Ein Framework zur Erstellung von Planspielen zur Softwaretechnik

Alexander Nassal und Matthias Tichy, Universität Ulm
{alexander.nassal, matthias.tichy}@uni-ulm.de

Zusammenfassung

Die Entwicklung von Software ist ein komplexer Prozess, der von einer Vielzahl an Aspekten aus unterschiedlichen Disziplinen beeinflusst wird. Planspiele, die Softwareentwicklung und das dazugehörige Projektmanagement adäquat veranschaulichen, müssen diese Aspekte entsprechend berücksichtigen.
Eine der schwierigsten Aufgaben bei der Entwicklung solcher Planspiele ist es, den Projektverlauf korrekt zu simulieren. Bestehende Planspiele nutzen dafür speziell auf sie abgestimmte Simulationsmodelle, die nur schlecht für andere Planspiele wiederverwendet werden können. So muss bislang für jedes neue Planspiel auch ein neues Modell entwickelt werden.
Wir haben eine Repräsentation der Aspekte und Strukturen von Softwareprojekten entwickelt, die es erlaubt, von einem konkreten Projekt zu abstrahieren. Damit sind wir in der Lage, ein allgemeineres Simulationsmodell zu bauen, das für unterschiedliche Planspiele im Softwaretechnikbereich eingesetzt und angepasst werden kann.
Dieses Simulationsmodell haben wir zusammen mit weiteren Werkzeugen zu einem Framework zusammengefasst. In diesem Beitrag stellen wir dieses Framework vor, und zeigen anhand von Beispielen, wie es verwendet werden kann, um die Entwicklung von Planspielen zu unterstützen.

1 Einleitung

Planspiele sind ein bewährtes Mittel, Ausbildung praxisnah zu unterstützen. Das gilt auch für die Hochschullehre im Bereich Softwaretechnik, und dort besonders beim Thema Projektmanagement. Durch das spielerische Leiten eines simulierten Projekts können die Studierenden eigene Erfahrungen sammeln, und das zuvor erlernte theoretische Wissen anwenden. Dadurch wird ein tieferes Verständnis ermöglicht, da die Studierenden die Problematik selbst erkennen und erfahren, anstatt sie von außen erzählt zu bekommen (Reich, 2008). Planspiele haben den Vorteil, dass sie im Gegensatz zu richtigen Projekten deutlich weniger Zeit in Anspruch nehmen. Fehler, die im Planspiel gemacht werden, gefährden außerdem kein reales Projekt, in dem viele Arbeitsstunden von unterschiedlichen Entwicklern stecken, und ermöglichen es den Studierenden so, zu experimentieren und unterschiedliche Strategien auszuprobieren.

Um optimale Lerneffekte zu erzielen, müssen die eingesetzten Planspiele gut auf die eigentlichen Lehrinhalte abgestimmt werden, was häufig die Entwicklung eines neuen Spiels erforderlich macht. Eine solche Entwicklung ist oft mit erheblichem Aufwand verbunden. Das gilt vor allem für Spiele, die nicht nur einen konkreten Sachverhalt veranschaulichen, sondern ein allgemeines Gefühl für Softwareprojekte und die damit verbundenen Probleme und Herausforderungen vermitteln sollen.

Unserer Erfahrung nach liegt der Hauptaufwand in der Entwicklung des Simulationsmodells, das für die Berechnung des Projektverlaufs benötigt wird, und mit dem der Spieler interagieren muss. Dieses Modell muss so realitätsnah sein, dass die Erfahrungen die der Spieler macht, später auch auf reale Projekte übertragen werden können. Dazu müssen nicht nur die Aspekte der Softwareentwicklung berücksichtigt werden, sondern beispielsweise auch der Einfluss von psychologischen Faktoren wie Motivation, Lernverhalten und soziale Interaktion, oder arbeitswissenschaftliche Aspekte wie die Gestaltung von Arbeitszeit und Leistungsdruck.

Die Modelle in existierenden Planspielen sind speziell auf ihren jeweiligen Einsatz angepasst und decken genau die Aspekte ab, die im jeweiligen Planspiel thematisiert werden. Aspekte, die nicht direkt mit dem Lernziel in Verbindung stehen, werden ignoriert. Eine Wiederverwendung der Modelle in neuen Planspielen und mit anderen Lernzielen ist meist nicht sinnvoll, da der Anpassungsaufwand den Aufwand der Entwicklung eines neuen Modells überschreitet.

Um die Entwicklung neuer Planspiele besser zu unterstützen, haben wir ein Simulationsmodell entwickelt, das von einem konkreten Planspiel abstrahiert, und so einen flexibleren Einsatz erlaubt, als es mit den bisher in Planspielen verwendeten Modellen möglich ist.

Das Modell ist in ein Framework eingebettet, welches es ermöglicht, Spiele in C# zu entwickeln. Die Gestaltungsmöglichkeiten sind durch diesen Ansatz weniger eingeschränkt, als durch die Verwendung von Editoren und Generatoren zur Erstellung von Spielen. Unser Modell enthält vor allem die allgemeinen Aspek-

te der Projektarbeit, und kann entsprechend erweitert, angepasst oder verändert werden, um spezielles Spielverhalten zu erzeugen.

In diesem Beitrag stellen wir unseren Simulationsansatz und die gewählte Abstraktion vor, und zeigen, wie beides verwendet werden kann, um eigene Planspiele zu entwickeln. Nach einem kurzen Überblick über schon existierende Planspiele und Lösungen zur Unterstützung der Planspielentwicklung, erläutern wir unser Framework mit den dazugehörigen Werkzeugen. Anschließend schlagen wir eine konkrete Vorgehensweise vor, um damit neue Spiele zu erstellen, und verdeutlichen diese an einem Beispiel. In einem weiteren Abschnitt zeigen wir beispielhaft, wie konkrete Aspekte der Softwareentwicklung mit unserem Ansatz umgesetzt werden können. Zum Schluss bilden wir ein kurzes Fazit und diskutieren mögliche Erweiterungen und Verbesserungen, die bislang noch nicht umgesetzt wurden.

2 Verwandte Arbeiten

Neben Planspielen in anderen Bereichen, wie dem Finanzwesen, der Betriebswirtschaft oder dem Militär, existieren auch im Bereich der Softwareentwicklung schon einige Projekte, um die Lehre zu unterstützen.

Ein Beispiel dafür ist *SimVBSE* (Jain u. Boehm, 2006), das entwickelt wurde, um Studierenden die Möglichkeit zu bieten, eigene Erfahrungen auf dem Gebiet des Value-Based Software Engineering zu sammeln. Bei SimVBSE kann der Spieler mittels unterschiedlicher Aktionen den Projektverlauf eines fiktiven Softwareprojekts beeinflussen. Aktionen sind dabei häufig mit Kosten verbunden, die es zu berücksichtigen gilt. Der Projektverlauf wird auf Basis der gewählten Aktionen mittels eines Regelsystems simuliert, und dem Spieler grafisch und textuell angezeigt.

SimjavaSP (Ye u. a., 2007) ist ein interaktives, webbasiertes Planspiel, bei dem der Spieler ebenfalls die Rolle des Projektleiters einnimmt, um ein Softwareprojekt zum Erfolg zu führen. Dabei lernt er, Projekte nach dem Wasserfall- und Spiralmodell durchzuführen. Das Ziel des Spiels ist es, eine hypothetische Software in einem vorgegebenen Zeitrahmen, sowie in einer akzeptablen Qualität zu entwickeln, und dabei auf unterschiedliche außergewöhnliche Ereignisse, wie Mitarbeiterausfall oder geänderte Anforderungen, zu reagieren. SimjavaSP setzt bei der Simulation des Projektverlaufs auf einen ereignisorientierten Ansatz.

Ein drittes Beispiel ist das Projekt *The Incredible Manager* (de Oliveira Barros u. a., 2006). Dieses Spiel basiert auf System Dynamics Modellen. Dazu wurde der Ansatz der System Dynamics so erweitert, dass die Modelle während der Simulation durch den Spieler manipuliert werden können. Um Einfluss auf den Spielverlauf zu nehmen, verändert der Spieler die Parameter der Modelle, indem er diverse Entscheidungen bezüglich seines Entwicklerteams, des Projektplans und anderer projektrelevanter Aspekte trifft, mit dem Ziel eine hochwertige Software in möglichst kurzer Zeit zu entwickeln.

Neben den Projekten, die ein konkretes Planspiel zum Ziel haben, gibt es auch Arbeiten, die sich mit deren Entwicklung auf einer allgemeineren Ebene beschäftigen. Ein Beispiel dafür ist *SESAM* (Ludewig u. a., 1992). Hier wurde eine konzeptionelle Basis für die Planspielentwicklung geschaffen, und die Erstellung neuer Simulationsmodelle durch eine eigene Modellierungssprache unterstützt. SESAM enthält neben den Konzepten und einer dazu passenden Methodik, auch die dafür benötigten Werkzeuge, um neue Simulationsmodelle zu erstellen.

Ein weiteres Projekt, das die Entwicklung von Planspielen unterstützt, ist *SimSE* (Navarro, 2006). Obwohl dort die Untersuchung der didaktischen Wirksamkeit von Planspielen im Softwaretechnikbereich im Vordergrund stand, ist in diesem Projekt ein Baukasten entstanden, mit dem Planspiele entwickelt werden können. Dazu wird ein Editor bereitgestellt, mit dem sich Spielinhalte und Simulationsmodelle erstellen, und anschließend mittels eines Generators in ein interaktives Spiel transformieren lassen.

Alle uns bekannten Arbeiten liefern entweder ein fertiges Planspiel, oder geben Hilfestellung bei der Entwicklung eines eigenen Spiels. Keine der Arbeiten liefert ein einsatzbereites Simulationsmodell, das für den Einsatz in neuen Spielen konzipiert ist. Projekte wie SESAM liefern jedoch zumindest die notwendigen Grundlagen, um eigene Modelle zu konzipieren und umzusetzen.

Neben den Projekten zur Planspielentwicklung gibt es auch Arbeiten, die sich mit den Aspekten der Softwareentwicklung an sich beschäftigen und diese in Form von Modellen beschrieben haben. Ein bekanntes Beispiel sind die System Dynamics Modelle von Abdel-Hamid et. al. (Abdel-Hamid u. Madnick, 1991). In (Madachy, 2007) wurde die Arbeit von Abdel-Hamid weitergeführt und um Modelle ergänzt, welche die in den letzten Jahren hinzugekommenen Aspekte der Softwareentwicklung abdecken. Diese Modelle behandeln die Vorgänge allerdings auf einer sehr abstrakten Ebene, die für Planspiele oft ungeeignet ist. So wird beispielsweise nicht zwischen den einzelnen Mitarbeitern und deren Eigenschaften unterschieden, sondern werden diese lediglich gesammelt als Arbeitskraft betrachtet. Diese wird in Form der Produktivität als ein einzelner Wert dargestellt und fließt so in die Simulation ein. Außerdem sind die Prozesse und Strategien, nach denen das simulierte Projekt geführt wird, Teil des Modells; der Projektleiter wird quasi mitsimuliert. Daher sind diese Modelle für den Einsatz in Planspielen meist ungeeignet.

3 Framework und Simulationsumgebung

Im Folgenden beschreiben wir unseren Ansatz zur Strukturierung von Planspielen, der Abstraktion von

konkreten Projekten, und den grundlegenden Simulationsansatz den wir für das auf diesen Konzepten basierende Simulationsmodell gewählt haben. Unser Ziel war es, eine Repräsentation von Softwareprojekten zu finden, die unabhängig von konkreten Prozessen, Produkten und weiteren planspielspezifischen Eigenschaften ist. Gleichzeitig muss es möglich sein, auf Basis dieser Struktur den Projektverlauf zu berechnen und diesen auf ein konkretes Planspiel zu übertragen.

Wir schlagen vor, Planspiele, wie in (Nassal, 2014) beschrieben, als vier aufeinander aufbauende Schichten zu entwickeln:

1. Die unterste Schicht enthält mit der *Simulationsengine* alle zur Simulationsausführung benötigten Funktionen. Sie kümmert sich um den grundsätzlichen Ablauf der Simulationsberechnung, und stellt weitere hilfreiche Funktionen, wie die Überwachung und Protokollierung des Simulationsverlaufs, zur Verfügung.
2. Darauf aufbauend definiert das *Simulationsmodell* mit seiner statischen Struktur das Datenmodell, welches dem Spiel zugrunde liegt, und die operationale Semantik, die vorgibt, wie sich die so abgelegten Daten über die Zeit verändern.
3. Mittels eines *Szenarios*, wird auf Basis der im Simulationsmodell beschriebenen statischen Struktur, ein konkretes Projekt erstellt und somit das allgemeine Modell an das konkrete Planspiel angepasst. Die so definierten Daten werden als Startzustand der Simulation verwendet. Dazu werden die im Datenmodell festgelegten Elemente instanziiert, deren Parameter auf konkrete Werte festgelegt, und die Beziehungen zwischen ihnen definiert.
   Zum Szenario gehört außerdem die Storyline des Spiels. Neben dem Projekt selbst, das die Rahmenbedingungen und die Ausgangssituation des Spiels beschreibt, können hier Ereignisse im Projektverlauf definiert werden, die das Projekt zusätzlich beeinflussen.
4. Die oberste Schicht enthält die *Benutzerschnittstelle*. Dazu gehören vor allem die Repräsentation der Simulation, die es dem Spieler erlaubt das Spielgeschehen zu beobachten, und die Interaktionsmöglichkeiten, die der Spieler hat, um das Spielgeschehen zu beeinflussen.

Da durch die Architektur klare Schnittstellen definiert sind, lassen sich die Teile relativ unabhängig voneinander entwickeln, warten, erweitern und wiederverwenden. Insbesondere existiert eine klare Trennung zwischen allgemeinem Simulationsmodell und konkretem Szenario bzw. Planspiel.

Um einen flexiblen Einsatz zu ermöglichen, haben wir darauf verzichtet, eine eigene Simulationssprache, spezielle Editoren oder andere Entwicklungsumgebungen für Planspiele zu erstellen. Stattdessen wurden Ausführungsumgebung und Simulationsmodell als reine C# Bibliotheken konzipiert. Das ermöglicht es dem Spielentwickler zum einen, die darauf aufbauenden Schichten auch algorithmisch zu erzeugen, und so beispielsweise Szenarien zu generieren, und zum anderen, Aspekte, die in unserem Modell nicht vorgesehen sind, leichter ergänzen zu können. Außerdem schränkt unser Ansatz die weitere Gestaltung der Spiele nur minimal ein. So können zum Beispiel auch Mehrspielerkonzepte und verteilte Spiele umgesetzt werden.

3.1 Datenmodell

Das Simulationsmodell besteht aus zwei Teilen: einem *Datenmodell*, welches die statische Struktur beschreibt, und einem *Regelwerk*, über das die operationale Semantik definiert ist und mit dem sich der Verlauf eines auf Basis des Datenmodells definierten Projekts berechnen lässt.

Eines der Ziele bei der Entwicklung des Frameworks war es, dem Entwickler die Verwendung des Simulationsmodells zu ermöglichen, ohne dass er das umfangreiche und teilweise komplexe Regelwerks im Detail verstehen muss. Aus diesem Grund, und weil das Regelwerk den hier gegebenen Rahmen deutlich überschreiten würde, verzichten wir hier auf dessen Beschreibung und beschränken uns auf das Datenmodell.

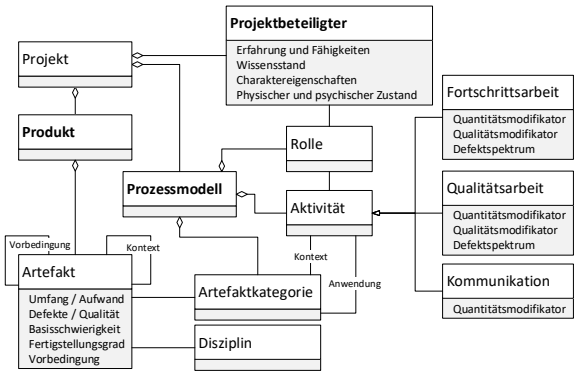

Abbildung 1: Datenmodell

Abbildung 1 zeigt ein UML-Klassendiagramm des Datenmodells. Auf Basis dieses Modells werden Szenarien modelliert, die anschließend simuliert werden können. Ein Szenario besteht im Wesentlichen aus *Produkt*, *Team* und *Prozessmodell*.

Produkt Das Produkt beschreibt die Software, die der Spieler entwickeln soll. Modelliert wird dazu das Endprodukt, bestehend aus den verschiedenen *Artefakten*, in die es sich unterteilen lässt. Die Granularität ist dabei dem Planspielentwickler überlassen und hängt stark vom Spieldesign und Lernziel des Spiels ab. Es kann beispielsweise jede einzelne Anforderung als ein eigenständiges Artefakt modelliert werden, es besteht aber auch die Möglichkeit, nur ein Artefakt für das gesamte Pflichtenheft zu erstellen.

Jedes Artefakt verfügt über verschiedene Parameter, die unter anderem dessen Umfang und Schwie-

rigkeit festlegen. Artefakte können für sie relevanten *Disziplinen* zugeordnet werden, die zusätzliche Anforderungen an die bearbeitenden Entwickler stellen. Beispielsweise kann es von Bedeutung sein, dass ein Entwickler neben der Fähigkeit grundsätzlich Anforderungen zu erheben, auch in der Lage sein muss, diese als UML-Modell zu dokumentieren. Der Umgang mit UML ist somit eine hierfür relevante Disziplin.

Artefakte können untereinander in Beziehung stehen, um die Abhängigkeiten zwischen ihnen auszudrücken. Diese werden beispielsweise als harte Vorbedingungen beschrieben, um auszudrücken, dass ein Artefakt erst bearbeitet werden kann, wenn ein anderes abgeschlossen wurde, oder sich in einem bestimmten Zustand befindet. Artefakte können mittels Beziehungen auch als Kontext eines anderen Artefakts festgelegt werden, sodass Vollständigkeit und Qualität des einen Artefakts, die Fortschrittsgeschwindigkeit und Qualität bei der Bearbeitung des anderen Artefakts beeinflussen.

Team Das Team ist die Menge der am Projekt beteiligten Personen. Dazu gehört neben den Entwicklern beispielsweise auch der Kunde. Der Projektleiter wird typischerweise nicht modelliert; diese Rolle übernimmt später der Spieler.

Jeder Projektbeteiligte enthält eine Vielzahl an Parametern, die unter anderem seine Fähigkeiten in Bezug auf die vorhandenen Aktivitäten und Disziplinen, sein Wissen über das Produkt, Charaktereigenschaften und seinen physischen sowie psychischen Zustand beschreiben. Durch die Belegung der Parameter kann gesteuert werden, wie sich das Teammitglied im Team verhält, und wie sich seine Arbeit auf das Projekt auswirkt (Nassal u. Tichy, 2016). Zu den Charaktereigenschaften gehören unter anderem, wie der Mitarbeiter mit Zeitdruck umgeht, welche Aktivitäten er gerne durchführt, mit welchen Kollegen er gerne zusammenarbeitet, was ihn motiviert, und nach welchen Kriterien er sein Arbeitsumfeld einschätzt.

Prozessmodell Mit dem Prozessmodell werden die Aktivitäten festgelegt, die die einzelnen Teammitglieder durchführen können, um das zuvor definierte Produkt zu entwickeln. Dazu stehen drei Basisaktivitäten zur Definition weiterer Aktivitäten zur Verfügung:

- Aktivitäten, die auf *Fortschrittsarbeit* basieren, haben das Ziel, die Artefakte des Produkts weiter zu vervollständigen. Darunter fällt beispielsweise die Codierung eines zuvor definierten Moduls.
- Mittels Aktivitäten der *Qualitätsarbeit*, wie beispielsweise dem Testen, lassen sich Defekte finden, die über Fortschrittsarbeit in die Artefakte gekommen sind.
- *Kommunikationsaktivitäten* dienen zum Austausch von Wissen zwischen den Teammitgliedern.

Aktivitäten können über Rollen, die die Teammitglieder einnehmen, auf diese beschränkt werden. So kann beispielsweise verhindert werden, dass der Kunde an der Implementierung teilnimmt. Jedes Artefakt wird außerdem einem Artefakttyp zugeordnet. Aktivitäten können auf einzelne Artefakttypen eingeschränkt werden, um beispielsweise eine Anforderungsanalyse auf dem Entwurf zu verbieten.

Ein Prozessmodell legt normalerweise neben den Aktivitäten und Rollen auch fest, wie diese anzuwenden sind. Dieser Teil des Prozessmodells wird nicht modelliert, da er vom Spieler, in seiner Rolle als Projektleiter, umgesetzt werden muss.

3.2 Ausführungsumgebung

Die Simulationsausführung berechnet den Projektverlauf anhand der Regeln des Simulationsmodells. Wir haben dafür den Ansatz eines Multiagentensystems gewählt, der schon erfolgreich in anderen Arbeiten zur Simulation von Softwareentwicklungsprozessen eingesetzt wurde (Cherif u. Davidsson, 2010; Wickenberg u. Davidsson, 2002). Dieser Ansatz hat den Vorteil, dass sich mit der ihm zugrundeliegenden Struktur die Vorgänge aus der Realität sehr direkt nachbilden lassen.

Das System setzt sich aus Elementen dreier Klassen zusammen:

- Die *Umgebung* repräsentiert die Grenzen der Simulation und deren Schnittstelle zur Außenwelt. Sie definiert die benötigten Rahmenparameter und abstrahiert Aspekte, die nicht Teil der Simulation sind, diese jedoch beeinflussen. Sie enthält außerdem die Agenten und Ressourcen.
- *Ressourcen* sind die passiven Elemente der Simulation. Sie sind Teil des Simulationszustands und können von den Agenten verändert werden. Alle Artefakte des Produkts sind als solche Ressourcen realisiert, aber auch Werkzeuge, Arbeitsgegenstände oder der Projektplan können so umgesetzt werden.
- Die *Agenten* sind der einzige aktive Teil der Simulation. Agenten agieren zunächst unabhängig voneinander. Das Gesamtverhalten der Simulation ergibt sich allein aus dem Zusammenspiel der einzelnen Agenten. Jeder Projektbeteiligte wird als ein eigenständiger Agent umgesetzt. Wie auch in der Realität, ergibt sich so die Leistung des Teams aus den Einzelleistungen der Projektbeteiligten. Diese werden nicht zentral gesteuert, sondern agieren selbstständig und koordinieren sich über die Kommunikation untereinander und die Wahrnehmung des Projektzustands, zu dem beispielsweise auch der Projektplan gehört.

Die Simulationsberechnung erfolgt in diskreten Zeitschritten. Die Schrittgröße ist dabei zunächst variabel, was auch durch das Simulationsmodell unterstützt wird. Die benötigte Rechenzeit steigt linear mit der Anzahl der Zeitschritte. Es hat sich herausgestellt, dass eine Schrittgröße von einer Stunde ein guter Kompromiss zwischen Performance und Genauigkeit der Simulationsergebnisse darstellt. Mit dieser Schrittgröße lässt sich ein vollständiges Projekt in wenigen Minu-

ten berechnen. Bei längeren Schrittgrößen reagieren die simulierten Mitarbeiter in manchen Fällen nicht schnell genug auf veränderte Situationen. So kann es beispielsweise sein, dass eine Aufgabe deutlich weniger Zeit in Anspruch nimmt als durch den Zeitschritt zur Verfügung steht, und der Mitarbeiter so viel Zeit ungenutzt verstreichen lässt, bevor er im nächsten Zeitschritt eine andere Aufgabe bearbeiten kann.

3.3 Werkzeugunterstützung

Wir haben mehrere Werkzeuge entwickelt, um die Verwendung, Anpassung und Weiterentwicklung des Simulationsmodells zu unterstützen.

Dateiformat für Szenarien Um Szenarien leichter handhaben zu können, wird ein XML-Format und ein dazugehöriges Modul für das Speichern und Laden dieses Formats bereitgestellt. Standardszenarien können damit ohne Programmiertätigkeit einfach erstellt und bearbeitet werden.

Grafische Oberfläche Mittels einer grafischen Oberfläche können Szenarien geladen, und die Simulation ausgeführt und überwacht werden. Abbildung 2 zeigt einen Ausschnitt der Oberfläche.

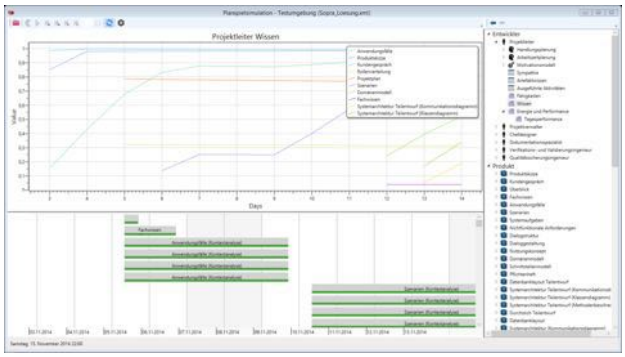

Abbildung 2: Grafische Oberfläche zur Simulationsbeobachtung

Die Oberfläche ist in drei Bereiche gegliedert. Auf der rechten Seite werden die Elemente des aktuell geladenen Szenarios angezeigt. Durch Auswahl der einzelnen Elemente werden im linken oberen Bereich entsprechende Darstellungen und Interaktionsmöglichkeiten bereitgestellt. Unter anderem wird so für die meisten Parameter, wie beispielsweise Artefaktfortschritt und -qualität oder Motivation und Wissensstand der Mitarbeiter, deren Verlauf über die Simulationszeit in Form von Liniendiagrammen dargestellt. So lassen sich Trends, Muster und markante Veränderungen leichter erkennen und nachvollziehen. Im linken unteren Bereich wird der aktuelle Projektplan in Form eines Gantt-Diagramms angezeigt. Dieser Plan kann jederzeit verändert werden, um die Auswirkungen auf die Simulation beobachten zu können.

Testunterstützung Um sicherzustellen, dass ein Szenario auch die gewünschten Aspekte aufweist, stellen wir ein Framework bereit, mit dem sich automatisierbare Tests entwickeln lassen, die unter anderem statistisch ausgewertet werden können. So lassen sich eine Vielzahl an Konfigurationen und unterschiedlichen Projektverläufen, trotz großer Parameterzahl und vielen Freiheitsgraden, überprüfen. Dazu werden mehrere Simulationsläufe durchgeführt, und dabei diejenigen Parameter zufällig belegt und verändert, die der Spieler während des Projektverlaufs beeinflussen kann. Jeder Versuchslauf entspricht so einem möglichen Projektverlauf, wie er später im Planspiel bei entsprechender Spielereingabe auftreten kann. Die Versuchsläufe können mittels Regeln, die im Framework enthalten sind, oder selbst definiert werden können, einzeln oder in ihrer Gesamtheit überprüft und ausgewertet werden.

4 Vorgehen bei der Erstellung eines neuen Planspiels

In diesem Abschnitt beschreiben wir das grundsätzliche Vorgehen bei der Erstellung eines neuen Planspiels mit unserem Framework. Dazu schlagen wir ein Vorgehen in 10 Schritten vor:

1. Auswahl und Formulierung des Lernziels.
2. Auswahl des Rahmenszenarios: Welches Produkt soll mit welchem Team und welchem Vorgehen unter welchen Bedingungen entwickelt werden?
3. Definition des konkreten Szenarios mit Produkt, Team und Aktivitäten.
4. Auswahl der nicht benötigten Aspekte des Modells und Deaktivierung der entsprechenden Modellteile.
5. Zusätzliche Definitionen speziellen Verhaltens oder spezieller Eigenschaften, die nicht im Framework vorgesehen sind.
6. Definition der Storyline und zusätzlicher externer Ereignisse.
7. Entwicklung von expliziten Testfällen und statistischen Tests zur Validierung des Modellverhaltens in Bezug auf die Lernziele.
8. Design und Implementierung der Rahmenanwendung und anschließende Integration von Modell und Ausführungsumgebung.
9. Anreicherung des Spiels um zusätzliches Spielmaterial, wie Storyelemente, Erklärungen, oder Hilfestellungen.
10. Probelauf mit kleinen Testgruppen, ggf. mit Evaluation, und entsprechende Anpassungen der vorherigen Schritte auf Basis der Ergebnisse.

Im Folgenden werden die einzelnen Punkte anhand des in (Nassal, 2015) beschriebenen Projekts näher erläutert. Dabei handelt es sich um ein Spiel, das wir entwickelt haben, um den Studierenden die Planungsaspekte des Softwaregrundpraktikums spielerisch näher zu bringen. In diesem Praktikum entwickeln die Studierenden über zwei Semester hinweg ihr erstes vollständiges Softwaresystem anhand einer realen Problemstellung. Dabei werden neben den technischen Aspekten auch Projektmanagement und Teamarbeit gelehrt. Um das zu unterstützen, haben wir mit

dem hier beschriebenen Framework Szenario und Simulation erstellt, und in *Microsoft Project* (Microsoft Corporation, 2013) integriert. Mit dem entstandenen Spiel können die Studierenden erste praktische Erfahrungen im Projektmanagement gewinnen, und dabei gleichzeitig den Umgang mit den dazu benötigten Werkzeugen üben.

Lernziel Im ersten Schritt werden die Lernziele definiert, die mit dem Spiel erreicht werden sollen. Je genauer die Ziele beschrieben werden, umso einfacher ist später deren Überprüfung.

Für unser Beispielprojekt haben wir folgende Ziele festgelegt:

- Die Studierenden lernen die Artefakte kennen, die sie in ihrem Praktikum erstellen müssen. Sie können deren Bedeutung für das Projekt, den Zusammenhang mit anderen Artefakten, und deren Umfang und den damit verbundenen Erstellungsaufwand einschätzen.
- Die Studierenden lernen das verwendete Prozessmodell kennen. Sie kennen anschließend dessen Aktivitäten, und wissen wann und wofür diese eingesetzt werden.
- Die Studierenden lernen, dass die gegebene Zeit nur ausreicht, wenn sie die Aufgaben sinnvoll auf die einzelnen Teammitglieder verteilen und parallel arbeiten.
- Die Studierenden lernen, dass kontinuierliche Qualitätssicherung notwendig ist, um gute Projektergebnisse zu erzielen.
- Die Studierenden lernen mit Microsoft Project einfache Projektpläne zu erstellen und zu pflegen.

Rahmenszenario Das Rahmenszenario ist durch die Vorgaben des Projekts, das die Studierenden im Rahmen des Praktikums durchführen müssen, in den meisten Punkten vorgegeben. Folgende Aspekte wurden für das Szenario festgelegt:

- Die Projektlaufzeit beträgt neun Monate und geht von Anfang November bis Ende Juli.
- Das Entwicklerteam besteht aus sechs Mitarbeitern, die verschiedene Rollen einnehmen und für die unterschiedlichen Gebiete der Softwaretechnik zuständig sind.
- In der Simulation wird ein abstraktes Produkt entwickelt, das unabhängig von einem konkreten Einsatzzweck ist. So können die Studierenden die verschiedenen Artefakte kennen lernen, die grundsätzlich bei der Softwareentwicklung vorkommen. Es ist zum Beispiel unerheblich, was die einzelnen Module der Software enthalten; wichtig ist nur, dass Module implementiert werden müssen.
- Da wir im Praktikum die an das Wasserfallmodell angelehnte Fusion-Methode (Coleman u. a., 1994) verwenden, sind die Aktivitäten und deren Reihenfolge durch diese vorgegeben.
- Weitere Managementaspekte, wie beispielsweise das Budget, werden nicht berücksichtigt, da sie im Praktikum keine Rolle spielen.

Produkt, Team und Aktivitäten Unser Produkt besteht aus 64 einzelnen Artefakten, die jeweils einem von 8 Artefakttypen zugeordnet sind. Diese sind:

- *Vorgabedokument* für Dokumente, welche das Projektteam vor Projektbeginn ausgehändigt bekommt.
- *Anforderungsdokument* für Artefakte, die während der Anforderungsanalyse entstehen.
- *UI-Spezifikation* für Artefakte, die die Benutzerschnittstelle beschreiben.
- *Entwurfsdokument* für Artefakte, die den Architekturentwurf enthalten.
- *Implementierung* für die zu codierenden Artefakte.
- *Qualitätssicherung* für alle Tests und Reviews.
- *Dokumentation* für die zusätzlich zu erstellenden Dokumente, wie Handbuch und Installationsanleitung.
- *Controlling* für die Dokumente, die für das Projektmanagement erstellt werden müssen, wie beispielsweise der Arbeitszeitnachweis.

Die unterschiedlichen Artefakte stehen untereinander in Beziehung. So ist es beispielsweise in unserem Szenario nicht möglich, etwas zu entwerfen, für das die Anforderungen nicht bekannt sind, und etwas zu implementieren, für das der Entwurf fehlt. Damit stellen wir sicher, dass das Prozessmodell im Wesentlichen eingehalten wird. Für Einflüsse zwischen den Artefakten wurden Kontextbeziehungen festgelegt, die bestimmen, wie sich die Qualität und Vollständigkeit der Kontextartefakte auf die Entwicklung anderer Artefakte auswirken.

Abbildung 3 zeigt die Beziehungen für die Artefakte der Anforderungsanalyse. Rechtecke stehen dabei für die Artefakte, durchgezogene Pfeile zeigen deren vorgegebene Bearbeitungsreihenfolge an. Gepunktete Pfeile zeigen auf diejenigen Artefakte, die die Bearbeitung des betrachteten Artefakts durch ihren Zustand beeinflussen können. Das Artefakt *Produktskizze und Kundengespräch* stellt den Ausgangspunkt der Anforderungsanalyse dar. Auf seiner Basis werden die einzelnen Aspekte des Pflichtenhefts erarbeitet, und am Ende zu einem vollständigen Dokument in Form des Artefakts *Pflichtenheft* zusammengesetzt.

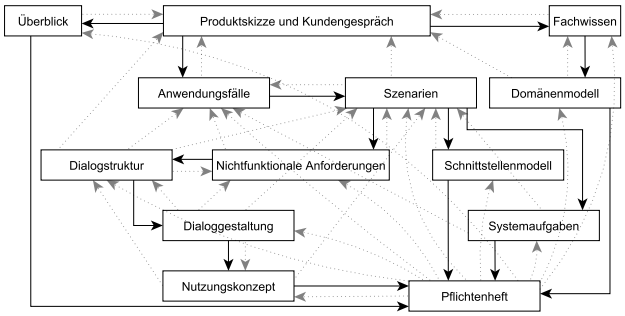

Abbildung 3: Beziehungen der Artefakte der Anforderungsanalyse

Die Umfänge der Artefakte wurden so gewählt, dass die dafür benötigte Zeit im Spiel dem Aufwand entspricht, den wir im echten Projekt dafür vorgesehen haben. Da die Studierenden nicht Vollzeit am Projekt arbeiten, die Mitarbeiter im Spiel aber schon, haben wir die Umfänge entsprechend dahingehend angepasst.

Einige Artefakte haben wir zusätzlich mit einer oder mehreren der Disziplinen *UML*, *Gestaltung*, *Relationale Datenbanken* und *Objektorientierte Programmierung* verknüpft, um besser zu verdeutlichen, wo die entsprechenden Schwerpunkte bei der jeweiligen Artefaktentwicklung liegen. So werden im realen Projekt beispielsweise Anwendungsfälle durch UML-Diagramme dokumentiert. Passend dazu wird im Spiel dem Artefakt *Anwendungsfälle* die Disziplin *UML* zugeordnet.

Im tatsächlichen Projekt besteht das Team aus sechs Mitgliedern, die die Rollen *Projektleiter*, *Projektverwalter*, *Architekt*, *Dokumentator*, *Qualitätssicherungsingenieur* und *Verifikations- und Validierungsingenieur* schwerpunktmäßig einnehmen. Entsprechend wurden im Spiel sechs Projektbeteiligte erstellt, die nach diesen Rollen benannt wurden, und deren Fähigkeiten dazu passen. Die Rolle des Projektleiters wurde nicht vergeben, da diese durch den Spieler eingenommen wird.

Abbildung 4 zeigt als Beispiel die Verteilung der Fähigkeiten des Architekten. Die ersten vier Balken beschreiben seine Fähigkeiten in Bezug auf die Disziplinen, die anderen acht beziehen sich auf die einzelnen Aktivitäten. Fähigkeiten beeinflussen die Geschwindigkeit und Qualität, mit der ein Projektbeteiligter ein entsprechendes Artefakt bearbeiten kann. Hat ein Mitarbeiter einen Wert von 0 für eine Fähigkeit, so besitzt er keinerlei Wissen, Erfahrung und Talent auf diesem Gebiet. Ein Wert von 1 hingegen steht für optimale Fähigkeiten auf dem Gebiet, die nicht weiter gesteigert werden können. Man kann sehen, dass insbesondere die für das Erstellen des Architekturentwurfs notwendigen Fähigkeiten *Entwurf*, *Objektorientierte Programmierung* und *UML* beim Architekten besonders hoch sind, während er Schwächen im Bereich *Dokumentation*, *Projektmanagement* und *Qualitätssicherung* hat.

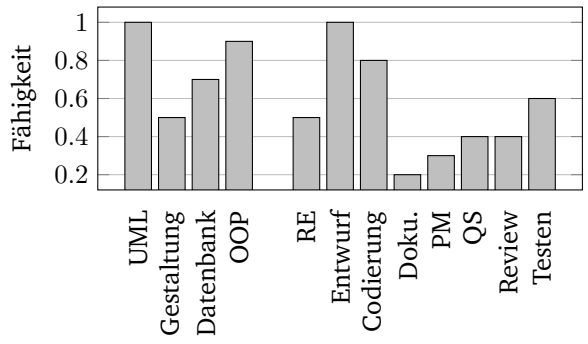

Abbildung 4: Fähigkeitenprofil des Architekten

Da sich der Spieler auf die Lernziele konzentrieren soll, wurden für alle anderen Parameter der Mitarbeiter die Standardwerte belassen. Die Mitarbeiter entsprechen somit durchschnittlichen Mitarbeitern und der Spieler muss sich nicht mit deren besonderen Eigenschaften auseinandersetzen.

Zusätzliche Anpassungen Das Simulationsmodell deckt alle Aspekte ab, die für unser Planspiel notwendig sind. Daher wurde kein weiteres spezielles Verhalten und keine besonderen Eigenschaften definiert. In unserem Spiel setzen wir das vollständige Modell mit allen Aspekten ein. Für manche Zwecke kann es jedoch sinnvoll sein, bestimmte Aspekte des Modells zu deaktivieren, um die Vorgänge einfacher zu gestalten. Bestimmte Zusammenhänge, wie beispielsweise der Einfluss von Fähigkeiten auf die Produktivität, können so besser beobachtet werden, da sie nicht von anderen Effekten, wie beispielsweise dem Einfluss von Motivation auf die Produktivität, überlagert werden. Welche Modellteile deaktiviert werden sollen, hängt von den jeweiligen Lernzielen ab.

Für das Spiel wurden keine zusätzlichen Ereignisse, wie beispielsweise der Ausfall eines Mitarbeiters hinzugefügt, da sich die Spieler auf den grundsätzlichen Ablauf konzentrieren sollen. Es wurde jedoch für alle Mitarbeiter zwei Wochen Urlaub an Weihnachten festgelegt, da auch im Praktikum in diesem Zeitraum keine Projektarbeit der Studierenden vorgesehen ist.

Validierung Das Framework enthält eine Vielzahl an Basistests, die die unterschiedlichen Eigenschaften unseres Modells zeigen. Diese Tests können verwendet werden, um sicherzustellen, dass auch das erstellte Szenario diese Eigenschaften erfüllt. Tests sind besonders wichtig, wenn das Modell geändert, erweitert, oder einzelne Aspekte davon deaktiviert wurden.

Unser Szenario kann mit mehreren zusätzlichen Tests validiert werden. Da die Parameter des Szenarios vom Spieler nicht verändert werden können, werden diese dabei als konstant betrachtet und nicht verändert. Da automatische Tests ohne einen Spieler auskommen müssen, wird dieser durch zusätzlichen Agenten ersetzt, die in die Simulation eingefügt werden und die Interaktion des Spielers mit der Simulation ersetzen. Dabei setzen die Agenten unterschiedliche Strategien um, um zu testen, wie sich diese auf den Projektverlauf und das Projektergebnis auswirken.

In unserem Beispielprojekt ist es sinnvoll, bezogen auf die Lernziele, vor allem die folgenden Strategien in unterschiedlichen Kombinationen zu testen:

- Die Reihenfolge, in der die Artefakte bearbeitet werden, entspricht der Reihenfolge, die im tatsächlichen Projekt vorgesehen ist.
- Die Reihenfolge der Artefakte wird beliebig gewählt. Es wird dabei darauf geachtet, dass die jeweiligen Vorbedingungen erfüllt sind.
- Die Reihenfolge der Artefakte wird rein zufällig gewählt.

- Die Mitarbeiter werden entsprechend ihrer Rolle den Aktivitäten zugeordnet.
- Die Mitarbeiter werden zufällig den Aktivitäten zugeordnet.
- Die Aktivitäten werden einzeln nacheinander ausgeführt.
- Die Aktivitäten werden möglichst parallel ausgeführt.
- Qualitätssicherungsaktivitäten werden regelmäßig nach der Erstellung der Artefakte eingesetzt.
- Qualitätssicherungsaktivitäten werden erst am Ende des Projekts eingesetzt.
- Es werden keine Qualitätssicherungsaktivitäten eingesetzt.

Die Simulationsläufe werden aufgezeichnet und ausgewertet. Dabei wird unter anderem betrachtet, wie hoch die Qualität des Produkts ist und in welcher Zeit es entwickelt wurde. Im tatsächlichen Projekt sind mehrere Meilensteine definiert, an denen gewisse Artefakte vollständig vorliegen müssen. Zusätzlich zum Gesamtergebnis des Projekts, wird daher ermittelt, wie lange vor oder nach einem Meilenstein die dazugehörigen Artefakte vollständig waren und in welcher Qualität sie vorgelegen haben.

Anhand der Ergebnisse dieser Tests werden die Parameter des Szenarios, insbesondere die Fähigkeiten der Mitarbeiter, die Schwierigkeit der Artefakte, und deren Umfang angepasst. Dazu werden mehrere Iterationen von Testdurchläufen und Parameteranpassung durchgeführt, bis die Ergebnisse der Tests zu den anfangs definierten Lernzielen passen. Das bedeutet, das Projektergebnis ist am besten, wenn die Artefakte in der richtigen Reihenfolge und durch die richtigen Mitarbeiter möglichst parallel erstellt werden und kontinuierliche Qualitätssicherung durchgeführt wird.

Rahmenanwendung Wir haben uns entschieden, das Planspiel in Form eines AddIns in Microsoft Project einzubetten. So können die Studierenden auf alle dort vorhandenen Werkzeuge der Projektplanerstellung und -pflege zurückgreifen. Sie lernen gleichzeitig mit einer professionellen Projektplanungssoftware umzugehen, anstatt sich in eine künstlich geschaffene Spieloberfläche einarbeiten zu müssen, die sie später nicht mehr benutzen werden. Dieser Weg erspart uns zudem die Entwicklung einer eigenen Oberfläche und eigener Werkzeuge.

Das AddIn enthält zusätzliche Schaltflächen zur Bedienung der Simulation, und verschiedene Dialoge zur Anzeige der Artefakte, Aktivitäten und Mitarbeiter. Der Standardprojektplan von Project wurde um zwei zusätzliche Spalten erweitert, in denen zu jedem Vorgang die Aktivität und das Artefakt, das bearbeitet wird, festgelegt werden muss. Die Zuordnung der Mitarbeiter geschieht, wie in Project üblich, über die Ressourcenzuteilung. Am Ende des Spiels bekommt der Spieler eine Übersicht über das Projektergebnis, das er erzielt hat.

Zu den einzelnen Artefakten, Aktivitäten, Disziplinen und Rollen wurden ausführliche Hilfetexte erstellt, die direkt im Spiel angezeigt werden können. Die Studierenden können sich damit selbstständig einarbeiten und lernen, was die einzelnen Artefakte enthalten, wie sie zusammenhängen, und wie sie erstellt werden müssen. Weiterhin gibt es Hilfetexte zum Projekt, dem Prozessmodell, der Aufgabenstellung und der Bedienung des Spiels. Das Spiel kann so ohne tutorielle Unterstützung eingesetzt werden.

Testlauf Im letzten Schritt der Entwicklung haben wir das Spiel mit einer Testgruppe von 14 Studierenden evaluiert und die Ergebnisse mittels eines Fragebogens erfasst. Es hat sich dabei gezeigt, dass die Konzeption des Spiels grundsätzlich erfolgversprechend ist und zu den gewünschten Lerneffekten führt. Die Studierenden fanden den simulierten Projektverlauf realistisch, und konnten die Zusammenhänge zwischen ihren Entscheidungen und deren Auswirkungen nachvollziehen. Beim Spielspaß wurden jedoch Defizite festgestellt. Eine ausführlichere Zusammenfassung der Ergebnisse findet sich in (Nassal, 2015).

5 Beispielszenarien

In diesem Abschnitt zeigen wir anhand von Beispielen, wie Szenarien definiert werden können, um bestimmte Aspekte der Softwareentwicklung zu simulieren. Die hier gezeigten Szenarien sind Minimalbeispiele und jeweils als Ausschnitt eines umfangreicheren Szenarios zu verstehen. Zu jedem Aspekt erläutern wir, wie eine mögliche Umsetzung aussehen kann, und beschreiben anschließend, zu welchen Simulationsergebnissen diese Umsetzung führt.

5.1 Schnittstellenentwurf und Integration

Der Entwurf einer Systemarchitektur mit wohldefinierten Schnittstellen ist die Grundvoraussetzung einer erfolgreichen Integration (Sommerville, 2001). Die Studierenden sollen die Relevanz der Schnittstellenspezifikation begreifen, und lernen, welche negativen Konsequenzen eine schlechte oder gar fehlende Schnittstellenspezifikation für die Integration hat.

Umsetzung Um diesen Sachverhalt darzustellen, werden vier Artefakte betrachtet, die Teil des zu entwickelnden Produkts sind. Der Entwurf wird auf zwei einzelne Artefakte aufgeteilt: Der *Schnittstellenentwurf* beinhaltet gesammelt die Spezifikationen zu den Schnittstellen der einzelnen Komponenten, der *Komponentenentwurf* die restlichen Spezifikationen, die notwendig sind, um diese zu implementieren. Passend dazu enthält die *Komponentenimplementierung* den Code zu den einzelnen Komponenten, und die *Integration* den Code, der diese Komponenten zu einem Gesamtsystem integriert. Zwischen den einzelnen Artefakten bestehen unterschiedliche Beziehungen, die in Abbildung 5 dargestellt sind.

Um sicherzustellen, dass Komponentenentwurf, Komponentenimplementierung und Integration in der

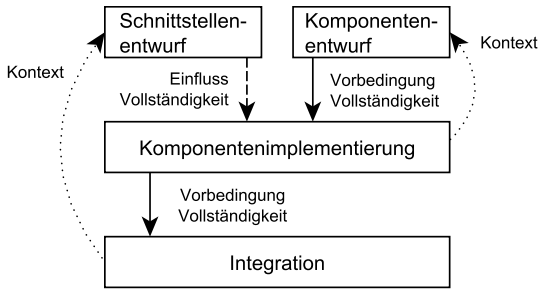

Abbildung 5: Artefakte des Integrationsszenarios und deren Beziehungen

richtigen Reihenfolge bearbeitet werden, werden dafür entsprechende Vorbedingungen eingeführt, die sicherstellen, dass ein Artefakt erst bearbeitet werden kann, wenn sein Vorgänger vollständig ist.

Da wir die Rolle des Schnittstellenentwurfs untersuchen wollen, wird dieser nicht durch eine Vorbedingung gefordert, sondern durch eine Beziehung zwischen Schnittstellenentwurf und Komponentenimplementierung realisiert, die sicherstellt, dass bei einer Änderung der Schnittstelle auch die Komponentenimplementierung angepasst werden muss. Solche Beziehungen bezeichnen wir als speziellen Einfluss. Spezielle Einflüsse sind frei durch eine entsprechende Implementierung definierbar. Im Fall dieses Szenarios wird bei einem Fortschritt des Schnittstellenentwurfs ein gegebenenfalls schon vorhandener Fortschritt der Komponentenimplementierung um einen entsprechenden Anteil reduziert.

Der Schnittstellenentwurf wird als Kontext der Integration angegeben, um die Beziehung zwischen diesen beiden Artefakten zu realisieren. Somit wirkt sich dessen Qualität und Vollständigkeit auf die Umsetzung der Integration aus. Ebenso wird eine Kontextbeziehung zwischen Komponentendesigns und Komponentenimplementierung eingefügt.

Ergebnisse Um die Auswirkungen des Schnittstellenentwurfs du zeigen, betrachten wir drei beispielhafte Vorgehensweisen, für die sich ein Spieler bei der Erstellung der Artefakte entscheiden könnte:

V1 Das vorbildliche Vorgehen, bei dem zuerst der Schnittstellenentwurf entwickelt, und anschließend durch Qualitätssicherungsmaßnahmen auf Fehler und Probleme untersucht und korrigiert wird.

V2 Ein Vorgehen, bei dem der Schnittstellenentwurf zwar erstellt, jedoch keine Qualitätssicherungsmaßnahmen vorgenommen werden.

V3 Ein Vorgehen, bei dem kein Schnittstellenentwurf erstellt wird.

Der betrachtete Projektteil wird von einem Mitarbeiter bearbeitet, der die jeweiligen Aufgaben der Reihe nach durchführt. Um den Zusammenhang zwischen Schnittstellenentwurf und Integration zu beobachten, betrachten wir die Arbeitszeit in Stunden, die der Mitarbeiter für die Durchführung der einzelnen Aufgaben benötigt, und die Qualität der dabei entstehenden Artefakte. Diese wird auf ein Intervall zwischen 0 und 1 abgebildet, wobei 1 für vollständige Defektfreiheit bzw. ein optimales Ergebnis steht, und 0 für ein gänzlich unbrauchbares Artefakt. Tabelle 1 zeigt die Simulationsergebnisse des oben beschriebenen Szenarios.

	V1	V2	V3
Dauer Schnittst.entwurf	72h	32h	–
Qualität Schnittst.entwurf	0,89	0,57	–
Dauer Integration	40h	144h	256h
Gesamtdauer Projekt	248h	312h	392h

Tabelle 1: Vergleich der unterschiedlichen Vorgehensweisen im Integrationsszenarios

Man kann beobachten, dass sich ein guter Schnittstellenentwurf positiv auf die Projektzeit auswirkt, bzw. bei fehlendem oder qualitativ unzureichendem Schnittstellenentwurf erheblicher Mehraufwand betrieben werden muss. Bei einem fehlenden Schnittstellenentwurf ist der Mehraufwand so groß, dass es sich lohnt, den Schnittstellenentwurf nachträglich zu erstellen, und die entsprechenden Teile der Komponenten neu zu entwickeln, um anschließend eine gute Integration durchführen zu können.

Der Effekt im hier beschriebenen Szenario ist drastischer dargestellt, als es in der Realität vermutlich der Fall ist, um ihn besser sichtbar zu machen. Die Effektgröße wird durch den Faktor der Kontextbeziehung zwischen Schnittstellenentwurf und Integration bestimmt, der als einfacher Wert vorliegt und leicht angepasst werden kann, um ein realistischeres Bild zu erzeugen.

5.2 Referenzdokumente für Reviews

Die Effektivität, mit der Fehler durch ein Review gefunden werden, hängt maßgeblich von der Qualifikation der Inspektoren, und der Vollständigkeit und Korrektheit der Referenzdokumente ab (Weller, 1993).

Dieser Sachverhalt soll in einem Planspiel verdeutlicht werden. Das dazugehörige Szenario kann dazu wie folgt aussehen:

Umsetzung Um den Sachverhalt nachzubilden, werden zwei Artefakte, *Prüfling* und *Referenzdokument*, benötigt. Diese beiden Artefakte werden mittels einer Beziehung in einen Kontext gesetzt, sodass der Zustand des Referenzdokuments einen entsprechenden Einfluss auf die Bearbeitung des Prüflings hat. Außerdem erstellen wir eine Reviewaktivität, die es dem Spieler erlaubt, sein Team mit dem Review des Prüflings zu beauftragen. Dieses Team besteht aus vier Mitarbeitern. Der simulierte Prüfling wurde so konfiguriert, dass er 66 Defekte enthält, die mit unterschiedlicher Schwierigkeit zu entdecken sind.

Ergebnisse Um den oben beschriebenen Aspekt zu untersuchen, betrachten wir die durchschnittliche Anzahl der gefundenen Defekte pro Tag und die Anzahl der insgesamt aufgedeckten Defekte unter verschiedenen Voraussetzungen. Dazu variieren wir die Voll-

ständigkeit und Korrektheit des Referenzdokuments, und die Fähigkeit der Mitarbeiter, ein Review durchzuführen. Eine Qualität des Referenzdokuments von 1,0 entspricht dabei einem Dokument ohne Defekte, eine Qualität von 0,7 dem, was durchschnittliche Dokumente an Defekten aufweisen, und eine Qualität von 0,2 einem sehr fehlerbehafteten und somit ungeeigneten Dokument. Die Fähigkeiten der Inspektoren werden im Intervall von 0 bis 1 abgebildet, wobei 0,5 einer durchschnittlichen Fähigkeit auf diesem Gebiet entspricht. Tabelle 2 zeigt die Ergebnisse der Simulationsläufe.

Referenzdok.		Fähigkeit	gefundene Defekte	
Vollst.	Qualität	Inspekt.	p. Tag	insgesamt
100%	0,7	0,5	8,0	48
50%	0,7	0,5	3,5	42
0%	0,7	0,5	2,7	36
100%	0,2	0,5	4,2	42
100%	1,0	0,5	13,2	66
100%	0,7	1,0	16,5	66
100%	0,7	0,1	1,4	18

Tabelle 2: Ergebnisse des Reviews unter verschiedenen Bedingungen

An den Daten lassen sich die oben beschriebenen Effekte erkennen. Je besser das Referenzdokument ist, umso schneller werden die im Prüfling vorhandenen Defekte aufgedeckt. Die Qualität des Referenzdokuments hat außerdem, genauso wie die Fähigkeit der Inspektoren, einen Einfluss darauf, welche Defekte nicht entdeckt werden, weil sie zu schwierig zu finden sind. Um tatsächlich alle Defekte zu finden, müssen entweder die Referenzdokumente vollständig und in optimaler Qualität vorliegen, oder alternativ die Fähigkeiten der Inspektoren entsprechend hoch sein.

Abbildung 6 zeigt den zeitlichen Verlauf einer simulierten Reviewaktivität. Gezeigt wird die Qualität des Prüflings (durchgezogene Linie) und der Umfang der Defekte im Artefakt (gestrichelte Linie).

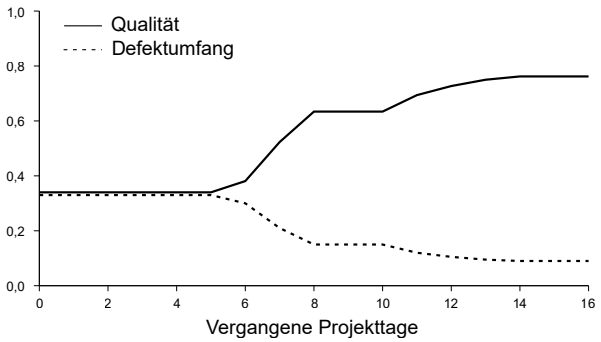

Abbildung 6: Veränderung des Prüflings während eines Reviewprozesses

Auffällig ist hier, dass in den ersten fünf Tagen kein erkennbarer Fortschritt stattfindet. Zwei der Tage entfallen auf ein Wochenende, an dem nicht gearbeitet wird, die anderen drei Tage werden von den Inspektoren genutzt, um sich in den umfangreichen Prüfling und das Referenzdokument einzuarbeiten. Am Tag 5 beginnt das Team erste Defekte zu identifizieren. Im Verlauf des weiteren Prozesses steigt die Qualität des Prüflings, während der Umfang der unentdeckten Defekte sinkt. Zu Beginn werden die Defekte dabei schneller entdeckt, da die Defektdichte im Prüfling höher ist als am Ende des Prozesses. Außerdem werden die einfacher zu entdeckenden Defekte schneller gefunden und beseitigt als die schwierigeren Defekte.

Ein weiterer Aspekt, der bislang noch nicht betrachtet wurde, ist die Anzahl der Mitarbeiter, die am Review teilnehmen. Abbildung 7 visualisiert die Ergebnisse des zu diesem Szenario gehörigen Tests, bei dem die Auswirkung der Teamgröße auf die Effizienz des Reviews untersucht wird. Sie zeigt die gefundenen Defekte pro Inspektor und Zeiteinheit in Abhängigkeit der Größe des Reviewteams. Der dazugehörige Test variiert neben der Teamgröße auch die meisten anderen Parameter des Szenarios, wie Artefaktumfang, Fähigkeiten der Inspektoren oder die Schwierigkeit der Defekte im Prüfling. Zu jeder dieser zufällig erzeugten Konfigurationen existiert ein Punkt im Schaubild. Während die Geschwindigkeit, mit der Defekte gefunden werden, mit der Teamgröße steigt, lässt sich bezogen auf die Effizienz ein Optimum bei einer Teamgröße von vier Inspektoren erkennen, was sich mit der Aussage in (Weller, 1993) deckt.

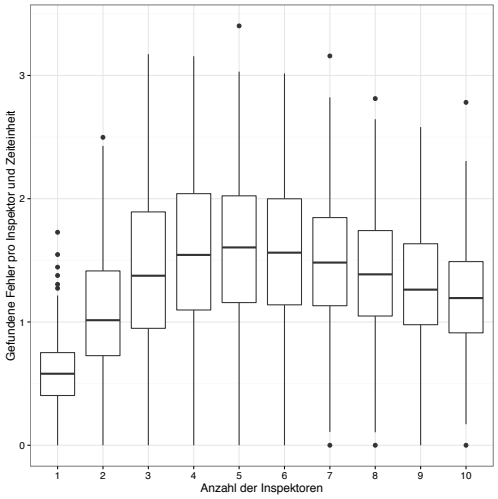

Abbildung 7: Boxplot der Effizienz in Abhängigkeit der Anzahl der Inspektoren

5.3 Brooks' Law

Neben den oben gezeigten Beispielen lassen sich mit unserem Simulationsmodell einige weitere Aspekte darstellen, von denen viele keine explizite Modellierung benötigen, sondern direkt durch das Simulationsmodell erzeugt werden.

Da wir mit unserem Modell Wissen und Kommunikation simulieren, entsteht beispielsweise der Effekt, der in Brooks' Law (Brooks, 1975) beschrieben

wird: Werden einem Projekt zu einem späten Zeitpunkt neue Mitarbeiter hinzugefügt, sind diese nicht sofort produktiv, sondern müssen sich zuerst in das Projekt einarbeiten. Dabei benötigen sie die Hilfe der schon länger im Projekt arbeitenden Mitarbeiter und halten diese dadurch von ihrer eigentlichen Arbeit ab. So verzögert sich das Projekt insgesamt, da diese Zusatzaufwände in der kurzen Projektrestzeit nicht mehr durch das größere Team kompensiert werden können. Zusätzlich bedeutet ein größeres Team einen höheren Kommunikationsaufwand, der sich negativ auf die Produktivität auswirkt.

Im Gegensatz zu anderen Modellen, wie beispielsweise das System Dynamic Modell in (Madachy, 2007), beschreiben wir diesen Effekt nicht, indem wir die Anzahl der neuen Mitarbeiter zählen und darauf basierend für einen bestimmten Zeitraum die Produktivität des Teams reduzieren, sondern erzeugen ihn so, wie er laut Brooks zustande kommt. Dazu enthält unser Simulationsmodell unter anderem Teilmodelle für Kommunikation, Lernen und soziale Interaktion. Um den Effekt zu erzeugen betrachten wir das für die Durchführung einer bestimmten Aktivität benötigte Wissen, und die Möglichkeiten, dieses Wissen zu erwerben. Eine davon besteht in der Kommunikation mit den Mitarbeitern, die schon länger im Projekt sind. Eine andere ist die Begutachtung der im Projekt vorhandenen Dokumente. Auch weitere Maßnahmen, wie Schulungen sind denkbar. Welche Möglichkeiten gewählt werden entscheiden die neuen Mitarbeiter selbstständig, oder legt der Spiel in seiner Rolle als Projektleiter fest.

Abbildung 8 zeigt den durch die Simulation erzeugten Zusammenhang zwischen dem Zeitpunkt, zu dem neue Mitarbeiter einem Projekt hinzugefügt werden, und der Veränderung der Produktivität des Teams. Dazu wurden 2.000 Projektverläufe mit zufällig gewählten Parametern simuliert. Zu einem zufällig gewählten Zeitpunkt des Projekts wurde die Mitarbeiterzahl verdoppelt. Anschließend wurde die durchschnittliche Produktivität des Teams vor und nach dem Hinzufügen der Mitarbeiter gemessen. Am Schaubild lässt sich erkennen, dass eine Verdopplung der Anzahl der Teammitglieder zu Beginn des Projekts die Produktivität um ca. 80% steigert. Die Differenz zu einer Steigerung um 100% wird vom zusätzlichen Kommunikationsaufwand verursacht. Die Produktivitätssteigerung nimmt umso mehr ab, je später die neuen Teammitglieder zum Projekt hinzugefügt werden. Der Effekt ist so stark, dass die im letzten Viertel des Projektzeitraums hinzugefügten Mitarbeiter, die Produktivität senken anstatt sie zu erhöhen.

Da wir den Effekt implizit erzeugen, berücksichtigen wir auch, dass er in bestimmten Fällen schwächer oder gar nicht auftritt. Das passiert beispielsweise, wenn die neuen Mitarbeiter schon Wissen über das Projekt mitbringen, die Einarbeitung besonders leicht ist, das neue Personal über deutlich bessere Fähigkeiten auf dem Problemgebiet besitzt als das bisherige Team, oder sich die Aufgaben so gut vom restlichen Projekt trennen lassen, dass eine Einarbeitung in die bisherigen Teile nicht notwendig ist.

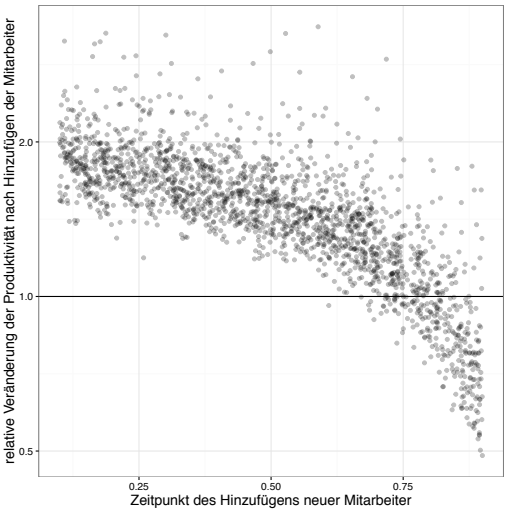

Abbildung 8: Änderung der durchschnittlichen Produktivität durch das Hinzufügen neuer Mitarbeiter

6 Fazit und Ausblick

Um die Planspielentwicklung zu unterstützen, haben wir ein Simulationsmodell entwickelt, das als Basis neuer Spiele verwendet werden kann. Dieses Modell ist zusammen mit weiteren Werkzeugen in ein Framework eingebettet. In diesem Beitrag haben wir beschrieben, wie dieses Framework eingesetzt werden kann. Das vorgeschlagene Vorgehen wurde in 10 einzelne Schritte unterteilt, die wir anhand eines Beispielprojekts erläutert haben. In weiteren Beispielen haben wir gezeigt, wie man mit dem Modell andere Aspekte der Softwareentwicklung nachbilden kann.

Unser Framework ermöglicht es, Planspiele zu entwickeln, ohne sich ausführlich mit allen einzelnen Aspekten der Projektarbeit auseinandersetzen zu müssen, da diese in unserem Modell schon enthalten sind. Das Modell kann an die jeweiligen Bedürfnisse angepasst werden; Aspekte, die wir nicht vorgesehen haben, können dem Modell hinzugefügt werden. Das Framework bietet Schnittstellen an, die direkt über C# angesprochen werden können, um den Entwickler bei der Gestaltung des Spiels alle Freiheiten zu bieten.

Eigene Erfahrungen zeigen, dass sich unser Framework und Vorgehen gut eignen, um neue Planspiele zu entwickeln. Eines unserer Ziele ist es, in Zukunft mehrere neue Spiele zu entwickeln, um zu sehen, ob sich diese Erfahrung weiterhin bestätigen lässt.

Obwohl unser Simulationsmodell schon viele Aspekte der Softwareentwicklung enthält, gibt es noch einige offene Punkte, wie beispielsweise die Arbeitsumgebung und Gestaltung des Arbeitsplatzes, die bislang bei der Berechnung des Projektverlaufs noch nicht berücksichtigt werden. Unser Ziel ist es, das Modell

kontinuierlich zu erweitern und dabei immer besser auf die Realität abzustimmen. Das kann neben dem Hinzufügen neuer Aspekte, auch durch die Verfeinerung bestehender Aspekte geschehen.

Des Weiteren wollen wir die Werkzeugunterstützung ausbauen. Momentan ist die grafische Oberfläche nur für Szenarien verfügbar, die sich im XML-Format speichern lassen. Spezielle Ereignisse und Veränderungen, die nur durch ausführbaren Code definiert werden können, werden von der Oberfläche momentan nicht unterstützt. Ein Ziel ist es, unabhängig vom XML-Format auch anders definierte Szenarien mit dem Werkzeug analysieren zu können.

Das Evaluationsframework wird im derzeitigen Stand rein über Quellcode konfiguriert und über die Kommandozeile ausgeführt. Ziel ist es, auch hierfür, soweit möglich, eine grafische Oberfläche bereitzustellen, welche die Tests ausführen und deren Auswertung entsprechend aufbereiten kann. Hier bietet sich auch die Möglichkeit, eine noch genau zu definierende Methodik zur Evaluation von Planspielszenarien und Modellen in das Werkzeug zu integrieren, um diesen Schritt auch methodisch besser unterstützen zu können.

Literatur

[Abdel-Hamid u. Madnick 1991] ABDEL-HAMID, Tarek K. ; MADNICK, Stuart E.: *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs, NJ : Prentice Hall, 1991. – 264 S.

[Brooks 1975] BROOKS, Frederick P. Frederick P.: *The Mythical Man-Month: Essays on Software Engineering*. Reading, Mass. Addison-Wesley Pub. Co., 1975. – ISBN 0–201–00650–2

[Cherif u. Davidsson 2010] CHERIF, Redha ; DAVIDSSON, Paul: Software Development Process Simulation: Multi Agent-Based Simulation versus System Dynamics. In: *Proceedings of the 10th International Conference on Multi-Agent-Based Simulation*, Springer-Verlag, 2010 (MABS'09), S. 73–85

[Coleman u. a. 1994] COLEMAN, Derek ; ARNOLD, Patrick ; BODOFF, Stephanie ; DOLLIN, Chris ; GILCHRIST, Helena ; HAYES, Fiona ; JEREMAES, Paul: *Object-oriented Development: The Fusion Method*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1994

[Jain u. Boehm 2006] JAIN, Apurva ; BOEHM, Barry W.: SimVBSE: Developing a Game for Value-Based Software Engineering. In: *CSEE&T*, IEEE Computer Society, 2006, S. 103–114

[Ludewig u. a. 1992] LUDEWIG, J. ; BASSLER, T. ; DEININGER, M. ; SCHNEIDER, K. ; SCHWILLE, J.: SESAM-simulating software projects. In: *Software Engineering and Knowledge Engineering, 1992. Proceedings., Fourth International Conference on*, 1992, S. 608–615

[Madachy 2007] MADACHY, R.J.: *Software Process Dynamics*. Wiley, 2007

[Microsoft Corporation 2013] MICROSOFT CORPORATION: *Microsoft Project*. Version 2013. 2013

[Nassal 2014] NASSAL, Alexander: A General Framework for Software Project Management Simulation Games. In: *Information Systems and Technologies (CISTI) 2014, 9th Iberian Conference on Information Systems and Technologies*, 2014, S. 321–325

[Nassal 2015] NASSAL, Alexander: Projektmanagement spielend lernen. In: *Tagungsband des 14. Workshops "Software Engineering im Unterricht der Hochschulen"2015, Dresden, Deutschland, 26. - 27. Februar 2015*, 2015, 53–64

[Nassal u. Tichy 2016] NASSAL, Alexander ; TICHY, Matthias: Modeling Human Behavior for Software Engineering Simulation Games. In: *Proceedings of the 5th International Workshop on Games and Software Engineering*. New York, NY, USA : ACM, 2016 (GAS '16). – ISBN 978–1–4503–4160–8, 8–14

[Navarro 2006] NAVARRO, Emily: *SimSE: A Software Engineering Simulation Environment for Software Process Education*, University of California, Irvine, Diss., 2006

[de Oliveira Barros u. a. 2006] OLIVEIRA BARROS, Márcio de ; DANTAS, Alexandre R. ; VERONESE, Gustavo O. ; WERNER, Cláudia Maria L.: Model-driven Game Development: Experience and Model Enhancements in Software Project Management Education. In: *Software Process: Improvement and Practice* 11 (2006), Nr. 4, S. 411–421

[Reich 2008] REICH, K.: *Konstruktivistische Didaktik: Lehr- und Studienbuch mit Methodenpool*. Beltz, 2008 (Beltz Pädagogik). – ISBN 9783407254924

[Sommerville 2001] SOMMERVILLE, Ian: *Software Engineering (6th Ed.)*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001. – ISBN 0–201–39815–X

[Weller 1993] WELLER, E. F.: Lessons from three years of inspection data (software development). In: *IEEE Software* 10 (1993), Sept, Nr. 5, S. 38–45. – ISSN 0740–7459

[Wickenberg u. Davidsson 2002] WICKENBERG, Tham ; DAVIDSSON, Paul: On Multi Agent Based Simulation of Software Development Processes. In: *In Sichman*, 2002, S. 104–113

[Ye u. a. 2007] YE, En ; LIU, Chang ; POLACK-WAHL, J.A.: Enhancing Software Engineering Education Using Teaching Aids in 3-D Online Virtual Worlds. In: *Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, 2007