Das Produkt muss auf Basis der Fusee Engine entstehen
- Ziel ist es in Cinema 4D ein Fusee Projekt anzulegen, zu speichern und es zu öffnen
- Assets sollen ins Spiel integriert werden können die von Artists, Designern und Entwicklern bearbeitet werden können.
- Das Fusee C# Projekt sollte aus C4D heraus gebaut werden können.
- Eine Stakeholderanalyse schafft klarheit, welche Parteien des Teams mit dem zu erstellenden Tool arbeiten müssen.

[12] dt. Anforderungsmanagement

- Es ist zu analysieren, welche Schritte für welche Art der Arbeit des Teams notwendig sind. Hierzu werden Usecases der verschiedenen Rollen und Aufgaben erstellt.

Requirements Dokumentation

Während der Requirements Dokumentation werden alle Ansprüche an die Software dokumentiert und von den Stakeholdern geprüft. Über dieses Dokument lässt sich der spätere Funktionsumfang des Tools genaustens definieren und verfolgen. Das Dokument stellt in der Praxis ein rechtlich relevantes Dokument dar und ist aus diesem Grund nicht zu vernachlässigen. Ein Dokument mit Bezug auf den Praktischen Teil dieser Arbeit findet sich wie bereits erwähnt im Anhang.

System Design / System Modeling

Anschließend an die Analyse folgt das System Modeling in welchem die Anforderungen des Programs zu einem Softwareprodukt modeliert werden. Oft bedient sich das Entwicklerteam hierbei Notationen wie UML [13] um ihre Ideen und Entwürfe in ein allgemein verständliches Format von Diagrammen und Schrift Formen zu bringen. Das System Design ist ein kritischer Punkt. Hier müssen die Anforderungen des Kunden genau in die geplante Entwicklung des Systems übernommen werden. Oftmals arbeitet das System Design

Abgleich des System Designs mit den Anforderungen

Vor der Implementierung muss immer auch geprüft werden ob das geplante System aus dem Designprozess mit den tatsächlichen Anforderungen der Stakeholder zusammen passt. Hierfür werden die Konzeptionierten Software-Bestandteile mit der Zielgruppe diskutiert. Eventuelle Einwände und Vorschläge werden zum Konzept hinzugefügt und in die weitere Planung miteinbezogen.

Implementierung

Die Implementierung ist der praktische Schritt des ganzen Prozesses. Während der Implementierung wird das Tool entwickelt, auf Fehler geprüft eventuell bei Verwendung von iterativen Projektmanagementmodellen wieder durch iterierende Abläufe im Konzept verändert oder erweitert und anschließend korrekt implementiert.

[13] Unified Modeling Language

3.3 Stakeholderanalyse intern

Um herauszufinden, welche Entwickler eines Teams von der Entwicklung neuer Development Tools im Blick auf ein Spieleentwickler Unternehmen betroffen sind, wurde eine Analyse durchgeführt, um die Stakeholder innerhalb des Teams zu identifizieren. Stakeholder sind ein wichtiger Bestandteil des Prozesses der Softwareentwicklung aber warum ist das so und was genau sind Stakeholder?

> "Stakeholders are persons or organizations [...], who are actively involved in the project or whose interests may be positively or negatively affected by the performance or completion of the project." Project Management Institute. »A Guide to the Project Management Body of Knowledge«. In: *IEEE Std 1490-2011* (Nov. 2011), S. 23

Stakeholder sind für dieses Projekt also alle Personen und auch Organisationen die einen indirekten oder direkten Bezug zum Projekt haben (siehe Abbildung 3.1). Somit gehört zu den Stakeholdern auch das interne Entwicklerteam, welches nach Fertigstellung des Projektes mit dem entwickelten Tool arbeiten soll.

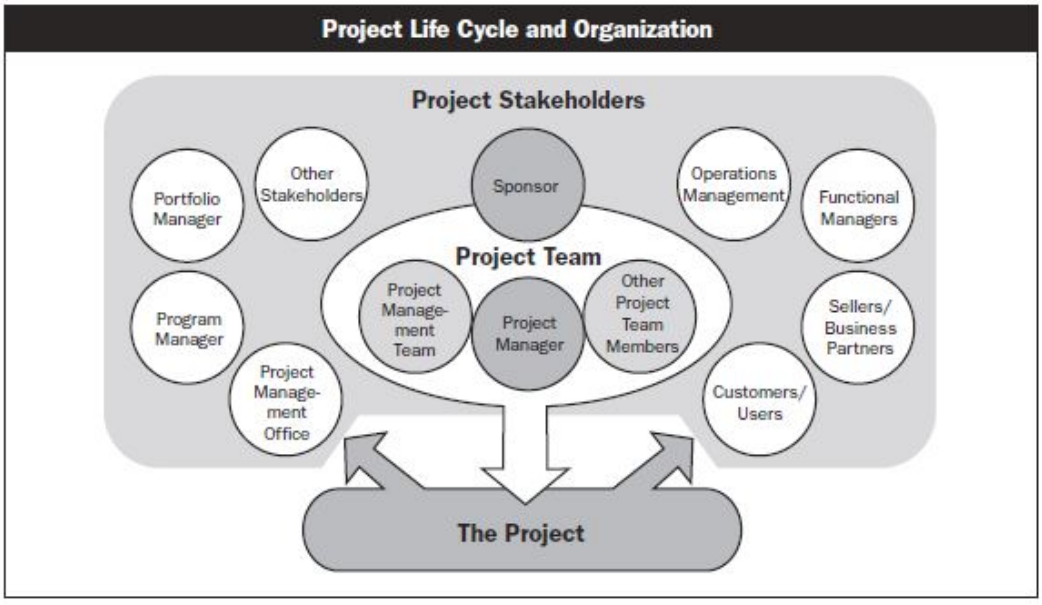


Abbildung 3.1: Überblick Stakeholder. Alle am Projekt beteiligten Personen oder Organisationen müssen beachtet werden. Grafik entnommen aus [Pro11, S. 24].

Wihlidal beschreibt ein weit verbreitetes Problem welches durch Fehler in der Stakeholder analyse entsteht:

> It is very important to ask the right questions to your stakeholders [...] a lot of design and development time is wasted because of incorrect user requirements. Getting them right from the start will help alleviate this problem. [Wih06, S. 28]

In der Praxis wäre für die Konzeption bzw. das Systemdesign eines neuen Tools der Besuch der von der Tool Entwicklung betroffenen Teammitglieder am Arbeitsplatz und das beobachten der jeweils verrichteten Aufgaben sehr aufschlussreich. Durch diese Methode könnten Probleme im Arbeitsablauf frühzeitig identifiziert, behoben und besonders wichtige Features rechtzeitig vor Beginn der Implementierung in Erfahrung gebracht und geplant werden.
[Wih06] beschreibt Stakeholder bezüglich des Gebietes Tool Development folgendermaßen:

> "They (Stakeholders, Anmerkung des Autors) are the users who are most affected by the introduction of a tool and they ultimately contribute to the design and goals. Stakeholders are (Anmerkung des Autors) defined as anyone who stands to gain or lose from the succes or failure of an application [...]." [Wih06, S. 4-5]

Diese Definition deckt sich mit der bereits erwähnten allgemeinen Definition der Stakeholder. Sollte für das Softwareprodukt keine Veröffentlichung der Tools zur Generierung von User Generated Content geplant sein, beschränkt sich der Kreis der Stakeholder auf das interne Entwicklerteam. Eventuell sind noch andere Publishereigene Studios und Teams zu berücksichtigen. Dies kann vor allem dann der Fall sein, wenn eine Spiele Engine in mehreren Studios eines Publishers eingesetzt wird. Als Beispiel sei hier die 3D Engine Frostbite Engine [14] von Digital Illusions CE [15] unter dem Publisher Electronic Arts [16] angeführt. Dieser Publisher setzt für alle aktuellen Projekte[17] seiner Studios auf die gleichen Tools und die gleiche Engine.
Aus diesen Gründen ist eine Stakeholderanalyse (auch bei kleineren Projekten) ein Wichtiger Bestandteil des Entwicklungsprozesses. Eine Bedarfsanalye und eine allgemeine Analyse der Aufgabengebiete der jeweiligen Entwickler sollte in das System Design bzw. das Requirements Engineering ebenfalls mit einfließen. Hierzu könnten Interviews (Gespräche und Befragungen zu den Wünschen der Anwender) mit den Anwendern geführt werden. Dies ist eine schnelle Methode

[14] Digital Illusions CE. *Frostbite Engine*. http://www.frostbite.com/. 2015. (Geprüft am 16. 05. 2015)
[15] http://www.frostbite.com
[16] http://www.ea.com
[17] http://www.frostbite.com/games/future-games/

welche sich gut mit einem agilen Prozess wie Scrum oder iterativen Prozessen vereinbaren lässt.

3.3.1 Prozess der Stakeholderanalyse

Die Stakeholderanalyse wurde in folgenden Schritten durchgeführt und skizziert. Da diese Arbeit sich nicht auf ein reales Team bezieht, wurden statt der jeweiligen Personen oder Organisationen die Berufsbilder der eigentlichen Personen in den Vordergrund gestellt.

- Identifizieren der Stakeholder durch kreative Techniken wie z.B. Brainstorming
- Stakeholder in eine Gittergrafik einteilen um ihre Nähe zum Projekt zu bestimmen
- Interpretation der Stakeholder Absichten während des Projekts
- Entwickeln von Maßnahmen um:
  - Risiken zu identifizieren
  - Beteiligungen im Projekt herauszuarbeiten
  - Eine Umsetzungsstrategie zu entwickeln

Der dritte Punkt die Interpretation der Absichten und Einstellung des jeweiligen Stakeholder zum Projekt entfällt in diesem Fall. Es ist ohne konkrete Personen schlicht nicht möglich eine persönliche fachliche Meinung des Stakeholders zu repräsentieren. Es wird an Stelle dessen hier davon ausgegangen, dass jeder Stakeholder ein Benutzerfreundliches praktisches Tool erhalten möchte welches den definierten Anforderungen dieser Arbeit entspricht.

3.4 Zusammenfassung des Arbeitsprozesses

Zu Beginn des Tool Development Projektes sollte erst einmal die Struktur des Entwickler Teams analysiert werden. Hierbei muss sich für eine der möglichen Methoden der Organisation entschieden werden. Die beiden von Wihlidal [Wih06, S.] beschrieben Organisationsstrukturen "Dedicated Tools Teamünd "Content Team Develops and Supports"wurden als praktisch für das interne zur Produktentwicklung parallel ausgerichtete Tool Development vorgestellt. Sobald sich das Entwickler Team für eine Struktur entschieden hat, kann es sich um die Auswahl eines geeigneten Projektmanagementmodells kümmern. Hierbei wurde aufgrund seiner unkomplizierten Art und einfachen adaption

das Scrum Projektmanagementmodell ausgewählt. Es vereinfacht den Entwicklungsprozess und reduziert den nötigen Verwaltungsaufwand auf ein Minimum. Nach der Planung der Organisationsstrukturen kann das Team soch dem Konzeptionellen Part widmen und diesen mit einer Requirements Analyse beginnen. Diese Phase der Entwicklung ist mit größt möglicher Sorgfalt auszuführen da alle weiteren Schritte, insbesondere der Erfolg der Implementierung hiervon abhängig sind. In dieser Phase war es nötig, die Stakeholder zu identifizieren, UseCases zu verstehen und die benötigten Features der Software herauszuarbeiten. Im Anschluss an die Analyse folgt das Requirements Engineering. Hierbei handelt es sich um einen Arbeitsschritt welcher technische und theoretische Praktiken vereint. Eventuell werden während der Requirements Analyse auch besonders kritische Features durch eine Prototypische Implementierung ergründet. Weiterhin entsteht hier das gesamte Systemdesign. Die Implementierungs und Testphase schließt das Projekt ab. Hierbei wird der Erfolg aller vorherigen Phasen auf die Probe gestellt. Die Implementierung kann von Alpha und Beta Tests begleitet werden. Somit können die Mitglieder des Produktionsteams das Tool bereits kennen lernen und einige Features in Rücksprache mit den Entwicklern noch verändern. Der hier skizzierte Prozess wurde so weit es möglich war auch auf die Konzeption des FuseeAT angewendet.

4 Konzeptentwicklung

4.1 Zeitersparnis durch die Entwicklung eines Tools als Plugin für einen bereits existenten 3D Editor

Im Gegensatz zu großen Monumentalen Editoren wie in der Unreal Engine 4 und der Unity Engine vorhanden, sieht das hier angewendete Konzept des Tool splitting vor, die verschiedenen Entwickler Disziplinen auf bekannte Tools aufzuteilen und diese bereits bekannten Tools mit neuer Funktionalität auszustatten. So minimiert sich die Einarbeitungszeit der Entwickler. Weiterhin sollte die Konzentration der Entwickler auf einem höheren Level erhalten bleiben als es bei Entwicklern der Fall ist, welche für jeden Schritt das Tool wechseln müssten. Natürlich bieten bereits jetzt große Editoren wie Unreal Engine 4 und Unity fast jegliche Funktionaltität in einem großen Tool an, wie aber bereits gesagt ist es nötig dazu die bereits erlernte Software wie z.B. Cinema 4D zu verlassen.
Eine Zeitersparnis auf Seiten der Tool Entwickler kann erreicht werden, da sie sich nur mit der Erweiterung einer bereits fertiggestellten, getesteten Software beschäftigen müssen anstatt ein komplett neues Tool zu entwickeln. Es entfällt hierbei ein großer Teil der Software Architektur. Natürlich gibt es auch hier einschränkungen. Das zu erweiternde Tool muss den Tool Developern bekannt und eine Schnittstelle vorhanden sein. Im Rahmen dieser Arbeit konnten beide Bedingungen ausreichend erfüllt werden. Cinema 4D bietet eine Schnittstelle in Form einer API und das Tool war dem Autor aus früheren Projekten ausreichend bekannt.
Das hier entwickelte Konzept versucht nun die in der Analyse gewonnenen Erkenntnisse über den Entwicklungsprozess umzusetzen und dabei ein Tool zu konzipieren welches den Cinema 4D Model Editor um Fusee Editing Funktionalität erweitert. Hierbei wird die Pluginfunktionalität von Cinema 4D als Schnittstelle zum Benutzer ausgenutzt. Das konzipierte Plugin nutzt im Hintergrund als Businesslogik das FuseeAT und beinhaltet selbst nur Funktionalität welche durch die Cinema 4D API bedingt wird. Begonnen werden aber soll die

Konzeption des Tools mit einer Analyse des Teams und dem Erstellen von Use Cases.

4.2 Integration von Tools in die Production Pipeline - Sechs Fragen nach Chandler

In Abschnitt 2.4.1 wurde auf den Fragenkatalog bezüglich der Integration von Tools in die Production Pipeline von Chandler verwiesen [Cha06, S. 223-224]. Hier sollen diesen Fragen nun im Bezug auf das FuseeAT beantwortet werden bevor eine Requirements Analyse und ein System Design erstellt werden. Hiermit soll die Production Pipeline für Fusee untersucht werden. Es muss geklärt sein, ob weitere Tools involviert sind und wenn ob sich diese eventuell mit der geplanten Funktionalität von FuseeAT überschneiden. Somit könnte in einem solchen Fall möglicherweise Entwicklungszeit eingespart werden.

Welche Werkzeuge und Software wird benötigt?

Verwendet wird für ein Projekt mit der Fusee Engine ein Binary Build der Engine selbst. Weiterhin werden verschiedene Code Editoren wie Visual Studio oder XCode für den Mac eingesetzt. Dies hängt jedoch von den Entwicklern selbst ab. Modelle werden für Fusee in einer Modeling Applikation erstellt. Cinema 4D eignet sich hier aktuell am besten. Das Programm unterstützt durch ein Plugin den direkten Export in das Fusee eigene Format .fus. Texturen und 2D Grafiken können mit handelsüblichen Bildbearbeitungsprogrammen erstellt werden.

Ist es möglich Assets in das Engine Format und wieder zurück in das Ausgangsformat zu konvertieren? (Zwei Wege Funktionalität)

Es ist aktuell nicht möglich .fus Dateien wieder nach Cinema 4D zu exportieren. Es müssen also im Fall von FuseeAT jegliche Rohdaten vorgehalten werden. FuseeAT sollte also alle nativen Cinema 4D Dateien für ein Fusee Projekt zusätzlich speichern oder diese zumindest in einem verwalteten Projektordner ablegen.

Gibt es irgendwelche Engpässe? Kritische Konzepte im Game Design?

Da aktuell kein Spiele Konzept existiert können dort keine Engpässe existieren. Jedoch ist im Bereich des tool Developments das konvertieren der C++Funktionalität der Cinema 4D API nach C# eine Problematische Stelle und könnte im Verlauf des Projektes eine Herausforderung darstellen.

Wann muss das Tool umgesetzt sein?

Es existiert keine Deadline für die fertige Umsetzung des Tools da sich diese Arbeit hauptsächlich mit der Konzeptionierung beschäftigt. Das Tool an sich wird voraussichtlich auch nach dieser Arbeit im Rahmen des Fusee Projektes weiterentwickelt werden.

Wie werden Assets im Tool verwaltet? (Entscheidung für eine "Version Control Software")

Assets werden im Tool nicht direkt verwaltet. Die Verwaltung der Assets erfolgt durch das vom Team eingesetzte Version Control System. Assets sollen von FuseeAT lediglich im nativen Format vorgehalten werden während sie aber gleichzeitig im Engine Format .fus im Projekt eingesetzt werden. Das native Format von Cinema 4D wäre .c4d. Es wäre aber zu empfehlen die Modeling Dateien während der Arbeit im fbx oder Wavefront Object Format .obj abzuspeichern. Für 2D Grafikdateien wie Texturen gilt die gleiche Vorgehensweise. Sie sollten im gewünschten Format gesichert werden und zur Verwendeung in Fusee als verlustfreies PNG Bild exportiert werden.

Welche Teile eines Prozesses, einer Software können automatisiert werden?

Es könnten verschiedene Prozesse des Tools automatisiert werden. Diese können dann im Hintergrund des Editors ablaufen. Hierzu zählt das Builden eines Projekts, das Speichern eines Projekt mit Hilfe einer Automatischen Speicherfunktion im Datenverlust vorzubeugen und das bereinigendes Projektverzeichnisses von nicht mehr benötigten Referenzen auf Assets und weiteres.

4.3 Anforderungsentwicklung und Spezifikation

Der hier folgende Abschnitt 4.3 beschreibt das Requirements Engineering welches die Funktionalität des Authoring Tools beschreibt. Die Gliederung des Requirement Dokumentes orientiert sich an den Vorgaben des IEEE Standards 830-1984 definiert in »IEEE Guide for Software Requirements Specifications«. In: *IEEE Std 830-1984* (Feb. 1984), S. 1–26[1]. Der Standard "Guide for Software Requirements Specifications" beschreibt die Inhalte einer Requirement Specification und das Vorgehen während der Erstellung eines solchen Dokuments.

[1]Erreichbar unter folgender Adresse http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=278253. Zuletzt geprüft am 18.05.2015

4.3.1 Geplanter Einsatzzweck und Anwendungsbereich der Software

Das Fusee Authoring Tools Projekt ist eine Bibliothek die es ermöglicht Plugins (und Ähnliches) für 3D Modelling Software zu schreiben welche diese Software um Authoring Tool Funktionen erweitert. Dieses Dokument beschäftigt sich mit den Anforderungen der Software Bibliothek selbst und der Anbindung an Cinema 4D.

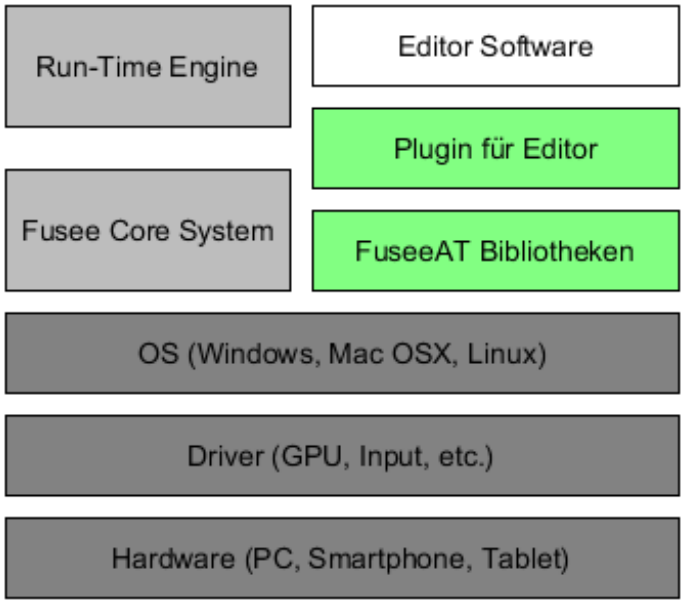


Abbildung 4.1: Stand alone Tool Architektur. Fusee AT Verortung mit Bezug auf die Fusee Engine.

Das Architekturdiagramm 4.1 zeigt die verortung des Authoring Tool Frameworks neben der Fusee Engine und unter dem Aufgesetzen proprietären Cinema 4D Editor. Wie bereits erwähnt könnte in späteren Erweiterungen jeder Editor mit Pluginfunktionalität verwendet werden. Es müsste dann lediglich ein neues Plugin entwickelt werden, dass die Funktionalität des FuseeAT implementiert. Die hier grün eingefärbten Module wurden während dieser Arbeit Konzipiert und eine Umsetzung ist geplant. Die Software soll es ermöglichen, dass Artists, Designer und Engineers an ein und dem selben Projekt arbeiten können ohne die gewohnte eigene Arbeitsumgebung (3D-Modellierungssoftware, Developer IDEs etc.) zu verlassen und etwas komplett neues zu erlernen. Das Projekt wird als Teil des Fusee Engine Projekts entwickelt. Das Projekt ist ein Teil des Fusee Engine Uniplug Projektes. Jeglicher Code ist als Open Source Code zu veröffentlichen bzw. der Zugriff auf den Code ist unter freier akademischer Lizenz sicher zu stellen.

4.3.2 Definitionen und Abkürzungen

FuseeAT - Hierbei handelt es sich um das Fusee Authoring Tool. Die Abkürzung bezeichnet die Authoring Tool Software Bibliothek bzw. das Framework.
Fusee - Die Furtwangen University Simulation and Entertainment Engine. Eine 3D Echtzeit Engine zur Entwicklung interaktiver Applikationen.
Editor - In diesem Fall ein 3D Modeling Editor. Speziell für dieses Projekt der Modeling Editor Cinema 4D von Maxon.
Plugin - Eine Software die eine bereits bestehende Software um selbst geschriebene Funktionalität erweitert.

4.3.3 Unterstütze Funktionalität des FuseeAT

Die hier aufgeführten Funktionen sind Kernaspekte der Software und wären in einer realen Produktion zu erfüllende rechtskräftige Ansprüche. Alle hier genannten Anforderungen gelten für das FuseeAT und die Implementierung des Cinema 4D Plugins.

- Ein Fusee Projekt muss anlegt werden können.
- Ein Fusee Projekt muss bearbeitet werden können.
- Ein Fusee Projekt muss geöffnet werden können
- Ein Fusee Projekt muss gespeichert werden können.
- Das Fusee Engine Projekt soll gebaut werden können.

Identifizieren der Stakeholder

Der Prozess der Stakeholder Analyse wurde bereits beschrieben. An dieser Stelle soll die Analyse einmal Beispielhaft für ein Entwicklerteam durchgeführt werden. Die Stakeholder wurden durch die kreative Methode eines Brainstormings als folgende Berufsgruppen innerhalb des Entwicklerteams des Unternehmens identifiziert. Auf Personengruppen außerhalb des Entwicklerteams wie z.B. Finanzabteilungen etc. soll nicht eingegangen werden, da sich diese Arbeit mit dem Prozess zur Konzeption eines Tools und der eigentlichen Konzeption zu diesem beschäftigt.

- Engineer
  - Tool Eng.
  - Graphic Eng.
  - Network Eng.

  - Articicial Intelligence Eng.

- Artist
  - Animator
  - World Builder / Level Designer

- Game-Designer
  - Level Designer
  - Game Logic Scripter
  - User Interface Designer

- Producer
  - Development Producer
  - Publisher Producer

Die Producer werden innerhalb der Konzeption dieser Arbeit als Designer betrachtet und unter dieser Gruppe geführt. Sollten die Producer denn Aufgaben an der Entwicklung übernehmen sind diese meist deckungsgleich mit den Aufgabenstellungen der Designer. Trotz allem sind die Producer hier aufgrund ihrer tragenden Rolle im Projekt als Stakeholder identifiziert worden. In vielen Fällen haben sie auch massive Entscheidungsgewalt über die verwendeten Techniken und Ressourcen während des Projektverlaufs. Um zu erkennen in welchem Bezug ein Stakeholder zu einem Projekt steht, wurde ein Gittermodell 4.2 der Entwickler erstellt. Hier finden sich Beispielhaft weitere Stakeholder zur Verdeutlichung der Grafik. Hierzu zählen der Publisher, welcher als Organisation auftritt und weitere Entwicklerteams des gleichen Publishers die möglicherweise mit der gleichen Engine und evtl. den gleichen Werkzeugen arbeiten.

4.3.4 Externe Schnittstellen

Das FuseeAT wird als eigenständige Softwarebibliothek entwickelt. Es bestehen keine engen Kopplungen zwischen dem FuseeAT und einem speziellen Editor. FuseeAT kann durch das Implementieren eines Interfaces für jeden gewünschten Editor angepasst werden. Die Software ist offen gestaltet und verwendet ein einfaches System welches Nutzern der Bibliothek das implementieren von Error handling Operationen unterstützt. Das FuseeAT ist Open Source und kann jederzeit eingesehen werden.

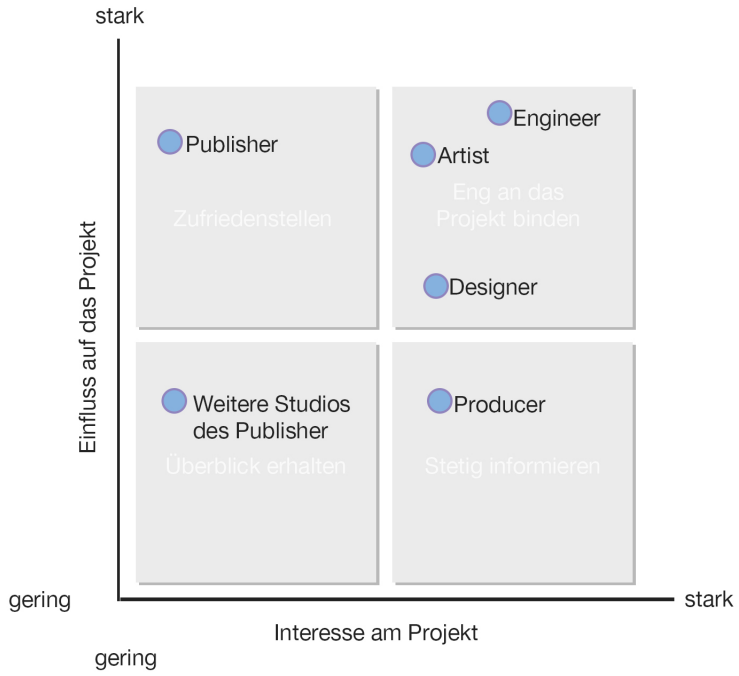


Abbildung 4.2: Überblick über die Identifizierten Stakeholder und deren Inovelement in die Planung und Umsetzung des Tool Projektes.

4.3.5 Restriktionen

- Das gesamte Projekt nutzt die Basis des Uniplug Projektes.
- Das Projekt wird fast komplett in C# und maximal geringe Teile in C++entwickelt.
- Die Entwicklungsdauer ist begrenzt auf Teile des Wintersemesters 14/15.
- Das Projekt muss auf Windows 8 lauffähig sein.
- Der Code muss auf GitHub zur Verfügung gestellt werden.

4.3.6 Projektabhängigkeiten

Das Projekt ist auf Grund verschiedener Abhängigkeiten im Fusee Uniplug Projekt aktuell auf Windows beschränkt. Das Projekt ist auf C# beschränkt. Das Projekt nutzt eine gewrappter Version des Maxon C++SDK.

4.3.7 Funktionale Anforderungen

Hierbei handelt es sich um spezifische Funktionalität welche die Fusee Authoring Tools Bibliothek anbieten soll:

Projekt anlegen

Ein Visual Studio C# Projekt muss vom Editor aus angelegt werden können. Hierzu bedarf es folgender in Schaubild 4.3 Operationen.

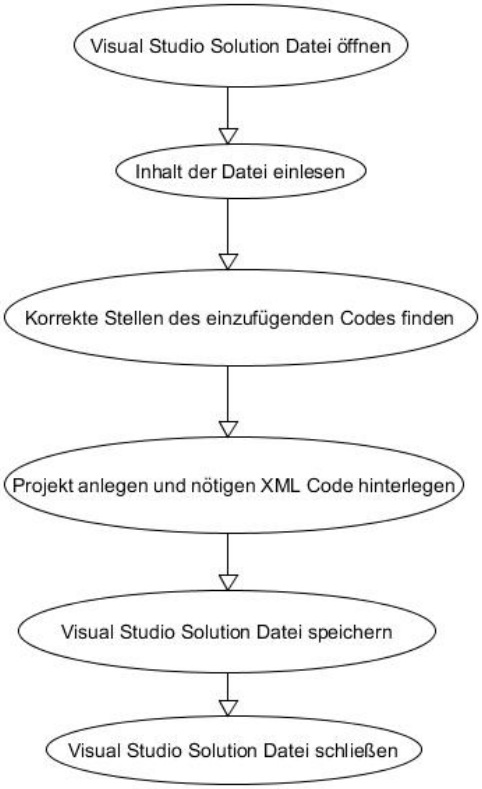


Abbildung 4.3: Ablauf der beschriebenen Funktionalität: Projekt anlegen

Projekt bearbeiten (Durch das Hinzufügen von Assets oder Funktionalität)

Die Grafik **??** beschreibt die Bearbeitung eines Engine Projekts. Hierbei kann der Abschnitt Änderungen vornehmen wiederum andere Tasks implizieren und ist aus diesem Grund an dieser Stelle allgemein gehalten.

Projekt öffnen - Ein bestehendes Projekt im Editor öffnen

Das Projekt muss im Editor geöffnet werden können. Hierzu sind wenige Schritte wie in Abbildung 4.5 beschrieben nötig.

Projekt bauen

Das Projekt soll von einem Entwickler während der Arbeit gebaut werden können. Somit kann der Entwickler seine durchgeführten Änderungen jederzeit testen. Die Abbildung 4.6 zeigt den Ablauf dieser Funktionalität.

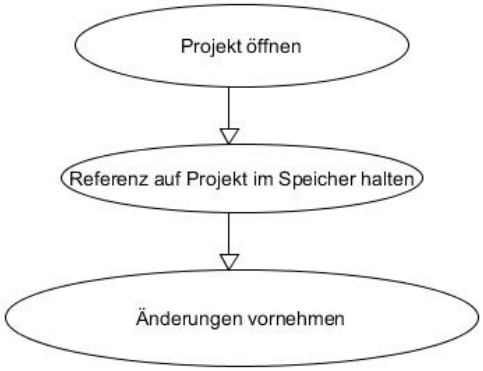


Abbildung 4.4: Ablauf der beschriebenen Funktionalität: Projekt bearbeiten

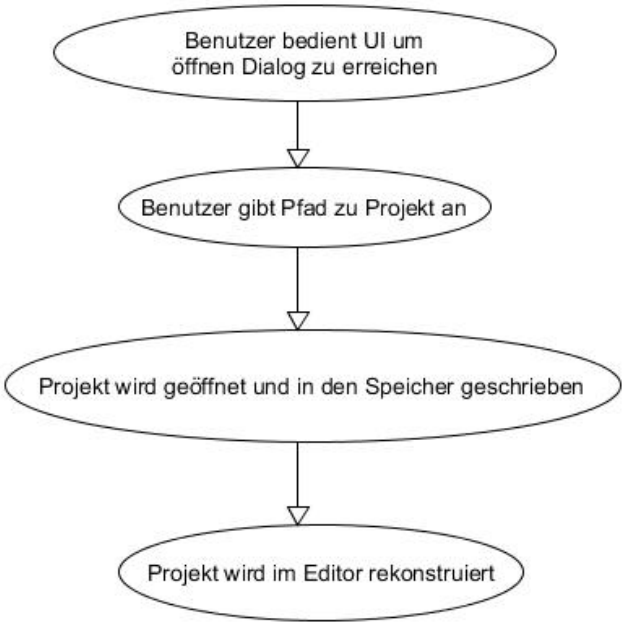


Abbildung 4.5: Ablauf der beschriebenen Funktionalität: Projekt öffnen

Nicht Funktionale Anforderungen

Das Projekt muss während des Projektzeitraums des Wintersemesters 2014/2015 mit einer ausreichenden Basisfunktionalität erstellt werden. Das Projekt muss mit Sandcastle oder ähnlichem Dokumentiert werden und die Dokumentation muss dem Projekt beigelegt werden.

Schnittstellen der Bibliothek

Die Fusee Game Authoring Tool Bibliothek bietet eine Schnittstelle um Plugins für unterschiedliche Modellingsoftware zu erstellen. Die Bibliothek ist nicht von Cinema 4D Funktionen abhängig sondern unterstützt Windows spezifische C# Funktionalität.

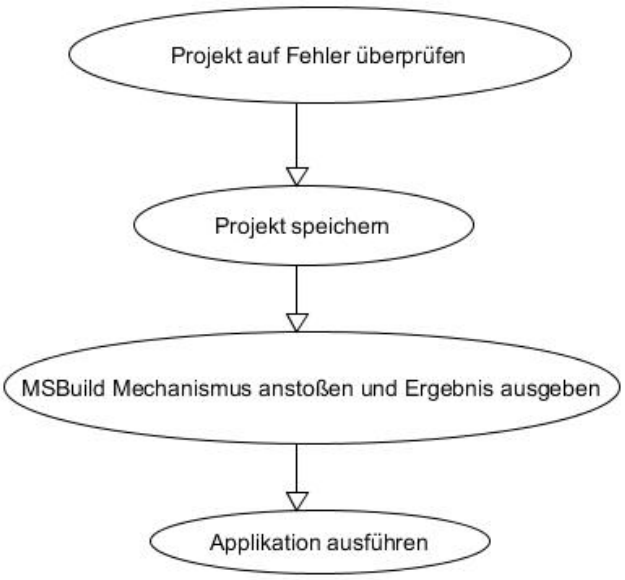


Abbildung 4.6: Ablauf der beschriebenen Funktionalität: Projekt bauen

Design Beschränkungen

Das Interface wird nicht großartig frei gestaltet. Es orientiert sich an den gegebenen Cinema 4D Interface Optionen und nutzt die Cinema 4D SDK API um die Interface Elemente darzustellen.

Qualitätsanforderungen

Das Projekt wird im Alpha Stadium beendet und stellt daher nur einen gewissen Pool an Basisfunktionalitäten zur Verfügung. Das Projekt kann Problemlos um weitere Funktionalität erweitert werden und es wird das Fusee und das Uniplug Projekt weiterhin begleiten.

4.4 Use Cases der verschiedenen Entwickler

Diese Abschnitt behandelt eine Auswahl alltäglicher Tasks (z. Dt. Aufgaben) der Mitarbeiter des Produktionsteams einer Fusee Applikation. Aus den aufgelisteten Tasks wurden dann die Anforderungen für die Requirementsanalyse herausgearbeitet. Um an diese Auflistung heranzukommen, empfiehlt sich in der Praxis das Interviewen und das Beobachten von Teammitgliedern um den Arbeitsalltag und die Probleme die das Tool lösen soll besser zu verstehen. Weiterhin können Ziele des Projektes und spezielle kritische Marken im Gamedesign Aufschluss über benötigte Features liefern. Die hier aufgeführten Tasks beschreiben Arbeitsschritte welche das Fusee Authoring Tool unterstützen sollte. Die Kommunikation der Benutzer zum Fusee Authoring Tool unterscheidet

sich durch den Prozess des Tool splitting je nach Entwickler. Während Artists und Designer über eine Grafische Schnittstelle auf das Fusee Authoring Tool zugreifen, arbeiten Ingenieure mit Visual Studio direkt auf dem Projekt der Engine. Hier ist keine weitere Schnittstelle zum Fusee Authoring Tool nötig.

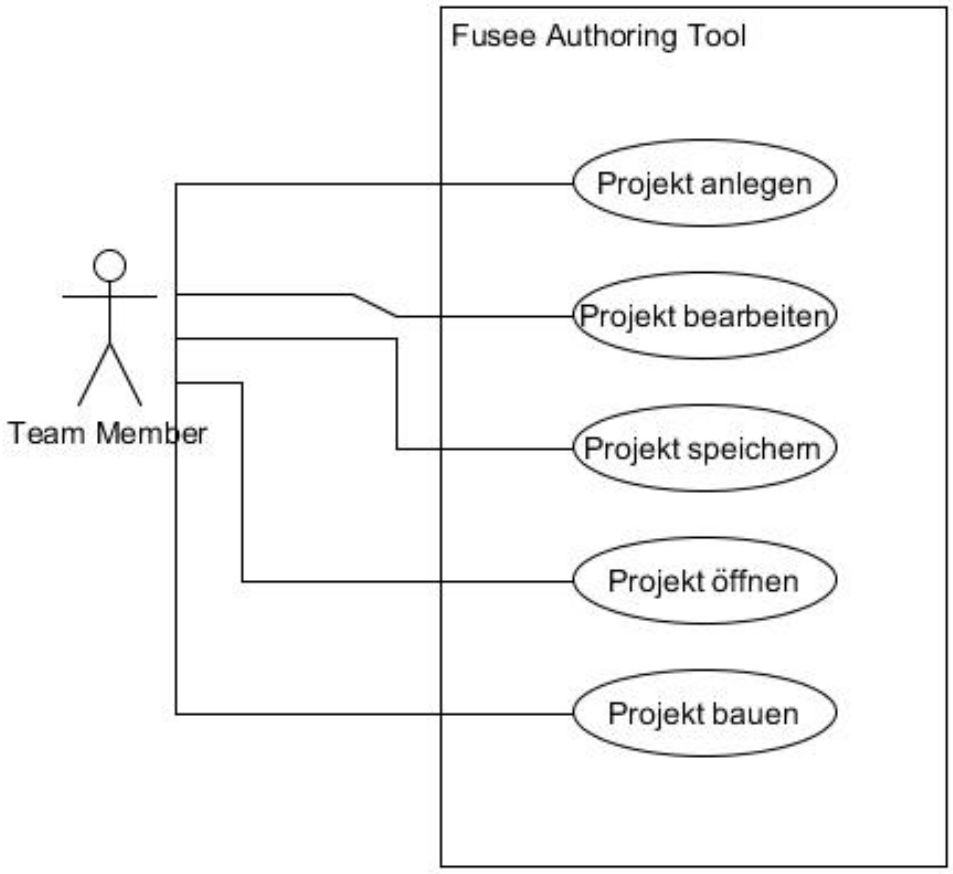


Abbildung 4.7: Überblick über die Use Cases der Entwickler in einem Fusee Authoring Tool.

Die hier dargestellten Aufgaben umschreiben die Basisfunktionalität eines Authoring Tools für die Fusee Engine. Solche Funktionen können in der meisten Software sowohl per Code aufgerufen, als auch ber GUI Funktionalität erreicht werden. In den folgenden Diagrammen wird auf eine erneute Auflistung der Tasks verzichtet und sie werden nur noch exemplarisch dargestellt.

4.4.1 Use Cases für Artists

Es folgt eine Auflistung der Tasks die den normalen Arbeitsablauf eines Artists in der der Entwicklung mit der Fusee Engine bezeichnen. Um die Tasks zu ermitteln wurde sich verschiedener kreativer Hilfsmittel wie z.B. Mindmaps und Brainstorming bedient. Die folgenden Use Cases beschreiben die Tätigkeiten

von Artists im Fusee Authoring Tool.

- Asset erstellen
- Asset exportieren
- Szene erstellen
- Szene bearbeiten
- Szene speichern

Die Abbildung 4.8 beschreibt die Einbindung des Artist in das System und skizziert seine Use Cases in Verbindung mit den Modulen des gesamten Systems. Das Modul “Fusee Authoring Tool” beschreibt hier eine Software Bibliothek und Editoren Schnittstelle.

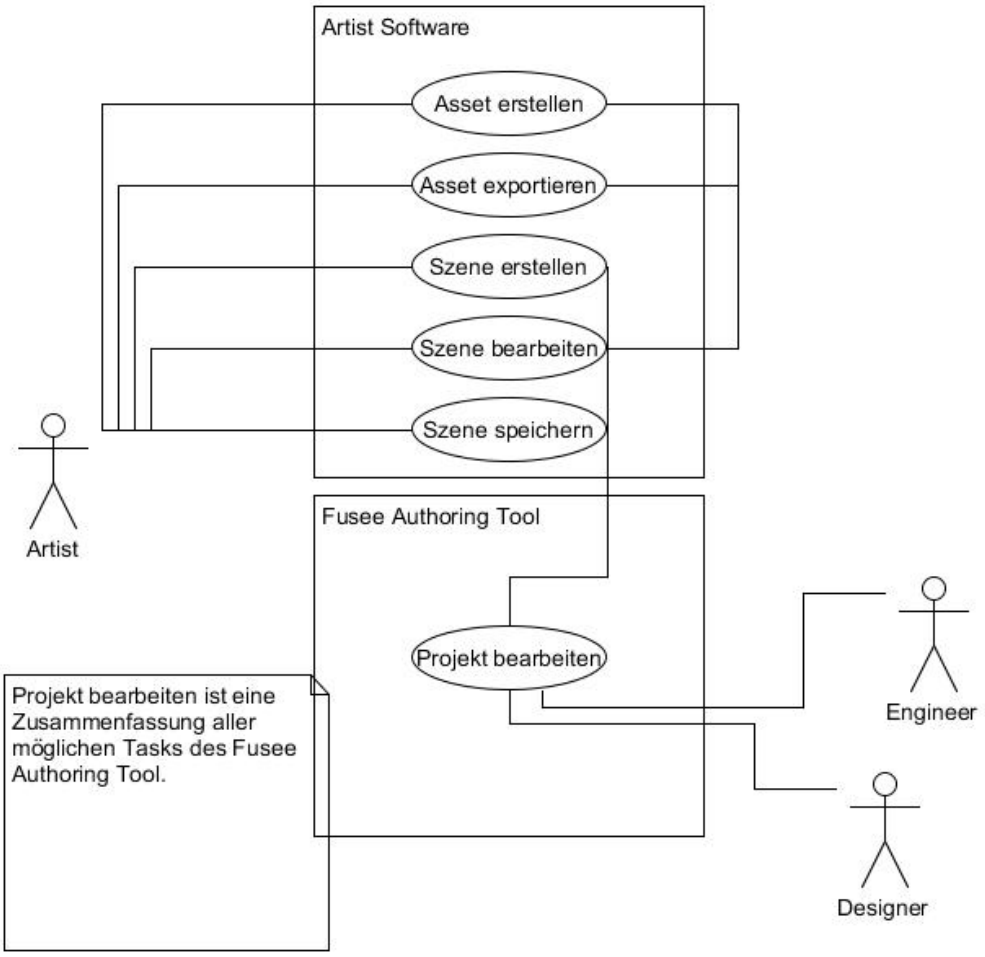


Abbildung 4.8: Überblick über die Use Cases eines Artist.

Es folgt eine Betrachtung der einzelnen Tasks vom Standpunkt des Requirements Engineering aus. Anschliessend wurden die wichtigsten Aktivitäten eines Tasks herausgefiltet und hierraus ein System entwickelt.

Asset erstellen

Der Artist erstellt ein Asset in seiner gewohnten Software. Normalerweise handelt es sich hier um 3D Modelle oder 2D Texturen. Texturen werden in einem 2D Grafikprogramm erstellt und in das 3D Programm importiert. Es folgt das mapping auf die Modelle.

Asset exportieren

Ein Modell wird vom Artist exportiert und in der Projekstruktur gespeichert. Dieses verorten in der Projektstrutkur übernimmt das Plugin. Das Asset wird in die Solution Dateien integriert.

Szene erstellen

Eine Szene wird über das Interface der 3D Applikation erstellt und in der Projektstruktur über das Plugin verortet.

Szene bearbeiten

Eine bereits bestehende Game Szene kann vom Artist bearbeitet werden.

Szene speichern

Eine Szene wird vom Artist gespeichert und steht danach anderen zur Bearbeitung zur Verfügung.

4.4.2 Use Cases für Designer

Die Use Cases von Designern bilden sich häufig aus einer Schnittmenge der Use Cases von Artists und Engineers. Was Designer einzigartig macht ist meist ihr Überblick über die gesamte Entwicklung. Folgende Usse Cases beschreiben die Tasks von Designern im geplanten Fusee Authoring Tool. Das Use Case Diagramm 4.9 visualisiert die aufgeführten Use Cases.

- Skript-Datei hinzufügen
- Asset importieren
- Projekt bearbeiten
- Builds anfertigen
- Szene erstellen
- Szene bearbeiten

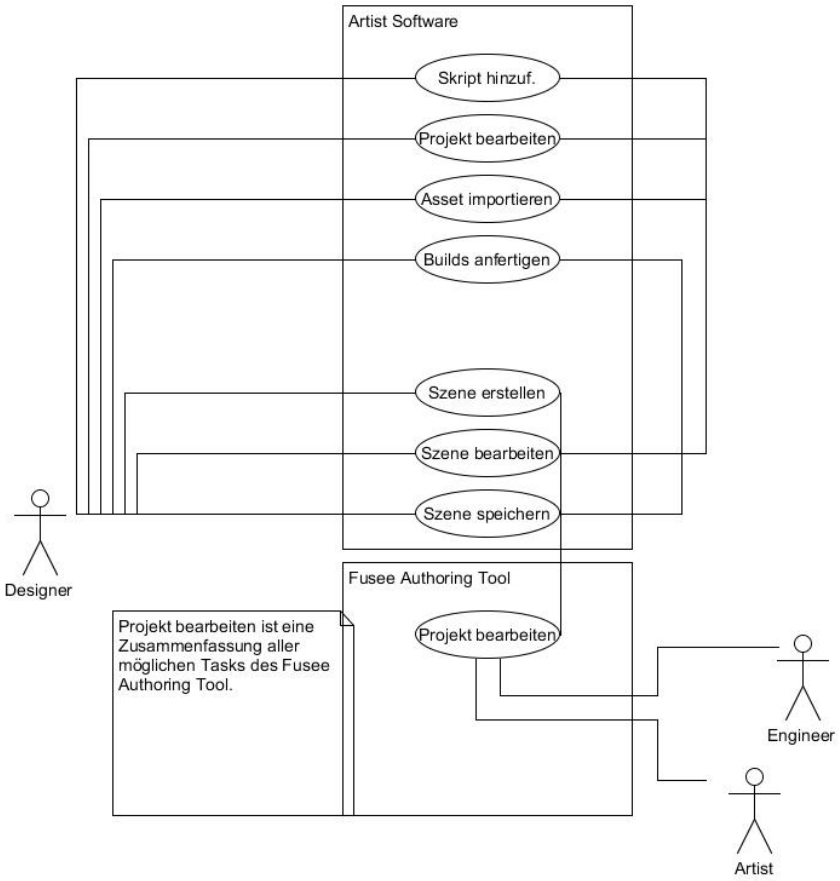


Abbildung 4.9: Überblick über die Use Cases eines Designers.

- Szene speichern

Skript-Datei hinzufügen

Eine Skript Datei wird an ein Objekgt angefügt. Im Falle von Fusee handelt es sich hier entweder um Javascript Code für Web Builds oder um eine C# Code Datei.

Projekt bearbeiten

Der Designer bearbeitet das Projekt. Dies bedeutet er muss das Projekt öffnen und betrachten können.

Asset importieren

Der Designer kann vom Artist erstellte Assets in eine Szene des Editors importieren.

Assets platzieren, skalieren und transformieren

Der Designer platziert verschiedenste Assets in die erstellten Welten mit Hilfe des Editors. Er skaliert diese Assets auf die korrekte Größe und transformiert ihre Position im Raum. Somit passt er die vom Artists erhaltenen Assets an seine eigenen Bedürfnisse an.

Builds anfertigen

Das erstelle Spiel wird durch einen Build getestet. Sollte ein Build erfolgreich sein, kann der Designer seine erstellen Welten und Level in der Fusee Applikation ausprobieren. Er kann die Wirkung der Welt beurteilen und möglicherweise Veränderungen am Design und Layout vornehmen.

Szene erstellen, Szene bearbeiten, Szene speichern

Diese drei UseCases sind deckungsgleich mit denen des Artist und werden aus diesem Grund hier nicht mehr gesondert beschrieben.

4.4.3 Use-Cases eines Engineer

Engineers bewegen sich eher auf der technischen Seite der Entwicklung und beschäftigen sich demnach im Fall von Fusee hauütsächlich mit einer IDE wie Visual Studio. Die hier aufgeführten Tasks beschreiben die Use Cases eines Engineers im Bereich des Fusee Developments. Hier sei noch einmal erwähnt, dass der Engineer nicht auf das Fusee Authoring Tool zugreift. Seine Tasks beziehen sich nur auf die Arbeit in der IDE.

- Code-Datei hinzufügen
- Asset verwenden
- Projekt bearbeiten
- Projekt builden
- Szene erstellen
- Szene bearbeiten
- Szene speichern

Die Übersicht 4.10 skizziert die Verwebung des Engineer mit seinen Use Cases und dem System.

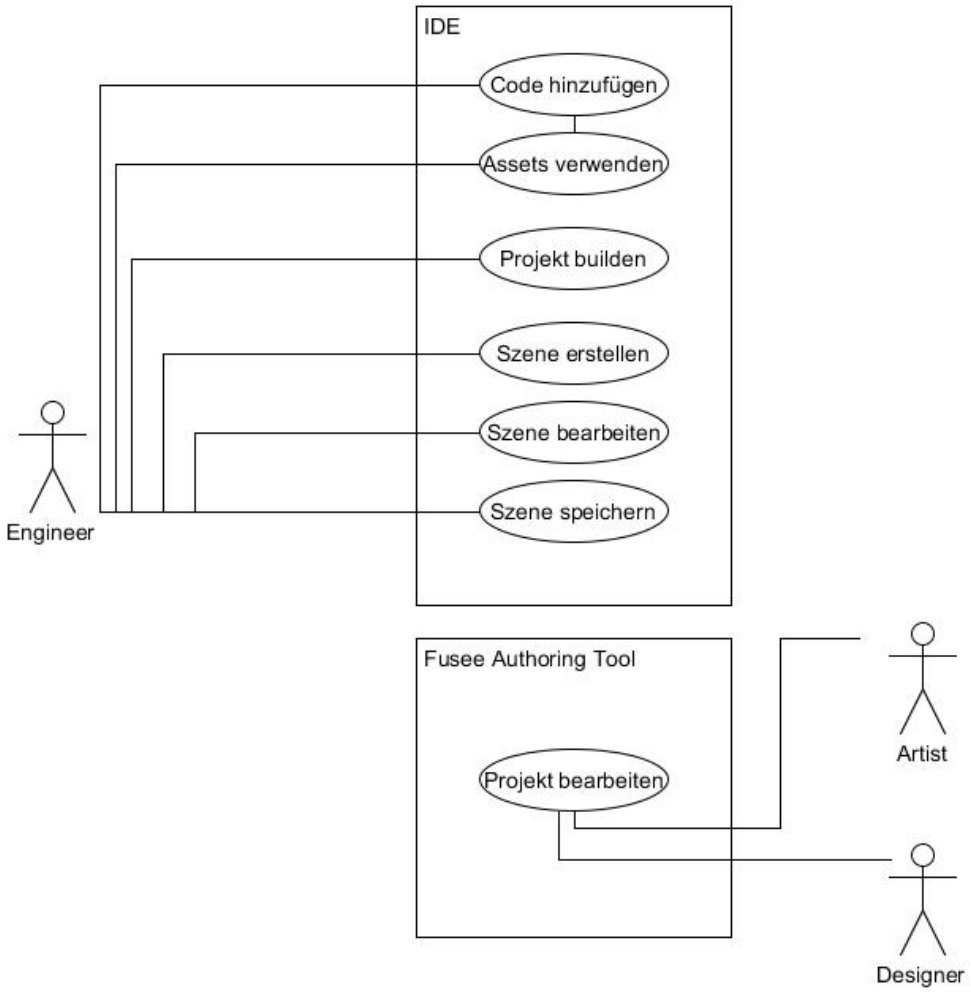


Abbildung 4.10: Überblick über die Use Cases eines Engineer.

Der Engineer beschäftigt sich mit einem programmierlastigeren Aspekt des Projektes. Zu seinen Aufgaben gehört die Verwaltung des Projekts und das Verwenden der vom Artist erstellten Assets.

Code-Datei hinzufügen

Der Engineer möchte neu geschriebenen Code in das Projekt einfügen. Ersucht sich somit die passende Stelle und kann den Code in der Projektstruktur erzeugen / einfügen. Der Code wird durch die IDE in der Projektstruktur untergebracht.

Asset verwenden

Der Engineer verwendet Assets (Models, Texturen, etc.) um diese in einer Szene anzuprogrammieren. Hierzu wird das Asset durch Code in die Engine geladen, texturiert und mit Shadern versehn etc. Bei einem Build wird das

Model korrekt kopiert bzw. in das Build Format überführt.

Projekt builden

Der Engineer baut das Projekt für die gewünschte Distributions oder Testplattform und erhält ein lauffähiges Binary Programm. Hierzu wird meist eine einfache Menükonstruktion wie Schaltflächen oder Dropdown Menüs genutzt. In Visual Studio entspricht dies den Build Targets.

Projekt bearbeiten

Der Engineer verändert etwas an der Projektstruktur oder der Codebase des Projektes. Die Änderungen werden dauerhaft gespeichert und in der Projektstruktur verortet.

Szene erstellen

Der Engineer erstellt während des hinzufügens von Code eine neue Szene in der GameEngine. Diese neue Szene hat verschiedene im Code übergebene Eigenschaften. Der ProjectHandler muss diese Szene möglicherweise registrieren und so für eventuelle Artists in der jeweiligen Software zugänglich machen.

Szene bearbeiten

Der Engineer öffnet eine Szene und bearbeitet diese in seiner IDE. Hierzu muss der ProjectHandler die Szene bereitstellen und mögliche Änderungen verwalten und an Artists weitergeben.

Szene speichern

Eine Szene wird gespeichert und vom ProjectHandler in ein Format überführt welches ein Artist in seiner gewohnten 3D Umgebung ebenfalls nutzen kann.

4.5 Analyse von beliebten Editoren und derer Module im Bezug auf die Konzeption des FuseeAT Plugins

Aufgrund der Thematik mit dem Bezug zur 3D Engine Fusee beschränkt sich die Auswahl der betrachteten Editoren auf die an der Hochschule Furtwangen populärsten Game Engines und Editoren, analysiert deren Editoren Features und stellt die Arbeitsabläufe in diesen Editoren und Tools dar. Es folgt eine Wertung und ein Vergleich mit den für das Fusee Authoring Toolkit geplanten Features. Bei den hier gewählten Editoren handelt es sich um stand alone

Applikationen welche das Gegenteil des hier entworfenen Konzeptes eines in das Content Creation Tool integrierten Editors darstellen. Allerdings gleichen sich die Authoring Tool Ansätze maßgeblich. Funktionen, Module und Konzepte sind größtenteils deckungsgleich mit denen des Cinema 4D Editors. Aus diesem Grund lassen sich einige der Module für FuseeAT und vor allem das Cinema 4D Plugin adaptieren.

4.5.1 Unreal Engine 4

Die UnrealEngine4[2] (kurz UE4) wird von EPIC Games Inc. entwickelt und stellt eine im grafischen Sektor im High End Bereich angesiedelte Game Engine dar. Bei ihr handelt es sich um eine an einen Authoringeditor (Welt Editor / Level Editor) gebundene Game Engine.Alle Teile des Projekts kommen in diesem Editor zusammen und Projekte müssen dort bearbeitet werden. Die Engine sieht die Verwendung von Visual Studio 2013 auf Windows Rechnern und der IDE XCode auf Mac Rechnern zum schreiben von Code und dem kompilieren von Projekten vor. Seit kurzem ist die Unreal Engine kostenfrei unter http://www.unrealengine.com zu beziehen. Sollte ein mit der UE4 entwickeltes Produkt kommerziell vertrieben werden, so sind 5% des Umsatzes an Lizenzgebühren an EPIC zu bezahlen.
Sourcecode wird in der Unreal Engine in C++ und Blueprint geschrieben. Bei Blueprint handelt es sich um eine von EPIC entwickelte Visual Scripting Language (ein Node basiertes System) die sich verstärkt an Designer und Artists richtet.
Die UE4 kann Builds für folgende Plattformen erzeugen:

- Windows PC
- Linux
- Mac
- iOS
- Android
- XBox One (In Verbindung mit einer DevKit Lizenz zu erwerben bei Microsoft.)
- Playstation 4 (inkl. dem noch nicht veröffentlichten VR Projekt Morpheus) (In Verbindung mit einer DevKit Lizenz zu erwerben bei Sony.)

[2]EPIC GAMES, INC. *Unreal Engine 4*. http://www.unrealengine.com. 2004-2015. (Geprüft am 18. 05. 2015)

Plattformen auf welchen mit der Unreal Engine entwickelt werden kann sind:

- Windows PC
- Mac OSX
- Linux (in Entwicklung, unstable)

Der Sourcecode der Unreal Engine steht auf GitHub[3] zur freien Verfügung. Aus diesen Gründen wird die aktuellste Version der Unreal Engine, Version 4, für die Analyse dieser Arbeit zugrunde gelegt.
Der Unreal Engine 4 Editor ist sehr flexibel an die eigenen Bedürfnisse anpassbar. Abhängig von den Wünschen der Entwickler kann die Oberfläche mehr auf das Entwickeln oder Designen ausgerichtet werden und bietet ähnliche Funktionen wie z.B. das Modeling Programm Cinema 4D. Der Editor bietet zu fast jeder Aktion eine Grafische Oberläche an. So genannte Projekt Wizards, führen den Benutzer bei komplexeren Prozessen an der Hand und erleichtern ihm so die Konfiguration von Assets im Editor. Sie unterstützen z.B. beim erstellen einer neuen Klasse oder dem Anlegen eines neuen Materials. Diese Wizards sind besonders für Neueinsteiger nützlich. Sie verdeutlichen den Ablauf eines Prozesses und unterstützen ein Fehlerfreies Abarbeiten der unterstützen Aktion.

Coding in der Unreal Engine 4

Anwendungen in der UE4 können mit Hilfe von verschiedenen Codingmechanismen erstellt werden. Die erste Möglichkeit wäre die dauerhafte Verwendung des Editors und des Konzeptes des Visual Programming mit Hilfe des Unreal eigenen Blueprint Systems. Beim Visual Programming handelt es sich wie die Bezeichnung bereits andeutet um visuelles Programmierung mit Hilfe von Grafischer Darstellung des Codes anstelle der Darstellung als Text. Hierbei werden oft Systeme eingesetzt die auf Nodes basieren. Die Abbildung 4.11 zeigt einen Ausschnitt aus einem Visual Programming Code welcher mit Blueprint erstellt wurde. Als Blueprint wird eine zusammengesetze Konfiguration aus Assets und Node basiertem Code bezeichnet.
Die etablierte Programmiersprache C++ ist zweite Möglichkeit mit der UE4 Code zu schreiben. Allerdings hat EPIC die Unreal Engine Compiler dahingehend erweitert, dass ein spezielles Reflection System für die UE4 spezielle Sprachkonstrukte erkennen und in der Engine verwenden kann. Das Beispiel

[3] https://www.unrealengine.com/ue4-on-github

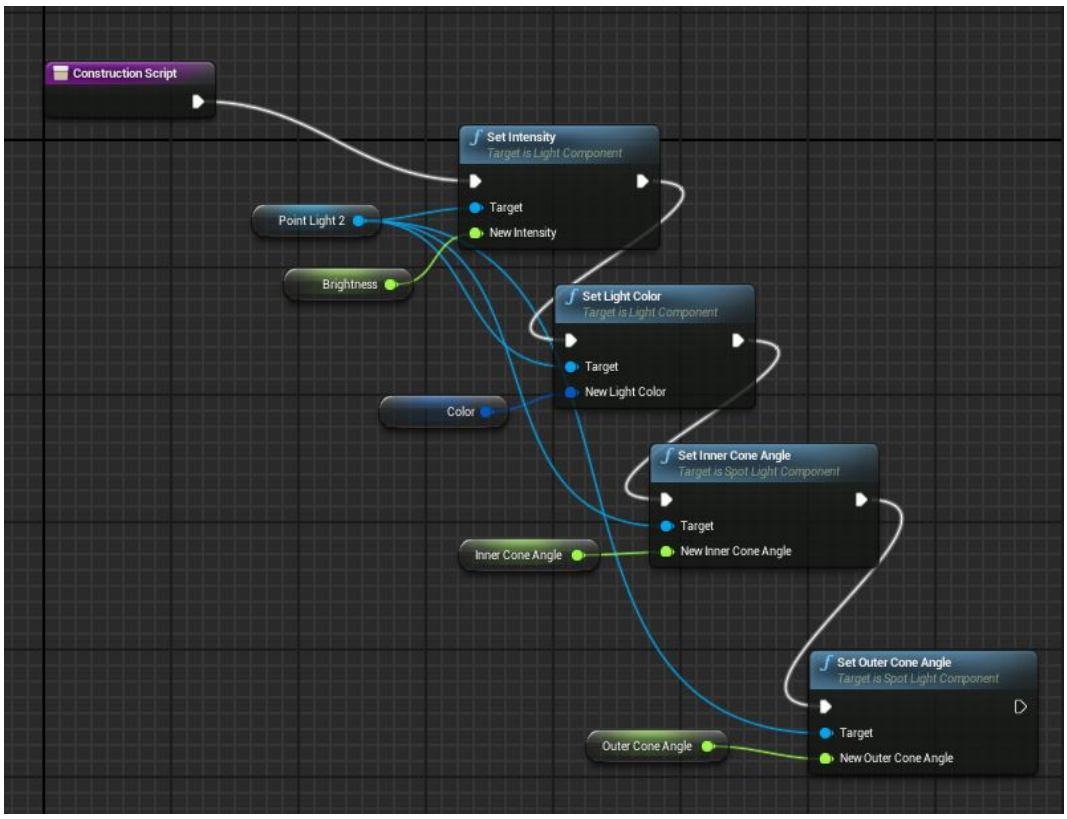


Abbildung 4.11: Programmierung in der Unreal Engine 4 mit Hilfe des Blueprint Visual Programming Systems/Interfaces

4.1[4] zeigt mehrere solcher Markup Befehle vor den jeweiligen Funktionen, Properties der Klasse. Es handelt sich hierbei um Bezeichnungen wie z.B. UENUM(), UCLASS(), USTRUCT(), UFUNCTION(), und UPROPERTY().

Listing 4.1: Unreal Engine C++ Header Datei. Makros als Markup Befehle

```
1  #pragma once
2  #include "GameFramework/Actor.h"
3  #include "UECodeAnalysisProjectile.generated.h"
4
5  UCLASS(config=Game)
6  class AUECodeAnalysisProjectile : public AActor
7  {
8
9    UPROPERTY(VisibleDefaultsOnly, Category=Projectile)
10   class USphereComponent* CollisionComp;
11
12 public:
```

[4]Entnommen und abgeändert/erweitert aus der C++ Datei UECodeAnalysisProjectile.h des Beispiels “First Person Scene” des Unreal Engine 4 Editors.

```
13    UFUNCTION()
14    void OnHit(AActor* OtherActor, UPrimitiveComponent*
        OtherComp, FVector NormalImpulse, const FHitResult& Hit);
15
16    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =
        Gameplay)
17    class USoundBase* FireSound;
18  };
```

Diese Makros und weitere sind alle in der UE4 Dokumentation https://docs.unrealengine.com/latest/INT/ aufgelistet. Allerdings ist diese Art der Deklaration nur für bestimmte Schnittstellen Klassen essentiell. Der meiste selbst geschriebene Code kann ohne diese Makros geschrieben werden. Sie dienen nur der Kommunikation mit dem Editor und Kernsystemen der Engine. Es ist aber zu empfehlen diese Möglichkeit zu nutzen, so kümmert sich Beispielsweise der Garbage Collector der UE4 um Properties und Objekte welche mit den korrekten Makros ausgezeichnet wurden.

> For instance, a variable with a declaration prefaced by a UPROPERTY() macro can be garbage collected by the engine, and can be displayed and edited within Unreal Editor. [EPI15] - Dokumentation Unreal Engine 4 https://docs.unrealengine.com.

Der gesamte C++ Code eines Projektes wird je nach Entwicklungsplattform entweder in Visual Studio (auf Windows) oder in der IDE XCode (auf Mac OSX) geschrieben. In der Realität werden Projekte mit einer Mischung der beiden Möglichkeiten (C++ und Blueprint) umgesetzt. Für das schnelle Prototyping eignet sich aber Blueprint alleine bereits sehr gut.

Asset Verwaltung in der Unreal Engine 4

Assets können in der Unreal Engine 4 z.B. im Dateiformat .fbx (Version 2013) Importiert werden. Die Modelle und Materialien können hierbei z.B. direkt in 3DS Max, Modo oder Cinema 4D erstellt und übernommen werden. Es ist also möglich, die UE4 direkt in die AssetPipeline einzubinden. Durch die offene Verfügbarkeit des Quellcodes können auch eigene Module für die UE4 geschrieben werden, welche die Assetpipleine erweitern. Die Unreal Engine bietet einen Contentbrowser (siehe Abbildung 4.12) an, welcher es ermöglicht Assets zu sortieren und diese zu durchsuchen. Assets können dann per Drag and Drop in die Szene der Engine eingefügt werden. Eine solche Lösung ist optimal um Designer und Artists bei ihrer Arbeit zu unterstützen.

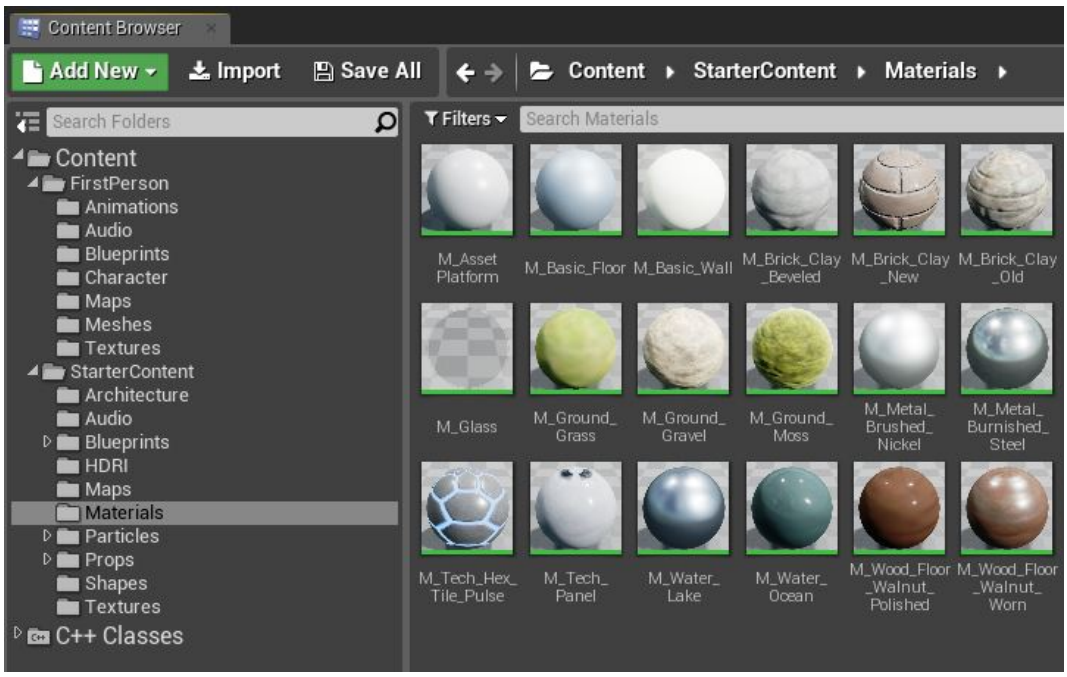


Abbildung 4.12: Ansicht des Contentbrowsers der Unreal Engine 4. Mit Suchfeld und Filterfunktionen, Assetvorschau ausklappbarer Ordnerstruktur.

Zur Versionsverwaltung von Assets ist die Möglichkeit zur Integration eines Versionskontrollsystems gegeben. Eingebunden werden können auf GIT, Subversion und Perforce basierte Systeme.

Szenengraph in der Unreal Engine

Hier gehts um den UE4 Szenengraphen 4.13 und seine Gemeinsamkeiten mit dem Fusee und Cinema 4D Szenengraphen. Der Szenengraph hängt unmittelbar mit der Detailansicht der Elemente zusammen. In der Detailansicht können jegliche “Components” an einem Objekt der Szene betrachtet und verändert werden.

Was kann für das FuseeAT Konzept aus der UE4 übernommen werden?

Um mit der Unreal Engine effektiv zu arbeiten, muss jeder Entwickler das Tool von Grund auf neu erlernen. Genau diese Problematik soll FuseeAT umgehen indem es bereits bekannte Tools, für diese Arbeit Cinema 4D, verwendet um eine von den Mitarbeitern bereits erlernte Oberfläche weiter zu verwenden. Allerdings bietet auch eine so komplexe und tief gängige Engine wie die Unreal Engine 4 Ansätze die für das Fusee AT interessant sind. Das Blueprint System funktioniert wirklich gut und unterstützt Game Designer und Skripter während ihrer Arbeit und bietet ihnen eine gut durchdachte Visual Program-

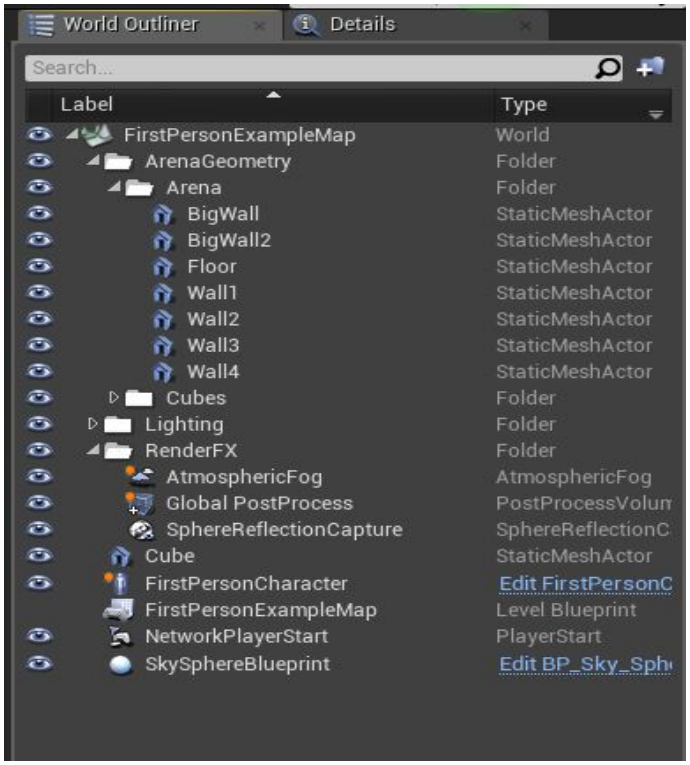


Abbildung 4.13: Beispielhafte Szene im Szenengraph der Unreal Engine 4. Er gleicht dem Fusee und Cinema 4D Szenengraphen stark.

ming Lösung für die täglichen Aufgaben an. Ein solches System könnte sich in FuseeAT in Zukunft ebenfalls umsetzen lassen. Diese Funktionalitäten wurden bereits an der Hochschule Furtwangen in einem Projekt (Circuit 3D 2011 bis 2012) bei Herrn Prof. Müller diskutiert und an implementiert. Ein Teil der Ergebnisse dieses Projektes fand damals den Weg in das Fusee Projekt Uniplug. Weiterhin sind die Makros bzw. das Markup System ein guter Ansatz, Code zu markieren um im FuseeAT darauf einzugehen. C# bietet hier mit seinem Reflection und Attributes System genügend Möglichkeiten diese Funktionalität in FuseeAT zu integrieren. Die Funktionen des Contentbrowsers der UE4 könnten in Cinema 4D über verschiedene API Funktionen nachgebildet werden. Der Szenengraph der UE4 gleicht dem Fusee und Unity 3D Szenengraphen stark. Es handelt sich bei dieser Ausgestaltung um eine Konvention im Sektor der 3D Engines und darum kann das System von Cinema 4D übernommen werden ohne voraussichtlich dem Arbeitsfluss der Designer und Artists zu schaden.

4.5.2 Unity 3D

Die Unity 3D Engine [5] in der Version 5 wird von Unity Technologies entwickelt. Das Lizenzmodell sieht eine kostenfreie Variante mit einigen Einschränkungen

[5] Unity Technologies. *Unity 5*. http://www.unity3d.com. 2015. (Geprüft am 16. 05. 2015)

und eine an Professionelle Nutzer gerichtete kostenpflichtige Version für in etwa 75$ pro Monat vor. Enterprise Versionen, welche auch den Sourcecode der Engine enthalten, können über eine gesonderte Kontaktaufnahme gekauft werden. Die Unity Engine ist an den Unity Editor gebunden. In diesem Editor werden alle Entwickleraufgaben erledigt. Die IDE kann vom Programmierer frei gewählt werden. Es empfiehlt sich aktuell aber auf das von Unity mitgelieferte Mono Develop, Xamarin Studio oder Visual Studio in einer Version ab 2013 zu setzen. Vollen Support bietet aber aktuell nur das mitgelieferte Mono Develop. Unity unterstützt die Programmiersprachen C#, Javascript und Boo. Bei Boo handelt es sich um eine von Unity Technologies selber entwickelte Skriptsprache.
Unity kann Builds für die folgenden Plattformen erzeugen:

- iOS
- Android
- Windows Phone 8
- Black Berry 10
- Windows
- Windows Store
- Mac OSX
- Linux
- Web Player
- Playstation 3 *[6]
- Playstation 4 *
- Playstation Vita *
- Playstation Mobile *
- XBox One **[7]
- XBox 360 **
- Wii U ***[8]

[6]* In Verbindung mit einer DevKit Lizenz zu erwerben bei Sony.
[7]** In Verbindung mit einer DevKit Lizenz zu erwerben bei Microsoft.
[8]*** In Verbindung mit einer DevKit Lizenz zu erwerben bei Nintendo.

Als Entwicklerplattformen unterstützt Unity die Betriebssysteme Windows und Mac OSX. Der Sourcecode der Engine ist nur in einer Enterprise Version erhalten. Zum Erhalt dieser müssen gesonderte Verhandlungen mit Unity Technologies geführt werden. Aus diesen Gründen wird die kostenfreie "free" Version für die Analyse in dieser Arbeit zu Grunde gelegt.

Coding in der Unity Engine 5

In Unity 5 kann wie bereits erwähnt in C#, Javascript und Boo entwickelt werden. Die beiden Skriptsprachen Javascript und Boo wurden für diese Arbeit nicht betrachtet, da sich auch Fusee für die Desktopentwicklung auf C# fokussiert hat. Unity bringt eine Version der Mono Develop IDE mit, welche alle drei Sprachen unterstützt. Unity unterstützt in der Version 5.0 aktuell das .Net Framework 3.5 und damit die C# Version 3.0 und die damit verbundenen Sprachfeatures. Unity setzt stark auf das Konzept des Programmierens mit Components. Die gesamte Engine ist für die Arbeit mit Components ausgelegt. Hierbei wird Funktionalität der Anwedungen so weit heruntergebrochen, bis sie in einzelne Code Dateien ausgelagert werden kann. Aus einem Pool an Component Skripts wird dann das Verhalten eines einzelnen Szenenobjekts aufgebaut. Dieses System ist sehr dynamisch und unterstützt die Entwickler beim experimentieren in der Game Engine. Eine solche Component basierte Lösung ist für das FuseeAT ebenfalls denkbar. Unity bietet genau wie die Unreal Engine einige Standardfunktionen welche im C# Code überschrieben werden sollten um ein Objekt in der Engine zu instanziieren. So wird ein GameObject (Die Bezeichnung eines Laufzeitobjektes in Unity) wie in Codebeispielt 4.2 implementiert. Das Beispiel enthält keinen funktionalen Code sondern stellt lediglich die Struktur dar. Fusee wird keine dieser Strukturen einbinden. Das Arbeiten mit FuseeAT soll für den Programmierer so frei wie möglich gestaltet werden.

Listing 4.2: Unity GameObject Struktur

```
1 using UnityEngine;
2
3 public class CubeScript : MonoBehaviour {
4
5   // Use this for initialization
6   void Start () {
7     // ...
8   }
9
```

```
10    // Update is called once per frame
11    void Update () {
12      // ...
13    }
14  }
```

FuseeAT wird auf eine solche Einschränkung der Programmierung verzichten. Das hat mehrere Gründe. Zum einen soll Fusee als Engine in der Lehre eingesetzt werden und hier besonders das Verständnis für tiefer gehende Prozesse einer Game Engine fördern. Zum anderen sollen jegliche Projekte mit der Fusee Engine durch einen einfachen Build auch im WebBrowser ausgeführt werden. Weiterhin ist die Fusee Engine ein großes Teamprojekt und deren Erweiterung in diesem Maße kann während dieser Arbeit nicht entschieden und/oder umgesetzt werden.

Asset Verwaltung in der Unity Engine 5

Die Verwaltung von Assets in Unity ist ein recht einfacher Prozess. Modelle und Grafiken werden in den jeweiligen Programmen wie z.B. 3DS Max, Cinema 4D, Photoshop oder anderer Grafik Software erstellt und im passenden Format einfach in den Projektordner von Unity kopiert. Das Asset kann dann im Unity Editor selbst noch auf das Projekt abgestimmt werden. Ein Modell kann Materialien erhalten, Texturen können getauscht und Grafiken skaliert und auf Objekte angewendet werden. Aus mehreren Assets und/oder Components können in Unity so genannte Prefabs erstellt werden. Bei diesen handelt es sich um ein Paket aus Assets und Code welches z.B. zur Laufzeit der Engine instanziiert werden kann. Prefabs könnten auch als Abbild eines GameObjects in einem spezifischen Moment (Der Moment ihrer Initialisierung) bezeichnet werden. Sie verhalten sich immer genau gleich wie das Ursprungs Prefab soweit sie nich individuell zur Laufzeit angepasst werden. Ein solches System ist für FuseeAT vorerst nicht geplant, könnte sich aber in der Zukunft als nützlich erweisen. Fusee kann bereits in Cinema 4D gemodelte Objekte oder Modelle in der Fusee Engine mit Hilfe von Prototype serialisieren, allerdings ist dieses System noch nicht zu Ende implementiert und wird stetig erweitert.

Um Assets in Unity zu bearbeiten, zu verwalten und zu durchsuchen steht ein Contentbrowser, siehe Abbildung 4.14, mit eben genau diesen Funktionen zur Verfügung. Grundsätzlich bildet dieser Contentbrowser alle Möglichkeiten ab, die auch FuseeAT unterstützen möchte.

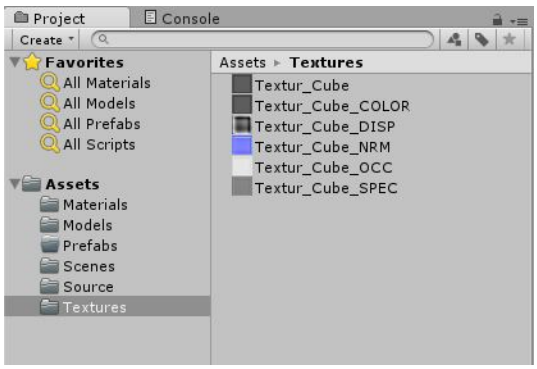


Abbildung 4.14: Ansicht des Contentbrowsers in Unity. Mit Suchfeld und Filterfunktionen und ausklappbarer Ordnerstruktur.

Szenengraph in Unity

Der Unity Szenengraph gleicht dem Fusee Szenengraph. In der Tat lehnt sich das Design des Fusee Szenengraphen am System der Unity Engine an, denn beide basieren auf Nodes welche auf der gleichen Ebene durch Components erweitert werden. Kindobjekte von Nodes sind wiederum weitere Nodes. Dieses Design des Szenengraphen ist in Heutigen Game Engine bereits als eine Best Practice Lösung anzusehn und könnte aus diesem Grund auch Problemlos in FuseeAT integriert werden. Das System kann schnell durchdrungen und adaptiert werden.

Was kann für das FuseeAT Konzept aus Unity übernommen werden?

Aus den Funktionen der Unity Engine können verschiedene interessante Konzepte gefunden und für FuseeAT adaptiert werden. Hierzu gehört zu aller erst das Component System. Die Unreal Engine setzt wie bereits erwähnt ebenfalls auf ein Component System und das neue Szenengraphen Modell in Fusee unterstützt diesen Ansatz ebenfalls. So sollte auch FuseeAT dieses Konzept berücksichtigen und die in Fusee möglichen Components auch in den Cinema 4D Editor integrieren. Weiterhin ist das einfache Build und Deploymentsystem von Unity eine stärke der Engine. Für FuseeAT wäre es ein guter Ansatz die Arbeit der Designer und Artists mit der Fusee Engine in Cinema 4D zu erleichtern wenn Builds für den Desktop und später evtl. auch Web-Builds eines gesamten Fusee Projekts (welche aktuell wie bereits erwähnt schon mit der Fusee Engine möglich sind) aus der C4D Software heraus erstellt werden könnten. Hiermit können Designer und auch Artists jederzeit das Ergebnis ihrer Arbeit durch einen einfachen Build Vorgang direkt in der Engine betrachten.

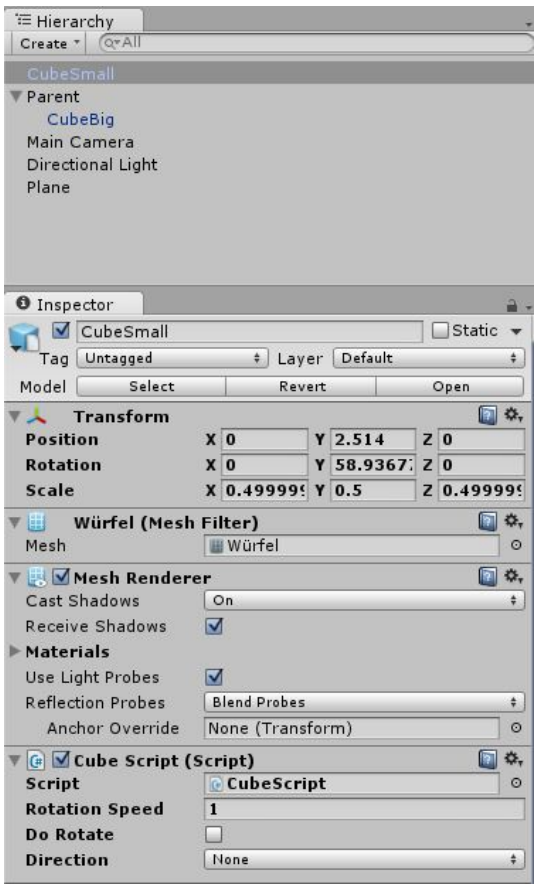


Abbildung 4.15: Beispielhafte Szene im Szenengraph von Unity. Er gleicht dem Fusee und Cinema 4D Szenengraphen stark. Das Detailpanel zeigt Components eines Nodes.

4.6 Systemdesign

Das Ziel hier ist es, eine technische Konzeption für ein Softwareprodukt zu entwickeln. Dieses technische Konzept ist Grundlage für eine nachfolgende Implementierung des Softwareprojektes FuseeAT. Einige Features werden diskutiert und die Entscheidung für oder gegen diese erläutert.

4.6.1 Warum Fusee und Cinema 4D?

Für diese Arbeit wurden die beiden Applikationen Fusee[9] und Cinema 4[[MAX15] aus verschiedenen Gründen als Ausgangsbasis für die Forschung ausgewählt. Das komplette Fusee Projekt inklusive seiner Abhängigkeiten ist von GitHub[10] zu beziehen - was eine Integration des hier in dieser Arbeit angedachten Projektes einfacher gestaltet. Weiterhin wurde das Uniplug Projekt in die Fusee Engine integriert und nutzt deren Formate, Strukturen und einen Teil der Codebase. Hierbei handelt es sich um verschiedene Mathematik Bibliotheken und

[9]Furtwangen University Simulation and Entertainment Engine
[10]https://github.com/FuseeProjectTeam/Fusee

Tools wie den Fusee Project Generator.
Der 3D Modeling Editor Cinema 4D steht kostenfrei für Studierende der Hochschule Furtwangen zur Verfügung und bietet eine relativ gut dokumentiert API. Weiterhin ist das Projekt Uniplug auf die Verwendung mit Cinema 4D hin konzipiert und unterstützt zum Zeitpunkt der Erstellung dieser Arbeit noch keine weitere Software. Ein großer Teil der Cinema 4D C++ API war bereits in das Uniplug Tool integriert bzw. von C++ nach C# konvertiert / gewrapped. So konnte relativ Problemlos mit der Arbeit begonnen werden.

Hinweis: Alle hier verwendeten Diagramme finden sich im Anhang in einer höher aufgelösten Version im Format Din A4.

4.6.2 Systemdesign für das Toolkit

Das Fusee Authoring Toolkit (kurz FuseeAT), in der folgenden Abbildung 4.16 in grün dargestellt, ist das Kernsystem dieses Konzeptes. Es handelt sich hierbei um eine Sammlung von Funktionen in Form einer C# Softwarebibliothek die es ermöglichen soll einen Editor[11] in ein Game Authoring Tool für die Erstellung von Interaktiven Welten zu transformieren. Wie bereits früher in der Abbildung 4.1 erwähnt, ist das FuseeAT als Standalone Tool architecture konzipiert. Hierbei ist das FuseeAT Framework parallel zur Engine verortet und nicht wie bei Built In Architekturen direkt in die Engine integriert. Ein Beispiel für die Integration des Editors sind die Engine ist die Unreal Engine 3 und die Unreal Engine 4 Vgl. [Ger09, S. 54-55].
Die grau hinterlegten Module der Abbildung 4.16 beschreiben Systeme welche in das Projekt mit einbezogen wurden, an deren Code jedoch nichts verändert wurde. Der "Fusee Project Generator" wurde zwar aus seiner ursprünglichen Form, einer als .Exe gebauten C# Anwendung, in ein neues C# Projekt innerhalb des FuseeAT verlagert. Das Projekt ist jetzt als Bibliothek zum Einbinden in C# Projekte verfügbar. An seiner Funktionsweise hat sich aber nichts getan.
Die drei dargestellten Akteure verdeutlichen die getrennten Welten der jeweiligen Positionen während des Entwicklungsprozesses eines Interaktiven Software Produktes, z.B. eines Spiels und ihrer Sichtweise auf dieses Projekt. Das gelb hinterlegte Fusee Projekt beschreibt ein sich in Produktion befindliches Engine Projekt. Im Unterschied zu den aktuell meisten Fusee Projekten verwendet es allerdings für den Verlauf dieser Arbeit einen Binary Build der Engine. Dies hat den Vorteil, dass nicht der komplette Engine Code während der Arbeit für ein einziges Projekt mitgetragen werden muss. Der Engine Code direkt, wird

[11] Modeling Editor, Welt Editor, in Zukunft möglicherweise auch dem bereits früher erwähnten Sony "Level Editor"

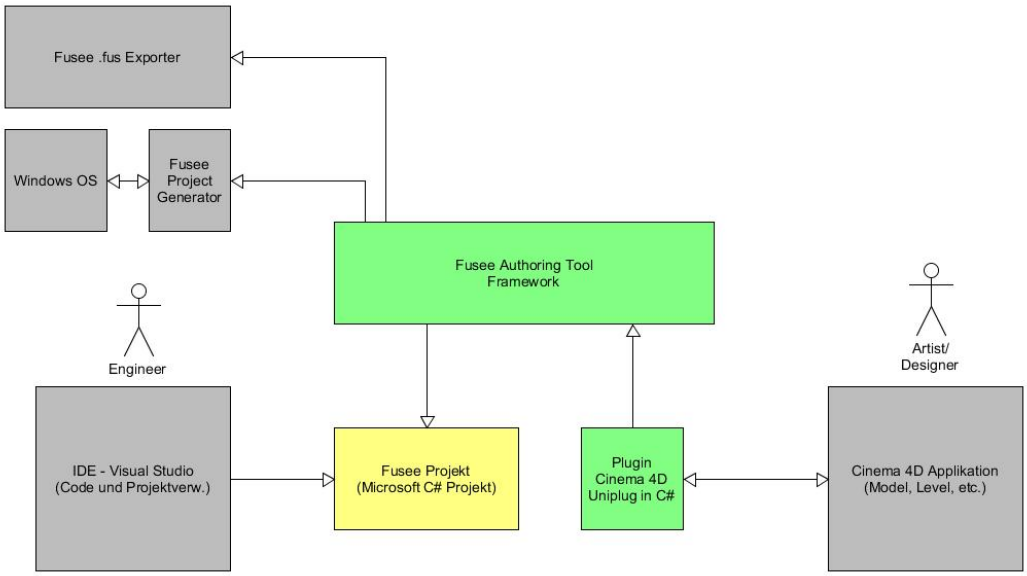


Abbildung 4.16: Überblick über die Systemarchitektur

während dieser Arbeit nicht erweitert, sie beschränkt sich lediglich auf die Erweiterung des Uniplug Projektes welches hier im Plugin Cinema 4D Uniplug in C# Container dargestellt ist. Das Uniplug Projekt ist nicht direkt im Kern der Engine angesiedelt und lässt sich deshalb getrennt von dieser bearbeiten.

Damit das System erfolgreich arbeiten kann ist es nötig, aus Cinema 4D verschiedene Information an das FuseeAT zu übertragen. So muss im Plugin zu Cinema 4D ein Fusee Projekt geöffnet werden. Dies kann über einen UI Dialog geschehen, der die vom Benutzer eingetragenen Informationen über den Projektort auf der Festplatte des Nutzers dann an das FuseeAT weiterleitet. Somit kann das FuseeAT dann die komplette Steuerung des Projektes übernehmen und Cinema 4D stellt nur eine Benutzerschnittstelle zur Interaktion dar. Somit wäre jegliche Funktionalität des FuseeAT in das eigentliche Authoring Framework ausgelagert und somit vom Programm Cinema 4D (oder anderen) abgekoppelt und unabhängig. Das Cinema 4D Plugin, in der Abbildung 4.16 ebenfalls in grün dargestellt, stellt also nur die Schnittstelle des FuseeAT zur Software Cinema 4D dar und dient in diesem Fall als Kommunikationssystem zwischen den verschiedenen Architekturen C++ und C#.

4.6.3 Systemdesign für ein Tool Framework

Die folgende Abbildung 4.17 zeigt das Design für das FuseeAT im Überblick. Der grau hinterlegte Container steht für die mit Hilfe des Uniplug Projektes zu entwickelnde Cinema 4D API Benutzer Schnittstelle. Das FuseeAT kann

in vier Bereiche aufgeteilt werden. Zuerst einmal wäre hier das Schnittstellen Modul des FuseeAT für die Kommunikation mit einem Plugin das außerhalb des Systems verankert ist. Dieses FuseeAT Modul nimmt jegliche Calls (z. Dt. Aufrufe) entgegen und wandelt sie in Operationen des FuseeAT um. Hierzu verwendet weitere Module. Zu den Modulen gehören:

- FuseeAT Projekt Manager
- FuseeAT File I/O Manager
- FuseeAT Build Manager

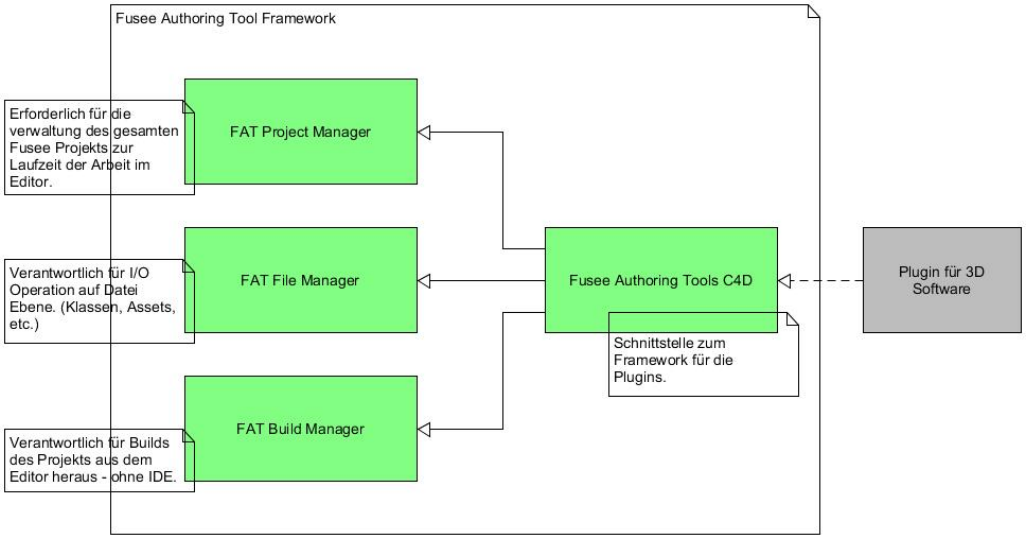


Abbildung 4.17: Überblick über das Fusee Authoring Toolkit Framework

Diese Module teilen sich die Aufgaben des FuseeAT. Der Projekt Manager behandelt hierbei jegliche Aufgaben die im Zusammenhang mit der Verwaltung, zur Laufzeit, eines FuseeAT (bzw. Fusee Engine) Projektes stehen. Das Modul FuseeAT File Manager ist ein Modul welches sich um jegliche I/O (kurz für Input Output) Operationen kümmert. Hier werden Tasks auf Betriebssystem Dateiebene erledigt. Dazu gehört beispielsweise das Erstellen und das Speichern von Dateien. Dieses Modul kann weiterhin für das anlegen von C# Klassen im Bereich der Programmierung zuständig sein und somit die Verbindung zwischen Assets und Code herstellen können. Der FuseeAT Build Manager ermöglicht das Bauen eines C# Projektes durch Verwenden des von Microsoft bereitgestellten MSBuild[12] Systems. Er kümmert sich um Ausgabeverzeichnisse und weitere Optionen die während eines Buildprozesses Angaben

[12]https://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx

des Benutzers benötigen. Der Buildprozess könnte durch die Plugin Kommunikation direkt aus Cinema 4D heraus angestoßen werden um das Fusee C# Projekt zu erstellen.
Das Klassendiagramm in Abbildung 4.18 beschreibt in einer Detaillierteren Variante das FuseeAT. Die einzelnen Module sind hier in Klassen aufgeteilt und ein Teil der benötigten Methoden ist enthalten. Das Interface FuseeATBase (hier in Gelb hinterlegt) bildet die Ausgangsposition. Es bietet dem Cinema 4D Plugin System die Möglichkeit mit dem FuseeAT zu kommunizieren. Das FuseeAT wurde so designed, dass es Softwareunabhängig implementiert werden kann. FuseeATCinema 4D implementiert dieses Interface und verteilt die Aufgaben weiter an den ProjektManager. Dieses Modul kommuniziert selbst mit dem FileManager und dem BuildManager um die gewünschten Operationen zur Laufzeit zu erledigen. Das FuseeATCinema 4D Modul erhält Rückmeldungen über die return Werte der aufgerufenen Funktionen und kann somit dem Plugincode Rückmeldung über den Erfolg einer Aktion geben. Somit ist es möglich einem Benutzer in gewisser Weise ein Feedback zu seiner gerade ausgeführten Operation zu geben. Bei diesen Operationen könnte es sich z.B. um das Anlegen einer Datei, oder das Öffnen eines Projektes handeln. Das Klas-

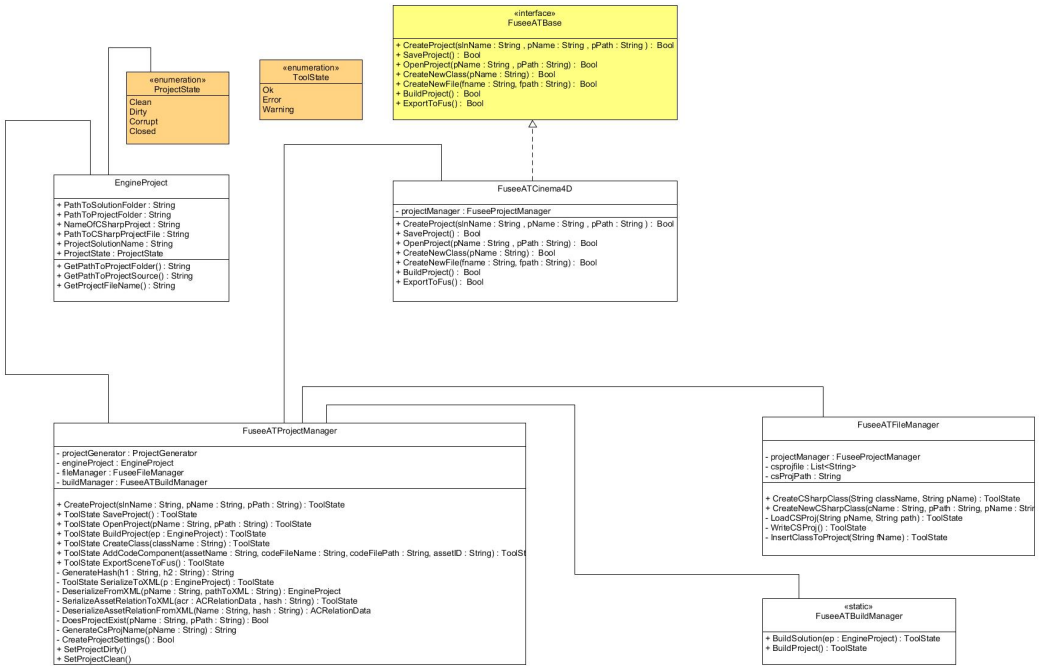


Abbildung 4.18: Klassendiagramm für das Fusee Authoring Toolkit

sendiagramm 4.18 zeig bereits die Kompositionen der einzelnen Klassen auf. Die beiden Enumerationen ProjectState und ToolState (in Orange hinterlegt) dienen dazu die Integrität eines Fusee Engine-Projektes zu jeder Zeit zu gewährleisten. Sie werden als Rückgabewerte der Schnittstellen Funktionen nach

außen eingesetzt und können somit einen möglichen Entwickler, welcher das FuseeAT erweitern möchte, darüber Informieren ob eine Aktion erfolgreich war oder nicht. Der Entwickler kann dann entscheiden, wie er mit der gewonnenen Information umgehen möchte. In manchen Situationen würde es Sinn machen auf diese Information, evtl. mit einen Rollback oder Ähnlichem zu reagieren. Hier sei angemerkt, dass sich mit dem vormals erwähnten ATF Framework von Sony ein solches Do und Undo System in C# realisieren ließe. Es bietet laut Dokumentation ein solches System zur Integration an.

4.6.4 Systemdesign für ein Plugin System

Die Abbildung 4.19 bietet einen Überblick über das Design des Uniplug Teil des Projektes. Damit verschiedene Funktionen in Cinema 4D angeboten werden können, benötigt das gesamte System verschiedene Arten von Plugin Typen. Das CommandPlugin designed zum öffnen, bauen und speichern eines Projektes. Es wird vom Benutzer durch den Aufruf eines Dialogs im Cinema 4D Plugin Menü gesteuert.

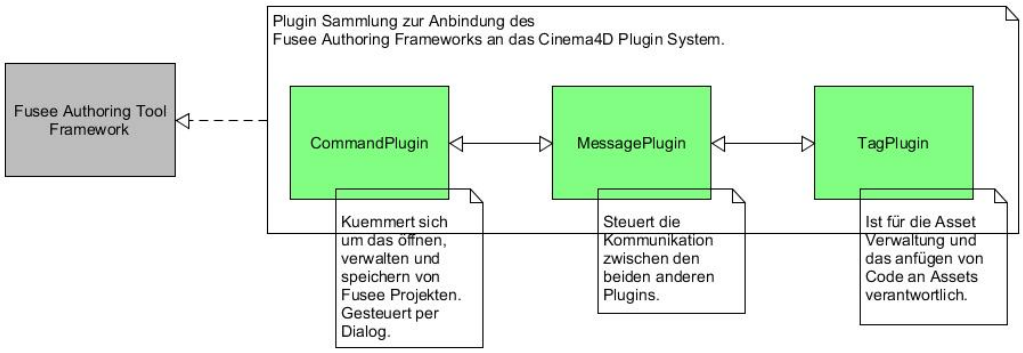


Abbildung 4.19: Überblick über die Pluginarchitektur

Das TagPlugin kümmert sich um die Zuordnung von Assets. Es ermöglicht es in Cinema 4D so genannte Tags, welche Informationen in verschiedenen Formen speichern können, an ein 3D Objekt zu heften (in Form eines Icons im Szenengraphen) und so eine Verbindung zwischen Code und Content herzustellen. Das MessagePlugin ist ein Hilfsplugin und implementiert keine Funktionalität sondern unterstützt die per Nachrichtensystem gesteuerte Kommunikation innerhalb von Cinema 4D. Durch dieses Plugin können Nachrichten des CommandPlugin an das TagPlugin und umgekehrt versandt werden. Die Nachrichten werden dann in den jeweiligen Plugins geparsed (sie bestehen aus einer Integer ID und einem Datenanteil[13]) und weiter verarbeitet. Das hier angeführte Klassendiagramm 4.20 zeigt die für jedes Plugin benötigten

[13]Von der Cinema 4D API festegelegt.

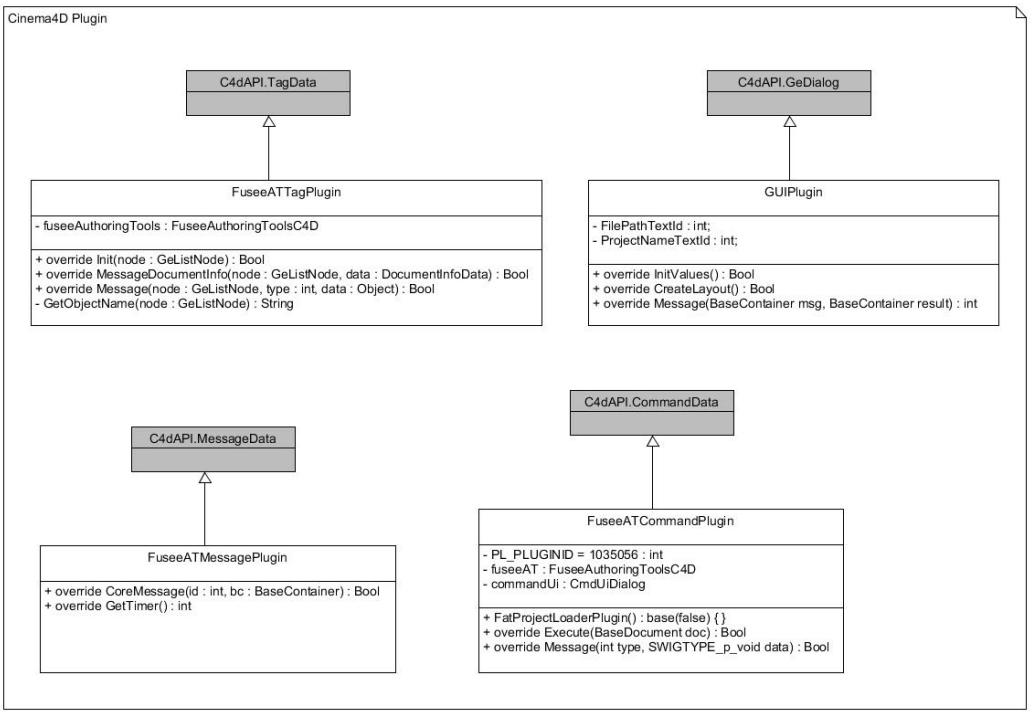


Abbildung 4.20: Klassendiagramm der Plugin Architektur

API Funktionen. Diese Funktionen müssen im Plugin Code als überschriebene Funktionen implementiert werden. Die Klasse GUIPlugin implementiert ein UserInterface um ein Fusee Projekt von Cinema 4D aus über einen Grafischen UI Dialog zu laden.

4.6.5 Zeitersparnis durch wiederverwendbare Fusee Module

Um nicht jegliche Funktionen neu implementieren zu müssen, wurde vor dem Beginn der Implementierung geprüft welche Funktionalitäten bereits in der Fusee Engine gegeben sind. Durch die offene Struktur der Engine konnte bereits früher schon Code wiederverwendet werden. Diese Synergien der bereits vorhandenen Systemen sollten auch in zukünftigen Entwicklungen genutzt werden um, Zeit und Ressourcen zu sparen. Die folgende Module des Fusee Projekts wurden für den Rahmen dieser Arbeit angepasst und verwendet.

Der Fusee Projekt Generator

Der Fusee Project Generator ist ein Projekt das bereits vom ersten Fusee Entwicklerteam angestoßen wurde. Es handelt sich hierbei um ein Bibliotheksmodul welches es ermöglicht, ein neues C# Projekt innerhalb einer Fusee Solution anzulegen. Im Rahmen dieser Arbeit wurde der Fusee Projekt Generator lediglich neu innerhalb des Uniplug Projektes verdrahtet. Davor war der Generator

noch eine eigenständige Konsolen .Exe Anwendung. Aktuell können mehrere Projekte mit ein und dem selben Engine Binary Build gebaut und verdrahtet werden. Die folgende Abbildung 4.21 zeigt die Struktur eines solchen Fusee Projektes. Die Solution Datei beinhaltet die zwei Projekte “Simple” und “Test-

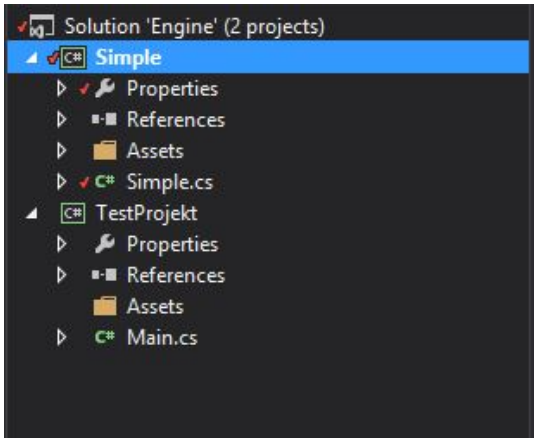


Abbildung 4.21: Fusee Binary Projekt Struktur

Projekt”. Diese Projekte beherbergen in den Referenzen jeweils die Binary Bibliotheken der Fusee Engine. Das Projekt TestProjekt wurde mit dem Fusee Generator angelegt. Hierfür benötigt der Generatur nur den gewünschten Namen des Projekts und den Pfad zum Fusee Binary Projekt auf der Festplatte des Benutzers.

Fusee .fus Exporter

Beim Fusee Exporter handelt es sich um ein von Herrn Prof. C. Müller (Hochschule Furtwangen, Fakultät Digitale Medien) entwickeltes Cinema 4D Plugin. Das Plugin wurde ebenfalls mit Hilfe des Projektes Uniplug entwickelt. Es ermöglicht das exportieren einer Cinema 4D Szene in ein von Fusee lesbares Format (.fus). Dieses binary Format lässt sich in Fusee direkt im Code einer Engine Applikation laden und enthält sowohl Geometrie als auch weitere Bestandteile wie z.B. Materialien. Das Systemdesign sieht vor, dieses Plugin in das FuseeAT zu integrieren. Es erfüllt die Aufgabe des exports von Model bzw. Szenendateien in das Fusee Format zufriedenstellend und muss für eine Implementierung noch aus der Form eines Plugins in die Form einer Softwarebibliothek gebracht werden. Der Fusee Exporter ist an Cinema 4D gebunden und kann daher lediglich für den Export aus Cinema 4D genutzt werden. Für eine Verwendung in anderen Programmen müsste eine Schnittstelle geschaffen werden. Der Fusee Exporter operiert direkt auf den von Maxon in der API bereitgestellten Datentypen und Methoden einer Cinema 4D Szene. Das Plugin bietet weiterhin die Möglichkeit eine Szene aus Cinema 4D in eine im Web

Browser lauffähige Version einer Fusee Szene zu exportieren. Dies geschieht mittels der Verwendung von JSIL und verschiedener Javascript Bibliotheken. Weitere Informationen finden sich im Wiki[14] zu Fusee.

4.7 Asset Management und Asset Pipeline in der Fusee Engine

Da sich die Fusee Engine in stetiger Entwicklung befindet, gibt es bis jetzt noch keine gesondert etablierte Asset Management Pipeline. Assets werden bis jetzt in Visual Studio direkt im Code per Include eingebunden. Genau so wie alle Fusee Projekte ohne grafische Oberfläche in Visual Studio erstellt werden müssen. Meist handelt es sich bei in Fusee eingebundenen Assets um eines der folgenden Asset-Typen:

- 3D-Modelle in den Formaten .obj[15] (Bei dem Dateiformat handelt es sich um das lesbare ASCII Model Datenformat "Wavefront Object"[16])
- 2D-Texturen in den Formaten (JPEG, JPG, PNG)
- Audio Datei im Format Wav, Ogg, etc. (Unterstützt alle von der SFML "Simple and Fast Multimedia Library http://www.sfml-dev.org/ unterstützen Formate.

4.7.1 Asset Pipelines in Fusee Authoring Toolkit

Die folgende Grafik 4.22 [17] ergänzt die Erläuterung zum Asset Authoring (Eines .fus Objektes mit angehängtem Code Component[18] oder anderen) mit Hilfe der Asset Pipeline während der Produktion eines Fusee Engine Projektes. Hierbei handelt es sich um einen iterativen Prozess welcher je nach Ergebnis des Tests und der Evaluation im Engine Projekt eine neue Iteration des Prozesses nach sich ziehen kann. In der Grafik 4.22 wird Beispielhaft ein Model in Cinema 4D erstellt, ein Component angehängt und anschließend vom FuseeAT System zur Laufzeit zu einer XML Datei serialisiert. Diese XML Datei kann dann von jedem Entwickler editiert werden. Die Datei wird weiterhin von FuseeAT in der Cinema 4D Szene benötigt um eine Zuordnung eines Asset in

[14] Fusee Wiki zum Web Export: https://github.com/FuseeProjectTeam/Fusee/wiki/HowTo:-Fusee-on-Web

[15] Weitere Informationen zum Format unter: http://www.fileformat.info/format/wavefrontobj/egff.htm Online: geprüft am 27.04.2015

[16] Entwickelt von Wavefront, das Unternehmen war später unter dem Namen Alias bekannt und wurde zwischenzeitlich von Autodesk aufgekauft http://www.autodesk.de/

[17] Abbildung entnommen aus [Car04, S. 7] und neu Illustriert und beschriftet.

[18] Ein Extra Element welches an einem Asset Objekt angehängt ist

der Szene wiederherzustellen. Das erstellte Asset bleibt weiterhin im Cinema 4D Format, gespeichert in der Cinema 4D Datei des Projektes, vorhanden. Somit besteht das Asset in einer nativen Version weiterhin und kann in einem iterativen Prozess weiterhin verändert werden.

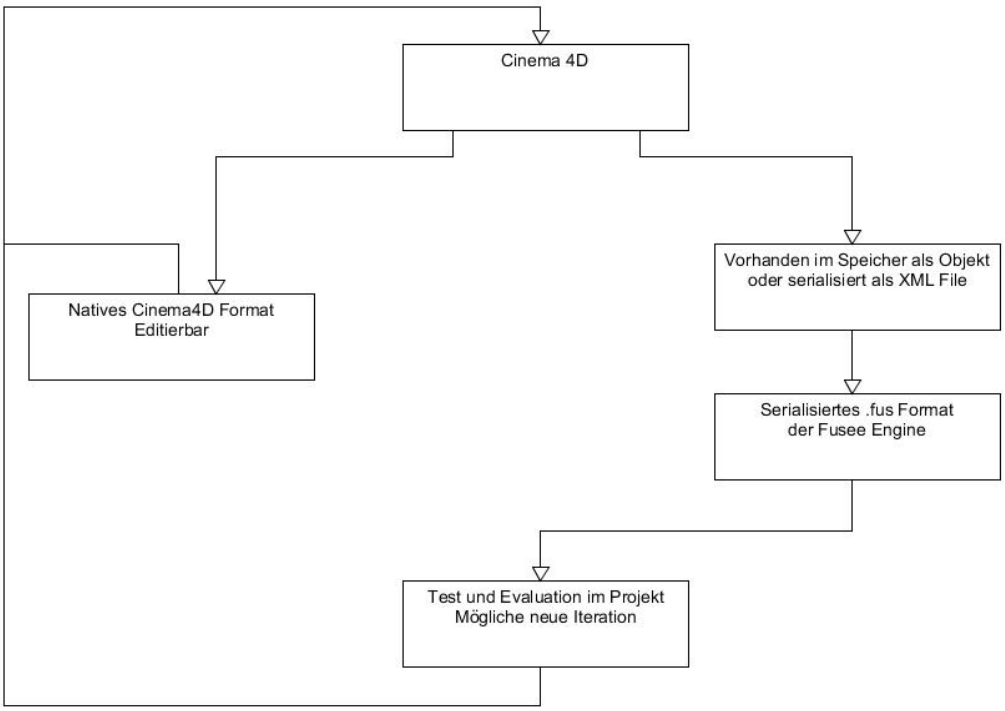


Abbildung 4.22: Asset Loop während der Produktion eines Projektes in Fusee und FuseeAT

Während des Build- und/oder Speichervorgangs des Cinema 4D und Fusee Projekts über das FuseeAT wird das Asset vom Framework in ein Asset des Fusee Typs .fus gewandelt. Allerdings geht durch die bereits erwähnte redundante Speicherung des Asset im Cinema 4D Format keine Information der ursprünglichen Datei verloren.

4.7.2 Assetmanagement in Fusee Authoring Toolkit

Oft reicht eine einfache Versionierung, durch z.B. Git, von binären Asset Dateien wie Texturen oder Modellen nicht aus und muss daher von geeigneteren Tools übernommen werden. Das Assetmanagement innerhalb eines Projektes wird von FuseeAT nicht abgedeckt und sollte vom Entwicklerteam übernommen werden. Hierzu können verschiedene Tools eingesetzt werden die auf ein Management von Binary Asset Files spezialisiert sind. Beispiele für solche Systeme sind die Software Alienbrain [Avi09][19] und die Software Shotgun

[19]Webadresse: http://www.alienbrain.com/

[Sho13][20]. Diese kommerziellen Tools erlauben das Verwalten von Binären Assets (z.B. Audio Dateien, 3D-Modelle, Grafiken) während der Produktion von Entertainment Produkten. Eine solche Software kann ohne Probleme neben dem FuseeAT eingesetzt werden, da sich die beiden Wirkungsbereiche nicht überschneiden. Da FuseeAT jedoch auf human readable file formats setzt und diese als XML Dateien im Projekt Ordner ablegt, kann für die Versionierung von FuseeAT Projekten ein VCS wie Git herangezogen werden. Ein weiterer Vorteil der lesbaren XML Dateien ist der unkomplizierte Merge-Vorgang innerhalb des VCS, Bei Problemen können Entwickler diese Dateien auch jederzeit von Hand verändern und das Projekt in Cinema 4D erneut laden.

[20]https://www.shotgunsoftware.com/

5 Umsetzung des Konzeptes

5.1 Die Cinema 4D C++ API und deren Verwendung in diesem Projekt

Für die Implementierung in Fusee und dem Cinema 4D Pluginsystem war es nötig, auf das Uniplug System der Hochschule Furtwangen [1] zurückzugreifen. Damit das Uniplug System in dieser Arbeit genutzt werden konnte, waren einige Erweiterungen nötig. Da dies einen erheblichen Zeitaufwand vor und während der Implementierung bedingte und eine Abhängigkeit von FuseeAT darstellt, soll hier darauf eingegangen werden.

5.1.1 Ausgangssituation und in der Implementierung verwendete Softwareprojekte

Das Projekt "Fusee Authoring Toolkit" [2] (kurz: FuseeAT) basiert auf mehreren Software Projekten. Ein Teil dieser Projekte wurde an der Hochschule Furtwangen entwickelt. Ein anderer Teil stammt von externen Entwicklen und gehört zu proprietärer Software. Dieser Abschnitt gibt einen Überblick über die verwendeten Softwarebestandteile und ihrer Einbindung in der Implementierung zur Zeit dieser Arbeit.

Verwendet wurden:

- Fusee Math Bibliothek, eine von Fusee unabhängige Sammlung in C# geschriebener Mathematischer Funktionen und Datentypen

- Uniplug, ein in Fusee beinhaltetes Projekt zum Schreiben von Plugins in C# für Cinema 4D

[1] Entstanden durch die Leitung und Entwicklung von Herrn. Prof. C. Müller, Fakultät Digitale Medien, in ständiger Weiterentwicklung durch Mitglieder und Studierende des Fusee Teams.

[2] Der Name der zu dieser Arbeit zugehörigen Softwarebibliothek.

- Die Fusee Engine, eine Interaktive Simulations Engine zur Darstellung zweidimensionaler und dreidimensionaler Szenen

- Die Maxon Cinema 4D API(Hier sind nur C++ Headerfiles verfügbar. Jeglicher implementierter Cinema 4D Programmcode ist nicht einsehbar.)

- Verschiedene Windows Bibliotheken aus dem .NET Framework auf welche hier nicht weiter eingegangen werden soll

5.1.2 Cinema 4D Plugin API und SDK

Cinema 4D R16 [MAX15] ist ein kommerzielles proprietäres Produkt des Unternehmens MAXON Computer GmbH und steht nicht als Open Source Projekt zur Verfügung. Maxon stellt daher für die Entwicklung von Plugins[3] und die Erweiterung des Cinema 4D Funktionsumfangs eine API[4] bereit. Bei dieser API handelt es sich um eine Bibliothek geschrieben in C++. Dieser Code kann in Form von Header Files in die eigene Applikation integriert werden. Durch diese C++ Headerfiles ergibt sich eine Schnittstelle zu internen Methoden in der Cinema 4D Software. Die tatsächliche Implementierung kann mit der API nicht eingesehen werden, es handelt sich hier lediglich um Header Files ohne den tatsächlichen Code. Das C++ SDK wird bei einer Installation von Cinema 4D automatisch mit installiert. Auf einem Windows PC findet sich das SDK und die dazugehörigen Visual Studio Solution Dateien unter folgendem Pfad ausgehend vom Installations Ordner der 64Bit Version von C4D[5]: "Cinema 4D/plugins/cinema4dsdk".
Maxon bietet Beispiele zur Cinema 4D API auf einem GitHub Account[6] unter der Apache License Version 2.0, January 2004[7] zum kostenfreien Download an. Das C++ SDK/API ist seit der Version R16 fester Bestandteil der Cinema 4D Installation und muss nicht extra heruntergeladen oder gebaut werden.
Voraussetzung für das Erstellen eines Plugins ist, dass der eigene Plugin Code laut Maxons Dokumentation einige Funktionen überschreiben bzw. Klassen vererben muss. Ansonsten ist der Inhalt des Quellcodes seitens Maxon nicht beschränkt. Das folgende Schaubild 5.1 erläutert die Funktionsweise des Cinema 4D Plugin Systems. Der Entwickler kann seinen C++ Code fast frei

[3]Plugins in C4D - Erweiterungen für die Software Cinema 4D und deren Funktionen
[4]Application Programming Interface, z. Dt. Programmierschnittstelle
[5]Cinema 4D liegt seit der Version R15 nur noch als 64 Bit Version vor und benötigt laut http://www.maxon.net/?id=311(Maxon C4D System Requirements) ein 64 Bit System
[6]https://github.com/PluginCafe
[7]Apache Licence 2.0 https://github.com/PluginCafe/cinema4d_cpp_sdk/blob/master/LICENSE

entwickeln und ist nur dazu angehalten sich an Maxons Richtlinien zur Kommunikation mit dem Produkt Cinema 4D zu halten. Der Entwickler hat dabei die volle Kontrolle über seinen eigenen Plugin Code. Die Schnittstelle (C4D API/SDK) ist zumindest durch die mitgelieferten C++ Headerfiles noch einsehbar. Dies unterstützt den Entwickler eines Plugins während des debuggens. Sobald Aufrufe die eigentliche Software Cinema 4D erreichen, ist das einsehen des Codes nicht mehr möglich. Was wie später noch erläutert im Falle von Uniplug zu einigen Problemen führen kann.

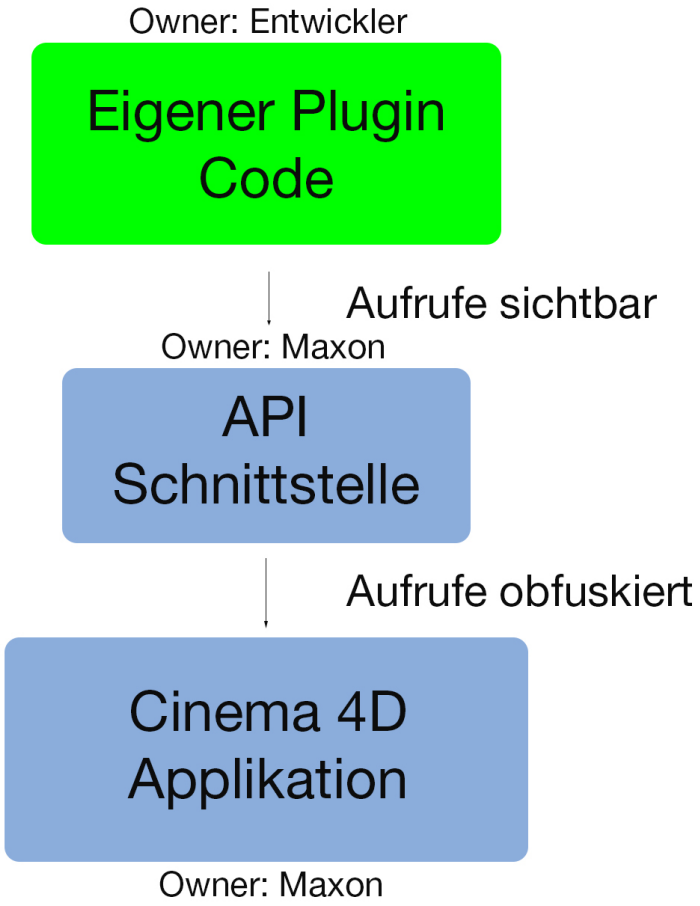


Abbildung 5.1: Schaubild des Cinema 4D API Systems

Das hier dargestellte Minimalbeispiel 5.1 in C++ Code zeigt ein einfaches minimalistisches Plugin welches beim Ausführen nur eine Ausgabe auf der Konsole von Cinema 4D erzeugt.

Listing 5.1: Einfaches Cinema 4D Plugin in der Programmiersprache C++

```
1  #define ID_PLUGINID 1234567 // Dies ist eine einzigartige
       Plugin ID.
2
3  #include "c4d.h"
4
5  class MinimalesPlugin : public CommandData // Das Plugin ist
       vom Plugin Typ "Command Plugin"
6  {
7  public:
8    virtual Bool Execute(BaseDocument *doc)
9    {
10     GePrint("Eine Ausgabe zum Cinema 4D Konsolenfenster.");
11     return TRUE;
12   }
13 };
14
15 // Mit dieser funktion steigt das Plugin ein.
16 Bool PluginStart(void)
17 {
18   return RegisterCommandPlugin(ID_PLUGINID,"MinimalesPlugin"
       , 0,NULL,String("MinimalesPlugin"), gNew MinimalesPlugin)
       ;
19 }
20
21  // Wird beim beenden des Plugin aufgerufen.
22 void PluginEnd(void)
23 {
24 }
25
26 // Ist teil des Message Systems von C4d. Dient der
       Kommunikation.
27 Bool PluginMessage(LONG id, void *data)
28 {
29   return TRUE;
30 }
```

Dieser Code ist ein Wichtiger Bestandteil der Cinema 4D API und letztendlich auch des FuseeAT. Das Beispiel wird im nächsten Abschnitt relevant. Es folgt ein Überblick über das Uniplug Projekt welches das Schreiben von Cinema 4D Plugins mit Hilfe der Programmiersprache C# ermöglicht. Die oben gezeigte

Codestruktur wird dann noch einmal als C# Version erläutert.

5.1.3 Uniplug

Bei Uniplug handelt es sich um ein Teil des Projektes Fusee der Hochschule Furtwangen. Uniplug bietet die Möglichkeit Cinema 4D Plugins in C# zu schreiben. Hierfür bedient es sich eines Interface Compilers SWIG [SWIG.org Beazley, David Fulton, William. *SWIG-3.0*. http://www.swig.org/. 1995-2015. (Geprüft am 18. 05. 2015)]. Das gesamte Uniplug Projekt ist ein Open Source Projekt und steht auf einem GitHub Repository unter https://github.com/FuseeProjectTeam/Fusee innerhalb des Fusee Projekts zum Download bereit. Das Uniplug Projekt stand bereits vor dieser Arbeit zur Verfügung. Allerdings hatte es nich den nötigen Umfang und das Projekt wurde während der Arbeit um wichtige Funktionen zur Entwicklung von Plugins erweitert. Folgende in Abbildung 5.2 dargestellten Module sind Bestandteil des Uniplug Systems.

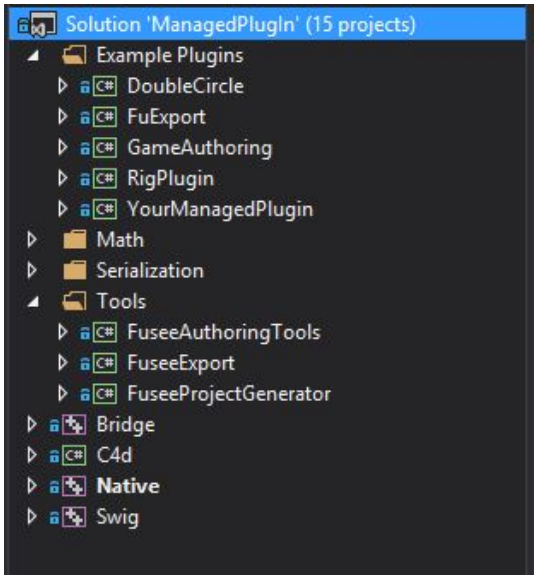


Abbildung 5.2: Die Uniplug Projektstruktur in VisualStudio

Die im Unterordner Tools aufgeführten Projekte sind das Kernelement dieser Arbeit. Es handelt sich dort um das FuseeAT. Das Plugin für die Cinema 4D Anbindung ist im Unterordner “Example Plugins” verortet und wird hier als “Game Authoring” bezeichnet. Es enthält jeglichen Plugin Code für diese Arbeit. Die Projektstruktur des UniPlug Projektes wurde während dieser Arbeit und dem Einfügen der neuen Funktionen weiter optimiert.

Was ist SWIG?

Bei SWIG handelt es sich um einen SoftWare Interface Generator. Swig unterstützt Entwickler dabei, eine Codebase aus z.B. einer nativen Programmier-

sprache wie C++ in eine managed Programmiersprache wie z.B. C# zu "übersetzen". Hierbei handelt es sich allerdings nicht wirklich um einen übersetzungs bzw. Compilevorgang. Vielmehr unterstützt Swig den Entwickler durch das generieren spezifischer Interface Files welche die Aufrufe (calls) einer "externen" Software an den "gewarppten" Teil der Software weiterleiten. Die Ausgangssoftware wird also immernoch benötigt (im Falle von Uniplug) und auch verwendet (das Cinema 4D API Framework). Vgl Fulton, William Beazley, David. *Swig Introduction Documentation*. Available online http://www.swig.org/exec.html. 2014. (Geprüft am 15. 04. 2015).
Eine Vollständige Dokumentation des SWIG Projektes findet sich unter folgender Adresse: http://www.swig.org/Doc3.0/index.html geprüft, letzter Stand der Erreichbarkeit 15.04.2015.
Swig wird bereits erfolgreich in vielen Projekten eingesetzt. Unter anderem finden sich in der Liste das bekannte Version Control System Subversion, die 3D Engine Ogre bzw. PyOgre und die große Image Processing Bibliothek OpenCV. Die offizielle Liste der SWIG Nutzer findet sich unter http://www.swig.org/projects.html - geprüft am 15.04.2015.

Wrapping von C++ Code nach C# mit SWIG

Um aus dem C++ Code der Cinema 4D API aufrufbaren C# Code zu erzeugen sind verschiedene Schritte notwendig. Ein SWIG Projekt welches wie in Abbildung 5.3 im Uniplug Projekt verfügbar ist enthält eine C4dApi.i (interface) Datei.

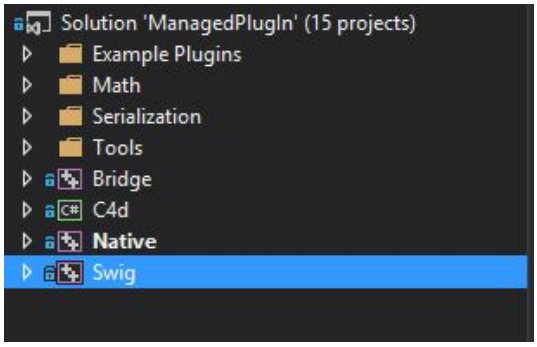


Abbildung 5.3: SWIG im Uniplug Projekt (ManagedPlugIn) markiert durch blaue Selektion.

Diese Datei kümmert sich um das Wrappen des nativen C++ Codes nach C#. Der eigentliche Wrapping Vorgang wird über ein SWIG make script aufgerufen welches während des Build Vorgangs des Projektes angestoßen wird. Dieser Vorgang muss bei der eigentlichen Plugin Entwicklung nicht bei jedem Build durchgeführt werden. Es genügt, die API erneut zu wrappen wenn Ver-

änderungen am API Code seitens Maxon oder des Benutzers vorgenommen wurden. Änderungen durch den Benutzer schließen auch das Erweitern der API Funktionalität mit ein. Das Uniplug Projekt ist zum Zeitpunkt dieser Arbeit weit von einem Vollständigen Wrapping der Cinema 4D API entfernt. Aus diesem Grund, ist noch relativ häufig die Erweiterung des API Projektes nötig. Verschiedene Eintragungen die für das Erweitern des Uniplug Projektes vorgenommen wurden, werden im nächsten Abschnitt genauer erläutert. Hierzu gehören einfache Inklusionen von Source Files, aber auch komplexe overrides von bereits bestehenden C++ Funktionen.
Das hier abgebildete Schaubild zeigt den Ablauf des Wrapping-Vorgangs. Hierbei sind folgende Schritte zu erkennen:

- Erstellen des *.i Files und dortiges inkludieren der gewünschten nativen Code Dateien.
- Erweitern von nativen Code Dateien um eigenen Nativen Code (nicht Zwangsweise nötig)
- Überschreiben von nativen Code Dateien durch eigenen Nativen Code (nicht Zwangsweise nötig)
- Kompilieren des SWIG Projektes durch Aufruf des Compilers.
- Verwenden des kompilierten SWIG Codes in eigenen Projekten.

Die grauen Flächen im Diagramm beschreiben Aktionen in welche der Entwickler aktiv eingreifen muss. Das Interface File mit den gewünschten Dateien muss manuell geschrieben werden. Das erweitern oder überschreiben von Code muss ebenfalls manuell erfolgen. Der Aufruf des SWIG Compilers verläuft parallel zum Prozess des Erweiterns weil für eine Funktionsfähige wandlung des Codes keine Erweiterungen nötig sind. Es muss nur im Problemfall eingegriffen werden. Meist zeigt sich aber, dass in besonders komplexen Fällen, wie des wrappings der Cinema 4D API, doch recht häufig eingegriffen werden muss. Das Überschreiben (manuell) von Klassen und Methoden ist ein Optionaler Fall und wurde zum besseren Erkennen blau eingefärbt.

Erweiterung des Uniplug Codes für Plugins vom Typ TagPlugin

Um das Projekt auf den Aktuellen Stand zu bringen und das schreiben des Plugins zu ermöglichen musste zuerst das Uniplug Projekt erweitert werden. Auch hier wurde mit iterativen Methoden gearbeitet. Falls Probleme mit verschiedene gewrappten Cinema 4D API Methoden auftragen, wurde zunächst

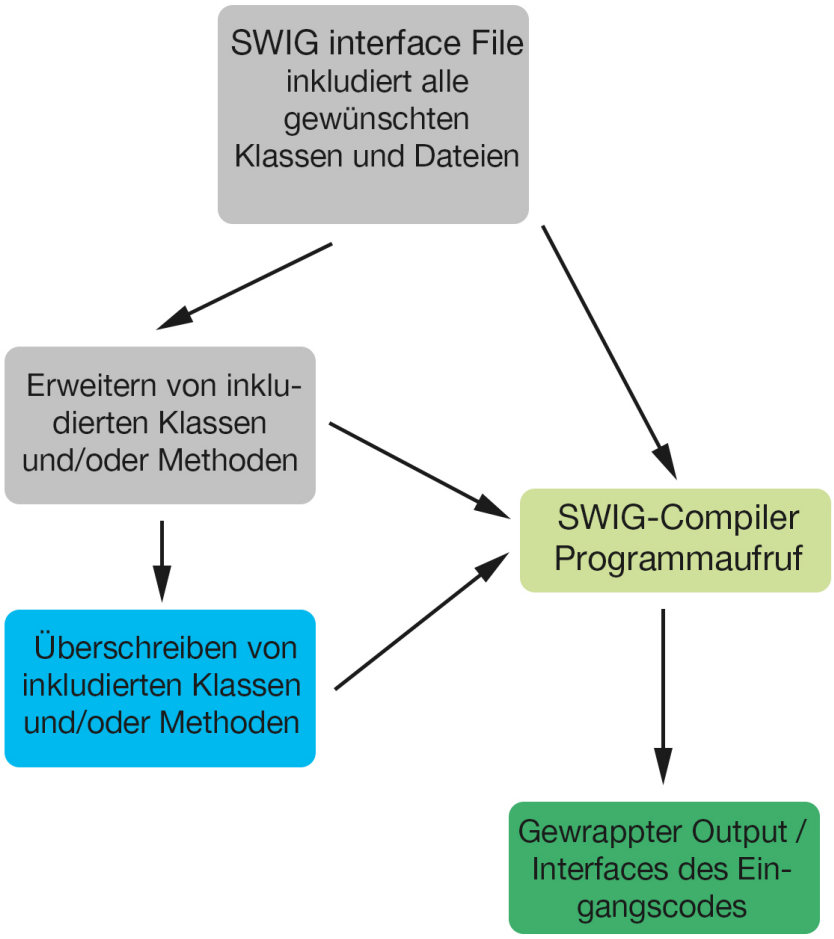


Abbildung 5.4: Wrapping Vorgang der Cinema 4D API von C++ nach C#

geprüft, welche "pseudo" SWIG Dateien diese Probleme verursachten und welche Cinema 4D Api Funktionen aus diesem Grund in der C# API Code nicht vorhanden sind. Diese fehlenden Dateien wurden dann inkludiert und es wurde versucht das SWIG Projekt zu bauen. Sollte es bei diesem Schritt zu Problemen kommen, mussten die API Dateien genauer analysiert werden. In manchen Fällen war es nun nötig den nativen Code der API Dateien zu erweitern oder zu überschreiben. Bei der Implementierung des Cinema 4D Plugin Typs "TagPlugin" waren besondere Änderungen im API Code nötig. Hier sollen die Problematik und die Lösung dieser aufgezeigt und erläutert werden.

Listing 5.2: Einbinden des Maxon Cinema 4D C++ Datentyps TagPlugin in die C# API von Uniplug

```
1 #include "c4d_tagdata.h"
```

```
2  #include "c4d_tagplugin.h"
3
4  // "c4d_tagdata.h"
5  //%include "c4d_tagdata.h";
6  %feature("director") TagDataM;
7  %csmethodmodifiers TagDataM::TagDataM "private";
8  %typemap(cscode) TagDataM %{
9    public TagDataM(bool memOwn) : this(C4dApiPINVOKE.
       new_TagDataM(), memOwn) {
10     SwigDirectorConnect();
11   }
12 %}
13 %include "c4d_tagdata.h";
14 %include "TagDataM.h";
15
16 // "c4d_tagplugin.h"
17 //%include "c4d_tagplugin.h";
18 %feature("director") TagPlugin;
19 %csmethodmodifiers TagPlugin::TagPlugin "private";
20 %typemap(cscode) TagPlugin %{
21   public TagPlugin(bool memOwn) : this(C4dApiPINVOKE.
       new_TagPlugin(), memOwn) {
22     SwigDirectorConnect();
23   }
24 %}
25 %include "c4d_tagplugin.h";
```

Um den Datentyp TagData korrekt im C# Code zu verwenden, waren noch weitere Anpassungen im Code notwendig. Die Methode Message() des TagData ElternTyps NodeData musste überschrieben werden um das Message System in C# nutzen zu können. Dies wurde nötig durch die Verwendung eines Void Pointers seitens Maxons in der Parameterliste der Message Funktion im C++ Code der API. SWIG kann an dieser Stelle durch den nicht auf einen Datentypen festgelegten Void Pointer des Datentyps der Parameterliste keine zufriedenstellende Wrappingfunktion bereit stellen. Das folgende Codebeispiel der Message() Funktion verdeutlicht den Fix im C++ Code des Uniplug Native Projektes.

Die Message() Funktion ist Teil eines Nachrichtensystems innerhalb von Cinema 4D. Viele Elemente des Programs und selbst geschrieben Plugins kommunizieren über dieses Messagesystem miteinander. Hierbei werden Messages

über einen Broadcast versandt. Dieser Broadcast sendet in einer Parameterliste einen ID Code. ID Konstantenvon Cinema 4D werden meist als Konstante (mit define angelegte) Integer Variablen übergeben und können so in if else oder switch case Anweisungen vom Plugin Entwickler identifiziert und verarbeitet werden.

Im folgenden Code Beispiel sind einige dieser Konstanten dargestellt. Bei dem hier in Beispiel 5.3 abgebildeten Code handelt es sich um die überschrieben Message Funktion, welche die Verwendung des Systems für Plugins in C# erst möglich macht. Der void Pointer der Parameterliste wird während des Aufrufs der Funktion in andere Datentypen gecastet. Für die Wandlung der Message Funktion für die TagData Klasse war es von nöten ein Object des Typs DocumentInfoData zu erhalten. Aus diesem Grund, wird das Objekt dieses Typs bei einer bestimmten empfangenen Message ID zurück gegeben.

Listing 5.3: Überschreiben der Message Funktion des TagData Datentyps

```
1  Bool TagDataM::Message(GeListNode *node, Int32 type, void *
       data)
2  {
3  switch (type)
4  {
5  case MSG_EDIT:
6    break;
7  case MSG_GETCUSTOMICON:
8    break;
9  case COLORSYSTEM_HSVTAB:
10   break;
11 case MSG_DOCUMENTINFO:
12   {
13     DocumentInfoData* did = (DocumentInfoData*)data;
14     return MessageDocumentInfo(node, did);
15   }
16   break;
17 case MSG_DESCRIPTION_GETINLINEOBJECT:
18   break;
19 case DRAW_PARAMETER_OGL_PRIMITIVERESTARTINDEX:
20   break;
21   }
22 return true;
23 }
```

Durch die eingeschränkte Möglichkeit nur bis zum call des Debugging API Codes debuggen zu können (Hierzu mehr im Kapitel Probleme und Herausforderungen), war die Lösung des Problems nicht wie meistens in mit SWIG gewrappten Codebasen durch einen einfachen %typemap[8] sondern nur durch die nachträgliche Implementierung der oben dargestellten switch case Anweisung innerhalb des Uniplug Projektes zu bewerkstelligen.

5.2 Der Fusee Szenengraph

Dieser Abschnitt geht auf die Funktionsweise des Szenengraphen in Fusee ein und zeigt seine Ähnlichkeiten und Unterschiede zum Szenenaufbau einer Cinema 4D Szene auf. Da der Fusee Szenengraph zum Zeitpunkt gerade komplett überarbeitet wird, orientiert sich dieser Absatz bereits am neuen Design des Moduls. Der Fusee Szenengraph bietet essentielle Werkzeuge für die Implementierung von Visitor Patterns und ist gleichzeitig auch das Zielformat der Konvertierung von Cinema 4D Szenen nach Fusee. Der Fusee Szenen Graph wird durch einen SceneContainer, wie im Quellcode-Beispiel 5.4 gezeigt, umgesetzt.

Listing 5.4: Anlegen eines Fusee SceneContainers und damit Verwendung des Szenengraphen.

```
1 sceneContainer = new SceneContainer();
2 sceneContainer.Header = new SceneHeader();
3 sceneContainer.Children = new List<SceneNodeContainer>();
```

Der Fusee Szenengraph ermöglich das Traversieren und Vorhalten verschiedener Elemente einer Fusee Szene. Der Graph ist aus einer Kombination von Nodes und Components aufgebaut. Er ähnelt stark dem Unity3D Ansatz des Szenengraphen. Eine Node ist ein Knotenpunkt ohne spezielle Daten. Das Code Beispiel 5.5 zeigt: Die Node enthält lediglich einen Namen, eine Liste an Components und falls nötig, eine Liste an Kind-Nodes.

Listing 5.5: Anlegen eines Fusee Node Containers und initialisieren einer Component Liste.

```
1 SceneNodeContainer node1 = new SceneNodeContainer();
2 node1.Name = "firstObjectNode";
3 node1.Components = new List<SceneComponentContainer>();
```

[8] Ermöglicht das mappen eines Datentyps der Eingangssprache auf einen anderen Datentyp der Ausgangssprache

Das Traversieren wird durch Visitor Implementierungen ermöglicht. Diese Visitor Patterns wurden während dieser Arbeit noch durch das Fusee Team implementiert und erweitert. Ein Visitor ermöglicht das Traversieren einer Szene und seiner Nodes nach einem vom Entwickler definierten Algorithmus. So ist es möglich z.B. Transformationen durchführen, welche die Struktur einer Szene verändern.

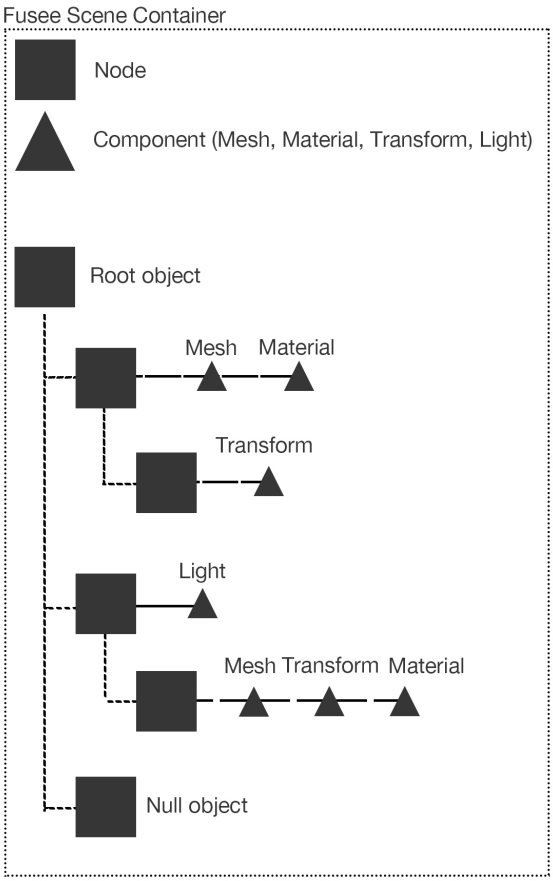


Abbildung 5.5: Der Fusee Szenengraph in exemplarischer bildlicher Darstellung.

Die Abbildung 5.5 zeigt einen Beispielhaft dargestellten “SceneContainer” einer Fusee Szene. Diese Szene besteht aus mehreren Nodes deren Components aus verschiedensten Typen bestehen. So kann ein Node als reiner Anker einer Szene dienen. In der Abbildung 5.5 dargestellt durch das “Root object”. Ein von einem Artist erstelltes Modell kann als Node Object mit angehängtem Mesh und Material Component umgesetzt werden. Sollte sich ein Mesh zusätzlich im Raum bewegen, kann es über eine weitere Transform Komponente verfügen. Sowohl Components als auch Visitor Pattern lassen sich vom Entwickler nach den eigenen Wünschen nach-implementieren oder anpassen. Das anlegen einer Component erfolgt gleich wie das Anlegen eines Fusee SceneNodeContainers. Zu sehen ist dies in Beispiel 5.6. Ein komplett kommentiertes

Source Code Beispiel zur Anwendung des Fusee Szenengraphen findet sich im Anhang in Listing 6.1.

Listing 5.6: Erstellen einer Component aus einem Mesh Objekt und Anhängen an einen Fusee SceneNodeContainer.

```
1 MeshComponent mesh1 = new MeshComponent();
2 LoadMesh(@"Assets/Cube.obj.model", out mesh1);
3 [...]
4 node1.Components.Add(mesh1);
```

Unterschiede des Fusee Szenengraphen zum Cinema 4D Szenengraphen

Der Fusee- und der Cinema 4D Szenengraph unterscheiden sich in gewissen konzeptionellen Ansätzen. Um also den C4D Szenengraph in einen Fusee Szenengraphen umzuwandeln, bedarf es einer Analyse der Unterschiede. Wie bereits erwähnt, verfügt der Fusee Szenengraph über Nodes welche bis die Liste von Childnodes und der Liste an Components keine weiteren relevanten Daten speichern. Der Cinema 4D Graph ist an dieser Stelle um einiges komplexer. Seine Nodes bestehen bereits aus Instanzen von Datentypen die sich vom Standard Cinema 4D "BaseObject" Datentypen ableiten lassen. An ihnen sind bereits Nutzdaten gespeichert. Zu den möglichen Datentypen zählen unter anderem:

- Geometrische Objekte (PointObject) davon abgeleitet wiederum:
  - Polygon Objekte (PolygonObject)
  - Spline Objekte (SplineObject)
  - Linien Objekte (LineObject)
- Partikel Effekte (ParticleObject)
- Kamera Objekte (CameraObject)
- und weitere hier nicht erwähnte Typen.

Diese Objekte können wieder Kind-Objekte der gleichen Datentypen beinhalten. Diese Hierarchie zeigt die Abbildung 5.6. Die in der Abbildung 5.6 am rechten Rand durch kleine Abbildungen neben den Nodes dargestellten Tags, beinhalten weitere Daten oder Attribute und Optionen für das jeweils zugehörige Node. Diese können in einem anderen Fenster (Attribute Fenster/Panel) in Cinema 4d bearbeitet werden.

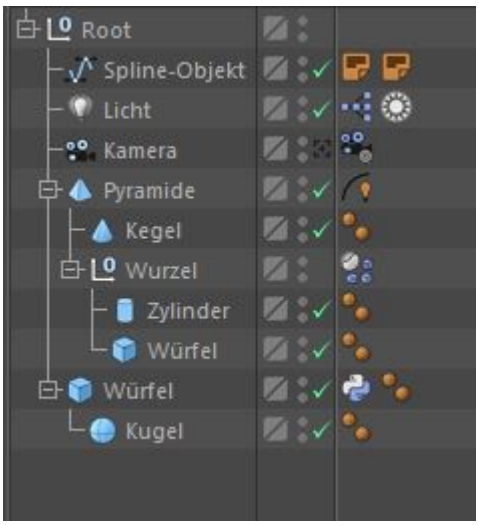


Abbildung 5.6: Der Cinema 4D Szenengraph. Verschiedenste Objekte als "Nodes", daneben als Icons exemplarisch Tags verschiedener Tag-Datentypen am rechten Rand der Grafik.

Der Export einer Cinema 4D Szene in den Fusee Szenengraphen

Die folgende Vorgehensweise beschreibt nun einen möglichen Ansatz wie das neue Fusee Szenengraphen System für einen Export von Cinema 4D Daten genutzt werden kann. Die verschiedenen Datentypen der beiden Programme werden gegenüber gestellt und einander zugeordnet.
Die folgende Tabelle stellt die verschiedenen Typen von Fusee und Cinema 4D gegenüber.

Szenengraph: Cinema 4D und Fusee Datentypen zugeordnet.

Datentypen Fusee	Datentypen Cinema 4D
MeshComponent	Geometrie Polygon Objekt
Material Component	Material
Light Component	Light
Transform Component	Koordinaten des Geometrie Objekts

Tabelle 5.1: Diese Tabelle stellt die Fusee Component Datentypen den Cinema 4D Datentypen des Szenengraphen gegenüber.

Wie ersichtlich ist, lässt sich jeder aktuell benötigte C4D Datentyp einem Typen aus dem Fusee Code zuordnen. Hier soll nun kurz auf den Prozess der Konvertierung eingegangen werden.
Jegliche zu konvertierende Objekte einer Cinema 4D Szene können auf Fusee Seite zuerst einmal durch eine neue Node repräsentiert werden. Der Übersichtlichkeit halber kann sogar der Name des Objektes in der Cinema 4D Szene an das Fusee Node übertragen werden. Das C4D Geometrie Objekt kann ganz simpel mit dem Fusee Exporter in das .fus Format konvertiert werden. So kann

nachfolgend im Converter der Mesh Datensatz des Geometrie Objektes in ein neues Mesh Component kopiert werden. Das Material Component wird ebenfalls über den fus Exporter übertragen und eine Textur kann dann dem aktiven Node als Material Component zugewiesen werden. Die Transform Komponente wird an jedes neue Fusee Node Objekt übergeben und enthält die direkten Positionsdaten eines Cinema 4D Geometrie Objektes.

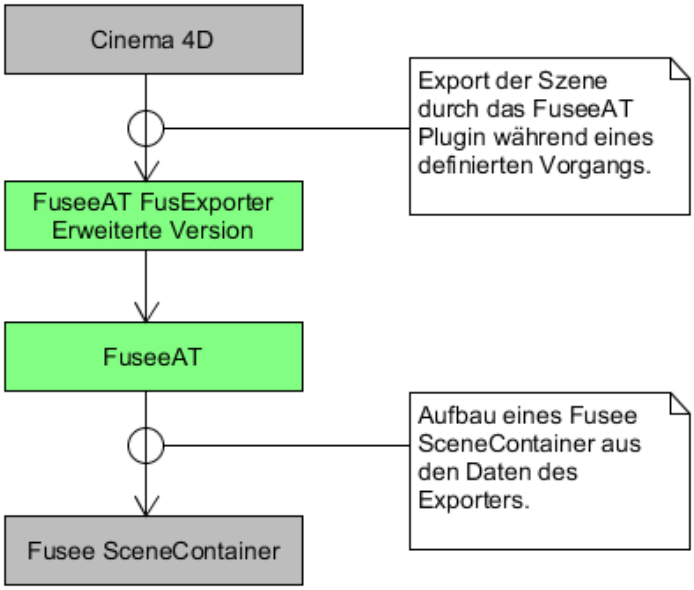


Abbildung 5.7: Export der Cinema 4D Szene. Als Ergebnis entsteht ein Fusee Szenengraph.

Die hier in Abbildung 5.7 angemerkte Erweiterte Version des FusExportes soll nicht einfach nur den Export nach .fus unterstützen sondern auch eine Schnittstelle bieten um verschiedene Daten wie z.B. die Material und Mesh Daten eines Objektes direkt abzugreifen. So können diese durch das FuseeAT in einen Fusee SceneNodeContainer umgesetzt werden. Der hier beschrieben definierte Vorgang des Exports beschreibt in etwa das Speichern einer Szene. Während des Speichervorgangs einer in C4D bearbeiteten Szene mit FuseeAT Funktionalität könnte der Export angestoßen werden. Somit könnte ein solcher Zeitintensiver Vorgang im Hintergrund stattfinden, während der Benutzer des Tools weiter arbeiten kann.

5.3 Stand der Implementierung

Der folgende Abschnitt der Arbeit gibt nun einen Überblick über die bereits erfolgte Implementierung und zeigt welche Aufgaben sich mit dem zum Zeitpunkt der Arbeit bereits implementierten Teil der Software umsetzen lassen

und welche Bereiche noch weiter entwickelt werden können und müssen. Weiterhin wird auf einige Probleme während der Umsetzung hingewiesen die im Rahmen dieser Arbeit nicht gelöst werden konnten.

5.3.1 Bereits umgesetzte Module des Konzepts

Während der Implementierungsphase wurde versucht mit Hilfe der Cinema 4D API und des Uniplug Projekts eine Softwarebibliothek zu implementieren welche Basisfunktionalität für die Verwendung von Cinema 4D als Authoring Tool bieten kann.
Folgende Funktionalität wurde bereits umgesetzt:

- Ein C# Projekt kann in einem Binary Fusee Engine Projekt angelegt werden
- Ein Projekt kann mit der Hilfe eines Plugins in Cinema 4D geöffnet werden
- Eine neue C# Klasse kann mit der Bibliothek angelegt und in das Projekt integriert werden
- Ein Tag kann im Cinema 4D Szenengraphen erzeugt und an ein Objekt der Szene geheftet werden
- Verschiedene Module des Cinema 4D Plugins können untereinander über das Cinema 4D Message System Informationen austauschen

Auf die genauere Umsetzung und einzelne interessante Aspekte der Implementierung wird in den nächsten Abschnitten weiter eingegangen.

Implementierung: Generieren eines Fusee Projektes

Um ein Projekt für die Fusee Engine zu generieren, wird die folgende Funktion 5.7 des FuseeAT eingesetzt.

Listing 5.7: Funktion zum Generieren eines neuen Projekts in der Binary Version der Fusee Engine.

```
1  public bool CreateProject(String slnName, String pName,
     String pPath);
```

Das Projekt kann durch die Angabe des Namens des Fusee Binary Solution Files, des vom Benutzer wählbaren Projektnamens und eines Pfades zur Solution erzeugt werden. Diese Funktion ist unabhängig von Cinema 4D implementiert.

Alle Funktionen des FuseeAT sind nicht von einem bestimmten Modeling oder Level Editor abhängig und können somit wiederverwendet werden sollte das FuseeAT auf einen anderen Editor angepasst werden.

Implementierung: Öffnen eines Fusee Projektes in Cinema 4D

Ein bereits angelegtes Fusee Projekt kann durch den Plugin Code auch in Cinema 4D geöffnet werden. Die hierzu nötige Funktion 5.8 ist die folgende:

Listing 5.8: Funktion zum öffnen eines neuen Projekts in der Binary Version der Fusee Engine.

```
1  public ToolState OpenProject(String pName, String pPath);
```

Das gewünschte Projekt wird, falls existent, direkt aus einer XML Datei deserialisiert. Diese XML Datei enthält alle wichtigen Informationen über das Projekt. Hierzu gehören verschiedene Dateinamen als auch verschiedene Pfade welche beim ersten Erstellen eines Projektes generiert oder durch den Benutzer über Dialoge und Abfragen eingegeben werden. Die im Codebeispiel 5.9 dargestellten Attribute unterstützen das XML System beim serialisieren und deserialisieren der Objekte.

Listing 5.9: Teil des Structs zum Serialisieren des Fusee Projektes aus Cinem a 4D. Die Serialisierung speichert das Projekt auf der Festplatte als lesbare XML Datei.

```
1  [XmlElement("PathToSolutionFolder")]
2  public String PathToSolutionFolder;
3
4  [XmlElement("PathToProjectFolder")]
5  public String PathToProjectFolder;
```

Das Projekt wird nach der Deserialisierung im Speicher gehalten. So kann die Applikation, in diesem Fall das C4DPlugin dauerhaft auf alle wichtigen Informationen für die Verwaltung des Projektes zur Laufzeit zugreifen.

Implementierung: Erzeugen einer neuen C# Klasse und Einfügen in das Projekt

Das Erzeugen einer neuen Klasse kann über die Funktion 5.10 erreicht werden. Die erzeugte Klasse wird vom FuseeAT in das C# Projekt eingefügt. Der Nutzer muss nur den Namen der neuen C# Klasse übergeben. Die Klasse wird

nach einem im FuseeAT abgelegten TEmplate erzeugt und enthält bis auf die Standardstrukturen einer C# Klasse (Klassendefinition und Konstruktor) keine weiteren Bestandteile. Das Template kann aber einfach in FuseeAT um gewünschte Settings erweitert werden.

Listing 5.10: Funktion zum Erstellen einer neuen Klasse im C# Projekt.

```
1   public bool CreateNewClass(String pName);
```

Implementierung: Erstellen eines Tags und Anfügen von Code an ein Asset

Durch die Cinema 4D API und das wrappen der dort vorhandenen Funktionen nach C# ist es möglich in der Cinema 4D Applikation an Objekte ein Tag anzuhängen. Ein Tag ist ein Objekt im Speicher welches in Referenz mit dem anhängenden Objekt steht und dieses um weitere Funktionalität oder Optionen erweitert. Es ist möglich dem Tag verschiedene Funktionen zuzuweisen. Leider muss die Funktionalität des Tags sehr stark an Cinema 4D gekuppelt werden da seine gesamte Implementierung über die C4D API erfolgt. Um Informationen an einem Tag zu speichern, wird das gleiche XML Serialisierungs und Deserialisierungs System eingesetzt welches auch beim Laden und Speichern eines Engine Projektes verwendet wird.
Das hier angeführte Quellcodebeispiel stellt ein C# Struct dar. Das Element ModelEditorName enthält den Namen des Objektes im C4D Szenengraphen. Es ist angedacht diesen im Exporter als Namen eines Fusee NodeContainers zu verwenden. Das Element ModelEditorID beschreibt eine in Cinema 4D generierte Hash Id welche genau einem spezifischen Objekt der Szene zugeordnet werden kann. Durch das Speichern dieses Hashes kann beim Öffnen eines Projektes kann für jedes Objekt wieder ein Bezug zur Cinema 4D Szene hergestellt werden. Das element ClassName beschreibt schliesslich eine an das Objekt angehängte C# Code Datei.

Listing 5.11: Asset Informationsspeicher Objekt. Kann zu XML serialisiert werden.

```
1  public struct AssetObject
2  {
3          [XmlElement("ModelEditorName")]
4          public String objectName;
5
```

```
6           [XmlElement("ModelEditorID")]
7           public String objectID;
8
9           [XmlElement("ClassName")]
10          public String className;
11  }
```

Das ToolState System des FuseeAT

Das FuseeAT ToolState System ist ein Konzept welches bei der Implementierung des Frameworks umgesetzt wurde. Das System besteht auf einem einfachen State Machine Konzept und erlaubt es dem Entwickler zur Identifizierung von Schwierigkeiten einige Enum Variablen als Returnwerte der Funktionen einzusetzen. Durch die Verwendung des Enums 5.12 können Probleme während der Laufzeit im FuseeAT System aufgedeckt werden. Das Konzept sieht vor, als return Werte der FuseeAT Funktionen jeweils einen Status des ToolState Enums zurückzugeben. So kann der Entwickler eines Plugins oder Editors auf diese Funktionen eingehen und vermeidet es so mit einem korrumpierten Projektdateien weiter zu arbeiten.

Listing 5.12: ToolState System: Unterstützt die Stabilität des FuseeAT bei der Zusammenarbeit mit Plugins.

```
1  public enum ToolState
2  {
3    OK = 0,
4    ERROR = 1,
5    WARNING = 2,
6  }
```

Der ProjectState in FuseeAT

Durch die simple Implementierung des Enums ProjectState 5.13 und dessen Verwendung während der Laufzeit des FuseeAT kann der Status des Projektes jederzeit überprüft werden. Ein Entwickler, welcher das FuseeAT erweitern oder verwenden möchte, kann jederzeit auf den Status des Projektes zugreifen und seine Funktionen mit absichern. Durch dieses recht einfache Konzept, können Probleme in den Solution und Projektdateien des FuseeAT erkannt werden. Es ist aber unabdingbar, dass fehleranfällige Operationen und Code

diese Funktion auch einbinden und abfragen.

Listing 5.13: Code des ProjectState Enums. Wird in FuseeAT verwendet um die Integrität eines Projekts zu repräsentieren.

```
1 public enum ProjectState
2 {
3   Clean = 0, // means open, too
4         Dirty = 1,
5         Corrupt = 2,
6         Closed = 3
7 }
```

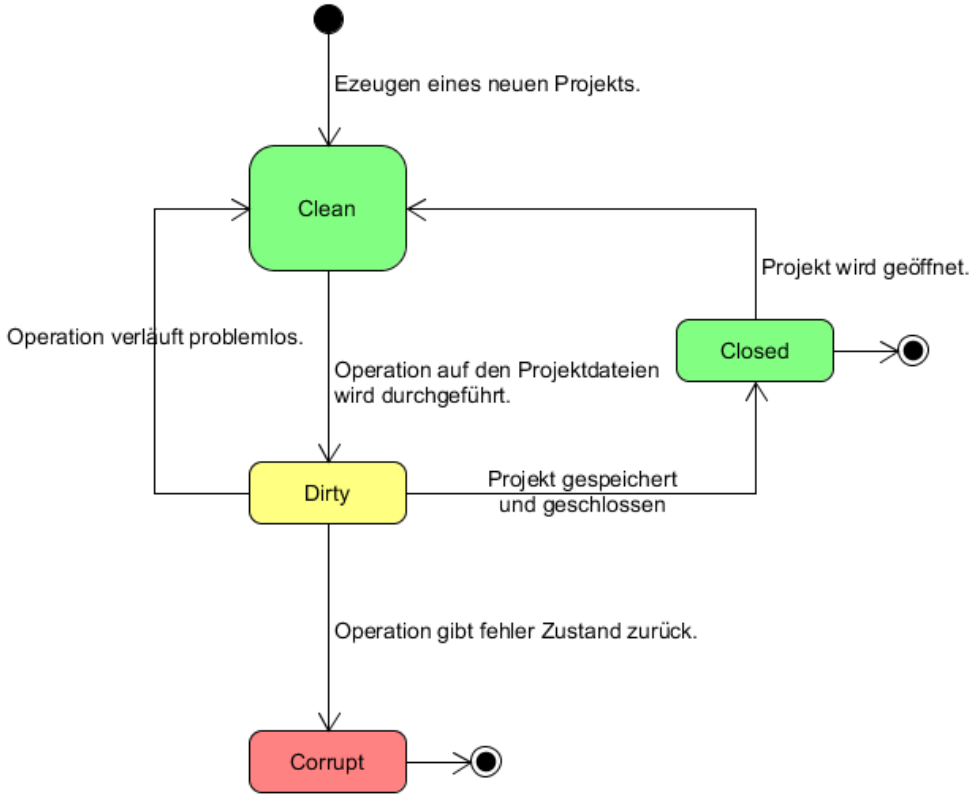


Abbildung 5.8: Zeigt das State Diagram zum Projektstatus eines Fusee Engine Projektes.

In der Abbildung 5.8 wird die Funktion des Zustands Handlings grafisch dargestellt. Eine Operation bezieht sich hier auf das Durchführen von Änderungen an den Projektdateien (z.B. Visual Studio Solution, C# Projekt Dateien). Der Corrupt Status stellt hier einen Endstatus des Zustandsdiagramms dar. Dieser Status kann nicht mehr verlassen werden und bedingt unbedingtes manuelles eingreifen eines Entwicklers.

5.3.2 Problematische Aspekte während der Implementierung

Problematisch war, dass die Uniplug Softwarebibliothek zum Zeitpunkt der Implementierung nicht die nötigen Funktionen der Cinema 4D API unterstützte. Zudem wurde die Cinema 4D API Bibliothek und deren Integration in die Cinema 4D R16 Applikation während der Entwicklung durch ein Update seitens Maxon stark verändert und Uniplug musste daraufhin erneut auf die API angepasst werden. Um die Entwicklung des Plugins für die Verwendung der FuseeAT bibliothek zu beginnen, wurden verschiedenste Funktionen der Cinema 4D API in das Uniplug Projekt integriert. Die Phase dieser Integration konnte aufgrund der Komplexität des Cinema 4D API nicht als abgeschlossen betrachtet werden. Während der Entwicklung des Plugins und des FuseeAT musste immer weitere Cinema 4D API Bestandteile nach C# gewrapped werden um die Funktionalität zwischen FuseeAT und dem Cinema 4D Plugin herzustellen. Da Uniplug ein essentielles Projekt für die Implementierung des Konzeptes darstellt, war es unerlässlich zuerst die fehlende Funktionalität in Uniplug zu ergänzen was einen großen Anteil an der Entwicklungszeit beanspruchte.

Die Komplexität der Cinema 4D C++API und dessen Transformation nach C# verzögerte die Entwicklung des Authoring Tool Frameworks und des Plugins deutlich, so dass während der Arbeit nicht alle geplanten Funktionen implementiert werden konnten. Hierzu zählt vor allem die GUI repräsentation der einzelnen Funktionen als auch das Build System.Weiterhin existieren aktuell nur schwer zu lösende Probleme welche durch die Verwendung von SWIG in Verbindung mit der komplexen API von C4d auftraten. Die Funktionalität der User Interface Erweiterungen in Cinema 4D ist nur eingeschränkt vorhanden. Es ist aktuell nicht möglich, Cinema 4D GUI Fenster mit UI Elementen wie z.B. Buttons und Textfeldern zu erweitern. Es wurde versucht das System zu debuggen um den Grund für diese Problematik zu beheben. Allerdings konnte aufgrund des verschleierten Cinema 4D API Codes nur auf ein gewisses Level debugged werden. So ergibt sich beim Hinzufügen einer UI Komponentenen ein Fehler welcher nur bis zum Aufruf des Cinema 4D API Codes verfolgt werden konnte. Dort endet die Möglichkeit des Debuggings. Diese Komponente returned dann nur eine Boolean Variable mit der Belegung false. Aufgrund der zeitlichen beschränkung dieser Arbeit wurde dieses Modul ersteinmal hinten angestellt und die Entwicklung auf andere Teile des Konzepts konzentriert.

Weiterhin verkompliziert sich die Entwicklung durch das wrappen der C++Api nach C# stark. Einfache Funktionen der Cinema 4D API können meist nur kompliziert nach C# übersetzt werden. Hier zeigt sich ein Nachteil des Uniplug Projektes. Es ist zwar einfach, während der Entwicklung in C# gegebene

Funktionen von Uniplug zu verwenden, allerdings ist die Erweiterung von Uniplug fast immer mit Problemen und großem Zeitaufwand verbunden. Somit erschwerte die immer wieder nötige Rückkehr nach Uniplug während der Implementierung das FuseeAT den effektiven Fortschritt massiv.

6 Ergebnisse der Arbeit und Ausblick

6.1 Wie weit ist das Projekt fortgeschritten?

Die Konzeption eines Authoring-Tools und dieAnalyse eines Arbeitsprozesses für das Entwickeln von Authoring Tools in internen Teams wurde abgeschlossen. Der Arbeitsprozess beschreibt eine unbürokratische Möglichkeit zur Entwicklung von Tools und betrachtet hierbei sowohl Aspekte des Projektmanagements einer Tool Entwicklung als auch technische Aspekte. Der Prozess wurde grundsätzlich auf kleine interne Teams ausgerichtet und versucht das Projektmanagement mit so wenig Management Overhead[1] wie möglich umzusetzen. Weiterhin wurden zwei weitere Game Engines und deren Workflows und Ansätze untersucht. Die Szenengraphen der jeweiligen Engines wurden so weit es möglich war analysiert und gewonnene Erkenntnisse in ein Konzept für die Fusee Engine und das FuseeAT übersetzt. Das FuseeAT wurde während des Systemdesigns unabhängig einer speziellen Oberfläche entwickelt. Das Konzept für das FuseeAT wurde anschließend in der Implementierungsphase teilweise umgesetzt um ein Framework mit Schnittstellen nach Außen zu schaffen. Eine grundsätzliche Funktionalität ist gegeben und kann nun in verschiedene Editoren wie z.B. Cinema 4D integriert werden. Das Cinema 4D Plugin wurde in seinen Grundzügen erstellt und kann jederzeit erweitert werden. Das Uniplug Projekt umfasst noch nicht alle nötigen API Funktionen aber unterstützt aktuell die nötige Funktionalität um das Plugin in einer kontrollierten Umgebung zu testen.

6.2 Integration des Systems in den weiteren Projektverlauf von Fusee

Die im Abschnitt “Problematische Aspekte während der Implementierung” aufgeführten Probleme beeinflussten die Entwicklung des Cinema 4D Plugins

[1] Zusätzlicher Aufwand durch Managementaufgaben welcher die Umsetzung des Projekts verzögert.

stark negativ. Aus diesem Grund ist die Implementierung des FuseeAT nicht komplett abgeschlossen. Das Fusee Projekt an sich ist entsprechend seiner Natur als in der Lehre eingesetzte Simulations- und Entertainment Engine ein Projekt welches sich in ständiger Entwicklung befindet. Somit kann in Zukunft das FuseeAT für verschiedene Editoren und Toolsals Ausgangsprojekt genutzt werden um eine grafische Oberfläche für das Produzieren von Applikationen mit der Fusee Engine zu schaffen.

6.3 Fazit

Aufgrund der gewonnenen Erkenntnisse lässt sich sagen, dass die Authoring Tool Entwicklung auch in Zukunft ein wichtiger Bestandteil der Spiele und Software Entwicklung bleiben wird und seinen Einfluss auf das erfolgreiche Abschliessen von Projekten sicherlich noch weiter vergrößert. Verschiedenste angeführte Quellen belegen, dass dieser Bereich der Projekte immer weiter strukturiert und ausgebaut wird. Hierbei sei noch einmal auf Sonys Studio SnSystems[2] verwiesen welches sich rein um diese Aufgabe kümmert. Es ist aber auch für kleine Teams möglich während der Entwicklung Lücken im Tool Lineup des Unternehmens zu füllen und mit Hilfe von Frameworks und effizienter Planung schnell neue Tools zu erschaffen. Um die Tool Entwicklung noch effizienter schneller zu gestalten wurde im Konzept zu FuseeAT besonders darauf geachtet, das FuseeAT Framework ungebunden von Editor Software zu gestalten. FuseeAT kann durch seinen Aufbau als modulares Framework schnell und unkompliziert erweitert werden. Die Schnittstelle zu FuseeAT steht als Interface bereit. Cinema 4D wiederum konnte aufgrund der gesetzten Ziele (Entwicklung des Frameworks in C#, Anbindung an Cinema 4D mit dem Uniplug Projekt) nicht als effizientes Werkzeug für die Tool Entwicklung identifiziert werden. Die Idee, ein Authoring Tool so zu gestalten, dass jedes Mitglied des Teams seine gewohnte Arbeitsumgebung während der Entwicklung mit dem Tool nicht verlassen muss, scheiterte hier an der komplexen API und dem “Wrapping” der Cinema 4D API von C++nach C# . Das Konzept an sich scheint allerdings aufgrund der in der Untersuchung von Unity3d und der Unreal Engine 4 erkannten ähnlichen Konzepte einer 3DModelling Software und eines Game Engine Editors vielversprechend. Hier ergab sich ein größtenteils Deckungsgleiches Bild der Softwarearchitektur. Sicherlich kann die FuseeAT durch einen höheren Zeitaufwand und ein größeres Entwicklerteam in die Cinema 4D Software integriert werden. Letztendlich wurden viele Grundsteine für die Entwicklung von und mit FuseeAT gelegt und ein kompletter Ent-

[2] http://snsystems.com/

wicklungsprozess für die Tool Entwicklung aufgezeigt. Eine Erweiterung der Implementierung findet in Zukunft sicherlich statt. Das Projekt wurde in die Fusee Engine integriert und kann als Open Source Software jederzeit eingesehen, angepasst und verändert werden.

Anhang

Literaturverzeichnis

[Flo14] Stefan, Göbel Ralf, Steinmetz Florian, Mehm Christian,Reuter. »Future Trens in Game Authoring Tools«. In: (2014). S. 536-541.

[Kli15] Feldbacher, Christian Klinge, Heiko, Chefredakteur. »User-Generated Content in the V-Play Game Engine«. In: *Making Games* 3 (2015), S. 24–27.

[Lan15] Klose, Jan Lange, Thorsten. »Lords of the Fallen Tackling a new gen game with an emerging studio«. In: *Making Games* 3 (2015), S. 44–45.

[Ger09] Jason Gergory. *Game Engine Architecture*. Taylor & Francis Ltd., 2009. ISBN: 978-1568814131.

[Car04] Ben Carter. *The Game Asset Pipeline*. Delmar Cengage Learning, 2004. ISBN: 1-58450-342-4.

[Wih06] Graham Wihlidal. *Game Engine Toolset Development*. Thomson Course Technologies, 2006. ISBN: 1-59200-963-8.

[Cha06] Heather Chandler. *The Game Production Handbook*. Thomson Delmar Learning, 2006. ISBN: 1-58450-416-1.

[Mül14] Gärtner, Fabian Müller, Christoph. »Student Project Portable Real-Time 3D Engine«. In: *EUROGRAPHICS2014* (2014). Education Paper. URL: http://fusee3d.org/wp-content/uploads/2014/04/FUSEE_Educational.pdf (geprüft am 07. 04. 2015).

[Bet03] Erik Bethke. *Game Development and Production*. Wordware Publishing Inc.; Auflage: Pap/Cdr (Februar 2003), 2003. ISBN: 978-1556229510.

[Fre14] Will Freeman. *Kingdom Come: Deliverance Video Update 9*. Available online http://www.develop-online.net/tools-and-tech/inside-the-system-sony-s-internal-console-tools-team/0191319. 2014. (Geprüft am 07. 04. 2015).

[Waw14] Alex Wawro. *Sony releases level editor that's open source and engine-agnostic*. Available online http://www.gamasutra.com/view/news/224682/Sony_releases_level_editor_thats_open_source_and_engineagnostic.php. 2014. (Geprüft am 07. 04. 2015).

[Cif05] Frank Cifaldi. *E3 Report: The Path to Creating AAA Games*. Available online http://www.gamasutra.com/view/feature/130722/e3_report_the_path_to_creating_.php. 2005. (Geprüft am 08. 04. 2015).

[Sch14] Christopher Schmitz. »Projektmanagement mit Scrum«. In: (2014). Available online http://www.makinggames.biz/features/projektmanagement-mit-scrum,2534.html. (Geprüft am 07. 04. 2015).

[Hir86] Ikujiro, Nonaka Hirotaka, Takeuchi. *The new product development game*. Available online https://hbr.org/1986/01/the-new-new-product-development-game. 1986. (Geprüft am 15. 04. 2015).

[CCP09] CCP Games. *Eve Online Fanfest 2009 - Scrum and Agile development processes*. Available online https://www.youtube.com/watch?v=GqsReCZD4hc. 2009. (Geprüft am 07. 04. 2015).

[Sch13] Christopher Schmitz. *The art of waiting*. Available online http://www.warhorsestudios.cz/index.php?page=blog&entry=blog_031. 2013. (Geprüft am 07. 04. 2015).

[Mar14] Daniel Vavra Martin Klekner Votja Nedved. *Kingdom Come: Deliverance Video Update 9*. Available online https://www.youtube.com/watch?v=LfklQR-36-8. 2014. (Geprüft am 07. 04. 2015).

[Sch04] Ken Schwaber. *Agile Project Management with Scrum*. Micorosoft Press, 2004. ISBN: 073561993x.

[Pro11] Project Management Institute. »A Guide to the Project Management Body of Knowledge«. In: *IEEE Std 1490-2011* (Nov. 2011).

[84] »IEEE Guide for Software Requirements Specifications«. In: *IEEE Std 830-1984* (Feb. 1984), S. 1–26.

[Bea14] Fulton, William Beazley, David. *Swig Introduction Documentation*. Available online http://www.swig.org/exec.html. 2014. (Geprüft am 15. 04. 2015).

Software und Applikationsverzeichnis

[Dec14] Deck13. *Lords of the Fallen*. http://www.deck13.de/. 2014. (Geprüft am 18. 05. 2015).

[EPI15] EPIC GAMES, INC. *Unreal Engine 4*. http://www.unrealengine.com. 2004-2015. (Geprüft am 18. 05. 2015).

[Uni15] Unity Technologies. *Unity 5*. http://www.unity3d.com. 2015. (Geprüft am 16. 05. 2015).

[MAX15] MAXON Computer GmbH. *Cinema 4D R16*. http://www.maxon.net/de/products/cinema-4d-studio.html. 2014-2015. (Geprüft am 18. 05. 2015).

[Nau11] NaughtyDog. *Uncharted*. http://www.naughtydog.com/games/uncharted_drakes_fortune/. 2007, 2009, 2011. (Geprüft am 20. 05. 2015).

[Gue13] Guerrilla Games. *Killzone 1, Killzone 2, Killzone 3, Killzone: Mercenary, Killzone: Shadow Fall*. http://www.guerrilla-games.com/. 2004, 2007, 2011, 2013, 2013. (Geprüft am 12. 05. 2015).

[Qua13] Quantic Dream. *Beyond: Two Souls*. http://www.quanticdream.com/en/. 2013. (Geprüft am 15. 05. 2015).

[Aut15a] Autodesk, Inc. *3DS Max*. http://www.autodesk.de/products/3ds-max/overview. 2015. (Geprüft am 18. 05. 2015).

[The15] The Foundry Visionmongers Ltd. *Modo*. https://www.thefoundry.co.uk/products/modo/. 2015. (Geprüft am 18. 05. 2015).

[Ble15] Blender Foundation. *Blender 2.74*. http://www.blender.org/. 2014-2015. (Geprüft am 18. 05. 2015).

[Aut15b] Autodesk Inc. *Autodesk Scaleform*. http://gameware.autodesk.com/scaleform. 2014-2015. (Geprüft am 18. 05. 2015).

[Dig15] Digital Illusions CE. *Frostbite Engine*. http://www.frostbite.com/. 2015. (Geprüft am 16. 05. 2015).

[Avi09] Avid Technology. *Alienbrain*. http://www.alienbrain.com/. 2009. (Geprüft am 16. 05. 2015).

[Sho13] Shotgun Software Inc. *Shotgun*. https://www.shotgunsoftware.com/. 2013. (Geprüft am 16. 05. 2015).

[Bea15] SWIG.org Beazley, David Fulton, William. *SWIG-3.0*. http://www.swig.org/. 1995-2015. (Geprüft am 18. 05. 2015).

Quellcodeverzeichnis

Listing 6.1: Beispiel zum Fusee Szenengraphen. Erstellt im Rahmen dieser Arbeit, während der Implementierung des Fusee Szenengraphen durch das Fusee Entwicklerteam und Herrn Prof. C. Müller.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using Fusee.Engine;
5  using Fusee.Math;
6  using Fusee.Serialization;
7  using Fusee.Engine.SimpleScene;
8
9  namespace Examples.SimpleSceneContainer
10 {
11     [FuseeApplication(Name = "SimpleSceneGraph Example",
       Description = "A very simple example for the new
       SceneGraph system.")]
12     public class Simple : RenderCanvas
13     {
14         // We only need one ref to a container for the root
       container of the scene.
15         private SceneContainer sceneContainer;
16
17         // We have a scene visitor with which we can travel
       through our scene.
18         private SceneRenderer sceneVisitorRender;
19
20         // is called on startup
21         public override void Init()
22         {
23             RC.ClearColor = new float4(0, 0, 0, 1);
24
25             // Init the SceneContainer.
26             sceneContainer = new SceneContainer();
27             sceneContainer.Header = new SceneHeader();
28             sceneContainer.Children = new List<
       SceneNodeContainer>();
29
30             // Fill the headers information.
31             sceneContainer.Header.CreatedBy = "Fusee Team";
32             sceneContainer.Header.CreationDate = "Year 2015"
       ;
```

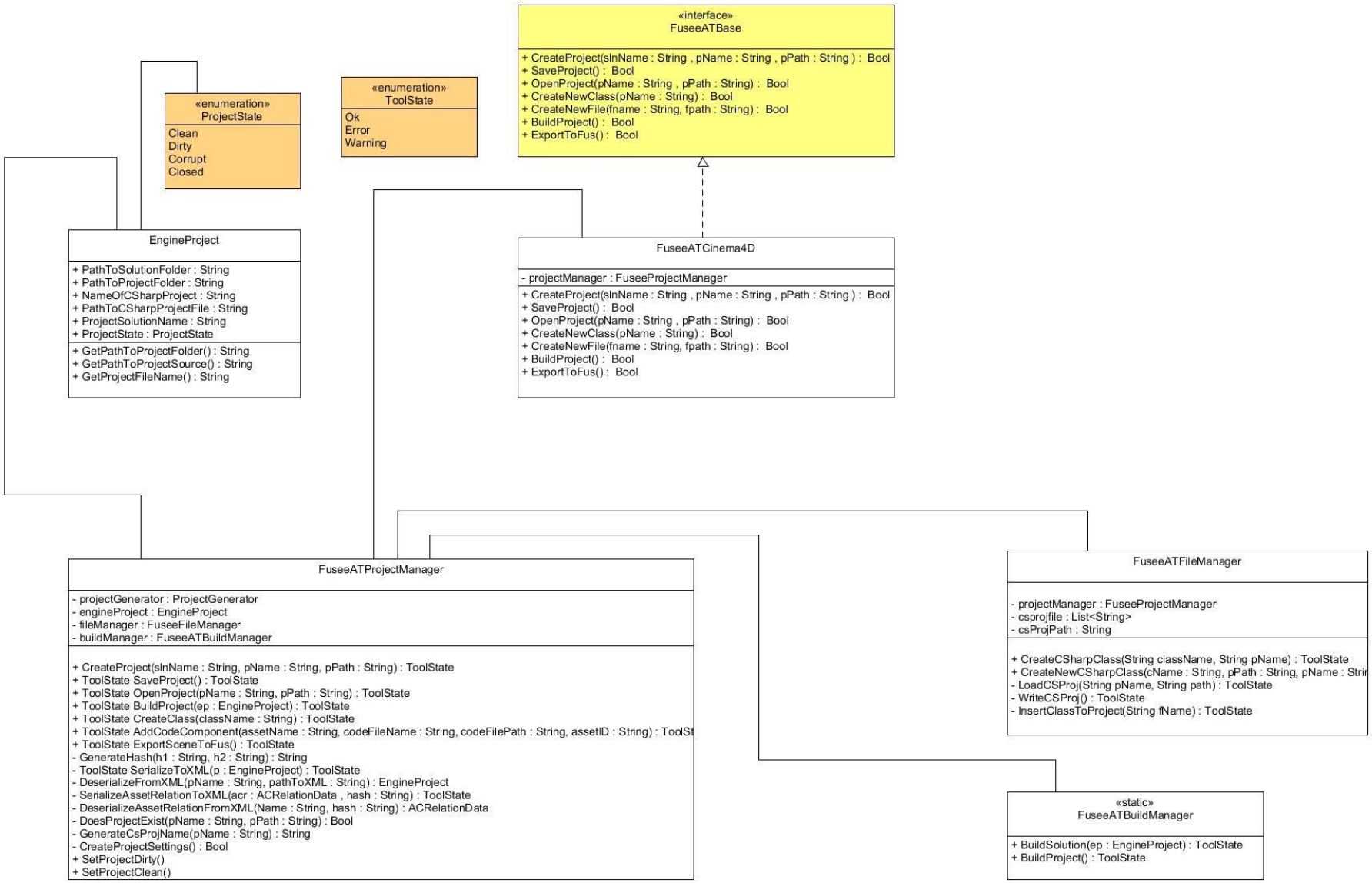
```
33              sceneContainer.Header.Generator = "Some
    Generator";
34              sceneContainer.Header.Version = 1;
35
36              // Create a node as our first object node in the
     scene.
37              SceneNodeContainer node1 = new
    SceneNodeContainer();
38              node1.Name = "firstObjectNode";
39              node1.Components = new List<
    SceneComponentContainer>();
40
41              // Add another node to the scene which is in
    fact empty.
42              SceneNodeContainer node2 = new
    SceneNodeContainer();
43              node2.Name = "secondObjectNode";
44              node2.Components = new List<
    SceneComponentContainer>();
45
46              // Loading a mesh from a file to a MeshComponent
     object.
47              MeshComponent mesh1 = new MeshComponent();
48              LoadMesh(@"Assets/Cube.obj.model", out mesh1);
49
50              // Creating a transform component to manipulate
    transformations.
51              TransformComponent transf1 = new
    TransformComponent();
52              //transf1.Scale = new float3() { x = 1f, y = 1,
    z = 1};
53              //transf1.Translation = new float3() { x = 1f, y
     = 1, z = 1 };
54              //transf1.Rotation = new float3() { x = 1f, y =
    1, z = 1 };
55
56              // Material Component
57              MaterialComponent mat1 = new MaterialComponent()
    ;
58              mat1.Diffuse = new MatChannelContainer();
59              mat1.Diffuse.Color = new float3(1f, 0, 0);
```

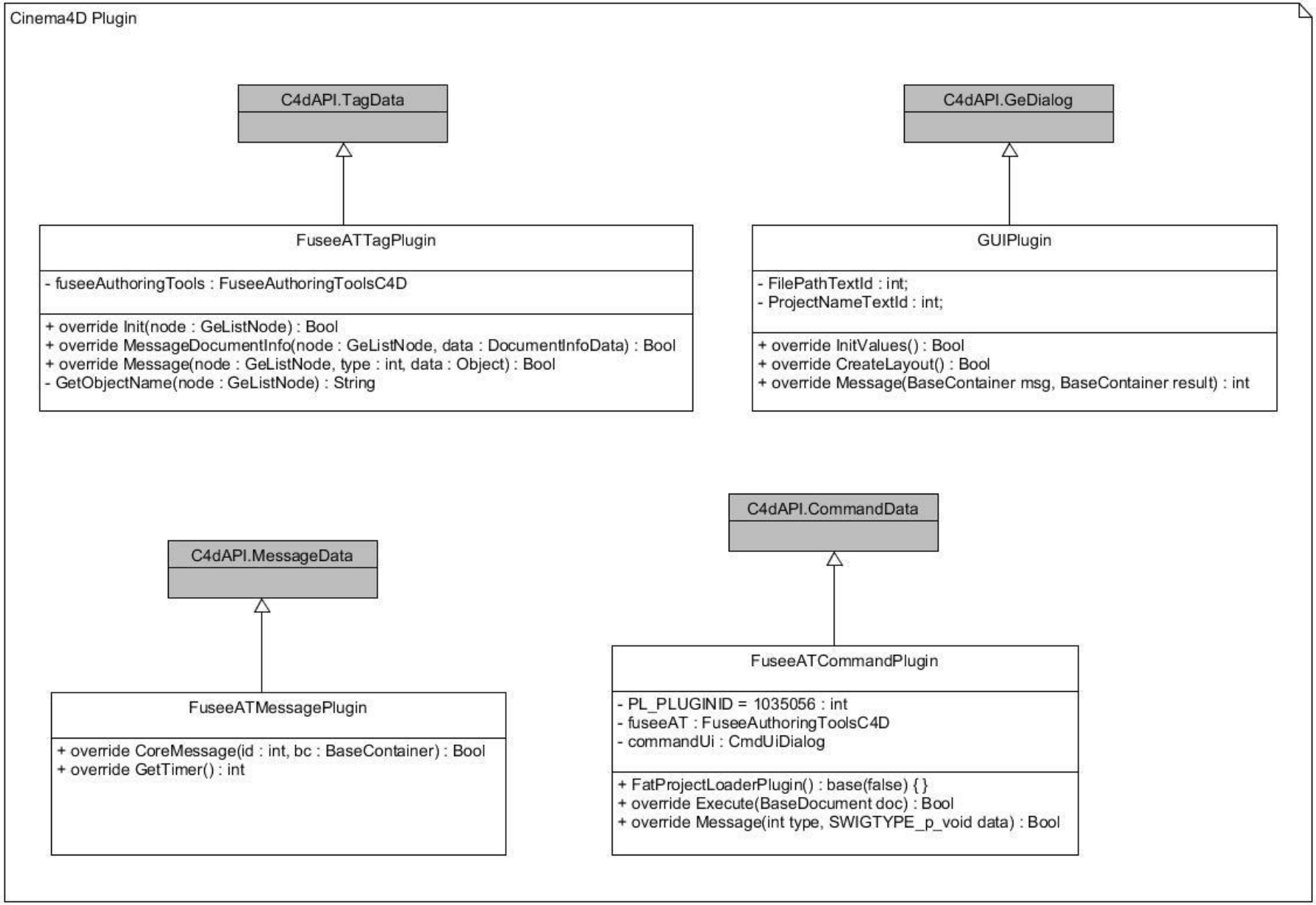
```
60
61              // Add the components to the first node.
62              node1.Components.Add(mesh1);
63              node1.Components.Add(transf1);
64              node1.Components.Add(mat1);
65
66              // Add another node to the node1s children.
67              SceneNodeContainer node1child = new
    SceneNodeContainer();
68              node1child.Name = "node1secondObjectNode";
69              node1child.Components = new List<
    SceneComponentContainer>();
70
71              node1.Children = new List<SceneNodeContainer>();
72              node1.Children.Add(node1child);
73
74              // Add the node with all the components to the
    scene.
75              sceneContainer.Children.Add(node1);
76              sceneContainer.Children.Add(node2);
77
78              // Create the renderer and add the scene
    Container to it. Also append the render context.
79              sceneVisitorRender = new SceneRenderer(
    sceneContainer, @"Assets/Cube.obj.model");
80              sceneVisitorRender.SetContext(RC);
81          }
82
83          // is called once a frame
84          public override void RenderAFrame()
85          {
86              // Clear the buffers
87              RC.Clear(ClearFlags.Color | ClearFlags.Depth);
88
89              // Render all the children in the sceneNode.
90              sceneVisitorRender.Render(RC);
91
92              // swap buffers
93              Present();
94          }
95
```

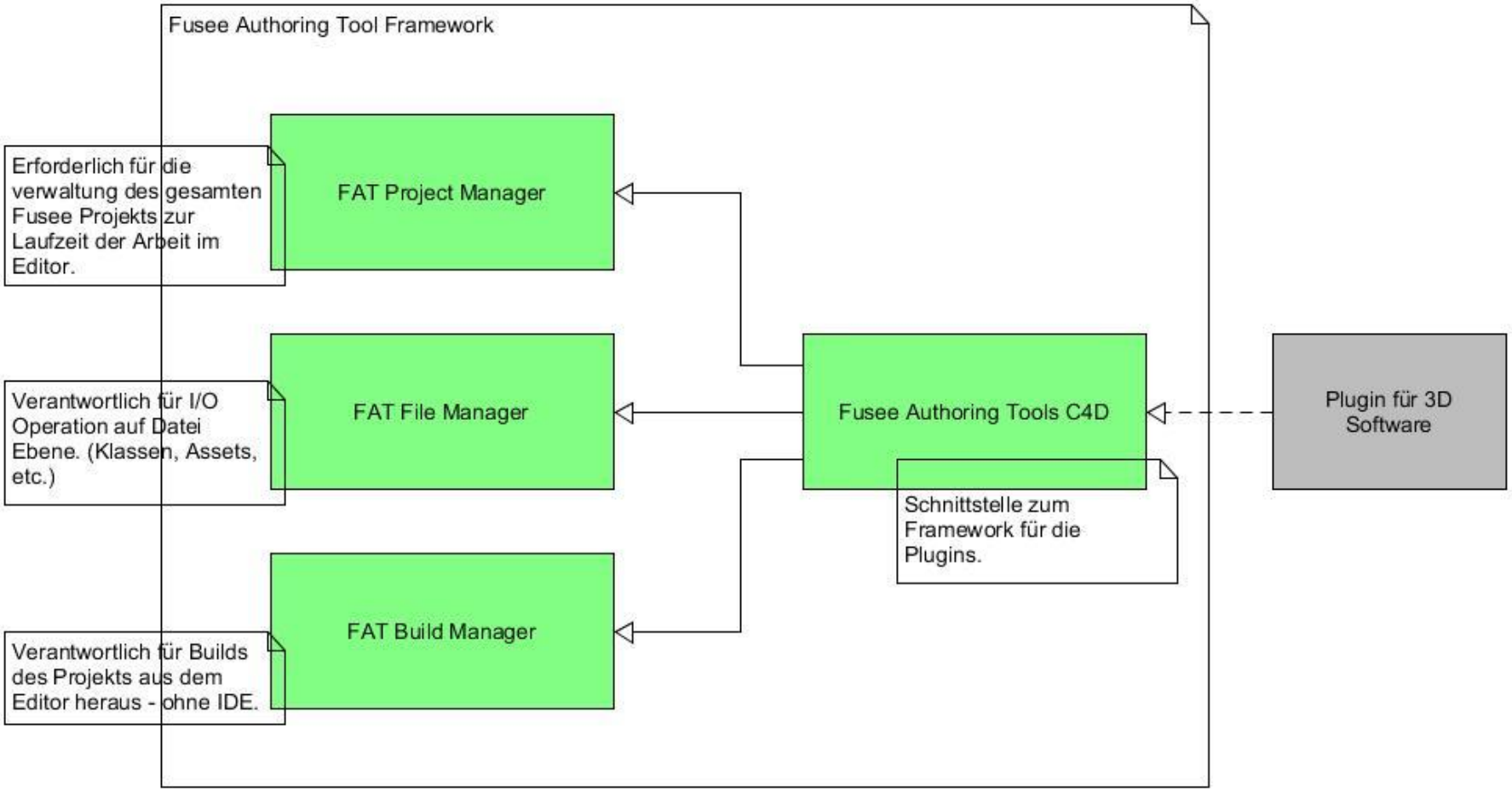
```
96          // is called when the window was resized
97          public override void Resize()
98          {
99              RC.Viewport(0, 0, Width, Height);
100
101             var aspectRatio = Width/(float) Height;
102             var projection = float4x4.
    CreatePerspectiveFieldOfView(MathHelper.PiOver4,
    aspectRatio, 1, 5000);
103             RC.Projection = projection;
104         }
105
106         public static void Main()
107         {
108             var app = new Simple();
109             app.Run();
110         }
111
112         /// <summary>
113         /// Can load a meshs values to a mesh component
    object.
114         /// This method is purely for convencience.
115         /// </summary>
116         /// <param name="path"></param>
117         /// <param name="m1"></param>
118         /// <returns>Mesh</returns>
119         public Mesh LoadMesh(String path, out MeshComponent
    m1)
120         {
121             MeshComponent m = new MeshComponent();
122             Mesh mesh = MeshReader.LoadMesh(path);
123             m.Vertices = mesh.Vertices;
124             m.Triangles = mesh.Triangles;
125             m.Normals = mesh.Normals;
126             m.UVs = mesh.UVs;
127             // TODO: Bounding box.
128             //m1.BoundingBox = mesh;
129
130             m1 = m;
131             return mesh;
132         }
```

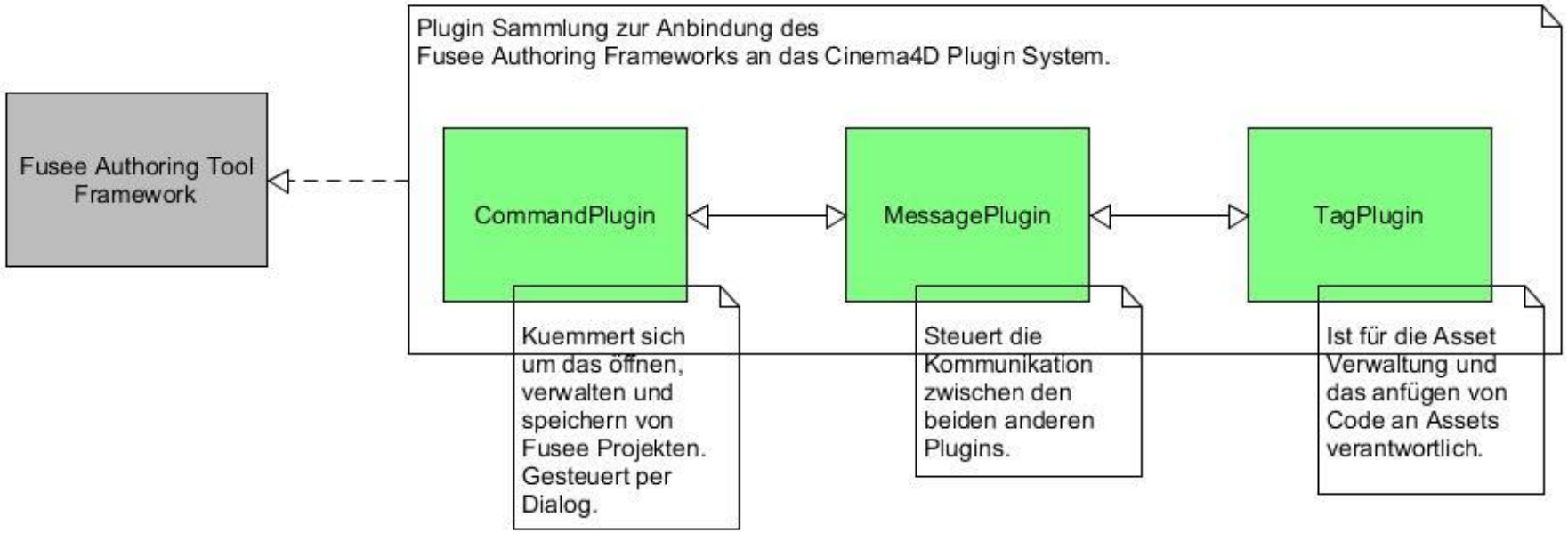
```
133         }
134 }
```

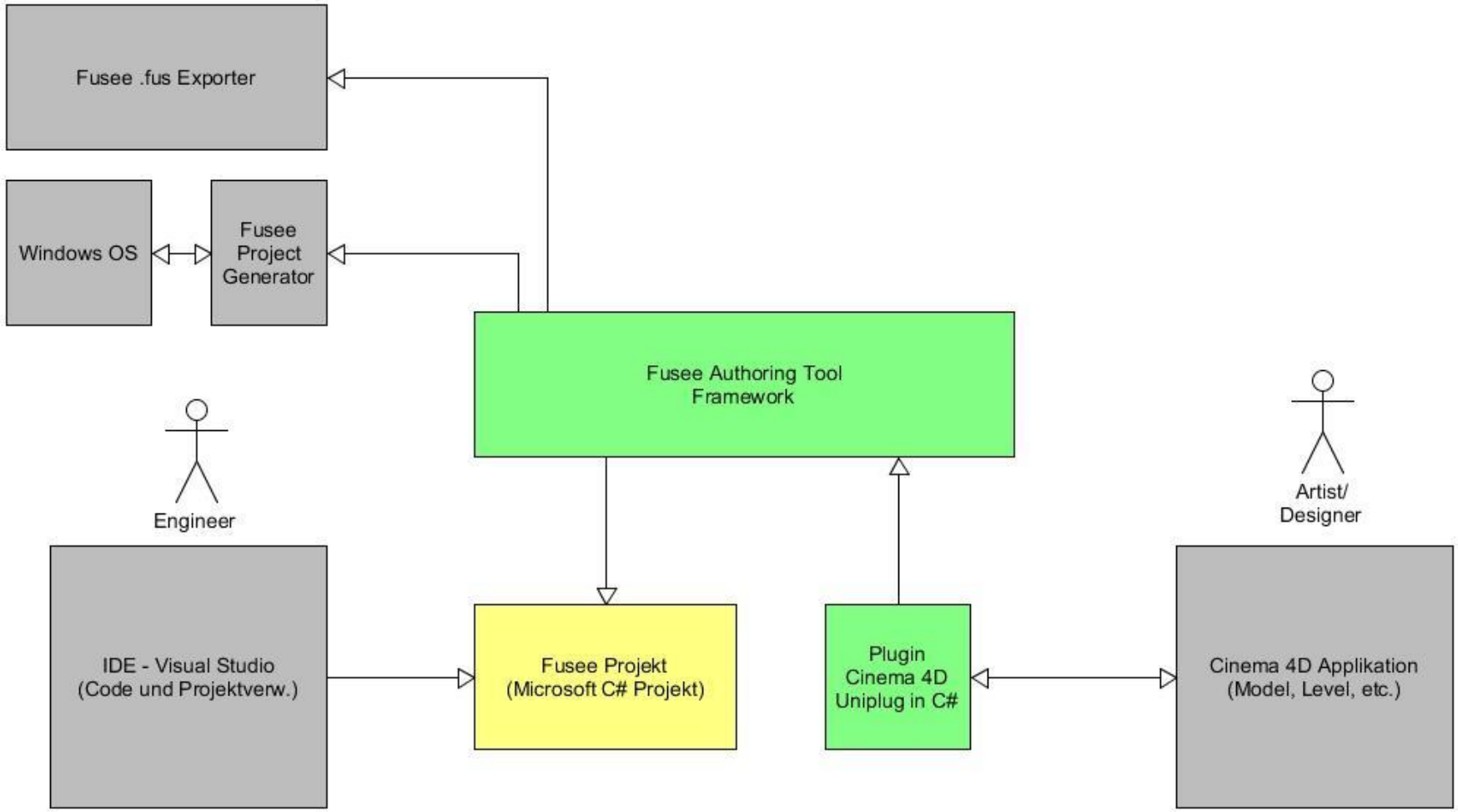
Abbildungsverzeichnis


«interface»
FuseeATBase
+ CreateProject(slnName : String , pName : String , pPath : String) : Bool
+ SaveProject() : Bool
+ OpenProject(pName : String , pPath : String) : Bool
+ CreateNewClass(pName : String) : Bool
+ CreateNewFile(fname : String, fpath : String) : Bool
+ BuildProject() : Bool
+ ExportToFus() : Bool
«enumeration»
ProjectState
Clean
Dirty
Corrupt
Closed
«enumeration»
ToolState
Ok
Error
Warning
EngineProject
+ PathToSolutionFolder : String
+ PathToProjectFolder : String
+ NameOfCSharpProject : String
+ PathToCSharpProjectFile : String
+ ProjectSolutionName : String
+ ProjectState : ProjectState
+ GetPathToProjectFolder() : String
+ GetPathToProjectSource() : String
+ GetProjectFileName() : String
FuseeATCinema4D
- projectManager : FuseeProjectManager
+ CreateProject(slnName : String , pName : String , pPath : String) : Bool
+ SaveProject() : Bool
+ OpenProject(pName : String , pPath : String) : Bool
+ CreateNewClass(pName : String) : Bool
+ CreateNewFile(fname : String, fpath : String) : Bool
+ BuildProject() : Bool
+ ExportToFus() : Bool
FuseeATProjectManager
- projectGenerator : ProjectGenerator
- engineProject : EngineProject
- fileManager : FuseeFileManager
- buildManager : FuseeATBuildManager
+ CreateProject(slnName : String, pName : String, pPath : String) : ToolState
+ ToolState SaveProject() : ToolState
+ ToolState OpenProject(pName : String, pPath : String) : ToolState
+ ToolState BuildProject(ep : EngineProject) : ToolState
+ ToolState CreateClass(className : String) : ToolState
+ ToolState AddCodeComponent(assetName : String, codeFileName : String, codeFilePath : String, assetID : String) : ToolSt
+ ToolState ExportSceneToFus() : ToolState
- GenerateHash(h1 : String, h2 : String) : String
- ToolState SerializeToXML(p : EngineProject) : ToolState
- DeserializeFromXML(pName : String, pathToXML : String) : EngineProject
- SerializeAssetRelationToXML(acr : ACRelationData , hash : String) : ToolState
- DeserializeAssetRelationFromXML(Name : String, hash : String) : ACRelationData
- DoesProjectExist(pName : String, pPath : String) : Bool
- GenerateCsProjName(pName : String) : String
- CreateProjectSettings() : Bool
+ SetProjectDirty()
+ SetProjectClean()
FuseeATFileManager
- projectManager : FuseeProjectManager
- csprojfile : List<String>
- csProjPath : String
+ CreateCSharpClass(String className, String pName) : ToolState
+ CreateNewCSharpClass(cName : String, pPath : String, pName : Strir
- LoadCSProj(String pName, String path) : ToolState
- WriteCSProj() : ToolState
- InsertClassToProject(String fName) : ToolState
«static»
FuseeATBuildManager
+ BuildSolution(ep : EngineProject) : ToolState
+ BuildProject() : ToolState


Cinema4D Plugin
C4dAPI.TagData
C4dAPI.GeDialog
FuseeATTagPlugin
- fuseeAuthoringTools : FuseeAuthoringToolsC4D
+ override Init(node : GeListNode) : Bool
+ override MessageDocumentInfo(node : GeListNode, data : DocumentInfoData) : Bool
+ override Message(node : GeListNode, type : int, data : Object) : Bool
- GetObjectName(node : GeListNode) : String
GUIPlugin
- FilePathTextId : int;
- ProjectNameTextId : int;
+ override InitValues() : Bool
+ override CreateLayout() : Bool
+ override Message(BaseContainer msg, BaseContainer result) : int
C4dAPI.CommandData
C4dAPI.MessageData
FuseeATMessagePlugin
+ override CoreMessage(id : int, bc : BaseContainer) : Bool
+ override GetTimer() : int
FuseeATCommandPlugin
- PL_PLUGINID = 1035056 : int
- fuseeAT : FuseeAuthoringToolsC4D
- commandUi : CmdUiDialog
+ FatProjectLoaderPlugin() : base(false) { }
+ override Execute(BaseDocument doc) : Bool
+ override Message(int type, SWIGTYPE_p_void data) : Bool


Fusee Authoring Tool Framework
Erforderlich für die verwaltung des gesamten Fusee Projekts zur Laufzeit der Arbeit im Editor.
FAT Project Manager
Verantwortlich für I/O Operation auf Datei Ebene. (Klassen, Assets, etc.)
FAT File Manager
Verantwortlich für Builds des Projekts aus dem Editor heraus - ohne IDE.
FAT Build Manager
Fusee Authoring Tools C4D
Schnittstelle zum Framework für die Plugins.
Plugin für 3D Software


Plugin Sammlung zur Anbindung des
Fusee Authoring Frameworks an das Cinema4D Plugin System.
Fusee Authoring Tool Framework
CommandPlugin
MessagePlugin
TagPlugin
Kuemmert sich um das öffnen, verwalten und speichern von Fusee Projekten. Gesteuert per Dialog.
Steuert die Kommunikation zwischen den beiden anderen Plugins.
Ist für die Asset Verwaltung und das anfügen von Code an Assets verantwortlich.


Fusee .fus Exporter
Windows OS
Fusee
Project
Generator
Fusee Authoring Tool
Framework
Engineer
Artist/
Designer
IDE - Visual Studio
(Code und Projektverw.)
Fusee Projekt
(Microsoft C# Projekt)
Plugin
Cinema 4D
Uniplug in C#
Cinema 4D Applikation
(Model, Level, etc.)