WissGrid

Deliverable 2.1.7

Arbeitspaket 2: Blaupausen und Beratung

Strategien für den Aufbau und Betrieb von Testumgebungen[1]

Deliverable	2.1.7 Aufbau und Betrieb von Testumgebungen
Autoren	Arbeitspaket 2: Blaupausen und Beratung
Editoren	C. Grimme
Datum	22-02-2011
Dokument Version	1.0.1

A: Status des Dokuments

Deliverable 2.1.7, Version 1.0.1, Draft.

B: Bezug zum Projektplan

Dieses Dokument stellt Best Practices für den Aufbau und Betrieb einer Testumgebung zur Qualitätssicherung im Umfeld einer Community-Grid-Entwicklung zur Verfügung.

C: Abstract

Der Leitfadens sammelt und strukturiert die Erfahrungen der akademischen Projekte im D-Grid bezüglich Testvorgehen und den Aufbau von Testumgebungen. Darauf basierend wird ein Testprozess entwickelt, der zusammen mit einer vorgeschlagenen Testumgebung asl Grundlage für die

[1]This work is created by the WissGrid project. The project is funded by the German Federal Ministry of Education and Research (BMBF).

Etablierung eines Zertifizierungsprozesses dient.

D: Änderungen

Version	Date	Name	Brief summary
0.0.1	18.08.2010	C. Grimme	Erstellung des Arbeitsdokumentes
0.0.2	24.09.2010	C. Grimme	Fertigstellung der Auswertung der Teststrategien
0.0.3	11.10.2010	C. Grimme	Fazit aus der Auswertung
0.0.4	15.10.2010	C. Grimme	Grundlagen für Tests und Testprozesse
0.0.5	21.10.2010	C. Grimme	Testprozesse, Einleitung
0.0.6	22.10.2010	C. Grimme	Testprozesse und Integration (Grafiken)
0.0.7	25.10.2010	C. Grimme	Beschreibung Testprozess, Integration und Zertifizierung
0.0.8	26.10.2010	C. Grimme	Zertifizierung vorerst abgeschlossen
0.0.8	05.11.2010	B. Fritzsch	kleinere Korrekturen
0.0.9	03.01.2011	C. Grimme	Tabelle der eingesetzten Technologien in den Communities
1.0.0	17.02.2011	C. Grimme	Finalisierung nach RFC-Phase
1.0.1	22.02.2011	C. Grimme	Korrekturen in Summary und Einleitung

Inhaltsverzeichnis

Summary

This document provides a blueprint for establishing test processes in a grid community while respecting the usually decentralized structure of a grid project in the context of D-Grid. Additionally, the documents strives to propose a first approach for building up a testing environment for local development testing (component and local integartion tests) and globlal integration of project components (global integration, system and usability tests). In order to provide a solid basis for the processes and testing environments this document initially evaluates the current state of testing activity in existing academic projects of the D-Grid context. Subsequently, the process is defined and important elements are introduced. Finally, a basic testing environment is proposed together with a certification process for quality management during cooperative development.

Einleitung

Es ist inzwischen eine weitgehend anerkannte Einsicht im Bereich der Softwaretechnologie, dass zu einer qualitativ hochwertigen Software- und Systementwicklung gut strukturierte und nachhaltig ausgeführte Testprozesse gehören. Erstaunlicherweise haben sich solche Testprozesse in der Realität nur langsam etablieren können. Diese Situation ist auch im Umfeld der Systementwicklung des D-Grid-Projektes zu beobachten. Bei dem Aufbau verschiedener experimenteller Grid-Infrastrukturen für den akademischen Nutzen ist aufgrund der vorrangig notwendigen grundlegenden Erforschung solcher neuer Infrastrukturen ein geordneter Testprozess in den Hintergrund getreten. Ebenso wurden nur wenige ausgefeilte Testumgebungen und Zertifizierungsprozesse zur Qualitätssicherung etabliert.

Aufbauend auf den trotzdem vorhandenen grundlegenden Erkenntnissen zu Testprozessen im Grid, ausgehend von den allgemeinen Erfahrungen bezüglich der Softwareentwicklung in solchen Umgebungen und basierend auf modernen Erkenntnissen der Softwaretechnologie, soll in diesem Dokument ein generischer Vorschlag für den Aufbau eines Testprozesses vorgeschlagen werden. Zudem werden ein Konzept für eine im Grid-Kontext sinnvolle Testumgebung und ein den Testprozess und die Umgebung integrierender Zertifizierungsprozess entworfen.

Das Dokument ist folgendermaßen aufgebaut: Der erste Abschnitt des Leitfadens sammelt und strukturiert die Erfahrungen der akademischen Projekte im D-Grid. Danach wird zusammen mit den notwendigen Grundlagen der Testtechnik ein Testprozess eingeführt. Schließlich wird eine Testumgebung entwickelt, die einen parallelen Betrieb von produktiven und sich in der Entwicklung befindlichen Komponenten ermöglicht. Der Testprozess und die Nutzung der vorgeschlagenen Testumgebung ermöglicht schließlich im letzten Schritt die Etablierung eines Zertifizierungsprozesses für entwickelte Infrastruktur- und Mehrwertkomponenten im Grid.

Kapitel 1

Auswertung von Teststrategien in den Community-Grids

Um generische Strategie zum Aufbau und Betrieb von Softwareumgebungen im Kontext von Community-Grids zu entwickeln, werden im folgenden Kapitel die Ansätze der bisherigen akademischen Community-Grids analysiert. Dies geschieht auf Basis der Ergebnisse, die bei der Materialerhebung in Deliverable 2.1.1 zusammengetragen wurden [1]. Dabei wird hier bewusst auf die Einzeldarstellung der Ansätzen verschiedener Community-Grids verzichtet, jedoch detailliert auf spezielle Aspekte eingegangen. In einem zweiten Schritt werden strukturelle Vorgehensunterschiede und -gemeinsamkeiten herausgearbeitet und zu einem Ausgangspunkt für die Erstellung einer generischen Struktur weiterentwickelt.

Der hauptsächliche Gegenstand der Materialsammlung war die Erfassung von Testumgebungen, Testverfahren und Methoden des Qualitätsmanagements in den einzelnen Community-Grids. Aus diesem Grund wird die folgende Zusammenfassung entlang dieser Punkte strukturiert.

1.1 Test- und Qualitätsziele des Projektes

Die Festsetzung der Test- und Qualitätsziele der meisten Community-Projekte orientiert sich stark an den UseCases der wissenschaftlichen Communities und den Anforderungen, die im Projektantrag festgelegt wurden. Aus den Anforderungen werden gewöhlicherweise im ersten Teil des Projektes konkrete Funktionalitäten abgeleitet, die anschließend nicht nur als Zielsetzungen für die Entwicklung selbst, sondern auch als Qualitätsmaße für die Tests der Software und Infrastruktur herangezogen werden.

Damit ergeben sich implizit verschiedenartige Testziele, die insgesamt in funktionale und nichtfunktionale Kriterien differenziert werden können. Diese unterscheiden sich darin, dass erstere besonders die Überprüfung und Bewertung von Funktionalität der Infrastruktur und der darin enthaltenen Dienste zulassen. Die nicht-funktionalen Anforderungen resultieren vorwiegend in Tests auf Performanz, Ausfallsicherheit und Nutzerakzeptanz.

Es muss schließlich darauf hingewiesen werden, dass die Auswertung der Vorgehensweisen in den Community-Grids kein formales Vorgehen zur Spezifikation und Dokumentation von Testfällen ermitteln konnte. In den Projekten wurden die Meilensteine und Deliverables des Antrages als Mo-

mente der Prüfung von Anforderungserfüllung und Softwarefunktionalität gesehen.

1.2 Testverfahren für Softwareentwicklung eines Partners

Durch die konzeptionelle und organisatorische Struktur der durchgeführten Verbundprojekte waren technische Partner als weitgehend eigenständige Entwicklungsteams an einer abgegrenzten Aufgabe tätig. In den meisten Projekten umfasste dies die Entwicklung einer Softwarekomponente oder eines Middlewaredienstes, der später in die Gesamtinfrastruktur integriert wurde.

Dies führte gewöhnlich zu weitgehend voneinander unabhängigen Entwicklungsprozessen und damit zu unterschiedlichen Ansätzen Tests durchzuführen. Teile der technischen Partner definierten einen rudimentären Test- und Dokumentationsprozess, der vor der Bereitstellung eines Releasekandidaten oder eines Patches durchlaufen wurde. Andere Partner beschränkten sich auf die Durchführung von Unit-Tests, um die Funktionalität von Bestandteilen ihrer Software zu überprüfen und sicherzustellen.

Zusätzlich wurden oft Testumgebungen aufgebaut, die eine lokale Evaluation der entwickelten Komponenten und Dienste ermöglichte, ohne sie direkt in das Gesamtsystem zu integrieren. Diese können als eine Vorstufe von Integrationstests gesehen werden, um Schnittstellen und interne Funktionalität der entwickelten Komponente zu testen.

1.3 Testverfahren und Umgebungen für den integrierten Betrieb

Für das Gesamtsystem eines Community-Grids wurde in den Projekten kein erkennbarer definierter Testprozess verfolgt. Allerdings sind verschiedene Aspekte von Tests durchaus zu finden.

Integrationstests: Für den Aufbau des Gesamtsystems „Grid“ mussten verschiedene verteilt laufende Dienste und Komponenten gekoppelt werden. Dazu war periodisch eine gemeinsame Integrations- und Abstimmungsphase zwischen den beteiligten technischen Partnern notwendig. Um die fehlerfreie Zusammenarbeit der unterschiedlichen Komponenten zu garantieren, wurden funktionale Tests entwickelt. Diese umfassten zumeist die Überprüfung der für ein Release zuvor festgelegten Funktionalitäten anhand standardmäßiger Nutzungsschemata der Infrastruktur. Bei dem Projekt MediGrid wurden die neu entwickelten Workflows ebenso wie bei C3Grid getestet. Insbesondere bei der C3Grid-Infrastruktur umfasst die Bearbeitung eines Workflows verschiedenste unabhängig voneinander entwickelte Dienste. Für den Test der Integration wurden vorher dedizierte Workflows ausgewählt und anschließend als festgelegte Testfälle durchgeführt.

Nutzerakzeptanztests: Neben den oben beschriebenen Integrationstests führten die meisten Projekte in regelmäßigen Abständen sogenannte Akzeptanztests mit potentiellen Nutzern der entwickelten Infrastruktur durch. Dabei wurde Nutzern das System präsentiert (im Rahmen individueller Präsentation oder auf Nutzerworkshops) und entsprechendes Feedback zur Nutzbarkeit und weiteren nicht-funktionalen Anforderungen aufgenommen. Zudem bestand hier die Möglichkeit einer Überprüfung der Projektanforderungen und damit der Grundlage für die funktionalen Tests.

Langzeittests: Zusätzlich zur kurzzeitigen Überprüfung der Infrastrukturfunktion und -bedienbarkeit wurden von fast allen Projekten Langzeittests der entwickelten und noch nicht produktiv gestellten Infrastruktur durchgeführt. Dazu wurden sogenannte Testnutzer eingeladen, das System in einem lang angelegten, täglichen Arbeitsprozess zu evaluieren.

Die oben beschriebenen Testmethoden implizieren weitgehend die Existenz einer Testumgebung für die Infrastruktur, in der neben dem produktiven Betrieb der Infrastruktur Integrations- und Nutzungstests durchgeführt werden können. Damit wird zugleich ein rudimentärer Test- und Integrationsprozess definiert: Die Entwicklung einer neuen Version der Infrastruktur erfolgt stets in einer nicht produktiven Umgebung. Hier können alle Fehler evaluiert und behoben werden. Erst bei einem fehlerfreien Betrieb der neuen Infrastruktur erfolgt eine Produktivstellung. Dabei ist allerdings offen, wie die Entscheidungsgrundlage für Produktivstellung gestaltet wird. Dieser Punkt wurde in jedem Projekt individuell festgelegt und folge der bisherigen Analyse zufolge keiner festgelegten Methodik. Der folgende Abschnitt adressiert diesen Aspekt kurz.

1.4 Test- und Zertifizierungsprozesse in den Projekten

Ein festgelegter Prozess zur Testdurchführung oder sogar zur Zertifizierung von Entwicklungen war in keinem Projekt vorhanden. Ausschließlich Teile des MediGrid definieren einen Testprozess für die Integration neuer Softwarekomponenten für das Userinterface [5]. Dabei werden Neuentwicklungen zuerst in einer Testumgebung betrieben und auf Fehlfunktionen überprüft. Erst wenn diese Tests erfolgreich abgeschlossen sind, wird das neue System in die produktive Umgebung übernommen.

TextGrid bildet für die gleiche Aufgabe eine Projekt-übergreifende Arbeitsgruppe, die Tests bewertet und über eine Aufnahme von Funktionalität in den Produktivbetrieb entscheidet. Schließlich findet sich auch im C3Grid-Projekt ein ähnlicher, impliziter Prozess, der aber nicht formalisiert ist.

1.5 Tabellarische Zusammenfassung der genutzen Prozesse und Werkzeuge

In Tabelle 1.1 sind überblicksartig die wichtigsten Daten zu den Prozessen der Testdurchführung innerhalb der Community-Grids zusammengestellt. Dabei wurde insbesondere versucht neben der Darstellung der durchgeführten Prozesse ebenfalls eine Darstellung der gewählten Werkzeuge zu erreichen.

1.6 Fazit

Insgesamt ist anhand der vorherigen, kurzen Auswertung festzustellen, dass die Community-Grid-Projekte zwar die Notwendigkeit von Softwaretests erkannt haben, jedoch bisher kein umfassendes Konzept entwickeln konnten, das einen umfassenden Testprozess zur Verfügung stellt und damit die Voraussetzung für eine Zertifizierung der Softwareentwicklung ermöglicht.

Diese Beobachtung lässt sich sicherlich zu einem großen Teil auf die im Kontext des Aufbaus einer Grid-Infrastruktur sehr spezielle Art der Softwareentwicklung zurückführen. Durch die Struk-

turierung eines Projektes als Verbundvorhaben weitgehend eigenständiger Partner ist ein üblicher oft sehr stringenter und geregelter Entwicklungsprozess nicht leicht umzusetzen, siehe auch die Ausführungen in Deliverable 2.1.6 [2]. Wie dargestellt, hat fast jeder Partner einen individuellen und lokalen Testprozess entwickelt, um die Qualität der eigenen Softwarekomponente zu gewährleisten. Allerdings gab es auch hier keinen gemeinsamen Ansatz oder eine gemeinsame Grundlage für die Definition von Qualitätszielen innerhalb des Projektes. Diese Festlegung wurde einerseits in die Integrationsphase und die damit verbundenen Test, andererseits in die Nutzungstests verlagert.

Aus den Erfahrungen der Projekte kann geschlossen werden, dass dieser Ansatz zu erheblichen Verzögerungen im Projektfortschritt führen kann. Schlimmstenfalls kann das Gesamtsystem nur das niedrigste festgelegte Qualitätsniveau einer lokalen Entwicklung erfüllen oder alternativ sich der Projektfortschritt stark verzögern, weil einzelne Komponenten auf das angestrebte Niveau gehoben werden müssen.

Hieraus ergeben sich für den folgenden Entwurf zwei Vorgaben: die Strategie zum Aufbau und Betrieb von Testumgebungen muss neben einem grundlegenden Testprozess ebenfalls eine Methode zur Festlegung projektweiter Qualitätsstandards und damit von Zertifizierungsanforderungen vorschlagen. Dabei sollen aber insbesondere die bisherigen Vorgehensansätze der Community-Grids berücksichtigt und integriert werden. Speziell der Aufbau von parallelen Umgebungen zum Testen und für die Sicherstellung des produktiven Betriebes soll aufgegriffen werden.

Prozesse	Spezifische Methoden und Metriken	AstroGrid-D	C3Grid	HEP	MediGRID	TextGrid
Komponententest	Unittests Whitebox-Tests		jUnit manuelles Debugging		jUnit	
	Metriken		keine spezifischen Metriken, Anforderungen aus der Spezifikationsphase als Leitlinien			
Integrationstest	Werkzeuge		weitgehend manuelle Integration auf Basis von festen Workflows		automatische Integration, unterstützt von Apache Maven	
	Metriken		keine spezifischen Metriken, Anforderungen auf Klima-Workflow-definitionen			
Nutzerakzeptanztest	Methodik		Workshops zur Demonstration und zur Sammlung von Feedback			
	Metriken		Nutzerfreundlichkeit, Stabilität, Performanz			
Langzeittest	Umgebungen, Werkzeuge		Produktions- und Testumgebung		Produktions- und Testumgebung	
	Metriken		Stabilität, Performanz			

Tabelle 1.1: Tabellarische Sammlung der Vorgehensweisen in den unterschiedlichen Community-Grids bezüglich der Durchführung von Tests.

Kapitel 2

Grundlagen für Tests und Testprozesse

Dieses Kapitel erläutert kurz grundlegende Begriffe des Sortwaretest, die im Folgenden häufig verwendet werden. Dabei wird an dieser Stele keine umfassende Darstellung gegeben. Diese ist vollständig in den vielen Lehrbüchern zum Thema nachzulesen. Dazu sei etwa auf die Arbeiten von Salomon [4] oder Spillner und Linz [6] verwiesen. Hier soll vielmehr auf einzelne Aspekte des Testens eingegangen und noch auf deren Bedeutung im Grid-Umfeld hingewiesen werden. Im zweiten Teil dieses Kapitels wird eine kurze Zusammenfassung der wichtigen Elemente für die Entwicklung eines Prozesses gegeben. Dabei werden diese kurz in den Kontext der Prozessdefinition eingeordnet und erläutert (siehe auch Deliverable 2.1.6 [2]). Auch hier wird keine abschließende Darstellung über Prozesstheorie angestrebt, sondern lediglich die im folgenden genutzten Elemente eingeführt.

2.1 Grundbegriffe des Testens

In diesem Abschnitt werden drei grundlegende Themen angesprochen: die Frage, wofür überhaupt getestet wird und wie Qualitätsziele für Tests innerhalb eines Projektes festgelegt werden können sowie die verschiedenen Arten von Tests.

2.1.1 Qualitätsziele beim Testen

Die gute Qualität eines Softwareproduktes ist oft grundlegende Voraussetzung für den erfolgreichen Einsatz und zugleich ausschlaggebend für die langfristige Bindung der Nutzer an das Produkt. Auch im Kontext von Grid-Softwareentwicklungen ist dies ein ausschlaggebender Faktor: eine Gridinfrastruktur, die auf qualitativ sehr unterschiedlichen Softwarekomponenten basiert oder deren Diensteinteraktion nur rudimentär getestet wurde, ist in der Praxis nicht oder sehr eingeschränkt produktiv einsetzbar und wird kaum Nutzer gewinnen können. Daher ist eine Festlegung von Qualitätszielen für Projekte im Allgemeinen unumgänglich. Neben den gewöhnlich über Nutzeranforderungen definierten funktionalen und nicht-funktionalen Zielen bestimmt sich die Qualität einer Software über Messungen, die angeben, inwieweit die Entwicklung

1. dem *zuvor spezifizierten Pfad* folgt,
2. wie groß die *Distanz zu den gesetzten Zielen* ist und

3. wann das *gesetzte Ziel erreicht* ist.

Für solche Messungen können Qualitätsziele definiert werden. Diese können natürlich niemals ohne den entsprechenden Kontext der Anforderungen an ein Gesamtsystem betrachtet werden. Insgesamt kann man aber folgende einige Punkte als generische Qualitätsziele beschreiben, die später konkretisiert werden müssen:

Funktionalität: Die Abdeckung der vom System geforderten Fähigkeiten gilt als einer der entscheidenden Faktoren für den sinnvollen und zweckmäßigen Einsatz des Softwareproduktes. Es muss immer das Ziel der Entwicklung sein, die zuvor spezifizierten Funktionen, anwenderspezifischen Normen und Vereinbarungen umzusetzen. Somit bezieht sich das Ziel insbesondere auf die Nähe zum zuvor festgelegten Entwicklungspfad.

Testabdeckung: Das Kriterium der Testabdeckung gibt Aufschluss über die Intensität der durchgeführten Tests und muss prinzipiell mehrschichtig betrachtet werden. Ein komplexes System ist im Allgemeinen nicht durch eine einzige Variante von Tests zu prüfen, sondern muss vielfältigen Arten von Tests ausgesetzt werden (siehe Abschnitt 2.1.2). Eine gute Auswahl von verschiedenen Testarten stellt dann sicher, dass ein System bezüglich verschiedener Aspekte beurteilt wird und damit der Großteil der möglichen Systemzustände abgedeckt wird. Diese Abdeckung wird gewöhnlicherweise vor Entwicklungsbeginn als Qualitätsniveau definiert in Prozent angegeben.

Zugleich trifft das Kriterium der Testabdeckung auf jede einzelne Testmetrik zu, die im Gesamtkontext eingesetzt wird. Das bedeutet, dass auch jeder Test eine gute Testabdeckung vorweisen muss. Nur wenn etwa eine prozentual große Anzahl von möglichen Testfällen, Codepfaden oder Fehlerzuständen von den Tests abgedeckt wurde, sind sie bezüglich ihrer Qualität aussagefähig.

Restfehlerrate und Zuverlässigkeit: Die Zuverlässigkeit eines Systems lässt sich allgemein als Fähigkeit des Systems definieren, sein Leistungsniveau unter zuvor spezifizierten Bedingungen über einen vorgegebenen Zeitraum oder eine festgelegte Anzahl von Aktionen aufrecht zu erhalten. Dabei spielen zwei weitere Komponenten eine für die Bewertung wichtige Rolle. Die *Reife* eines Systems drückt sich durch die Häufigkeit des Versagens und die *Fehlertoleranz* als Beständigkeit des Systems gegen Defekte und Fehlbedienung aus.

Es ist jedoch festzuhalten, dass im Allgemeinen auch bei erfolgreichen Tests und einer hohen Zuverlässigkeit ein Restrisiko für den Systembetrieb nicht auszuschließen ist. Durch eine entsprechende Testabdeckung ist die Fehlerrate jedoch als entsprechend klein sicherzustellen.

Testdokumentation: Die Testdokumentation ist nicht nur wichtige Voraussetzung für die Durchführung von Tests sondern gleichzeitig ein wichtiges Qualitätsmaß für die Tests selbst. Ein schlecht spezifizierter oder schlecht dokumentierter Testprozess (oder auch Einzeltest) hat zwei entscheidende negative Einflüsse: Ist die vorherige Spezifikation der Tests unzureichend, kann nicht sichergestellt werden, ob der Test die Anforderungen, die an das System gestellt wurden, abdeckt. Damit kann aber auch nicht sichergestellt werden, dass der Test irgendeine Aussage bezüglich des Entwicklungspfades, des Abstands zum Ziel oder über das Erreichen des Ziels liefern kann.

In ähnlicher Weise verhält es sich in dem Falle, dass Testergebnisse selbst nicht angemessen dokumentiert werden. Dann ist eine spätere Beurteilung der Ergebnisse nicht möglich und

eine Einordnung der Ergebnisse in das Gesamtergebnis des Testprozess schwierig. Weiterhin ist aber aufgrund fehlender Dokumentation eine Überprüfung (Review) des Testprozesses selbst im Nachhinein unmöglich. Dies ist jedoch ein wichtiger Aspekt, um die Sinnhaftigkeit des Testvorgehens kontinuierlich zu prüfen.

Weitere Kriterien für Softwarequalität sind nach ISO-Norm 9126 die *Benutzbarkeit*, *Effizienz* und *Änderbarkeit* des Systems. Diese nicht-funktionalen Anforderungen sind ebenso wichtig wie die vorherigen detailliert besprochenen Aspekte, müssen oft jedoch speziell für die jeweiligen Systeme gefasst werden. Wie bereits einleitend angesprochen ist Benutzbarkeit des Systems elementar für die Akzeptanz und Nachhaltigkeit der Entwicklung. Gleiches gilt für die Effizienz. Beide Anforderungen sind evaluierbar, setzen aber eine sehr gute Spezifikation der Leistungskriterien voraus. Ein entsprechender Test dieser Aspekte ist speziell anzupassen. Die Anforderung der Änderbarkeit bezieht sich auf technische und Architektonische Details der Entwicklung. Ein Test dieses Kriteriums ist grundsätzlich schwer durchzuführen. Abgesehen von einigen sehr rudimentären Metriken zur Codequalität und Beurteilung des Kontrollflusses sind wenige generelle Methoden vorhanden.

2.1.2 Arten von Tests

Um die zuvor beleuchteten Qualitätsziele zu erreichen, werden verschiedene Arten von Softwaretests gewöhnlicherweise entlang des strikten Wasserfallprozesses der Softwareentwicklung motiviert. Trotzdem sind die so festgelegten Testarten unabhängig vom Wasserfallmodell in jedem anderen Prozess nutzbar. Aus diesem Grund werden hier ausschließlich die unterschiedlichen Testarten erwähnt und kurz erläutert. Wie diese später in einem Prozess für die Entwicklung einer Gridinfrastruktur genutzt werden, ist Gegenstand des Kapitels 3.

Komponententest

Der Komponententest kann als unterste Stufe der Tests gesehen werden, der nur einzelne Bausteine des Gesamtsystems betrachtet, deren Verhalten quasi von externen Einflüssen isoliert untersucht. Diese Komponenten können abhängig von der Entwicklung und der verwendeten Programmiersprache Programmmodule, Units oder Klassen sein. Der Test soll einen systematischen Ansatz für das Auffinden von Fehlverhalten darstellen und hat durch die ausschließliche Betrachtung einer atomaren Komponente den Vorteil, dass die Zuordnung von Fehlern zu der speziellen Komponente leichter ist, als in einem integrierten System.

Die Zielsetzung eines Komponententests ist zweiteilig: Zum einen werden funktionale Tests durchgeführt, die die Spezifikation der Komponente überprüfen. Dies geschieht weitgehend über Ein-/Ausgabetests, die die Korrektheit der Ausgabe abhängig von einer Eingabe in die Komponente überprüfen. Zweitens wird die Robustheit der Schnittstellen gegen Fehleingaben und falsche Benutzung getestet.

Für die Entwicklung einer Teststrategie werden ebenfalls zwei wesentliche Fälle unterschieden: Entweder hat der Tester Zugang zum Quellcode der Komponente oder nicht. Im ersten Falle spricht man von einem *Whitebox-Test*, bei dessen Testvorgehen die Kenntnis des Quellcodes vorausgesetzt wird, etwa über die Programmstruktur, Methoden, Variablen oder Algorithmen. Diese Information kann genutzt werden, um Codepfade der Komponente zu prüfen. Diese Prüfung sollte so geplant werden, dass möglichst jeder Codepfad der Komponente abgedeckt wird und damit auch auf sein

Verhalten überprüft wird. Diese Aufgabe wird oft technisch unterstützt und automatisiert (z.B. mittels Jtest für Java-Klassen). Die Alternative zum Vorgehen mit vollständigem Quellcode-Einblick stellen die sogenannten *Blackbox-Test* dar. Diese setzen keine Kenntnis der internen Struktur der Komponenten voraus und nehmen nur die (Funktions-)Spezifikation der Schnittstellen und die Schnittstellen selbst an. Darauf basierend werden Testfälle entworfen, die das Verhalten Eingabe-/Ausgabeverhalten der Schnittstellen überprüfen. Auch für diese Fälle ist eine entsprechende Testabdeckung sicherzustellen. Allerdings können oft nur repräsentative Fälle getestet werden, von denen dann auf das gesamte Komponentenverhalten geschlossen werden muss. Auch hier sind technische Hilfsmittel verfügbar, die die Ausführung der Testfälle als Testtreiber übernehmen. Frameworks wie JUnit, nUnit oder CppUnit bilden oft eine gute Grundlage für projektweit standardisiertes Testen.

Integrationstest

Der dem Komponententest nachgelagerte Schritt ist der Integrationstest. Dabei werden die Komponenten nicht mehr als atomare Funktionseinheiten betrachtet sondern zu Teilsystemen und schließlich dem Gesamtsystem zusammengefügt und bezüglich ihrer Interaktion getestet. Der Ausgangspunkt und die Grundlage für die Testdurchführung sind Dokumente, die das Systemdesign und die Architektur definieren, schnittstellenübergreifende Workflows, die die Interaktion mehrerer Komponenten betreffen sowie Anwendungsfälle.

Ähnlich dem Komponententest müssen auch beim Integrationstest sogenannte Testtreiber entwickelt werden, die die einzelnen Testfälle repräsentieren. Allerdings betrachten diese Treiber nicht mehr einzelne Komponenten sondern das Gesamtsystem oder zumindest ein Teilsystem als Testobjekt. Wichtig ist in diesem Kontext eine Überwachung der Schnittstellen, nicht nur das Ein- und Ausgabeverhaltens des Gesamtsystems, da die Interaktion der Komponenten Situationen aufdecken kann, die während der Komponententests nicht ermittelt werden konnten.

Zielsetzung des Testvorgehens ist es, Schwachstellen in Schnittstellen festzustellen, die insbesondere bei der Integration mit anderen Komponenten auftreten können. Dies sind durchaus unterschiedliche Interpretationen der spezifizierten Schnittstelle oder Probleme mit der Interpretation ausgetauschter Daten. Solche Fehler sind während eines Komponententest nicht immer auffindbar.

Systemtest

Die Systemtests betrachten nicht mehr die Interaktion der einzelnen Komponenten im Detail und auf Ebene der Schnittstellen. Vielmehr wird das Gesamtsystem hier auf die Erfüllung der zuvor spezifizierten Anforderungen überprüft. Die Betrachtung geschieht daher nicht vorwiegend aus technischer Sicht sondern aus der Perspektive des Nutzers. Dazu wird neben dem vollständig integrierten System als Testobjekt auf die Anforderungsspezifikationen und auch auf Beschreibungen wie das Handbuch als Grundlage für die Systemtests zurückgegriffen.

Insbesondere ist die Umgebung für den Systemtest gut zu wählen. Diese sollte idealerweise der späteren Produktivumgebung entsprechen, um das Systemverhalten realistisch prüfen zu können. Neben der Vollständigkeit und Robustheit kann in einer solchen Umgebung auch die Effizienz, kurzum eine Reihe nicht-funktionaler Anforderungen an das System getestet werden.

Abnahmetest

Den letzten Testabschnitt in der Reihe der Softwaretests bildet die Gruppe der Abnahmetests. Diese gliedern sich in drei wesentliche Bestandteile auf, nämlich die Benutzerakzeptanztests, Tests durch Systembetreiber und langfristiger angelegte Feldtests. Die Benutzerakzeptanztests sollen die Eignung des entwickelten Systems für eine Verwendung durch den Nutzer prüfen. Diese Tests müssen regelmäßig durchgeführt werden, um frühzeitig die Anforderungen an die Nutzbarkeit des Systems anpassen zu können. Tests durch den Systembetreiber stellen die Unbedenklichkeit der entwickelten Software sicher, während Feldtests schließlich die langfristige Evaluation der Software durch ausgewählte Nutzer und Betreiber darstellen.

2.2 Prozesse

Prozesse sind im Allgemeinen Abläufe, in denen Handelnde zeitlich gesteuert Aktivitäten verrichten und dabei Artefakte verwenden oder erzeugen. Dieser formale Ablauf wird als Workflow bezeichnet, dessen Komponenten im Folgenden kurz generell definiert werden. Die Notation soll im folgenden Kapitel 3 verwendet werden, um verschiedenste Abläufe in einem Test- und Zertifizierungsprozess für die Entwicklung von Gridinfrastrukturen darzustellen. Zudem wird an dieser Stelle die Notation aus Deliverable 2.1.6 [2] der Vollständigkeit halber wiederholt.

2.2.1 Rollen

Rollen bezeichnen Verantwortlichkeiten der im Prozess Handelnden. Dabei ist eine Rolle nicht auf eine einzige Person beschränkt. Auch Gruppen können die Funktion einer Rolle übernehmen und die damit verbundenen Verantwortlichkeiten und Aufgaben teilen. Ebenso ist es möglich, dass mehrere Rollen von einer einzigen Person oder der gleichen Gruppe eingenommen werden. Die genaue Zuordnung der Rollen im Projektteam ist nicht Bestandteil des Prozesses. Obwohl die Verantwortlichkeiten und Tätigkeiten einer Rolle im Prozess definiert werden, ist die personelle Zuordnung zu Beginn des Projektes oder einer Projektphase von der Projektleitung zu treffen.

2.2.2 Aktivitäten

Die Aktivitäten in einem Prozess bezeichnen die von einer Rolle durchzuführenden Tätigkeiten. Deshalb wird einer Aktivität generell eine Rolle zugeordnet, die diese ausführt und dabei eine kleine Anzahl von Artefakten nutzt, verändert oder erzeugt. Jede Aktivität kann als abstrakte Handlung verstanden werden, die selbst wieder einen komplexen Aufbau hat, wobei es notwendig ist, sie auf einem bestimmten Betrachtungslevel des Prozesses als atomare Handlung zu verstehen, um den Prozess nicht zerfasern zu lassen. Die Literatur rät eine Aktivität in ihrer zeitlichen Komplexität nicht über wenige Tage hinaus und nicht unter wenigen Stunden zu definieren. Für eine einheitliche Darstellung nehmen wir hier jedoch verschiedene Komplexitätsebenen für eine Aktivität an. So können in Prozessbeschreibung Aktivitäten auftauchen, die sich über einen langen Zeitraum erstrecken und selbst im Sinne von Workflows in Unteraktivitäten unterteilt werden können.

2.2.3 Artefakte

Diese Elemente bezeichnen in einen Prozess einfließende und aus dem Prozess entstehende Informationen und Produkte. Sie stehen den Rollen zur Ausführung einer Aktivität zur Verfügung und werden ggf. von einer Aktivität produziert. Dabei kann es sich um einfache Informationen, komplexe Produkte, Dokumente oder auch das fertige Endprodukt des Prozesses handeln. Generell werden Artefakte genutzt, modifiziert oder erzeugt. Dabei kann implizit eine Hierarchie oder Teil-von-Beziehung angenommen werden: Rollen erzeugen auf Basis existierender Artefakte neue Artefakte während ihrer Aktivitäten. Bei dem Endprodukt handelt es sich folglich um das umfassenste Artefakt, die Gesamtmenge aller Artefakte.

2.2.4 Grafische Notation

Zur Darstellung eines Prozesses wird gewöhnlicherweise auf eine grafische Darstellung des Workflows zurückgegriffen, da dieser sowohl inhaltliche Aspekte (Aktivitäten, Rollen und Artefakte), wie auch zeitliche Aspekte zusammenfasst. Abbildung 2.1 fasst die in diesem Dokument zur Verfügung stehenden grafischen Elemente zusammen.

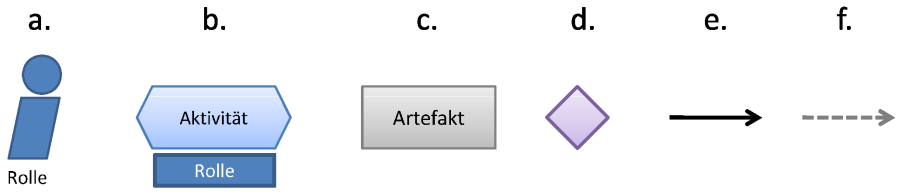

Abbildung 2.1: Legende der verwendeten grafischen Notation: (a) Rollen im Prozess, (b) Aktivitäten mit beteiligter Rolle, (c) Artefakte, Produkte, (d) Entscheidungspunkte, (e) zeitliche Abhängigkeiten, (f) Artefaktgenerierung.

Rollen werden grafisch als stilisierte Personen dargestellt (Abb. 2.1a), können jedoch der Übersichtlichkeit halber auch an ausgeführten Aktivitäten anotiert werden. Aktivitäten werden als atomare Handlung dargestellt, die von einer oder mehreren Rollen ausführbar sind (Abb. 2.1b). Artefakte werden als Rechtecke (Abb. 2.1c) dargestellt und knapp mit ihrem Informationsgehalt bezeichnet. Als zusätzliche grafische Elemente tauchen Diamanten auf, die zur Konstruktion von Bedingungen und des zeitlichen Ablaufes notwendig sind. Diese Entscheidungspunkte (Abb. 2.1d) erlauben das Einschließen von Entscheidungen und Verzweigungen in Workflows. Der zeitliche Ablauf wird durch gerichtete Kanten zwischen den Aktivitäten gekennzeichnet (Abb. 2.1e), während Informationseinfluss und die Erstellung von Informationen und Produkten über gestrichelte Kanten dargestellt wird (Abb. 2.1f).

Kapitel 3

Ein Test- und Zertifizierungsprozess für Grid-Infrastrukturen

Die Konzeption eines Testprozesses und zugehöriger Testumgebungen ist immer im Kontext des zugehörigen Softwaremanagementprozesses zu betrachten. Zwar ist eine enge Anlehnung nicht an allen Stellen notwendig, es ist aber ratsam Synchronisationszeitpunkte der Entwicklung und des Testvorgehens festzulegen. Dies soll im Folgenden gemacht werden. Daher wird in diesem Kapitel zuerst kurz auf die grundlegenden Aspekte des bereits in Deliverable 2.1.6 entwickelten Softwaremanagementprozess [2] eingegangen. Danach werden die Elemente des entwickelten Softwareprozesses dargestellt und schließlich der auf die Grid-Entwicklung angepasste Workflow vorgestellt, dem ein Gerüst für die Umsetzung einer Testumgebung zur Seite gestellt wird. Der Prozess wird umfassend, aber weitgehend generisch entwickelt, erlaubt also an vielen Stellen eine individuelle Instanziierung nach den Bedürfnissen der jeweiligen umsetzenden Community.

In einem zweiten Teil wird auf Aspekte der Zertifizierung innerhalb des Prozesses eingegangen. Dabei werden insbesondere verschiedene Ebenen im Prozess identifiziert, die Vorbedingung für Zertifizierung und Zertifizierungspunkte darstellen. Auch die Frage der Testumgebung wird in diesem Zusammenhang erneut aufgegriffen.

3.1 Durchführung von Tests im Software-Managementprozess für Gridinfrastrukturen

Da die hier vorgestellten Testprozesse an den generischen Software-Managementprozess im Grid angelehnt werden sollen, wird dieser Prozess der Vollständigkeit halber kurz erläutert. Anschließend werden die für den Prozess benötigten Elemente definiert bevor der Prozess und enthaltene Aktivitäten genau dargestellt werden.

3.1.1 Kurzdarstellung des Software-Managementprozesses

Die Untersuchungen der bisherigen Software-Managementprozesse in Community-Grids und die Rahmenbedingungen der Förderung und der Projektstruktur haben in der generischen Umsetzung des Deliverables 2.1.6 zur Definition eines Vorgehens geführt, das in seiner Grundidee dem evo-

lutionären Wachstumsprozess folgt. Abbildung 3.1 stellt diesen Prozess schematisch dar. Die wesentlichen und treibenden Elemente des Prozesses sind sogenannte Meilensteine, die Kernziele des Vorhabens definiern, Funktionalität und Ziele der Umsetzung vorgeben sowie zeitliche Rahmenbedingungen schaffen.

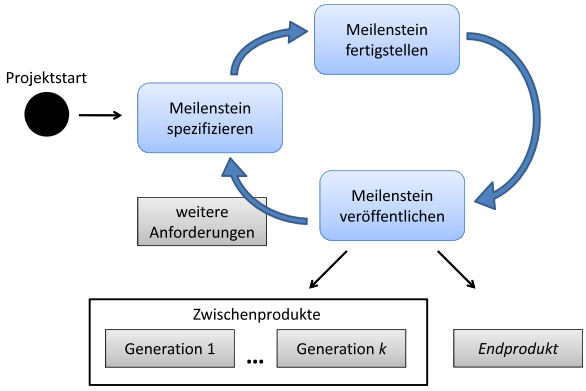

Abbildung 3.1: Schematische Darstellung des Software-Managementprozesses für das Grid, nach Deliverable 2.1.6.

Die Meilensteine definieren jeweils Generationen in der Entwicklungsgeschichte der Software und bestehen selbst aus verschiedenen Teilzielen, die innerhalb eines Meilensteins erreicht werden müssen. Der Prozess sieht aber keinen festen und bis zum Projektende definierten Projektplan vor. Dieser entwickelt und spezifiziert sich über die Generationen der Entwicklung, macht den evolutionären Charakter aus und erlaubt eine dynamische Anpassung der Arbeitsabläufe. Insgesamt besteht daher eine Iteration des Entwicklungsprozesses aus drei elementaren Aktivitäten: Meilenstein spezifizieren, Meilenstein fertig stellen und Meilenstein veröffentlichen.

Im ersten Schritt werden die Anforderungen an die nächste Softwaregeneration analysiert und in einem spezifiziert, woraus sich anschließend eine Aufteilung in Lieferobjekte ergibt. Diese werden nach Prüfung durch den Nutzer und eventuellen Korrekturiterationen festgelegt und bilden die Grundlage für die Definition der Produktbausteine, die schließlich zusammen den Meilenstein ergeben.

Der zweite Schritt umfasst die Bearbeitung und Fertigstellung des Meilensteins. Dieser besteht aus einer großen Anzahl verschiedener Aktivitäten, die auf verschiedene Rollen verteilt sind. Neben dem Projektmanager, der die Meilensteine und Lieferobjekte überwacht, besteht ein iterativer Prozess Lieferobjekte zu erstellen, sie zu Projektbausteinen zusammenzufügen und schließlich den Meilenstein fertig zu stellen und auch zu testen. Hier ist neben der Rolle des Arbeitspaketmanagers, der für die Erstellung der Produktbausteine verantwortlich zeichnet, der Fördernehmer als Produzent der Lieferobjekte beteiligt. Der Integrator und Tester wird im Managementprozess im Kontext der Meilensteinerstellung beschrieben.

Der letzte Schritt einer Generation des Entwicklungsprozesses ist durch die Veröffentlichung des Meilensteins gegeben. Dieser Vorgang enthält Aktivitäten, die auf eine Release-Entscheidung hinwirken, indem Anforderungen verifiziert und – durch eine Rückkopplung in die vorherige Aktivität der Fertigstellung des Meilensteins – Änderungen eingepflegt werden können. Insgesamt können verschiedene Ergebnisse der Veröffentlichung entstehen. Neben dem Release und einem Bericht zum Meilenstein entstehen ggf. Änderungsanforderungen oder ein modifizierter Projektplan, der sich auf die nächsten Generationen auswirkt.

3.1.2 Elemente des allgemeinen Testprozesses

Wie bereits zuvor dargestellt, ist ein Testprozess nicht sinnvoll als alleinstehendes Vorgehensmodell zu betrachten sondern nur im Zusammenhang mit einer Softwareentwicklung. Insbesondere die zeitliche Abfolge der Aktivitäten in einem Testprozess ist eng mit den Abläufen in der Softwareentwicklung verbunden. Dennoch können für einen Testprozess Rollen festgelegt, elementare Komponenten leicht identifiziert und Aktivitäten in einen zeitliche Abfolge gebracht werden, ohne den eigentlichen Softwareentwicklungsprozess im Detail zu betrachten. Eine explizite und sinngebende Verknüpfung kann anschließend durchgeführt und je nach der Ebene der Anwendung des Prozesses instanziiert werden.

Bevor eine konkrete Einordnung des Testprozesses in den Kontext des Softwaremanagements im Grid durchgeführt wird, sollen hier die elementaren Komponenten identifiziert werden.

Entgegen der üblicherweise strikten Aufteilung der Komponentenbeschreibung in Rollen, Aktivitäten und Artefakte und der anschließenden Darstellung des Zusammenwirkens dieser Elemente werden hier anhand des generischen Testworkflows in Abbildung 3.2 die Bedeutung und Interaktion der Komponenten erläutert.

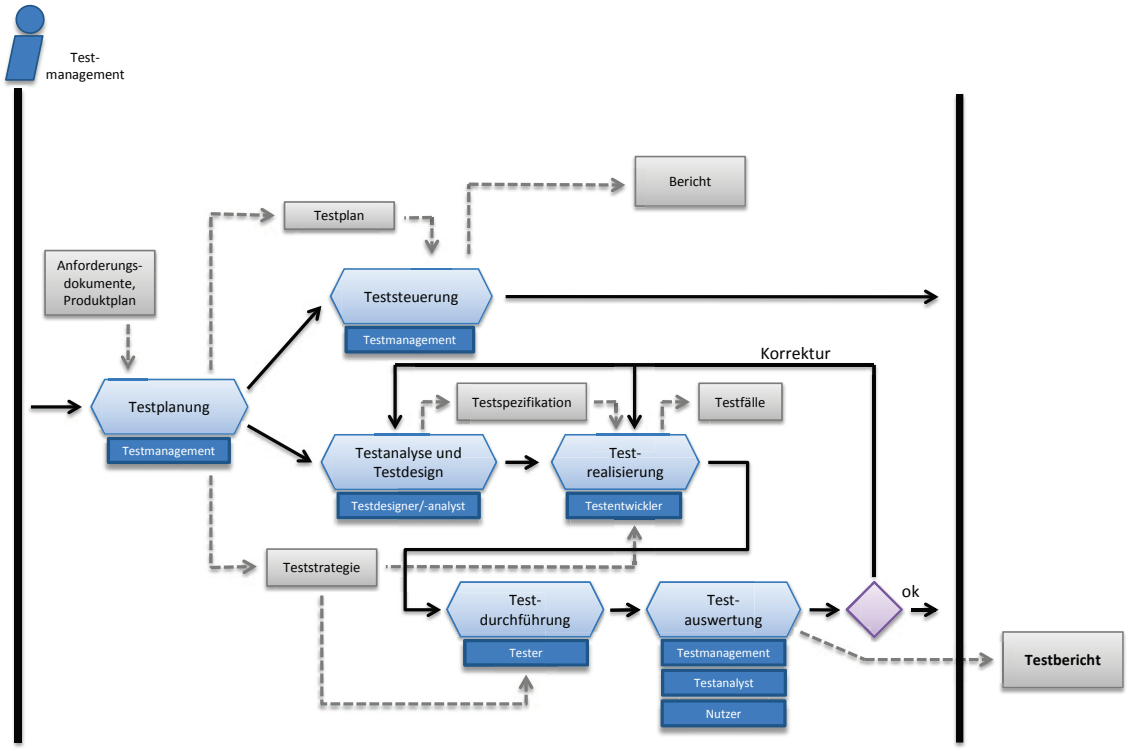

Abbildung 3.2: Generischer Testprozess für die Softwareentwicklung. Die Aktivitäten, Rollen und Artefakte sind auf jeder Ebene der Prozessimplementierung individuell anzupassen.

Startpunkt für einen jeden Prozess ist die *Testplanung*. Diese Aktivität wird von der Rolle des *Testmanagements* durchgeführt und bereitet die konzeptionelle, organisatorische und infrastrukturelle Grundlage des Testens vor. Es werden verschiedene Voraussetzunggen für den weiteren Verlauf des Testprozesses geschaffen, die sich auf das Testkonzept und die zugehörige Testumgebung beziehen. Zum Testkonzept gehört eine Festlegung der benötigten Ressourcen, der Teststrategie, die

Identifikation von Risiken, Prioritäten und die Auswahl geeigneter Testmethoden. Hinzu kommt die Definition von zu überprüfenden Testkriterien. Gleichzeitig muss eine zur Strategie passende Testumgebung festgelegt werden. Dies bezieht sich sowohl auf strukturelle Aspekte wie auf Werkzeuge, die zur Umsetzung verwendet werden.

Während des gesamten Testprozesses übernimmt das Testmanagement ebenfalls die Aufgabe der Teststeuerung. Diese wird auf Grundlage des zuvor ausgearbeiteten Testplans über den gesamten Verlauf durchgeführt. Dabei stützen sich Steuerungsentscheidungen auf Berichterstattung und ggf. werkzeuggestützte Auswertungen des Testprozesses. Neben der Überwachung der aktuellen Testaktivitäten umfasst diese Aufgabe auch einen ständigen Abgleich des aktuellen Testprozesses und der Planung sowie ggf. eine Anpassung der Planung aufgrund neuer Situationen. Als Ergebnis der Teststeuerung entstehen Berichte, die den Verlauf und die Verwaltung des Prozesses nachvollziehbar machen.

Der eigentliche Testvorgang geschieht parallel zur Teststeuerung und beginnt mit der *Testanalyse* und dem *Testdesign*, welche durch den *Testdesigner* durchgeführt werden. Dabei werden auf Grundlage der Anforderungsspezifikationen und der Produktplanung Testszenarien entworfen, die für die spätere Umsetzung als Grundlage und Richtschnur dienen. Die Analyse umfasst drei wichtige Schritte:

1. *Analyse der Testbasis*: Als Ausgangspunkt der Analyse werden Dokumente und Spezifikationen der eigentlichen Entwickler zusammengetragen und evaluiert. Es wird auf Spezifikationen der Anforderungen und der Architektur ebenso zurückgegriffen wie auf Dokumentationen der Entwicklung. Für diese wird geprüft, ob sie zur Erstellung eines Testszenarios ausreichend sind.

2. *Betrachtung des Testobjekts*: Nachdem die grundlegenden Informationen vorhanden sind, können Testobjekte für den Testprozess festgelegt werden. Dabei werden die Anforderungen an ein solches Objekt festgeschrieben und erwartete Ergebnisse und Systemverhalten spezifiziert.

3. *Konstruktion der Testfälle*: Anhand der Testspezifikation und einer Auswahl geeigneter Testmethoden (siehe Abschnitt 2.1.2) werden Testfälle konstruiert. Dabei wird zuerst ein Schwerpunkt auf den Entwurf logischer Testfälle gelegt, die das prinzipielle Vorgehen definieren. Eine Festlegung konkreter Testfälle geschieht erst im Schritt der Testrealisierung.

Die *Testrealisierung* setzt nun die Vorgaben der vorherigen Aktivität um und realisiert die Testinfrastruktur und den vorgegebenen Testrahmen im Detail. Dies umfasst neben der Überführung von logischen Testfällen in konkrete Testfälle auch die sinnvolle Gruppierung und Sequenzierung von Einzeltests zu Testszenarien durch die Rolle des *Testentwicklers*. Dabei ist es wichtig, die Vollständigkeit der Testfälle zu überprüfen, sie also so anzulegen, dass alle spezifizierten Funktionen und Anforderungen überprüft werden und die konstruierten Fälle in der Testumgebung durchführbar sind. Als Ergebnis des Schrittes sind spezifizierte Testfälle vorgesehen, die als Grundlage für die eigentliche Durchführung dienen.

Die Durchführung der Tests wird durch den *Tester* vorgenommen. Wichtige Aspekte bei der Durchführung sind insbesondere die Einhaltung von Prioritäten der Tests. Dabei werden zuerst Hauptfunktionen getestet, später Nebenfunktionen. Eine wichtige Aufgabe ist hier die Protokollierung der Tests, um eine umfassende Nachvollziehbarkeit und Reproduzierbarkeit der Tests zu ermöglichen. Eine weitere, für die Behebung gefundener Fehler ausschlaggebende Aufgabe ist die

Prüfung auf Fehlerwirkung und Ursache. Wenn möglich soll bereits der Tester auf Grundlage der Spezifikationen und der durchgeführten konkreten Testfälle eine Aussage zu Fehlergründen und Effekten machen und dokumentieren.

Die zuvor gemachten Tests und die entsprechende Protokollierung bilden den Abschluss eines Testzyklus in Form der *Testauswertung*. Diese Aktivität wird von dem Testmanagement, dem Testanlyst und ggf. unter Beteiligung des Nutzers durchgeführt. Die Ergebnisse fließen einerseits in den Entwicklungsprozess zurück, müssen aber außerdem ausführlich auf ihre Aussagefähigkeit überprüft werden. Hier steht die Frage im Vordergrund, ob die angewendeten Tests und zuvor erstellten Spezifikationen ausreichend für eine abschließende Bewertung der Entwicklung sind (Review des Testverfahrens). Gegebenenfalls wirken die Auswertungsergebnisse nicht nur auf den Entwicklungsprozess sondern auch auf den Testprozess zurück.

Der Testprozess endet mit einem Testbericht, der die Aktivitäten, Spezifikationen und Testfälle dokumentiert, die Durchführung protokolliert und entsprechende Hinweise für den Entwicklungsprozess liefert. Wie bereits angedeutet soll der Testbericht auch eine Evaluation des Testverfahrens enthalten um zukünftige Testiterationen verbessern und die durchgeführte Iteration nachträglich bewerten zu können.

3.1.3 Integration in den Software-Managementprozess für Grids

Die Integration des zuvor vorgestellten generischen Testprozesse ist im allgemeinen Software-Managementprozess für die Entwicklung von Grid-Infrastrukturen auf verschiedenen Ebenen mit unterschiedlichem Fokus möglich. An dieser Stelle sollen drei Ebenen betrachtet werden, die eine jeweils individuelle Umsetzung des Prozesses erlauben.

Auf oberster Ebene ist der Prozess in den evolutionären Prozess der Meilenstein-getriebenen Entwicklung möglich. Abbildung 3.3 stellt dies schematisch dar und repräsentiert auf der Ebene einen Prozess für Systemtests: Aufbauend auf der Spezifikation des Meilensteins werden Anforderungen und Funktionen zur Testanalyse und zum Testdesign herangezogen und in der Testrealisierung zu Testfällen formuliert. Nebenläufig zum Prozess der Meilensteinbearbeitung wird die Umsetzung der Testinfrastruktur und der Testfälle vorangetrieben. Bei Fertigstellung des Meilensteins beginnt die Testdurchführung. Der Veröffentlichung des Meilensteins geht die Testauswertung voraus, die einen Testbericht als Produkt erzeugt. Dieser Bericht fließt im Veröffentlichungsschritt des Software-Manangementprozesses in die Verifikation der Anforderungen und die Releaseentscheidung ein, hat also indirekt auch Einfluss auf Änderungsanforderungen und eventuelle Modifikationen des Projektplanes sowie die Meilenstein-Spezifikation der nächsten Softwaregeneration.

Eine weitere Umsetzung des Testprozesses lässt sich auf Ebene der Aktivität 'Meilenstein fertig stellen' im allgemeinen Software-MAnagementprozess in Form eines Integrationstests finden. Wie bereits dargestellt besteht diese Aktivität aus der parallelen Entwicklung von Produktbausteinen durch die Fördernehmer. Deren Zusammenstellung zu Lieferobjekten wird vom Arbeitspaketmanagement überwacht und koordiniert. Ebenso werden in einem späteren Schritt die Lieferobjekte zum endgültigen Meilenstein zusammengefügt. An beiden Stellen ist es notwendig die Integration der Produktbausteine zu Lieferobjekten sowie der Lieferobjekte zum Meilenstein zu testen. Hier kann der generische Prozess ebenfalls an die einzelnen Aktivitäten angelehnt werden.

Schließlich kann der generische Prozess zusätzlich auf der Ebene der Produktbausteinerstellung durch den Fördernehmer angesiedelt werden. Insbesondere die lokale Ausprägung und Durchführung

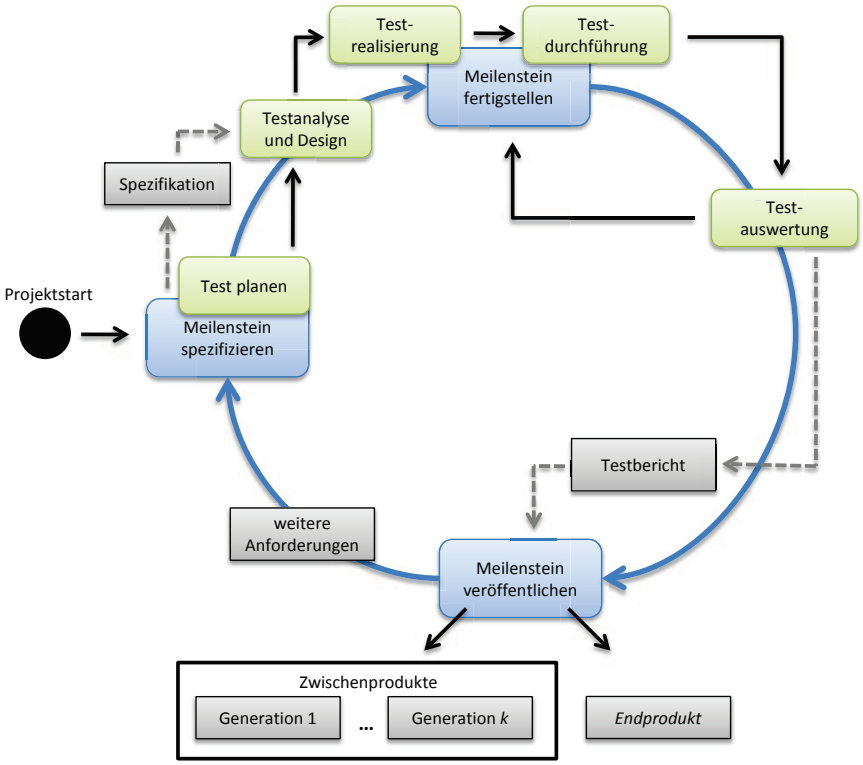

Abbildung 3.3: Integration des generischen Testprozesses in die Evolutionsschleife des Softwaremanagementprozesses für das Grid (Meilensteinebene).

ist jedoch im Kontext der Umsetzung von Community-Grid-Projekten dem Fördernehmer vorbehalten. Da wie in Deliverable 2.1.6 ein Fördernehmer als Projektpartner weitgehend eigenständige Entwicklungen durchführen kann, ist er auch für die Qualität seiner Produkte direkt verantwortlich. Die Implementierung eines geeigneten Testprozesses ist daher dringend notwendig. Dabei muss zwischen der individuellen Planung und Durchführung der Tests und der Planung und Auswertung der Tests unterschieden werden. Für die Gesamtqualität des Projektes und auch für die Durchführung der zuvor beschriebenen Integrations- und Systemtests ist eine spezifizierte Qualität der lokalen Einzelentwicklungen wichtige Voraussetzung. Aus diesem Grund muss es für Entwicklungen in und von Community-Grids eine Trennung der Verantwortlichkeiten und der Kontrolle über den Testprozess auf Ebene der Fördernehmer geben. Die Testanforderungen und auch die Testauswertung sollte nicht ausschließlich in den Händen der Fördernehmer liegen, sondern muss Teil der Aktivität der Testplanung und Testanalyse des übergeordneten Testprozesses zur Integration der Produktbausteine zu Lieferobjekten sein. Im folgenden Kapitel wird auf diesen Aspekt noch einmal aus Sicht der Zertifizierung eingegangen.

3.2 Zertifizierung im Entwicklungs- und Testprozess

Der Begriff der Zertifizierung bezeichnet im allgemeinen Sinne ein Verfahren, mit dem man die Einhaltung bestimmter Spezifikationen oder Anforderungen an Produkten, Dienstleistungen oder Systeme sicherstellt. Dieses Verfahren basiert konsequenterweise auf der Überprüfung von Spezifikationen

und Anforderungen und ist somit relativ eng mit Testprozessen verbunden. Sie sind ein elementarer Bestandteil der Konformitätsprüfung einer Zertifizierung, da sie – wie bereits dargestellt – die Methodik zur Qualitätssicherung aber auch Qualitätsbeurteilung im Software-Managementprozess darstellen.

Im Bereich der Softwareentwicklung ist die Zertifizierung oft mit der Frage der Versionierung verknüpft. Die funktionalen und nicht-funktionalen Anforderungen einer Software werden oftmals als Ziele in Verbindung mit einer Version der Software spezifiziert. Eine solche Version beschreibt damit eine Art Synchronisationspunkt in der Entwicklung an dem alle Komponenten zusammenarbeiten müssen. Insbesondere der Softwareprozess aus Deliverable 2.1.6 betont durch sein evolutionäres Konzept eine Versions- oder Generationenfolge durch Meilensteine, die als solche Synchronisations- und Zertifizierungspunkte angesehen werden können.

Die folgenden Abschnitte schaffen einen allgemeinen Rahmen für die Etablierung eines solchen Zertifizierungsprozesses während der Grid-Infrastrukturentwicklung. Neben den wichtigen Voraussetzungen für die Zertifizierung wird spezifisch auf den Kontext der Entwicklung von Community-Grids anhand des evolutionären Prozesses zum Softwaremanagement eingegangen. Gleichzeitig wird die Konzeption von Produktivumgebungen und Testumgebungen betrachtet, die ebenfalls als wichtige Voraussetzung für einen Zertifizierungsprozess gilt.

3.2.1 Ausgangspunkt und Voraussetzung für die Zertifizierung

Der Ausgangspunkt für die Realisierung einer Zertifizierung ist einerseits die lokale Qualitätssicherung und andererseits die Existenz einer geeigneten Testumgebung neben der eigentlichen Produktivumgebung in der die Software läuft. Hier sollen beide Aspekte Grid-spezifisch diskutiert werden.

Lokale Qualitätssicherung

Die lokale Qalitätssicherung bezieht sich im Kontext der Grid-Entwicklung auf das Testen einzelner Produktbausteine aber auch ganzer Komponenten, die in ihrem Zusammenspiel ein integriertes System bilden. Da viele Grid-Architekturen aus einer Middlewareschicht mit höherwertigen Diensten im Sinne einer SOA besteht, ist deren Entwicklung weitgehend unabhängig zu betrachten und kann somit von einzelnen Partnern oder kleinen Entwicklergruppen durchgeführt werden. Erst später geschieht eine Integration zu einem Gesamtsystem.

Bevor das Gesamtsystem entstehen kann, müssen die einzelnen Dienste und Komponenten ausreichend getestet werden. Ein solcher Test ist dringend notwendig, um ein gemeinsames Mindestmaß an Produktqualität sicherzustellen. Bei den Tests handelt es sich um Komponententests (siehe Abschnitt 2.1.2). Damit aber nicht jede Entwicklergruppe eigene Tests und insbesondere Testkriterien entwickelt, müssen gemeinsame Richtlinien vorliegen, die von allen als Qualitätsstandard verfolgt werden. Diese Richtlinien sollten sich im Sinne der Komponententests folgende Aspekte berücksichtigen:

Verifikation der Schnittstellen: Die Tests sollen die Korrektheit der Schnittstellen entlang der meist Projekt-übergreifend festgelegten Spezifikationen geprüft werden. Hierzu können einfache Ein- und Ausgabetests mit einem Test-First-Ansatz umgesetzt werden. Bereits vor oder spätestens zu Beginn der Entwicklung werden Testtreiber für die definierten Schnittstellen

erstellt, die nach der Fertigstellung der Komponente angewendet werden. Richtet sich der Treiber strikt nach der Schnittstellenspezifikation, kann durch dieses Testvorgehen jede Abweichung detektiert werden.

Funktionale Konsistenz: Anschließend an die eigentliche Verifikation der Scnhittstellen sollte deren spezifizierte Funktion und ihr Verhalten überprüft werden. Dies kann ebenfalls in den zuvor beschriebenen Test-First-Ansatz integriert werden, indem das Komponentenverhalten auf verschiedene Ein- und Ausgaben getestet wird. Dabei werden spezifische Eingaben gemacht und die entstehenden Ausgaben gegen eine erwartete Vorgabe evaluiert. Abweichungen von der erwarteten Ausgabe deuten dann auf einen internen, algorithmischen Fehler hin.

Neben der technischen Überprüfung von Komponenten anhand von Spezifikationen ist es ausschlaggebend, zwei wichtige Kennzahlen als Qualitätsziel festzulegen. Diese betreffen die Menge des getesteten Codes und die Abdeckung der Komponente durch geeignete Tests.

Codecoverage durch Whiteboxtests: Diese Metrik Codecoverage bezeichnet, wieviel des Codes einer Komponente von Tests abgedeckt wird. Ziel ist es, alle Codepfade, die in einer Komponente existieren, mindestens einmal zu durchlaufen. Dazu ist allerdings (im Gegensatz zu den zuvor besprochenen Testmethoden) ein umfassender Einblick in den Quellcode notwendig. Die Testfälle werden so aufgebaut, dass einzelne Anweisungen, Verzweigungen und Bedingungen durchlaufen werden. Dabei ist es wichtig, zuvor einen angestrebten *Überdeckungswert* festzulegen, der von jedem Test erreicht werden muss. Gewöhnlicherweise liegt dieser Wert recht hoch, oft $> 90\%$.

Testcoverage: Eine verwandte Betrachtung kann auf Ebene aller Testverfahren genutzt werden. Dabei wird initial festgelegt, welche Testabdeckung erreicht werden soll, also welche Mindestmenge von Tests auf eine Komponente anzuwenden ist. Dies stellt sicher, dass mehrere Verfahren angewendet werden und damit ein großer Querschnitt der Funktionalität und des Codes durch Tests abgedeckt sind.

Die vorgestellten Aspekte bilden damit eine Grundlage für die Definition eines gemeinsamen Qualitätsniveaus, das durch die verteilt entwickelten Komponenten im Softwareprozess für Grid-Infrastrukturen erreicht werden muss, um eine Zertifizierung durchführen zu können.

Aufbau einer Testumgebung

Als weitere wichtige Voraussetzung für einen Zertifizierungsprozess ist eine neben der Produktivumgebung existierende Testumgebung zu nennen. Die oft verteilte Architektur einer Grid-Infrastruktur basiert auf einzelnen Diensten, die nur lose gekoppelt über spezifizierte Schnittstellen und Protokolle zusammenarbeiten. In diesem Kontext ist es normal, dass die einzelnen Dienste selbst nach einer ausführlichen Qualitätsprüfung nach der Entwicklung eigenständig funktionsfähig sind, jedoch die Interaktion mit anderen Diensten nicht selten scheitert. Dies liegt oft an der unterschiedlichen Interpretation von Schnittstellenbeschreibungen oder Protokollen. Eine Integration fertiger und lokal getesteter Dienste in ein produktives System ist daher nicht sinnvoll.

Daraus resultiert die Erkenntnis, dass neben der eigentlichen Produktivumgebung eine verteilte Testinfrastruktur bestehen soll, die eine Überprüfung der Funktionalität der neuen Komponenten

erlaubt, ohne dass das bestehende System davon beeinträchtigt wird. Abbildung 3.4 stellt einen solchen Parallelbetrieb von Produktiv- und Testumgebungen dar. Zusätzlich zum gleichzeitigen Betrieb der Infrastrukturen muss ein Prozess gefunden werden, der idealerweise teilautomatisiert einerseits das Testen lokaler Komponenten unterstützt und deren Integration als Gesamtsystem zertifiziert. Ein solcher Prozess wird im anschließenden Abschnitt vorgeschlagen.

3.2.2 Zertifizierungsprozess

Die Zertifizierung einer Software im Kontext der Grid-Infrastruktur bezieht sich auf die feste Spezifikation von Anforderungen und Funktionalität und ist damit implizit einem Versionierungsansatz untergeordnet. Eine implizite Versionierung findet sich ebenfalls im evolutionären Software-Managementprozess wieder und kann hier als Einstiegspunkt für die Entwicklung des Zertifizierungsprozesses genutzt werden.

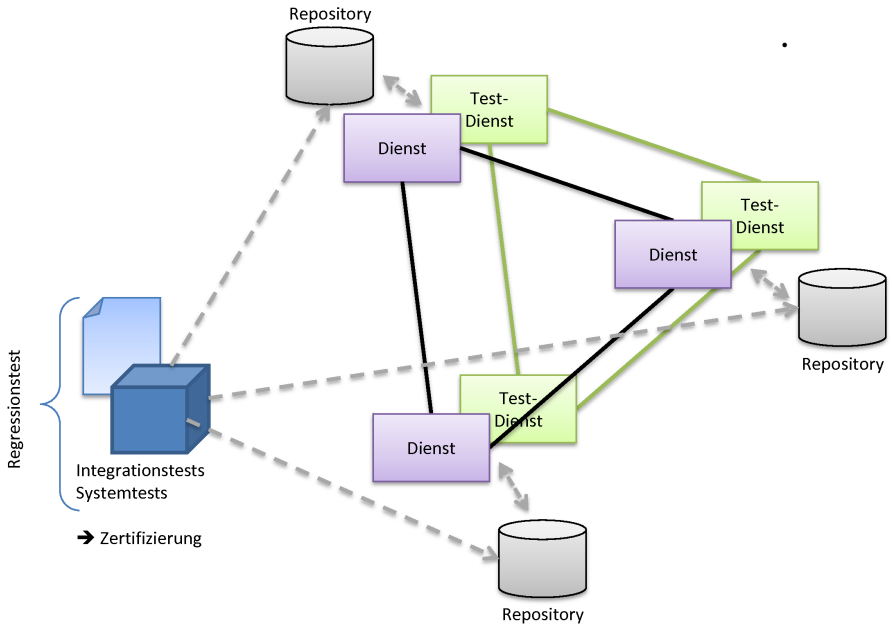

Abbildung 3.4: Skizze des Zertifizierungsprozesses mit Produktivumgebung (violett) und Testumgebung (grün).

Im Folgenden wird jeder Meilenstein oder jede Generation des Entwicklungsprozesses als eine Hauptversion betrachtet, gegen die eine Zertifizierung durchgeführt wird. Damit besteht eine Hauptversion aus verschiedenen Elementen, die spezifiziert werden müssen:

- Funktionsumfang: Diese Spezifikation bestimmt die von der Hauptversion zur Verfügung gestellte Mindestfunktionalität und beschreibt diese im Detail. Sie ist einerseits Richtlinie für die Entwicklung, muss gleichzeitig aber derart detailliert sein, dass daraus Testfälle für Funktionstests auf Gesamtsystemebene erzeugt werden können.
- Schnittstellen: Die vollständige Definition der Komponentenschnittstellen ist sowohl Ausgangspunkt für die gemeinsame Entwicklung an einem verteilten System aber auch wichtige Voraussetzung für die Überprüfung der Integration und des Zusammenspiels der unterschiedlichen Komponenten.

- Nicht-funktionale Anforderungen: Zu diesem Punkt gehören verschiedene Aspekte, die die Betriebsqualität, Nutzbarkeit und Stabilität des Gesamtsystems betreffen. Es ist hier wichtig in einem Dokument die nicht-funktionalen Anforderungen an Robustheit, Leistungsfähigkeit, Skalierbarkeit und Nutzbarkeit festzulegen, damit sie ebenfalls in die Zertifizierung und die zugehörigen Testverfahren einbezogen werden können.
- Qualitätsziele: Neben den vorherigen Aspekten sind zusätzlich stärker Test-bezogene Kriterien zu berücksichtigen. Diese beziehen sich auf Festlegungen der Testabdeckung (Testcoverage) und der Quellcodeabdeckung (Codecoverage) für alle angewendeten Tests. So kann etwa für alle Komponenten eine Testabdeckung von mindestens 80% und eine Codeabdeckung von mindestens 90% gefordert werden. Gleiches gilt für die Testabdeckung der Integrations- und Systemtests auf Grundlage der zuvor aufgeführten Zertifizierungsaspekte.

Mittels der Spezifikation der Hauptversion ist nun eine Zertifizierung auf der Ebene der Integrations- und Systemstests möglich. Auf der Grundlage der festgelegten Funktionalität, Schnittstellen, nicht-funktionalen Anforderungen und Qualitätszielen können Testfälle entwickelt werden, die unter Regressionstest eine kontinuierliche Zertifizierung erlauben.

Entsprechend der Darstellung in Abbildung 3.4 muss für die Zertifizierung eine zentrale Stelle geschaffen werden, die auf Grundlage der Spezifikationen der Hauptversion die Integrations- und Systemtests überwacht. Beide Testarten müssen in diesem Zusammenhang als Regressionstests konzipiert sein, die regelmäßig und vollständig auf das integrierte System angewendet werden. Das System kann somit nur als ganzes bezügliche einer Hauptversion zertifiziert werden.

Gleichzeitig ist an dieser Stelle auf eine Teilautomatisierung des Zertifizierungsprozesses hinzuwirken, indem eine entsprechende Build-Umgebung entworfen wird. Strukturell stellt bereits Abbildung 3.4 eine entsprechende Umgebung dar. Da die Komponentenentwicklung (Dienste) voneinander unabhängig in lokalen oder kleinen verteilten Teams durchgeführt wird, kann die Codebasis des Gesamtsystem entsprechend der Diensteaufteilung auf verteilten Repositories verteilt werden, die nur von der jeweiligen Entwicklergruppe genutzt werden. Hier können lokale Tests entsprechend des generischen Prozesses aus Abschnitt 3.1.2 genutz werden, um die Qualitätssicherung auf lokaler Ebene sicherzustellen und die Voraussetzungen aus Abschnitt 3.2.1 für die Zertifizierung zu schaffen. Parallel dazu kann eine weitgehend automatische Integration der Komponenten durch das zentrale Management geschehen, die schließlich das Zusammenspiel, Funktionalität und die Systemeigenschaften mittels Regressionstest kontinuierlich prüft und zertifiziert. Erst ein zertifiziertes Gesamtsystem wird in die Produktivumgebung überführt.

Literaturverzeichnis

[1] Rapp A., Grimme, C., Enke, H. (Editoren), Deliverable 2.1.1 Community - Überblick / Report, WissGrid, 2009.

[2] Grimme, C., Enke H. (Editoren), Leitfaden Software-Management-Prozesses, WissGrid, 2010.

[3] Versteegen, G., Software Management. Springer, Heidelberg, 2002.

[4] Salomon, K., Das ständige Testen nicht vergessen, in Versteegen, G., Software Management. Springer, Heidelberg, 2002.

[5] Hoheisel, A., Sommerfeld, D., Peter, K: Guidelines for the Deployment of Applications in MediGRID,Version 1.4

[6] Spillner, A., Linz, T., Basiswissen Softwaretest, dpunkt.verlag, Heidelberg, 2010.