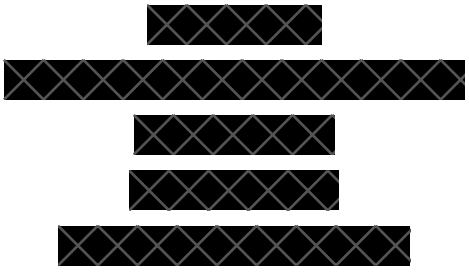
Assignment zum Modul SWE02

Thema: Erstellen eines Testplanes für das Unternehmen S-W-E



Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung

Softwarequalität begegnet uns tagtäglich in unserem Alltag. Wir sind umgeben von Programmen und softwaregesteuerten Geräten denen wir teilweise unser Leben anvertrauen. Ein Auto besitzt als Beispiel Fahrerassistenzsysteme die Beschleunigungs- und Bremsvorgänge steuern, auf deren Funktionalität wir uns verlassen. Gerade von der Automobilindustrie wird des Öfteren in den Medien oder Fachzeitschriften berichtet, dass zum Beispiel Softwarepannen von Steuergeräten, Navigationssystemen oder dem Motormanagement auftreten. Durch fehlerhafte Software entstehen hohe Kosten, Verärgerungen von Kunden treten auf oder ganze Rückrufaktionen werden gestartet.[1] Programme können aber auch leicht zu bedienen sein und zuverlässig laufen. Solche Programme haben zuvor ein ausgearbeitetes Software-Qualitäts-Konzept durchlaufen, welches auf die jeweilige Software angepasst wurde.[2]

1.2 Ziel und Aufbau der Arbeit

Ziel dieser Arbeit ist es einen Testplan für eine Individualsoftware der Firma S-W-E zu erstellen. Mittels dieser Software kann die Firma einen Mehraufwand von 50000 Euro im Jahr einsparen. Die Voraussetzung dafür ist, dass durch die erstellten Testfälle eine ausreichende Überdeckung der Software gewährleistet ist und diese im Betrieb fehlerfrei funktioniert.
Im vorliegenden Assignment wird die systematische Vorgehensweise zur Definierung von Testfällen, welche eine Anweisungs-, Zweig- und Pfadüberdeckung besitzen sollen, dokumentiert.
Dabei soll darauf geachtet werden, welche Testfälle für welche Überdeckung notwendig sind.
Um die Testfälle zu erstellen, sollte sich ein passender Kontrollfluss, der sich im Programmtext enthält, überlegt werden.
Anschließend ist zu zeigen, ob eine Funktions- und Ausgabenüberdeckung für einen Black-Box-Test mittels der bereits erstellten Testfälle erreicht wird.
Zum Abschluss soll dargestellt werden, welche Integrations- und Systemtests für das Projekt erforderlich sind.

[1] Vgl. Spillner/Linz (2005) S.1
[2] Vgl. Schneider (2012) S.1

1.3 Begriffsdefinitionen

„II. Software Engineering:

1. Begriff: Überprüfung eines Programms oder eines Softwaresystems auf Funktionsfähigkeit.

2. Zweck: Aufspüren und Beseitigen von Fehlern, nicht aber der Nachweis der Korrektheit. Letzteres ist durch Testen nicht möglich (Programmverifikation).
Vgl. auch Testdaten, Testhilfe.

3. Stufen: Im Softwarelebenszyklus steht das Testen für eine Phase, in der verschiedene Stufen durchlaufen werden:

a) Modultest: Überprüfung des Verhaltens eines einzelnen Moduls; der Modultest erfolgt in engstem Zusammenhang mit der Implementierung des Moduls, wird z.T. auch der Implementierungsphase zugerechnet.
b) Integrationstest: i.Allg. schrittweises Zusammenführen und Überprüfen mehrerer Module, bis alle Module eines Softwaresystems integriert sind.
c) Systemtest: Überprüfung eines Softwaresystems auf Vollständigkeit und Funktionstüchtigkeit auf der Grundlage des Pflichtenhefts bzw. der Anforderungsdefinition durch den/die Entwickler.
d) Abnahmetest (Akzeptanztest): Überprüfung des Softwareprodukts durch den Auftraggeber, i.Allg. auf Basis des Pflichtenheftes.

4. Formen:
a) Black-Box-Test: Beim Modultest angewendet; Modul wird als „Black Box" betrachtet. Testen erfolgt gegen die Spezifikation: Überprüfung, ob die Implementierung mit der Spezifikation übereinstimmt, d.h. ob das Modul das leistet, was als Aufgabe spezifiziert wurde (Liefert das Modul bei bestimmten Eingangswerten die erwarteten Ergebniswerte?).
b) White-Box-Test: Ausgangspunkt ist die interne Struktur eines Programms bzw. Programmsystems; überprüft wird die Programmlogik, v.a. die Steuerung des Programmablaufs."[3] Der Begriff Glass-Box-Test wird als Synonym verwendet.

[3] http://wirtschaftslexikon.gabler.de/Archiv/16891/testen-v11.html

2 Dynamische Testverfahren

2.1 Strukturtest (Glass-Box-Test)

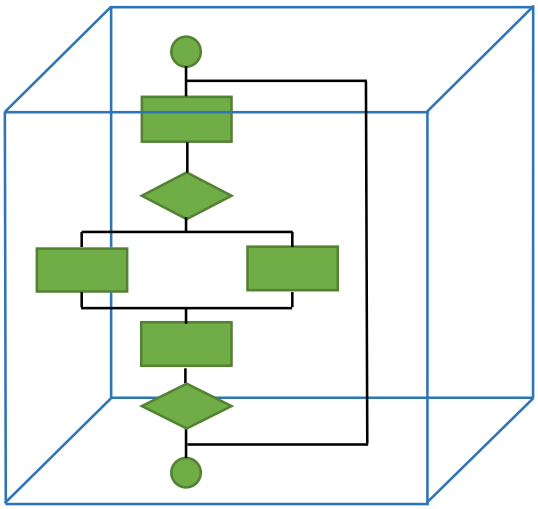


Abbildung 1: Glass-Box-Test (White-Box-Test)

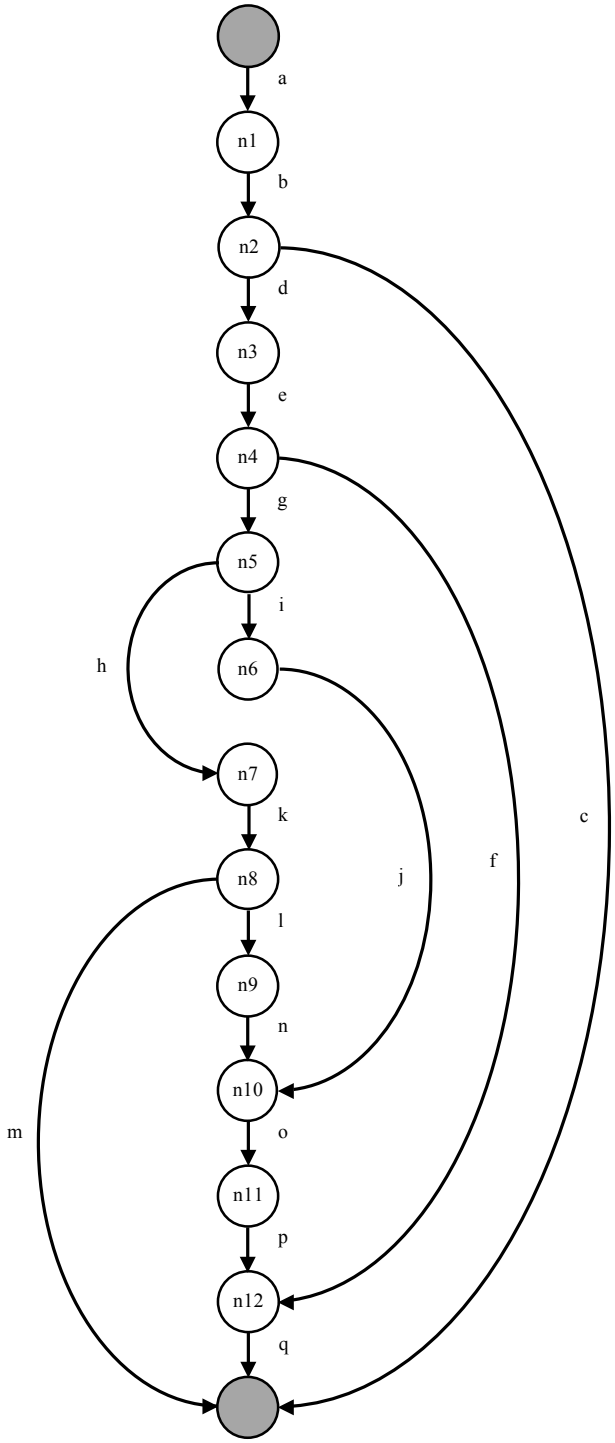
Das Glas-Box- oder White-Box-Verfahren wird auch oft als codebasiertes Testverfahren bezeichnet, da als Grundlage der Programmtext des Testobjektes verwendet wird. Im Gegensatz zum Black-Box-Verfahren muss der Programmtext vorliegen und unter Umständen auch manipuliert, d.h. ergänzt werden können. Beim Glass-Box-Verfahren ist die grundlegende Idee, dass alle Anweisungen, Bedingungen und Pfade eines Testobjektes mindestens einmal ausgeführt werden. Um ablauforientierte Testfälle zu ermitteln und durchzuführen, muss eine Programmlogik bestehen, die unter Berücksichtigung der Spezifikation zur Erstellung der Testfälle führt. Nach einer Durchführung kann entschieden werden, ob ein Fehlverhalten vorliegt. Folgende Glass-Box-Testfallentwurfsmethoden lassen sich unterscheiden:

- ➔ Anweisungsüberdeckung (C0)
- ➔ Zweigüberdeckung (C1)
- ➔ Bedingungsüberdeckung (C2, C3)
- ➔ Pfadüberdeckung (C4)

Im folgenden Abschnitt wird auf die Anweisungs-, Zweig- und Pfadüberdeckung näher eingegangen.[4]

[4] Vgl. Spillner/Linz (2005) S.144

2.1.1 Kontrollflussgraph



n_start

n1. Eingabe Konfiguration

n2. Einkaufen oder beenden

n3. Kundennummer plus Geheimcode

n4. Weiter oder Bestell-status abfragen

n5. Bestand verfügbar

n6. Lagerverwaltung

n7. Kundenanfrage wegen Produktionsdauer

n8. Antwort vom Kunden

n9. Produktion wird gestartet

n10. Bankeinzug bei Kunde

n11. Versand der Ware

n12. Bestellstatus

n_final

Abbildung 2: Kontrollflussgraph

2.1.1.1 Kontrollflussgraph Prinzip

Der Kontrollflussgraph, welcher ein gerichteter Graph ist, beginnt immer mit einem Startknoten und endet mit einem Endknoten, welche bei der Erstellung oft ausgelassen werden. Die im Programm befindlichen Befehle (Anweisungen) werden durch den Knoten gekennzeichnet. Mehrere aneinander folgende Anweisungen, die keine Verzweigung besitzen, können zusammengefasst werden. Pfade zwischen den Anweisungen werden durch Kanten (Kontrollflüsse) dargestellt.[5]

2.1.2 Anweisungsüberdeckung (statement coverage)

Um eine Anweisungsüberdeckung (C0-Test) durchzuführen, ist die Transferierung eines Programmtextes in einen Kontrollflussgraphen erforderlich. Der Graph selber besteht aus Knoten (Anweisungen) und Kanten (Kontrollfluss). Der Sinn der Anweisungsüberdeckung ist, dass jeder Knoten mindestens einmal durchlaufen wird. Wenn dies der Fall ist dann spricht man von einer vollständigen Anweisungsüberdeckung.[6]

2.1.2.1 Testfälle

Damit bei dem Anweisungsüberdeckungstest alle Knoten durchlaufen werden, sind bei unserem Programm zwei Pfade erforderlich, um eine vollständige Überdeckung zu erreichen:

- ➔ Pfad 1 = (a, b, d, e, g, h, k, l, n, o, p, q)
- ➔ Pfad 2 = (a, b, d, e, g, i, j, o, p, q)

Beim Testen sollte darauf geachtet werden, dass der Testaufwand möglichst gering gehalten wird, d.h. man sollte mit möglichst wenigen Testfällen das zu erfüllende Ziel erreichen.

2.1.2.2 Testendkriterium

Definierung der Kriterien zur Beendigung der Tests:

$$C0\text{-}Überdeckung = \frac{Anzahl\ durchlaufene\ Anweisungen}{Gesamtzahl\ Anweisungen} * 100\%$$

[5] Vgl. Fortbildungsunterlagen ISTQB Certified Tester (2014)
[6] Vgl. Spillner/Linz (2005) S.144f

„*Das obige Maß zur Berechnung der Überdeckung der Anweisungen ist auch als C0-Maß bekannt. Es ist ein in der Aussage sehr schwaches Kriterium. Manchmal ist eine 100%ige Überdeckung der Anweisungen schwer erreichbar, z.B. wenn Ausnahmebedingungen im Programm vorkommen, die während der → Testphase nur mit erheblichem Aufwand oder gar nicht herzustellen sind*."[7]

2.1.3 Zweigüberdeckung (decision coverage)

Um eine vollständige Zweigüberdeckung zu erreichen, müssen alle Zweige (Kanten) des Kontrollflussgraphen durchlaufen werden (if-Anweisungen, Schleifen). Beim Zweigüberdeckungstest (C1-Test) müssen strengere Kriterien erfüllt werden als beim Anweisungsüberdeckungstest. Jeder Zweig im Graphen wird durch mindestens einen Testfall abgedeckt. Mit der Zweigüberdeckung können auch fehlende Anweisungen in leeren Zweigen entdeckt werden.
Eine 100%ige Zweigüberdeckung enthält auch eine 100%ige Anweisungsüberdeckung.

2.1.3.1 Testfälle

Bei der Anweisungsüberdeckung wurden die Zweige c, i und m nicht ausgeführt. Um die vollständige Überdeckung zu erreichen, sind folgende Zweige zu durchlaufen:

- ➔ Pfad 1 = (a, b, c)
- ➔ Pfad 2 = (a, b, d, e, f, q)
- ➔ Pfad 3 = (a, b, d, e, g, h, k, m)
- ➔ Pfad 4 = (a, b, d, e, g, i, j, o, p, q)

Einige der Zweige sind mehrfach ausgeführt worden, dies bleibt aber nicht aus, da es keine Alternative für diese Kanten gibt.[8]

2.1.3.2 Testendkriterium

Definierung des Überdeckungsgrades der Zweige:

$$C1\text{-}\ddot{U}berdeckung = \frac{Anzahl\ durchlaufene\ Zweige}{Gesamtzahl\ Zweige} * 100\%$$

[7] Spillner/Linz (2005) S.146
[8] Vgl. Spillner/Linz (2005) S.147

„*Die Überdeckung der Zweige wird als C1-Maß bezeichnet. Bei der Berechnung wird nur gezählt, ob ein Zweig bei der Ausführung überhaupt durchlaufen wurde, die Häufigkeit der Ausführung spielt keine Rolle.*“[9]

2.1.4 Pfadüberdeckung (path coverage)

Bei der Pfadüberdeckung müssen alle möglichen Pfade (Zweigfolgen) getestet werden, beginnend beim Startknoten und endend beim Endknoten des Kontrollflussgraphen. Treten im Programm if-Verzweigungen auf, dann entstehen zwei Möglichkeiten, um weiterzulaufen. Enthält das Programm Schleifen, dann gibt es eine unbegrenzte Anzahl von Pfadmöglichkeiten, somit ist eine 100%ige Pfadüberdeckung nicht zu erreichen.[10]

2.1.4.1 Testfälle

Damit alle Pfade in unserem Programm abgedeckt sind, müssen folgende Testfälle durchgeführt werden:

- ➔ Pfad 1 = (a, b, c)
- ➔ Pfad 2 = (a, b, d, e, f, q)
- ➔ Pfad 3 = (a, b, d, e, g, h, k, l, n, o, p, q)
- ➔ Pfad 4 = (a, b, d, e, g, i, j, o, p, q)
- ➔ Pfad 5 = (a, b, d, e, g, h, k, m)

2.1.4.2 Testendkriterium

„*Es wird deutlich, dass es beliebig viele Pfade im Kontrollflussgraphen gibt. Auch bei einer Beschränkung der Anzahl der Schleifenwiederholungen steigt die Zahl von Pfaden ins Unermessliche an.*“[11]

Definierung des Überdeckungsgrades der Pfade:

$$C\infty\text{-}\ddot{U}berdeckung = \frac{Anzahl\ durchlaufener\ Pfade}{Gesamtzahl\ Pfade} * 100\%$$

[9] Spillner/Linz (2005) S.148
[10] Vgl. Schneider (2012) S.110
[11] Spillner/Linz (2005) S.154

3 Funktionstests (Black-Box-Tests)

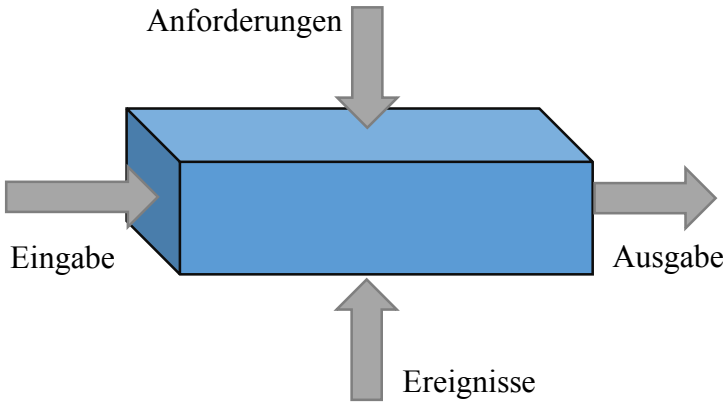


Abbildung 3: Black-Box-Test

„*Beim sog. Black-Box Test wird das System als schwarzer Kasten betrachtet, dessen innere Struktur und Funktionalität dem Tester verborgen bleibt. Es ist deswegen auch kein Zugriff auf interne Operationen und Zustände von außen möglich, so dass sich der Tester auf die Betrachtung des Ein-/Ausgabeverhaltens beschränken muss. Testfälle können somit nur anhand extern sichtbarer Schnittstellen und der Spezifikation des Testobjekts abgeleitet werden.*“[12]

3.1 Überdeckungsgrad bei Black-Box-Tests:

- Funktionsüberdeckung:
  Jede ausgeführte Funktion in der Spezifikation wird überprüft.
- Eingabeüberdeckung:
  Alle Eingabedaten sollten mindestens einmal auftreten.
- Ausgabeüberdeckung:
  Alle Ausgabedaten sollten mindestens einmal erzeugt werden.

Black-Box-Tests sind kein Ersatz für White-Box-Tests oder umgekehrt. Ein Entwickler könnte die Spezifikation falsch interpretieren und dadurch eine andere Funktion in das Programm einbauen. Mit der Hilfe des White-Box-Verfahren könnte dieser Fehler nicht entdeckt werden, da die innere Struktur des Programms fehlerfrei funktioniert. Erst beim Black-Box-Test würde

[12] http://www.software-kompetenz.de/?10142

auf Grund der Funktionsüberdeckung (Spezifikation) und Ausgabeüberdeckung der Fehler gefunden werden.
Mit den oben erstellten Testfällen könnte keine Funktions- und Ausgabeüberdeckung für einen Black-Box-Test erreicht werden.

4 Integrations- und Systemtests

4.1 Erforderliche Integrationstests

Eine Voraussetzung für den Integrationstest ist, dass die einzelnen Komponenten keine Fehler beinhalten und zuvor ausgiebig getestet worden sind. Bei der Integration werden die Gruppen dieser Komponenten von Integrationsteams zu Teilsystemen zusammengeführt. Der anschließende Integrationstest dient dazu, ob alle Einzelteile miteinander funktionieren.
Basisstrategien für Integrationstests:

- ➔ Top-down-Integration
- ➔ Bottom-up-Integration
- ➔ Ad-hoc-Integration
- ➔ Big bang Integration

In unserem Fall würde ich die Ad-hoc-Integration wählen, dabei werden die einzelnen Bausteine in einer zufälligen Reihenfolge ihrer Fertigstellung integriert. Der Vorteil liegt darin, dass jeder Baustein frühestmöglich in seine passende Umgebung integriert wird, wodurch man Zeit gewinnt.[13]
Ein Nachteil liegt darin, dass für einen Integrationstest mindestens zwei Bausteine vorhanden sein müssen, welche in Abhängigkeit zueinanderstehen. Aufgrund dessen, dass die Testumgebung nicht im Vorfeld vorbereitet werden kann, ist ein hohes Maß an Flexibilität und Improvisationsfähigkeit des Testteams erforderlich.[14]

4.2 Erforderliche Systemtests

Beim anschließenden Systemtest wird überprüft, ob die zuvor vom Kunden spezifizierten Anforderungen auch eingehalten worden sind.

[13] Vgl. Spillner/Linz (2005) S.56f
[14] Vgl. Winter et al. (2013) S.164

„*In den niedrigen Teststufen wurde gegen technische Spezifikation geprüft, aus der Perspektive des Softwareherstellers. Der Systemtest betrachtet das System hingegen aus der Perspektive des Kunden und des späteren Anwenders.*“[15]

Grundsätzlich unterscheidet man zwei Arten von Systemtests:

- ➔ Funktionaler Systemtest: Funktionale Anforderungen werden auf Vollständigkeit und Korrektheit überprüft.
- ➔ Nicht-funktionaler Systemtest: Nicht-funktionale Anforderungen wie z.B. Sicherheit, Benutzerfreundlichkeit, Zuverlässigkeit und Geschwindigkeit werden überprüft.[16]

In unserem Fall müssen funktionale und nicht-funktionale Systemtests durchgeführt werden. Das System sollte der Spezifikation und somit auch dem Kundenwunsch entsprechen. Dabei steht die Benutzerfreundlichkeit, Sicherheit, Stabilität und Geschwindigkeit besonders im Vordergrund.

5 Fazit und Ausblick

Die Ausarbeitung der Testfälle wurde nach der Anweisungsüberdeckung, Zweigüberdeckung und Pfadüberdeckung durchgeführt. Das allgemeine Ziel war, mit wenig Aufwand und einer geringen Anzahl von Testfällen eine vollständige Abdeckung des zu testenden Programmes zu erlangen. Auf sorgfältig ausgewählte Testfälle sollte geachtet werden. Mit dem Erstellen des Programmablaufplanes konnte man sich ein Bild über das gewünschte Programm des Kunden machen. Mittels diesem Ablaufplans wurde der Kontrollflussgraph generiert, welcher als Hilfe zur Erstellung der Testfälle gedient hat. Mit den erstellten Testfällen konnte für unser Programm eine hundertprozentige Abdeckung erreicht werden, da das Programm keinerlei Schleifen enthält. Mit der individuell angepassten Software wird das Unternehmen S-W-E seine spezifischen Prozesse besser handhaben können und dadurch auch mehr Geld einsparen. Mit den erstellten Testfällen wird die Sicherheit, Stabilität, Geschwindigkeit und Benutzerfreundlichkeit des Programmes gewährleistet. Eine hundertprozentig fehlerfreie Software wird es allerdings nie geben.[17]

[15] Spillner/Linz (2005) S.58
[16] Vgl. Badertscher/Scheuring (2007) S.74
[17] Vgl. Spillner/Linz (2005) S.162f

Anhang

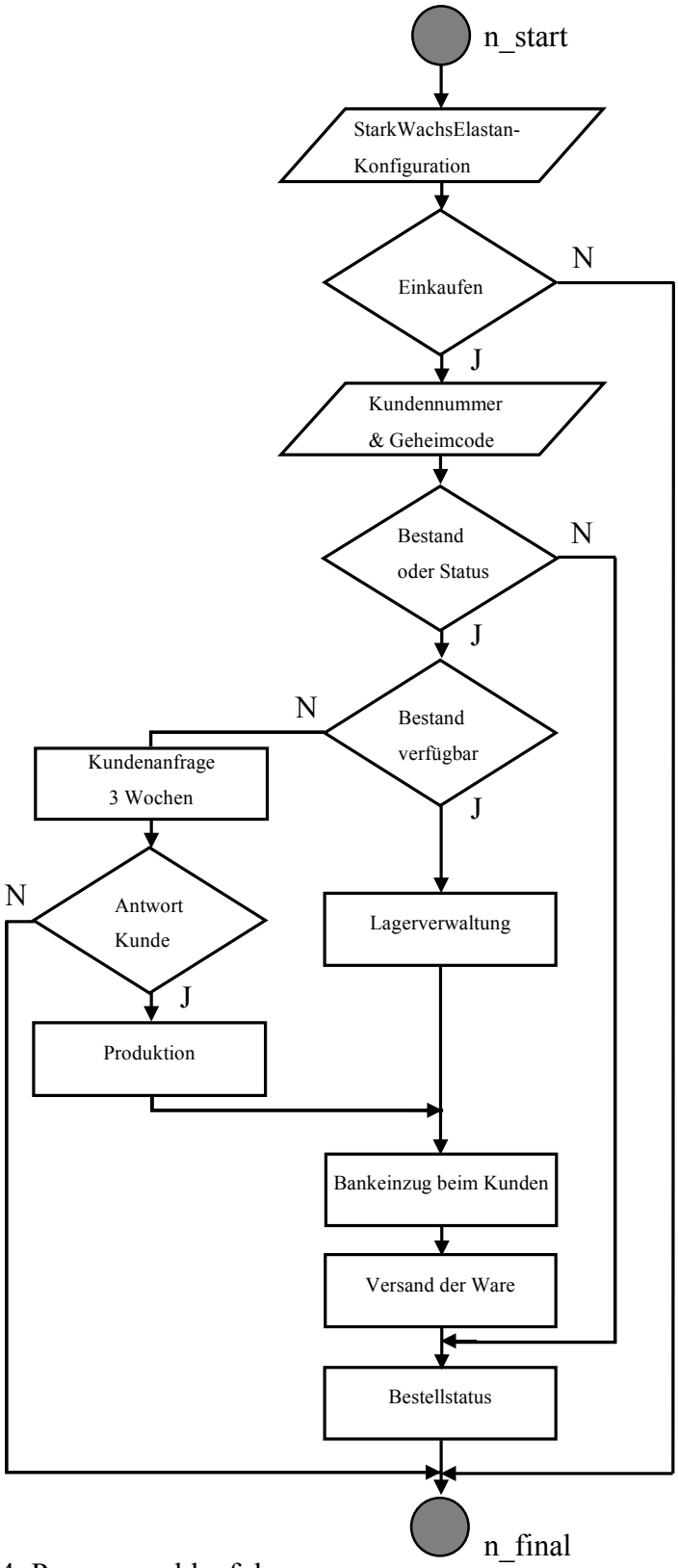


Abbildung 4: Programmablaufplan

Literaturverzeichnis

BADERTSCHER, K; SCHEURING, J.
Wirtschaftsinformatik: Entwicklung und Implementation eines Informations- und Kommunikationssystems: Methoden, Prozesse und Technologien - mit zahlreichen Illustrationen, Beispielen, Repetitionsfragen und Antworten, 1. Auflage, Zürich 2007.

IMBUS AG.
ISTQB Certified Tester Foundation Level, Software Qualitätssicherung und –Test, Schulungsunterlagen 2014 V2.6.

KALMAR, R.
Software-Engineering-Wissensdatenbank
URL: http://www.software-kompetenz.de/?10142 (Zugriff am 11.Januar 2016).

LACKES, R; SIEPERMANN, M; KIRCHGEORG, M.
Gabler Wirtschaftslexikon.
URL: http://wirtschaftslexikon.gabler.de/Archiv/16891/testen-v11.html (Zugriff am 14.Januar 2016).

SCHNEIDER, K.
Abenteuer Softwarequalität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement, 2. Auflage, Heidelberg 2012.

SPILLNER, A; LINZ, T.
Basiswissen Softwaretest, 3. Auflage, Heidelberg 2005.

WINTER, M; EKSSIR-MONFARED, M; SNEED, H; SEIDL, R; BORNER, L.
Der Integrationstest: Von Entwurf und Architektur zur Komponenten- und Systemintegration, 8. Auflage, München 2012.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]