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 Cinema 4D Plugins ist es möglich, dass Designer oder Entwickler jederzeit mit ihren eigenen Tools in die Entwicklung eines Projektes einsteigen. Das während der Arbeit entstandene Plugin bietet grundlegenden Funktionen um an einem Projekt mit der Fusee Engine zu arbeiten. Ein Fusee Projekt "managedßich durch die Nutzung des entstandenen Cinema 4D Plugins und den generierten Visual Studio Solution Dateien aus Sicht des Plugin Nutzers selbst. Das zuerst konzeptionell entworfene Tool wurde während dieser Arbeit als Prototyp umgesetzt und bietet ausreichende Funktionalität um ein Projekt als Entwickler als auch als Artist zu erstellen und zu bearbeiten. Hierzu wurden verschiedene Konzepte betrachtet und andere GameEngines auf Workflow und Anwendbarkeit untersucht. 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.

# 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 11. 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 und Ziele** **1**

1.1 Motivation . . . . . 1

1.2 Fragestellung der Arbeit . . . . . 2

1.3 Ziele der Implementierung . . . . . 4

1.4 Verwendete Software . . . . . 4

**2 Tool Development in internen Teams** **5**

2.1 Definition: Tools und Toolsets . . . . . 5

2.2 Internes Tool Developing oder Tool Licencing . . . . . 5

2.3 Asset Pipelines . . . . . 6

2.4 Die Rollenverteilung interner Tool Developer . . . . . 7

**3 Produktionsprozess: Projektplanung und Analyse** **9**

3.1 Entwicklungsprozesse in Interaktiver 3D Software und Games . 9

3.1.1 Projektmanagement Modelle . . . . . 9

3.1.2 Agiles Modell oder klassisches Modell . . . . . 10

3.1.3 Scrum . . . . . 10

3.2 Mitglieder eines Entwicklerteams . . . . . 12

3.2.1 Producer . . . . . 13

3.2.2 Artists . . . . . 13

3.2.3 Designer . . . . . 14

3.2.4 Engineer . . . . . 16

3.3 Stakeholderanalyse intern . . . . . 17

3.3.1 Prozess der Stakeholderanalyse . . . . . 19

3.4 Der Arbeitsprozess zur Authoring Tool Entwicklung . . . . . 22

3.4.1 Game Authoring / Game Development . . . . . 22

3.4.2 Tool Development und Asset Pipelines . . . . . 22

3.4.3 Tool Development - der organisatorische Ablauf . . . . . 23

**4 Entwicklung eines Konzeptes** **27**

4.1 Use Cases der verschiedenen Entwickler . . . . . 27

4.1.1 Was möchten Artists? . . . 27
4.1.2 Was möchte ein Engineer? . . . 29
4.2 Aktuelle Engines und deren Arbeitsprozesse . . . 31
4.2.1 Prozesse in Game Engines und/oder Frameworks . . . 31
4.2.2 Unreal Engine 4 . . . 31
4.2.3 Unity 3D . . . 37
4.3 Systemdesign . . . 42
4.3.1 Warum Fusee und Cinema 4D? . . . 42
4.3.2 Systemdesign für das Toolkit . . . 42
4.3.3 Systemdesign für ein Tool Framework . . . 44
4.3.4 Systemdesign für ein Plugin System . . . 46
4.3.5 Zeitersparnis durch bekannte Tools . . . 47
4.4 Asset Management und Asset Pipeline in der Fusee Engine . . . 49
4.4.1 Asset Pipelines in Fusee Authoring Toolkit . . . 50
4.4.2 Assetmanagement in Fusee Authoring Toolkit . . . 50
4.5 Die Cinema 4D C++ API und ihre Verwendung in diesem Projekt 51
4.5.1 In der Implementierung verwendete Softwareprojekte / Ausgangssituation . . . 52
4.5.2 Cinema 4D Plugin API und SDK . . . 52
4.5.3 Uniplug . . . 55
4.5.4 Aus Fusee verwendete Software-Module . . . 62
4.6 Fortschritt der Implementierung . . . 64
4.6.1 Bereits umgesetzte Teile des Konzepts . . . 64
4.6.2 Implementierung: Generieren eines Fusee Projektes . . . 65
4.6.3 Implementierung: Öffnen eines Fusee Projektes in Cinema 4D . . . 66
4.6.4 Implementierung: Erzeugen einer neuen C# Klasse und Einfügen in das Projekt . . . 66
4.6.5 Implementierung: Erstellen eines Tags und Anfügen von Code an ein Asset . . . 67
4.6.6 Das ToolState System des FuseeAT . . . 67
4.6.7 Der ProjectState in FuseeAT . . . 68
4.6.8 Problematische Aspekte während der Umsetzung des Konzepts . . . 68

**5 Ergebnisse der Arbeit und Ausblick 70**
5.1 Wie weit ist das Projekt Fortgeschritten? . . . 70
5.2 Integration des Systems in den weiteren Projektverlauf von Fusee 70
5.2.1 Ergebnisse der Arbeit . . . 71

**Literaturverzeichnis** **73**

**Source Code Verzeichnis und Beispiele** **75**

**Abbildungsverzeichnis** **81**

**UML Diagramme** **82**

**Requirements Dokumente** **88**

# 1 Einführung und Ziele

Arbeitsprozesse in heutigen Game Engines verlangen von Entwicklern meist das erlernen neuer Toolsets und dies während eines Zeitlich knapp bemessenen Projekt Zeitraums. Es wäre für Entwickler einfacher sich mit den bereits bekannten Tools zu beschäftigen und mit solchen großartige Ergebnisse zu erreichen. Authoring Tools ermöglichen also auch Teammitgliedern ohne weiteres tiefegend technisches Verständnis für die Programmierung von Spielen, an Projekten in der Entwicklung mitzuarbeiten. Dieser Ansicht ist auch F. Mehm. Sein Team veröffentlichte einen Überblick (Florian, Mehm, S. R., Christian,Reuter. (2014). Future trens in game authoring tools. S. 536-541) über Game Authoring Tools in der Gegenwart und in der Zukunft.

> „It [Developing "Authoring Tools", Anmerkung des Autors] is successful due to several factors: it allows non-technical users to work on projects that would otherwise be out of their reach (due to lack of expertise, especially concerning programming languages); it can bring struc-ture into unstructured domains (such as game development) and it can speed up de-velopment by streamlining and automating common tasks." Florian, Mehm, 2014

## 1.1 Motivation

Authoring Tools sind ein wichtiger Bestandteil professioneller und semiprofessioneller Entwickler Teams von interaktiver Software. Mit einer gesteigerten Produktivität durch einfach verständliche aber mächte Tools kann ein Projekt in kürzeren Zeiträumen umgesetzt werden. Authoring Tools verbreiten sich selbst auf dem Consumer Markt. Sei es hier durch so genannte Mod Kits (Tools die Entwickler zur Erweiterung durch Fans für ihre veröffentlichten Spiele bereitstellen) oder durch Game Engines die auf User Generated Content setzten wie die V-Play Game Engine siehe [Klinge, Heiko, Chefredakteur, F. (2015). User-Generated Content in the V-Play Game Engine. *Making Games*, *3*, 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. (2014). Lords of the Fallen. http://www.deck13.de/. City Interactive 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.

> "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."
>
> Lange, Thorsten, K. (2015). Lords of the Fallen Tackling a new gen game with an emerging studio. *Making Games*, *3*, 44–45

Diese Entwicklungen bieten einen interessanten Ansatz und zeigen, dass Tool Development in der Industrie durch immer komplexere Softwareprojekte einen hohen Stellenwert erreicht hat. Ben Carter schreibt in Carter, B. (2004). *The Game Asset Pipeline*. Delmar Cengage Learning, dass Entwickler so lange es geht in einem Tool arbeiten können sollten ohne dass sie das Tool dauernd wechseln müssten um Teilaufgaben zu erledigen.

> "The users of the tool [...] should spend as much of their time as possible in that tool. Every time theay have to switch to another application to perform some task [...] they are losing time and potentially, breaking their concentration. Carter, 2004, S. 18"

## 1.2 Fragestellung der Arbeit

Aus den oben angeführten Gründen beschäftigt sich diese Arbeit mit dem Gebiet des Authoring Tool Development und mit der Frage ob es möglich ist ein Tool zu konzipieren welches auf der Basis eines bereits bestehenden Modeling

Editors (hier Cinema 4D von Maxon[1]) das Erstellen einer "fertigen"[2] Szene für die 3D Engine Fusee[3] ermöglicht. Dazu wird der gesamte Weg der Entstehung eines Authoring Tools Betrachtet. Verschiedene Projektmanagement Modelle zur internen Software Entwicklung werden geprüft und ein rudimentäres Requirements Engineering wird durchgeführt und somit ein Softwarekonzept entwickelt. Nach der Konzeption wird versucht die Basis Funktionalität in Visual Studio mit Hilfe von C# Code und der nach C# gewrappten[4] 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 angedachte Implementierung genutzt und bietet einen geringen Umfang an Basisfunktionalität. So bietet es die Möglichkeit grundsätzlich Plugins für Cinema 4D in der Programmiersprache C# zu schreiben. Von Haus aus ermöglicht Maxon das Entwickeln von Plugins nur in C++, Python und Coffee (einer von Maxon selbst entwickelten Skriptsprache). Der Vollständigkeit halber sei gesagt, dass Maxon für C++ noch das Framework Melange anbietet welches es ermöglicht Cinema 4D Dateien ohne eine Cinema 4D installation zu erstellen, zu speichern und zu laden. Sollte eine Installation vorhanden sein kann ein Melange Plugin auch Szenen rendern. vgl. Maxon Cinema 4D Developer Dokumentation unter https://developers.maxon.net/.

Szenen in Cinema 4D werden grundsätzlich in einer Art Baumstruktur gespeichert und zur weiteren Verarbeitung im Speicher gehalten. Die hier konzipierte Software möchte diese Tatsache nutzen um Szenen aus einem Modeling Editor (Cinema 4D) in eine Szene des Fusee Szenen Formats (.fus) umzuwandeln. Eine ".fus" Datei ist ebenfalls in einer Baumartigen Struktur gespeichert. Dieses Prinzip der Szenendarstellung ist bereits aus verschiedenen Frameworks und Softwareprojekten für 2D Darstellung bekannt. Das ist zum einen der Übersichtlichkeit als auch verschiedenen Algorithmischen Operationen auf den Daten der Szene geschuldet. In Baumstrukturen organisierte Interfaces werden außerdem bei der Entwicklung des User Interfaces für das Mobile Betriebssystem Android verwendet.

[1] MAXON Computer GmbH. (2014-2015). Cinema 4D R16. http://www.maxon.net/de/products/cinema-4d-studio.html.
[2] Build fähige Version einer im Fusee Szenenformat abgespeicherten 3D Szene.
[3] Fusee (Furtwangen Simulationd and Entertainment Engine - http://www.Fusee3d.org)
[4] Eine Software welche von einem anderen Stück software umgeben wird.

## 1.3 Ziele der Implementierung

Die während dieser Arbeit implementierte Software hat das Ziel eine Basis für die Verwendung von Cinema 4D als Game Engine Editor aufzubauen. Es werden grundlegende Funktionen in Form einer C# Bibliothek entwickelt die es ermöglichen sollen das Projekt in Zukungt auch für andere 3D Modeling Software anzupassen. Diese Arbeit zielt nicht darauf ab ein komplettes Tool für die Entwicklung von Spielen in der Fusee Engine zu erschaffen. Es wird versucht eine art Grundstein für weitere Forschung und Entwicklung in den Bereich des Game Authoring Toolkit Developments für die Arbeit mit der Akademischen Simulations und Entertainment Software Fusee zu legen. Das Kernziel ist das Erstellen eines Konzeptes und die Erläuterung der einzelnen Module eines solchen Systems. Verschiedene bereits bestehende Tools und Game Engines werden zu vergleichen herangezogen und wurden im Laufe dieser Arbeit untersucht und verglichen.

## 1.4 Verwendete Software

- 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/.

# 2 Tool Development in internen Teams

## 2.1 Definition: Tools und Toolsets

Grob aufteteilt, können die Werkzeuge eines Entwickler Teams 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 die Produktions Pipeline des Produktes darstellen. Die Komplexität von Tools reicht von einfach Text Editoren bis hin zu Modeling und Level Design Software. Die Komplexität des Tools korreliert meist mit der Komplexität der zu Lösenden Problematik. Tools können oft für eine spätere Aufgabe wiederverwendet werden. So genannte Wegwerf-Tools wurden meist speziell für eine Aufgabe angefertigt ohne dabei die Wiederverwendbarkeit im Auge zu behalten. Vgl. (Wihlidal, 2006, 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 (Bethke, 2003) berichtet in davon, dass Michael Abrash[1] ihm einst mitteilte, "dass 50% der Entwickler Arbeit bei idSoftware in das Tool Development fliesse." vgl. Bethke, E. (2003). *Game Development and Production*. Wordware Publishing Inc.; Auflage: Pap/Cdr (Februar 2003), 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 Studios erstellen lassen, welches bereits während der Planung und Entwicklung der

[1] Ehemals idSoftware, ehemals Valve VR, aktuell Chief Scientist bei Oculus https://www.oculus.com/company/
[2] Playstation 4 - Erschienen im Herbst 2013
[3] Freeman, W. (2014). Kingdom come: deliverance video update 9.

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 ein Anlass das das praktische Projekt zu dieser Arbeit an zu implementieren.

## 2.3 Asset Pipelines

Um von Artists produzierte Assets in das Spiel zu bringen bedarf es meist einer so genannten Asset Pipeline. Bei dieser Asset Pipeline handelt es sich um eine angepasst Zusammenstellung mehrerer (auch selbst erstellter) Entwicklertools. Das Ziel der Asset Pipeline beschreibt (Carter, 2004) mit:

> "Quite simply, the term describes the sequence of processes that

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

[5] Wawro, A. (2014). Sony releases level editor that's open source and engine-agnostic.

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

[7] NaughtyDog. (2007, 2009, 2011). Uncharted. http://www.naughtydog.com/games/uncharted_drakes_fortune/. Sony Computer Entertainment.

[8] Guerrilla Games. (2004, 2007, 2011, 2013, 2013). Killzone 1, Killzone 2, Killzone 3, Killzone: Mercenary, Killzone: Shadow Fall. http://www.guerrilla-games.com/. Sony Computer Entertainment.

[9] Quantic Dream. (2013). Beyond: Two Souls. http://www.quanticdream.com/en/. Sony Computer Entertainment.

> 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 [...]. Carter, B. (2004). *The Game Asset Pipeline*. Delmar Cengage Learning, S. 6

Somit ist die Assetpipeline ein wichtiges Instrument und besteht meist aus einer Zusammenstellung von eigens Entwickelten Tools und lizensierter Software wie z.B. Game Engines, Modeling Editoren wie 3DS Max, Cinema 4D, oder Maya und weiterer Software. Diese Arbeit beschäftigt sich also mit einem Teilgebiet der Asset Pipeline. In einem späteren Kapitel wird das Fusee Asset Management betrachtet und die Asset Pipeline für das Entwickeln einer Fusee Anwendung mit Cinema4D konzeptionell untersucht.

## 2.4 Die Rollenverteilung interner Tool Developer

Wihlidahl (Wihlidal, 2006, S. 5) beschreibt verschiedene Organisationsmodelle 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

Diese Arbeit legt sich für den Aspekt des Software-Designs und der Organisatorischen Planung auf die Variante "Dedicated Tools Team" fest. (Wihlidal, 2006) beschreibt dieses Variante wie folgt:

> This model is based around a team that takes a tool from inception all the way to supporting it. This model works extremely well, though it generally requires a liason with both technical and esign skills to help faciliate effective communication between the tools team and the target audience when discussing features and workflow using the tool. A strong example of a game development studio following this model is BioWare Corp. (Wihlidal, 2006, S. 5)

Das Nachfolgende Kapitel beschäftigt sich mit dem Aspekt des Projektmanagement im Tool Development und den von der Tool Entwicklung betroffenen Teammitgliedern (Stakeholder und Zielgruppen).

# 3 Produktionsprozess: Projektplanung und Analyse

## 3.1 Entwicklungsprozesse in Interaktiver 3D Software und Games

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 ein agiles Modell zur Softwareentwicklung eingesetzt. Hier soll ein kurzer Überblick über aktuelle Modelle entstehen. Diese Modelle ermöglichen zum einen das schnelle Entwickeln von Tools während der knappen Entwicklungszeit eines Spiele Produkts und zum anderen unterstützen sie die Arbeit von kleinen Teams, in welchen meist Tool Developement betrieben wird, innerhalb eines großen Entwicklerteams um so gezielt plötzlich auftauchende Aufgaben ohne lange Planung und viel Bürokratie lösen zu können. Damit ist ein fortschreiten des gesamten Projektablaufs gesichert und Entwickler können ihre Zeit hauptsächlich für die Entwicklung der Tools investieren.

### 3.1.1 Projektmanagement Modelle

Um große Projekte wie Computergames oder Interaktive Software zu entwickeln, bedarf es meist einer detaillierten Planung und einer exakten Rollenverteilung im Entwicklerteam. Es existieren verschiedene Methoden des Projektmanagement auf welche hier kurz im Zusammenhang mit der Arbeit eingegangen werden soll. Einige der Projektmanagement Modelle wirken auf die Arbeitsweise der Teammitglieder aus. Daher wird diese Arbeit hier keinen Umfassenden Überblick über Projektmangament Methoden geben, sondern nur solche Ansprechen die sich direkt oder indirekt stark auf das Tool Develop-

[1] Allgemein: Hochqualitative Spiele Software mit großem Entwicklungsbudget und einer Breiten Zielgruppe. Vgl. Cifaldi, 2005

ment auswirken.

### 3.1.2 Agiles Modell oder klassisches Modell

Viele Entwickler (Ubisoft, siehe Schmitz, 2014) 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 schwierige Situationen. Klassische Modelle (Wasserfallmodell (starres klassisches Modell), Spiralmodell (weiter entwickeltes iterativ orientiertes Modell)) haben hier meist Probleme durch ungleich höhere Bürokratie und Komplexität und benötigen ein Zeitaufwändigeres re-iterieren im Falle von Updates und Umstrukturierungen in Folge von unvorhergesehenen Ereignissen und Problemen. Hochkomplexe Software Projekte die über längere Zeiträume entwickelt werden können meist nur durch klassiche Projektmanagement Modelle 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 schnell-lebigen und agilen Projektmanagementmodell wie Scrum zu organisieren.

### 3.1.3 Scrum

Der Scrum Prozess tauchte das erste mal in der Veröffentlichung Hirotaka, Takeuchi, I. (1986). The new product development game. 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, siehe CCP Games, 2009, Schmitz, 2013, Martin Klekner, 2014.
Scrum eignet sich besonders für unkomplizierte und schnelle Prozesse. (Chandler, 2006) 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." Chandler, 2006, 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." Chandler, 2006, 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.

Bei diesen Rollen handelt es sich um das interne Scrum Team - das Entwicklungsteam des Produktes. 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 Bürokratischer Prozess die Entwicklung blockiert. Somit ergänzt sich dieser Prozess 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 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:

- 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.

Die Stakeholder des Tools wären in diesem Fall die anderen Entwickler des Unternehmens die das Produkt im Produktiefbetrieb einsetzen möchten. 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:

**Szenario**: Ein Team benötigt einen Textur-Editor / Tool um Texturen in das Format der Game-Engine zu transformieren.

- Der Antrag für das 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.

Dieser Prozess schließt iterrativ ab bzw. nicht ab, da während der Entwicklung einer interaktiven Anwendung / Games eventuell auch auf externe Einflüsse wie Third Party Software oder Marktentwicklungen eingegangen wird. Als Beispiel: So könnte sich durch die Veröffentlichung einer neuen GPU Generation oder den vorgezogenen PC Release die Größe der benötigten Textur-Dateien ändern.

## 3.2 Mitglieder eines Entwicklerteams

Es soll hier ein kurzer Überblick über die gängigsten Mitglieder eines Entwicklerteams gegeben werden. Grob können Entwickler in die folgenden Gruppen aufgeteilt werden - Artist, Designer, Engineer und Producer. Jede Gruppe arbeitet hierbei meist interdisziplinär mit den 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 Chandler, H. (2006). *The Game Production Handbook*. Thomson Delmar Learning, in welchem er die Produktionsprozesse eines Spiels sowohl in designtechnischer Weise als auch aus technischer Sicht beschreibt.
Diese Auswahl beschränkt sich auf Mitglieder des Teams welche mit dem Entwicklungs Prozess des Tool Authorings mehr oder weniger direkt in Verbindung treten.

### 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. Chandler, H. (2006). *The Game Production Handbook*. Thomson Delmar Learning, S. 19-20
DP können hier auch in das Entwicklen von Toolsets eingreifen. Sie koordinieren 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 kreiiren Texturen und User Interfaces. 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 [2], für die Verwendung im Spiel vorzubereiten. Programme wie Cinema 4D[3], 3DS Max[4], oder Modo[5] sind Beispiele für Kernsoftware dieser Entwickler. Der vollständigkeit halber sei hier noch das Open Source Projekt Blender[6] 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. (Chandler, 2006, S. 24).

### 3.2.3 Designer

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

[2] Assets sind Bestandteile des Produktes welche eine Grafische oder logische Repräsentation im Produkt erfahren. Dazu zählen z.B. Modelle, Texturen und Code Dateien.
[3] MAXON Computer GmbH, 2014-2015.
[4] Autodesk, Inc., 2015.
[5] The Foundry Visionmongers Ltd., 2015.
[6] Blender Foundation, 2014-2015.

### 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[7] Editor für Spiele basierend auf der idTech3 und idTech4 Engine [8], 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 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. (Chandler, 2006, S. 31 )
Die Konzeptionierung dieser Arbeit wird nurn 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 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 mitteilt, was er in der Spieltwelt zu tun hat. Hier sind allerdings viele Bestandteile eines Spiels anzuordnen. Meist werden

### User Interface Designer

User Interface Designer kümmern sich um das Erstellen von grafischen Schnittstellen welche die Interaktion mit dem Benutzer ermöglichen. Hierfür verwen-

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

[8] 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

den sie oft Scriptsprachen wie Actionscript von Adobe (Zur programmierung von Adobe Flash Interfaces) oder gar fertige Middelware wie Scaleform[9] ein Cross Plattform UI Solution Tool[10] 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. (Chandler, 2006, S. 31)

### 3.2.4 Engineer

Engineers / Ingenieure arbeiten meist am Kern der Applikation und schrieben den Source Code für die Anwendung, Engine, Netzwerkfunktionen, KI, und Tools. Diese Entwickler arbeiten hauptsächlich in einer IDE [11] wie Visual Studio (auf welches sich das zu dieser Arbeit konzeptionierte Tool bezieht) oder XCode [12]. 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 [13]. Vgl. (Chandler, 2006, S. 26)

#### Tool Engineer

Diese Arbeit bezieht sich auf den Bereich des Tool Development. 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 Textureditoren 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 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. (Chandler, 2006, S. 27 )

[9] Autodesk Inc., 2014-2015.
[10] Ermöglicht das erstellen von 2D, 2.5D und 3D Ui Elementen. Wird z.B. von der Unreal Engine 4 verwendet.
[11] Integrated Developement Environment
[12] X-Code ist nur für MacOSX erhältlich
[13] Logik des Spiels, ermöglicht das interagieren etc. mit und in der Software

#### 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. Sie beschäftigen sich ausserdem häufig mit der Production Pipeline zum erstellen der Game Assets. Vgl. (Chandler, 2006, S. 27 )

## 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."
>
> Ieee guide–adoption of the project management institute (pmi(r)) standard a guide to the project management body of knowledge (pmbok(r) guide)–fourth edition. (2011 November). *IEEE Std 1490-2011*, 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. 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. (Wihlidal, 2006, 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

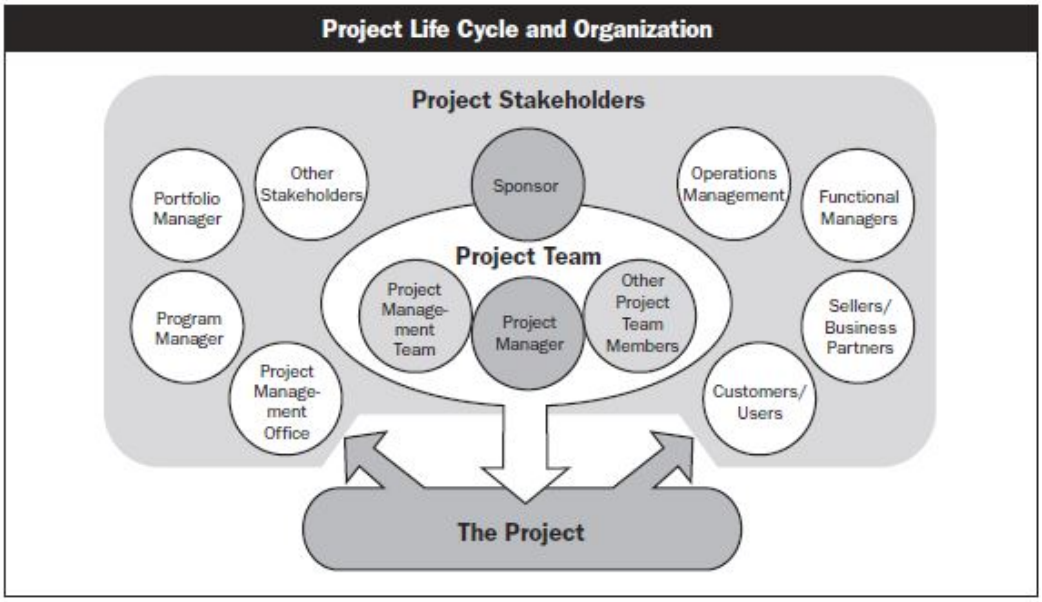



Abbildung 3.1: Überblick Stakeholder. Alle am Projekt beteiligten Personen oder Organisationen müssen beachtet werden. Grafik entnommen aus (»IEEE Guide–Adoption of the Project Management Institute (PMI(R)) Standard A Guide to the Project Management Body of Knowledge (PMBOK(R) Guide)–Fourth Edition«, 2011, S. 24).

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.
(Wihlidal, 2006) 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 [...]." (Wihlidal, 2006, S. 4-5)

Diese Definition deckt sich mit der ogen erwähnten allgemeinen Definition der Stakeholder. Lediglich die Weitläufigkeit der Stakeholder hält sich hier in Grenzen. 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 En-

gine [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 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.

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

### Identifizieren der Stakeholder

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.

### Involvement in das Projekt erkennen und bestimmen

Um zu erkennen in welchem Bezug ein Stakeholder zu einem Projekt steht, wurde ein Gittermodell 3.2 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.

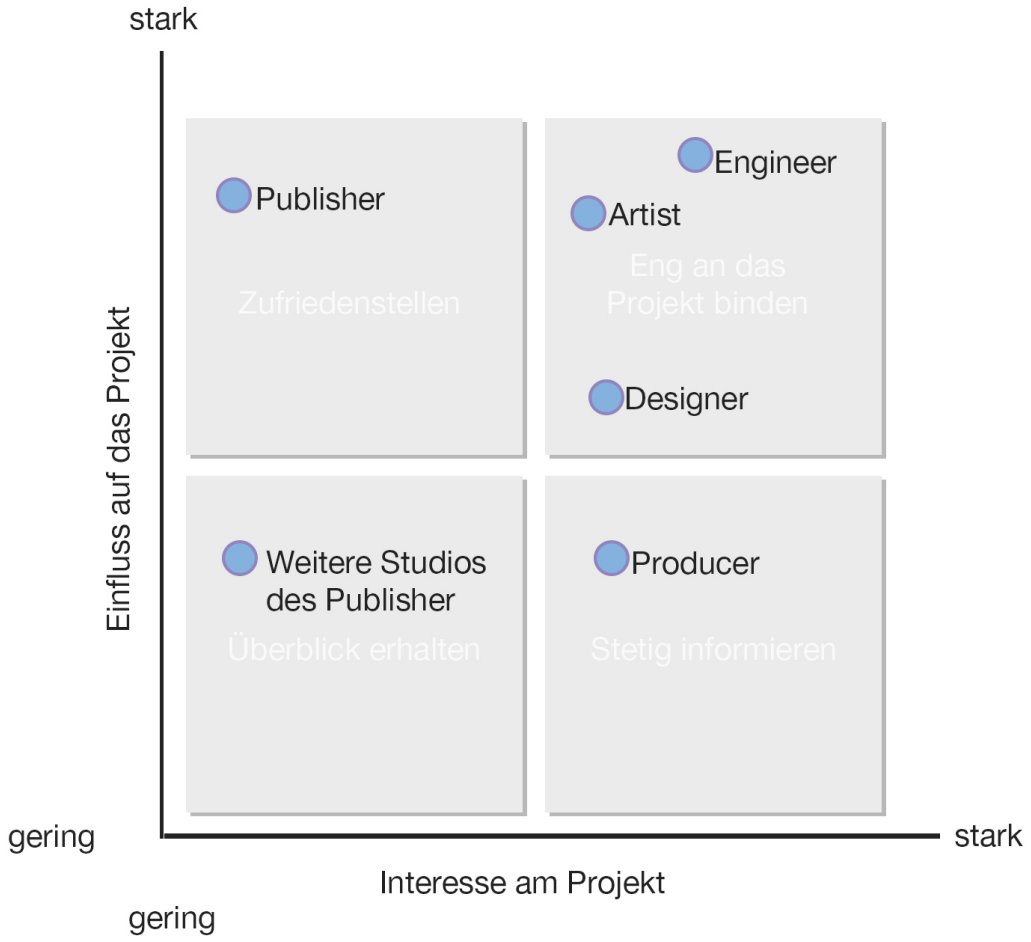



Abbildung 3.2: Überblick über die Identifizierten Stakeholder und deren Inovelement in die Planung und Umsetzung des Tool Projektes.

### Entwickeln einer Umsetzungsstrategie

Mithilfe der ermittelten Stakeholder und den oben angeführten Projektmanagementmodellen kann nach dieser Analyse ein Prozess und ein Konzept für ein Tool entwickelt werden. Die nächsten Abschnitte beschäftigen sich mit der

Entwicklung eines Arbeitsprozesses zur Konzeption eines Software Tools und dessen Umsetzung auf Basis der jetzt durchgeführten Analysen.

## 3.4 Der Arbeitsprozess zur Authoring Tool Entwicklung

### 3.4.1 Game Authoring / Game Development

Game Development beschreibt im Allgemeinen das Entwickeln einer Interaktiven Software, meist eines Spiels. Hierbei spielt es keine Rolle ob die Software der puren Unterhaltung dient oder sich dem Bereich der Serious Games [18] oder zuordnen lässt. Auch weitere interaktive Anwendungen ohne Unterhaltungsfaktor können in diese Kategorie fallen. Die Produktionsabläufe ähneln sich stark. Im Gegensatz zum Tool Development hat das Game Development meist den Auftrag am Ende ein für den Consumer zugängliches Produkt zu schaffen. Das Tool Development beschäftigt sich in erster Linie mit dem Erstellen der Werkzeuge welche benötigt werden um das Consumerprodukt zu entwickeln.
Es ist aber in der Tat so, dass Tool Development auch auf dem Consumer Markt ankommen kann. Verschiedene Entwickler stellen in der Vergangenheit ihre Tools den Fans und Kunden zur Verfügung. Aus diesen Tools haben sich selbst schon wieder Produkte wie Modifikationen[19], Patches oder Erweiterungen entwickelt. Einige Beispiele sind die das Creation Kit von Bethesda Softworks und die daraus entstandenen zahlreichen Fan Modifikationen wie Nehrim oder Enderal des Mod Teams SureAI[20].

### 3.4.2 Tool Development und Asset Pipelines

Um eine Produktion erfolgreich zu starten, bedarf es wie weiter oben angeführt einer Asset Pipeline und den darin enthaltenen Tools. Um zu ergründen, welche Tools für das Projekt nötig sind, sollten sich Lead-Developer zu Beginn des Projekts folgende von Chandler, 2006 beschriebene Fragen stellen.

- What tools and software are needed?
- Can the pipeline support two-way functionality? (Konvertieren von Assets in das Game Format und wieder zurück in das Ausgangsformat)
- What is the critical path? Are there any bottlenecks?

[18] "[...] serious games are (digital) games used for purposes other than mere entertainment." Tarja, Susi, 2014

[19] auch Mods genannt, Verändern das eigentliche Spiel oder ersetzen es in Form einer "Total Conversion" gleich ganz durch neue, eigens entwickelte Inhalte

[20] Freies Modifikations Team. http://www.sureai.net

- When does the system need to be fully functioning?
- How are assets managed and tracked in the system? (Entscheidung für eine “Version Control Software”)
- Which areas of the system can be automated?

Vgl. (Chandler, 2006, S. 224-225)

Carey Chico[21] (als Experte im Werk von Chandler, 2006 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 [...] 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. (Chandler, 2006, S. 224-225)

Hiermit zeigt Chico ein Kernthema dieser Arbeit auf: Wichtig sind Tools, welche auf dem Team bereits bekannter Software basieren, in ihrer Bedienung leicht zu verstehen und jedem Mitglied des Teams zur Verfügung stehen. Solche Authoring Tools könnten den Produktionsprozess zugänglicher gestalten. Da eine Erweiterung eines intern entwickelnten Tools durch die Entwickler dauerhaft möglich wäre, kann auch ein langfristiger Einsatz derer gewährleistet werden und rechtfertigt somit eine aufwändigere Entwicklungsarbeit.

### 3.4.3 Tool Development - der organisatorische Ablauf

Nach Wihlidals Auffasung (Wihlidal, 2006) unterscheidet sich die Planung eines Projektes zur Erstellung eines Developer Tools nicht sehr von der Planung zur allgemeinen Entwicklung von Software. Es gelten hier vier Planungsphasen die den Projektablauf kennzeichnen. Der erste Schritt wäre eine allgemeine

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

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 wieder iterativ an den Features des Tools gearbeitet werden und vor allem schnell auf Hindernisse und Wünsche der Stakeholder reagiert werden.
Im folgenden Abschnitt wird noch etwas genauer auf die jeweiligen Schritte der organisatorischen Planung eingegangen.

### Planung

Es folgt die Planungsphase des Tools. Hier wird zuerst eine Requirementsanalyse [22] 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

- 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.

[22] dt. Anforderungsmanagement

- 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.
- 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 sozusagen ein rechtlich relevantes Dokument dar. Ein Dokument mit Bezug auf den Praktischen Teil dieser Arbeit findet sich 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 [23] 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.

[23] Unified Modeling Language

### 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.

# 4 Entwicklung eines Konzeptes

## 4.1 Use Cases der verschiedenen Entwickler

Diese Abschnitt behandelt eine Aufstellung von alltäglich Tasks (z. Dt. Aufgaben) und den Arbeitsschritten die dazu nötig sind diese Aufgaben der einzelnen Teammitglieder zu erledigen. 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.

### 4.1.1 Was möchten 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 bedient.

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

Die Übersicht 4.1 beschreibt die Einbindung des Artist in das System und skizziert seine Use Cases in Verbindung mit den Modulen des gesamten Systems auf.
Es folgt eine Betrachtung der einzelnen Tasks vom Standpunkt des Requirementsengineering aus. Anschliessend wurden die wichtigsten Aktivitäten eines Tasks herausgefiltet und hierraus ein System entwickelt.


Abbildung 4.1: Überblick über die Use Cases eines Artist.

### 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.

### 4.1.2 Was möchte ein Engineer?

Diese Auflistung beschreibt die Tasks und Lösungwege des Engineer.

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

Die Übersicht 4.2 skizziert die Verwebung des Engineer mit seinen Use Cases und dem System.
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.

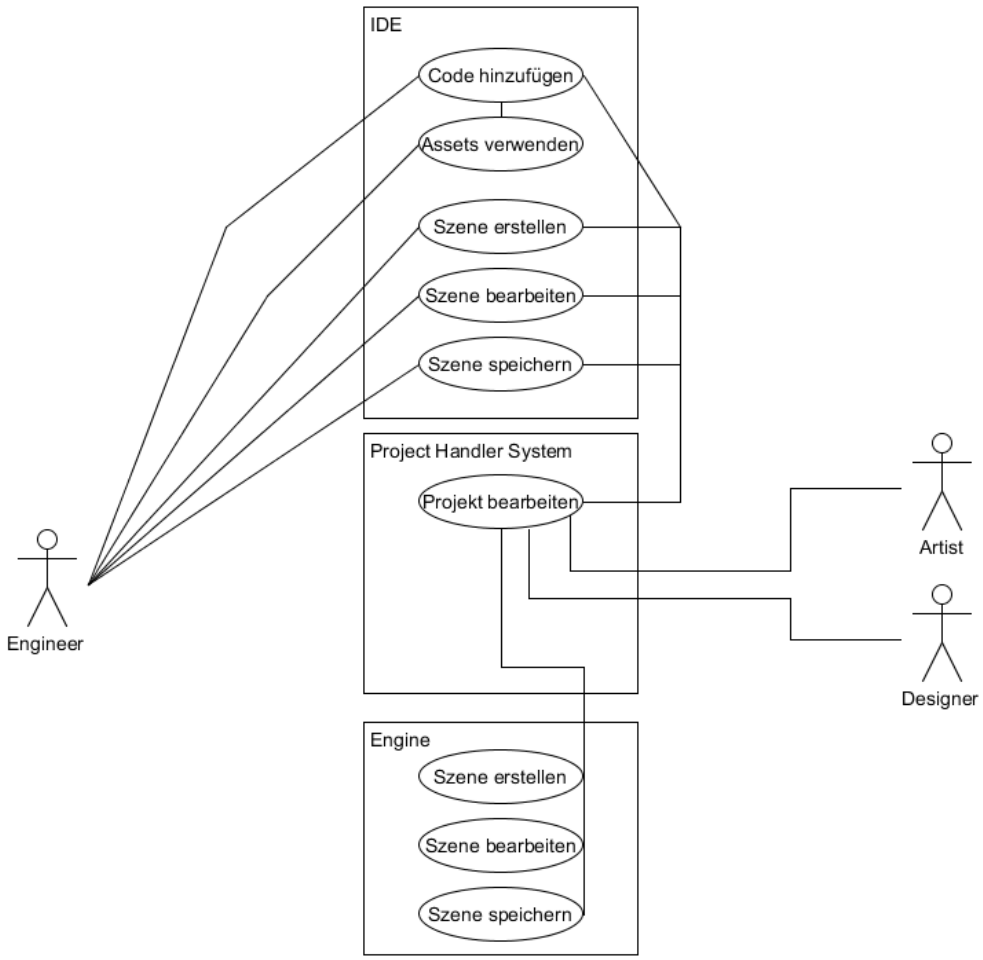



Abbildung 4.2: Überblick über die Use Cases eines Engineer.

### 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 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 Ei-

genschaften. 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.

## 4.2 Aktuelle Engines und deren Arbeitsprozesse

Es folgt ein Überblick über aktuell auf dem Markt vorhandene Game Engines. Aufgrund der Thematik beschränkt sich die Auswahl auf die an der Hochschule Furtwangen populärsten Game Engines und analysiert deren Gegebenheiten und stellt die Arbeitsabläufe in diesen Engines und Tools dar. Es folgt eine wertung und ein Vergleich mit den für das Fusee Authoring Toolit geplanten Features.

### 4.2.1 Prozesse in Game Engines und/oder Frameworks

### 4.2.2 Unreal Engine 4

Die UnrealEngine4[1] (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.

[1] EPIC GAMES, INC. (2004-2015). Unreal Engine 4. http://www.unrealengine.com. EPIC Games INC.

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.)

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[2] 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 gewünschten Aktion.

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

### 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 4.3 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.

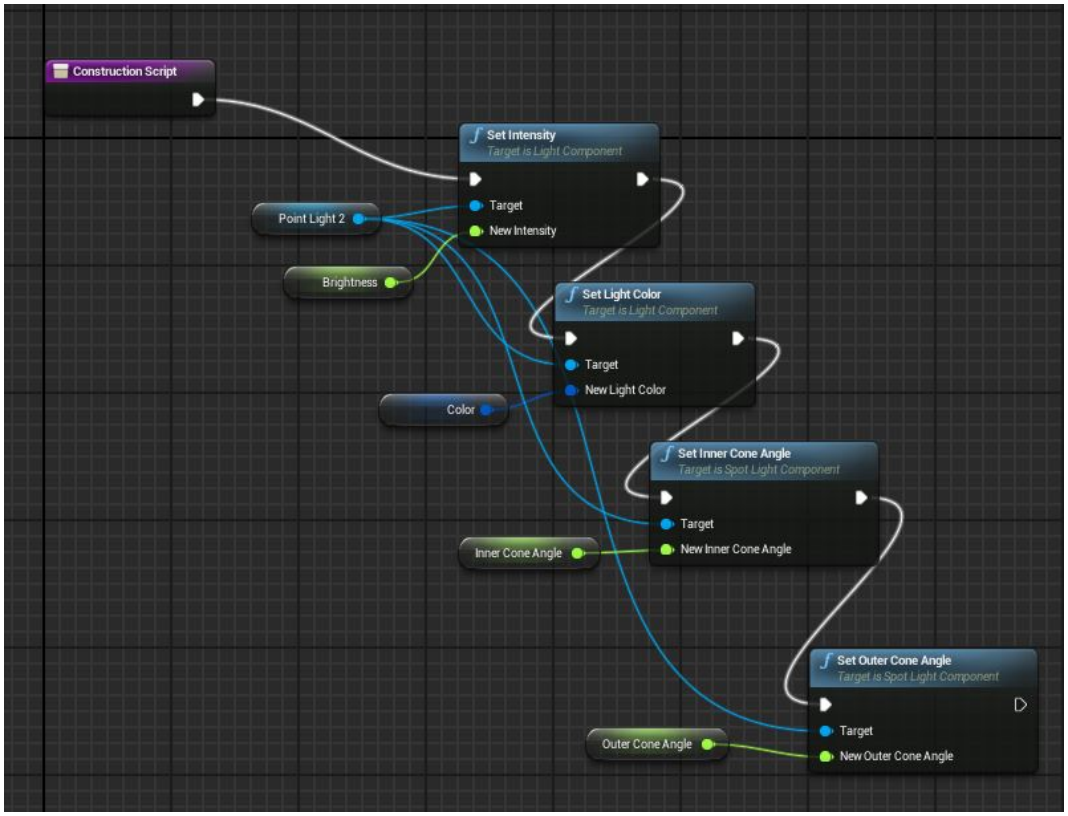



Abbildung 4.3: Programmierung in der Unreal Engine 4 mit Hilfe des Blueprint Visual Programming Systems/Interfaces

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 4.1[3] zeigt mehrere solcher Markup Befehle vor den jeweiligen Funktionen, Pro-

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

perties der Klasse. Es handelt sich hierbei um Bezeichnungen wie z.B. UENUM(), UCLASS(), USTRUCT(), UFUNCTION(), und UPROPERTY().

Sourcecode Beispiele 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:
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. (EPIC GAMES, INC., 2004-2015) - 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 4.4) 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.

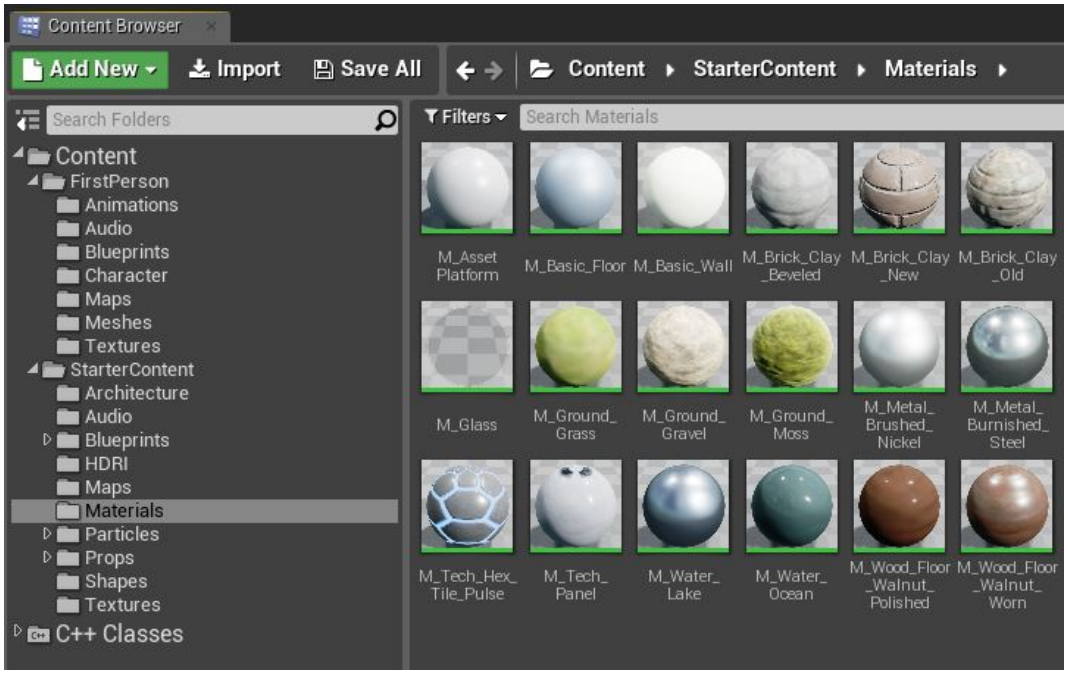



Abbildung 4.4: Ansicht des Contentbrowsers der Unreal Engine 4. Mit Suchfeld und Filterfunktionen, Assetvorschau ausklappbarer Ordnerstruktur.

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 4.5 und seine Gemeinsamkeiten mit dem Fusee und Cinema4D Szenengraphen. Der Szenengraph hängt unmittelbar

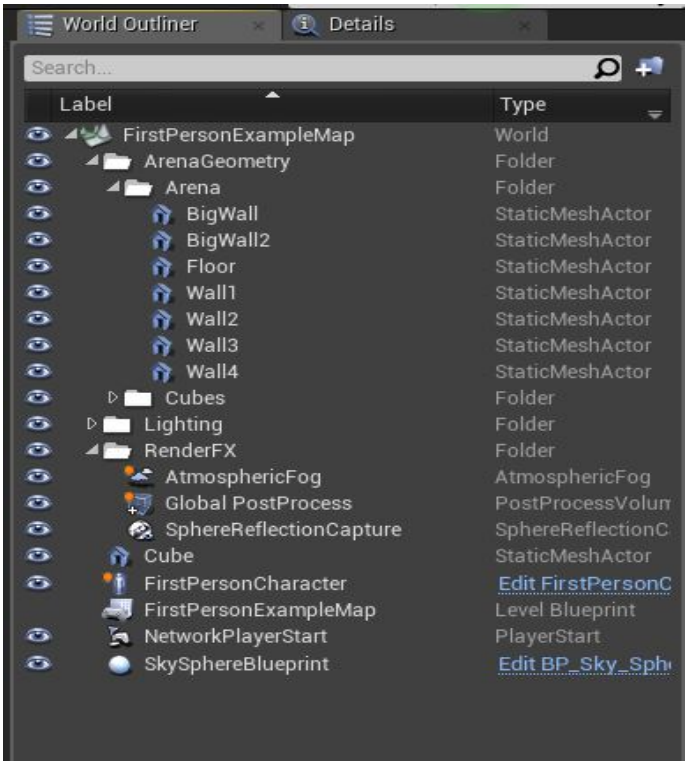



Abbildung 4.5: Beispielhafte Szene im Szenengraph der Unreal Engine 4. Er gleicht dem Fusee und Cinema 4D Szenengraphen stark.

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 beitet auch eine so komplexe und tiefgängige 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 Programming 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 fand damals den Weg in das 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.

### 4.2.3 Unity 3D

Die Unity 3D Engine [4] in der Version 5 wird von Unity Technologies entwickelt. Das Lizenzmodell sieht eine kostenfreie Variante mit einigen Einschränkungen 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

[4] Unity Technologies. (2015). Unity 5. http://www.unity3d.com. Unity Technologies

- Linux
- Web Player
- Playstation 3 (In Verbindung mit einer DevKit Lizenz zu erwerben bei Sony.)
- Playstation 4 (In Verbindung mit einer DevKit Lizenz zu erwerben bei Sony.)
- Playstation Vita (In Verbindung mit einer DevKit Lizenz zu erwerben bei Sony.)
- Playstation Mobile (In Verbindung mit einer DevKit Lizenz zu erwerben bei Sony.)
- XBox One (In Verbindung mit einer DevKit Lizenz zu erwerben bei Microsoft.)
- XBox 360 (In Verbindung mit einer DevKit Lizenz zu erwerben bei Microsoft.)
- Wii U (In Verbindung mit einer DevKit Lizenz zu erwerben bei Nintendo)

Als Entwicklerplattformen unterstützt Unity Windows und Mac OSX. Der Sourcecode der Engine ist nur in einer Enterprise Version erhalten. Zum Erhalt dieser müssen gesonderte Verhandlungen mit Unity Technologies aufgenommen 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 **??** 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.

Sourcecode Beispiele 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 gehenede Prozesse einer GameEngine 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 Erstellung) 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.

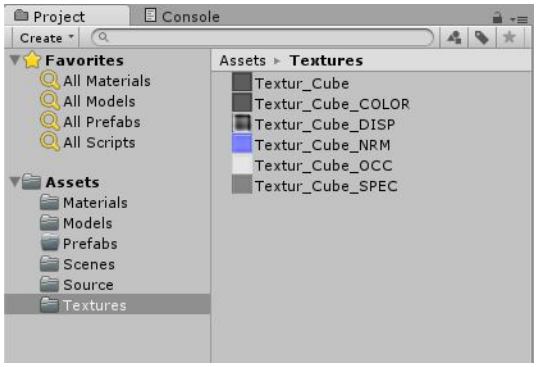



Abbildung 4.6: Ansicht des Contentbrowsers in Unity. Mit Suchfeld und Filterfunktionen und ausklappbarer Ordnerstruktur.

Um Assets in Unity zu bearbeiten, zu verwalten und zu durchsuchen steht ein Contentbrowser, siehe Abbildung 4.6, mit eben genau diesen Funktionen zur Verfügung. Grundsätzlich bildet dieser Contentbrowser alle Möglichkeiten ab, die auch FuseeAT unterstützen möchte.

### 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.

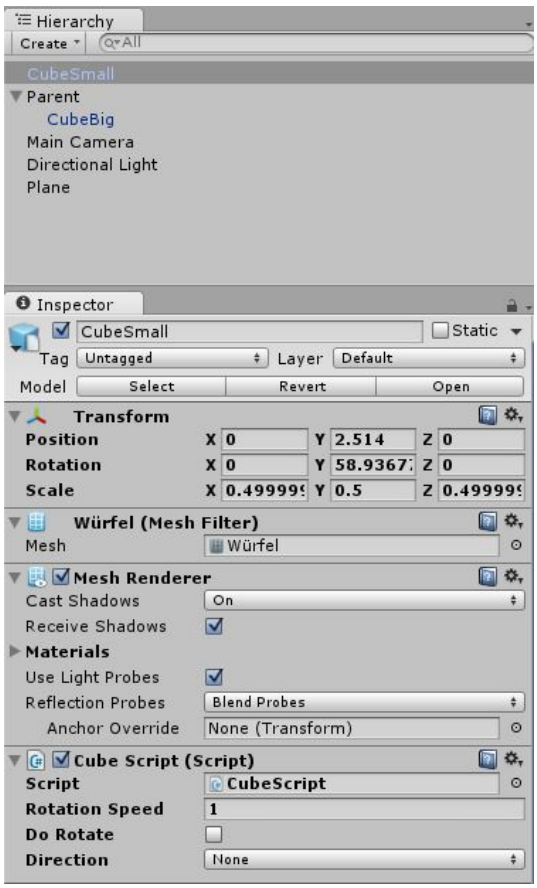



Abbildung 4.7: Beispielhafte Szene im Szenengraph von Unity. Er gleicht dem Fusee und Cinema 4D Szenengraphen stark. Das Detailpanel zeigt Components eines Nodes.

### 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. Ddie 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 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.

## 4.3 Systemdesign

Dieser Abschnitt beschreibt das durchgeführte 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. Einge Features werden diskutiert und die Entscheidung für oder gegen diese Erläutert.

### 4.3.1 Warum Fusee und Cinema 4D?

Für diese Arbeit wurden die beiden Applikationen Fusee[5] und Cinema 4[(MAXON Computer GmbH, 2014-2015) aus verschiedenen Gründen als Ausgangsbasis für die Forschung ausgewählt. Bei der Fusee Engine handelt es sich um eine von der Hochschule Furtwangen eigens Entwickelte 3D Engine unter der Leitung von Herrn Prof. C. Müller der Fakultät DM. Sie wird meist durch Projekte, Abschlussarbeiten und Arbeitsgruppen weiter entwickelt. Bei der Engine handelt es sich um eine 3D Engine deren Quellcode komplett Open Source ist. Das komplette Projekt inklusive seiner Abhängigkeiten ist von GitHub[6] 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 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 größeren, höher aufgelösten Version im Format Din A4.

### 4.3.2 Systemdesign für das Toolkit

Das Fusee Authoring Toolkit (kurz FuseeAT), in der folgenden Abbildung 4.8 in grün dargestellt, ist das Kernsystem dieses Konzeptes. Es handelt sich hierbei um eine Sammlung von Funktionen in Form einer C# Softwarebibliothek

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

die es ermöglichen soll einen Editor[7] in ein Game Authoring Tool für die Erstellung von Interaktiven Welten zu transformieren.
Die grau hinterlegten Bestandteile der Abbildung 4.8 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

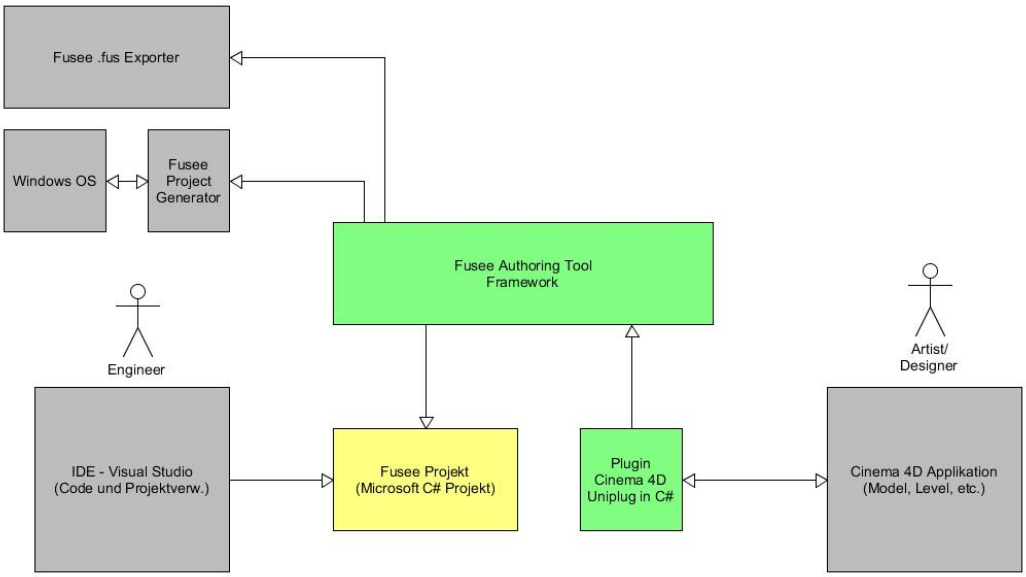



Abbildung 4.8: Überblick über die Systemarchitektur

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 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.

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

Damit das System erfolgreich arbeiten kann ist es nötig, aus Cinema 4D verschiedene Information an das FuseeAT weiterzureichen. 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 Cinema4D 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 4.8 ebenfalls in grün dargestellt, stellt also nur die Schnittstelle des FuseeAT zur Software Cinema 4D dar und dient in diesem Fall als Kommunikatorensystem zwischen den verschiedenen Architekturen C++ und C#.

### 4.3.3 Systemdesign für ein Tool Framework

Die folgende Abbildung 4.9 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

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

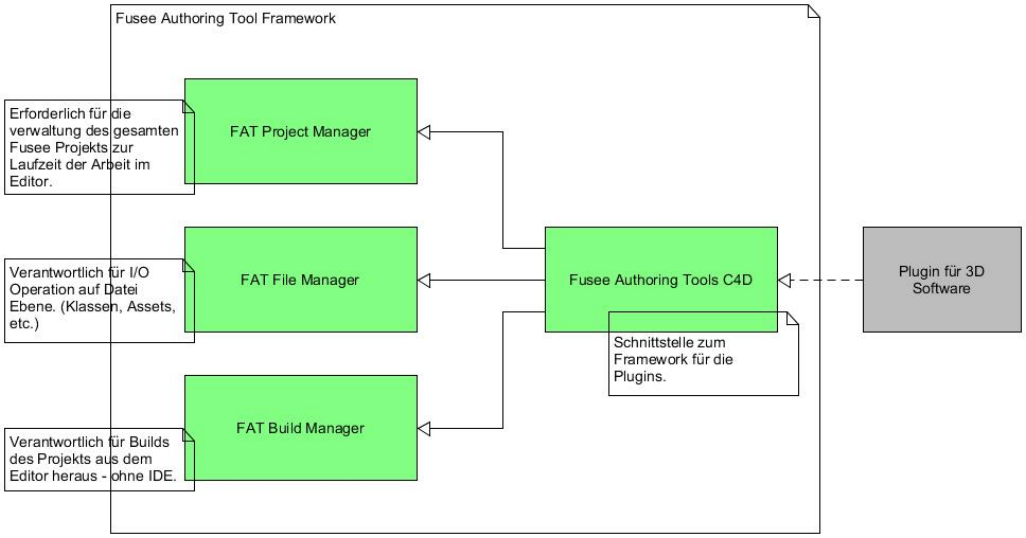



Abbildung 4.9: Überblick über das Fusee Authoring Toolkit Framework

Microsoft bereitgestellten MSBuild[8] Systems. Er kümmert sich um Ausgabeverzeichnisse und weitere Optionen die während eines Buildprozesses Angaben 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 4.10 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. FuseeATCinema4D 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 FuseeATCinema4D 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 Klassendiagramm 4.10 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 ge-

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

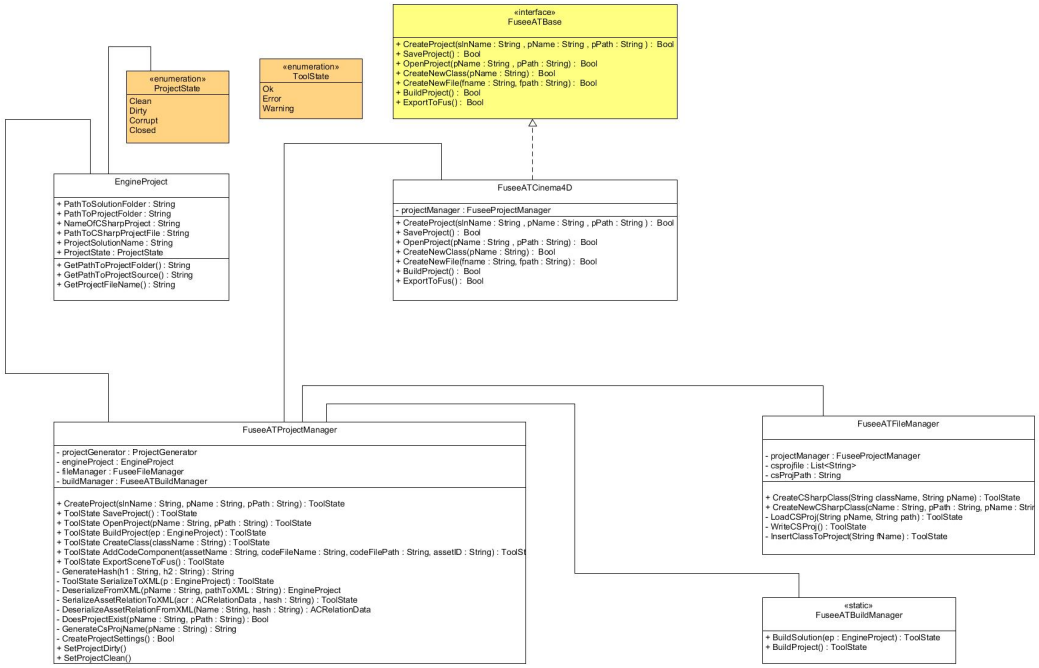



Abbildung 4.10: Klassendiagramm für das Fusee Authoring Toolkit

währleisten. 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.

### 4.3.4 Systemdesign für ein Plugin System

Die Abbildung 4.11 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.
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 Funktionali-

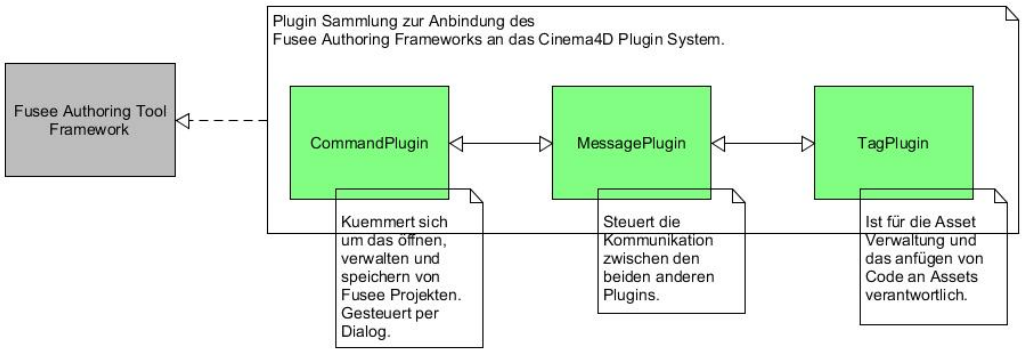



Abbildung 4.11: Überblick über die Pluginarchitektur

tät 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[9]) und weiter verarbeitet. Das hier angeführte Klassendiagramm 4.12 zeigt die für jedes Plugin benötigten 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.

### 4.3.5 Zeitersparnis durch bekannte Tools

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.

#### 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 in einerhalb 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

[9]Von der Cinema 4D API festegelegt.

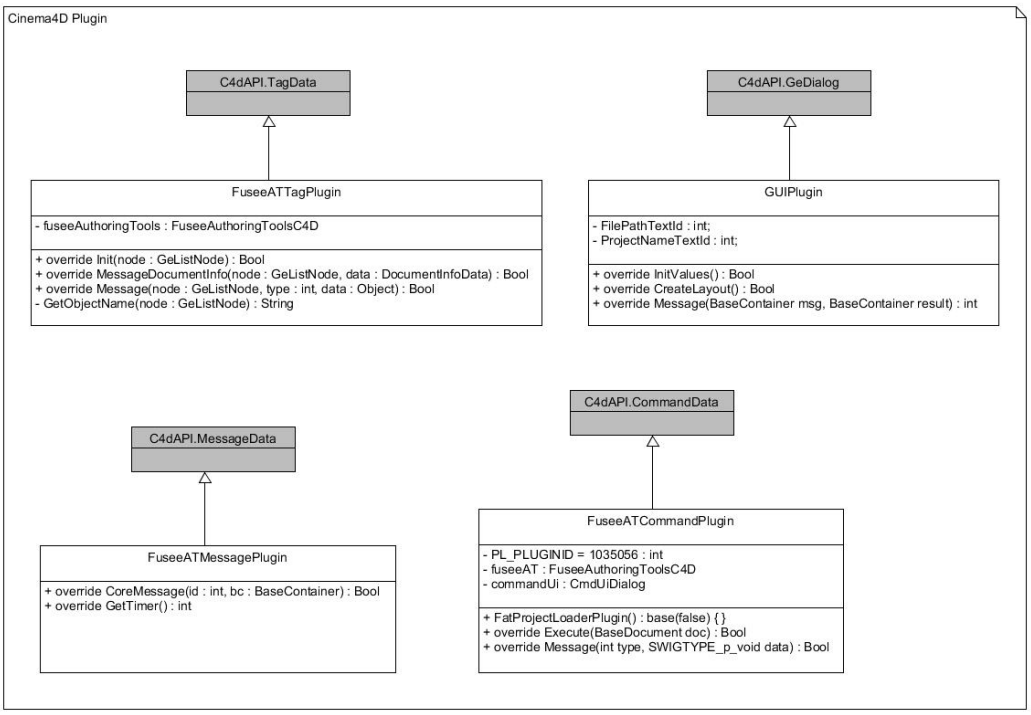



Abbildung 4.12: Klassendiagramm der Plugin Architektur

und verdrahtet werden. Die folgende Abbildung 4.13 zeigt die Struktur eines solchen Fusee Projektes. Die Solution Datei beinhaltet die zwei Projekte “Sim-

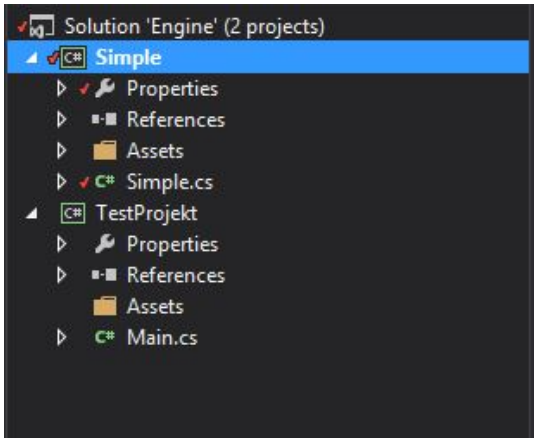



Abbildung 4.13: Fusee Binary Projekt Struktur

ple” und “TestProjekt”. 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 Cinema4D 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[10] zu Fusee.

## 4.4 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. Meist handelt es sich bei in Fusee eingebundenen Assets um eines der folgenden Asset-Typen:

- 3D-Modelle in den Formaten .obj[11] (Bei dem Dateiformat handelt es sich um das lesbare ASCII Model Datenformat “Wavefront Object”[12])
- 2D-Texturen in den Formaten (JPEG, JPG, PNG)
- Audio Datei im Format MP3, Wav, etc. (Unterstützt alle von der SFML “Simple and Fast Multimedia Library http://www.sfml-dev.org/ unter-

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

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

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

stützen Formate." Inkl. eigener Implementierung einer MP3 Unterstützung durch den Fusee Entwickler Fabian Gärtner.)

### 4.4.1 Asset Pipelines in Fusee Authoring Toolkit

Die folgende Grafik 4.14 [13] ergänzt die Erläuterung zum Asset Authoring (Eines .fus Objektes mit angehängtem Code Component[14] 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 wird Beispielhaft ein Model in Cinema4D 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 Cinema4D Szene benötigt um eine Zuordnung eines Asset in der Szene wiederherzustellen. Das erstellte Asset bleibt weiterhin im Cinema4D Format, gespeichert in der Cinema4D Datei des Projektes, vorhanden. Somit besteht das Asset in einer nativen Version weiterhin und kann in einem iterativen Prozess weiterhin verändert werden.
Während des Build- und/oder Speichervorgangs des Cinema4D und Fusee Projekts über das FuseeAT wird das Asset vom Framework in ein Asset des Fusee Typs .fus gewandelt. Allerdings geht durch die bereits erwähnre redundante Speicherung des Asset im Cinema4D Format keine Information der ursprünglichen Datei verloren.

### 4.4.2 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 Assetmanagment 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 (Avid Technology, 2009)[15] und die Software Shotgun (Shotgun Software Inc., 2013)[16]. 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

[13] Abbildung entnommen aus Carter, 2004, S. 7 und neu Illustriert und beschriftet.
[14] Ein Extra Element welches an einem Asset Objekt angehängt ist
[15] Webadresse: http://www.alienbrain.com/
[16] https://www.shotgunsoftware.com/

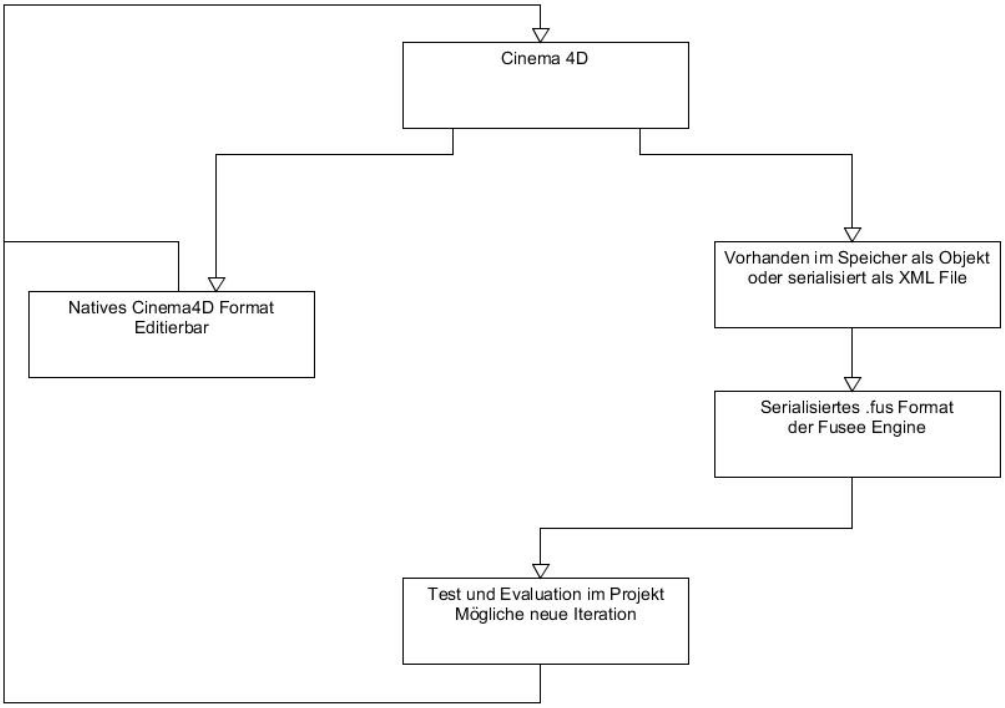



Abbildung 4.14: Asset Loop während der Produktion eines Projektes in Fusee und FuseeAT

kann ohne Probleme neben dem FuseeAT eingesetzt werden, da sich die beiden Wirkungsbereiche nicht überschneiden.

## 4.5 Die Cinema 4D C++ API und ihre Verwendung in diesem Projekt

Für die Implementierung in Fusee und dem Cinema 4D Pluginsystem war es nötig, auf das Uniplug System der Hochschule Furtwangen [17] 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.

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

### 4.5.1 In der Implementierung verwendete Softwareprojekte / Ausgangssituation

Das Projekt "Fusee Authoring Toolkit" [18] 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
- 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 Cinema4D Programmcode ist nicht einsehbar.)
- Verschiedene Windows Bibliotheken aus dem .NET Framework auf welche hier nicht weiter eingegangen werden soll

### 4.5.2 Cinema 4D Plugin API und SDK

Cinema 4D R16 (MAXON Computer GmbH, 2014-2015) 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[19] und die Erweiterung des Cinema 4D Funktionsumfangs eine API[20] 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 kannmit 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 mitinstalliert.

[18] Der Name der zu dieser Arbeit zugehörigen Softwarebibliothek
[19] Plugins in C4D - Erweiterungen für die Software Cinema 4D und deren Funktionen
[20] Application Programming Interface, z. Dt. Programmierschnittstelle

Auf einem Windows PC findet sich das SDK und die dazugehörigen Visual Studio Solution Dateien unter folgendem Pfad ausgehend vom Installations Ordner der 64Bit[21]: “Cinema 4D/plugins/cinema4dsdk”

Maxon bietet Beispiele zur Cinema 4D API auf einem GitHub Account[22] unter der Apache License Version 2.0, January 2004[23] 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 4.15 erläutert die Funktionsweise des Cinema 4D Plugin Systems. Der Entwickler kann seinen C++ Code fast frei 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.
Das hier dargestellte Minimalbeispiel 4.3 in C++ Code zeigt ein einfaches minimalistisches Plugin welches beim Ausführen nur eine Ausgabe auf der Konsole von Cinema 4D erzeugt.

[21] 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
[22] https://github.com/PluginCafe
[23] Apache Licence 2.0 https://github.com/PluginCafe/cinema4d_cpp_sdk/blob/master/LICENSE

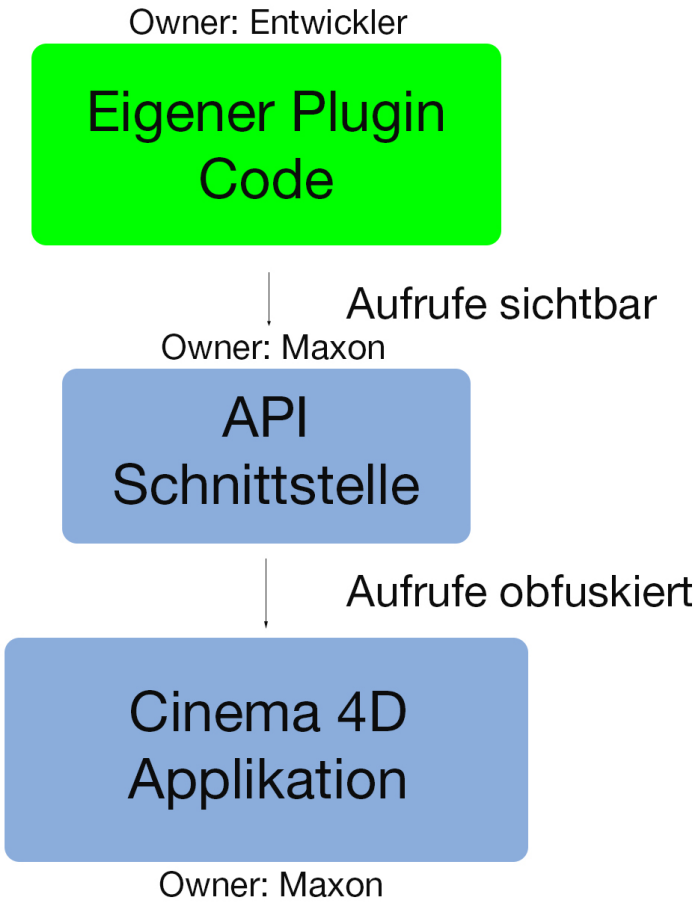



Abbildung 4.15: Schaubild des Cinema 4D API Systems

Sourcecode Beispiele 4.3: 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.

### 4.5.3 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 [Beazley, David, S., Fulton, William. (1995-2015). SWIG-3.0. http://www.swig.org/]. Das gesamte Uniplug Projekt ist ein Open Source Projekt und steht auf 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 4.16 dargestellten Module

sind Bestandteil des Uniplug Systems.

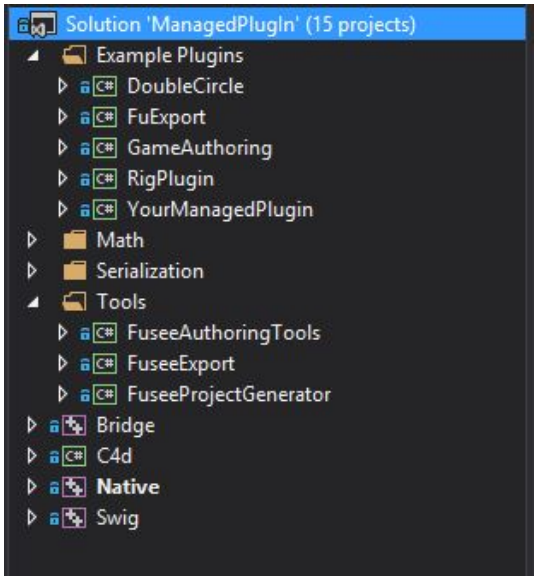



Abbildung 4.16: Die Uniplug Projektstruktur in VisualStudio

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 Programmiersprache 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).
Diese kurze und knappe Beschreibung von SWIG beschreibt ddie Möglichkeiten des Tools kurz und knapp:

> SWIG is an interface compiler that connects programs written in C and C++ with scripting languages such as Perl, Python, Ruby, and Tcl. It works by taking the declarations found in C/C++ header files and using them to generate the wrapper code that scripting

> languages need to access the underlying C/C++ code. In addition, SWIG provides a variety of customization features that let you tailor the wrapping process to suit your application.
>
> Auszug aus: - Beazley, David, F. (2014). Swig introduction documentation.

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 4.17 im Uniplug Projekt verfügbar ist enthält eine C4dApi.i (interface) Datei.

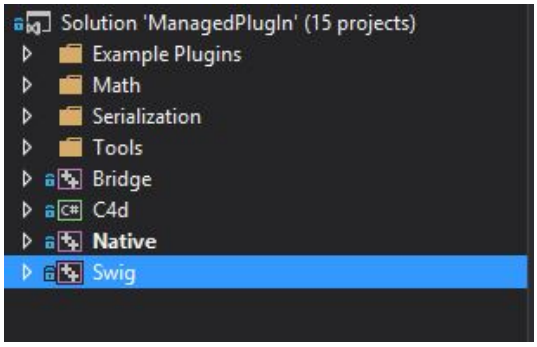



Abbildung 4.17: SWIG im Uniplug Projekt (ManagedPlugIn) markiert durch blaue Selektion.

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 Pluginentwicklung nicht bei jedem Build durchgeführt werden. Es genügt, die API erneut zu wrappen wenn Veränderungen 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 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

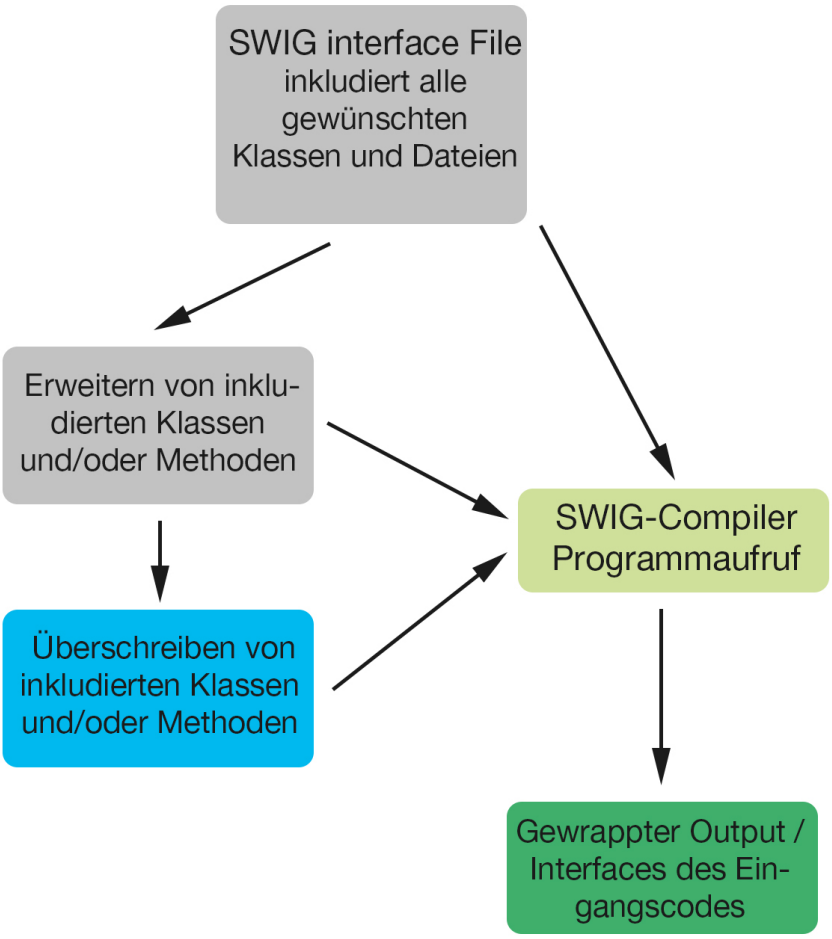



Abbildung 4.18: Wrapping Vorgang der Cinema 4D API von C++ nach C#

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.

Sourcecode Beispiele 4.4: 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 Cinema4D. 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 verar-

beitet werden.

Im folgenden Code Beispiel sind einige dieser Konstanten dargestellt. Bei dem hier in Beispiel 4.5 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.

Sourcecode Beispiele 4.5: 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
7      break;
8    case MSG_GETCUSTOMICON:
9      break;
10   case COLORSYSTEM_HSVTAB:
11     break;
12   case MSG_DOCUMENTINFO:
13     {
14       DocumentInfoData* did = (DocumentInfoData*)data;
15       return MessageDocumentInfo(node, did);
16     }
17     break;
18   case MSG_DESCRIPTION_GETINLINEOBJECT:
19     break;
20   case DRAW_PARAMETER_OGL_PRIMITIVERESTARTINDEX:
21     break;
22   }
23
24   return true;
25 }
```

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[24] sondern nur durch die nachträgliche Implementierung der oben dargestellten switch case Anweisung innerhalb des Uniplug Projektes zu bewerkstelligen.

### 4.5.4 Aus Fusee verwendete Software-Module

Wie bereits erwähnt, wird der Code der Fusee Engine für diese Arbeit nicht erweitert. Es wird im Systemdesign aber ein wichtiges Element der Engine, der Szenengraph, verwendet. Dieser Abschnitt geht auf die Funktionsweise des Szenengraphen in Fusee ein und zeigt seine Ähnlichkeiten und Unterschiede zum Szenenaufbau einer Cinema4D Szene auf. Hierzu werden die beiden Systeme analysiert und dann ein Design zum Mapping des Cinema4D Graphen auf das Fusee System erläutert.

#### Der Fusee Szenengraph

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. Die Node enthält lediglich einen Namen, eine Liste an Components und falls nötog, eine Liste an Kind-Nodes. Das Traversieren wird durch Visitor Implementierungen ermöglicht. Diese Visitor Patterns wurden während dieser Arbeit noch implementiert und erweitert. Ein Visitor ermöglicht das Traversieren einer Szene und seiner Nodes auf eine vom Entwickler definierte weise und kann so z.B. Transformationen durchführen, welche die Struktur einer Szene verändern.
Die Abbildung 4.19 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 Abbildunf 4.19 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. Ein

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

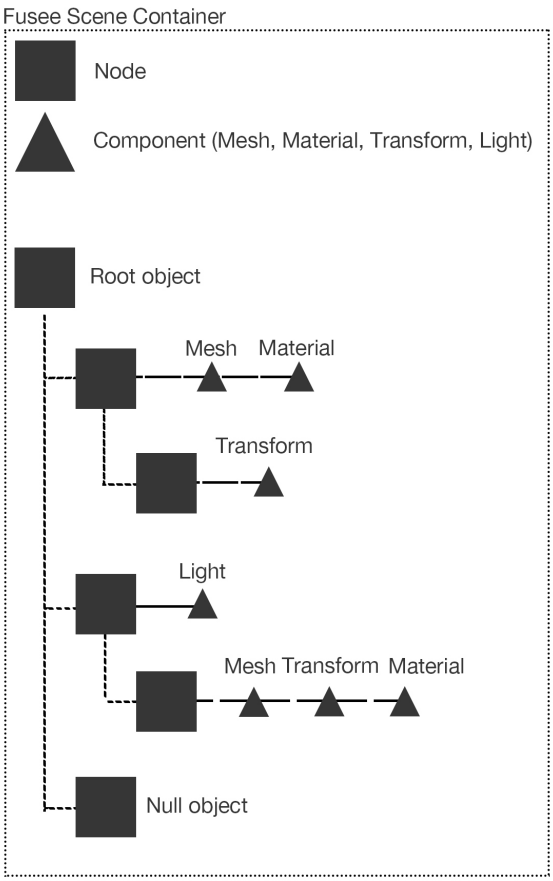



Abbildung 4.19: Der Fusee Szenengraph in exemplarischer bildlicher Darstellung.

kommentiertes Source Code Beispiel zur Anwendung des Fusee Szenengraphen findet sich im Anhang in Listing 5.1.

### Unterschiede des Fusee Szenengraphen zum Cinema4D Szenengraphen

Der Fusee- und der Cinema 4D Szenengraph unterscheiden sich in gewissen Konzeptionellen Ansätzen. Um also den C4D Szenengraph in einen Fusee Szenengraph 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.

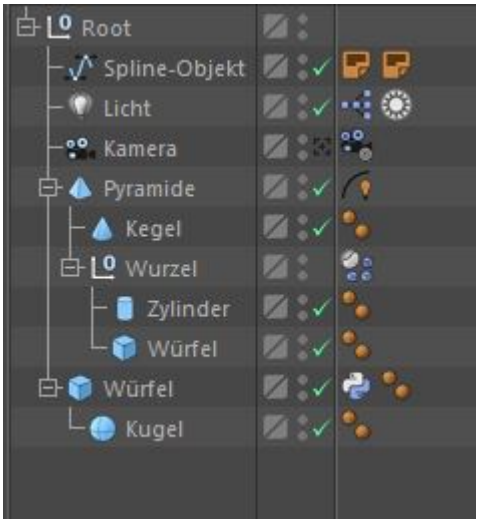



Abbildung 4.20: Der Cinema 4D Szenengraph. Verschiedenste Objekte als "Nodes", daneben als Icons exemplarisch Tags verschiedener Tag-Datentypen am rechten Rand der Grafik.

Diese Objekte können wieder Kind-Objekte der gleichen Datentypen beinhalten. Diese Hierarchie zeigt die Abbildung 4.20. Die in der Abbildung 4.20 am rechten Rand durch kleine Abbildungen neben den Nodes dargestellten Tags, beinhalten weitere Daten oder Einstellungen für das jeweils zugehörige Node. Diese können in einem anderen Fenster (Attribute Fenster/Panel) in Cinema 4d bearbeitet werden.

#### Der Export einer Cinema 4D Szene in den Fusee Szenengraphen

## 4.6 Fortschritt 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.

### 4.6.1 Bereits umgesetzte Teile 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
- TODO: Eine Code Datei kann mit Hilfe eines Tags in Cinema 4D an ein Objekt angehängt werden. Hierzu wird im Speicher eine XML Datei angelegt und auf die Festplatte geschrieben, welche die Assoziation der beiden Objekte beschreibt.

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

### 4.6.2 Implementierung: Generieren eines Fusee Projektes

Um ein Projekt für die Fusee Engine zu generieren, wird die folgende Funktion 4.6 des FuseeAT eingesetzt.

Sourcecode Beispiele 4.6: 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.

### 4.6.3 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 4.7 ist die folgende:

Sourcecode Beispiele 4.7: 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 4.8 dargestellten Attribute unterstützen das XML System beim serialisieren und deserialisieren der Objekte.

Sourcecode Beispiele 4.8: 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.

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

Das Erzeugen einer neuen Klasse kann über die Funktion 4.9 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.

Sourcecode Beispiele 4.9: Funktion zum Erstellen einer neuen Klasse im C# Projekt.

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

### 4.6.5 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.

### 4.6.6 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 4.10 können Probleme während der Laufzeit im FuseeAT System aufgedeckt werden. Das Konzept sieht vor, als return Werte der FuseeAT Funktionenjeweils 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 Projekt weiter zu arbeiten.

Sourcecode Beispiele 4.10: ToolState System unterstützt beim verbessern der Stabilität des FuseeAT.

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

### 4.6.7 Der ProjectState in FuseeAT

Durch die simple Implementierung des Enums ProjectState 4.11 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.

Sourcecode Beispiele 4.11: Code des ProjectState Enums. Wird in FuseeAT verwendet um die Integrität eines Projekts zu erhalten.

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

### 4.6.8 Problematische Aspekte während der Umsetzung des Konzepts

Problematisch war, dass die Uniplug Softwarebibliothek zum Zeitpunkt der Implementierung nicht die nötigen Funktionen der Cinema 4D API unterstützt. 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. Um die Entwicklung des Plugins für die Verwendung der FuseeAT bibliothek zu beginnen, mussten verschiedenste Funktionen der Cinema 4D API in das Uniplug Projekt integriert werden. 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.
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 gewünschten Funktionen implementiert werden konnten. 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.

# 5 Ergebnisse der Arbeit und Ausblick

## 5.1 Wie weit ist das Projekt Fortgeschritten?

Die Konzeption eines Authoring-Tools und die Untersuchung und Entwicklung eines Arbeitsprozesses für das Entwickeln von Authoring Tools in internen Teams wurde abgeschlossen. Der Arbeitsprozess beschreibt eine unkomplizierte Möglichkeit des Entwickelns und betrachtet hierbei sowohl Aspekte des Projektmanagements einer Tool Entwicklung als auch technische Aspekte. Er wurde grundsätzlich auf kleine interne Teams ausgerichtet und versucht das Projektmanagement mit so wenig Overhead wie möglich umzusetzen. Somit wurde bewusst auf viel Bürokratie während des Entwicklungsprozesses verzichtet. Weiterhin wurden zwei weitere Game Engines und deren Workflows untersucht. Die Szenengraphen der jeweiligen Engines wurden so weit es möglich war analysiert und in ein Konzept für die Fusee Engine und das FuseeAT übersetzt. Das FuseeAT wurde während des Systemdesignes 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.

## 5.2 Integration des Systems in den weiteren Projektverlauf von Fusee

Aufgrund der im Abschnitt “Problematische Aspekte während der Umsetzung des Konzepts” aufgeführten Probleme während der Entwicklung in Cinema4D ist die Implementierung des FuseeAT nicht komplett abgeschlossen. Das Fusee Projekt an sich ist entsprechend seiner Natur als Simulations- und Entertainment Engine ein Projekt welches sich in ständiger Entwicklung befindet. Somit kann in Zukunft das FuseeAT für verschiedene Editoren und Tools genutzt werden um eine grafische Oberfläche für das Produzieren von Applikationen mit

der Fusee Engine zu schaffen.

### 5.2.1 Ergebnisse der Arbeit

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. Verschiedenste angeführte Quellen belegen, dass dieser Bereich der Projekte immer weiter strukturiert und ausgebaut wird. Hierbei sei noch einmal auf Sonys Studio SnSystems[1] 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 Entwicklung noch schneller zu gestalten wurde im Konzept zu FuseeAT besonders darauf geachtet, dass das FuseeAT Framework ungebunden von Editor Software zu gestalten. FuseeAT kann durch seinen Aufbau schnell erweitert werden. Cinema4D wiederum konnte aufgrund der gesetzten Ziele (Entwicklung des Frameworks in C#, Anbindung an Cinema4D 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 Cinema4D 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 Cinema4D Software integriert werden. Letztendlich wurden viele Grundsteine für die Entwicklung von und mit FuseeAT gelegt und ein kompletter Entwicklungsprozess für die Tool Entwicklung aufgezeigt. Eine Erweiterung der Implementierung findet in Zukunft sicherlich statt.

[1] http://snsystems.com/

# Anhang

# Literaturverzeichnis


Autodesk Inc. (2014-2015). Autodesk Scaleform. http://gameware.autodesk.com/scaleform.

Autodesk, Inc. (2015). 3DS Max. http://www.autodesk.de/products/3ds-max/overview.

Avid Technology. (2009). Alienbrain. http://www.alienbrain.com/. Avid Technology.

Beazley, David, F. (2014). Swig introduction documentation.

Beazley, David, S., Fulton, William. (1995-2015). SWIG-3.0. http://www.swig.org/.

Bethke, E. (2003). *Game Development and Production*. Wordware Publishing Inc.; Auflage: Pap/Cdr (Februar 2003).

Blender Foundation. (2014-2015). Blender 2.74. http://www.blender.org/.

Carter, B. (2004). *The Game Asset Pipeline*. Delmar Cengage Learning.

CCP Games. (2009). Eve online fanfest 2009 - scrum and agile development processes.

Chandler, H. (2006). *The Game Production Handbook*. Thomson Delmar Learning.

Cifaldi, F. (2005). E3 report: the path to creating aaa games.

Deck13. (2014). Lords of the Fallen. http://www.deck13.de/. City Interactive.

Digital Illusions CE. (2015). Frostbite Engine. http://www.frostbite.com/. Digital Illusions CE.

EPIC GAMES, INC. (2004-2015). Unreal Engine 4. http://www.unrealengine.com. EPIC Games INC.

Florian, Mehm, S. R., Christian,Reuter. (2014). Future trens in game authoring tools. S. 536-541.

Freeman, W. (2014). Kingdom come: deliverance video update 9.

Guerrilla Games. (2004, 2007, 2011, 2013, 2013). Killzone 1, Killzone 2, Killzone 3, Killzone: Mercenary, Killzone: Shadow Fall. http://www.guerrilla-games.com/. Sony Computer Entertainment.

Hirotaka, Takeuchi, I. (1986). The new product development game.

Klinge, Heiko, Chefredakteur, F. (2015). User-Generated Content in the V-Play Game Engine. *Making Games*, *3*, 24–27.

Lange, Thorsten, K. (2015). Lords of the Fallen Tackling a new gen game with an emerging studio. *Making Games*, *3*, 44–45.

Martin Klekner, D. V., Votja Nedved. (2014). Kingdom come: deliverance video update 9.

MAXON Computer GmbH. (2014-2015). Cinema 4D R16. http://www.maxon.net/de/products/cinema-4d-studio.html.

NaughtyDog. (2007, 2009, 2011). Uncharted. http://www.naughtydog.com/games/uncharted_drakes_fortune/. Sony Computer Entertainment.

Ieee guide–adoption of the project management institute (pmi(r)) standard a guide to the project management body of knowledge (pmbok(r) guide)–fourth edition. (2011 November). *IEEE Std 1490-2011*.

Quantic Dream. (2013). Beyond: Two Souls. http://www.quanticdream.com/en/. Sony Computer Entertainment.

Schmitz, C. (2013). The art of waiting.

Schmitz, C. (2014). Projektmanagement mit scrum. Available online - last checked 07.04.2015 http://www.makinggames.biz/features/projektmanagement-mit-scrum,2534.html.

Shotgun Software Inc. (2013). Shotgun. https://www.shotgunsoftware.com/. Shotgun Software Inc.

Tarja, Susi, P., Mikael, Johannesson. (2014). Serious games – an overview.

The Foundry Visionmongers Ltd. (2015). Modo. https://www.thefoundry.co.uk/products/modo/.

Unity Technologies. (2015). Unity 5. http://www.unity3d.com. Unity Technologies.

Wawro, A. (2014). Sony releases level editor that's open source and engine-agnostic.

Wihlidal, G. (2006). *Game Engine Toolset Development*. Thomson Course Technologies.

# Source Code Verzeichnis und Beispiele


4.1 Unreal Engine C++ Header Datei. Makros als Markup Befehle . 34
4.2 Unity GameObject Struktur . . . . . 39
4.3 Einfaches Cinema 4D Plugin in der Programmiersprache C++ . 54
4.4 Einbinden des Maxon Cinema 4D C++ Datentyps TagPlugin in die C# API von Uniplug . . . . . 59
4.5 Überschreiben der Message Funktion des TagData Datentyps . . 61
4.6 Funktion zum Generieren eines neuen Projekts in der Binary Version der Fusee Engine. . . . . 65
4.7 Funktion zum öffnen eines neuen Projekts in der Binary Version der Fusee Engine. . . . . 66
4.8 Teil des Structs zum Serialisieren des Fusee Projektes aus Cinem a 4D. Die Serialisierung speichert das Projekt auf der Festplatte als lesbare XML Datei. . . . . 66
4.9 Funktion zum Erstellen einer neuen Klasse im C# Projekt. . . . 67
4.10 ToolState System unterstützt beim verbessern der Stabilität des FuseeAT. . . . . 67
4.11 Code des ProjectState Enums. Wird in FuseeAT verwendet um die Integrität eines Projekts zu erhalten. . . . . 68

5.1 Beispiel zum Fusee Szenengraphen. Erstellt im Rahmen dieser Arbeit, während der Implementierung des Fusee Szenengraphen durch das Fusee Entwicklerteam und Herrn Prof. C. Müller. . . 76

Sourcecode Beispiele 5.1: 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;
```

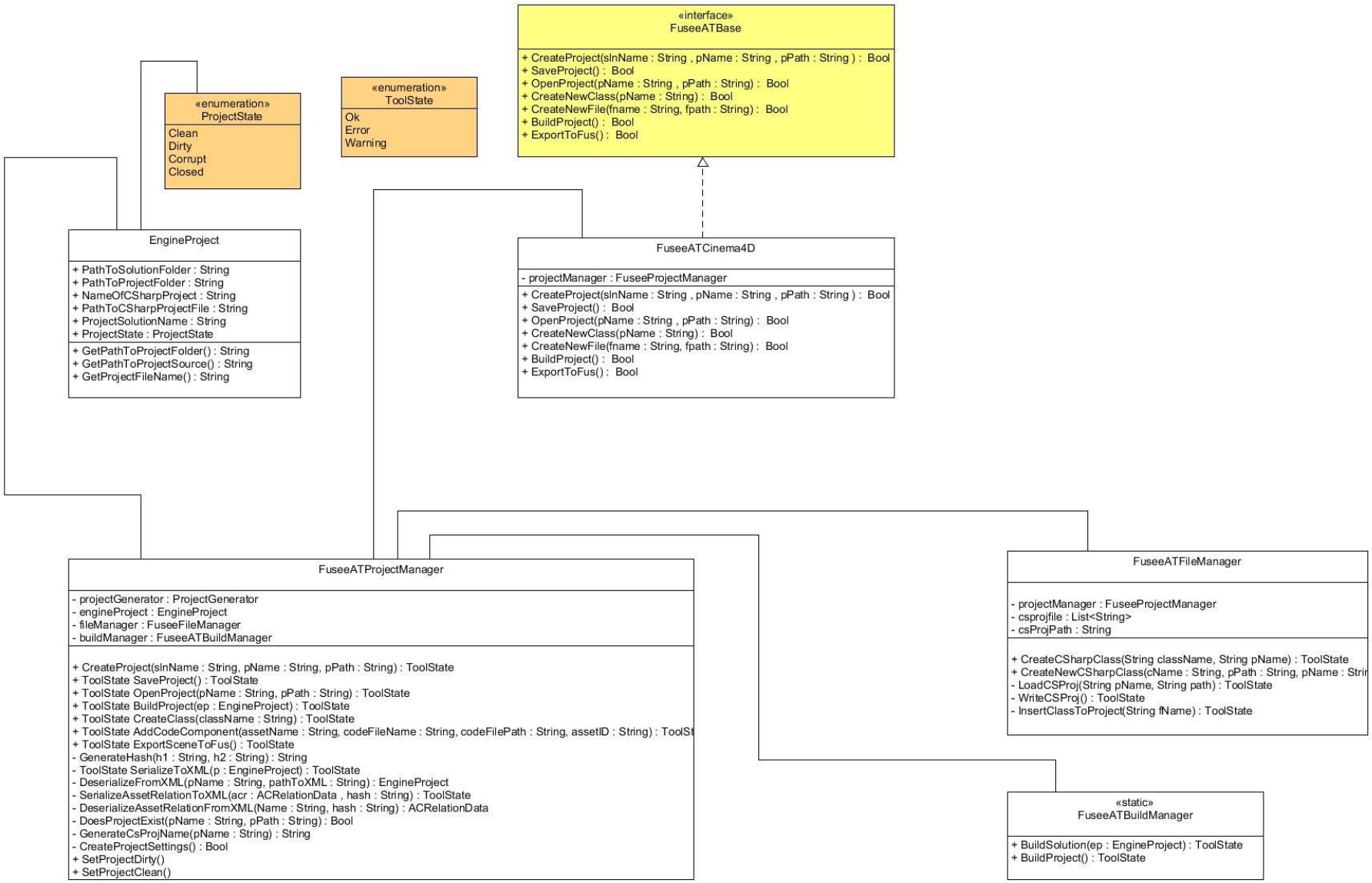
```
132             }
133         }
134 }
```

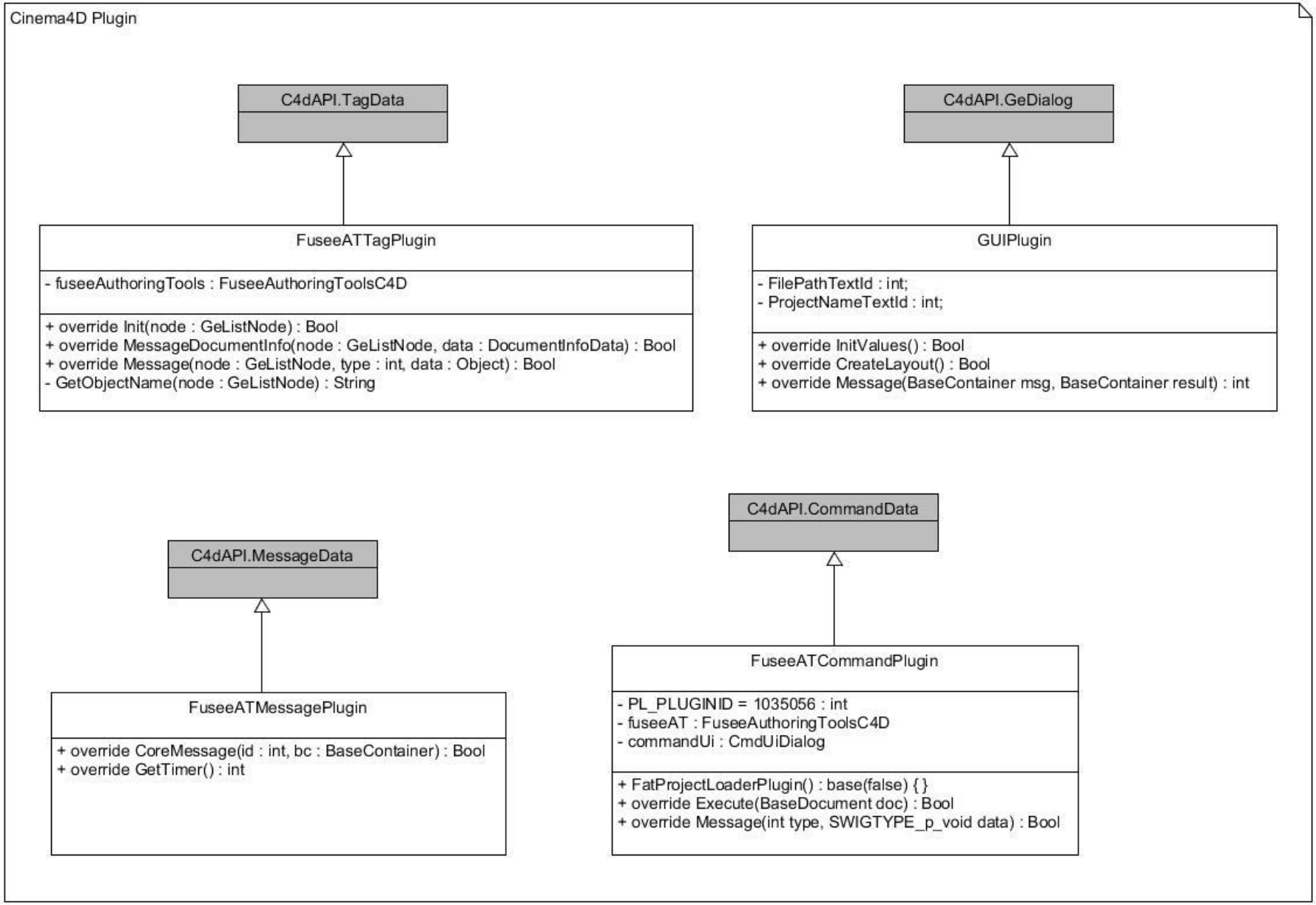
# Abbildungsverzeichnis

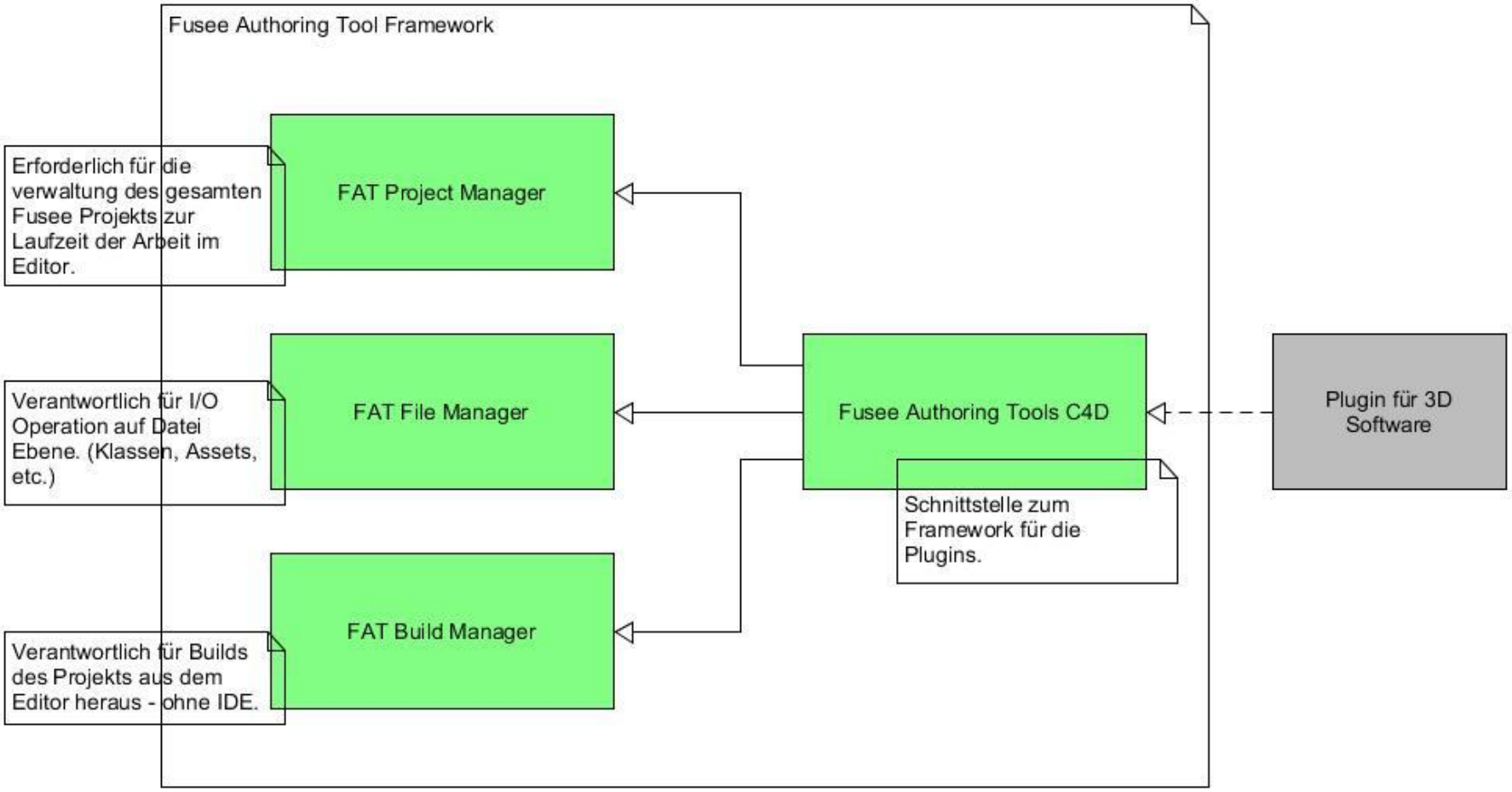

3.1 Überblick Stakeholder. Alle am Projekt beteiligten Personen oder Organisationen müssen beachtet werden. Grafik entnommen aus (»IEEE Guide–Adoption of the Project Management Institute (PMI(R)) Standard A Guide to the Project Management Body of Knowledge (PMBOK(R) Guide)–Fourth Edition«, 2011, S. 24). . . . . 18
3.2 Überblick über die Identifizierten Stakeholder und deren Inovelement in die Planung und Umsetzung des Tool Projektes. . . . 21

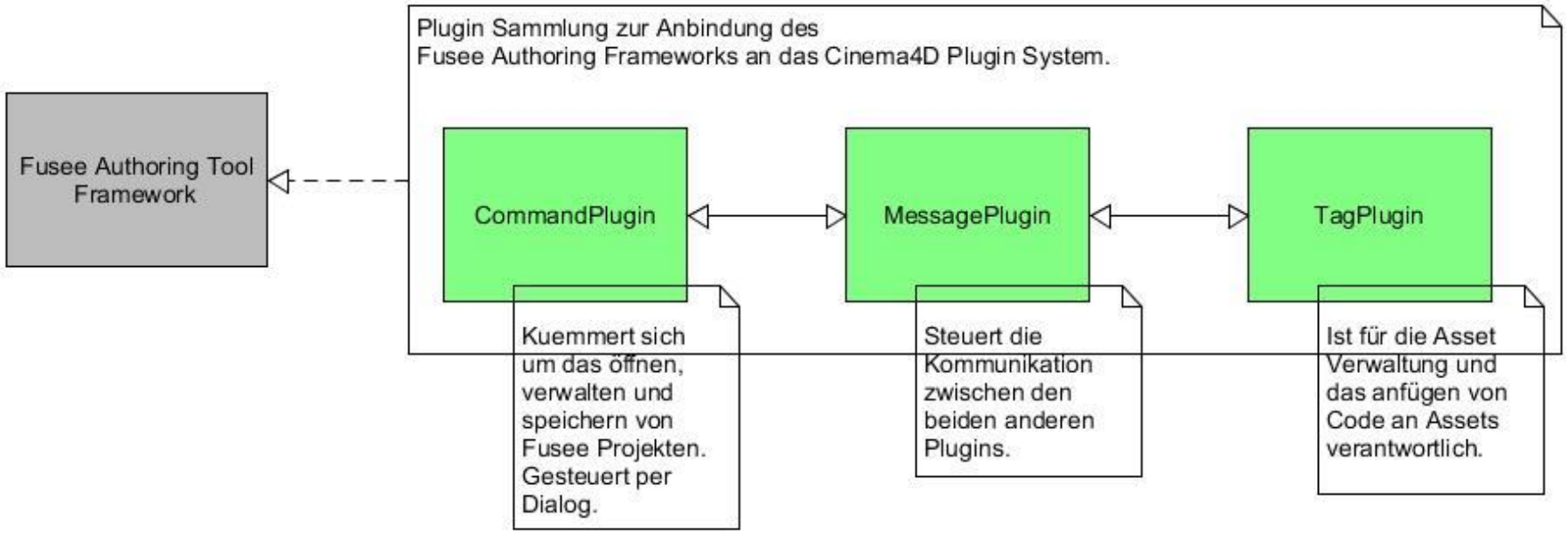
4.1 Überblick über die Use Cases eines Artist. . . . . 28
4.2 Überblick über die Use Cases eines Engineer. . . . . 30
4.3 Programmierung in der Unreal Engine 4 mit Hilfe des Blueprint Visual Programming Systems/Interfaces . . . . . 33
4.4 Ansicht des Contentbrowsers der Unreal Engine 4. Mit Suchfeld und Filterfunktionen, Assetvorschau ausklappbarer Ordnerstruktur. . . . . 35
4.5 Beispielhafte Szene im Szenengraph der Unreal Engine 4. Er gleicht dem Fusee und Cinema 4D Szenengraphen stark. . . . . 36
4.6 Ansicht des Contentbrowsers in Unity. Mit Suchfeld und Filterfunktionen und ausklappbarer Ordnerstruktur. . . . . 40
4.7 Beispielhafte Szene im Szenengraph von Unity. Er gleicht dem Fusee und Cinema 4D Szenengraphen stark. Das Detailpanel zeigt Components eines Nodes. . . . . 41
4.8 Überblick über die Systemarchitektur . . . . . 43
4.9 Überblick über das Fusee Authoring Toolkit Framework . . . . . 45
4.10 Klassendiagramm für das Fusee Authoring Toolkit . . . . . 46
4.11 Überblick über die Pluginarchitektur . . . . . 47
4.12 Klassendiagramm der Plugin Architektur . . . . . 48
4.13 Fusee Binary Projekt Struktur . . . . . 48

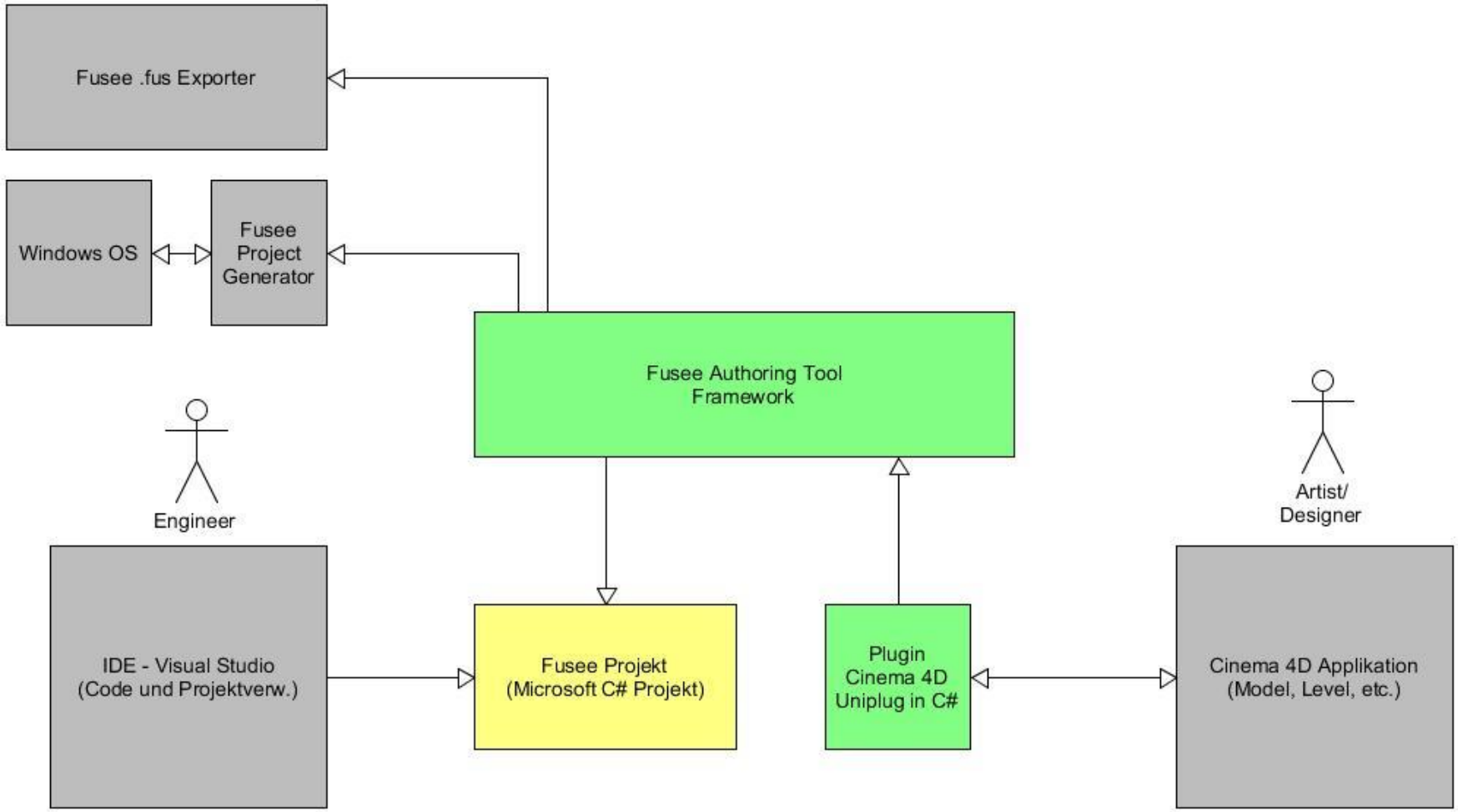
4.14 Asset Loop während der Produktion eines Projektes in Fusee und FuseeAT . . . . . 51
4.15 Schaubild des Cinema 4D API Systems . . . . . 54
4.16 Die Uniplug Projektstruktur in VisualStudio . . . . . 56
4.17 SWIG im Uniplug Projekt (ManagedPlugIn) markiert durch blaue Selektion. . . . . 57
4.18 Wrapping Vorgang der Cinema 4D API von C++ nach C# . . . 59
4.19 Der Fusee Szenengraph in exemplarischer bildlicher Darstellung. 63
4.20 Der Cinema 4D Szenengraph. Verschiedenste Objekte als "Nodes", daneben als Icons exemplarisch Tags verschiedener Tag-Datentypen am rechten Rand der Grafik. . . . . 64


«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.)

# SOFTWARE ENTWURF FUSEE AUTHORING TOOLS FÜR CINEMA 4D

Name: Fusee Authoring Tools

Autor: Master-Thesis an der Hochschule Furtwangen von Dominik Steffen WS14/15

Version: 0.2

## EINLEITUNG

Die Fusee Authoring Tools for Cinema 4D entstehen während einer Masterarbeit an der Hochschule Furtwangen. Das Projekt wird von Dominik Steffen (Medieninformatik) umgesetzt. Der Bearbeitungszeitraum erstreckt sich auf das Wintersemester 14/15.

Dieses Dokument beschreibt die Anforderungen an das Projekt Fusee Authoring Tools for Cinema 4D. Unter der Berücksichtigung einer Stakeholder Analyse unter Game Developern werden verschiedene Funktionen des Tools herausgearbeitet und beschrieben.

### UMFANG DER ARBEIT

Das Softwareprojekt wird als Plugin für Cinema 4D und als Bibliothek und Ergänzung des bereits bestehenden Uniplug Projektes im Rahmen des Fusee Game Engine Projektes entwickelt. Uniplug bietet bereits ein gewisses Maß an basis Funktionalität ium Plugins für Cinema 4D in .Net bzw. C# zu erstellen. Diese Funktionalität wird erweitert um dem Funktionsumfang der Anforderungen zu entsprechen.

### ERLÄUTERUNGEN ZU BEGRIFFEN UND / ODER ABKÜRZUNGEN

FUSEE / Fusee = Furtwangen University Simulation and Entertainment Engine

C4D = Cinema 4D, Software von Maxon

Fusee Authoring Tools for Cinema 4D = Plugin welches im Rahmen dieses Projektes entsteht

Fusee Authoring Tools = Bibliothek welche im Rahmen dieses Projektes entsteht

Uniplug = Bereits bestehendes Projekt welches das Erstellen Plugins mit Hilfe von C# für Cinema 4D erlaubt.

### VERWEISE UND QUELLEN

Maxon Developer Support: https://developers.maxon.net

Maxon Developer Forum: http://www.plugincafe.com/forum/default.asp

Fusee GitHub Projekt: https://github.com/FUSEEProjectTeam/Fusee

# ALLGEMEINES

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.

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 Open Source.

Hier könnte ein Vergleich zu anderen Game Engine Tools stehen.

## PRODUKTFUNKTIONEN

- Projekt anlegen
- Projekt bearbeiten (Durch hinzufügen von Assets oder Funktionalität)
- Projekt öffnen
- Projekt speichern
- Projekt bauen

## BENUTZERANALYSE

Bei den Stakeholdern des Projektes handelt es sich in erster Linie um Mitglieder des Teams welche durch die Entwicklung des Tools direkt betroffen sind.

Hierzu gehören in erster Mitarbeiter in Folgenden Positionen:

- Artist (Arbeit in 3D Modellierungssoftware)
    - Animator
    - World Builders
- Engineer (Software Entwickler, Programmierer)
    - Tools
    - Graphics
    - Network
    - AI
    - Sound
- Game Designer (Entwickeln und umsetzen von Spielideen)
    - Level
    - Scripter (Game Logic Scripts)
    - UI
- QA Tester (Testen des Spiels als Builds und in der Entwicklungsumgebung)

## 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.

### 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.

## ANFORDERUNGEN

### FUNKTIONALE ANFORDERUNGEN UND USE CASES

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

- Projekt anlegen
    - Visual Studio Solution Datei öffnen
        - Inhalt einlesen

      Visual Studio Solution Datei bearbeiten
        - Korrekte Stellen des einzufügenden Codes finden
        - Projekt anlegen und nötigen XML Code hinterlegen
    - Visual Studio Solution Datei speichern
- Projekt bearbeiten (Durch hinzufügen von Assets oder Funktionalität)
    - Neuen Code hinzufügen
        - Klassen anlegen
        - Klassen als Partial Classes anlegen
    - Neue Assets hinzufügen
        - Bilder
        - Modelle
        - Sound
- Projekt öffnen
    - Ein bestehendes Projekt in Cinema 4D öffnen
- Projekt speichern
    - Ein Projekt von Cinema 4D heraus speichern
    - Unnötige Dateien (Temporärer Natur) nach dem Speichern entfernen
- Projekt bauen
    - Ein Projekt von Cinema 4D heraus bauen

### 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.

## DESIGN CONSTRAINTS

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.