universität wien

MASTERARBEIT

Titel der Masterarbeit

„Blackbox-Test für das Planspiel COREmain Hospital“

Verfasst von

Christopher Böhm

angestrebter akademischer Grad

Master of Science (MSc)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A 066 915
Studienrichtung lt. Studienblatt:	Masterstudium Betriebswirtschaft
Betreuer / Betreuerin:	ao. Univ.-Prof. Mag. Dr. Marion Rauner

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Wien, am 19.06.2015

Christopher Böhm

Vorwort

Informationstechnologiebasierte Entscheidungsfindung wird in der Zukunft eine immer größere Rolle spielen. In Rahmen dieser Masterarbeit bekam ich die Möglichkeit zum Erfolg einer wissenschaftlichen Forschung beizutragen.

In diesem Sinne möchte ich vor allem Frau a.o.-Univ.-Prof. Dr. Marion Rauner und Herrn Mag. (FH) Jörg Gesslbauer danken, welche mich beim Konzipieren der Testfälle und dem Verfassen der Arbeit unterstützt haben.

Ganz besonders möchte ich mich auch bei meinen Eltern, bei meiner Freundin Kathrin und bei allen Freunden bedanken, die mir während der gesamten Studienzeit bei Seite gestanden haben.

Im Zuge dieser Arbeit beziehen sich sämtliche geschlechtsbezogenen Bezeichnungen selbstverständlich auf beide Geschlechter.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

bzw.	beziehungsweise
COREmain	**co**mpetition under different **re**imbursement systems – a **ma**nagement game via **in**ternet for hospitals
DRG	Diagonis-Related-Groups
inkl.	inklusive
LKF	leistungsorientierte Krankenanstaltenfinanzierung
MTR	Medizinisch-technische Radiologieassistenten
OP	Operation
PIN	persönliche Identifikationsnummer
s.	siehe
vgl.	vergleiche
z. B.	zum Beispiel

1. Einführung

Diese Masterarbeit beschäftigt sich mit der Durchführung eines das Blackbox-Tests, ein Testverfahren des dynamischen Softwaretests, welches an dem Planspiel COREmain Hospital (**co**mpetition under different **re**imbursement systems – a **ma**nagement game via **in**ternet for hospitals)[1] durchgeführt wird. In diesem Kapitel wird die Motivation und die Zielsetzung (s. Abschnitt 1.1.) erläutert und daraufhin der Aufbau der Masterarbeit (s. Abschnitt 1.2.) beschrieben.

1.2. Motivation und Zielsetzung

Ein Krankenhausbesuch ist ausgenommen bei einer Geburt, keine Freude. Grundsätzlich sind lange Wartezeiten, verbunden mit einer mittleren medizinischen Behandlungsqualität die Regel. Doch liegt dies meist nicht am vorhandenen Personal, sondern viel mehr am Kostendruck der einzelnen Krankenhäuser. In den vergangenen Jahren wurden die Finanzierungsmodelle für stationäre Patienten umgestellt, doch die Kosten und auch die Anzahl an medizinischen Behandlungen, allein aufgrund des demographischen Wandels und des technologischen Fortschritts, steigen immens. Die Entscheidungsträger benötigen daher Hilfestellung, damit sie die Menge an zur Verfügung stehenden Kennzahlen und die Komplexität eines Krankenhauses besser verstehen können. Mithilfe von Planspielen können die Verantwortlichen, etwa finanzielle, organisatorische und personelle Veränderungen simulieren und daraus Handlungsempfehlungen ableiten.[2]

Das Planspiel COREmain Hospital simuliert komplexe Situationen der realen Welt mit dem Ziel, Prozesse innerhalb sowie außerhalb der Krankenhäuser besser zu verstehen sowie die daraus resultierenden Konsequenzen abschätzen zu können um Entscheidungen zielgerichteter zu treffen. Das Planspiel befindet sich derzeit in der finalen Programmierungs- und Testphase, in der bereits erste Unstimmigkeiten zwischen Anforderungen und Realisierung auftreten könnten. Daher ist es wichtig, die Software einem ersten dynamischen Testverfahren zu unterziehen.[3]

[1] Vgl. Rauner et al. (2008), S. 950
[2] vgl. Rosen (2008), S. 159
[3] Vgl. Rauner et al. (2008), S. 949 ff.

Ziel dieser Masterarbeit ist es, aus dem vorhandenen Datensatz, einer Formelsammlung von Kraus et al. (2012) für das Planspiel COREmain Hospital basierend auf Rauner et al. (2008), Kraus et al. (2010) und Kraus (2012), Testfälle zu entwickeln, welche Fehler in der Implementierung aufdecken und somit die Qualität der Software steigern.

Aufgrund der dem Testersteller zur Verfügung gestellten Formelsammlung, welche in weiterer Folge noch genauer beschrieben wird, konnten die Formeln vier Teilbereichen zugeordnet werden. Die Herausforderung besteht darin, ohne Kenntnis des Quellcodes unterschiedliche Testfälle für die einzelnen Testobjekte entwickeln und diese mit Hilfe von Microsoft Excel dermaßen aufzubereiten, dass damit ein umfangreicher Test durchführt werden kann. Auf Basis des Tests kann eine korrekte Implementierung der mathematischen Formeln im Quellcode überprüft werden.

1.2. Aufbau der Arbeit

Über das Planspiel COREmain Hospital wurden bereits einige Publikationen veröffentlicht wie z.B. von Rauner et al. (2008), Kraus et al. (2010) oder die Dissertation von Kraus (2012). Diese befassten sich ausschließlich mit dem Inhalt des Planspiels sowie welche Funktionen notwendig sind, um einen Mehrwert für den Nutzer zu schaffen, da bereits einige ähnliche computerbasierte Planspiele für das Gesundheitswesen auf dem Markt existieren. Einen Überblick über das Planspiel mit seiner Funktionsweise wird im 2. Kapitel beschrieben.

Diese Masterarbeit legt den Fokus auf die computerbasierten Umsetzung des Planspiels. Im Rahmen dieser wurde ein Softwaretest der gesamten Formelsammlung von Kraus et al. (2012) durchgeführt. Zuvor werden die Grundlagen des Softwaretests im 3. Kapitel beschrieben. Für diesen stehen verschiedenen Testmethoden zur Auswahl, welche es ermöglichen sinnvolle Testfälle zu entwickeln. Diese werden unterteilt in statische Testverfahren, welche meist vor der Programmierung durchgeführt werden, und dynamische Testverfahren. Die Grundlagen der statischen Testverfahren werden im 4. Kapitel beschrieben.

Die dynamischen Testverfahren werden im 5. Kapitel erläutert und gegliedert in den Whitebox-Test und Blackbox-Test. Bei einem Whitebox-Test ist der Quellcode dem Testersteller bekannt, wohingegen beim Blackbox-Test der Quellcode für die Testerstellung unbekannt ist. Da dem Testersteller bei der Konstruktion verschiedener Testfälle ausschließlich die Formelsammlung des Planspiels COREmain Hospital zur Verfügung

gestellt wurde, ist das verwendete Testverfahren der Kategorie Blackbox-Test zuzuordnen, worauf der Fokus dieser Masterarbeit liegt.

Hauptziel der Masterarbeit ist es, die Funktionalität der Formeln und deren korrekte Implementierung in den Quellcode zu testen. Hierbei sollten Testfälle konstruiert werden, welche besonders fehleranfällig sein können. Die angewendete Methode und Datengrundlage sowie die Ergebnisse des Softwaretest am Planspiel COREmain Hospital werden detailliert im 6. Kapitel erläutert.

2. Das Planspiel COREmain Hospital

Bei dem Planspiel COREmain Hospital handelt es sich um ein internetbasiertes Simulationsplanspiel für das Gesundheitswesen, welches am Lehrstuhl für Innovations- und Technologiemanagement der Universität Wien in Zusammenarbeit mit der FH Wiener Neustadt und FH Münster entwickelt wird. Dieser basiert auf den Publikationen von Gesslbauer et al. (2008), Rauner et al. (2008), Kraus et al. (2011) und Kraus (2012).

Die Entscheidungsfindung in Krankenhäusern ist abhängig von internen und externen Umweltfaktoren. Sie ist, wie in anderen Unternehmen auch, ausschlaggebend für die Erreichung von zuvor definierten strategischen Zielen sowie für das Überleben am Markt. Die komplexe Organisationsstruktur von Krankenhäusern und die finanzielle Abhängigkeit aufgrund eines länderspezifischen extern regulierten Vergütungssystems erschweren die Entscheidungsfindung. Hierbei existieren drei unterschiedliche Vergütungsansätze.[4]

Ein Globalbudget, ein tagesbasiertes Vergütungssystem sowie ein fallbasiertes Vergütungssystem. Das Globalbudget stellt einen fest definierten Betrag in einem bestimmten Zeitraum zur Verfügung und ist unabhängig von der Anzahl an Patienten und deren Behandlung. Bei dem tagesbasiertem Vergütungssystem erhalten die Krankenhäuser einen fixen Betrag pro Patient am Tag, das in der Realität den Krankenhausaufenthalt von Patienten verlängert und das Serviceangebot ausschließlich gegen Bezahlung verbessert, da die Krankenhäuser zur Deckung ihrer Kosten ihr Einkommen erhöhen wollen. Die Idee des fallbasierten Vergütungssystems stammt aus den USA und wurde anschließend von einigen Ländern wie Deutschland sowie Österreich adaptiert. Die Vergütung im Diagonis-Related-Groups System (DRG-System) oder in Österreich als leistungsorientierte Krankenanstaltenfinanzierung (LKF) bezeichnet, erfolgt in Abhängigkeit von der Diagnose und Behandlung. Daher spielt das Vergütungssystem eine große Rolle in Bezug auf die durchgeführten Behandlungen, die Verweildauer und somit auf die medizinische Versorgungsqualität. Des Weiteren haben gesetzliche Bestimmungen einen großen Einfluss auf die Krankenhauspolitik bzw. Entscheidungsfindung.[5]

[4] Vgl. Rauner et al. (2008), S. 949 f.
[5] Vgl. ebenda

Das Planspiel COREmain Hospital ermöglicht dem Spieler die Wahl zwischen einem der drei Vergütungssysteme. Hierbei kann zudem bei der fallbasierten Finanzierungsmethode zwischen einem limitiertem und einem unlimitiertem Budget unterschieden werden.[6]

Das Planspiel ermöglicht dem Spielleiter eine Simulation von sechs Krankenhäusern pro Region. Der Spielzeitraum umfasst zwölf Perioden zu je 28 Tagen. Hierbei stehen die verschiedenen Krankenhäuser im Wettbewerb um Patienten zueinander und Spieler besitzen die Möglichkeit einzelne Abteilungen des Krankenhauses sowie die Führung über ein gesamtes Krankenhaus zu übernehmen. Vor Beginn der Simulation müssen vom Spielleiter diverse Voreinstellungen vorgenommen werden, welche nach sechs Spielperioden gegebenenfalls korrigiert werden können. Die regionale Gesundheitspolitik kann etwa durch die Gewichtung diverser Kennzahlen wie beispielsweise einer hohen Patientenzufriedenheit, einer hohen Personalzufriedenheit, einem hohen Gewinn, oder niedriger Kosten vom Spielleiter definiert werden. Das Planspiel dient als Entscheidungshilfe für die wirtschaftlichen Unternehmungen in einem Krankenhaus und/oder aller Krankenhäuser einer Region.[7]

Wie im realen Leben haben auch, in der Simulation, die monetären Werte einen starken Einfluss. Zudem können die staatliche oder auch regionale Gesundheitspolitik inklusive deren Regularien die Simulation beeinflussen. In dem Planspiel können neue Mitarbeiter im Krankenhaus angestellt und auch entlassen lassen werden. Außerdem besteht die Möglichkeit Investitionen wie beispielsweise den Ankauf von Diagnosegeräten zu simulieren, welche als Voraussetzungen zur Schaffung von Arbeitsplätzen dienen. Hierbei stehen die Hospitäler um Mitarbeiter, Diagnosegeräte und Patienten im Wettbewerb zueinander. Als Parameter zur Gewinnung neuer Patienten und damit verbunden die Bestimmung der Patientenanzahl werden die Kennzahlen Mitarbeiterzufriedenheit, Qualität der medizinischen Leistung und Patientenzufriedenheit herangezogen.[8]

Die folgende Abbildung 1 illustriert *„den detaillierten Aufbau des Spiels zuzüglich der Beziehung der Krankenhäuser zueinander und der unterschiedlichen externen Einflüsse“*[9].

[6] Vgl. Gesslbauer et al. (2008), S. 149
[7] vgl. Rauner et al. (2008), S. 952 ff.
[8] vgl. ebenda
[9] Wachabauer (2014), S. 11

Jedes Krankenhaus besteht aus vier Steuerungselementen Management, Pflege, Operation (OP) und Radiologie. Die Abbildung verdeutlicht die Konkurrenzsituation der einzelnen Krankenhäuser um den zur Verfügung stehenden Patientenpool inklusive der externen Einflüsse Vergütungssystem, Gesundheitspolitik sowie Arbeits- und Diagnosetechnolgiemarkt.[10] Die Veränderung eines dieser Steuerungselemente hat Einfluss auf die anderen. Es hat beispielsweise die Schließung eines Operationssaals direkten Einfluss auf die Verweildauer des Patienten und auf die Bettenbelegungsrate. Zudem wirkt sich dies auf den Arbeitsaufwand des Pflegepersonals aus, dessen Zufriedenheit daher sinken könnte.[11]

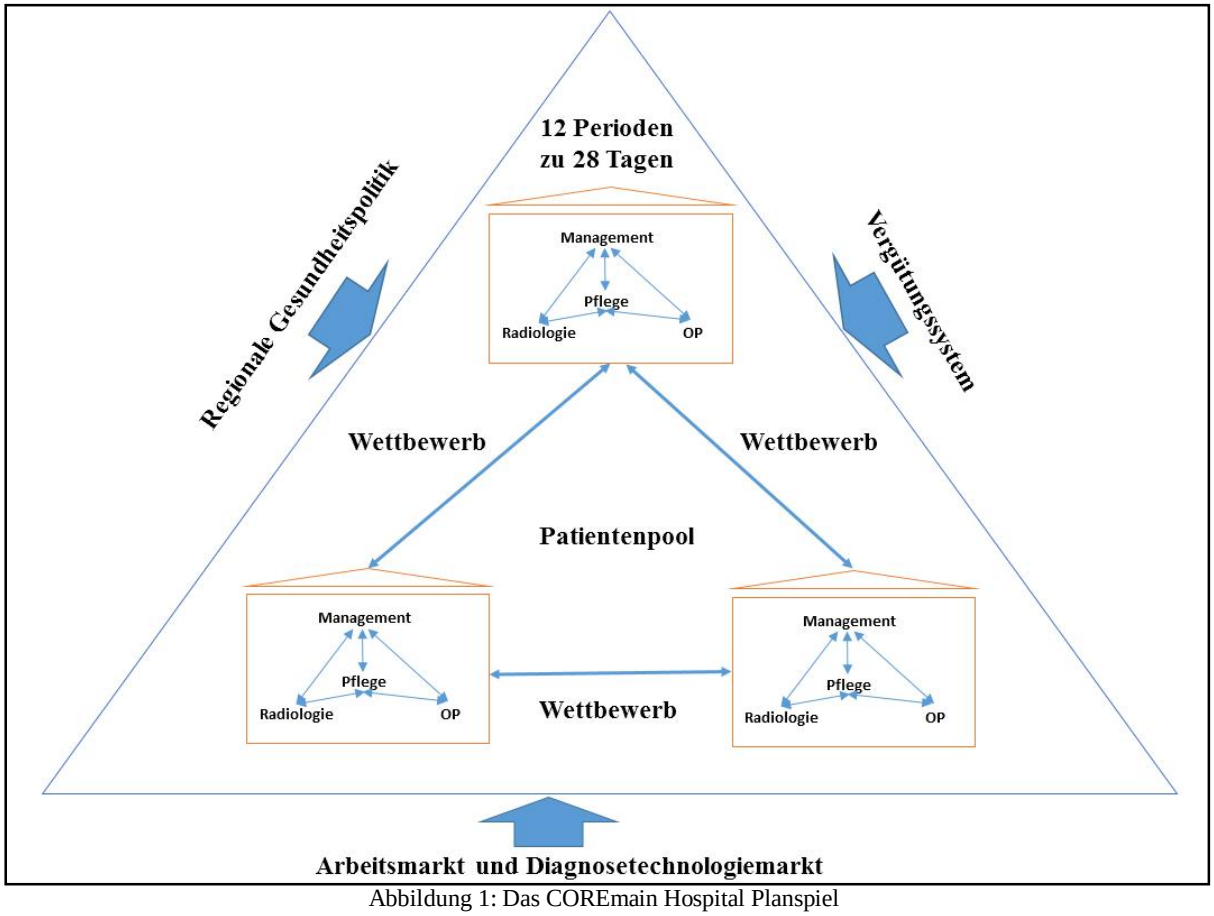


Abbildung 1: Das COREmain Hospital Planspiel

(Quelle: Wachabauer (2014) S. 12 in Anlehnung an Rauner et al. (2008), S. 952)

Nach jeder Periode wird ein Spielbericht erstellt, aus dem die Kennzahlen der Simulation hervorgehen. Anhand dieser können Anpassungen in der Ressourcenplanung, der Prozessplanung und der Finanzplanung vorgenommen werden. Gewinner des Spiels ist

[10] Vgl. Wachabauer (2014) S. 11
[11] Vgl. Gesslbauer et al. (2008), S. 149 ff.

derjenige Spieler, welcher die Ziele der regionalen Gesundheitspolitik sowie die vom Krankenhaus vorgebenden Ziele im Spielzeitraum am besten meistert.[12]

Für detailliertere Informationen zum Planspiel wird an dieser Stelle auf die bisherigen Publikation von Rauner et al. (2008) und Kraus et al. (2010) bzw. auf die Dissertation von Kraus (2012) verwiesen.

[12] Vgl. Gesslbauer et al. (2008), S. 149 ff.

3. Grundlagen von Softwaretests

In folgendem Kapitel werden die Grundbegriffe sowie die grundlegende Zielsetzung eines Softwaretests erläutert.

Erzeugnisse, dies können Teilprodukte oder Endprodukte sein, werden in der Regel auf an sie gestellte Anforderungen hin kontrolliert. Dies gilt im Allgemeinen nicht nur für materielle sondern auch für immaterielle Güter wie beispielsweise Software. Da eine Software ein immaterielles, daher kein für den Menschen greifbares, Gut ist, ist die Testung dieser schwieriger. Der überaus wichtigen und anspruchsvollen Testphase, welche mit dem Ziel, die Risiken beim Einsatz der Software zu minimieren, durchgeführt wird, kommt eine große Bedeutung zu.[13]

Klaus (2007) definiert den Begriff des Testens wie folgt: *„Testen ist der Prozess, der sämtliche Aktivitäten umfasst, welche dem Ziel dienen, für ein Software-Produkt die korrekte und vollständige Umsetzung der Anforderungen sowie das Erreichen der festgelegten Qualitätsanforderungen nachzuweisen.“* Hierbei werden diverse Tests durchgeführt. Bei einem Test wird das Testobjekt *„mittels geeigneter Testmethoden statisch überprüft oder dynamisch ausgeführt, mit dem Ziel, Fehler aufzudecken und die korrekte und vollständige Umsetzung der funktionalen und nicht-funktionalen Anforderungen nachzuweisen.“*[14]

Treten beim Testen unerwartete Ergebnisse auf, spricht man in der Literatur von einem Fehler. Ein Fehler wird als Abweichung zwischen dem Istverhalten und dem Sollverhalten, welches in den Spezifikationen oder Anforderungen festgelegt wird, definiert. Hierbei gilt es, den Fehlerbegriff vom Mangel zu unterscheiden. *„Ein Mangel liegt vor, wenn eine gestellte Anforderung oder eine berechtigte Erwartung nicht angemessen erfüllt wird.“*[15] Ein Mangel kann daher zum Beispiel die Beeinträchtigung des Nutzungsverhaltens der Software sein, obwohl die Software funktionstechnisch fehlerfrei ist. Jedem Fehler in einer Software liegt eine falsch programmierte oder vergessene Anweisung im Programm zu Grunde. Dies wird als „Defekt“ oder im englischen als „bug“ bezeichnet.[16] Um diese Fehler zu entdecken wird die Software vor ihrem kommerziellen Einsatz einem Test unterzogen. Die Begrifflichkeiten hierzu werden im Folgenden beschrieben.

[13] Vgl. Spillner und Linz (2012), S. 6
[14] Klaus (2007), S. 24
[15] Spillner und Linz (2012), S. 7
[16] Vgl. Spillner und Linz (2012), S. 7

Zur Korrektur eines Fehlers, muss dieser lokalisiert werden, da zuvor nur die Wirkung bekannt ist jedoch nicht die Ursache. Dies, die Lokalsierung sowie die Behebung des Fehlers, wird in der Literatur als „debugging“ oder „Fehlerbereinigung“ bezeichnet. Dieser Begriff ist nicht gleichzusetzen mit „Testen“. Unter dem Testen von Software wird *„jede Ausführung (im Allgemeinen stichprobenartige) eines Testobjekts verstanden, die der Überprüfung des Testobjekt dient“*[17]. Dabei wird das Soll- mit dem Istverhalten verglichen und auf die anforderten Beschaffenheit geprüft.[18]

Dabei verfolgt das Testen nach Spillner und Linz (2012) nicht nur die Fehlersuche, sondern mehrere Ziele[19]:

- Ausführung des Programms mit dem Ziel, Fehlerwirkungen nachzuweisen
- Ausführung des Programms mit dem Ziel, die Qualität zu bestimmen
- Ausführung des Programms mit dem Ziel, Vertrauen in das Programm zu erhöhen
- Analysieren des Programms mit dem Ziel, um Fehlerwirkungen vorzubeugen

Ein Testprozess besteht aus der Planung, Ausführung des Testobjekts mit Testdaten und Auswertung des Tests. Er kann in einen oder mehrere Testfälle untergliedert werden. Dieser wird anhand von festgelegten Randbedingungen, wie Voraussetzungen zur Ausführung, Eingabewerte und vorausgesagte Ausgabewerte, durchgeführt. Jeder einzelne Testfall sollte eine gewisse Wahrscheinlichkeit aufweisen, bisher unbekannte Fehler aufzuzeigen. Dabei besteht die Möglichkeit eine gesamte Reihe von Testfällen zu bilden, bei der das Ergebnis eines Falls die Basis für den Folgenden ist. Die Reihung verschiedener Testfälle hintereinander wird als Testszenario definiert. Dabei bleibt anzumerken, dass es aufgrund der Komplexität und dem Umfang der Programmierung kein fehlerfreies Softwaresystem gibt. Fehler in der Programmierung entstehen, da es immer Sonderfälle gibt, welche falsch implementiert oder einfach vergessen wurden. Auch eine Testung garantiert nicht die Fehlerfreiheit eines Systems.[20]

Der Softwaretest ist die Grundlage zur Steigerung der Softwarequalität. Um diese bestimmen zu können, werden verschiedene Qualitätsmerkmale definiert, getestet und beurteilt. Eines der Merkmale ist die Funktionalität eines Softwaresystems. Hierbei wird ein Funktionstest,

[17] Spillner (2012), S. 9
[18] Vgl. Spillner und Linz (2012), S. 8 ff.
[19] Vgl. ebenda
[20] Vgl. ebenda

welcher Defekte innerhalb des Systems zuerst entdeckt und daraufhin beseitigt, durchgeführt. Dies geschieht durch die Festlegung von Ein- und Ausgabeverhalten beziehungsweise von der Reaktion oder Wirkung des Systems auf passende Eingaben. Alle Ergebnisse müssen die Erwartungswerte ausgeben, um von einer ordnungsgemäßen Funktionalität der Software zu sprechen. Zudem müssen hierbei die anwendungsspezifischen Normen, Vereinbarungen und gesetzlichen Vorschriften erfüllt sein. Bei der Testdurchführung beziehungsweise Fehlerkorrektur ist zu beachten, dass keine neuen Fehlerquellen produziert werden. Daher sollten die Testfälle entsprechend einer späteren Benutzung konzipiert werden.[21]

Die Qualität einer Software wird zudem von weiteren Merkmalen bestimmt. Dazu gehören Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit sowie Übertragbarkeit. Die Zuverlässigkeit einer Software wird zuvor in den Spezifikationen festgelegt. Darunter wird die Bewahrung eines Leistungsniveaus über einen zuvor definierten Zeitraum inklusive der an sie gestellten Anforderungen verstanden. Heutzutage ist die Benutzbarkeit von Systemen einer der Haupterfolgsfaktoren. Je leichter ein Softwaresystem zu erlernen, zu verstehen und zu bedienen ist, desto höher ist die Akzeptanz der zukünftigen Nutzer. Zudem spielen hierbei die Einhaltung von Standards und Style Guides eine entscheidende Rolle. Ein weiteres Qualitätsmerkmal, welches vor allem im Berufsalltag zu tragen kommt, ist die Effizienz eines Systems. Diese wird anhand der *„benötigten Zeit und dem Verbrauch an Betriebsmitteln für die Erfüllung einer Aufgabe“*[22] gemessen. Weitere Merkmale zur Qualitätsbestimmung einer Software sind die Änderbarkeit und Übertragbarkeit, da Softwaresysteme mit anderen Systemen kompatibel sein müssen und über einen längeren Zeitraum auf verschiedenen Betriebssystemen und Hardwarekonfigurationen eingesetzt werden sollen. Nicht alle der aufgeführten Merkmale können bei der Programmierung einer Software in gleichem Maß erfüllt werden, daher ist es notwendig, zuvor das Qualitätsniveau in den Spezifikationen festzulegen. Die Erfüllung dieses wird dann mit Hilfe von zweckmäßigen Untersuchungen überprüft.[23] Die dafür existierenden Testverfahren werden in den folgenden Kapiteln erläutert und anhand von Beispielen veranschaulicht.

[21] Vgl. Spillner und Linz (2012), S. 13 ff.
[22] Spillner und Linz (2012), S. 13
[23] Vgl. Spillner und Linz (2012), S. 13 ff.

4. Statische Softwaretestverfahren

Im Unterschied zu den später beschriebenen dynamischen Testverfahren (s. Kapitel 5), wird bei statischen Testverfahren *„das Testobjekt nicht mit Testdaten versehen und ausgeführt, sondern einer Analyse unterzogen“*[24]. Dabei werden entweder Werkzeuge oder intensive Betrachtung für die Fehlerermittlung in den Dokumenten verwendet. Ziel ist es, Fehler, beispielweise in den Spezifikationen oder Standards, frühzeitig zu entdecken um spätere Nach- und Verbesserungsarbeiten vermeiden zu können. Die hierbei untersuchten Dokumente beschreiben die in den Quellcode implementierten dynamischen Prozesse.[25]

Unter intensiver Betrachtung wird die Testung der Dokumente durch Nutzung der menschlichen Analyse- und Denkfähigkeit verstanden. Dies wird meist durch intensives mehrmaliges Lesen und Nachvollziehen von verschiedenen Personen vollzogen, auch Inspektion oder Review genannt. Sie sind ein effizientes Mittel zur Qualitätssicherung der zu untersuchenden Dokumente und zudem eine kostengünstige Methode zur Fehlerbeseitigung, da eine frühzeitige Fehlererkennung Zeit und Kosten im Implementierungsprozess erspart. Aufgrund dessen verringert es zudem den Zeitaufwand für die dynamischen Testverfahren.[26]

Der Reviewprozess besteht aus sechs verschiedenen Arbeitsschritten:[27]

- Einführung
- Vorbereitung
- Reviewsitzung
- Überarbeitung
- Nachbereitung

Wichtig für die Durchführung einen effektiven Reviewprozess ist eine ausgereifte Aufbereitung der Dokumente in der benötigten Beschreibungstiefe sowie eine gute Ressourcenplanung bezüglich Personal und Zeitaufwand.[28]

[24] Spillner und Linz (2012), S. 81
[25] Vgl. Spillner und Linz (2012), S. 81 ff.
[26] Vgl. ebenda
[27] Vgl. ebenda
[28] Vgl. ebenda

Für die statische Analyse müssen die Dokumente einer formalen Struktur unterliegen, sodass diese mit Hilfe von Werkzeugen untersucht werden können. Ein Beispiel für ein statisches Analysewerkzeug ist ein Compiler, welcher die Einhaltung der Syntax der verwendeten Programmiersprache prüft. Zudem stellt dieser weitere Informationen wie die Prüfung der Konsistenz von Schnittstellen oder eine „cross reference list“ der verwendeten Programmelemente zur Verfügung. Die Durchführung einer statische Analyse ist nahezu für jedes Programm zu empfehlen, da jeder entdeckte Fehler die Qualität der Dokumente erhöht. Generell sollte dies vor dem Reviewprozess durchgeführt werden, da es die einfachste Methode der Fehlerentdeckung sowie Beseitigung ist.[29]

Durch die bloße Betrachtung der Dokumente oder werkzeugunterstütze Analyse kann die Funktionalität der Software nicht bestätigt werden. Daher ist ein statisches Testverfahren vor der Programmierung als sehr wichtig aber nicht als ausreichend einzustufen, da die Software nicht ausgeführt wird und einige Fehler bzw. Fehlerzustände erst bei der Ausführung auftreten.[30] Dies geschieht bei dem dynamischen Testverfahren, welche im folgenden Kapitel beschrieben werden.

[29] Vgl. Spillner und Linz (2012), S. 99 ff.
[30] Vgl. ebenda

5. Dynamische Softwaretestverfahren

Die zuvor beschriebenen statischen Testverfahren (s. Kapitel 4) dienen der Abgrenzung zu den im Folgenden beschriebenen dynamischen Testverfahren. Auf diese wird der Fokus in der vorliegenden Masterarbeit gelegt.

Bei einer dynamischen Testung muss bereits ein ablauffähiges Programm vorhanden sein, um Testobjekte mit Eingabedatendaten zu versehen und zur Ausführungen zu bringen. Diese wird grundsätzlich auf einem Computer aufgeführt. Hierbei kann das Programm bereits in einem frühen Stadium, in einen sogenannten Testrahmen eingebettet werden, um eine ablauffähiges Programm zu erhalten. Ziel des Testens ist es, die Erfüllung möglichst vieler festgelegter Anforderungen mit geringem Aufwand nachzuweisen und eventuelle Fehler und Abweichungen zu entdecken.[31]

Hierbei wird zwischen drei verschiedenen Verfahren unterschieden[32]:

- Blackbox Verfahren (s. Abschnitt 5.1)
- Whitebox Verfahren (s. Abschnitt 5.2)
- Erfahrungsbasierte Verfahren (s. Abschnitt 5.3)

5.1. Blackbox-Verfahren

Bei dem Blackbox Verfahren sind keine Informationen über den Programmcode und den inneren Aufbau der Software vorhanden. Man beobachtet das Testobjekt daher von außen, der sogenannte Point of Oberservation liegt daher außerhalb des Testobjekts. Dies beinhaltet zudem, dass keine Steuerung (Point of Control) des Ablaufs möglich ist. Dieses Verfahren wird auch als spezifikationsbasiertes Testentwurfsverfahren bezeichnet, da die Testfälle auf Anforderungen basieren.[33]

[31] Vgl. Spillner und Linz (2012), S. 109 ff.
[32] Vgl. ebenda
[33] Vgl. ebenda

Da eine vollständige Testung mit allen möglichen Eingabewerten und Kombinationen aufgrund der großen Anzahl meist nicht möglich ist, gibt es verschiedene Verfahren zur sinnvollen Auswahl der Testfälle[34]:

- Äquivalenzklassenbildung (s. Abschnitt 5.1.1)
- Grenzwertanalyse (s. Abschnitt 5.1.2.)
- zustandsbezogenen Tests (s. Abschnitt 5.1.3)
- Ursache-Wirkungs-Graph Analyse (s. Abschnitt 5.1.4)

5.1.1. Äquivalenzklassenbildung

In diesem Verfahren werden mögliche Eingabewerte in sogenannte Äquivalenzklassen unterteilt. Eine Äquivalenzklasse ist *„eine Menge von Eingabewerten, die ein identisches funktionales Verhalten eines Testobjekts auslösen, beziehungsweise eine Menge von Ausgabewerten, die ein gleichartiges Verhalten eines Testobjekts aufzeigen.*[35]*“* Es wird davon ausgegangen, dass die Eingabewerte einer Äquivalenzklasse dieselbe Reaktion zeigen, d.h. die Ergebnisse äquivalent zu einander sind. Um eine Repräsentativität in den Äquivalenzklassen zu gewährleisten sollte beachtet werden, dass nicht nur gültige, sondern auch ungültige Eingabewerte gewählt werden. Bei der systematischen Entwicklung wird jeder Klasse ein spezifischer Definitionsbereich zugeordnet, welcher zulässige und unzulässige Werte enthält.[36] Zur Veranschaulichung wird dies anhand des Beispiels der Bargeldabhebung an einem Bankautomaten erläutert. Zuvor wurde in den Spezifikationen festgelegt, dass der minimale Abhebungsbetrag 10 Euro und der maximale Abhebungsbetrag 400 Euro beträgt. Zudem können Beträge nur in Fünferschritten gewählt werden. Tabelle 1 veranschaulicht bei dem Parameter Bargeldabhebung die möglichen Äquivalenzklassen und den zufällig gewählten Repräsentanten jeder Klasse für einen Test.

[34] Vgl. Spillner und Linz (2012), S. 114 ff.
[35] Klaus (2007), S. 33
[36] Vgl. Spillner und Linz (2012), S. 114 ff.

Parameter	Äquivalenzklasse	Repräsentant
Bargeldabhebung	$0 \leq x < 10$	5
	$10 \leq x \geq 400$	250
	$x > 400$	500

Tabelle 1: Äquivalenzklassen einer beispielhaften Testung der Bargeldabhebung

(Quelle: eigene Darstellung)

Für den durchzuführenden Test ist ein repräsentativer Wert jeder Äquivalenzklasse zu wählen. Auf die Abhängigkeit der Bedingungen zueinander wird nicht geachtet, d.h. dass beispielsweise die Bargeldverfügbarkeit im Automaten nicht im Zusammenhang mit der Bargeldausgabe steht, sondern ausschließlich die Ausgabe von Bargeld bei entsprechender Eingabe. Anschließend wird jedem Eingabewert ein Erwartungswert unter Berücksichtigung der der Bedingungen zugeordnet. Jede Kombination von gültigen Äquivalenzklassen stellt einen eigenen Testfall dar. Kombinationen ungültiger Äquivalenzklassen, d.h. die Nutzung von falschen Eingabewerten, sollten mit nur ungültigen Repräsentanten verwendet werden.[37] Im obigen Beispiel ist ein Wert über 400 Euro und unter zehn Euro sowie eine Zahl, welche nicht in Fünferschritten eingegeben wurde, ein falscher Eingabewert. Die Erwartungswerte in diesem Beispiel sind die Aufforderung eines gültigen Eingabewerts für die erste und dritte Äquivalenzklasse. Da der eingegebene Betrag der zweiten Äquivalenzklasse gültig ist wird der Betrag ausgegeben. Daraus lässt sich zusammenfassen, dass die angewendete Systematik in diesem Verfahren, die Anzahl der sinnlosen Testfälle, d.h. Testfälle in einer Äquivalenzklasse die mit sehr hoher Wahrscheinlichkeit das gleiche Verhalten des Testobjekts auslösen, minimiert. Im folgenden Abschnitt wird eine Erweiterung der Äquivalenzklassenbildung beschrieben.[38]

5.1.2. Grenzwertanalyse

Die Grenzwertanalyse ist eine Ergänzung, welche auf der zuvor beschriebenen Äquivalenzklassenbildung basiert. Fehler in der Programmierung treten meist an den Grenzen der zuvor festgelegten Äquivalenzklassen auf. Daher werden hierbei die Ränder der verschiedenen Klassen überprüft. Es wird jeweils der exakte Grenzwert und die angrenzenden Werte, d.h. der Grenzwert +/- 1, getestet. Somit entstehen dadurch in der Regel drei unterschiedliche Testfälle. In einigen Fällen „existiert kein wirklicher Grenzwert“, da dieser

[37] Vgl. Spillner und Linz (2012) S. 114 ff.
[38] Vgl. ebenda

direkt einer Äquivalenzklasse zuzuordnen ist. Daher wird lediglich ein sehr nahe an der Grenze liegender und ein sehr nahe außerhalb der Grenze liegender Wert gewählt. Die Grenzwertanalyse benötigt nach Spillner (2012) eine hohe Kreativität, da die Bestimmung der Grenzen zwar oft einfach erscheint, dies aber nicht ist. Die Kombination der Grenzwertanalyse mit der Äquivalenzklassenbildung ist eine sehr effektive Methode der Fehlerentdeckung.[39]

Bezugnehmend auf das Beispiel der Bargeldabhebung werden die Repräsentanten an den Grenzwerten der jeweiligen Äquivalenzklassen gewählt (s. Tabelle 2). Hierbei wurden zur Veranschaulichung jeweils zwei Eingabewerte direkt an den Rändern der Grenzen gewählt. Die Erwartungswerte sind identisch mit dem vorherigen Beispiel. Die interessanten Testrepräsentanten sind bei der Grenzwertanalyse immer das erste und letzte Element einer Klasse, wie in diesem Beispiel ersichtlich. Auch bei der Grenzwertanalyse werden die Bedingungen unabhängig voneinander betrachtet.

Parameter	**Äquivalenzklasse**	**Repräsentanten**
Bargeldabhebung	$0 \leq x < 10$	0 & 9
	$10 \leq x \geq 400$	10 & 400
	$x > 400$	401 & 410

Tabelle 2: Grenzwertanalyse einer beispielhaften Testung der Bargeldabhebung

(Quelle: eigene Darstellung)

5.1.3. Zustandsbezogene Tests

Bei den zuvor beschrieben Testverfahren Äquivalenzklassenbildung und Grenzwertanalyse waren die Ergebnisse abhängig von den Eingabewerten. Der Zustand in welchem sich das System befindet wurde nicht berücksichtigt. Es kann unterschiedliche Zustände welche durch Ereignisse ausgelöst werden annehmen. [40] Hierbei gibt es zwei spezielle Zustände, den Start- und den Endzustand. Dies wird in einem endlichen Automat oder Zustandsautomat, welcher aus einer endlichen Zahl von Zuständen besteht, modelliert.[41] Ein in der Literatur oft gewähltes Beispiel für einen Zustandsautomaten ist der Geldautomat. Dieser kann in verschiedene Zustände gesetzt werden. Hier wird wieder das Beispiel der Bargeldabhebung

[39] Vgl. Spillner und Linz (2012) S. 125 ff.
[40] Vgl. ebenda
[41] Balzert (2000) nach Spillner und Linz (2012) S.133

aufgriffen. Häufige Testabfragen sind hierbei Abhebungen bei nur geringer Bargeldverfügbarkeit des Kunden oder bei geringer Bargeldverfügbarkeit des Automaten.[42] Bei diesem Testverfahren werden die Testfälle in einem Zustandsdiagrammen dargestellt und anschließend in einen sogenannten Übergangsbaum übertragen, in dem jeder Zustand mindestens einmal erreicht, jeder Zustandsübergang mindestens einmal ausgeführt und jede Verletzung der Spezifikation der Zustandsübergänge geprüft wurde. Die zustandsbezogenen Testmethode hat beim objektorientierten Testen eine sehr hohe Bedeutung, da Objekte unterschiedliche Zustände annehmen können und der Zustand die Funktionalität des Objekts beeinflusst.[43]

5.1.4. Ursache-Wirkungs-Graph-Analyse

Bisher wurden die Eingabewerte eines Testobjekts unabhängig voneinander bzw. ausschließlich der Zustand des Objekts betrachtet. Ein strukturierteres Verfahren ist die Ursache-Wirkungs-Graph-Analyse. Hierbei werden Kombinationen von Eingabewerten zur Erstellung von Testfällen herangezogen, welche die logischen Beziehungen, ohne Kenntnis vom Quellcode, von Ursachen und der daraus resultierenden Wirkung innerhalb der Software untersuchten. Ursachen sind in diesem Zusammenhang die Eingabebedingungen, welche beim Beispiel vom Geldautomat eine ungültige Bankkarte, eine korrekt eingegebene Persönliche Identifikationsnummer (PIN) oder die Verfügbarkeit von Geld im Automaten und auf dem Konto sein können. Wirkungen sind Ausgabebedingungen oder Reaktionen auf die Eingabe. Diese könnte beispielsweise eine Zurückweisung oder Einbehaltung der Bankkarte, die Aufforderung die PIN nochmals einzugeben oder die Auszahlung des gewünschten Betrages sein.[44]

Diese Ursachen und Wirkungen werden mit Hilfe von logischen Symbolen und Einschränkungen in Beziehung gesetzt und zu einem Ursache-Wirkungs-Graph zusammengefügt (s. Abbildung 2). Wenn die Bankkarte gültig ist besteht die Möglichkeit, dass alle fünf beispielhaften Wirkungen eintreten können, wobei die erste Aktion „Karte zurückweisen“ ausschließlich bei einer ungültigen Bankkarte eintreten soll. Der gewünschte

[42] Vgl. Seo und Choi (2006), S. 2
[43] Vgl. Spillner und Linz (2012) S. 140 ff.
[44] Vgl. Spillner und Linz (2012) S. 140 ff.

Geldbetrag wird aufgrund der logischen UND-Verknüpfung nur ausgegeben, wenn die Bankkarte gültig ist, die PIN korrekt eingegeben wurde und Geld verfügbar ist.[45]

Zur besseren Übersichtlichkeit und zur Ableitung von Testfällen wird der Ursache-Wirkungs-Graph anschließend in eine Entscheidungstabelle umgeformt. Ziel der Entscheidungstabelle ist es, „interessante“ Kombinationen von Eingaben zu entwickeln, d.h. mögliche Fehlerwirkungen zu entdecken. Bei der Testdurchführung müssen den mit dieser Methode ermittelten Testfällen noch Werte und eventuell zusätzliche Bedingungen zugeordnet werden.[46]

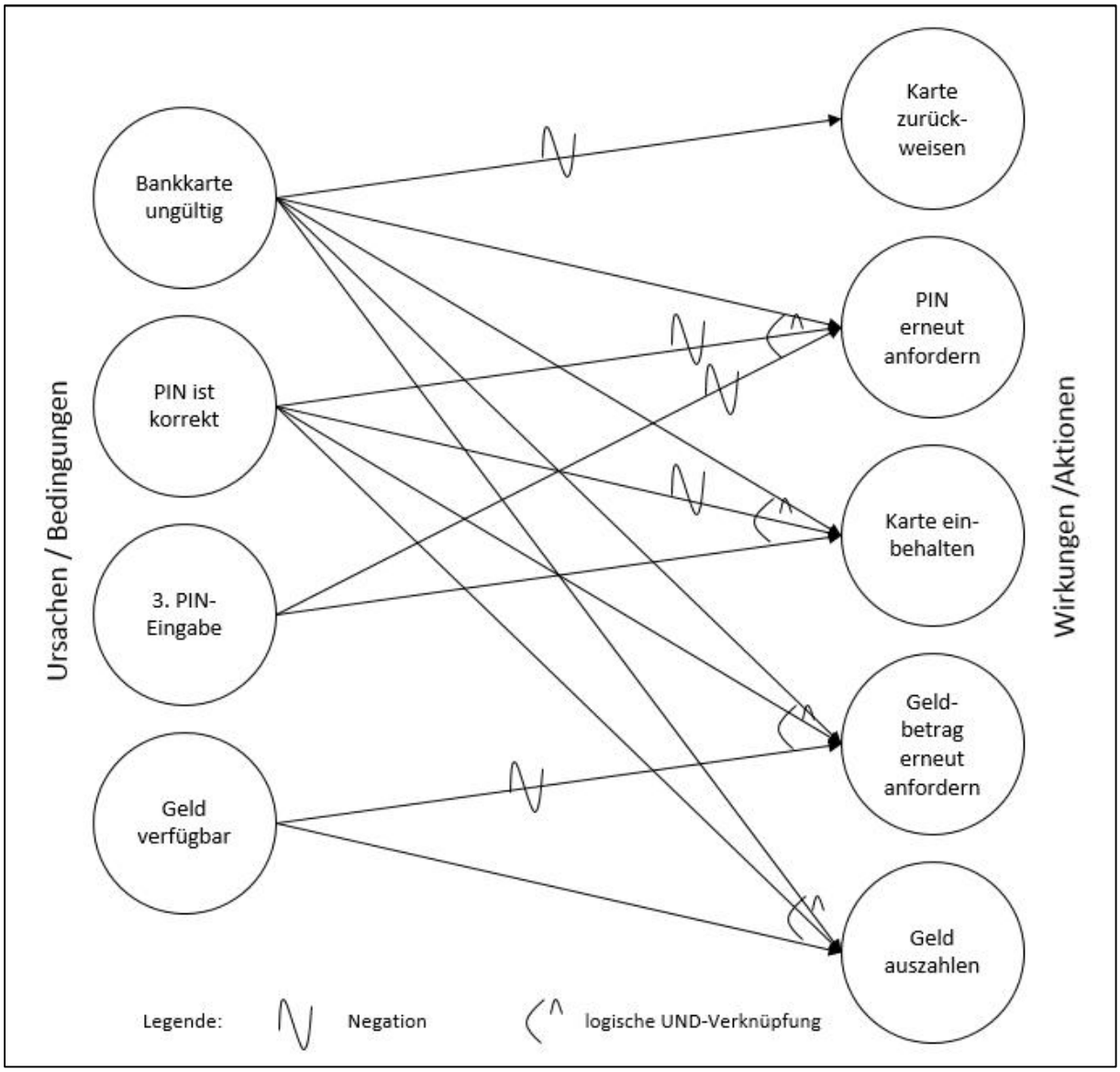


Abbildung 2: Ursache-Wirkungs-Graph eines Geldautomaten

(Quelle: eigene Darstellung in Anlehnung Spillner und Linz [2012] S.142)

[45] Vgl. Spillner und Linz (2012) S. 140 ff.
[46] Vgl. ebenda

Der Vorteil dieses Verfahrens ist es, Fehler logischer Zusammenhänge innerhalb der Software zu erkennen wobei dieses Verfahren sehr komplex und umfangreich werden kann. Dabei kann die Aufteilung der verschiedenen Spezifikation in kleinere Pakete die Komplexität reduzieren.[47]

5.2. Whitebox-Verfahren

Im Gegensatz zum Blackbox-Test ist die Basis des Whitebox-Verfahrens der Programmtext und damit die interne Struktur des Testobjekts. Daher wird dieses Verfahren auch strukturbasiertes Testentwurfsverfahren genannt. Die Grundidee ist es hierbei alle Befehle des Quellcodes mindestens einmal auszuführen. Jedoch sollte bei der Auswahl der Testfälle der Sinn eines vollständigen Tests hinterfragt werden, da die Anzahl von Testfällen bei einem vollständigen Test exorbitante Dimensionen annehmen kann. Zudem ist auch ein vollständiger Test keine Garantie für die Fehlerfreiheit. Fehlerhafte sowie fehlende Spezifikationen sind bei einem Whitebox-Test nicht ersichtlich. Ziel des Whitebox-Verfahrens sollte daher die Ausführung möglichst vieler sinnvoller im Programm enthaltenen Anweisung sein.[48]

Nach Spillner und Linz (2012) werden folgende Testfallentwurfsmethoden unterschieden:

- Anweisungstest (s. Abschnitt 5.2.1)
- Zweig- oder Entscheidungstest (s. Abschnitt 5.2.2)
- Bedingungstest und Mehrfachbedingungstest (s. Abschnitt 5.2.3)
- Pfadtest (s. Abschnitt 5.2.4)

5.2.1. Anweisungstest

Bei dem Anweisungstest werden Testfällen einer zuvor festgelegten Mindestquote identifiziert. Dabei sollen möglichst viele der bis zu diesem Zeitpunkt vorhandenen Anweisung zur Ausführung gebracht werden. Im Idealfall ist dies die Ausführung aller, dabei spricht man von einem Anweisungsüberdeckungsgrad von 100%.[49]

Bei diesem Verfahren wird der Programmtext in einen Kontrollflussgraphen transferiert, der die geforderten Überdeckungen spezifiziert. Die Anweisungen werden als Knoten dargestellt und der Kontrollfluss zwischen diesen Anweisungen als Kanten. Abfrageanweisung, darunter

[47] Vgl. Spillner und Linz (2012) S. 149 ff.
[48] Vgl. ebenda
[49] Vgl. ebenda

sind beispielsweise WENN (IF) - Abfragen oder SOLANGE (WHILE) -Anfragen zu verstehen, und Schleifen besitzen mindestens zwei Ausgangskanten.[50]

In dem Kontrollflussgraph eines Programmstücks (s. Abbildung 3) können alle Knoten bei Auswahl der richtigen Reihenfolge in einem Testfall zur Ausführung gebracht werden. In diesem Beispiel ist das die Knotenreihenfolge: a ➡b ➡d ➡e ➡f ➡h ➡j.

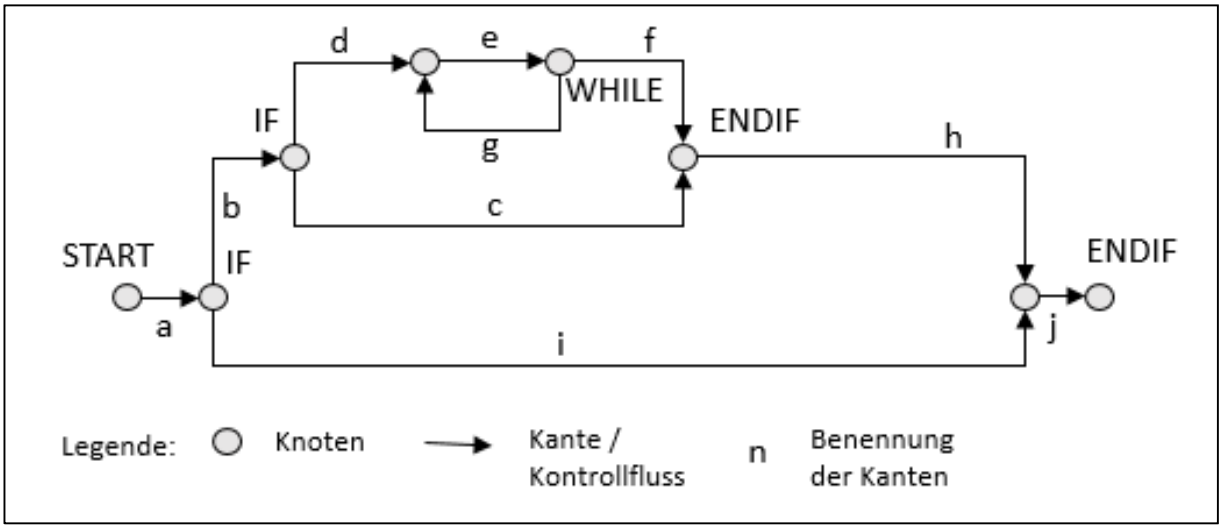


Abbildung 3: Kontrollflussgraph eines Porgrammstücks

(Quelle: eigene Darstellung in Anlehnung Spillner und Linz [2012] S.150)

5.2.2. Zweig- oder Entscheidungstest

Die Grundlage für den Zweig- oder Entscheidungstest ist der Kontrollflussgraph aus dem Anweisungstest. Die Auswertung einer Entscheidung stellt die Basis für die nächste Anweisung dar. Dabei wird keine Kombination von einzelnen Zweigen getestet sondern jeder Zweig unabhängig vom voraus oder nachhergehenden. Darunter ist zu verstehen, dass die in den Quellcode programmierten Bedingungen dahingehend geprüft werden, dass diese richtig ausgeführt werden. Ein Zweig ist die Kante zwischen zwei Knoten im Kontrollflussgraphen. Hierbei werden leere Zweige, d.h. Zweige die im Nichts enden, ausfindig gemacht. [51]

Im Beispiel aus Abbildung 4 wären die Kanten c und i leere Zweige der Abfrage. Daher sind anders als im Anweisungstest weitere Testfälle erforderlich. In obigem Beispiel sind das die Testfälle: a ➡ b ➡c ➡h ➡j / a ➡b ➡d ➡e ➡g ➡ e ➡f➡ h ➡ j / a➡ i➡ j. Durch die Überprüfung der drei Testfälle ist in diesem Beispiel die vollständige Überdeckung aller Kanten

[50] Vgl. Spillner und Linz (2012) Seite 150 ff.
[51] Vgl. ebenda

gewährleistet. Dieses Verfahren dient der logischen Prüfung und ist aufgrund diverser Schwächen erst bei Zweigüberdeckung von 100% aussagekräftig.[52]

5.2.3. Bedingungstest und Mehrfachbedingungstest

Beim Bedingungstest muss jede Bedingung einmal den Wert *wahr* und auch *falsch* annehmen. Dabei muss bei zusammengesetzten Entscheidungen bzw. mehreren Bedingungen, welche mittels logischer Verknüpfungsoperatoren (UND/ODER) verbunden sind, die Komplexität der Entscheidung beachtet werden.[53]

Beim einfachen Bedingungstest sollen die atomaren Teilbedingungen einmal den Wert *wahr* und einmal den Wert *falsch* annehmen. Die sogenannten atomaren Teilbedingungen enthalten hierin Relationssymbole und keine Verknüpfungsoperatoren. Zur Erklärung wird hierbei das Beispiel des Bargeldautomaten wieder aufgegriffen. Die Bargeldabhebung besteht aus zwei Bedingungen. Das Konto muss gedeckt sein und der Automat muss befüllt sein. Für den einfachen Bedingungstest ist es ausreichend, dass im ersten Testfall das Konto gedeckt und der Automat nicht befüllt ist und für den zweiten Testfall das Konto nicht gedeckt aber der Automat befüllt ist. Wie hierbei ersichtlich ist, spielt ausschließlich die Auswertung der Ergebnisse für den Test eine Rolle.[54]

Beim Mehrfachbedingungstest müssen alle möglichen Kombinationen getestet werden. Da im vorherigen Beispiel des Bedingungstest bereits die Kombinationen wahr/falsch und falsch/wahr ausgegeben wurden, fehlen noch zwei weitere Testfälle, wahr/wahr und falsch/falsch.[55]

Die Berechnung für die Anzahl der Testfälle ergibt sich aus:
Anzahl der Bedingungen$^{\text{Anzahl der Ergebniswahrheitswerte.}}$(für dieses Beispiel $2^2=4$)

Das Verfahren eignet sich für einfach zusammengesetzte Bedingungen, werden die Bedingungen komplexer ist dieses Testverfahren sehr fehleranfällig.[56]

[52] Vgl. Spillner und Linz (2012) Seite 152 ff.
[53] Vgl. ebenda
[54] Vgl. ebenda
[55] Vgl. ebenda
[56] Vgl. ebenda

5.2.4. Pfadtest

Der Pfadtest umschreibt die Testung aller möglichen Pfade durch ein Testobjekt, d.h. die Ausführung aller einzelnen Teile eines Programms. Ein Pfad wird als *„mögliche Abfolge von einzelnen Programmteilen in einem Programmstück“* [57] verstanden. Sobald das Testverfahren bei einer Software mit einer hohen Anzahl an Schleifen angewendet wird, stößt es an ihre Grenzen, da die Anzahl an Möglichkeiten schnell sehr stark steigt.

In diesem Abschnitt wurden die verschiedenen Testverfahren des Whitebox-Tests vorgestellt. Da dem Tester für die Testfallkonstruktion hierbei der Quellcode zur Verfügung steht, ist das sogenannte Testverfahren eher eine Kontrolle als ein Test, da *„die Güte der bisher durchgeführten Testfälle in Bezug auf die erreichte Überdeckung des Programmcodes“*[58] untersucht wird. Daher korrigiert Spillner (2013) die Bezeichnung auf *White-Box-Control*.[59]

5.3. Erfahrungsbasierte Verfahren

Ergänzend den zuvor beschriebenen methodischen Verfahren existiert ein drittes Verfahren zur Erstellung von Testfällen, die erfahrungsbasierte und intuitive Testfallermittlung. Hierbei wird auf den Erfahrungsschatz des Testers zurückgegriffen. Dabei sollen Fehlerquellen entdeckt werden, welche bei der methodischen Testfallermittlung nicht berücksichtig werden und bereits in der Vergangenheit bei anderen Programmen aufgetreten sind. Dieses Verfahren sollte jedoch lediglich die methodische Ermittlung von Testfällen ergänzen.[60]

Für das Beispiel des Geldautomaten sollen jene Tester gewählt werden, welche bereits für andere Banken deren Geldautomaten getestet haben. So könnte die Testfallermittlung bei einer anderen Bank mittels Unified Modeling Language (UML) vollzogen worden sein. Dabei werden Geschäftsvorfälle in sogenannten Use-Case-Diagrammen dargestellt. Diese stellen die Spezifikationen auf einer abstrakten Ebene dar und veranschaulichen klassische Nutzer-System-Interaktionen aus der Außensicht. Die Anwendung eines solchen Verfahrens ist nur anhand des Erfahrungsschatzes des Testers durchzuführen, da meist die typischen Verfahren ausgewählt werden.[61]

[57] Vgl. Spillner und Linz (2012), S. 155 ff.
[58] Spillner (2013)
[59] Vgl. Spillner (2013)
[60] Vgl. Spillner (2012), S. 165 ff
[61] Vgl. ebenda

6. Blackbox-Test am COREmain Planspiel

Die zuvor beschriebenen Theorien des Softwaretests wurden am Planspiel COREmain Hospital für einen Funktionstest angewandt. Im Folgenden wird die Vorgehensweise basierend auf der verwendeten Methode und Datengrundlage (s. Abschnitt 6.1) und die daraus resultierenden Ergebnisse der Testdurchführung (s. Abschnitt 6.2) am Planspiel COREmain Hospital erläutert.

6.1. Methode und Datengrundlage

Diese Masterarbeit baut auf den in der kumulierten Dissertation von Kraus (2012) erstellten Formeln für das Planspiel COREmain auf.[62,63,64] Für deren softwaretechnische Realisierung und Verfeinerung des Konzepts ist Herr Mag. Gesslbauer verantwortlich. Ziel dieser Masterarbeit ist es, die Richtigkeit der Formelimplementierung im Quellcode zu überprüfen.

Zur Erstellung eines geeigneten Testverfahrens erhielt der Testersteller die für den Test relevanten Formeln aus vier verschiedenen Teilbereichen des Planspiels[65]:

- Planspielziele (s. Abschnitt 6.1.1)
- Reports (s. Abschnitt 6.1.2)
- Kostenfunktion (s. Abschnitt 6.1.3)
- Budgetverteilungsfunktion (s. Abschnitt 6.1.4)

Der gesamte Formelsatz umfasste alle Steuerungselemente des Krankenhauses: OP, Röntgen, Pflege und Management. Die Formeln der Budgetverteilungsfunktion zielten ausschließlich auf das Managementelement ab, wohingegen die anderen drei Formelsätze die gesamten Steuerungselemente des Planspiels COREmain Hospital behandelten (siehe Abbildung 4).

[62] Rauner et al. (2008)
[63] Kraus et al. (2010)
[64] Gesslbauer et al. (2012)
[65] Kraus et al. (2012)

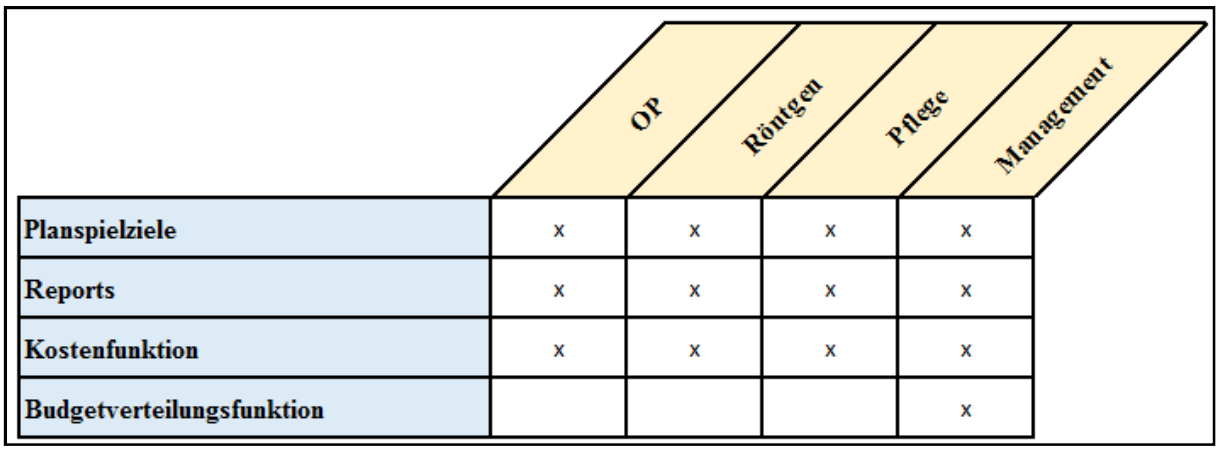


Abbildung 4: Übersicht der Formelkategorien in Bezug auf die Steuerungselemente

(Quelle: eigene Darstellung)

Ein geeignetes Testverfahren konnte ohne das theoretische Grundwissen der zuvor beschriebenen Verfahrenstheorien (s. Kapitel 4 und Kapitel 5) nicht entwickelt werden. Zu Beginn wurden die Formeln und die zur Ausführung benötigen Eingabedaten in Microsoft Excel implementiert. Dazu wurden verschiedene Eingabeblätter erstellt und nach Themengebiet unterteilt. Diese Eingabeblätter wurden sukzessive erweitert und beinhalten Hilfsformeln sowie Eingabewerte für jede einzelne Variable unter Berücksichtigung aller Indizes. Da jede Formel nacheinander in Excel erstellt wurde, veränderten sich die Verknüpfungen zu den Eingabefeldern und daher die Berechnungen ununterbrochen, welches eine große Schwierigkeit darstellte. Zum Funktionstest der in Excel erstellten Formeln wurden zu Beginn fiktive Werte gewählt, um die Richtigkeit der Berechnung innerhalb der Formel manuell zu überprüfen. Dieser Funktionstest wurde, aufgrund durchgehend auftretender Änderungen, mehrmals pro Formel ausgeführt.

Nach erfolgreicher Erstellung der Datengrundlage für die Erwartungswerte, wurde die Methode zur Testfallerstellung erarbeitet. Da dem Testersteller ausschließlich die Formeln zur Berechnung diverser Kennzahlen vorlagen und der Quellcode der Software nicht bekannt war, fällt die anzuwendende Methode unter den zuvor beschriebenen Blackbox-Test (s. Kapitel 5). Für den durchzuführenden Funktionstest wurden die Testfälle unabhängig voneinander konzipiert, da das Ziel der Tests die korrekte Implementierung der aufgestellten Formeln in den Quellcode verfolgte. Dabei wurden die Testfälle mittels des Äquivalenzklassen-Verfahrens sowie der darauf aufbauenden Grenzwertanalyse kreiert.

Bevor die Ergebnisse der Testdurchführung (s. Abschnitt 6.2) beschrieben werden, erfolgt zuerst eine Erläuterung der Formelkategorien und deren Bedeutung im Planspiel.

6.1.1. Planspielziele

Der Formelsatz der Planspielziele (s. Abbildung 5) setzt sich aus vielen unterschiedlichen Bereichen zusammen. Zum einen zielen die Formeln auf eine hohe medizinisch-pflegerische Versorgungsqualität (s. Punkt 1) ab. Diese bestehen aus Patientenkennzahlen wie Verweildauer aus medizinischer Sicht oder Abweisungsraten von Notfallpatienten sowie Personalkennzahlen wie Auslastungsgrad von Pflegepersonal oder OP-Teams. Zum anderen ermitteln die Formeln die Patientenzufriedenheit (s. Punkt 2), welche aus den Subzielen Auslastungsgrad des Krankenhauses, Patientenfreundlichkeit der Röntgengeräte und der Verweildauer aus Patientensicht besteht. Im dritten Abschnitt wird die Personalzufriedenheit (s. Punkt 3) berechnet. Diese setzt sich aus verschiedenen Zielen wie der Personalfreundlichkeit der Geräte und des Überstundenanteils der Mitarbeiter zusammen. Die weiteren Kennzahlen der Planspielziele umfassen Formeln des regionalen Gesundheitswesens (s. Punkte 4-9) und Finanzkennzahlen (s. Punkte 10-14).

Formeln: Planspielziele: A

ID nach Worddokument	Name	ID für objektorientierte Programmierung
1.	**Hohe medizinisch-pflegerische Versorgungsqualität (setzt sich aus den unteren sieben Subzielen zusammen)**	**A10**
1.1.	Verweildauer aus medizinischer Sicht	A11
1.2.	Abweisungsrate von Notfallpatienten	A12
1.3.	Mitarbeitermotivationsmaßnahmen	A13
1.4.	Anteil von Überstunden bei MTRs und OP-Teams	A14
1.5.	Personalauslastungsgrad im Pflegebereich	A15
1.6.	Medizinische Qualität von Röntgengeräten	A16
1.7.	Auslastungsgrad der OP-Teams	A17
2.	**Hohe Patientenzufriedenheit (setzt sich aus den unteren sieben Subzielen zusammen)**	**A20**
2.1.	Verweildauer aus Patientensicht	A21
2.2.	Mitarbeitermotivationsmaßnahmen	A13
2.3.	Anteil von Überstunden bei MTRs und OP-Teams	A14
2.4.	Personalauslastungsgrad im Pflegebereich	A15
2.5.	Anteil von verschobenen Röntgenuntersuchungen	A25
2.6.	Patientenfreundlichkeit von Röntgengeräten	A26
2.7.	Anteil von verschobenen OP	A27
3.	**Hohe Personalzufriedenheit (setzt sich aus den unteren fünf Subzielen zusammen)**	**A30**
3.1.	Mitarbeitermotivationsmaßnahmen	A13
3.2.	Anteil von Überstunden bei MTRs und OP-Teams	A14
3.3.	Personalauslastungsgrad im Pflegebereich	A15
3.4.	Personalfreundlichkeit von Röntgengeräten	A34
3.5.	Auslastungsgrad der OP-Teams	A17
4.	**Hoher Bettenbelegungsgrad**	**A40**
5.	**Niedrige Verweildauer**	**A50**
6.	**Hohe Anzahl an entlassenen Patienten**	**A60**
7.	**Hoher Marktanteil an Patienten**	**A70**
8.	**Schweregradindex**	**A80**
9.	**Niedrige Abweisungsrate**	**A90**
10.	**Hoher Gewinn**	**A100**
11.	**Niedrige Kosten**	**A110**
12.	**Hohe Anzahl an DRG-Punkten**	**A120**
13.	**Nicht entdeckter DRG-Creep**	**A130**
14.	**Hoher Punktwert der DRG-Punkte**	**A140**

Abbildung 5: Formelüberblick der Planspielziele

(Quelle: Kraus et al. (2012), Formelsammlung COREmain Hospital, Stand 2012, Wien)

6.1.2. Reports

Hierin setzen sich die Formeln aus den Teilbereichen Pflege (s. Punkt 1), Röntgen (s. Punkt 2) und OP (s. Punkt 3) zusammen. Sie sind, wie in Abbildung 6 ersichtlich, jeweils in die Überpunkte Tagesablauf, bereichsspezifische Formeln, Personal sowie Kosten untergliedert und dienen der Auswertung des Planspiels. Die hierin errechneten Kennzahlen stellen die Grundlage zur Berichterstellung für die verschiedenen Steuerungselemente dar. Anhand dieser Kennzahlen können nach jeder Periode Veränderungen durchgeführt werden. Im Abschlussbericht wird anhand dieser Kennzahlen der Sieger des Spiels ausfindig gemacht. Zudem dienen diese Kennzahlen zur Erstellung von Handlungsempfehlungen für Investitionsentscheidungen, Personalentscheidungen und/oder Prozessverbesserungen.

Formeln: Reports: B		
ID nach Worddokument	Name	ID für objektorientierte Programmierung
1.	**Pflegebereich**	B10
1.1.	**Tagesablauf**	
1.1.1.	Anzahl an zugewiesenen Nicht-Notfallpatienten	B11
1.1.2.	Anzahl der zugewiesenen Notfallpatienten	B12
1.2.	**Durchschnittliche Verweildauer**	
1.2.1.	Durchschnittliche Verweildauer pro Periode	B13
1.2.2.	Durchschnittliche Verweildauer pro Patiententyp	B14
1.3.	**Personal**	
1.3.1.	Anzahl an Ganztages- und Halbtagesschwestern	B15
1.3.2.	Anzahl an Arbeitsstunden von Ganztages- und	B16
1.4.	**Kosten**	
1.4.1.	Personalkosten	B17
1.4.2.	Gesamtkosten	B18
2.	**Röntgenbereich**	B20
2.1.	**Tagesablauf**	
2.1.1.	Anzahl an geplanten Röntgenuntersuchungen	B21
2.1.2.	Durchschnittliche Anzahl an Patienten in den Warteschlangen	B22
2.1.3.	Durchschnittliche Wartezeit der Patienten in den Warteschlangen	B23
2.1.4.	Geräteattribute	B24
2.2.	**Röntgengeräte**	
2.2.1.	Auslastungsgrad der Röntgengeräte	B25
2.2.2.	Auslastungsgrad pro Röntgengerätetyp	B26
2.3.	**Personal**	
2.3.1.	Anzahl an Ganztages- und Halbtages-MTRs	B27
2.3.2.	Anzahl an Arbeitsstunden von Ganztages- und Halbtages-MTRs	B28
2.4.	**Kosten**	
2.4.1.	Personalkosten	B29
2.4.2.	Gesamtkosten	B30
3.	**OP-Bereich**	B40
3.1.	**Tagesablauf**	
3.1.1.	Anzahl an geplanten Operationen	B41
3.1.2.	Durchschnittliche Anzahl an Patienten in den Warteschlangen	B42
3.1.3.	Durchschnittliche Wartezeit der Patienten in den Warteschlangen	B43
3.2.	**OP-Säle**	
3.2.1.	Auslastungsgrad der OP-Säle	B44
3.2.2.	Auslastungsgrad pro OP-Saal	B45
3.3.	**Personal**	
3.3.1.	Anzahl an Ganztages- und Halbtages-OP-Teams	B46
3.3.2.	Anzahl an Arbeitsstunden von Ganztages- und Halbtages-OP-Teams	B47
3.4.	**Kosten**	
3.4.1.	Personalkosten	B48
3.4.2.	Gesamtkosten	B49

Abbildung 6: Formelüberblick der Reports

(Quelle: Kraus et al. (2012), Formelsammlung COREmain Hospital, Stand 2012, Wien)

6.1.3. Kostenfunktion

Die Kostenfunktion besteht aus den Personalkosten (s. Punkt 1), fixen Sachkosten (s. Punkt 2), variablen Sachkosten (s. Punkt 3) und Abschreibungskosten (s. Punkt 4) sowie sonstigen Kosten (s. Punkt 5) für situationsrelevante Maßnahmen (s. Abbildung 7). Aus diesen setzt sich die Berechnung der Gesamtkosten (s. Punkt 6) eines Krankenhauses zusammen. Die daraus resultierenden Ergebnisse dienen dem Steuerungselement Management als Entscheidungsgrundlage zur Investitionsentscheidung und zur Entdeckung von Einsparungspotentialen. Die Einflussgrößen stammen aus allen Steuerungselementen.

Formeln: Kostenfunktionen: C		
ID nach Worddokument	Name	ID für objektorientierte Programmierung
1.	**Personalkosten**	C10
1.1.	Variable Personalkosten	C11
1.2.	Fixe Peronalkosten	C12
2.	**Fixe Sachkosten**	C20
2.1.	Fixe Sachkosten im Röntgenbereich	C21
2.2.	Fixe Sachkosten im OP-Bereich	C22
3.	**Variable Sachkosten**	C30
3.1.	Variable Sachkosten im Pflegebereich	C31
3.2.	Variable Sachkosten im Röntgenbereich	C32
3.3.	Variable Sachkosten im OP-Bereich	C33
4.	**Abschreibungskosten**	C40
4.1.	Abschreibungskosten von Röntgengeräten im Röntgenbereich	C41
4.2.	Abschreibungskosten von OP-Sälen im OP-Bereich	C42
5.	**Sonstige Kosten**	C50
5.1.	Kosten für Mitarbeitermotivationsmaßnahmen	C51 = A13 (ab jetzt A13)
5.2.	Kosten für Spionagedaten	C52
5.3.	Kosten für DRG-Creep	C53
6.	**Gesamtkosten**	C60 = A110 (ab jetzt A110)

Abbildung 7: Formelüberblick der Kostenfunktion

(Quelle: Kraus et al. (2012), Formelsammlung COREmain Hospital, Stand 2012, Wien)

6.1.4. Budgetverteilungsfunktion

Der Formelsatz der Budgetverteilungsfunktion enthält verschiedene Berechnungen unterschiedlicher Finanzierungsmodelle (s. Abbildung 8). Dabei unterscheiden sich die Modelle in deren Berechnungsgrundlage. Zum einem besteht die Möglichkeit eines Finanzierungsmodells auf Tagesbasis pro Patient, zum anderen wird das Budget eines Krankenhauses auf Basis eines Globalbudgets einer Region ermittelt. Hierbei werden den verschiedenen Krankenhäusern einer Region Gewichtungsfaktoren zugeordnet, welche von diversen Faktoren abhängig sind. Ein drittes Modell basiert auf dem LKF-System. Grundlage dafür ist das amerikanische DRG-System.[66]

[66] Vgl. Bundesministerium für Gesundheit, Familie und Jugend: Leistungsorientierte Krankenanstaltenfinanzierung - Systembeschreibung (2008), S. 7

Die Grundidee ist die pauschale Abrechnung pro Einzelfall, d.h. abhängig von der gestellten Diagnose und verordneten Therapie wird ein, in dem System fest definierter Betrag abgerechnet. Hierbei kann der Spielleiter vor Spielbeginn festlegen, ob ein limitiertes oder ein unlimitiertes Budget zur Verfügung steht. In Österreich existiert eine unterschiedliche Berechnungsweise wie in Deutschland. Daher werden diese Modelle differenziert voneinander betrachtet. In Österreich ist das LKF-System bundesweit einheitlich gestaltet und baut auf den leistungsorientierten Diagnosefallgruppen und speziellen Bepunktungsregeln auf.[67] Die Punkte werden anhand von stationären Aufenthalten und der Kostenkalkulation von Referenzkrankenhäusern ermittelt.[68]

Formeln: Budgetverteilungsfunktion: D		
ID nach Worddokument	**Name**	**ID für objektorientierte Programmierung**
1.	**Finanzierungsmodell: Pro Tag**	D10
3.	**Finazierungsmodell: Globalbudget**	D20
2.	**Finanzierungsmodell: Pro Fall**	
	Finanzierungsmodell: Pro Fall mit unlimitierten Budget, Österreich	D30
	Finanzierungsmodell: Pro Fall mit unlimitierten Budget, Deutschland	D40
	Finanzierungsmodell: Pro Fall mit limitierten Budget	
	Finanzierungsmodell: Pro Fall mit limitierten Budget, Österreich	D50
	Finanzierungsmodell: Pro Fall mit limitierten Budget, Deutschland	D60

Abbildung 8: Formelüberblick der Budgetverteilungsfunktion

(Quelle: Kraus et al. (2012), Formelsammlung COREmain Hospital, Stand 2012, Wien)

[67] Vgl. Bundesministerium für Gesundheit, Familie und Jugend: Leistungsorientierte Krankenanstaltenfinanzierung - LKF Modell (2008), S. 5

[68] Vgl. Bundesministerium für Gesundheit, Familie und Jugend: Leistungsorientierte Krankenanstaltenfinanzierung - Systembeschreibung (2008), S. 7

6.1.5. Zusammenfassung

In Summe wurden 84 Formeln in Excel implementiert und im Quellcode des Planspiels COREmain Hospital einem Funktionstest unterzogen. Der gesamte Formelsatz umfasst die vier zuvor beschriebenen Teilbereiche. Die Formeln des Teilbereichs der Planspielziele umfassten 22 verschiedene Berechnungen. Bei der Testdurchführung gaben 20 den Erwartungswert direkt aus und bei zwei Formeln mussten kleine Anpassungen im Quellcode vorgenommen werden, bis diese den erwarteten Wert berechneten. Ein ähnliches Bild ergab der Test des Formelsatzes der Reports. Hierin wurden in Summe 35 Formeln getestet, wovon 33 auf Anhieb den Erwartungswert ausgegeben haben und zwei im Quellcode leicht modifiziert wurden. Die Kostenfunktion besteht aus 18 Formeln, von denen 16 den Erwartungswert bei der ersten Testdurchführung erzielten und zwei Formeln kleine Anpassungen zur Folge hatten. Der Formelsatz der Budgetverteilungsfunktion umfasst neun Formeln, welche sehr komplex berechnet werden. Der Erwartungswert wurde von sieben direkt richtig berechnet und bei zwei Formeln wurde Modifikationen im Quellcode vorgenommen. Die Abbildung 9 veranschaulicht die Anzahl der Formeln je Kategorie und wie viele davon den Erwartungswert im ersten Testdurchlauf als richtiges Ergebnis ausgaben.

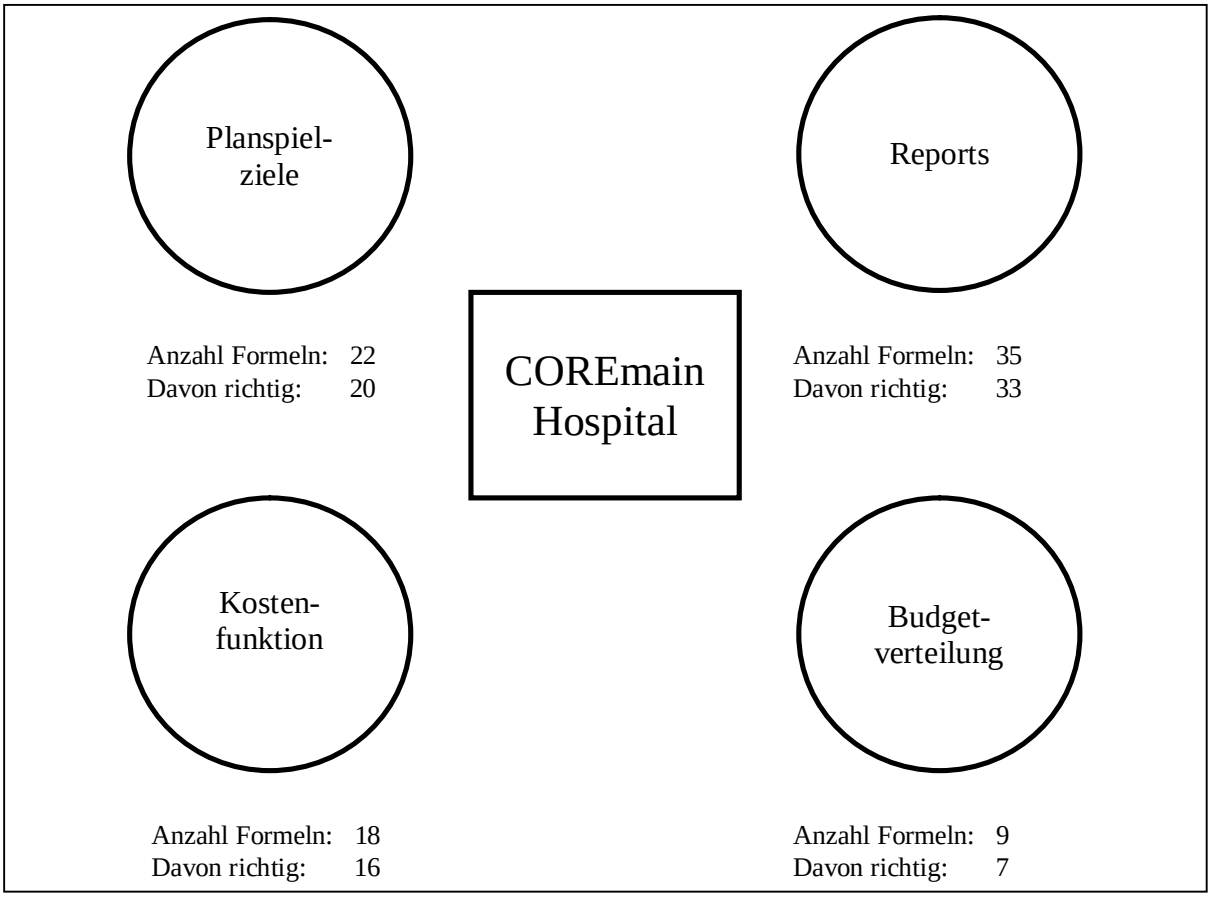


Abbildung 9: Übersicht der konstruierten und getesteten Formeln je Kategorie

(Quelle: eigene Darstellung)

6.1. Testdurchführung

Die Testdurchführung fand gemeinsam mit Herrn Mag. Jörg Gesslbauer, welcher für die Implementierung im Planspiel COREmain Hospital verantwortlich ist, statt. In den folgenden Abschnitten werden je zwei Bespiele der Testdurchführung der vier Teilbereiche: Planspielziele (s. Abschnitt 6.2.1.), Reports (s. Abschnitt 6.2.2.), Kostenfunktion (s. Abschnitt 6.2.3.) und Budgetverteilungsfunktion (s. Abschnitt 6.2.4.) erläutert. Dabei wird zunächst die zu testende Formel beschrieben, anschließend das angewendete Testverfahren inklusive der Eingabewerte erläutert und abschließend der Erwartungswert mit dem Ausgabewert verglichen und bei Abweichung eine Ursachenanalyse durchgeführt.

6.2.1. Planspielziele

In diesem Abschnitt werden die Formeln zur Berechnung der Überstunden von medizinisch-technischen Radiologieassistenten (MTRs) und OP-Teams sowie die Berechnung des Bettenbelegungsgrads je Periode erläutert.

Beispiel 1: Anteil von Überstunden bei MTRs und OP-Teams (A14)

Die in Abbildung 10 dargestellt Formel setzt sich in Summe aus zehn verschiedenen Variablen zusammen. Sie berechnet den prozentuellen Anteil von Überstunden von MTRs und OP-Teams (*perATUe*). Diese Formel dient dazu, das Verhältnis von der Regelarbeitszeit zu den Überstunden zu verdeutlichen, da im Krankenhausbetrieb bezahlte Überstunden zur Normalität gehören. Diese werden in der Regel für das Personal lukrativ vergütet, fallen aber stark ins Gewicht bei der Personalkostenbetrachtung.

Bei der Berechnung des Überstundenanteils wird die Anzahl von Überstunden von Ganz- sowie Halbtags- MTRs und OP-Teams (*perUE*) durch die Gesamtanzahl an Arbeitsstunden, bestehend aus regulärer Arbeitszeit (*perAStr*) und Überstunden, dividiert. Unter Ganz- bzw. Halbtagestageskräften wird hierbei eine 40-Stunden Arbeitswoche bzw. eine 20-Stunden Arbeitswoche als Berechnungsgrundlage definiert. Die Berechnung des Überstundenanteils sowie der Normalarbeitszeit erfolgte separat, da die Werte auch für andere Formeln relevant sind. Die Eingabewerte stammen aus dem eigens erstellten Excel-File und wurden mittels Verweisen und Verknüpfungen in die Ausgabedatei transferiert.

Anteil von Überstunden bei MTRs und OP-Teams (A14)

Formel

perATUe = $\frac{perUe}{(perUe + perAStr)}$

Berechnungen

Berechnung von perUe: (A14a)

perUe =	((40 x of_I3) - 40) x 4 x AGMTR +
	((20 x of_I4) – 20) x 4 x AHMTR +
	((40 x of_I11) - 40) x 4 x AGOPT x Anzahl an Mitarabeitern pro Ganztages-OP-Team
	((20 x of_I12) - 20) x 4 x AHOPT x Anzahl an Mitarbeitern pro Halbtages-OP-Team
perUe =	**3256**

Berechnung von perAStr: (A14b)

perAStr =	AGMTR x 40 x 4 +
	AHMTR x 20 x 4 +
	AGOPT x 40 x 4 x Anzahl an Mitarbeitern pro Ganztages-OP-Team +
	AHOPT x 20 x 4 x Anzahl an Mitarbeitern pro Halbtages-OP-Team
perAStr =	**7120**

perATUe =	**0,3138**	**31,38%**	***Anteil von Überstunden bei MTRs und OP-Teams***

Abbildung 10: Anteil von Überstunden bei MTRs und OP-Teams

(Quelle: Kraus et al. (2012), überarbeitet von Böhm)

Die bei der Testdurchführung verwendeten Eingabewerte sowie die Erläuterung der Variablen werden in Tabelle 3 veranschaulicht. Da das Ergebnis pro Periode dargestellt werden soll, wird jeder Wert mit vier multipliziert, um von einer Wochenbasis auf eine Monatsbasis (Periode) umzurechnen. Alle Variablenbezeichnungen und Beschreibungen stammen aus der Formelsammlung von Kraus et al. (2012). Für dieses Testobjekt wurden zwei Testfälle erstellt. Im ersten Testfall sollte die Formel im Quellcode zur Ausführung gebracht werden und im zweiten Testfall wurde das Verhalten bei fehlenden Eingabeparametern getestet. Dieses Verfahren ist dem Zustandstest zuzuordnen.

Variable	Beschreibung	Eingabewerte für die Test-durchführung
AGMTR	Anzahl an Ganztages-MTRs	10
AHMTR	Anzahl an Halbtages-MTRs	4
AGOPT	Anzahl an Ganztages-OP-Teams	5
AHOPT	Anzahl an Halbtages-OP-Teams	3
of_I_3	Überstundenfaktor der Ganztages-MTRs	1,5
of_I_4	Überstundenfaktor der Halbtages-MTRs	1,3
of_I_{11}	Überstundenfaktor der Ganztages-OP-Teams	1,5
of_I_{12}	Überstundenfaktor der Halbtages-OP-Teams	1,3
	Anzahl an Mitarbeitern pro Ganztages-OP-Team	5
	Anzahl an Mitarbeitern pro Halbtages-OP-Team	5

Tabelle 3: Variablenerläuterung von Formel A14 inklusive zugehöriger Eingabewerte im Testverfahren (Quelle: eigene Darstellung)

Die Eingabewerte des ersten Testfalls wurden zunächst in das Excel-File eingegeben um die Erwartungswerte zu bestimmen: *perUe* = 3.256, *perAStr* = 7.120, *perATUe* = 0,3138 (s. Abbildung 10). Anschließend wurden dieselben Werte im Quellcode der Software eingefügt und ergaben nach Ausführung der Software dasselbe Ergebnis, den Erwartungswert von *perATUe* = 31,38% (siehe Abbildung 11). Dabei wird zwischen einem skalierten und unskalierten Ergebnis unterschieden. Für die gesamte Testdurchführung ist ausschließlich das unskalierte Ergebnis von Relevanz. Für den zweiten Testfall wurde allen Überstundenfaktoren (of_I_3, of_I_4, of_I_{11} und of_I_{12}) der Wert null zugewiesen. Daraufhin sollte getestet werden, wie sich die Software bei fehlenden Eingabewerten verhält. Der Erwartungswert sollte kein Ergebnis aufweisen. Dies konnte bei der Testdurchführung bestätigt werden. Anhand dessen sind ergebnisleere Felder in Spielbericht, die Reaktion fehlender Eingabewerte in den Masken.

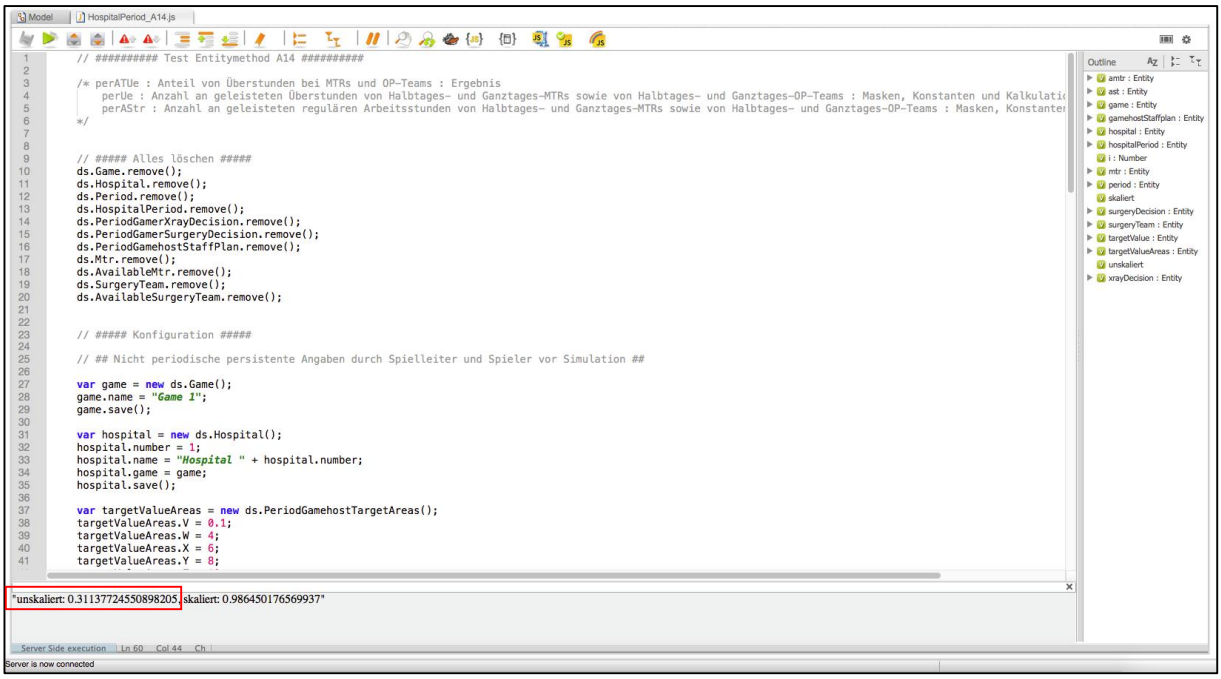


Abbildung 11: Screenshot aus dem Quellcode des COREmain Hospital Planspiels - Ergebnis der Berechnung von A14 (Quelle: Gesslbauer 2015)

Beispiel 2: Bettenbelegungsgrad (A40)

Als zweites Beispiel wurde die Berechnung des Bettenbelegungsgrads je Periode (*kBB*) gewählt, welcher die Bettenauslastung im Krankenhaus beschreibt. Diese setzt sich aus dem Mittelwert der tatsächlichen belegten Betten (Bb_{dt}), gemessen an drei verschiedenen Zeitpunkten pro Tag (Früh, Mittag, Abend) und den Planbetten (*PB*), welche die zur Verfügung stehenden Betten pro Tag darstellen, zusammen.

Die dazugehörige Formel (A40) wird folgt abgebildet:

$$kBB = \frac{\sum_{d=1}^{28} \frac{\sum_{t=1}^{3} Bb_{dt}}{3}}{PB \times 28}$$

Der Index *d* entspricht einem Tag in der Periode, welche aus 28 Tagen besteht und der Index *t* entspricht dem Messzeitpunkt an dem die tatsächlich belegten Betten pro Tageszeitpunkt gemessen werden.

Hierfür wurden drei Testfälle, welche der Äquivalenzklassenbildung zuzuordnen sind, konstruiert:

1) Eingabe der tatsächlichen Bettenbelegung in einer gesamten Periode.
   Erwartungswert: kBB = 76,92% für Bb_{dt} = 100 für d = 1,…,28 und t = 1,…,3 und *PB* = 130
2) Eingabe der tatsächlichen Bettenbelegung an nur 5 Tagen.
   Erwartungswert: kBB = 13,74% für Bb_{dt} = 100 (für d = 1,…,5 und t = 1,…,3), Bb_{dt} = 0 (für d = 6,…,28 und t = 1,…,3) und *PB* = 130
3) Eingabe von fünf Planbetten pro Tag und einer tatsächlichen Belegung von 12 Betten pro Tag.
   Erwartungswert: kBB = ERROR für Bb_{dt} = 12 (für d = 1,…,5 und t = 1,…,3), Bb_{dt} = 0 (für d = 6,…,28 und t = 1,…,3) und *PB* = 5

Im ersten Testfall war das Ergebnis identisch mit dem Erwartungswert (*kBB* = 76,92%). Der zweite Testfall wies anfangs ein unerwartetes Ergebnis aus. Bei der Fehlersuche wurde festgellt, dass die Formel in der Software dynamisch programmiert ist, d.h. wenn Werte nur an fünf Tagen eingegeben werden, wird der Bettenbelegungsgrad von fünf Tagen berechnet, wohingegen die starre Excel-Formel den Gesamtwert immer pro Periode, d.h. von 28 Tagen, berechnet. Dies wurde daraufhin im Quellcode ausgebessert, da das Ergebnis die gesamte Periode widerspiegeln soll, unabhängig davon, ob an bestimmten Tagen keine Patienten im Krankenhaus sind. Bei der erneuten Testdurchführung wurde dann der Erwartungswert von *kBB* = 0,1374 ausgegeben (s. Abbildung 12). Im dritten Testfall sollte kein Ergebnis bzw. ERROR angezeigt werden, da von der Logik her nicht mehr Betten belegt sein können als zur Verfügung stehen. Aufgrund der Umstellung auf eine dynamische Programmierung ist das Ergebnis *kBB* = 0,4286. Bei der vorherigen starren Programmierung wäre der Erwartungswert berechnet worden. Daher wurde hier eine Schwäche in der Formel entdeckt. Eine höhere Bettenauslastung als die Anzahl an zur Verfügung stehenden Betten ist in der Praxis nicht möglich. Die Formel ignoriert diese Tatsache jedoch, da an auslastungsschwachen Tagen dies kompensiert werden kann. Erst wenn die Gesamtanzahl an belegten Betten in der Periode die Planbetten übersteigt, wird dies ersichtlich. Daraufhin kann festgehalten werden, dass die Formel, nach der Korrektur, richtig in den Quellcode implementiert wurde. Jedoch ist es in der Programmierung nötig, dass während der Simulation nicht mehr Betten belegt werden können als zur Verfügung stehen. Diese Erkenntnis ist für die weitere Programmierung von großer Bedeutung.

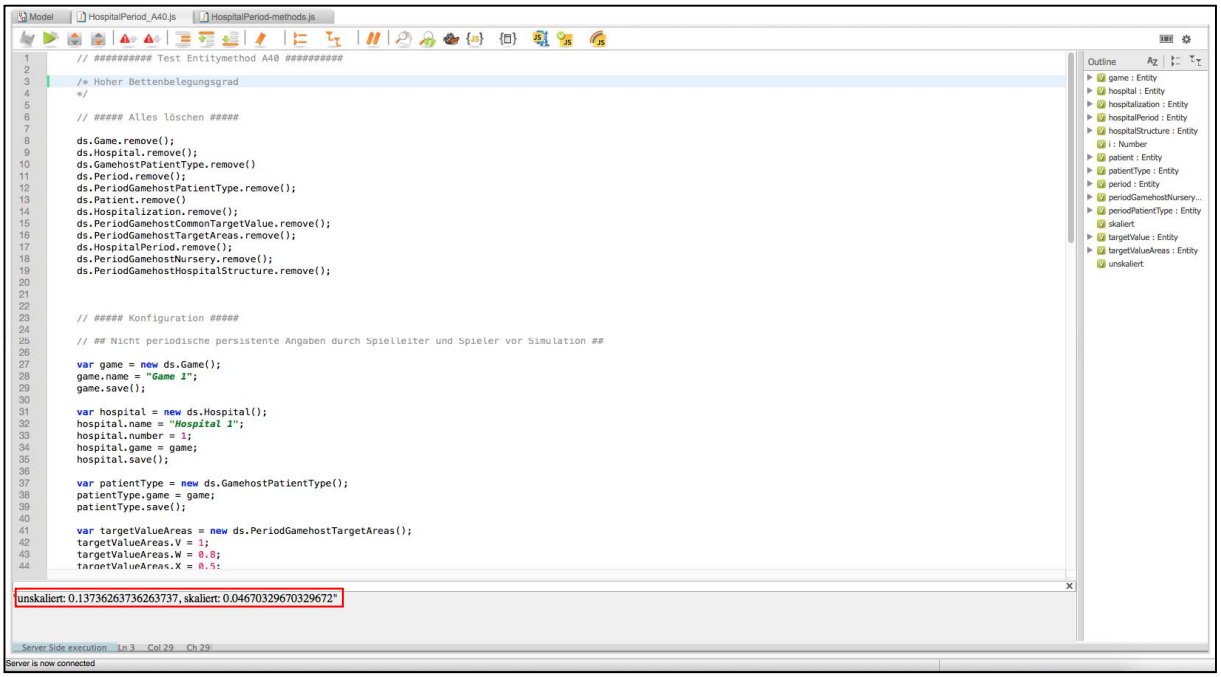


Abbildung 12: Screenshot aus dem Quellcode des COREmain Hospital Planspiels - Ergebnis der Berechnung von A40 (Quelle: Gesslbauer 2015)

6.2.2. Reports

Hierin wurden Beispiele ausschließlich aus dem Pflegebereich gewählt, da diese in den anderen Abschnitten nicht berücksichtigt wurden. Die Testdurchführung der Berechnung des Prozentsatzes von abgewiesenen Nicht-Notfallpatienten je Periode sowie die durchschnittliche Verweildauer je Periode werden im Folgenden beschrieben.

Beispiel 1: Prozentsatz der abgewiesenen Nicht-Notfallpatienten je Periode (B11a)

Die Formel, prozentuelle Anzahl von abgewiesenen Nicht-Notfallpatienten pro Periode, setzt sich aus der Gesamtanzahl an Nicht-Notfallpatienten, welche ein bestimmtes Krankenhaus besuchen, bestehend aus der Anzahl von aufgenommen Nicht-Notfallpatienten ($patANNauf_d$) am Tag d und der Anzahl von abgewiesenen Nicht-Notfallpatienten ($patANNab_d$) am Tag d zusammen. Als Notfallpatienten werden die Patientenpfade Herzinfarkt, Schlaganfall sowie Blinddarm definiert. Selbstverständlich können auch andere Diagnosen Notfälle sein. Alle Patienten, welche nicht den Notfallpatientenpfaden zuzuordnen sind, fallen unter Nicht-Notfallpatienten. Die erstellten Testfälle waren ein gültiger Wert und die direkte Eingabe eines niedrigeren Divisors als der Dividend, welcher kein Ergebnis ausgeben sollte.

$$\text{Prozentsatz der abgewiesenen Nicht-Notfallpatienten je Periode} = \frac{\sum_{d=1}^{28} patANNab_d}{\sum_{d=1}^{28} \left(patANNab_d + patANNauf_d \right)} \times 100$$

Hierbei wurden zwei Äquivalenzklassen gebildet, eine für gültige und eine ungültige Eingabewerte. Für den ersten Testfall wurden pro Tag eine Anzahl an angenommen und abgewiesenen Nicht-Notfallpatienten eingebeben. In Summe entsprach dies für eine Periode von 28 Tagen $patANNauf_d$ = 265 und $patANNab_d$ = 47. Die Berechnung ergab, wie Abbildung 13 veranschaulicht, einen Erwartungswert von 15,06%.

B11a	***Prozentsatz der abgewiesenen Nicht-Notfallpatienten in der Periode p***		**15,06%**

Abbildung 13: Ergebnis der Formel B11a im Ergebnisblatt B

(Quelle: eigene Darstellung)

Im zweiten Testfall wurde ein negativer Wert für $patANNauf_d$ = -59 und ein positiver für $patANNab_d$ = 47 eingegeben, um der Logik der Formel zu widersprechen und herauszufinden, inwieweit die Software darauf reagiert. Der Erwartungswert sollte die Ausgabe des Terminus „ERROR“ sein.

Nach der Eingabe der Werte in Quellcode des Planspiels COREmain Hospitals wurde diese zu Ausführung gebracht. Abbildung 14 weist das erwartete Ergebnis von 15,06% des ersten Testfalls aus. Im zweiten Testfall wurde richtigerweise der Terminus „ERROR“ ausgegeben.

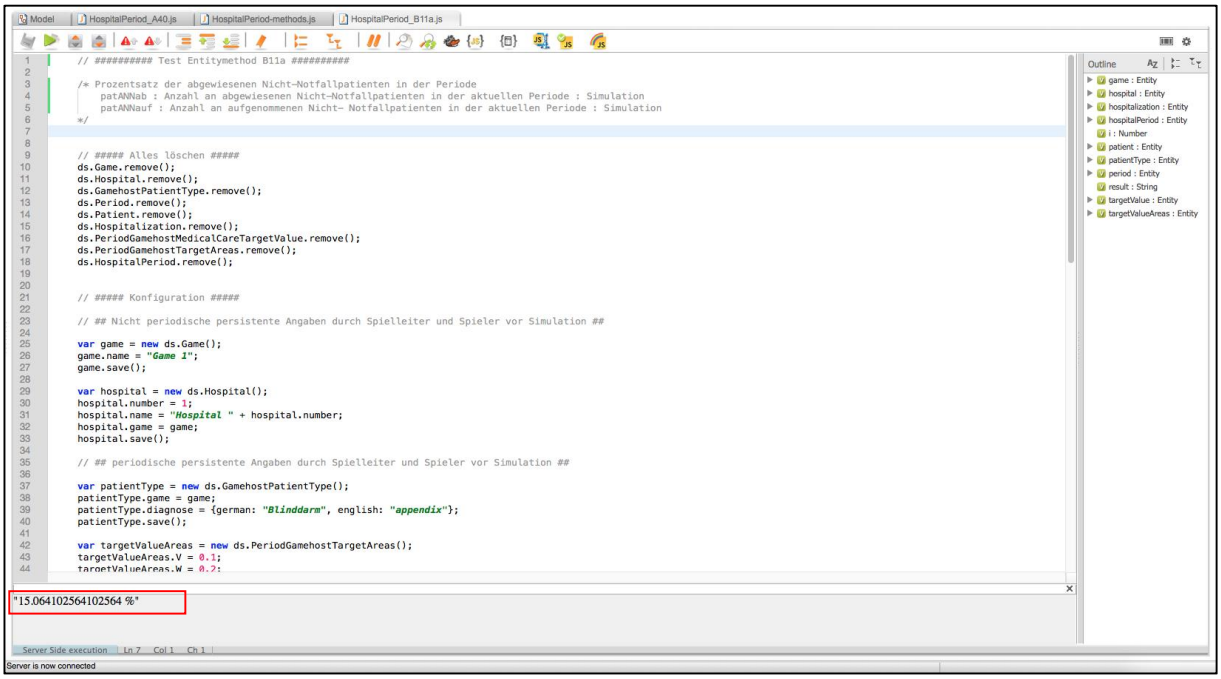


Abbildung 14: Screenshot aus dem Quellcode des COREmain Hospital Planspiels - Ergebnis der Berechnung von B11a

(Quelle: Gesslbauer 2015)

Beispiel 2: durchschnittliche Verweildauer je Periode (B13)

Im zweiten Beispiel wird die durchschnittliche Verweildauer pro Periode berechnet. Diese setzt sich aus drei unterschiedlichen Formeln zusammen. Der durchschnittlichen minimalen Verweildauer, der durchschnittlichen optimalen bzw. empfohlenen Verweildauer und der durchschnittlichen tatsächlichen Verweildauer. Die durchschnittliche minimale Verweildauer berechnet die minimale Aufenthaltsdauer aller Patiententypen auf Basis keiner Wartezeiten. Bei der durchschnittlichen optimalen bzw. empfohlenen Verweildauer wird die optimale Aufenthaltsdauer aller Patiententypen auf Basis der durchschnittlichen tatsächlichen Verweildauer je Patiententyp ermittelt. Die durchschnittliche tatsächliche Verweildauer wird durch die Simulation ermittelt.

Tabelle 4 zeigt die jeweiligen Formeln, inklusive der darin verwendeten Variablen und deren Beschreibung. Der Index *j* entspricht einem der 15 möglichen Patiententypen und der Index *k* beschreibt das Krankenhaus einer Region.

Formel	**Variable**	**Beschreibung**
durchschnittliche minimalen Verweildauer		
$\frac{\sum_{j=1}^{15}\left(pVDm_j \times patAe_{jk}\right)}{\sum_{j=1}^{15} patAe_{jk}}$	$pVDm_j$	Minimale Verweildauer des Patiententyps *j*
	$patAe_{jk}$	Anzahl an entlassenen Patienten des Patiententyps *j* im Krankenhaus *k*
durchschnittliche optimale bzw. empfohlene Verweildauer		
$\frac{\sum_{j=1}^{15}\left(pVDo_j \times patAe_{jk}\right)}{\sum_{j=1}^{15} patAe_{jk}}$	$pVDo_j$	Optimale/empfohlene Verweildauer des Patiententyps *j*
	$patAe_{jk}$	Anzahl an entlassenen Patienten des Patiententyps *j* im Krankenhaus *k*
durchschnittlich tatsächlichen Verweildauer		
$\frac{\sum_{i=1}^{I} pVDt_i}{I}$	$pVDt_i$	Tatsächliche Verweildauer des entlassenen Patienten *i*
	I	Gesamtanzahl an Patienten

Tabelle 4: Aufbau der Formel B13 durchschnittliche Verweildauer inklusive der Variablenerklärung

(Quelle: eigene Darstellung in Anlehnung an Kraus at al. 2012)

Für jede Formel wurden zwei Testfälle konstruiert, welche dem Äquivalenzklassenverfahren zuzuordnen sind. Da die Ausgabe für die durchschnittlich minimale und optimale bzw. empfohlene Verweildauer jeweils krankenhausbasiert ist, d.h. die Ergebnisse pro Krankenhaus ausgegeben werden, war die Testung für ein Krankenhaus ausreichend. Zudem wurde aus Komplexitätsgründen die Verweildauer von fünf Patiententypen als repräsentativ betrachtet. Die Anzahl der entlassenen Patienten der fünf Patiententypen im Krankenhaus betrug zehn. Für die minimale bzw. optimale Verweildauer wurden zum einen ein hoher Wert ($pVDm_j$ / $pVDo_j$ > 100) und zum anderen ein niedriger Wert ($pVDm_j$ / $pVDo_j$ < 10) gewählt. Die Erwartungswerte für die Testfälle der minimalen Verweildauer waren 3 Tage (für $pVDm_1$ = 3, $pVDm_2$ = 2, $pVDm_3$ = 3, $pVDm_4$ = 4, $pVDm_5$ = 5 und $patAe_{j1}$ = 10) sowie 158 Tage (für $pVDm_1$ = 120, $pVDm_2$ = 170, $pVDm_3$ = 160, $pVDm_4$ = 150, $pVDm_5$ = 190 und $patAe_{j1}$ = 10).

Für die Testfälle der optimalen bzw. empfohlenen Verweildauer betrugen die Erwartungswerte 6 Tage (für $pVDo_1$ = 6, $pVDo_2$ = 3, $pVDo_3$ = 5, $pVDo_4$ = 9, $pVDo_5$ = 7 und $patAe_{j1}$ = 10) und 128 Tage (für $pVDo_1$ = 101, $pVDo_2$ = 134, $pVDo_3$ = 121, $pVDo_4$ = 178, $pVDo_5$ = 103 und $patAe_{j1}$ = 10).

Alle Werte wurden nacheinander in den Quellcode des Planspiels COREmain Hospital eingegeben und daraufhin zur Ausführung gebracht. Die Ergebnisse der Berechnung der durchschnittlichen minimalen sowie der durchschnittlichen optimalen bzw. empfohlenen Verweildauer entsprachen den Erwartungswerten. Daher wurden die Formeln richtig in den Quellcode implementiert. Abbildung 15 zeigt das Ergebnis der Berechnung der durchschnittlichen minimalen Verweildauer des ersten Testfalls mit dem Ergebnis drei Tage. Hierbei ist zu beachten, dass alle Verweildauern grundsätzlichen in vollen Tagen dargestellt werden und daher die Ergebnisse im Quellcode sowie im Excel-File mit der Funktion „Aufrunden“ versehen wurden.

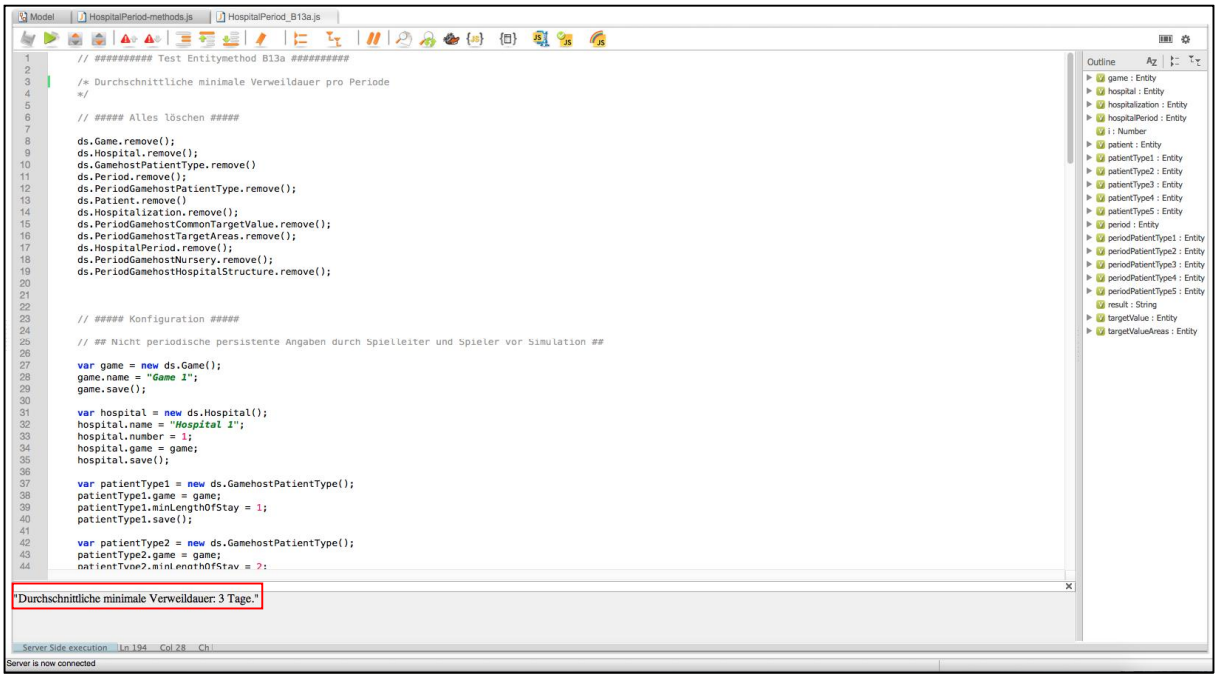


Abbildung 15: Screenshot aus dem Quellcode des COREmain Hospital Planspiels - Ergebnis der Berechnung von B13a (Quelle: Gesslbauer 2015)

Die Berechnung der tatsächlichen Verweildauer ist im Gegensatz zur minimalen und auch optimalen Verweildauer weitaus komplizierter, da diese in der Simulation entsteht. Hierbei ist während der Testung eine Schwierigkeit in der Programmierung aufgefallen. Innerhalb des Systems wird die Verweildauer in Sekunden berechnet und anschließend in die benötigte Einheit umgerechnet. Anhand des folgenden Beispiels wird die Problematik der Berechnung verdeutlicht. Wenn ein Patient das Krankenhaus um 11 Uhr vormittags betritt und am Folgetag um 9 Uhr verlässt, sollte er laut Definition zwei Tage im Krankenhaus verbracht haben. Jedoch wird bei der Berechnung die Aufenthaltsdauer, in diesem Fall 22 Stunden, aufgerundet auf einen Tag. Zur Behebung dieser Ungenauigkeit könnte man die Formel im Quellcode mit einem Zusatz +1 versehen. Jedoch ist dies nicht universell anwendbar. Ein anderer Patient kommt ebenfalls um 11 Uhr vormittags und verlässt das Krankenhaus am Folgetag um 16 Uhr. Dies entspricht einer Gesamtaufenthaltsdauer von 29 Stunden. Hierbei ist die Berechnung der Formel korrekt. Daraus lässt sich folgern, dass die Berechnung nur bei Patienten die ganze Tage (>24 Stunden) im Krankenhaus bleiben richtige Ergebnisse liefert, nicht aber bei Patienten die das Krankenhaus zu zwei unterschiedlichen Zeitpunkten an verschiedenen Tagen betreten und verlassen, bei denen der Zeitpunkt des Verlassens uhrzeittechnisch vor dem Zeitpunkt des Betretens liegt. Diese Problematik tritt bei allen Patienten, die länger als einen Tag im Krankhaus verbringen auf. Dies konnte während der Testdurchführung nicht gelöst werden. Für alle weiteren Testobjekte wurden in die

Eingabemasken korrekte Werte für die tatsächliche Verweildauer eingegeben, um diese dem Test unterziehen zu können.

6.2.3. Kostenfunktion

Die Kostenfunktion besteht aus Fixkosten, variablen Kosten, Personalkosten, Abschreibungskosten sowie sonstigen Einmalkosten. Die zur Veranschaulichung gewählten Testobjekte sind die Berechnung der variablen Sachkosten von Röntgenuntersuchungen und des Pflegebereichs sowie die Berechnung von Abschreibungskosten im Röntgenbereich.

Beispiel 1: Variable Kosten von Röntgenuntersuchungen (C32)

Die variablen Sachkosten im Röntgenbereich (*rKv*) sind unabhängig von den Anschaffungskosten, den Fixkosten und den Personalkosten im Röntgenbereich. Sie sind einerseits abhängig von der jeweiligen Untersuchungsart und andererseits vom Röntgengerätetyp auf dem die Untersuchungsart durchgeführt wird. Es gibt 15 unterschiedliche Röntgenuntersuchungsarten, welche mit dem Index *n* versehen sind und zehn unterschiedliche Röntgengeräte *m*, wobei jede Untersuchungsart auf jedem Röntgengerät unterschiedliche Kosten verursacht.

$$rKv = \sum_{n=1}^{15} rKvRU_{mn} \times rRU_{mn}$$

Die Formel multipliziert die variablen Sachkosten einer Röntgenuntersuchungsart an einem Röntgengerätetyp ($rKvRU_{mn}$) mit der Anzahl der durchgeführten Röntgenuntersuchungsarten an diesem Röntgengerätetyp (rRU_{mn}) und summiert die Ergebnisse aller variablen Kosten je Röntgenuntersuchungsart auf.

Die konstruierten Testfälle umfassten eine Eingabe von Werten für alle Untersuchungsarten mit jeweils einem Gerät pro Röntgengerätetyp und eine Variante mit fünf Untersuchungsarten und fünf Gerätetypen aber mit unterschiedlicher Anzahl von Geräten je Gerätetyp, welche jeweils eine Äquivalenzklasse bilden.

In Abbildung 16 wird das erstellte Eingabeblatt in Microsoft Excel zur Berechnung der variablen Sachkosten von Röntgenuntersuchungen veranschaulicht. Hierbei wurde die Anzahl der jeweiligen Untersuchungen pro Röntgengerätetyp sowie die variablen Kosten je

Untersuchungsart am Röntgengerät zufällig gewählt. Der Erwartungswert war *rKv* = 36.834.767 € an variablen Kosten im Röntgenbereich.

		Röntgengeräte m										SUMME variable Kosten je Untersuchungsart
n		1	2	3	4	5	6	7	8	9	10	
1	Anzahl	100	200	300	100	110	112	54	12	78	15	
	variable Kosten je Untersuchung	100	2.200	2.200	2.200	2.200	2.200	2.200	2.200	2.200	2.200	
	variable Kosten Gesamt	10.000	440.000	660.000	220.000	242.000	246.400	118.800	26.400	171.600	33.000	2.168.200
2	Anzahl	69	100	36	45	12	12	45	45	45	21	
	variable Kosten je Untersuchung	3.300	200	3.300	3.300	3.300	3.300	3.300	3.300	3.300	3.300	
	variable Kosten Gesamt	227.700	20.000	118.800	148.500	39.600	39.600	148.500	148.500	148.500	69.300	1.109.000
3	Anzahl	12	13	19	166	17	54	56	12	45	78	
	variable Kosten je Untersuchung	6.000	6.000	300	6.000	6.000	6.000	6.000	6.000	6.000	6.000	
	variable Kosten Gesamt	72.000	78.000	5.700	996.000	102.000	324.000	336.000	72.000	270.000	468.000	2.723.700
4	Anzahl	12	12	25	100	78	1	21	89	56	23	
	variable Kosten je Untersuchung	7.633	7.633	7.633	400	7.633	7.633	7.633	7.633	7.633	7.633	
	variable Kosten Gesamt	91.600	91.600	190.833	40.000	595.400	7.633	160.300	679.367	427.467	175.567	2.459.767
5	Anzahl	45	1	2	65	100	78	12	74	85	25	
	variable Kosten je Untersuchung	9.533	9.533	9.533	9.533	500	9.533	9.533	9.533	9.533	9.533	
	variable Kosten Gesamt	429.000	9.533	19.067	619.667	50.000	743.600	114.400	705.467	810.333	238.333	3.739.400
6	Anzahl	8	6	4	2	79	78	9	7	7	2	
	variable Kosten je Untersuchung	11.433	11.433	11.433	11.433	11.433	11.433	11.433	11.433	11.433	11.433	
	variable Kosten Gesamt	91.467	68.600	45.733	22.867	903.233	891.800	102.900	80.033	80.033	22.867	2.309.533
7	Anzahl	2	3	4	5	6	7	8	9	10	2	
	variable Kosten je Untersuchung	13.333	13.333	13.333	13.333	13.333	13.333	13.333	13.333	13.333	13.333	
	variable Kosten Gesamt	26.667	40.000	53.333	66.667	80.000	93.333	106.667	120.000	133.333	26.667	746.667
8	Anzahl	3	3	3	3	3	3	7	3	3	3	
	variable Kosten je Untersuchung	15.233	15.233	15.233	15.233	15.233	15.233	15.233	15.233	15.233	15.233	
	variable Kosten Gesamt	45.700	45.700	45.700	45.700	45.700	45.700	106.633	45.700	45.700	45.700	517.933
9	Anzahl	9	3	9	8	9	99	9	7	7	3	
	variable Kosten je Untersuchung	17.133	17.133	17.133	17.133	17.133	17.133	17.133	17.133	17.133	17.133	
	variable Kosten Gesamt	154.200	51.400	154.200	137.067	154.200	1.696.200	154.200	119.933	119.933	51.400	2.792.733
10	Anzahl	5	3	1	7	89	98	8	15	25	1	
	variable Kosten je Untersuchung	19.033	19.033	19.033	19.033	19.033	19.033	19.033	19.033	19.033	19.033	
	variable Kosten Gesamt	95.167	57.100	19.033	133.233	1.693.967	1.865.267	152.267	285.500	475.833	19.033	4.796.400
11	Anzahl	2	3	4	5	6	7	8	9	10	11	
	variable Kosten je Untersuchung	20.933	20.933	20.933	20.933	20.933	20.933	20.933	20.933	20.933	20.933	
	variable Kosten Gesamt	41.867	62.800	83.733	104.667	125.600	146.533	167.467	188.400	209.333	230.267	1.360.667
12	Anzahl	8	6	4	2	8	3	1	1	2	3	
	variable Kosten je Untersuchung	22.833	22.833	22.833	22.833	22.833	22.833	22.833	22.833	22.833	22.833	
	variable Kosten Gesamt	182.667	137.000	91.333	45.667	182.667	68.500	22.833	22.833	45.667	68.500	867.667
13	Anzahl	6	8	10	12	14	16	18	20	22	6	
	variable Kosten je Untersuchung	24.733	24.733	24.733	24.733	24.733	24.733	24.733	24.733	24.733	24.733	
	variable Kosten Gesamt	148.400	197.867	247.333	296.800	346.267	395.733	445.200	494.667	544.133	148.400	3.264.800
14	Anzahl	2	6	10	14	18	22	26	30	34	9	
	variable Kosten je Untersuchung	26.633	26.633	26.633	26.633	26.633	26.633	26.633	26.633	26.633	26.633	
	variable Kosten Gesamt	53.267	159.800	266.333	372.867	479.400	585.933	692.467	799.000	905.533	239.700	4.554.300
15	Anzahl	3	5	7	9	11	13	15	17	19	21	
	variable Kosten je Untersuchung	28.533	28.533	28.533	28.533	28.533	28.533	28.533	28.533	28.533	28.533	
	variable Kosten Gesamt	85.600	142.667	199.733	256.800	313.867	370.933	428.000	485.067	542.133	599.200	3.424.000
											SUMME	36.834.767

Abbildung 16: Screenshot zur Berechnung der variablen Kosten von Röntgenuntersuchungen

(Quelle: COREmain_DATAinput.xls – eigene Darstellung)

Im zweiten Testfall wurden die variablen Kosten je Untersuchungsart und Röntgengerät einheitlich auf 100 € gesetzt. Die Anzahl der Untersuchungen wurde ebenfalls einheitlich auf 100 gesetzt, wobei jede Untersuchungsart auf je einem Gerät durchgeführt wurde (rRU_{11} = 100, rRU_{22} = 100. rRU_{33} = 100, rRU_{44} = 100, rRU_{55} = 100). Die Anzahl an verschiedenen Röntgengerätetypen wurde auf $m_1 = 1$, $m_2 = 2$, $m_3 = 3$, $m_4 = 4$, $m_5 = 5$ festgelegt. Dies ist in der ursprünglichen Formel nicht enthalten wurde aber bereits in den Quellcode implementiert. Da es die Realität besser widerspiegelt, wurde der Testfall dahingehend modifiziert. Der Erwartungswert lag bei *rKv* = 150.000 €.

Die mühsame Eingabe der Testdaten des ersten Testfalls wurde aufgrund der Komplexität bewusst vollzogen, um die korrekte Einbeziehung aller Daten zu gewährleisten. Das Ergebnis gab zunächst einen falschen Ausgabewert aus, da eine zufällige Anzahl an verschiedenen Gerätetypen bereits voreingestellt war. Dies wurde daraufhin korrigiert und das Ergebnis von *rKv* = 36.834.767 € entsprach exakt dem Erwartungswert. Auch im zweiten Testfall gab die

Ausführung der Software das erwartete Ergebnis von *rKv* = 150.000 € aus (siehe Abbildung 17).

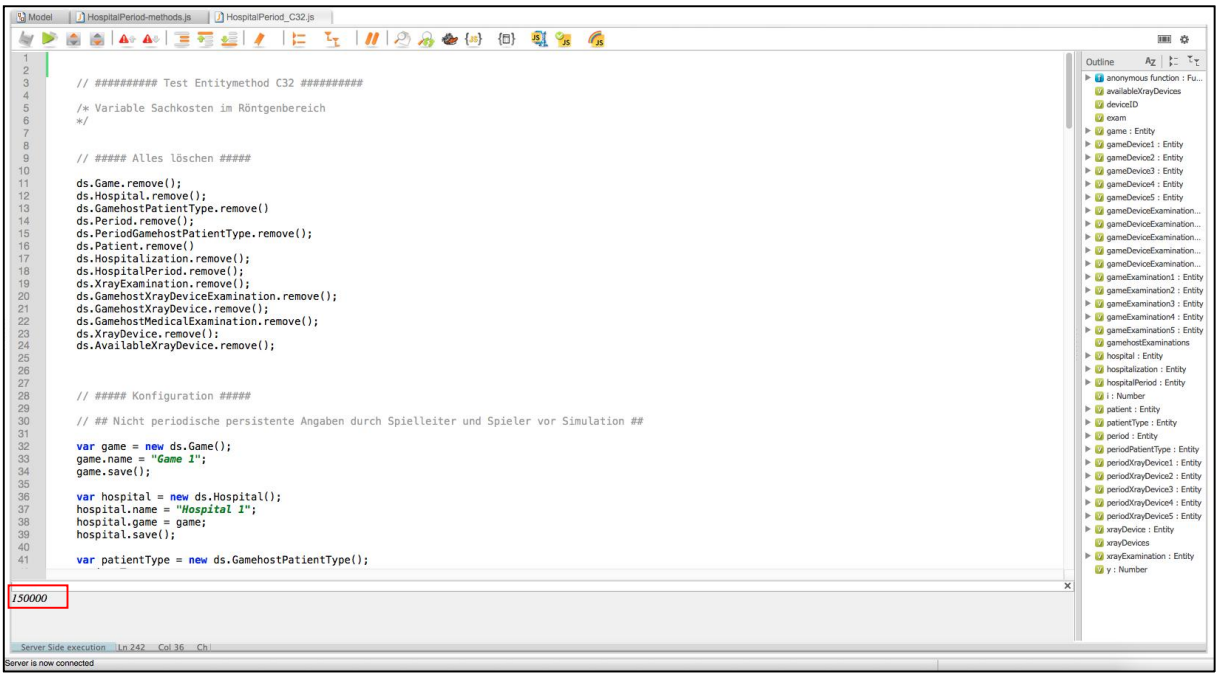


Abbildung 17: Screenshot aus dem Quellcode des COREmain Hospital Planspiels - Ergebnis der Berechnung von C32 (Quelle: Gesslbauer 2015)

Beispiel 2:Variable Kosten im Pflegebereich (C31)

Die variablen Sachkosten im Röntgenbereich (*pKv*) setzen sich aus den variablen Sachkosten eines Patienten in einer bestimmten Pflegestufe ($pKvPat_q$) und der durchschnittlichen Anzahl an Patienten am Tag pro Pflegestufe ($pPat_{dtq}$) zusammen. Unter den variablen Kosten werden hierbei, die von den Patienten verursachten Kosten pro Tag *d* und Pflegestufe *q* definiert. Die Anzahl der Patienten wird ähnlich wie bei der Berechnung des Bettenbelegungsgrads (s. Abschnitt 6.2.1) an drei Zeitpunkten am Tag (Früh, Mittag Abend) gemessen, woraus bei der Division durch die Anzahl an Messzeitpunkten der Durchschnittswert entsteht. Hierbei werden die Patienten den verschiedenen Pflegestufen zugeteilt. Die variablen Kosten im Pflegebereich werden pro Periode berechnet, indem die variablen Sachkosten eines Patienten in einer bestimmten Pflegestufe mit deren durchschnittlichen Anzahl an Patienten dieser Pflegestufe am Tag multipliziert werden. Anschließend werden die Ergebnisse über alle Tage und Pflegestufen aufsummiert.

Die dazugehörige Formel lautet wie folgt:

$$pKv = \sum_{d=1}^{28} \sum_{q=1}^{10} pKvPat_q \times \left(\frac{\sum_{t=1}^{3} pPat_{dtq}}{3} \right)$$

Zur Berechnung des Erwartungswerts wurden zwei Testfälle für jeweils drei Patiententypen konstruiert, welche jeweils eine Äquivalenzklasse bildeten. Für beide Testfälle wurden die variablen Kosten für die Pflegestufe 1 auf $pKvPat_1$ = 1.000 €, für die Pflegestufe 2 auf $pKvPat_2$ = 2.000 € und für die Pflegestufe 3 auf $pKvPat_3$ = 3.000 € festgelegt. Die Anzahl der Patienten variiert in den Testfällen. Im ersten wurden die Werte für 28 Tage und im zweiten für einen Tag gewählt. Der zweite Testfall diente der Überprüfung der dynamischen Programmierung, d.h. ob sie die Summe über alle Tage oder nur über die Eingabewerte bildet. An jedem Tag waren jeweils 100 Patienten in jeder Pflegestufe zu unterschiedlichen Zeitpunkten. Der Erwartungswert für den ersten Testfall war *pKv* = 5.600.000 € und für den zweiten Testfall *pKv* =200.000 €. Der Aufbau des Excel-Files zur Berechnung dieser Formel ist in Abbildung 18 ersichtlich. Hierbei ist nur der weiße und graue Bereich vom Tester auszufüllen. Der graue Bereich enthält die Eigenschaften der Pflegestufen und der weiße Bereich beinhaltet die Anzahl der Patienten zu den verschiedenen Zeitpunkten je Pflegestufe.

	Pflegestufe										Pkv	N
	1	2	3	4	5	6	7	8	9	10		
$pKvPat_q$	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000		
pMiniPfSq	7	15	21	36	37	39	42	45	46	52		
1	100	100	100	-	-	-	-	-	-	-		
1.1	100	-	-	-	-	-	-	-	-	-		
1.2	-	100	-	-	-	-	-	-	-	-		
1.3	-	-	100	-	-	-	-	-	-	-		
Tagesdurchschnitt	33	33	33	-	-	-	-	-	-	-		
N	233	233	233	-	-	-	-	-	-	-		700
kvar pro d und q	33.333	66.667	100.000	-	-	-	-	-	-	-	200.000	

Abbildung 18: Screenshot zur Berechnung der variablen Kosten im Pflegebereich

(Quelle: COREmain_DATAinput.xls – eigene Darstellung)

Die Testdaten wurden für den ersten Testfall für alle 28 Tage in den Quellcode eingegeben. Das Ergebnis entsprach dem Erwartungswert von *pKv* = 5.600.000 €. Für den zweiten Testfall konnten die Eingaben auf einen Tag reduziert werden. Hierbei wurde der Erwartungswert von *pKv* = 200.000 € bestätigt (siehe Abbildung 19).

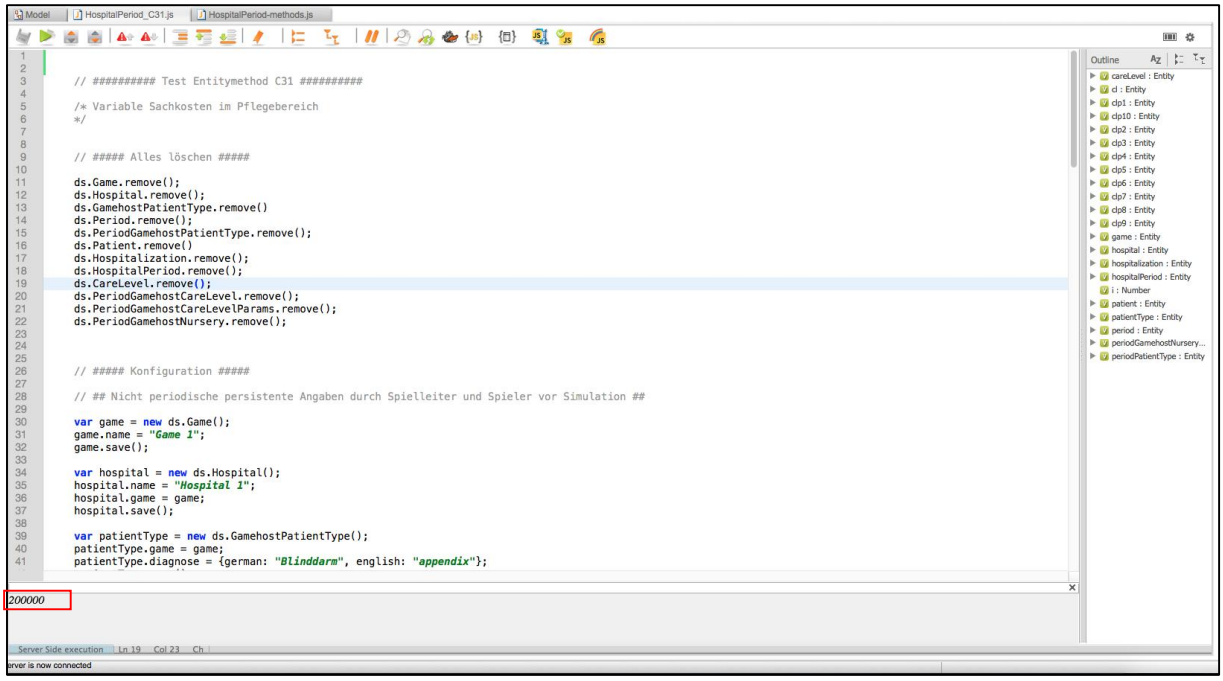


Abbildung 19: Screenshot aus dem Quellcode des COREmain Hospital Planspiels - Ergebnis der Berechnung von C31 (Quelle: Gesslbauer 2015)

Beispiel 3: Abschreibungskosten im Röntgenbereich (C41)

Das zweite Testobjekt, die Berechnung der Abschreibungskosten im Röntgenbereich ($rAbK_p$), setzt sich aus den Anschaffungskosten ($rAKRG_m$), der zum Berechnungszeitpunkt bestehenden Ausstattung, welche aus bereits vorhandenen Röntgengeräten und neu angeschafften Röntgengeräten (rGa_{mp}) besteht, sowie der Lebensdauer der Geräte in Jahren (LD_m) zusammen. Jeder Gerätetyp weist unterschiedliche Anschaffungskosten und Lebensdauern auf. Daher werden diese zur vollständigen Berechnung aufsummiert.

Die dazugehörige Formel lautet wie folgt:

$$rAbK_p = \sum_{m=1}^{10} \frac{rAKRG_m \times rRGa_{mp}}{LD_m \times 12}$$

Die Indizes sind zum einen der bereits in Abschnitt 6.2.3. beschriebene Röntgengerätetyp *m* und die Spielperiode *p*, da die Anschaffung bzw. gegebenenfalls auch Veräußerung der Geräte während des Spiels geschehen kann und sich somit die Abschreibungskosten verändern. Herr Mag. Gesslbauer merkte zudem an, dass die potentiellen Veräußerungskosten eines Gerätes noch nicht berücksichtigt wurden, dies aber in naher Zukunft geschehen wird.

Als Testfälle wurden hierbei verschiedene Eingabewerte für unterschiedliche Geräte gewählt. Um Fehler hierbei zu entdecken, wurden sechs Testfälle erstellt und durchgeführt, welche aus Kombinationen der verschiedenen Eingabewerte bestanden und verschiedene Äquivalenzklassen bildeten. Beispielhaft werden zwei dieser Testfälle vorgestellt.

Die Anschaffungskosten wurden im ersten Testfall einheitlich auf je 100.000 € je Gerätetyp eingestellt. Die Lebensdauer aller Röntgengerätetypen wurde auf fünf Jahre festgelegt. Die Anzahl der vorhandenen und angeschafften Geräte wurde auf fünf Gerätetypen eingeschränkt (rGA_{11} = 3, rGA_{21} = 1, rGA_{31} = 4, rGA_{41} = 3, rGA_{51} = 4) und es wurde die Periode 1 betrachtet. Der Erwartungswert betrug $rAbK_1$ = 25.000 €.

Im zweiten Testfall wurden alle Variablen verändert. Die Anzahl der Gerätetypen blieb auf fünf limitiert. Die Anschaffungskosten wurden wie folgt definiert: $rAKRG_1$ = 120.000 €, $rAKRG_2$ = 150.000 €, $rAKRG_3$ = 100.000 €, $rAKRG_4$ = 190.000 €, $rAKRG_5$ = 125.000 €. Die Lebensdauer wurde für alle Geräte auf sieben Jahre erhöht. Die Anzahl der Geräte wurde für Periode 1 unverändert übernommen. In Periode 2 wurden folgende Veränderungen in der Anzahl der vorhandenen und angeschafften Geräte vorgenommen: rGA_{12} = 3, rGA_{22} = 3, rGA_{32} = 5, rGA_{42} = 4, rGA_{52} = 4. Aufgrund der veränderten Anschaffungswerte und Lebensdauer der Röntgengeräte veränderten sich die Abschreibungskosten in Periode 1 auf $rAbK_1$ = 21.429 €. Die Mengenänderung an Röntgengeräten in Periode 2 zog eine Erhöhung der Abschreibungskosten auf $rAbK_2$ = 27.143 € mit sich.

Bei der Ausführung der eingegebenen Werte in den Quellcode des Planspiels COREmain Hospital wurden der Erwartungswert des ersten Testfalls ($rAbK_1$ = 25.000 €) ausgegeben (s. Abbildung 20). Auch im zweiten Testfall wurde der Erwartungswert jeweils für Periode 1 ($rAbK_1$ = 21.429 €) und für Periode 2 ($rAbK_2$ =27.143 €) korrekt berechnet.

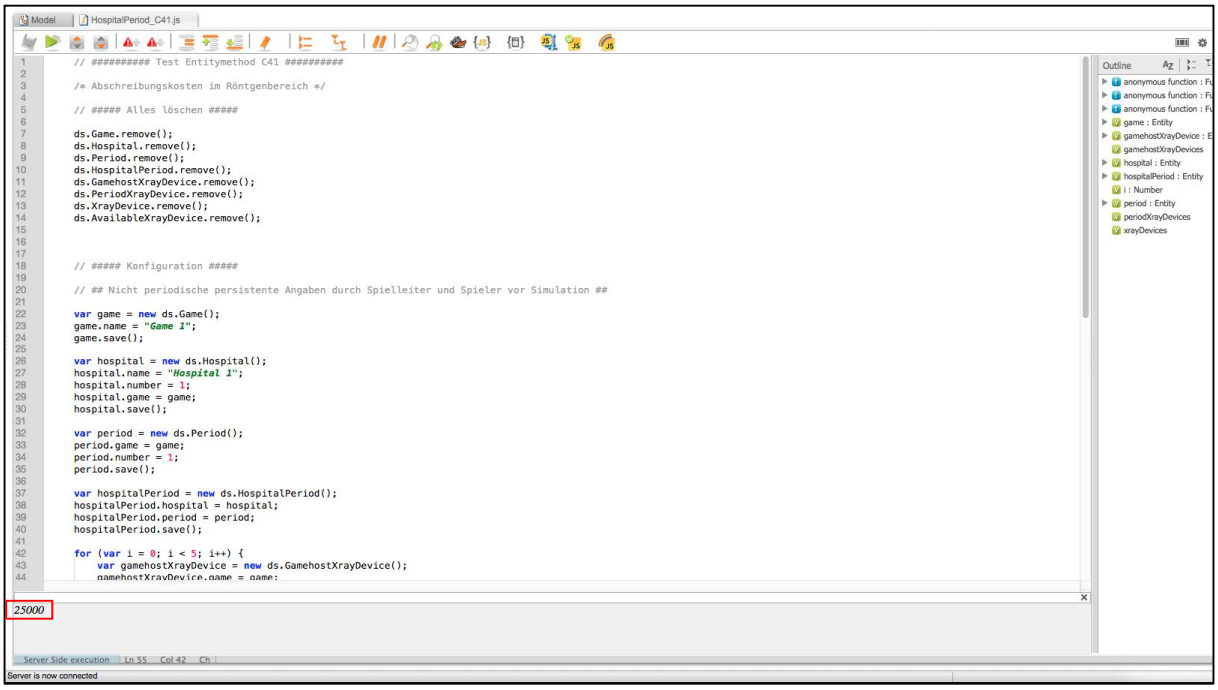


Abbildung 20: Screenshot aus dem Quellcode des COREmain Hospital Planspiels - Ergebnis der Berechnung von C41 (Quelle: Gesslbauer 2015)

6.2.4. Budgetverteilungsfunktion

Die Budgetverteilungsfunktion beinhaltet verschiedene Finanzierungsmodelle der einzelnen Krankenhäuser bzw. Regionen. In diesem Abschnitt werden das Finanzierungsmodell pro Tag sowie die Ermittlung der DRG-Punkte eines Patienten, welcher einem Patiententypen in einem Krankenhaus zugeordnet wird, vorgestellt. Die letztere Berechnung ist eine Teilformel zur Berechnung des Finanzierungsmodells pro Fall mit unlimitiertem Budget in Österreich. Die Auswahl des Finanzierungsmodells wird zu Beginn des Planspiels vom Spielleiter festgelegt.

Beispiel 1: Finanzierungsmodell pro Tag (D10)

Das Finanzierungsmodell pro Tag setzt sich aus der Multiplikation von einem zuvor definierten Tagessatz (*TS*) und der tatsächlichen Verweildauer jedes einzelnen Patienten *i* ($pVDt_{ik}$) in einem Krankenhaus zusammen. Die Summe über alle Patienten im Krankenhaus *k* ist das Budget des Krankenhauses (kB_k), d.h. das Krankenhaus erhält für jeden Patienten, egal welches Krankheitsbild bzw. Patientenpfad er aufweist, einen Tagessatz auf Basis der Aufenthaltsdauer im Krankenhaus. Die anfallenden Untersuchungskosten sind dabei unerheblich. Das Ergebnis, das daraus resultierende Budget, wird für alle Krankenhäuser einer Region einzeln ausgegeben.

Die dazugehörige Formel lautet wie folgt:

$$kB_k = \sum_{i=1}^{I} pVDt_{ik} \times TS$$

Die in Abschnitt 6.2.2. beschriebene Problematik in der Berechnung der tatsächlichen Verweildauer eines Patienten wurde hierbei mit Hilfe der Eingabe von ausschließlich richtigen Werten umgangen.

Zur Erstellung eines Datensatzes wurde die tatsächliche Verweildauer von 100 Patienten (I = 100) eingegeben und diese willkürlich einem von sechs Krankenhäusern zugeteilt. Der Tagessatz betrug 150 €/Tag. Für die Implementierung in den Quellcode des Planspiels COREmain Hospitals wurde die tatsächliche Verweildauer für jeweils zehn Patienten in Sekunden ausgerechnet. Hierbei wurde aufgrund der zuvor beschriebenen Problematik beachtet, dass ausschließlich ganze Tage gewählt wurden. Die Erwartungswerte der einzelnen Krankenhäuser sind in Abbildung 21 ersichtlich.

D10	Budget des Krankenhauses k **Finanzierungsmodell pro Tag**	kB_k	1	122.550
			2	125.700
			3	133.650
			4	130.500
			5	148.200
			6	135.000

Abbildung 21: Erwartungswerte des Finanzierungsmodells pro Tag

(Quelle: COREmain_DATAinput.xls – eigene Darstellung)

Die Ausführung des Tests im Planspiel COREmain Hospital wurde auf die ersten beiden Krankenhäuser beschränkt, da bei der Ausgabe des korrekten Ergebnisses, die Richtigkeit der Formel auf die anderen Krankenhäuser übertragbar ist und jeweils als Äquivalenzklassen angesehen werden können. Dies konnte durch die Ausführung mit den Ergebnissen kb_1 = 122.550 € und kb_2 = 125.700, welche den Erwartungswert widerspiegeln, bestätigt werden.

Beispiel 2: Ermittlung der DRG-Punkte eines Patienten mit einem Patiententyp in einem Krankenhaus (D31)

Das zweite Testobjekt ermittelt die DRG-Punkte eines Patienten, welcher einem Patiententyp in einem Krankenhaus zuordnet wird. Der Aufbau gestaltet sich deutlich komplexer als die bisherigen Formeln. Die Ermittlung der DRG-Punkte eines Patienten ist der erste Schritt zur Berechnung des Finanzierungsmodells pro Fall mit unlimitiertem Budget für Österreich. Im

Weiteren werden anschließend die DRG-Punkte des Krankenhauses, in dem sich der Patient befindet, ermittelt um abschließend das Budget des Krankenhauses zur Berechnen. Die Berechnung dieses Finanzierungsmodells basiert, wie in Abschnitt 6.1.4. beschrieben, auf der leistungsorientierten Krankenanstaltenfinanzierung, welche vom Österreichischen Bundesministerium für Gesundheit, Familie und Jugend im Jahre 1997 durch- bzw. eingeführt wurde und jährlich angepasst wird.[69] Zudem existieren weitere fallbasierte Finanzierungsmodelle mit limitiertem Budget und adäquat dazu, angepasste Formeln für das deutsche Gesundheitswesen, welche vom Spielleiter zu Beginn ausgewählt werden können.

Die Wahl der Formel zur Berechnung der DRG-Punkte eines Patienten ($DRGP_{ijk}$) ist abhängig von der tatsächlichen Verweildauer ($pVDt_{ijk}$), der Belagsdaueruntergrenze ($BDUG_j$) und der Belagsdauerobergrenze ($BDOG_j$). Die Indizes *i* bezeichnen den Patienten, *k* das Krankenhaus und *j* den Patiententyp.

Daraus resultieren drei verschiedene Bedingungen bezüglich der Anwendung der Formel (siehe Abbildung 22). Im Falle, dass die tatsächliche Verweildauer, kleiner gleich der Belagsdauerobergrenze und größer gleich der Belagsdaueruntergrenze ist, sind die DRG-Punkte des Patienten mit den DRP-Punkten des Patiententyps gleichzusetzten. Ist die tatsächliche Verweildauer des Patienten kleiner der Belagsdaueruntergrenze, wird von den DRP-Punkten des Patiententyps die Differenz der Belagsdaueruntergrenze und der tatsächlichen Verweildauer mit dem Punkteabschlag pro Tag des jeweiligen Patiententyps (PAb_j) multipliziert und anschließend abgezogen. Die dritte Formel wird bei einer tatsächlichen Verweildauer größer der Belagsdauerobergrenze angewendet. Hierbei wird die Belagsdauerobergrenze durch die Anzahl der vom Patienten verbrachten Tage im Krankenhaus (*dd*) dividiert. Dies geschieht solange bis Belagsdauerobergrenze, welche jeweils um eins erhöht wird, die tatsächliche Verweildauer erreicht. Die Summe dieser Werte wird mit dem Punktezuschlag pro Tag des jeweiligen Patiententyps (PZu_j) multipliziert und abschließend zu den DRG-Punkten des Patiententyps addiert. Graphisch kann man sich die Formel wie folgt vorstellen: stetige Steigung vom Nullpunkt (für $pVDt_{ijk} < BDUG_j$) bis zu den DRG-Punkten des Patiententyps, daraufhin ein horizontaler Verlauf (für $BDUG_j \leq pVDt_{ijk} \geq BDOG_j$) und anschließend eine degressive Steigung (für $pVDt_{ijk} > BDOG_j$).

[69] Vgl. Bundesministerium für Gesundheit, Familie und Jugend: Leistungsorientierte Krankenanstaltenfinanzierung - LKF Modell (2008), S. 5

Formel		
$DRGP_{ijk}$ =	DRGPj	falls $BDUG_j \leq pVDt_{ijk} \leq BDOG_j$
$DRGP_{ijk}$ =	$DRGP_j - (BDUG_j - pVDt_{ijk}) \times PAb_j$	falls $pVDt_{ijk} < BDUG_j$
$DRGP_{ijk}$ =	$DRGP_j + \left(\sum_{dd=BDOG_j+1}^{pVDt_{ijk}} \frac{BDOG_j}{dd} \right) \times PZu_j$	falls $pVDt_{ijk} > BDOG_j$

Abbildung 22: Formel zur Berechnung des Finanzierungsmodells pro Fall mit unlimitiertem Budget (Österreich)

(Quelle: Markus Kraus)

Aufgrund der drei Bedingungen wurden für jede Formel zwei Testfälle konstruiert. Hierbei wurde das Verfahren der Grenzwertanalyse angewendet, um mögliche Fehler an den Rändern aufzudecken. Für die erste Bedingung wurde die tatsächliche Verweildauer mit der Belagsdauerobergrenze sowie mit der Belagsdaueruntergrenze gleichgesetzt ($pVDt_{111}$ = 5 = $BDUG_1$ und $pVDt_{211}$ = 8 = $BDOG_1$). Die Wahl der Indizes war hierbei nicht von Bedeutung. Der Erwartungswert war in beiden Fällen 15, da dies den zuvor willkürlich gewählten DRG-Punkten von Patiententyp 1 entsprach. Abbildung 23 veranschaulicht die Berechnung des Erwartungswerts in Excel. Hierbei wird ausschließlich die Nummer des Patienten eingegeben und die benötigten Daten werden aus den jeweiligen Eingabeblättern ausgelesen.

D31	Ermittlung der DRG-Punkte des Patienten i mit Patiententyp j im Krankenhaus k (Version Österreich)	**Patient i**	1	**Eingabe erforderlich**
			Werte des Patienten und Patiententyps aus DATA_INPUT	
		$pVDt_{ijk}$	5	Tatsächliche Verweildauer des Patienten i mit Patiententyp j im Krankenhaus k, i = 1, ..., I; j = 1, ..., 15; k = 1, ..., 6
		Krankenhaus	1	
		Patiententyp	1	
		$BDOG_j$	8	Belagsdauerobergrenze des Patiententyps j, j = 1, ..., 15
		$BDUG_j$	5	Belagsdaueruntergrenze des Patiententyps j, j = 1, ..., 15
		DRGPj	15	DRG-Punkte des Patiententyps j, j = 1, ..., 15
		PZu_j	35	Punktezuschlag pro Tag im Patiententyp j, j = 1, ..., 15
		PAb_j	50	Punkteabschlag pro Tag im Patiententyp j, j = 1, ..., 15
		$BDOG_{j+1}$	-	
		dd	5	
		$DRGP_{ijk}$ =	**15**	

Abbildung 23: Erwartungswerte der Ermittlung von DRG-Punkten des Patienten 1

(Quelle: COREmain_DATAinput.xls – eigene Darstellung)

Für die zweite Bedingung wurden ebenfalls zwei Testfälle erstellt. Hierbei wurde ein Wert nahe Null und ein Wert nahe der Belagsdaueruntergrenze gewählt ($pVDt_{322}$ = 1 und $pVDt_{422}$ = 3 bei $BDUG_2$ = 4 und $DRGP_2$ = 12 und PAb_2 = 2). Der Erwartungswert für $pVDt_{322}$ war $DRGP_{322}$ = 6 (siehe Abbildung 24) und für $pVDt_{422}$ lag er bei $DRGP_{422}$ = 10.

D31	Ermittlung der DRG-Punkte des Patienten i mit Patiententyp j im Krankenhaus k (Version Österreich)	**Patient i**	3	**Eingabe erforderlich**
			Werte des Patienten und Patiententyps aus DATA_INPUT	
		$pVDt_{ijk}$	1	Tatsächliche Verweildauer des Patienten i mit Patiententyp j im Krankenhaus k, i = 1, ..., I; j = 1, ..., 15; k = 1, ..., 6
		Krankenhaus	2	
		Patiententyp	2	
		$BDOG_j$	7	Belagsdauerobergrenze des Patiententyps j, j = 1, ..., 15
		$BDUG_j$	4	Belagsdaueruntergrenze des Patiententyps j, j = 1, ..., 15
		DRGPj	12	DRG-Punkte des Patiententyps j, j = 1, ..., 15
		PZu_j	3	Punktezuschlag pro Tag im Patiententyp j, j = 1, ..., 15
		PAb_j	2	Punkteabschlag pro Tag im Patiententyp j, j = 1, ..., 15
		$BDOG_{j+1}$	-	
		dd	1	
		$DRGP_{ijk}$ =	**6**	

Abbildung 24: Erwartungswerte der Ermittlung von DRG-Punkten des Patienten 3

(Quelle: COREmain_DATAinput.xls – eigene Darstellung)

Die Testfälle der dritten Bedingung wurden ähnlich zu den beiden vorherigen aufgestellt. Die Werte für die tatsächliche Verweildauer des Patienten wurden zum einem sehr nahe und zum anderen sehr weit entfernt von der Belagsdauerobergrenze gewählt ($pVDt_{533}$ = 10 und $pVDt_{633}$ = 25 bei $BDOG_2$ = 9 und $DRGP_3$ = 26 und PZu_3 = 8). Zur besseren Übersicht wurden in diesen Testfällen die Patiententypen 3 sowie das Krankenhaus 3 eingestellt. Der Erwartungswert lag für $pVDt_{533}$ bei $DRGP_{533}$ = 33,2 und für $pVDt_{633}$ bei $DRGP_{633}$ = 110,48 (s. Abbildung 25).

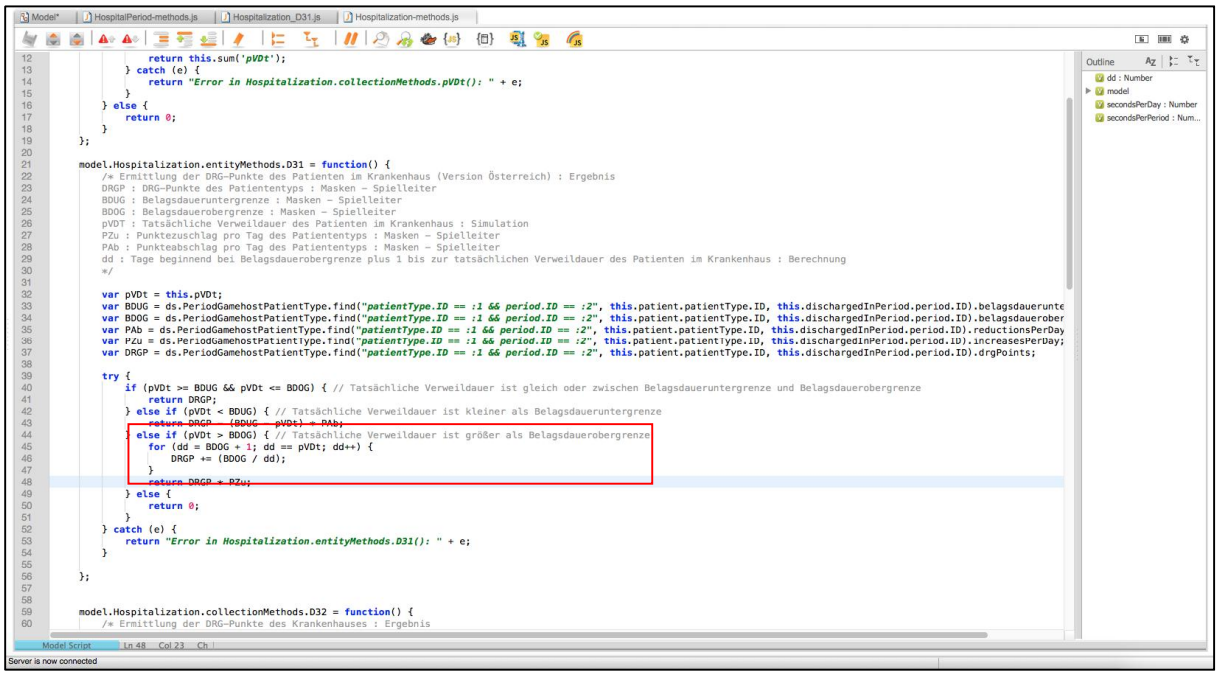
D31	Ermittlung der DRG-Punkte des Patienten i mit Patiententyp j im Krankenhaus k (Version Österreich)	**Patient i**	6	**Eingabe erforderlich**
			Werte des Patienten und Patiententyps aus DATA_INPUT	
		$pVDt_{ijk}$	25	Tatsächliche Verweildauer des Patienten i mit Patiententyp j im Krankenhaus k, i = 1, ..., I; j = 1, ..., 15; k = 1, ..., 6
		Krankenhaus	3	
		Patiententyp	3	
		$BDOG_j$	9	Belagsdauerobergrenze des Patiententyps j, j = 1, ..., 15
		$BDUG_j$	3	Belagsdaueruntergrenze des Patiententyps j, j = 1, ..., 15
		DRGPj	26	DRG-Punkte des Patiententyps j, j = 1, ..., 15
		PZu_j	8	Punktezuschlag pro Tag im Patiententyp j, j = 1, ..., 15
		PAb_j	5	Punkteabschlag pro Tag im Patiententyp j, j = 1, ..., 15
		$BDOG_{j+1}$	-	
		dd	25	
		$DRGP_{ijk}$ =	**110,48**	

Abbildung 25: Erwartungswerte der Ermittlung von DRG-Punkten des Patienten 6

(Quelle: COREmain_DATAinput.xls – eigene Darstellung)

Für die Testfälle der ersten beiden Bedingungen entsprachen die Erwartungswerte den bei der Ausführung der Software erzielten Ergebnissen ($DRGP_{111}$ = 15, $DRGP_{211}$ = 15, $DRGP_{322}$ = 6, $DRGP_{422}$ = 10).

Dies war bei der Testung der dritten Bedingung nicht der Fall. Daraufhin wurde die Formel auf ihre korrekte Implementierung hin untersucht und es wurde festgestellt, dass ein Fehler aufgetreten ist. Die Zeilen 46-48 in Abbildung 26 zeigen die Programmierung der Formel im Quellcode. Hierbei fällt auf, dass durch Unterteilung der Formel in zwei Teilbereiche, die mathematische „Punkt vor Strich-Rechnung“ nicht korrekt durchgeführt wird.

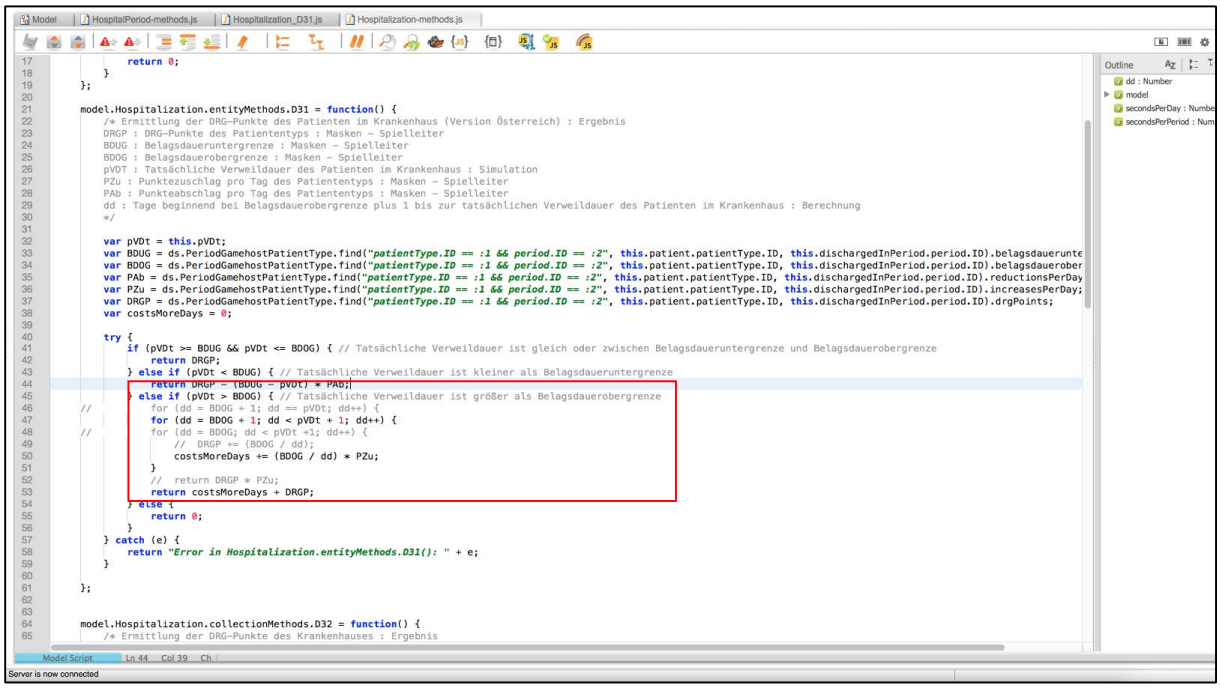


```
            return this.sum('pVDt');
        } catch (e) {
            return "Error in Hospitalization.collectionMethods.pVDt(): " + e;
        }
    } else {
        return 0;
    }
};

model.Hospitalization.entityMethods.D31 = function() {
    /* Ermittlung der DRG-Punkte des Patienten im Krankenhaus (Version Österreich) : Ergebnis
    DRGP : DRG-Punkte des Patiententyps : Masken - Spielleiter
    BDUG : Belagsdaueruntergrenze : Masken - Spielleiter
    BDOG : Belagsdauerobergrenze : Masken - Spielleiter
    pVDT : Tatsächliche Verweildauer des Patienten im Krankenhaus : Simulation
    PZu : Punktezuschlag pro Tag des Patiententyps : Masken - Spielleiter
    PAb : Punkteabschlag pro Tag des Patiententyps : Masken - Spielleiter
    dd : Tage beginnend bei Belagsdauerobergrenze plus 1 bis zur tatsächlichen Verweildauer des Patienten im Krankenhaus : Berechnung
    */

    var pVDt = this.pVDt;
    var BDUG = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).belagsdauerunte
    var BDOG = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).belagsdauerober
    var PAb = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).reductionsPerDay
    var PZu = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).increasesPerDay;
    var DRGP = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).drgPoints;

    try {
        if (pVDt >= BDUG && pVDt <= BDOG) { // Tatsächliche Verweildauer ist gleich oder zwischen Belagsdaueruntergrenze und Belagsdauerobergrenze
            return DRGP;
        } else if (pVDt < BDUG) { // Tatsächliche Verweildauer ist kleiner als Belagsdaueruntergrenze
            return DRGP - (BDUG - pVDt) * PAb;
        } else if (pVDt > BDOG) { // Tatsächliche Verweildauer ist größer als Belagsdauerobergrenze
            for (dd = BDOG + 1; dd == pVDt; dd++) {
                DRGP += (BDOG / dd);
            }
            return DRGP + PZu;
        } else {
            return 0;
        }
    } catch (e) {
        return "Error in Hospitalization.entityMethods.D31(): " + e;
    }

};


model.Hospitalization.collectionMethods.D32 = function() {
    /* Ermittlung der DRG-Punkte des Krankenhauses : Ergebnis
```

Abbildung 26: Screenshot aus dem Quellcode des COREmain Hospital Planspiels – Fehler in der Formelberechnung (Quelle: Gesslbauer 2015)

Daraufhin wurde die Formel von Herrn Mag. Gesslbauer neu programmiert. Dafür wurde eine zusätzliche Hilfsvariable „costsMoreDays“ kreiert, welche den zweiten Teil der Formel zuerst berechnet um mögliche neue Fehlerwirkungen vorzubeugen (siehe Abbildung 27 [Zeilen 49-53]).

Anschließend wurden die Testfälle wiederholt und lieferten abermals nicht das zu erwartende Ergebnis. Bei abermaliger Betrachtung konnte festgestellt werden, dass die Schleife zur Berechnung der in Klammern stehenden Summenformel ebenfalls fehlerhafte Ergebnisse ausgab. Aufgrund dessen wurde die Schleife im Quellcode mehrmals korrigiert (siehe Abbildung 27 [Zeilen 46-48]).



```
            return 0;
        }
    };

    model.Hospitalization.entityMethods.D31 = function() {
        /* Ermittlung der DRG-Punkte des Patienten im Krankenhaus (Version Österreich) : Ergebnis
        DRGP : DRG-Punkte des Patiententyps : Masken - Spielleiter
        BDUG : Belagsdaueruntergrenze : Masken - Spielleiter
        BDOG : Belagsdauerobergrenze : Masken - Spielleiter
        pVDT : Tatsächliche Verweildauer des Patienten im Krankenhaus : Simulation
        PZu : Punktezuschlag pro Tag des Patiententyps : Masken - Spielleiter
        PAb : Punkteabschlag pro Tag des Patiententyps : Masken - Spielleiter
        dd : Tage beginnend bei Belagsdauerobergrenze plus 1 bis zur tatsächlichen Verweildauer des Patienten im Krankenhaus : Berechnung
        */

        var pVDt = this.pVDt;
        var BDUG = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).belagsdauerunte
        var BDOG = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).belagsdauerober
        var PAb = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).reductionsPerDay
        var PZu = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).increasesPerDay;
        var DRGP = ds.PeriodGamehostPatientType.find("patientType.ID == :1 && period.ID == :2", this.patient.patientType.ID, this.dischargedInPeriod.period.ID).drgPoints;
        var costsMoreDays = 0;

        try {
            if (pVDt >= BDUG && pVDt <= BDOG) { // Tatsächliche Verweildauer ist gleich oder zwischen Belagsdaueruntergrenze und Belagsdauerobergrenze
                return DRGP;
            } else if (pVDt < BDUG) { // Tatsächliche Verweildauer ist kleiner als Belagsdaueruntergrenze
                return DRGP - (BDUG - pVDt) * PAb;
            } else if (pVDt > BDOG) { // Tatsächliche Verweildauer ist größer als Belagsdauerobergrenze
//              for (dd = BDOG + 1; dd == pVDt; dd++) {
                for (dd = BDOG + 1; dd < pVDt + 1; dd++) {
//              for (dd = BDOG; dd < pVDt +1; dd++) {
                    //  DRGP += (BDOG / dd);
                    costsMoreDays += (BDOG / dd) * PZu;
                }
                //  return DRGP * PZu;
                return costsMoreDays + DRGP;
            } else {
                return 0;
            }
        } catch (e) {
            return "Error in Hospitalization.entityMethods.D31(): " + e;
        }
    };


    model.Hospitalization.collectionMethods.D32 = function() {
        /* Ermittlung der DRG-Punkte des Krankenhauses : Ergebnis
```

Abbildung 27: Screenshot aus dem Quellcode des COREmain Hospital Planspiels – Formelkorrektur von D31 (Quelle: Gesslbauer 2015)

Nachdem die Korrekturen im Quellcode implementiert wurden, wurden die Testfälle abermals zur Ausführung gebracht. Dabei konnten diese den Erwartungswert als das richtige Ergebnis für $DRGP_{533} = 33{,}2$ und für $DRGP_{633} = 110{,}48$ bestätigen.

7. Schlussbetrachtung

In diesem Kapitel werden die wichtigsten Ergebnisse des Blackbox-Tests für das Planspiel COREmain Hospital zusammengefasst. Zudem wird ein Ausblick gegeben, wie die weitere Vorgehensweise in Bezug auf das Release des Planspiels aussieht.

Der gesamte Testaufbau zielte auf die Überprüfung der zur Verfügung stehenden Formelsammlung in Bezug auf deren Implementierung in den Quellcode ab. Hierzu wurden je Testobjekt mindestens zwei Testfälle konstruiert. Anschließend wurden die Excel-Files mit Werten befüllt, um Erwartungswerte zu ermitteln. Dieselben Eingabewerte wurden während des Tests in den Quellcode eingegeben und anschließend die Software zur Ausführung gebracht. Die daraus resultierenden Ergebnisse wurden mit den Erwartungswerten verglichen. Von einem erfolgreichen Test des Testobjekts bzw. korrekter Implementierung wird bei einem identischen Ergebnis bei der Ausführung der Software und dem Erwartungswert gesprochen. Sobald dies nicht der Fall war, wurde die Ursache für den aufgetretenen Fehler, entweder in der Testerstellung im Excel-File oder im Quellcode ausfindig gemacht. In Summe wurden 84 Formeln getestet, wobei sich die Komplexität der Formeln zueinander unterschied. Die Anzahl der korrekt in den Quellcode implementieren Formeln betrug 76. Lediglich bei acht Formeln mussten zum Teil kleinere Korrekturen vorgenommen werden. Daraus lässt sich folgern dass über 90 % davon korrekt in den Quellcode implementiert wurden. Einige wurden mit Hilfe dieses Softwaretest korrigiert und unterstreichen die Notwendigkeit eines Testverfahrens bereits während der Programmierung, um frühzeitig Fehler beheben zu können, welche bei späterer Entdeckung eventuell weitere Fehler zur Folge hätten. Komplexere Fehler, wie beispielsweise die Berechnung der tatsächlichen Verweildauer, haben gezeigt, dass neue Konzepte nötig sind, um diese Fehler zu beheben. Zudem war es empfehlenswert, einen unabhängigen Tester für die Testdurchführung zu gewinnen, welcher die Testung aufgrund seiner Außensicht objektiver als der Entwickler konzipieren kann.

Diese Arbeit liefert die Grundlage für spätere Testverfahren. Die erstellten Excel Dateien wurden dem Entwickler nach der Testung überlassen, um vor dem Release weitere notwendige Tests durchzuführen zu können. Die Einbettung realitätsgetreuer Daten können die Überprüfung weiter verbessern. Derzeit befindet sich die Software in der Finalisierungsphase.

Der heutige Trend geht zunehmend in Richtung der Simulation von Entscheidungen bevor diese in die Realität umgesetzt werden, um Fehler mit großen Folgen vorzubeugen. In den Medien wird immer mehr über die Ersetzung von klassischen Lehrmitteln durch neue Technologien diskutiert oder werden in der Forschung Spiele im nicht-spielrelevanten Kontext genutzt, auch „Gamification“ genannt. Erste Konzepte wurden bereits im E-Learning implementiert und finden immer größere Anwendung. Daher ist anzunehmen, dass Planspiele zunehmend an Bedeutung gewinnen. Der Nutzen dieser ist nicht nur für den Lehrbereich, sondern auch für Entscheidungsträger offensichtlich. Im Rahmen des Planspiels COREmain Hospital könnten zudem Mitarbeiter des Gesundheitsministeriums die Auswirkungen ihrer Gesundheitspolitik simulieren und einen Nutzen daraus ziehen. Ebenso ist es für Studierende sehr hilfreich.

Weiterhin könnten die erstellten Formeln auf ihren logischen Aufbau hin überprüft werden, um somit die Softwarequalität auf Basis ihrer richtigen Umsetzung zu bestätigen. Außerdem ist die Erhebung und Implementierung von realen Kosten in allen Bereichen hilfreich und notwendig, um das Planspiel besser an die Realität anzupassen. Ein zusätzlicher Faktor, der unabdingbar mit dem Erfolg zusammenhängt, ist, dass die Software eine gute Usability aufweist und nicht nur Daten liefert, sondern diese gut aufbereitet, um tatsächlich Handlungsempfehlungen ablesen zu können. Die Durchführung eines solchen Tests ist aber erst nach Fertigstellung des Planspiels möglich.

Literaturverzeichnis

Bundesministerium für Gesundheit, Familie und Jugend (2008): *Leistungsorientierte Krankenanstaltenfinanzierung - LKF Modell.*

Bundesministerium für Gesundheit, Familie und Jugend (2008): *Leistungsorientierte Krankenanstaltenfinanzierung - LKF Systembeschreibung.*

Gesslbauer, J., Rauner, M.S., Kraus, M., Schwarz, S. (2008): *Entwicklung eines internet-basierten Planspiels mit integrierter Konkurrenzsituation und unterschiedlichen Patientenvergütungssystemen*. 2. Forschungsforum der österreichischen Fachhochschulen, 146-151

Jobst, A. (2008): *Ausgewählte Patientenpfade für das internetbasierende multiple Krankenhausplanspiel COREmain Hospita*l. Diplomarbeit, Universität Wien, Wien.

Klaus, F. (2007): *Handbuch zum Testen von Web-Applikationen: Testverfahren, Werkzeuge, Praxistipps*. Berlin/Heidelberg: Springer Verlag.

Kraus, M., Rauner, M., Schwarz, S., Gesslbauer, J. (2012): *Formelsammlung COREmain Hospital*, Stand 2012, Wien.

Kraus, M. (2012): *Strategic Benchmarking for Hospitals*. Dissertation, Universität Wien, Wien.

Kraus, M., Rauner, M. S., Schwarz, S. (2010): *Hospital management games: a taxonomy and extensive review.* Central European Journal of Operation Research 18 (4), S. 567–591.

Rauner, M. S., Kraus, M., Schwarz, S. (2008): Competition under different reimbursement systems: The concept of an internet-based hospital management game. *European Journal of Operational Research* 185 (3), S. 948–963.

Rosen, K. R. (2008): *The history of medical simulation*. Journal of Critical Care 23 (2), S. 157–166.

Seo, K. I.,Choi, E. M. (2006): *Comparison of Five Black-box Testing Methods for Object-Oriented Software*. Software Engineering Research, Management and Applications, 2006. Fourth International Conference on , vol., no., pp.213,220, 9-11
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1691383&isnumber=35650 (Zugriff am 03.04.2015).

Spillner, A.; Linz, T. (2012): *Basiswissen Softwaretest*, 5. Auflage, dpunkt Verlag.

Spillner, A.(2013): *Warum White-Box-Test kein Test ist,* URL: http://pi.informatik.uni-siegen.de/stt/33_4/01_Fachgruppenberichte/TAV/ 05_Spillner.pdf (Zugriff am 26.04.2015).

Wachabauer, D. (2014): *Patientenpfade und Diagnosetechnologien für das Planspiel COREmain Hospital*. Masterarbeit, Universität Wien, Wien.

Anhang

Zusammenfassung

Der Bedeutung von Planspielen scheint von Jahr zu Jahr zuzunehmen. Zu Beginn wurden diese zur Lehr- und Bildungszwecken eingesetzt aber deren Anwendungsgebiete scheinen deutlich weiter zu gehen. Mittlerweile dienen Planspiele zudem zur Entscheidungsfindung für komplexe Problemstellungen. Das Planspiel COREmain Hospital könnte nach der Fertigstellung von Entscheidungsträgern in Krankenhäuser und auch von Studenten genutzt werden.

Die vorliegende Masterarbeit bietet dem Leser eine Einführung die Grundbegriffe des Softwaretests und eine übersichtliche Darstellung der damit verbundenen Testverfahren. Hierin wird die Erstellung sowie die Durchführung eines Blackbox-Tests für das Planspiel COREmain Hospital beschrieben, welche einen Mehrwert hauptsächlich für den Programmierer mit sich bringt, da Fehler frühzeitig entdeckt wurden. Die endeckten Fehler wurden teilweise bereits ausgebessert, jedoch sind für einige Fehler neue Konzepte nötig.

Zudem wird dem interessierten Leser veranschaulicht, wie umfangreich die Erstellung einer Software für das Gesundheitswesen ist und welches detaillierte Know-How aus verschiedenen Bereichen dafür nötig ist. Die Arbeit baut auf einer Reihe von vorherigen Arbeiten auf, welche alle ihren eigenen Teil zum Erfolg des Planspiels beigetragen.

Lebenslauf

PERSÖNLICHE DATEN

Name	Christopher Böhm
Geburtsdatum	14.07.1987
Geburtsort	Berlin

BERUFLICHER WERDEGANG

Seit 07/2013 **"SIEMENS Aktiengesellschaft", Wien, ÖSTERREICH**
Werksstudent
- Unterstützung des kaufmännischen Angebotsleiters bei der Erstellung der Angebotskalkulation und Angebotsdokumenten
- Unterstützung des Projektleiters der U-Bahn München bei der Kundenberichterstattung sowie bei Managementreportings
- Administrative und organisatorische Aufgaben

06/2014 – 12/2014 **"SIEMENS Malaysia Sdn. Bhd.", Kuala Lumpur, MALAYSIEN**
Projekt Management / Delegation
- Projektcontrolling
- Erstellung einer kaufmännischen Bewertung des „Technologie- und Knowhowtransfers"
- Koordination der Währungsaktivitäten mit dem lokalen Partner und relevanten internen Abteilungen
- Analyse und Verbesserung des Materialbeschaffungsprozess

07/2012- 11/2012 **"WebMedian GmbH", Wien, ÖSTERREICH**
Social media Analyst / Sales
Kundenpräsentationen und Erstellung von Social Media Analysen

11/2011-08/2012 **"Orange Austria GmbH", Wien, ÖSTERREICH**
Verkaufs- und Kundenberater

05/2008 **"Irish Welcome Tours", Dublin, IRLAND**
Praktikum
Vorbereitung und Ausführung von Incentives

2008-2010 **"DOCKX Marketing", München, DEUTSCHLAND**
Organisation von internationalen Kongress und Messen

AUSBILDUNG

2012 – 06/2015	Universität Wien, Österreich Master of Science - Betriebswirtschaftslehre
2008 – 2012	Universität Wien, Österreich Bachelor of Science - Betriebswirtschaftlehre
2004 – 2008	Theodolinden-Gymnasium München, Deutschland
2000 – 2004	Oberhachinger Gymnasium, Deutschland
1997 – 2000	Max-Planck-Gymnasium, München – Pasing, Deutschland

SPRACH- UND COMPUTERKENNTNISSE

Sprachen	Deutsch	(Muttersprache)
	Englisch	(Verhandlungssicher)
	Niederländisch	(sehr gute Kenntnisse)
	Französisch	(Grundkenntnisse)
Computer	MS Office (Word, PowerPoint, Outlook und Excel) SAP PM 22 SQL databases HTML & PHP ADONIS / ADOscore / ADOit	

INTERSSEN UND SOZIALES ENGAGEMENT

Politik, Wirtschaft, verschieden Kulturen und Sport

Jugendfußballtrainer beim TSV Grünwald (01/2003 – 07/2008)

Englisch Lehrer in einem “Summer Day Camp” für Kinder (2006 – 2008)

Mitarbeit im Ferienprogramm der Gemeinde Grünwald (07/2004 – 10/2007)