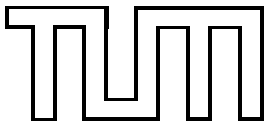
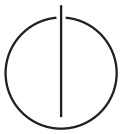


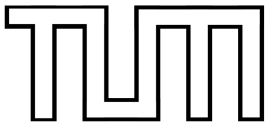
TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK



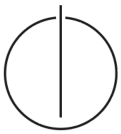
Lehrstuhl für
Angewandte Softwaretechnik

Ein Modell zum begründungsbasierten Freigabemanagement

Korbinian Herrmann



TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK



Lehrstuhl für
Angewandte Softwaretechnik

Ein Modell zum begründungsbasierten Freigabemanagement

Korbinian Herrmann

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. U. Baumgarten
Prüfer der Dissertation:	1. Univ.-Prof. Dr. J. Schlichter 2. Univ.-Prof. B. Bruegge, Ph.D.

Die Dissertation wurde am 24.06.2009 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 21.10.2009 angenommen.

Für Annette, Veronika und Ferdinand
– K.H.

Danksagung

Mein Dank geht an erster Stelle an Bernd Bruegge für seine intensive Unterstützung, insbesondere durch seine Visionen und Reviews. Ich danke Johann Schlichter für sein Interesse am Thema Freigabemanagement sowie sein überaus schnelles und genaues Feedback. Weiter danke ich meinen Kollegen vom Lehrstuhl für Angewandte Softwaretechnik, insbesondere dem Sysiphus Projektteam für Diskussionen sowie Monika Markl und Helma Schneider für die organisatorische Unterstützung. Vielen Dank an meine Familie für die viele Ruhe zu Hause, in der ich die vorliegende Arbeit schreiben durfte.

Inhalt

Zusammenfassung

Software-Systeme sind komplex und unterliegen ständigen Änderungen, die Softwareprojekte zum Scheitern bringen können. Um Projekte mit Änderungen erfolgreich durchzuführen, sehen agile Vorgehensmodelle kurze, teils nur wenige Wochen dauernde, Freigabezyklen vor. War das Freigabemanagement bisher vorwiegend auf die Lieferung des Systems am Projektende ausgerichtet, wird es nun zur Projektfunktion und begleitet somit das gesamte Projekt. Damit wächst die Bedeutung des Roadmappings, der strategischen Planung einer Folge von Freigaben: Erste Freigaben werden nun bereits kurz nach Projektstart erstellt und an den Kunden geliefert. Als Konsequenz müssen Projektmanager stärker als früher mit verschiedenen Änderungen, die in einem Softwareprojekt auftreten, umgehen können. Die Entscheidung, ob eine Änderung durchgeführt werden kann ohne die Lieferung der Freigabe zur Deadline zu gefährden, stellt eine wichtige Forschungsfrage dar. [Penny 2002] Existierende Werkzeuge und Methoden zur Lösung dieser Frage basieren lediglich auf der Abschätzung von Ressourcen oder wirtschaftlichen Abwägungen, wie Kosten/Nutzen-Rechnungen für Anforderungen, berücksichtigen jedoch nicht die dem System zugrunde liegende Architektur. [Vähäniitty et al. 2002]

Der Beitrag dieser Dissertation ist ein neuartiges Modell zum begründungsbasierten Freigabemanagement, kurz BEEF: Es unterstützt die Entscheidungsfindung im Freigabemanagement bei Änderungen, indem es Systemmodelle, Kommunikationsmodelle und Organisationsmodelle sowie deren Abhängigkeiten für mehrere aufeinanderfolgende Freigaben berücksichtigt. Das Modell zum begründungsbasierten Freigabemanagement erlaubt, Systemmodelle beliebig detailliert darzustellen, um Aspekte, die von der

Analyse über die Software-Architektur bis zur Implementierung reichen, als Entscheidungsgrundlage heranziehen zu können: Beispielsweise können ausgehend von einer Methode über Objekte aus Modellen verschiedenen Abstraktionsgrades betroffene Anforderungen ermittelt werden. Freigabeoperationen in BEEF vereinfachen den Umgang mit diesen Abhängigkeiten, indem sie zusammen mit Anforderungen auch von ihnen abhängige Modelle zwischen Freigaben verschieben.

Zur Darstellung der Modelle aufeinanderfolgender Freigaben führt diese Dissertation das Konzept der Knowledge Nuggets ein. Ein Knowledge Nugget vereinfacht das Management von Freigaben, insbesondere durch die Verwaltung von Systemmodellen, Kommunikationsmodellen und Organisationsmodellen und ihrer Abhängigkeiten untereinander. Knowledge Nuggets verbessern Entscheidungen während der Weiterentwicklung eines Systems von Freigabe zu Freigabe, indem sie eine detaillierte Modellierung und Erkundung der Freigaben einer Roadmap und alternativer Lösungsvorschläge gleichermaßen erlauben, die sowohl vage Anforderungsmodellen als auch detaillierte, implementierungsnahe Modelle zur Entscheidungsfindung heranziehen.

Ein weiterer Beitrag dieser Dissertation ist die empirische Evaluierung des begründungsbasierten Freigabemanagements in einer Fallstudie. In dieser Fallstudie wurden die Auswirkungen von 184 Änderungen beobachtet: In über 43% der Änderungen lagen komplexe Abhängigkeiten vor, welche die Teilnehmer mit EXPLoRE analysierten und dann auch unter Verwendung von Freigabeszenarios eine Entscheidung trafen, die von 80% der Teilnehmer als gut beurteilt wurde. 80% der Teilnehmer gaben in der Befragung an, dass die erfassten System-, Kommunikations- und Organisationsmodelle mit ihren Abhängigkeiten dazu dienen, die Herausforderungen im Freigabemanagement zu reduzieren.

Abstract

Software projects often fail as software systems are complex and change frequently. To perform projects with changes successfully, agile life cycle models propose short release cycles, often lasting only few weeks. Whereas release management has focused on the delivery of the system at the end of a project up to now, now it is turning in a project function accompanying the whole project. Thus, significance of roadmapping, i.e. the strategical planning of a series of releases, increases: First releases are already produced and delivered soon after project kick-off. As a consequence, project managers must deal with different types of changes occurring in a software project. The decision to accept a change without delaying the delivery of a system is an important research question. [Penny 2002] Existing tools and methods to solve this question, only base on estimating resources or on economic considerations such as cost-benefit analysis for requirements, but do not consider the architecture of the system.

The contribution of this dissertation is a novel model, called rationale-based release management model (BEEF). It supports release management decisions if changes occur by considering system, communication and organization models as well as their dependencies for several consecutive releases. The rationale-based release management model represents system models at any level of abstraction. These serve to provide rationale ranging from analysis, software architecture to implementation for decisions in release management: For instance, one can trace from a method over objects in the models of different levels of abstraction to affected requirements. Release operations allow dealing with these dependencies in an easier way

by moving dependent system models together with their requirements between releases.

To represent models of consecutive releases this dissertation introduces the concept of knowledge nuggets. A knowledge nugget makes the management of a release easier, especially by managing system, communication and organizational models as well as their dependencies. Knowledge nuggets improve decisions during the development of a system from release to release: They support making a good decision by precise modeling and exploring of releases in a roadmap and of alternative solutions in the same way, including vague requirements as well as detailed implementation-centric models.

Another contribution of this dissertation is the empirical evaluation of the rationale-based release management model in a case study. In this case study we observed the consequences of 184 changes: In more than 43% of the changes there were complex dependencies, which the participants analyzed using the rationale-based release management model and made a decision after exploring alternative solutions. 80% of the participants judged that decisions made were good. 80% of the participants stated in an interview that the system, communication and organization models with their dependencies serve to reduce challenges in release management.

1 Einleitung

Heutige Software-Systeme sind komplex und unterliegen ständigen Änderungen. Als Konsequenz häufiger Änderungen erfolgt die Lieferung von Systemen oft zur Unzufriedenheit der Kunden verspätet und/oder in schlechter Qualität. Agile Vorgehensweisen akzeptieren Änderungen als integralen Bestandteil der Systementwicklung und unterstützen sie durch kurze, meist nur wenige Wochen dauernde Freigabezyklen. Dem Freigabemanagement kommt dadurch eine neue Bedeutung zu, denn kurze Lieferzyklen verlangen eine unmittelbare Reaktion auf eine Änderung: Handelt es sich um eine Änderung am System, beispielsweise um eine neue Anforderung oder einen Fehlerbericht, muss entschieden werden, in welcher Freigabe die Änderung durchgeführt werden soll und kann und was die Konsequenzen für die Einführung sind, das heißt ob es dadurch notwendig wird, zum Beispiel andere Anforderungen auf spätere Freigaben zu verschieben. Nicht immer ist das System die Ursache einer Änderung, auch Änderungen aus der Organisation, beispielsweise ein Mitarbeiterausfall, können bei kurzen Freigabezyklen schnell dazu führen, dass ein Teil der Funktionalität nicht mehr erstellt werden kann und die Lieferung der Freigabe in Gefahr gerät.

1.1 Ziel und Ansatz

Hypothese

Die Hypothese dieser Dissertation ist, dass ein Freigabemanager beginnend vom Projektstart parallel zu Aktivitäten der Softwareentwicklung, wie Analyse, Systementwurf, detailliertem Entwurf, Implementierung sowie Integration und Testen, bei Eintritt einer Änderung deren Auswirkungen auf die nachfolgenden Freigaben analysiert und bessere Entscheidungen erreicht,

wenn Systemmodelle, Kommunikationsmodelle und Organisationsmodelle sowie wirtschaftliche Betrachtungen als Begründungen klar dargestellt sind.

Begründungen

Systemmodelle sind vereinfachte Darstellungen der technischen Aspekte eines Software-Systems, beispielsweise der Anforderungen oder einer Software-Architektur. Kommunikationsmodelle verbessern die Entscheidungsfindung, indem sie die Kommunikation in einem Softwareprojekt festhalten und Begründungen für Entscheidungen bei konkreten Fragestellungen der Software-Entwicklung bereitstellen. Organisationsmodelle beschreiben die Organisation eines Projekts oder Unternehmens und bilden Mitarbeiter mit Verantwortlichkeiten, Fähigkeiten und ihrer Verfügbarkeit für ein Projekt ab. Wirtschaftliche Betrachtungen analysieren die Auswirkungen von Änderungen auf den Markt oder Kunden, beispielsweise indem sie die Kosten und den Mehrwert von Änderungen abschätzen.

Begründungsbasiertes Freigabemodell

Diese Dissertation stellt ein begründungsbasiertes Freigabemodell vor, das den Freigabemanager in seinen Entscheidungen unterstützt und den Freigabeprozess erheblich verbessert. Das begründungsbasierte Freigabemodell, kurz BEEF, führt Knowledge Nuggets als neues Konzept ein, um die in einer Roadmap definierte Folge von Freigaben zu modellieren. Ein Knowledge Nugget kennt für jede Freigabe Systemmodelle, Kommunikationsmodelle und Organisationsmodelle sowie deren Abhängigkeiten. Knowledge Nuggets dokumentieren bereits gelieferte Freigaben und planen zukünftige Freigaben, indem sie Freigaben unterschiedlich detailliert modellieren: Gelieferte Freigaben sind genau modelliert, während die Modelle zukünftiger Freigaben noch vage sind. Aus diesen Modellen aufeinanderfolgender Freigaben generiert BEEF Freigabepläne, die sich automatisch an Änderungen der Modelle anpassen. Knowledge Nuggets erleichtern auch Änderungen an den Freigabeplänen, indem sie abhängige Artefakte zusammen mit einer Anforderung auf spätere Freigaben verschieben und dadurch Freigabemanagern und Entwicklern in gleicher Weise eine konsistente Sicht einer Roadmap liefern.

Alternativen

Bei Eintritt einer Änderung stehen oft mehrere Lösungsalternativen zur Auswahl. Existierende Ansätze, die alternative Lösungsvorschläge vorsehen, wie [Greer & Ruhe 2004], unterscheiden die Lösungsvorschläge bisher nur

auf der Grundlage von Kriterien. BEEF verfolgt für den Vergleich alternativer Lösungsalternativen, in BEEF Freigabeszenarios genannt, neben der freien Wahl von Kriterien [Herrmann 2007b] einen neuartigen Ansatz, indem es Knowledge Nuggets für jedes Freigabeszenario einsetzt. Knowledge Nuggets für Freigabeszenarios sind ein mächtiges Hilfsmittel, um jedes Freigabeszenario auf mehreren Abstraktionsgraden beliebig detailliert zu modellieren und somit um die Aspekte der Software-Architektur, Implementierung, Organisation und Kommunikation als Grundlage der Entscheidungsfindung heranzuziehen. Freigaben, Freigabeszenarios, Abstraktionsgrade, Versionen, Kriterien sowie organisatorische Rahmenbedingungen bilden in BEEF einen mehrdimensionalen Entscheidungsraum, den der Freigabemanager erkundet und bei seiner Entscheidungsfindung heranzieht.

EXPLoRE

Das begründungsbasierte Freigabemodell ist in einem Werkzeug EXPLoRE implementiert, das den Freigabemanager bei der Entscheidungsfindung unterstützt. Dazu stellt EXPLoRE eine dreidimensionale Visualisierung des mehrdimensionalen Entscheidungsraums bereit, mit der der Freigabemanager durch den Entscheidungsraum navigiert und Änderungen in den Modellen betrachtet. EXPLoRE unterstützt die Entscheidungsfindung, indem es Unterschiede in den Modellen verschiedener Dimensionen farblich hervorhebt.

Evaluierung

EXPLoRE wurde in einer Mehrfachfallstudie eingesetzt, die das Vorgehen und die Auswirkungen von 184 Änderungen beobachtet. Die Fallstudie wurde in zwei verschiedenen Projekten, einem Multimedia Mobiltelefon und einem Infotainment System, durchgeführt und zeigt, dass der Freigabemanager im Fall des Mobiltelefons bei einem Drittel und im Fall des Infotainmemt Systems bei der Hälfte der Änderungen komplexe Abhängigkeiten analysierte. In beiden Projekten setzten die Teilnehmer Freigabeszenarios zur Erkundung alternativer Lösungsvorschläge ein und passten bei etwa einem Drittel der Änderungen die Freigabe nach einer Änderung an. Aus dem hohen Anteil komplexer Abhängigkeiten und dem aktiven Einsatz von Freigabeszenarios kommt diese Dissertation zu dem Entschluss, dass BEEF die Herausforderungen im Freigabemanagement reduziert. Dies stimmt auch mit dem Ergebnis einer Befragung überein, in der 80% der Teilnehmer angaben, dass die getroffenen Entscheidungen gut bis sehr gut waren.

1.2 Gliederung der Arbeit

Änderungen

Diese Arbeit ist wie folgt gegliedert: Kapitel 2 stellt eine Taxonomie für Änderungen auf und diskutiert, wie Vorgehensmodelle, Buildmanagement, die Systemmodellierung, Begründungsmodelle sowie das Freigabemanagement Änderungen umgehen. Vorgehensmodelle haben sich von starren festen Prozessbeschreibungen zu agilen Prozessen entwickelt, um Änderungen zu erlauben. Buildmanagement automatisiert die Integration unabhängig voneinander entwickelter Komponenten, um bei Änderungen im Quellcode schnell und einfach eine lauffähige Version eines Software-Systems herstellen zu können. Die Systemmodellierung unterstützt die Entwicklung und Kommunikation über ein Software-System und ist daher in hohem Maße von Änderungen betroffen. Begründungsmodelle enthalten Begründungen für Entscheidungen und stellen bei Änderungen Wissen bereit, weshalb eine bestimmte Entscheidung getroffen wurde.

Freigabe-modell

Kapitel 3 stellt ein Modell für Freigaben vor, das an das Konfigurationsmanagement anknüpft, indem es Änderungen über die Zeit in Versionen festhält. Neu ist die Darstellung einer Roadmap als Folge von Knowledge Nuggets, welche die explizite Modellierung jeder einzelnen Freigabe erlauben. Knowledge Nuggets sind ein mächtiges Hilfsmittel, um bereits gelieferte Freigaben zu dokumentieren, aktuelle und zukünftige Freigaben zu planen, zu ändern und zu entwickeln: Knowledge Nuggets sorgen für konsistente Freigabepläne, indem sie Freigabepläne aus den Beschreibungen in der Roadmap aufeinanderfolgender Freigaben generieren und diese Freigabepläne automatisch bei Änderungen aktualisieren. Hier ist die Berücksichtigung der Nachvollziehbarkeit bei Änderungen an Freigaben besonders hervorzuheben, denn das Freigabemodell garantiert, bei der Änderung eines Artefakts auch abhängige Modelle an die Änderung anzupassen. Wird beispielsweise eine Anforderung auf eine spätere Freigabe verschoben, betrifft dies auch abhängige Klassen aus den Objektmodellen.

Begründungs basiertes Freigabe-modell

Kapitel 4 fügt dem Freigabemodell aus Kapitel 3 Begründungen hinzu, um zwischen alternativen Freigabeszenarios abzuwägen. Als Begründungen dienen der Feststellung von Carlshamre [Carlshamre 2002] folgend für jede Freigabe frei definierbare Kriterien, die entweder von Interessenvertretern

bewertet oder vom begründungsbasierten Freigabemodell berechnet werden. Neben diesen Kriterien kommen Begründungen aus der Kommunikation, wie Kommentare, Fehlerberichte, Fragestellungen oder Terminaufgaben. Das begründungsbasierte Freigabemodell berücksichtigt Aspekte der Organisation, wie Mitarbeiter, Zugehörigkeiten zu Projekten und ihre Fähigkeiten und Verfügbarkeiten, als Begründung. Zudem bilden Systemmodelle verschiedenen Abstraktionsgrads, beispielsweise der Analyse, des Systementwurfs oder Objektentwurfs sowie deren Abhängigkeiten eine fundierte Grundlage für Entscheidungen.

Knowledge Nugget Modell

Kapitel 5 beschreibt das neue Konzept der Knowledge Nuggets. Knowledge Nuggets bilden einen zentralen Bestandteil des begründungsbasierten Freigabemodells, weil sie die System-, Kommunikations- und Organisationsmodelle einer Freigabe oder Alternative kennen und verwalten. Für jede Freigabe können mehrere Knowledge Nuggets eingesetzt werden, um die Freigabe beliebig detailliert zu modellieren, beispielsweise kann ein Knowledge Nugget das Analysemodell einer Freigabe und ein weiterer Knowledge Nugget das Systementwurfsmodell der Freigabe beschreiben. Knowledge Nuggets propagieren Änderungen an andere Knowledge Nuggets, das heißt die Änderung des Modells eines Knowledge Nuggets führt automatisch auch zu einer Änderung des Modells anderer Knowledge Nuggets und sorgt somit auch für Konsistenz in den Modellen aufeinanderfolgender Freigaben.

Visualisierung

Kapitel 6 zeigt, dass das begründungsbasierte Freigabemodell einen mehrdimensionalen Entscheidungsraum aufstellt, den der Freigabemanager bei der Entscheidungsfindung durchsuchen kann. Vor dem Hintergrund existierender Visualisierungen des Projekt- und Buildmanagements wird eine neuartige dreidimensionale Visualisierung des Entscheidungsraums vorgestellt, die eine Navigation durch Begründungen, Roadmaps und alternative Freigabeszenarios erlaubt. Die Entscheidungsfindung zwischen zwei Freigabeszenarios wird durch eine direkte Vergleichsansicht unterstützt, indem sie Unterschiede zwischen ausgewählten Freigabeszenarios oder Freigaben hervorhebt. Ein wichtiger Teil des Freigabemodells ist der Nachvollziehbarkeits-Graph, mit dem der Freigabemanager die Abhängigkeiten der erfassten Sy-

stem-, Kommunikations- und Organisationsmodelle aufeinanderfolgender Freigaben durchlaufen kann .

EXPLoRE

Kapitel 7 beschreibt die prototypische Implementierung des begründungsbasierten Freigabemodells in dem neuen Werkzeug EXPLoRE. EXPLoRE folgt einer Schichtenarchitektur mit 4 Schichten. Die Visualisierungsschicht stellt die in Kapitel 6 beschriebenen Visualisierungen als Benutzerschnittstelle bereit. Sie wird unterstützt von einer Modelldiensteschicht, welche die Daten aus dem begründungsbasierten Freigabemodell für die Visualisierung aufbereitet. Das begründungsbasierte Freigabemodell ist in der Begründungsschicht abgebildet. Die vierte Schicht ist die Elementspeicherschicht, welche die Speicherung der Modelle aus der Begründungsschicht übernimmt.

Evaluierung

In Kapitel 8 wird das Konzept des begründungsbasierten Freigabemanagements empirisch evaluiert. In einer Mehrfachfallstudie wurden die Auswirkungen von 184 Änderungen in zwei Projekten beobachtet und nach ihrer Ursache unterschieden. Die Fallstudie stellt fest, dass bei über 43 % aller Änderungen komplexe Abhängigkeiten untersucht wurden und dass BEEF durch die explizite Verfügbarkeit dieser Modelle die Qualität der Entscheidung deutlich verbessert, da diese herkömmliche Werkzeuge nicht ausreichend abbilden. In der Fallstudie bewährte sich der Einsatz von Freigabeszenarios, insbesondere wenn eine Entscheidung zwischen verschiedenen Lösungsalternativen aus dem Systementwurf oder der Implementierung zu treffen war. Als Konsequenz der Analysen führten über 33% der Änderungen zu einer Anpassung der Freigabe durch Freigabeoperationen. Die Befragung der Teilnehmer ergab, dass die Analyse der Abhängigkeiten, Freigabeszenarios und Freigabeoperationen das Freigabemanagement unterstützen und somit die Herausforderungen im Freigabemanagement reduzieren.

Zusammenfassung und Ausblick

Kapitel 9 fasst die Ergebnisse dieser Dissertation zusammen und gibt einen Ausblick auf künftige Forschungsarbeiten und Anwendungsbereiche.

2 Änderungen

Änderungen

Änderungen sind Ereignisse, die nicht immer vorhergesehen und vorweggenommen werden können und somit verhindern, ein Projekt nach einem vorgegebenen Plan ausführen zu können. Ursachen für Änderungen können in der Anforderungsanalyse, dem System- oder Objektentwurf, der Implementierung liegen. Zudem gibt es Änderungen durch Fehler, in der Projektorganisation und –umgebung (siehe Abb. 1).

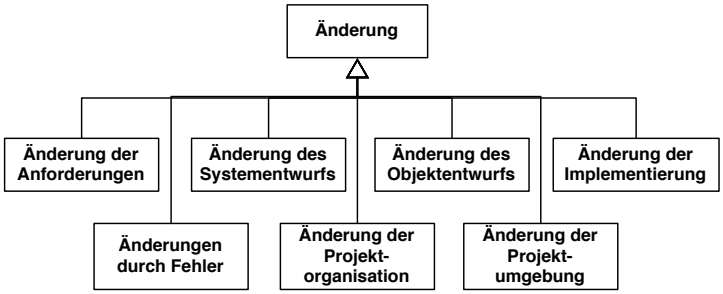


Abb. 1: Taxonomie für Änderungen, welche die Entwicklung eines Software-System beeinflussen. Die Taxonomoie folgt den Ursachen für Änderungen: Änderungen können in den Anforderungen, dem Systementwurf, dem Objektentwurf oder der Implementierung liegen oder durch Fehler verursacht worden sein. Zudem können sich die Projektorganisation und Projektumgebung ändern.

Änderungen in den Anforderungen

Anforderungen ändern sich, wenn Analysten im Projektverlauf mit der Anwendungsdomäne immer vertrauter werden, sie diese zunehmend besser verstehen und somit Fehler, Unvollständigkeiten oder Inkonsistenzen in den Anforderungen aufdecken und verbessern. In ähnlicher Weise bekommen Kunden während der Entwicklung oder dem Einsatz des Systems eine immer genauere Vorstellung von dem unter Entwicklung stehenden System,

aus der geänderte Wünsche und Anforderungen über das zu entwickelnde System resultieren. [Bruegge et al. 2005] Änderungen in den Anforderungen sind zu berücksichtigen, damit Freigaben die Bedürfnisse der Kunden erfüllen und damit verbunden Geschäftsziele erreicht werden. [Bagnall et al. 2001] Hier haben *technology enabler* eine besondere Rolle. *Technology enabler* sind Technologien, die neue Systeme erlauben. Sie schnell zu unterstützen ist oft ein Schlüssel zum Erfolg und deshalb gehören sie oft zu den Anforderungen des Kunden. Mit neuen Technologien bestehen jedoch dann meist noch keine Erfahrungen, so dass ihre Verwendung zu häufigen Änderungen führen kann.

Änderungen im Systementwurf

Änderungen des Systementwurfs treten ein, wenn die Subsystem-Dekomposition angepasst wird. Architekten können Subsysteme beispielsweise verschmelzen, um die Kopplung zwischen Subsystemen zu erniedrigen. Ebenso können Subsysteme weiter aufgeteilt werden, um sie auf kleinere Teams zu verteilen und den Zusammenhalt im Systementwurf zu erhöhen. Ein weiteres Beispiel für eine Änderung aus dem Systementwurf sind Änderungen an der Hardware/Software, beispielsweise wenn eine Anwendung von einer Arbeitsstation auf ein mobiles Endgerät portiert werden soll.

Änderungen im Objektentwurf

Änderungen im Objektentwurf treten auf, wenn die Schnittstellen von Klassen nicht klar spezifiziert sind oder angepasst werden und als Folge davon nicht zusammenpassen. Ursache sind zum einen Änderungen an den Signaturen öffentlicher Methoden, beispielsweise wenn neue Anforderungen in das Modell eingearbeitet werden. Zum anderen können externe Komponenten, Klassenbibliotheken oder Frameworks Schnittstellenänderungen hervorrufen, denn auch diese entwickeln sich häufig parallel zum eigenen System weiter. Bei Schnittstellenänderungen in externen Komponenten, Klassenbibliotheken oder Frameworks kommt erschwerend hinzu, dass Entwickler diese Änderungen meist nicht beeinflussen können. Sind sie auf die neue Version angewiesen, weil sie beispielsweise Fehlerverbesserungen bringt, bleibt nur die Anpassung des eigenen Objektentwurfs an die geänderten Schnittstellen.

Die Implementierung ruft weitere Fragestellungen hervor, die zum Zeitpunkt der Analyse oder des Entwurfs nicht vorhersehbar waren. Sieht der

Änderung in der Implementierung

Entwurf beispielsweise nur die Verwendung eines Frameworks vor, müssen Entwickler mit dem Framework in der Implementierung konkret arbeiten. Ist dieses dann beispielsweise schlecht dokumentiert, müssen sich die Entwickler langsam mit dem Framework vertraut machen, indem sie mit kleinen Programmen die Funktionsweise ausprobieren oder den Quellcode des Frameworks studieren.

Änderungen durch Fehlerberichte

Tests dienen dazu Software-Systeme zu falsifizieren; Entwickler und Kunden testen Software-Systeme, um Fehler zu finden. [Bruegge & Dutoit 2003] Fehler sind algorithmische Ursachen für ein Abweichen des beobachteten Systemverhaltens von der Spezifikation des Systems. Die Fehlersuche hat das Ziel, die Ursache eines Fehlers im System zu lokalisieren und kann sich über mehrere Methoden, Klassen und Komponenten erstrecken. Die Fehlersuche führt zur Fehlerbehebung, das heißt einer Änderung des Systems mit dem Ziel, dass der Fehler nicht mehr auftritt. Die Fehlerbehebung dient dazu, dass das System die Anforderungen trifft und die Kunden mit dem System zufrieden sind.

Änderungen in der Organisation

Bei Änderungen in der Organisation können Mitarbeiter das Projekt verlassen, beispielsweise durch Kündigung oder Krankheit. Fällt ein kritischer Mitarbeiter aus, kann dies den Erfolg eines gesamten Projekts gefährden, wenn nur er Wissen über zentrale Komponenten des Systems hat. Denn neue, aber auch vorhandene Mitarbeiter müssen sich zunächst in die Komponenten einarbeiten, für die der ausgefallene Mitarbeiter verantwortlich war. Neben persönlichen Gründen führen Umstrukturierungen im Unternehmen dazu, dass Mitarbeiter Verantwortlichkeiten, Abteilungen, Teams und Projekte wechseln. Die Zusammenlegung zweier Teams ist ebenfalls eine nennenswerte Änderung, denn die neuen Teammitglieder müssen sich und ihre Gewohnheiten erst kennenlernen und mit den geänderten Verantwortlichkeiten umgehen.

Änderungen in der Projektumgebung

Änderungen in der Projektumgebung beinhalten die Einführung eines neuen Werkzeugs und den Wechsel von Werkzeugen. Ursachen hierfür sind, dass neue Werkzeuge die Herausforderungen des Projekts besser unterstützen können. Soll die neue Version eines Systems auch für tragbare Geräte entwickelt werden, kann ein Wechsel der Entwicklungsplattform nötig

sein. Dies führt dazu, dass betroffene Mitarbeiter lernen müssen, die Plattform zu bedienen, was zu Verzögerungen im Projekt führen kann.

Die nächsten Abschnitte diskutieren, wie Vorgehensmodelle, Integrationsstrategien, Systemmodellierung und Begründungsmodelle Änderungen berücksichtigen und zeigen wie, ein Freigabemanagement Änderungen berücksichtigen kann.

2.1 Berücksichtigung von Änderungen in Vorgehensmodellen

Code & fix

Änderungen treten immer wieder, in unregelmäßigen Abständen während eines Softwareprojekts ein. Vorgehensmodelle definieren, wann ein Softwareprojekt Änderungen berücksichtigt. In den 1960er Jahren gingen Softwareprojekte nach dem Prinizip *„code and fix“* vor. Dabei wurde ein Stück Quellcode erstellt, anschließend Änderungen eingearbeitet, beispielsweise Fehler behoben bis der Code einigermaßen funktionierte. Das stetige Berücksichtigen von Änderungen führte dazu, dass der produzierte Quellcode durch die Änderungen schlecht strukturiert und kaum mehr wartbar war. Als Konsequenz wurde das Einarbeiten von Änderungen im Laufe eines Softwareprojekts immer aufwändiger und fehlerintensiver.

Softwarekrise

Als Folge scheiterten zahlreiche Softwareprojekte, weil Software-Systeme nicht oder nur mit geringer Qualität zum Liefertermin fertig gestellt waren. Die Probleme der Softwareentwicklung waren derart gravierend, dass man von einer Softwarekrise spricht. Um die Probleme zu lösen, hat die NATO Software Engineering Konferenz in Garmisch, 1968 Ursachen für die Softwarekrise gesucht: Als Ergebnis wurde festgehalten, dass Probleme vor allem im Management der Programmierung bestehen. Die bisher übliche ständige Einarbeitung von Änderungen in Software-Systeme sollte nun dem Vorbild der Ingenieursdisziplinen folgen. [Naur & Randell 1969]. Ziel war die Entwicklung einer allgemeingültigen Vorgehensweise für die Entwicklung von Software-Systemen. Vorbild für diese Vorgehensweise war die industrielle Fertigung am Fließband, bei der Produkte automatisiert gefertigt werden, indem zuvor festgelegte Schritte durchlaufen werden. Dementspre-

chend sollte die Vorgehensweise allgemeingültig und wiederholbar für alle Arten von Softwareprojekten Schritte festlegen, durch die Software-Systeme erfolgreich erstellt werden können.

Wasserfall-modell

Ein Versuch einer solchen Vorgehensweise war das Wasserfallmodell von Royce [Royce 1970]. Das Wasserfallmodell teilt die Softwareentwicklung in die Entwicklungsphasen Anforderungsermittlung, Analyse, Entwurf, Implementierung, Testen und Wartung auf. Die Phasen werden sequentiell, das heißt eine nach der anderen durchlaufen, wobei man zur nächsten Phase nur dann weitergehen darf, wenn zuvor definierte Abnahmekriterien für die vorhergehende Phase erfüllt sind (siehe Abb. 2).

Änderungen des Kunden

Der Projektmanager legt im Wasserfallmodell zu Projektbeginn mit dem Kunden Anforderungen, Umfang und Lieferzeitpunkte fest. Sie bilden die Grundlage für die Planung des Projekts; die Entwicklung selbst ist für den Kunden eine *black box*. Er bekommt das entwickelte System zum ersten Mal bei der Lieferung in einem Abschlusstreffen präsentiert. Änderungen in den Anforderungen kann der Kunde während der Projektlaufzeit nicht einbringen.

Wartungsphase für Änderungen

Entspricht das gelieferte System dann nicht den Vorstellungen des Kunden oder ist es gar nicht einsatzfähig, muss das System in einer lang dauernden Wartungsphase angepasst werden. Ebenso dient diese Wartungsphase dazu fehlende Anforderungen zu implementieren oder Fehler zu beheben, wenn das System bis zum vereinbarten Liefertermin nicht fertig gestellt werden kann.

Änderungen während der Entwicklung

Um Änderungen, die in der Analyse, dem Systementwurf, dem Objektentwurf, der Implementierung oder dem Testen des Systems begründet sind, zu berücksichtigen, sieht das Wasserfallmodell zwei Mechanismen vor: Zum einen kann im Wasserfallmodell um Phasen zurückgegangen werden, zum anderen führt es das „Do it twice“-Prinzip [Royce 1970] ein.

Kurskorrek-turen

Durch ein Zurückkehren zu früheren Phasen erlaubt das Wasserfallmodell kleinere Kurskorrekturen. Wird beispielsweise ein Problem in der Implementierung festgestellt, kann nochmals zum Systementwurf zurückgegangen werden. Jedoch geht das Wasserfallmodell dann davon aus, dass die Phasen, um die zurückgegangen worden ist, nochmals wiederholt werden

müssen. Änderungen, beispielsweise im Testen, deren Lösung verlangt, bis zur Analyse zurückzugehen, stellen ein Problem dar, weil dann eine größere Überarbeitung des Entwurfs nötig ist.

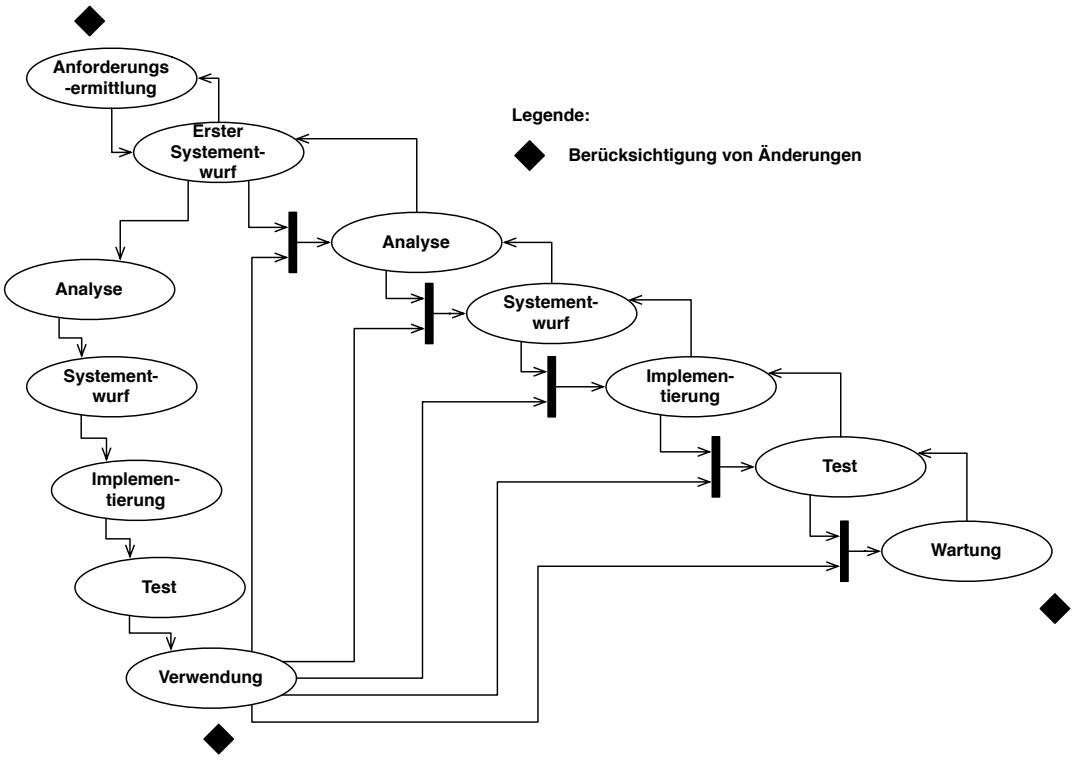


Abb. 2: Berücksichtigung von Änderungen im Wasserfallmodell: Änderungen werden nur am Anfang und Ende eines Projekts berücksichtigt. Um Änderungen zu vermeiden wird ein erster Systementwurf nach der Anforderungsermittlung eingeführt. Das „Do it twice"[Royce 1970] - Prinzip sieht einen ersten sequentiellen Durchlauf durch alle Phasen nach dem ersten Systementwurf vor, deren Ergebnisse Änderungen in der Systementwicklung vermeiden sollen. Für dennoch nötige Kurskorrekturen erlaubt das Wasserfallmodell zu der vorangegangenen Phase zurückzukehren.

„Mach es Zweimal" - Prinzip

Das „Do it twice"-Prinzip [Royce 1970] kann man als ein frühes Prinzip des *Prototyping* ansehen. Es beinhaltet zwei Durchläufe des Wasserfallmodells von der Analyse bis zur Wartung. Der erste Durchlauf dauert ca. 1/3 der Projektlaufzeit und dient der Machbarkeitsüberprüfung der als kritisch eingeschätzten Systemteile. Die gewonnenen Ergebnisse des ersten Durchlaufs sollen Änderungen im zweiten Durchlauf vermeiden.

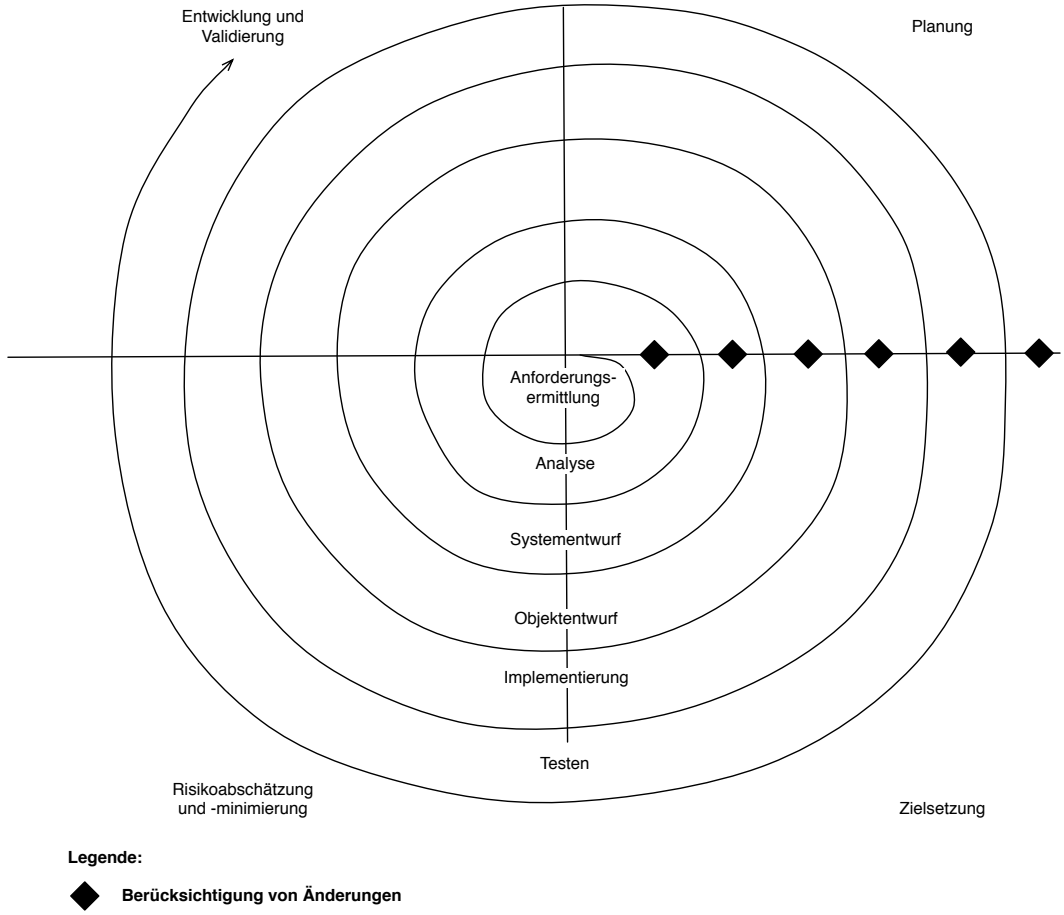


Abb. 3: Das Spiralmodell: Jede Runde ist eine Phase des Wasserfallmodells und besteht aus Zielsetzung, Risikoabschätzung und -minimierung, -entwicklung und -validierung sowie Planung. Änderungen berücksichtigt das Spiralmodell nach jeder Phase.

Spiralmodell

Boehm [Boehm 1988] erstrebte mit dem Spiralmodell (siehe Abb. 3) mit Änderungen in einem Softwareprojekt zurechtzukommen. Das Spiralmodell besteht aus mehreren Runden, wobei jede Runde eine Phase des Wasserfallmodells adressiert; beispielsweise ist die Anforderungsanalyse eine Runde. Jede Runde besteht aus einer Zielsetzung, Risikoabschätzung und Risikominimierung, Entwicklung und Validierung sowie Planung. Während der Zielsetzung wird ein detaillierter Managementplan erstellt, in dem auch Risiken, das heißt potentielle Probleme, für das Projekt identifiziert werden. Die Risikoabschätzung und -minimierung betrachtet alle identifizierten Risiken und plant, wie diese vermieden werden können. Hier schlägt Boehm

die Verwendung eines Prototyps vor. Dieser Prototyp dient beispielsweise in der innersten Runde zur Überprüfung der Machbarkeit, in der nächsten Runde, also der Anforderungsanalyse, der Überprüfung der Anforderungen, im Systementwurf dem Vergleich verschiedener Entwurfsmöglichkeiten.

Änderungen im Spiralmodell

Wie das Wasserfallmodell sieht auch das Spiralmodell vor, den Umfang des Systems zu Projektbeginn in einem Treffen mit dem Kunden festzulegen und am Ende zu liefern. Die Prototypen in jeder Runde dienen dazu, Risiken zu erkunden und damit möglicherweise auftretende Änderungen vorwegzunehmen und zu vermeiden. Am Ende jeder Runde erfolgt ein Treffen für ein *Review* des Projekts, das erlaubt Änderungen am Projektplan vorzunehmen und nochmals zu entscheiden, ob mit der nächsten Runde fortgefahren werden soll.

Lineare Vorgehensmodelle

Im Wasserfallmodell und Spiralmodell folgen die Phasen der Softwareentwicklung sequentiell nacheinander; daher sind sie Beispiele für lineare Vorgehensmodelle. In linearen Vorgehensmodellen ist eine Roadmap der strategische Plan für die nächsten Jahre und legt Lieferzeitpunkte und Umfang, Anforderungen und Technologien der nächsten Freigaben fest. In linearen Vorgehensmodellen können einmal getroffene Entscheidungen nicht revidiert werden. Beispielsweise sehen sie keine Änderungswünsche des Kunden nach abgeschlossener Anforderungsanalyse vor. Doch Änderungswünsche des Kunden beeinflussen heute die Systementwicklung: Anforderungen unterliegen während der Entwicklung Änderungen, die weder vorhersehbar noch planbar sind und stellen somit eine Unbekannte in der Softwareentwicklung dar [Bruegge & Dutoit 2003], [Schwaber & Beedle 2002].

Agile Vorgehensmodelle

Agile Methoden berücksichtigen häufige Änderungen auch während einer Entwicklungsphase, indem sie das Produkt iterativ entwickeln und inkrementell um neue Anforderungen erweitern. Dabei erfolgt die Entwicklung einer Anforderung vertikal, das heißt komplett durch alle Schichten, so dass sie sofort ausführbar ist. Dies erlaubt es in kurzen, etwa monatlichen, Zyklen Freigaben des Systems an den Kunden zu liefern und neues Feedback zu erhalten. Diese Folge von Änderungen plant die Roadmap, allerdings ist sie im Gegensatz zu linearen Vorgehensmodellen dynamisch, das heißt sie wird

häufig an Änderungen, die sich durch die Lieferung der vorangegangenen Freigabe ergeben, angepasst. [Larman 2004]

Scrum

SCRUM [Schwaber & Beedle 2002] verwendet Sprints, das sind 15 bis 30 Tage lang dauernde Iterationen. Zu Beginn eines Sprints findet ein Sprintplanungstreffen statt. In diesem Treffen wählen Kunde und Management zusammen mit dem Entwicklungsteam Anforderungen aus, die in diesem Sprint umgesetzt werden sollen. Um auf unerwartete Änderungen reagieren zu können, findet ein tägliches SCRUM Treffen statt. Dieses ist ein etwa 15 Minuten dauernder Stehkreis, in dem jeder Entwickler kurz über seine Arbeit, Probleme und nächsten Ziele berichtet. Das Ergebnis eines Sprints ist ein *Potentially Shippable Product Increment*, eine Freigabe die das Produkt um die neuen Anforderungen erweitert.

Extreme Programming

Extreme Programming [Beck 2001] verwendet zur Planung Anforderung, sogenannte *user stories*, die informell auf Karten geschrieben werden. In einem Planungsspiel legen Kunde und Entwickler fest, welche *user stories* die nächste Freigabe implementieren soll. Man verzichtet auf eine vollständige Analyse aller Anforderungen zu Beginn, sondern startet mit einer kleinen Menge *user stories*, die dann schrittweise erweitert wird. Dadurch stellt jede neue *user story* eine Änderung in den Anforderungen dar. Entwurf und Code bleiben schlank und einfach, um schnell auf Änderungen reagieren zu können. Freigaben mit den Änderungen werden alle ein bis vier Wochen er

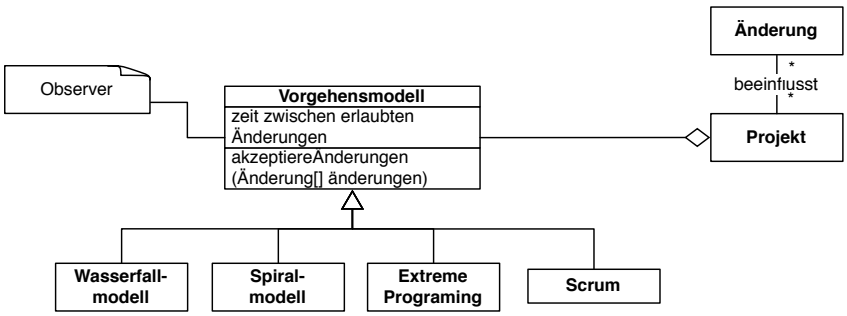


Abb. 4: Vorgehensmodelle beobachten die Änderungen eines Projekts. Jedes Vorgehensmodell erlaubt Änderungen zu bestimmten Abständen, die durch die Zeit zwischen erlaubten Änderungen festgelegt ist. Ist diese Zeit abgelaufen, akzeptiert das Vorgehensmodell die bis dahin aufgetretenen Änderungen.

stellt. Um Probleme durch die Änderungen früh zu erkennen und stets eine mögliche Freigabe verfügbar zu haben, werden Änderungen regelmäßig integriert, kompiliert, getestet.

Vorgehensmodell als Beobachter

Vorgehensmodelle legen fest, wie mit Änderungen umgegangen werden soll. Abb. 4 modelliert daher ein Vorgehensmodell als Beobachter-Entwurfsmuster, indem es Vorgehensmodelle als Beobachter für die Änderungen eines Projekts sieht. Dabei legen Vorgehensmodelle Zeitspannen fest, zu denen sie alle bis dahin eingetretenen Änderungen eines Projekts akzeptieren. Die Wahl dieser Zeitspanne ist eine Abwägung zwischen Flexibilität in der Berücksichtigung von Änderungen und einem Abschotten der Entwicklung vor Änderungen. Zu häufige Änderungen können zu einem Flattereffekt führen, wenn sich die Entwicklungsarbeit nur noch auf das Einarbeiten von womöglich widersprüchlichen Änderungen konzentriert. Dann führen die Änderungen dazu, dass die Ziele eines Projekts unklar sind und gefährden den Erfolg eines Projekts.

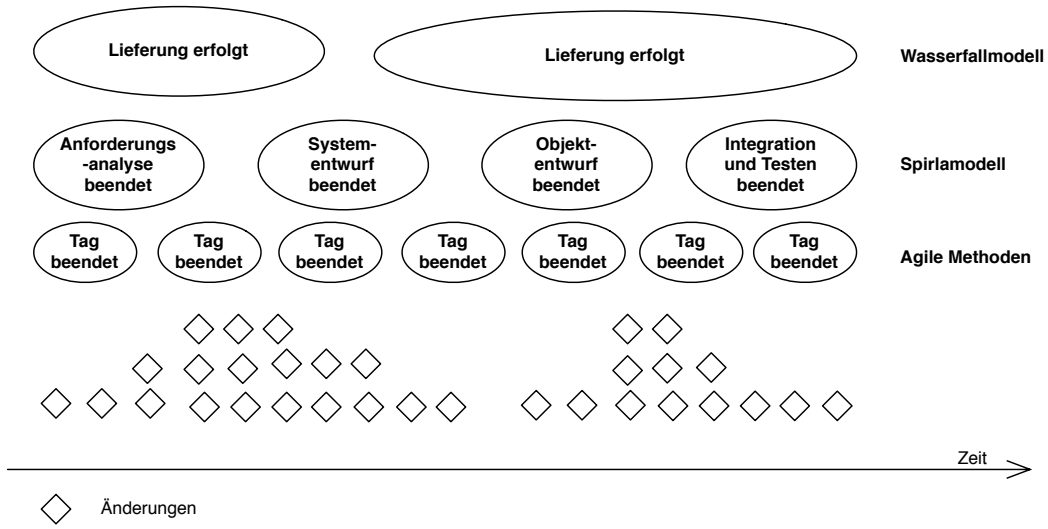


Abb. 5: Änderungen in verschiedenen Vorgehensmodellen: Das Wasserfallmodell berücksichtigt Änderungen erst nach der Lieferung des Systems, also zweimal in einem Projekt bei Verwendung des „Do it twice“[Royce 1970]–Prinzips. Das Spiralmodell lässt Änderungen jeweils zwischen den Entwicklungsphasen zu. Agile Methoden erlauben Änderungen auch während der Entwicklungsphasen zu Beginn eines jeden Tages.

Abb. 5 visualisiert die erlaubte Zeit zwischen Änderungen in den oben beschriebenen Vorgehensmodellen. Unten symbolisieren die Rauten Änderun-

gen, die über die Zeit in einem Projekt auftreten, wobei die Verteilung der Änderungen zufällig gewählt ist. Das Wasserfallmodell akzeptiert nach jeder Lieferung des Systems Änderungen, wobei es dann mit einer ganzen Menge bis dahin abgefallener Änderungen überhäuft wird. Das Spiralmodell unterscheidet sich im Umgang mit Änderungen nur geringfügig, es akzeptiert Änderungen zwar häufiger, nämlich nach jeder Entwicklungsphase, muss dann aber immer noch mit vielen Änderungen zum gleichen Zeitpunkt umgehen können. Agile Methoden überprüfen täglich, ob Änderungen aufgetreten sind und können dadurch flexibel und schnell auf die typischerweise kleine Zahl von eingetretenen Änderungen reagieren.

Erlaubte Zeit zwischen Änderungen

2.2 Berücksichtigung von Änderungen im Buildmanagement

Buildmanagement synchronisiert die Änderungen verschiedener Teams, indem es aus dem Quellcode unabhängig entwickelter Subsysteme ein ausführbares Gesamtsystem (*build*) zusammenbaut und dieses testet. Buildmanagement kann feststellen, ob die Änderungen seit dem letzten *build* einen Fehler im System eingeführt haben.

Build-management

Früher wurden ausführbare Systeme manuell durch Programmierer erstellt, sobald eine signifikante Änderung des Quellcodes vorlag. Da die Erstellung eines ausführbaren Systems immer wieder die Ausführung derselben Kommandos erfordert, führte Feldmann 1977 das *make file* Konzept ein. [Gold 2004], [Bays 1999]. Das *make file* Konzept kann die Aktivitäten der Erstellung eines ausführbaren Systems für C-Programme automatisiert durchführen. Ähnlich eines Skripts definiert es dazu eine Reihe von Kommandos für die Erstellung des Systems, die dann durch einen Befehl, beispielsweise *make*, aufgerufen werden können.

make file

Ant [Holzner 2005] und Maven [Sonatype 2008] sind Weiterentwicklungen des *make file* Konzepts und erstellen automatisch für Java Programme [Gosling et al. 2000] ausführbare Systeme. Ant und Maven Skripten basieren auf XML [Ray 2003] und sind daher übersichtlicher und einfacher zu handha-

Ant und maven

ben als *make files*. Skripten für Ant und Maven sind portabel und können auf verschiedenen Plattformen eingesetzt werden.

Big bang Integration

In linearen Vorgehensmodellen ist es eine Prämisse, dass alle Subsysteme zu Beginn der Integrations- und Testphase fertig gestellt sind. Auch dann sollte die Integration nicht als *big bang* durchgeführt werden, das heißt alle Subsysteme auf einmal zu einem Gesamtsystem zusammengesetzt und getestet werden. Denn Ursachen für auftretende Fehler können bei einer *big bang* Integration im gesamten System liegen und sind daher schwer zu lokalisieren. Daher ist eine *big bang* Integration ein Zeichen für schlechtes Buildmanagement.

Inkrementelle Integration

Diese Probleme adressiert die inkrementelle Integration, indem sie das erfolgreich getestete System in jedem Integrationsschritt um ein neues Subsystem erweitert. Dabei simulieren *Stubs* und *Driver* noch nicht integrierte Subsysteme. Fehler können dann nur in dem neu hinzugekommen Subsystem oder der Interaktion mit diesem liegen. Ein Beispiel für die inkrementelle Integration ist die horizontale Integration; sie folgt den Schichten der Software-Architektur: Die *top-down* Integration beginnt mit den Subsystemen der obersten Architekturschichten und nimmt jeweils ein Subsystem einer tieferen Schicht hinzu. Die *bottom-up* Integration beginnt mit den untersten Schichten und erweitert das System, indem sie jeweils ein Subsystem der nächst höheren Architekturschicht hinzunimmt. [Mynatt 1990] [Bruegge & Dutoit 2003]

Vertikale Integration

Bei der vertikalen Integration kann jede neue Anforderung zu Änderungen in allen Schichten der Software-Architektur führen. Nach Fertigstellung einer Anforderung integrieren Entwickler die Änderungen des Quellcodes mit dem bestehenden System, kompilieren und testen dieses. Um Fehlerursachen gering zu halten, müssen Entwickler ihre Änderungen bei vertikaler Integration früh integrieren. Ein Beispiel früher Integration ist die Integrationsstrategie *test first* von *eXtreme Programming*, das propagiert, den Testfall für eine *user story* vor ihrer Implementierung im System zu entwickeln. [Beck 2001]

Reguläre *builds* erstellen zu definierten Zeitpunkten ein ausführbares System und testen dieses. Die Dauer, ein ausführbares System zu erstellen, be-

schränkt die Anzahl regulärer *builds*, die erstellt werden können; früher dauerte das Erstellen eines ausführbaren Systems mehrere Stunden, weshalb lauffähige Systeme nur selten erstellt werden konnten. Beispielsweise erstellte Microsoft jede Nacht einen *nightly build*, um die Änderungen des letzten Arbeitstages zu integrieren und zu testen. Fehler nach der Integration können in allen Änderungen liegen, die seit dem letzten erfolgreichen Build vorgenommen wurden. [Cusumano & Selby 1997]

Reguläre builds

Größere Speicher und schnellere Prozessoren erlauben nun häufiger ein ausführbares System zu erstellen und zu testen. So stößt die kontinuierliche Integration [Fowler 2006] nun nicht mehr zu einem festgelegten Zeitpunkt, sondern nach jeder Änderung des Quellcodes den Buildprozess an. Rufen die letzten Änderungen Probleme hervor, schlagen das Erstellen des Systems oder die Tests fehl. Durch dieses Vorgehen werden die Änderungen aller Anforderungen regelmäßig in den Quellcode integriert. Probleme kann man dadurch sofort nach dem Laden einer Änderung in das Konfigurationsmanagement feststellen. Um ein schnelles Feedback zu ermöglichen und Fehler schnell aufzuzeigen, sollte die Dauer für die Erstellung eines *builds* 10 Minuten nicht überschreiten. [Duvall et al. 2007] Bei großen, komplexen Systemen oder vielen Entwicklern skaliert die kontinuierliche Integration nicht. Denn dann dauert die Erstellung eines Builds zum einen länger, weshalb Entwickler zu lange auf das Feedback warten müssen und Fehler dann nicht schnell genug korrigieren können. Zum anderen werden Builds dann nur noch zu diskreten Zeitpunkten erstellt, es handelt sich also um reguläre Builds; dementsprechend werden mehrere Änderungen zusammen in einem einzelnen neuen Build integriert und getestet. Die Zeit für die Erstellung eines Builds kann reduziert werden, indem die Anzahl der durchzuführenden Tests, also Unittests, Integrationstests oder Systemtests, sowie die Anzahl durchgeführter Inspektionen verringert wird.

Kontinuierliche Integration

Dennoch sollte ein Build mindestens Unittests umfassen, wenn es die Zeit zur Erstellung des Builds erlaubt, auch noch Integrations- und Systemtests. Werden keine Fehler entdeckt, kann der build als interne Freigabe (*promotion*) verwendet werden, das heißt die Version steht direkt nach ihrer Erstellung anderen Entwicklern zur Verfügung. Sie ist zugleich ein Kandidat für die nächste externe Freigabe (*release*), also einer Version die an den

Interne und externe Freigaben

Kunden geliefert werden kann. [IEEE 1987] Kompiliert das System nicht oder werden in der lauffähigen Version beim Testen Fehler entdeckt, sollte das Konfigurationsmanagement die letzte in das Zentralverzeichnis geladene Änderung des Quellcodes rückgängig machen. Betroffene Entwickler können die aufgetretenen Fehler dann auf ihren Rechnern korrigieren und die korrigierten Änderungen dann zu einem späteren Zeitpunkt neu in das Konfigurationsmanagement laden. Dieses Vorgehen soll garantieren, dass der Quellcode im Konfigurationsmanagement stets lauffähig ist. [Fowler 2006]

2.3 Berücksichtigung von Änderungen in der Systemmodellierung

Systemmodelle

Modelle sind vereinfachte Abbilder der Realität. Systemmodelle sind Modelle, die ein Software-System beschreiben; Beispiele für Systemmodelle sind das Analysemodell, Systementwurfsmodell und ein detailliertes Entwurfsmodell. Das Analysemodell konzentriert sich auf Entitäten der realen Welt, die für ein System wichtig erscheinen. Ein detailliertes Entwurfsmodell ist eine vereinfachte Darstellung des Quellcodes, die von Details der Implementierung abstrahiert. Ein Systementwurfsmodell vereinfacht sowohl das Analysemodell als auch das detaillierte Entwurfsmodell und fokussiert, wie aus dem Analysemodell ein Software-System entwickelt werden kann. [Balzert 2001]

SA/SD

In der strukturierten Analyse und dem strukturieren Design (SA/SD) [Demarco 1979] verwendet jedes dieser Modelle eigene Formalismen und Syntax: Die strukturierte Analyse (SA) sieht zur Modellierung der Anforderungen Datenflussdiagramme vor, die schrittweise verfeinert werden. Dadurch existieren bereits im Analysemodell abstrakte Modelle sowie Modelle mit sehr detaillierten Informationen über das zu entwickelnde System. Das strukturierte Design (SD) nimmt das Analysemodell als Eingabe und erzeugt aus diesem durch Transformation oder Transaktionsanalyse eine Softwarestruktur. Sie ist ein komplett neues Modell mit anderer Notation, sog. *structure charts*. *Structure charts* enthalten eine Dekomposition des Systems in

Funktionen. Schließlich werden die Funktionen im detaillierten Entwurf durch Nassi-Shneiderman Diagramme mit einer dritten Notation neu beschrieben. [Mynatt 1990] Nassi-Shneidermann Diagramme beschreiben bereits konkrete Algorithmen. Konstrukte wie Bedingungen oder Schleifen stellen sie grafisch dar, wohingegen Anweisungen in Pseudo Code beschrieben werden.

Änderungen in SA/SD

Vertikale Systementwicklung [Bruegge & Dutoit 2003] führt zu regelmäßigen Änderungen im Analyse-, Systementwurfs- und detaillierten Entwurfsmodell. Da SA/SD davon ausgeht, dass zu Beginn des Systementwurfs ein vollständiges Analysemodell zur Verfügung steht, führen diese Änderungen in SA/SD dazu, dass das Systementwurfsmodell bei jeder Änderung neu übersetzt werden muss.

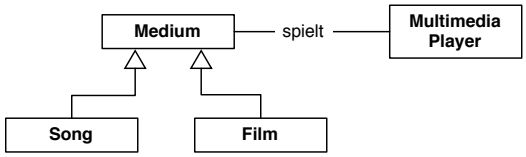


Abb. 6: Analyse-Objektmodell eines Multimedia Players: Der Multimedia Player spielt Medien ab. Als Medien unterstützt er Songs und Filme.

Analyse-Objektmodell

Im Gegensatz zu SA/SD verwendet die objektorientierte Softwareentwicklung mit UML [Fowler 2004] Objektdiagramme als durchgehende Notation zur Beschreibung der Modelle aus Analyse, Systementwurf und detailliertem Entwurf. Die objektorientierte Analyse identifiziert Objekte der Anwendungsdomäne und erstellt mit diesen ein Analyse-Objektmodell. Abb. 6 zeigt ein Analyse-Objektmodell für einen Multimedia Player, der Songs und Filme abspielen kann.

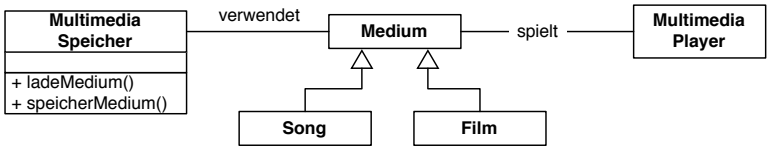


Abb. 7: Systementwurfs-Objektmodell eines Multimedia Players: Die Klasse Multimedia Speicher dient dazu, Medien zu speichern und laden.

Systementwurfs-Objektmodell

Dieses Analyse-Objektmodell verfeinert der Systementwurf um zusätzliche Objekte aus der Lösungsdomäne; das Ergebnis ist ein Systementwurfs-Objektmodell. Das Systementwurfs-Objektmodell des Multimedia Players in Abb. 7 erweitert das Analyse-Objektmodell aus Abb. 6 um einen Multimedia Speicher, einen zentralen Datenspeicher, um Multimedia Dateien abzulegen und zu laden.

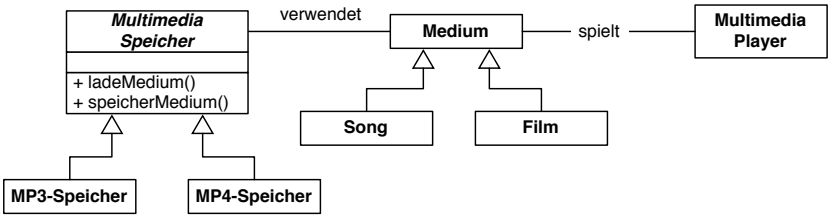


Abb. 8: Objektentwurfs-Objektmodell für einen Multimedia Player: MP3- Speicher und MP4- Speicher sind Strategien zur Speicherung von Multimedia Dateien.

Objektentwurfs-Objektmodell

Der detaillierte Entwurf, in der objektorientierten Modellierung Objektentwurfs-Objektmodell genannt, verfeinert das Systementwurfsmodell und schließt die Lücke zwischen Systementwurf und Implementierung. Dazu werden Klassenschnittstellen spezifiziert, die Wiederverwendbarkeit erhöht, und Optimierungen vorgenommen. [Bruegge & Dutoit 2003] Abb. 8 zeigt als Beispiel ein Objektentwurfs-Objektmodell für den Multimedia Player; es setzt das Strategiemuster für die Speicherung von Multimedia Dateien ein. MP3-Speicher und MP4-Speicher sind konkrete Multimediaspeicher; dabei komprimiert der MP3-Speicher die Medien in MP3, der MP4-Speicher in MP4.

Objektmodelle in der vertikalen Systementwicklung

Bei der vertikalen Systementwicklung enthält das Analysemodell eine erste, kleine Menge von Anforderungen. Während der Entwicklung dieser Anforderungen verwaist das Analyse-Objektmodell, denn aus ihm entwickelt sich zunächst ein Systementwurfs-Objektmodell und dann ein Objektentwurfs-Objektmodell. Als Konsequenz stehen bei der Entwicklung weiterer Anforderungen, beispielsweise für eine zweite Freigabe, das Analyse- und Systementwurfs-Objektmodell nicht mehr zur Verfügung.

Bedeutung der Objektmodelle

Analyse-, Systementwurfs- und Objektentwurfs-Objektmodell dienen der Kommunikation verschiedener Interessenvertreter. Das Analyse-Objektmodell beschränkt sich auf Entitäten, die der Kunde kennt und unterstützt dadurch das Verständnis von Kunde und Analysten. Das Systementwurfs-Objektmodell zeigt auf, wie ein System die Anforderungen des Kunden umsetzen kann. Es dient der Kommunikation zwischen Analysten und Systemarchitekten sowie den Architekten untereinander. Das Objektentwurfs-Objektmodell schließlich zeigt, wie das System implementiert ist und unterstützt damit das Verständnis von Architekten und Entwicklern sowie unter den Entwicklern selbst.

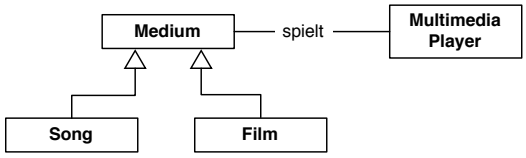


Abb. 9: Analyse-Objektmodell der ersten Freigabe des Multimedia Players: Sie unterstützt Songs und Filme.

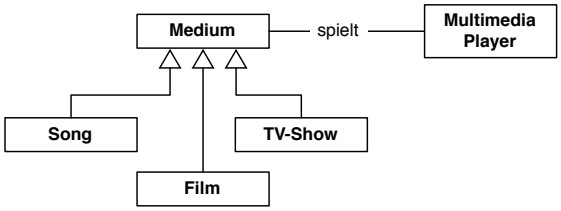


Abb. 10: Änderung des Analyse-Objektmodells des Multimedia Players aus Abb. 9 für die zweite Freigabe: TV-Show wird nun als neues Medium unterstützt.

Änderungen im Analyse - Objektmodell

Die Modellierungstechnik der inkrementellen Verfeinerung erlaubt auf Änderungen zu reagieren, indem Modelle inkrementell um neue Objekte ergänzt werden, die für die Umsetzung einer neuen Anforderung nötig sind. Abb. 10 zeigt als Beispiel die Verfeinerung des Analyse-Objektmodells des Multimedia Players aus Abb. 9: Die erste Freigabe kann nur Songs und Filme abspielen und enthält daher nur Song und Film als Medium; in einer zweiten Freigabe soll der Multimedia Player auch TV-Shows anzeigen. Folglich wird das Analyse-Objektmodell um die Klasse TV-Show erweitert.

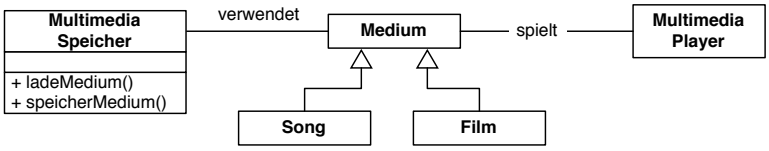


Abb. 11: Systementwurfs-Objektmodell der ersten Freigabe des Multimedia Players.

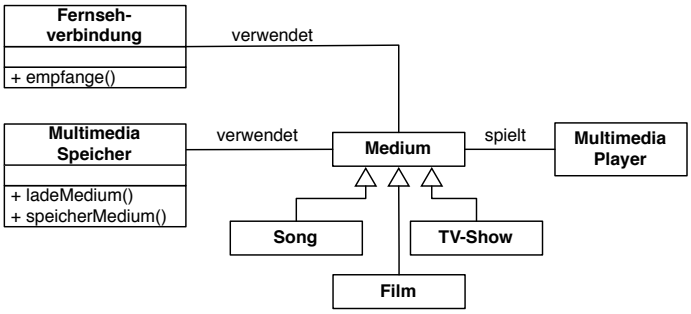


Abb. 12: Das Systementwurfs-Objektmodell der zweiten Freigabe des Multimedia Players ist eine Verfeinerung des Analyse-Objektmodells der zweiten Freigabe aus Abb. 10 und eine Verfeinerung des Systementwurfs-Objektmodells der ersten Freigabe aus Abb. 11: Um TV-Shows empfangen zu können, führt das Systementwurfs-Objektmodell zusätzlich eine Fernsehverbindung ein.

Änderungen im Systementwurfs-Objektmodell

Der Systementwurf zur Umsetzung der neuen Anforderung, TV-Shows anzuzeigen, verfeinert nun sowohl das Analyse-Objektmodell der zweiten Freigabe (siehe Abb. 10) als auch das Systementwurfs-Objektmodell der ersten Freigabe (siehe Abb. 11). Das Systementwurfs-Objektmodell der zweiten Freigabe enthält die Klasse TV-Show als Spezialisierung von Medium. Um TV-Shows empfangen zu können führt das Systementwurfs-Objektmodell Fernsehverbindung als neue Klasse ein.

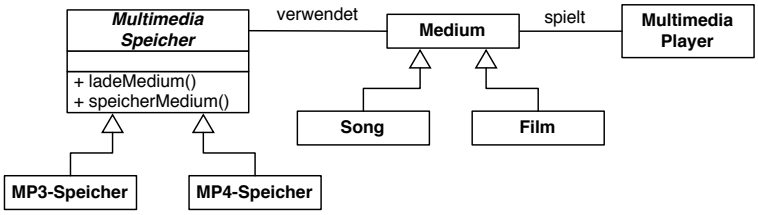


Abb. 13: Objektentwurfs-Objektmodell für die erste Freigabe des Multimedia Players.

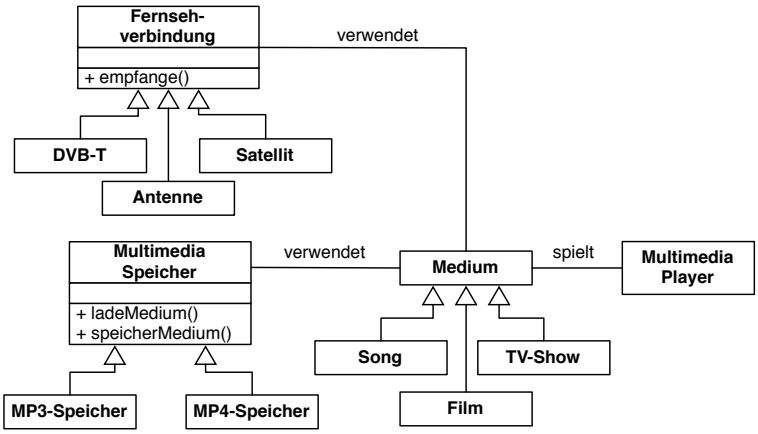


Abb. 14: Das Objektentwurfs-Objektmodell der zweiten Freigabe ist eine Verfeinerung des Systementwurfs-Objektmodells des Multimedia Players aus Abb. 12 und eine Verfeinerung des Objektentwurfs-Objektmodells der ersten Freigabe aus Abb. 13: Die Fernsehverbindung kann über DVB-T, Antenne oder Satellit erfolgen.

Änderungen im Objektentwurfs-Objektmodell

Auch das Objektentwurfs-Objektmodell der zweiten Freigabe berücksichtigt die Änderungen des Systementwurfs-Objektmodells in Abb. 12 und enthält die Klassen Fernsehverbindung und TV-Show aus dem Systementwurfs-Objektmodell. Zudem führt es das Strategie Muster ein, um auf verschiedene Weisen TV-Shows empfangen zu können: Über DVB-T, Antenne oder Satellit (siehe Abb. 14).

2.4 Berücksichtigung von Änderungen in Begründungsmodellen

Begründungsmodelle

Begründungsmodelle enthalten ein angesprochenes Problem, alternative Lösungsvorschläge für das Problem, Gründe für und gegen diese Lösungsvorschläge sowie schließlich die getroffene Lösung. Sie umfassen Rechtfertigungen für getroffene Entscheidungen und erlauben eine strukturierte Diskussion von Fragestellungen, um alternative Lösungsvorschläge zu erkunden und Gründe für Entscheidungen festzuhalten. [Dutoit et al 2006]

Änderungen

Der Eintritt einer Änderung bietet Gelegenheit bereits getroffene Entscheidungen, nochmals zu überdenken. Beispielsweise kann eine Änderung aus dem Objektentwurf zu einem neuen Lösungsvorschlag führen. Begrün-

dungsmodelle erlauben, neue Lösungsvorschläge mit anderen, bereits diskutierten, Lösungsvorschlägen zu vergleichen.

Implizite Begründungen

Wurden Begründungen nicht explizit erfasst, sondern sind sie implizit in den Köpfen der Interessenvertreter, müssen die Begründungen bei Eintritt einer Änderung rekonstruiert werden. Fragestellungen werden dann häufig nochmals durchdiskutiert, um Alternativen und Argumente festzustellen und in Anbetracht der Änderung zu beleuchten. Verlässt ein kritischer Mitarbeiter dann das Projekt, kann das Wissen über die Gründe der Entscheidungen nicht mehr verfügbar sein.

Explizite Begründungen

Werden Begründungsmodelle während der Entscheidungsfindung explizit erfasst, so kann an die bestehende Diskussion angeknüpft werden, indem die getroffene Entscheidung unter Berücksichtigung der Änderung nochmal überdacht und eventuell revidiert wird.

IBIS

Ein Beispiel für ein Begründungsmodell ist das *Issue-Based Information System*, kurz IBIS [Kunz & Rittel 1970]. Es verwendet Fragestellungen, Standpunkte, Argumente und Lösungen, um Begründungen festzuhalten. Fragestellungen sind „kritische Probleme ohne klare Lösung“ [Bruegge & Dutoit 2003]; als Lösung stehen mehrere Standpunkte, das heißt Alternativen, zur Auswahl. IBIS diskutiert Standpunkte mit Argumenten, um dann zu einer Lösung der Fragestellung zu kommen.

QOC

Questions, Options and Criteria [MacLean et al. 1991], kurz QOC, ist eine Erweiterung von IBIS. Fragen *(questions)* entsprechen den Fragestellungen aus IBIS; Optionen *(options)* sind mögliche Antworten auf Fragen und kommen den Positionen aus dem IBIS Modell gleich. Argumente sprechen für oder gegen Optionen und werden im Unterschied zu IBIS durch Kriterien bewertet.

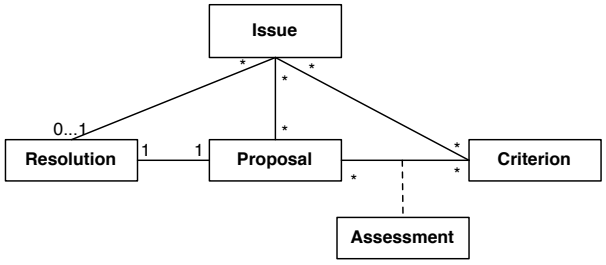


Abb. 15: RUSE – Begründungsmodell [Wolf 2007]

RUSE – Begründungsmodell

Das RUSE-Begründungsmodell [Wolf 2007] basiert auf dem Begründungsmodell *questions, options, criteria* und unterscheidet sich in der Bewertung von Kriterien. Für eine Fragestellung *(issue)* können mehrere alternative Lösungsvorschläge (*proposals*) aufgestellt werden. Kriterien (*criterion*) bewerten Lösungsvorschläge, um sie gegeneinander abzuwägen. Dabei wird jeder Lösungsvorschlag in Bezug auf ein Kriterium in einer Bewertung (*assessment*) beurteilt. An Hand dieser Beurteilung wird schließlich entschieden, welcher Lösungsvorschlag als Lösung (*solution*) verwendet wird (siehe Abb. 15).

2.5 Berücksichtigung von Änderungen im Freigabemanagement

Begründungsbasiertes Freigabemanagement

Um den Freigabemanager im Umgang mit Änderungen zu unterstützen, stellt diese Dissertation ein Modell zum **BE**gründungsbasi**E**rten **F**reigabemanagement, kurz BEEF, vor. Das Modell zum begründungsbasierten Freigabemanagement berücksichtigt die der Freigabe zu Grunde liegenden Modelle sowie ihre Abhängigkeiten und bildet die Grundlage für die Entscheidung des Freigabemanagers, ob und unter welchen Konsequenzen, beispielsweise durch eine Verschiebung des Liefertermins, die Änderung durchgeführt werden kann. BEEF unterstützt die Modellierungstechnik der inkrementellen Verfeinerung durch Freigabedeskriptoren.

Freigabedeskriptoren

Freigabedeskriptoren beschreiben die Entitäten und Abhängigkeiten einer Freigabe in Form von Analysemodellen, Systementwurfsmodellen und detaillierten Entwurfsmodellen. Roadmap und Releasepläne sind dynamische Sichten auf die Freigabedeskriptoren: Änderungen eines Freigabedeskriptors sind zugleich Änderungen in einem Releaseplan und der Roadmap. Fügt ein Analyst im Beispiel des Multimedia Players die neue Anforderung TV-Shows zum Analysemodell einer Freigabe hinzu, erscheinen TV-Shows sofort im Freigabeplan und der Roadmap. Der Freigabemanager ist sich dadurch über Änderungen aus der Entwicklung bewusst und kann die Freigabe für die neue Anforderung mit Freigabeoperationen ändern. [Herrmann 2007]

Freigabeszenario

Für den Fall, dass es mehrere Alternativen dafür gibt, wie mit der Änderung umgegangen werden kann, sieht BEEF Freigabeszenarios vor. Freigabeszenarios sind mögliche Freigaben und haben wie Freigaben einen Freigabedeskriptor. Der Freigabedeskriptor eines Freigabeszenarios beschreibt detailliert, wie eine Freigabe aussehen könnte. Daher kann der Freigabemanager mit Freigabeszenarios alternative Modelle einer Freigabe vergleichen und die Auswirkung einer Änderung auf die Freigabe simulieren. Entscheidet sich der Freigabemanager für ein Freigabeszenario kann er das gesamte Freigabeszenario mit seinem Freigabedeskriptor beschließen und als Freigabe verwenden. Oder der Freigabemanager passt die aktuell beschlossene Freigabe mit Freigabeoperationen an.

Freigabeoperationen

Hat der Freigabemanager eine Entscheidung getroffen, wie mit einer Änderung, beispielsweise der neuen Anforderung TV-Shows, umgegangen werden soll, kann er die Freigabe mit Freigabeoperationen verändern. Freigabeoperationen verändern Freigaben, indem sie Anforderungen zu einer Freigabe hinzufügen, von einer Freigabe entfernen oder von einer Freigabe auf eine andere Freigabe verschieben. Verschiebt der Freigabemanager die Anforderung Filme des Multimedia Players auf eine spätere Freigabe, passen sich die Freigabemodelle der betroffenen Freigaben an.

Knowledge Nuggets

Zur Beschreibung von Freigabemodellen verwendet BEEF *Knowledge Nuggets*. Ein Knowledge Nugget [Herrmann & Bruegge 2006] beschreibt ein konkretes Systemmodell für Entscheidungen im Freigabemanagement. Beispielsweise können die erste und zweite Freigabe des Multimedia Players durch sechs Knowledge Nuggets beschrieben werden, jeweils drei für eine Freigabe: Abb. 16 zeigt die drei Knowledge Nuggets der ersten Freigabe. Der erste Knowledge Nugget enthält das Analysemodell der ersten Freigabe, der zweite Knowledge Nugget beschreibt das Systementwurfsmodell der ersten Freigabe und der dritte enthält das detaillierte Entwurfsmodell der ersten Freigabe. Für die zweite Freigabe beschreibt wiederum ein Knowledge Nugget das Analysemodell, einer das Systementwurfsmodell und einer das detaillierte Entwurfsmodell. (siehe Abb. 17)

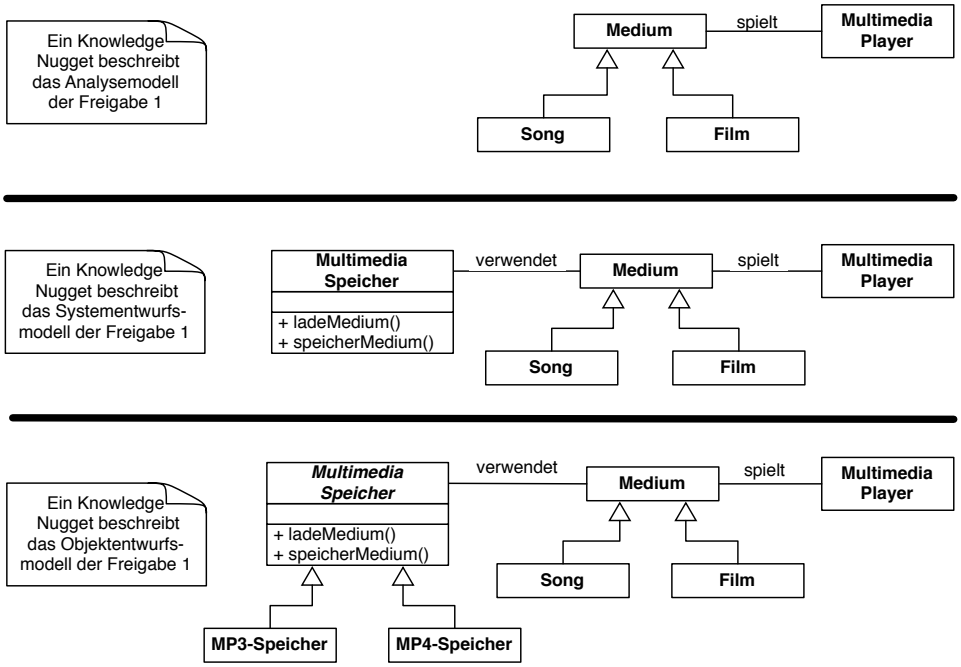


Abb. 16: Drei Knowledge Nuggets für die Modellierung der ersten Freigabe des Multimedia Players. Der erste Knowledge Nugget beschreibt das Analysemodell, der zweite das Systementwurfsmodell und der dritte das Objektentwurfsmodell.

Parallele Entwicklung von Systemmodellen

Knowledge Nuggets machen die Modelle aus Analyse, Systementwurf und detailliertem Entwurf einer Freigabe explizit. Sie verhindern, dass Systemmodelle durch die Modellierungstechnik der inkrementellen Verfeinerung verwässern. Beispielsweise erlauben sie, das Analysemodell oder das Modell einer bestimmten Freigabe festzuhalten. Als Konsequenz werden das Analysemodell, das Systementwurfsmodell und das Objektentwurfsmodell für mehrere Freigaben parallel entwickelt. Damit die Systemmodelle Entscheidungen unterstützen können, müssen sie konsistent sein, das heißt die Modelle dürfen sich nicht widersprechen. Folglich muss bei einer Änderung in einem Modell geprüft werden, wie sich die Änderung auf die übrigen Modelle auswirkt. Im Beispiel des Multimedia Players führt das Einführen von TV-Shows im Analysemodell der zweiten Freigabe zu einer Änderung des Systementwurfsmodells und des detaillierten Entwurfsmodells der zweiten Freigabe (siehe Abb. 17). Knowledge Nuggets helfen, diese Modelle konsistent zu halten, indem sie die Systemmodelle anderer Knowledge Nuggets bei Änderungen informieren.

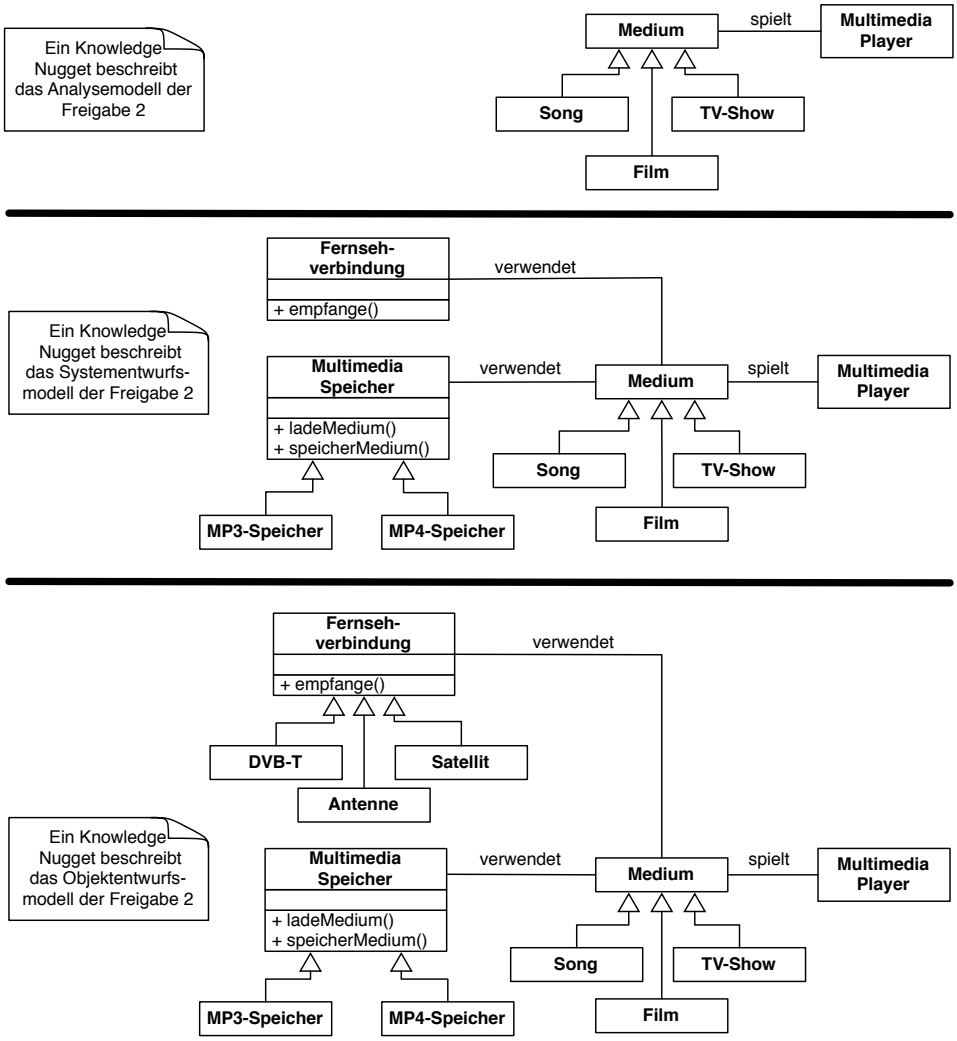


Abb. 17: Drei Knowledge Nuggets beschreiben die zweite Freigabe des Multimedia Players: Einer das Analysemodell, einer das Systementwurfsmodell und einer das Objektentwurfsmodell.

Visualisierung

Um Entwickler im Umgang mit mehreren parallelen Systemmodellen zu unterstützen, bietet BEEF Visualisierungen an, um das geeignete Modell zu finden sowie um Modelle zu vergleichen. Eine dreidimensionale Visualisierung [Bruegge et al. 2007] unterstützt den Entwickler bei der Auswahl eines Modells. Dazu gibt die dreidimensionale Visualisierung eine Übersicht für die Modelle jedes Knowledge Nuggets und erlaubt durch die Modelle zu navigieren. Der Entwickler kann in der dreidimensionalen Visualisierung Modelle auswählen und diese dann vergleichen. Der Vergleich der

Systemmodelle hebt in einer zweidimensionalen Ansicht die Unterschiede zwischen den Modellen farblich hervor. Zudem erlaubt ein Filtermechanismus, das Modell eines Knowledge Nuggets zu fokussieren und dieses weiterzuentwickeln. Beispielsweise kann sich der Analyst in der Analysephase nur das Analysemodell der zweiten Freigabe des Multimedia Players anzeigen lassen.

3 Freigabemodell

Strategische Planung

Roadmaps sind traditionell ein Instrument des Marketing und der Geschäftsführung, um Unternehmensstrategien festzulegen und Geschäftsziele, wie Absatzzahlen oder Gewinnerwartungen, zu planen. Um diese Ziele zu erreichen, planen Roadmaps strategisch die Entwicklung eines Systems über mehrere Jahre. Eine traditionelle Definition von Roadmaps enthält Geschäftsziele und Merkmale sowie Liefertermine für die nächsten Freigaben. [Ruhe 2005]

Operationelle Planung

Die in der Roadmap definierten Ziele gelten als Vorgaben für die operationelle Planung. Die operationelle Planung versucht die Systementwicklungsaktivitäten so zu planen, dass die Vorgaben der Roadmap erreicht werden. Änderungen der Roadmap unter Berücksichtigung der Ergebnisse der Systementwicklung sind nicht vorgesehen. [Ruhe 2005]

Roadmap

Agile Prozesse benötigen jedoch dynamische Roadmaps, um auf Änderungen aus der Systementwicklung flexibel reagieren zu können. Diese Dissertation verwendet daher eine technische Definition von Roadmap als Folge von Freigaben. In Abb. 18 besteht eine Roadmap daher aus mehreren, zeitlich geordneten Freigaben, die sowohl in der Vergangenheit als auch in der Zukunft liegen können.

Freigabedeskriptoren

Ein Freigabedeskriptor ist einer konkreten Freigabe zugeordnet und beschreibt diese, indem er die zu dieser Freigabe gehörigen Systemmodelle repräsentiert. Der Freigabedeskriptor einer in Vergangenheit liegenden Freigabe dokumentiert das gelieferte System durch detaillierte Systemmodelle, beispielsweise eine genaue Modellierung der Anforderungen, der Systemarchitektur und einem Objektentwurfs-Objektmodell mit allen

implementierten Klassen, Methoden und Assoziationen. Der Freigabedeskriptor einer in der Zukunft liegenden Freigabe enthält die detaillierten Modelle der letzten ihr zugrundeliegenden Freigabe. Erweiterungen sind – abhängig davon wie weit die Entwicklung der zukünftigen Freigabe bereits fortgeschritten ist – zunächst visionär und vage modelliert und beispielsweise nur durch Merkmale beschrieben.

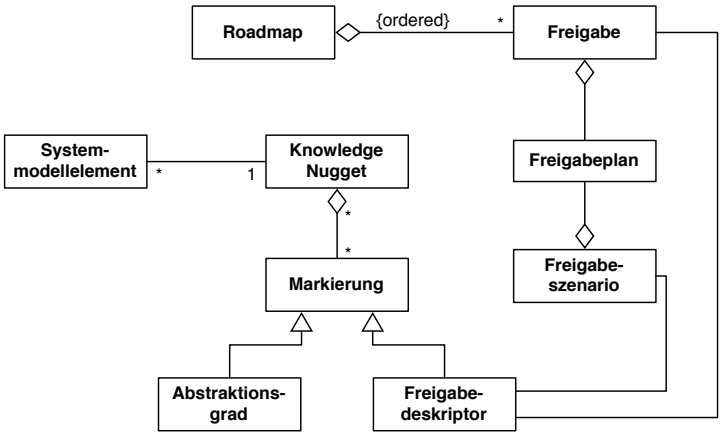


Abb. 18: Überblick über Elemente von BEEF. Eine Roadmap besteht aus mehreren aufeinanderfolgenden Freigaben. Knowledge Nuggets enthalten das Systemmodell des freizugebenden Systems und werden durch Markierungen zu Freigaben oder Freigabeszenarios zugeordnet.

Freigabepläne

Eine Freigabe hat einen Freigabeplan (siehe 3.4), das heißt eine Übersicht, was erledigt oder implementiert werden soll, um die Freigabe an den Kunden liefern zu können. Freigabepläne für zukünftige Freigaben können aus den Modellen zweier aufeinanderfolgender Freigaben generiert werden, indem sie die Unterschiede zwischen den Freigabedeskriptoren zweier aufeinanderfolgender Freigaben bestimmen und filtern. Der Informationsgehalt des Freigabeplans einer zukünftigen Freigabe hängt davon ab, wie detailliert die zukünftige Freigabe bereits modelliert ist. Ist eine zukünftige Freigabe noch wenig unkonkret und visionär, kann der Freigabeplan auch nur Visionen enthalten. Entwickeln sich aus den Visionen konkrete Merkmale oder Anwendungsfälle, so kann ein Freigabemanager diese zur Freigabeplanung verwenden.

Freigabeszenarios

Freigabeszenarios dienen als Alternativen für Freigaben und erlauben dem Projektmanager, verschiedene Möglichkeiten, eine Freigabe umzusetzen und zu vergleichen. Freigabeszenarios bestehen wie Freigaben aus einem Freigabedeskriptor und einem Freigabeplan.

Knowledge Nuggets

Knowledge Nuggets dienen der Modellierung dieser aufeinanderfolgenden Freigaben. Ein Knowledge Nugget besteht aus mehreren Systemmodellelementen, das heißt Entitäten zur Beschreibung und Dokumentation des Systems. Beispiele für Systemmodellelemente sind Anforderungen, wie Merkmale, Anwendungsfälle oder *user stories*, oder Klassen. Knowledge Nuggets einer nachfolgenden Freigabe enthalten in der Regel mehr Anforderungen als das Freigabemodell der vorangegangenen Freigabe. Ein einzelnes Systemmodell für jede Freigabe zu haben, unterstützt die Modellierungstechnik der inkrementellen Verfeinerung: In *Extreme Programming* wird das Systemmodell inkrementell mit jeder *user story* erweitert. In jeder Freigabe werden dem System, und damit den Knowledge Nuggets der Freigabe, nur solche Artefakte hinzugefügt, die für die Umsetzung einer neuen *user story* nötig sind. [Beck 2001] Andere Anforderungen können den Knowledge Nuggets künftiger Freigaben zugeordnet und erst bei ihrer Entwicklung genauer spezifiziert werden.

Markierungen

Da die Entwicklung des Systems das Modell nicht nur von Freigabe zu Freigabe schrittweise verfeinert, sondern die inkrementelle Verfeinerung auch innerhalb einer Freigabe von einem Analysemodell zu einem detaillierten Entwurfsmodell stattfindet, können bei der Modellierung einer Freigabe mehrere Knowledge Nuggets zum Einsatz kommen. Um die Modelle eines Knowledge Nuggets einordnen zu können, verwenden wir Markierungen. Eine Markierung ist eine Bezeichnung, die angibt, welches Systemmodell ein Knowledge Nugget beschreibt. Zum Beispiel kann ein Modell mit Analyse und Freigabe 1 markiert sein, um auszudrücken, dass es sich um ein Analysemodell der Freigabe 1 handelt. Diese Dissertation verwendet Freigabedeskriptoren und Abstraktionsgrade als Markierungen. Freigabedeskriptoren geben an, welcher Freigabe das Modell eines Knowledge Nugget zuzuordnen ist. Ein Abstraktionsgrad gibt an, wie viel Implementierungsdetail ein Modell enthält. Ein Modell mit einem hohen Abstraktionsgrad enthält mehr Konzepte der Anwendungsdomäne und weniger Details der Implementierung als ein Mo-

dell mit geringem Abstraktionsgrad. Beispielsweise hat ein Analysemodell einen höheren Abstraktionsgrad als ein detailliertes Entwurfsmodell.

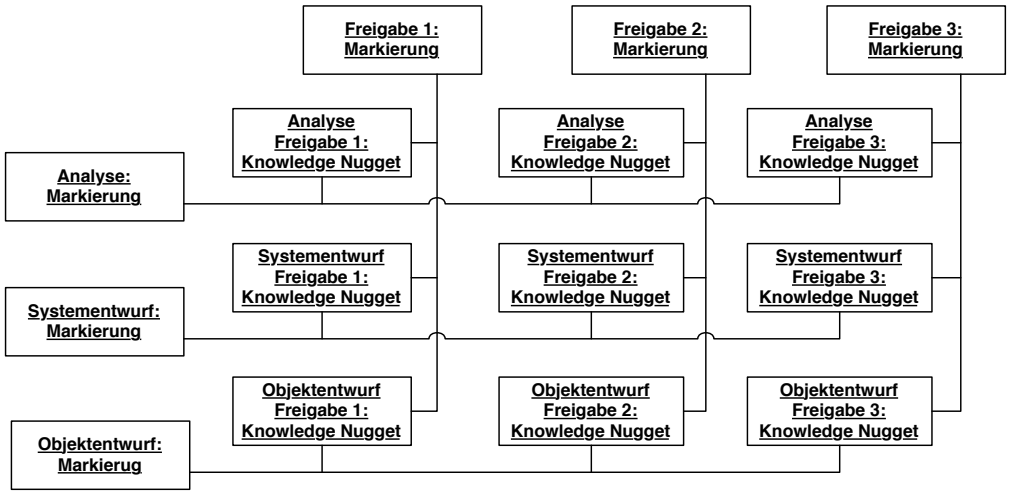


Abb. 19: Knowledge Nuggets zur Darstellung von Modellen mit Abstraktionsgraden mehrerer aufeinanderfolgender Freigaben

Instanzen von Markierungen

Knowledge nuggets sollten eine Instanz jeder Markierung haben, so dass klar ist, was sie beschreiben. Stehen Abstraktionsgrad und Freigabedeskriptoren als Markierungen zur Verfügung, sollte der Knowledge Nugget einer Freigabe und einem Abstraktionsgrad zugewiesen sein. Abb. 19 zeigt ein Instanzdiagramm für die Knowledge Nuggets dreier Freigaben und dreier Abstraktionsgrade: Es besteht aus neun *knowledge nugget* Instanzen, von denen jeder das Modell eines Abstraktionsgrades einer Freigabe beschreibt.

3.1 Freigaben

Freigaben im Konfigurationsmanagement

Freigaben kommen traditionell aus dem Konfigurationsmanagement und bezeichnen Versionen, die anderen Entwicklern in Form von internen Freigaben zur Verfügung gestellt werden oder als externe Freigaben an den Kunden geliefert werden. Versionen im Konfigurationsmanagement dienen dem Speichern von Änderungen an Artefakten eines Softwareprojekts, sogenannten kontrollierten Teilen: Jede Änderungen an einem kontrollierten Teil führt zu einer neuen Version. Das Speichern von Änderungen in Versionen

ermöglicht zu einem späteren Zeitpunkt den alten Zustand eines kontrollierten Teils wiederherzustellen, beispielsweise wenn eine frühere stabile Version für eine Freigabe verwendet werden soll. [IEEE 1987]

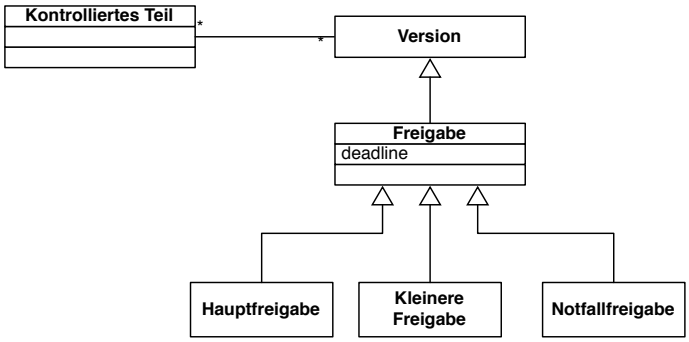


Abb. 20: Arten von Freigaben: Hauptfreigaben, kleinere Freigaben und Notfallfreigaben sind im Konfigurationsmanagement Versionen von kontrollierten Teilen.

Arten von Freigaben

Abb. 20 zeigt drei verschiedene Arten von Freigaben [Elsässer 2006] [Bays 1999]: Hauptfreigaben (*major releases* oder *upgrades*), kleinere Freigaben (*minor releases* oder *updates*) und Notfallfreigaben (*emergency fixes* oder *patches*). Hauptfreigaben sind Freigaben, die wesentliche neue Funktionalität sowie eine Reihe an Fehlerverbesserungen mit sich bringen. Mit einer Hauptfreigabe sind häufig Änderungen an der grafischen Benutzeroberfläche verbunden. Kleinere Freigaben adressieren Fehler oder Probleme in Hauptfreigaben und liefern für diese Anpassungen der Funktionalität, beispielsweise an neue Hardware, sowie eine Menge von Fehlerverbesserungen. Notfallfreigaben dienen der schnellen Behebung kritischer Fehler; kritische Fehler sind Fehler in wichtigen Teilen des Systems mit der Folge, dass Anwender ihre Arbeit nicht mehr ausführen können und so zu Produktivitätsausfällen führen. Eine Notfallfreigabe wird schnell entwickelt, um einen solchen kritischen Fehler zu beheben, wenn mit der Behebung nicht bis zur nächsten kleineren Freigabe gewartet werden kann.

Kontrollierte Teile

Für kontrollierte Teile verwaltet das Konfigurationsmanagement Änderungen, wobei man hier zwischen Konfigurationselementen und Konfigurationsaggregaten unterscheidet. Konfigurationselemente sind Teile des Systems,

die für das Freigabemanagement als eine Einheit angesehen werden. Beispielsweise sind Dateien des Quellcodes, Komponenten, externe Systeme oder Systemmodelle kontrollierte Teile. Konfigurationsaggregate dienen der Zusammenstellung mehrerer Konfigurationselemente, zum Beispiel für eine Lieferung. Beispielsweise kann ein Konfigurationsaggregat den lauffähigen Code, weitere benötigte Komponenten oder Systeme sowie die Dokumentation enthalten.

Builds

Für jede Version des Quellcodes kann durch Kompilieren genau ein *build* erstellt werden, der das lauffähige System dieser Version darstellt. Dieser *build* kann im Nachhinein nicht mehr verändert werden: Schlägt das Erstellen des *builds* fehl oder werden durch Tests, die beispielsweise im Zuge einer kontinuierlichen Integration ausgeführt werden, Fehler identifiziert, führen diese zu neuen Änderungen an kontrollierten Teilen. Diese Änderungen führen zu einer neuen Version und somit zu neuen *builds*.

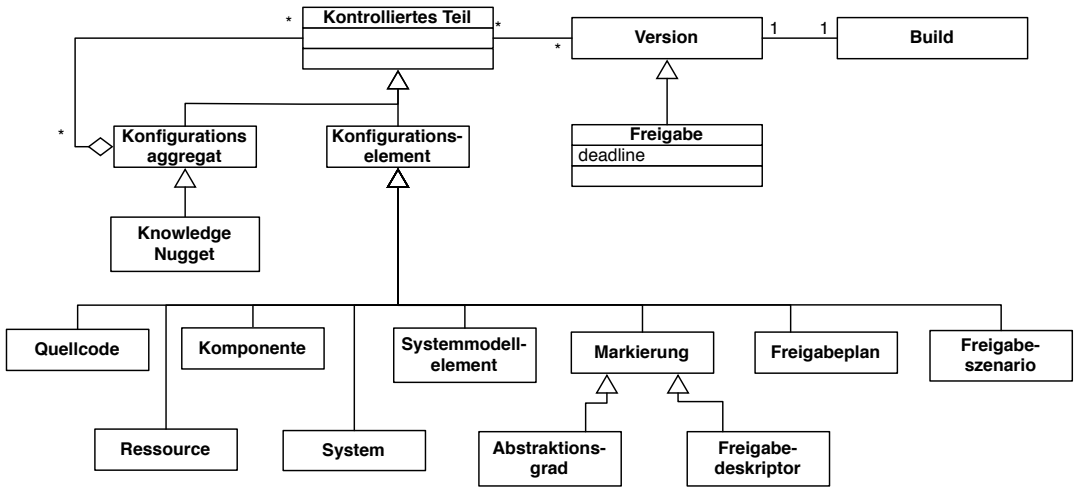


Abb. 21: Bestandteile einer Freigabe: Eine Freigabe ist eine Version eines kontrollierten Teils. Kontrollierte Teile können durch das Kompositum Muster [Gamma et al. 2004] modelliert werden: Konfigurationsaggregate bestehen aus mehreren Konfigurationselementen, die wiederum kontrollierte Teile, d.h. andere Konfigurationsaggregate oder Konfigurationselemente, sind.

Definitionen von Freigabemanagement

Das Modell aus Abb. 21 deckt drei verschiedene Ansichten von Freigabemanagement ab: Erstens unterstützt das Modell die Zusammenstellung einer Freigabe als Dienstleistung, die aus mehreren, aufeinander abgestimmten

Software- und Hardwaresystemen von Fremdherstellern bestehen [Victor & Günther 2005]. Zweitens baut das Modell auf den bestehenden Ansätzen des Konfigurationsmanagements auf und behält die Freigabe als eine spezielle Version eines sich entwickelnden Systems. [IEEE 1987] Drittens unterstützt das Modell Roadmapping und Freigabeplanung, indem es Knowledge Nuggets als neue Konfigurationsaggregate einführt, die Systemmodellelemente einer Freigabe beschreiben. Mit ihnen kann ausgehend von geplanten Merkmalen ein sich entwickelndes System modelliert werden.

Knowledge Nuggets als kontrolliertes Teil

Durch die Modellierung von Knowledge Nugget und Systemmodellelementen als kontrollierte Teile unterliegen diese auch der Kontrolle des Konfigurationsmanagements, das heißt Änderungen an ihnen können festgehalten werden. Knowledge Nuggets enthalten Systemmodellelemente und Markierungen als Konfigurationselemente. Der Freigabedeskriptor ist eine Markierung, um die Brücke zwischen dem Knowledge Nugget und der zugehörigen Freigabe zu schlagen.

Merkmale einer Freigabe

Dem Knowledge Nugget einer Freigabe können so zunächst eine Menge von Merkmalen, die in der Freigabe umgesetzt werden können, zugewiesen werden. Ein Merkmal ist ein Systemmodellelement, das eine kurze prägnante Beschreibung einer Funktionalität eines Systems beinhaltet [Palmer & Felsing 2002]. Es gibt einen Überblick, wie eine Funktionalität aussehen soll, und ist im Laufe der Systementwicklung, beispielsweise durch Anwendungsfälle noch genauer zu spezifizieren. Wie das Merkmal dann genau implementiert wird, hängt nicht zuletzt davon ab, wie viele Ressourcen für eine Freigabe zur Verfügung stehen.

3.2 Ressourcen

Ressourcen

Ressourcen, das heißt Wirtschaftsgüter die zur Implementierung der Freigaben verwendet werden können, leitet der Freigabemanager traditionell aus den Unternehmenszielen ab. Diese Dissertation erlaubt auch eine Änderung von Ressourcen, beispielsweise um ein kritisches Merkmal noch umsetzen zu können, indem Ressourcen dem Konfigurationsmanagement unterworfen

werden. Neue Ressourcen können hinzugefügt oder die Verfügbarkeit einer Ressource angepasst werden.

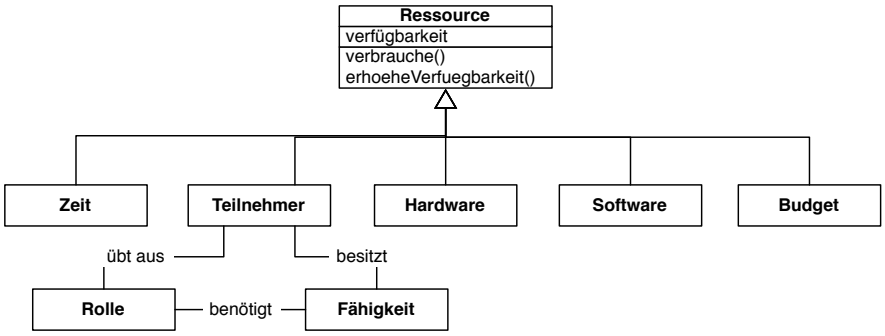


Abb. 22: Ressourcen zur Erstellung einer Freigabe sind Software, Hardware, Budget und Zeit sowie Teilnehmer mit ihren Fähigkeiten und Rollen. Ressourcen haben eine Verfügbarkeit, die verbraucht sein kann oder bei Bedarf auch erhöht werden kann.

Beispiele für Ressourcen

Abb. 22 gibt einen Überblick über Ressourcen bei der Erstellung einer Freigabe und enthält Zeit, Teilnehmer, Hardware sowie das Budget. Die Zeit, die zur Entwicklung einer Freigabe zur Verfügung steht, hängt von der Deadline ab, die für die Freigabe definiert worden ist. Die Entwicklungszeit einer aktuellen Freigabe F_A ist typischerweise auf die Zeitspanne zwischen der aktuellen Freigabe F_A und der ihr vorangehenden Freigabe begrenzt. Die Entwicklungszeit kann verlängert werden, wenn die Entwicklung mehrerer Freigaben parallel erfolgt, beispielsweise wenn die Auswirkungen von Änderungen zunächst erprobt werden sollen.

Teilnehmer

Teilnehmer sind alle Personen, die an der Erstellung einer Freigabe mitwirken. Beispiele für Teilnehmer sind Entwickler, Kunden und Endanwender. Teilnehmer übernehmen Rollen, die ihre Verantwortlichkeiten für die Freigabe beschreiben; Rollen sind beispielsweise ein Freigabemanager, Analysten, oder Architekten. Nach Möglichkeit sollte die Zuweisung von Projektteilnehmern zu Rollen die Fähigkeiten des Teilnehmers berücksichtigen.

Hardware

Hardware bezeichnet Geräte, die zur Erstellung der Freigabe zur Verfügung stehen. Dazu gehören neben den Rechnern für die Systementwickler auch spezielle Geräte, auf denen das zu entwickelnde System laufen oder mit denen das zu entwickelnde System interagieren soll. Solche Geräte können beispielsweise Steuergeräte für die Entwicklung von Software in Fahrzeugen

oder mobile Endgeräte, wie Kameras, GPS oder Mobiltelefone, bei der Entwicklung mobiler Anwendungen sein. Sollen innovative Technologien unterstützt werden, kann es vorkommen, dass benötigte Geräte nicht unbegrenzt zur Verfügung stehen, vor allem dann wenn sich Technologien oder Geräte selbst noch in Entwicklung befinden.

Budget

Das Budget gibt an, wie viel Geld zur Erstellung einer Freigabe zur Verfügung steht. Mit dem Budget können andere Ressourcen erhöht werden, beispielsweise zusätzliche Hard- oder Softwarekomponenten eingekauft oder neue Entwickler eingestellt werden.

3.3 Erzeugung einer Freigabe

Anzahl benötigter Knowledge Nuggets

Bei der Erzeugung einer Freigabe werden Knowledge Nuggets zur Darstellung der Systemmodellelemente erstellt. Die Anzahl benötigter Knowledge Nuggets hängt davon ab, wie viele Markierungen existieren. Für die Erzeugung einer Freigabe mit drei Abstraktionsniveaus sind beispielsweise drei neue Knowledge Nuggets nötig. Sollen Abstraktionsgrade hingegen nicht berücksichtigt werden, reicht es aus, nur einen Knowledge Nugget für die neue Freigabe zu erzeugen.

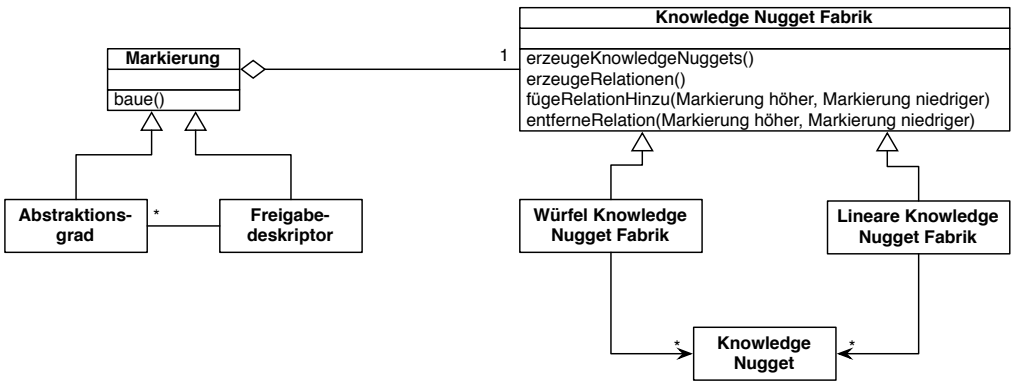


Abb. 23: *Abstract Builder* Entwurfsmuster [Gamma et al. 2004] zur Erzeugung von Knowledge Nuggets für eine Freigabe: Die Markierung entscheidet darüber, welche Fabrik zur Erzeugung von Knowledge Nuggets eingesetzt wird.

Kowledge Nugget Fabrik

Wird eine neue Freigabe geplant, führt dies auch zur Erstellung eines neuen Freigabedeskriptors, der genau der neuen Freigabe zugeordnet ist. Dieser Freigabedeskriptor trifft dann die Entscheidung darüber, wie die Knowledge Nuggets erzeugt werden und verwendet eine Knowledge Nugget Fabrik. Eine Knowledge Nugget Fabrik kann Knowledge Nuggets für Markierungen erzeugen und Relationen zu bestehenden Knowledge Nuggets erstellen (siehe Abb. 23).

Relationen zwischen Knowledge Nuggets

Relationen zwischen Knowledge Nuggets zeigen, wie diese zueinander in Beziehung stehen. Knowledge Nuggets zweier aufeinanderfolgender Freigaben stehen analog zu der in der Roadmap definierten zeitlichen Reihenfolge der Freigaben in Relation. Eine Relation zwischen den Knowledge Nuggets zweier Freigaben ist also zum einen zeitlich, zum anderen aber auch als Weiterentwicklung des Systems zu interpretieren, weil eine spätere Freigabe neue Merkmale, Verbesserungen oder Fehlerbehebungen enthält. Ebenso stehen Knowledge Nuggets für Modelle auf verschiedenen Abstraktionsgraden in Relation: Ein Knowledge Nugget zur Beschreibung des Analysemodells ist auf einer höheren Abstraktionsebene als ein Knowledge Nugget zur Beschreibung des Systementwurfsmodells.

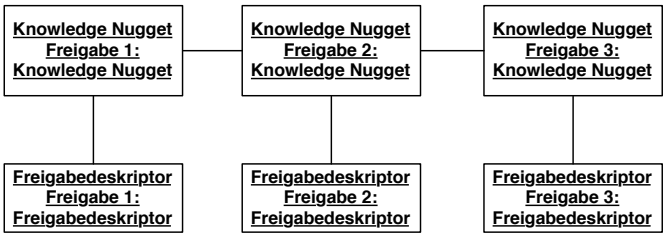


Abb. 24: Ergebnis einer Linearen Knowledge Fabrik: Sie erzeugt einen Knowledge Nugget für jede Instanz einer Markierung.

Lineare Knowledge Nugget Fabrik

Eine Lineare Knowledge Nugget Fabrik erzeugt eine Reihe von linear angeordneten, aufeinanderfolgenden Knowledge Nuggets. Sie findet Einsatz, wenn entweder nur eine Freigabe oder kein Abstraktionsgrad berücksichtigt wird. Das Ergebnis einer Linearen Knowledge Nugget Fabrik für drei Freigaben und keinen Abstraktionsgrad (siehe Abb. 24) sind drei Knowledge Nuggets, die miteinander in Relation stehen. Der Knowledge Nugget der

Freigabe 1 ist höher, das heißt er beschreibt ein abstrakteres Modell, als der Knowledge Nugget der Freigabe 2 und dieser ist wiederum höher als der Knowledge Nugget der Freigabe 3.

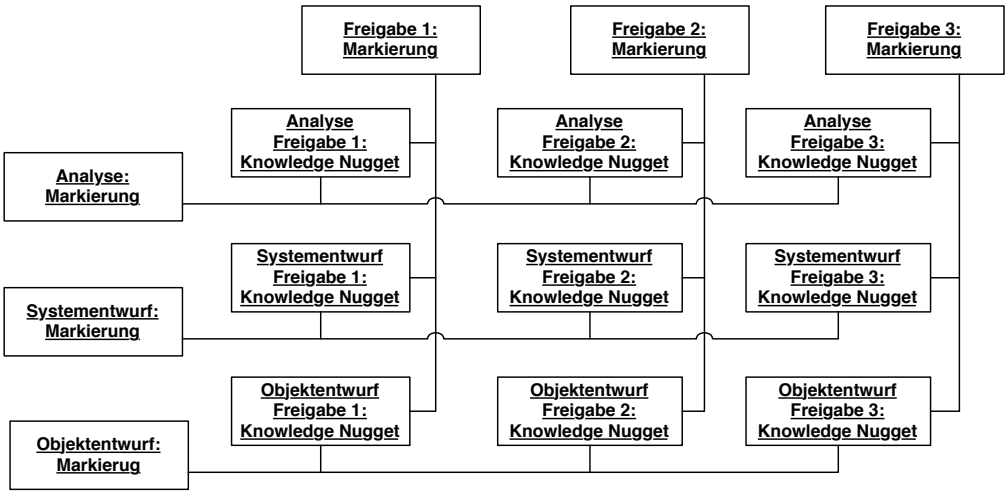


Abb. 25: Ergebnis einer Würfel Knowledge Nugget Fabrik: Es wird ein Knowledge Nugget für jede Kombination von Markierungen erzeugt.

Würfel Knowledge Nugget Fabrik

Wird in einem Projekt mehr als eine Markierung eingesetzt, also beispielsweise Freigabedeskriptoren und Abstraktionsgrade, verwendet eine neu erzeugte Markierung die Würfel Knowledge Nugget Fabrik. Sie erzeugt für jede Kombination der Markierungsklassen genau einen Knowledge Nugget. Abb. 25 zeigt als Beispiel das Resultat einer Würfel Knowledge Nugget Fabrik bei 3 Freigaben und 3 Abstraktionsgraden für Analyse, Systementwurf und detaillierten Entwurf. Es entstehen 9 Knowledge Nuggets, pro Freigabe einer für das Analysemodell, einer für das Systementwurfsmodell und einer für das detaillierte Entwurfsmodell.

Inkrementelle Entwicklung von Freigaben

Die erzeugten Knowledge Nuggets stellen den Ausgangspunkt für die Roadmap und somit die schrittweise Entwicklung einer Folge von Freigaben dar. Während der strategischen Planung werden den Knowledge Nuggets, die das Analysemodell beschreiben, erste Merkmale zugewiesen. Diese werden dann während der Systementwicklung in Richtung ausführbarem Code schrittweise weiter verfeinert. Bei Änderungen ist es Aufgabe des Freigabemana-

gers zu entscheiden, ob die Systemmodellelemente der Knowledge Nuggets angepasst werden müssen.

3.4 Freigabepläne

Freigabeplanungsentitäten

Ein Freigabeplan basiert auf dem Knowledge Nugget einer Freigabe und zeigt Freigabeplanungsentitäten, das sind Entitäten, die noch erledigt werden müssen, um die Freigabe an den Kunden liefern zu können. Freigabeplanungsentitäten können Anforderungen, Merkmale, Anwendungsfälle, Szenarios, bei *eXtreme programing user stories*, Fehlerberichte, Fragestellungen oder Arbeitseinheiten sein (siehe Abb. 26). Welche Freigabeplanungsentitäten in einem Projekt konkret gewählt werden, hängt von der gewählten Vorgehensweise ab. Bei einer entitätsbasierten Vorgehensweise wird der Freigabeplan aus Entitäten des Systemmodells, also beispielsweise Anforderungen bestehen. Bei einem aktivitätsorientierten Vorgehen hingegen werden Arbeitseinheiten den Freigabeplan dominieren.

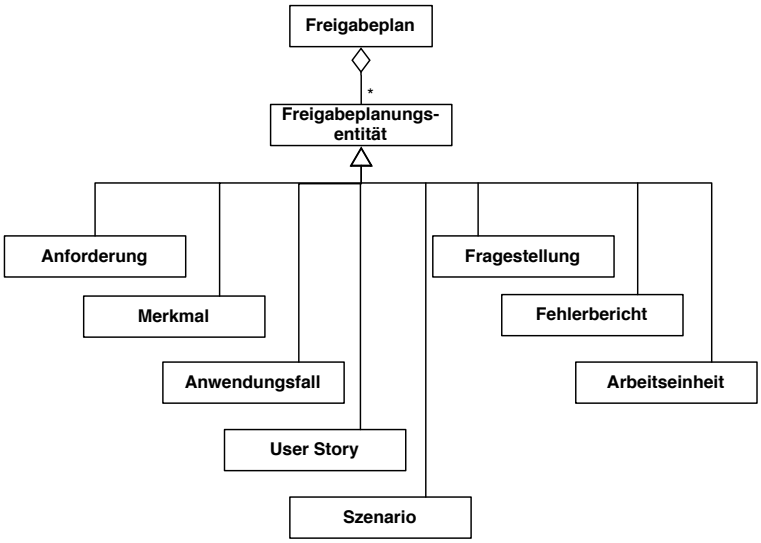


Abb. 26: Freigabepläne: Ein Freigabeplan besteht aus mehreren Freigabeplanungsentitäten. Freigabeplanungsentitäten können Anforderungen, Anwendungsfälle, Merkmale, Fehlerberichte, Fragestellungen, User Stories Szenarios oder Arbeitseinheiten sein.

Änderungen der Freigabe

Um Änderungen aus der Systementwicklung zu berücksichtigen, werden Freigabepläne aus den Freigabedeskriptoren generiert. Betrachtet man zwei aufeinanderfolgende Freigaben, so beschreiben die Freigabeplanungsentitäten, die in der nachfolgenden Freigabe neu hinzugekommen sind, den Freigabeplan der Freigabe. (siehe Abb. 27)

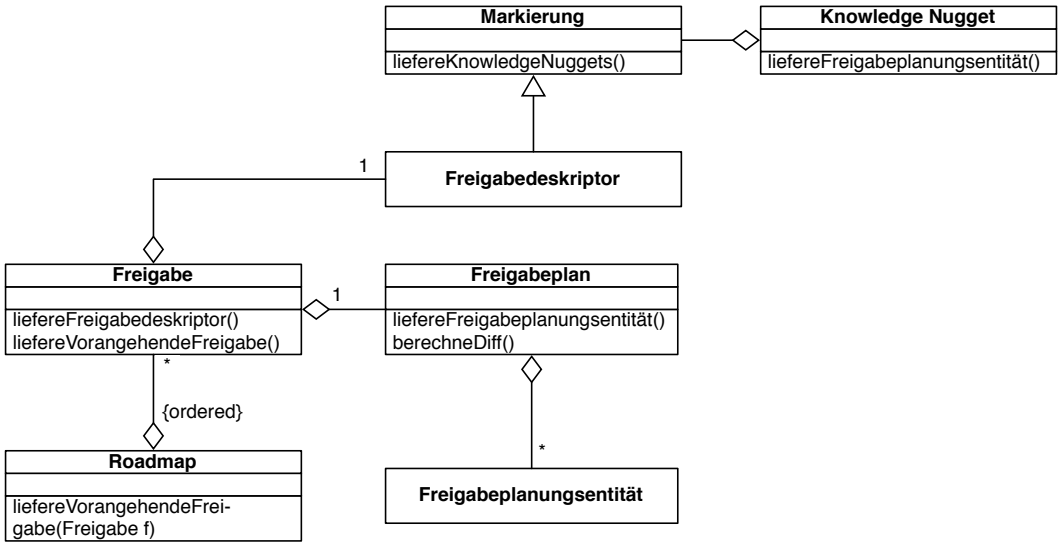


Abb. 27: Freigabenpläne: Eine Freigabe hat einen Freigabeplan bestehend aus Freigabeplanungsentitäten. Diese filtert der Freigabeplan aus dem Freigabedeskriptor, den Knowledge Nuggets verwalten.

Generierung eines Freigabeplans

Abb. 28 und Abb. 29 illustrieren, wie die Freigabeplanungsentitäten für den Freigabeplan einer Freigabe F2 bestimmt werden können. Der Freigabeplan fragt dazu die Roadmap nach der Freigabe, die der Freigabe F2 vorangeht, und erhält die Freigabe F1. Jede Freigabe kann mit liefereFreigabedeskriptor() ihren Freigabedeskriptor zurückgeben. Freigabedeskriptoren verfügen über eine Methode liefereKnowledgeNuggets(), um diejenigen Knowledge Nuggets zu erhalten, die zu einer Freigabe gehören (siehe Abb. 27). Der Aufruf dieser beiden Methoden liefert in Abb. 29 den Freigabedeskriptor Freigabe F1 und den Knowledge Nugget Freigabe F1 für die Freigabe F1 sowie in Abb. 29 den Freigabedeskriptor Freigabe F2 und den Knowledge Nugget Freigabe F2 für die Freigabe F2 zurück. Diese Knowledge Nuggets können ihre Freigabeplanungsentitäten liefern. Der Freigabeplan

bestimmt in der Methode berechneDiff() nun die Freigabeplanungsentitäten, die in Freigabe F2 und nicht in Freigabe F1 vorkommen.

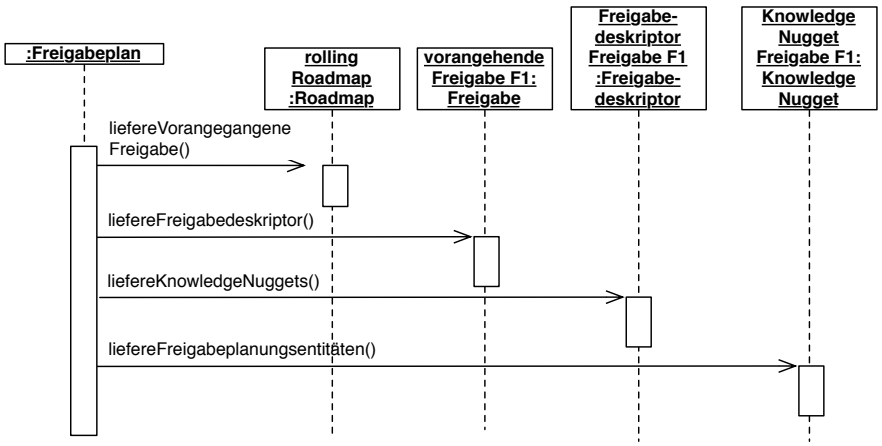


Abb. 28: Dynamisches Modell zur Generierung eines Freigabeplans (UML-Sequenzdiagramm)

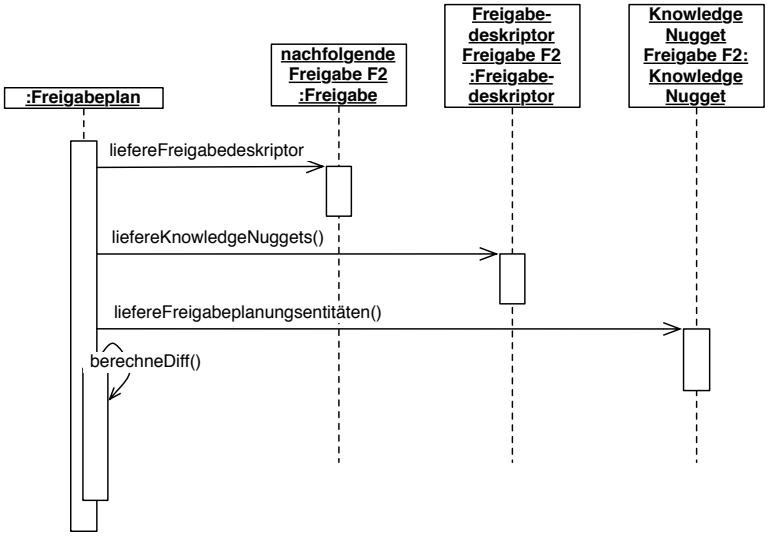


Abb. 29: Dynamisches Modell zur Generierung eines Freigabeplans. Forts. von Abb. 28 (UML-Sequenzdiagramm)

Freigabeplan für Multimedia Player

Abb. 30 zeigt als Beispiel ein UML Instanzdiagramm für die Freigaben 1 und 2 des Multimedia Players. In der Freigabe 1 soll der Anwendungsfall Songs-Abspielen geliefert werden, in Freigabe 2 zusätzlich der Anwendungsfall VideoAbspielen. Der Anwendungsfall SongsAbspielen ist bei der Erstellung der Freigabe 2 bereits implementiert und erscheint daher nur im Freigabe-

plan der Freigabe 1. Da der Anwendungsfall VideoAbspielen in Freigabe 2 neu ist, muss er dort noch umgesetzt werden und erscheint deshalb im Freigabeplan der Freigabe 2.

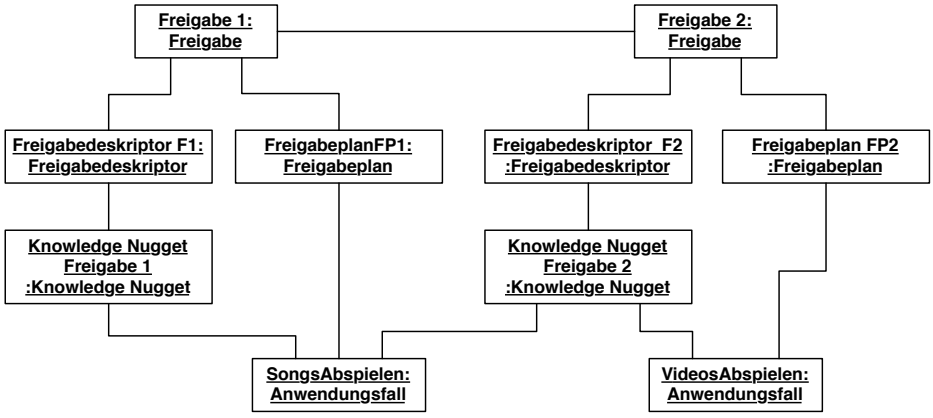


Abb. 30: Freigabepläne und Knowledge Nuggets für zwei aufeinanderfolgende Freigaben 1 und 2. Der Knowledge Nugget der Freigabe 1 enthält nur den Anwendungsfall Songs Abspielen. Der Knowledge Nugget der Freigabe 2 umfasst SongsAbspielen und Video Abspielen. Im Gegensatz dazu umfasst der Freigabeplan der Freigabe 2 nur den Anwendungsfall VideoAbspielen.

Dynamische Roadmaps

Ein Freigabeplan kann also direkt aus den Freigabedeskriptoren abgeleitet werden und ist damit stets konsistent mit den aktuellen Ergebnissen der Systementwicklung. Dadurch verschmilzt das hier vorgestellte Freigabemodell die operationelle und strategische Planung und erlaubt dynamische Roadmaps: Änderungen aus der Systementwicklung wirken sich unmittelbar auf die Planung einer Freigabe und die Roadmap aus. Identifiziert beispielsweise ein Entwickler ein neues Merkmal und fügt es einer Freigabe hinzu, führt die Verbindung von Roadmaps mit Knowledge Nuggets direkt zu einer Anpassung der Roadmap und des Freigabeplans. Kann das Merkmal nicht mehr unter Berücksichtigung der zur Verfügung stehenden Ressourcen implementiert werden, muss der Freigabemanager entscheiden, wie die Roadmap geändert werden soll. Dazu verwendet der Freigabemanager Freigabeoperationen, die im nächsten Abschnitt beschrieben werden.

3.5 Freigabeoperationen

Änderungen der Freigaben

Der vorangehende Abschnitt hat gezeigt, wie Freigabepläne direkt aus den Modellen der Knowledge Nuggets generiert werden. Ändern Entwickler diese Modelle führt dies als Folge zu einer Veränderung der Freigabepläne, wodurch sich der Freigabemanager über Änderungen aus den Systementwicklungsaktivitäten bewusst ist. Analog kann ein Freigabemanager Freigabeoperationen verwenden, um die Freigabepläne sowie die Modelle der Knowledge Nuggets zu modifizieren. Die sofortige Änderung der Modelle bei Anpassungen durch den Freigabemanager führt zu einer erhöhten Bewusstheit über Änderungen bei den Entwicklern.

Freigabeoperationen

Freigabeoperationen verändern aufeinanderfolgende Freigaben, indem sie Freigabeplanungsentitäten hinzufügen, entfernen oder verschieben; zudem können Freigaben zusammengelegt oder gesplittet werden.

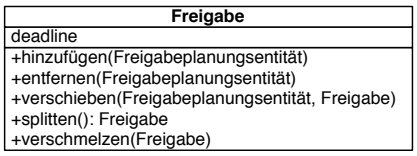


Abb. 31: Detaillierter Entwurf einer Freigabe

Hinzufügen von Freigabeplanungsentitäten

Die Operation hinzufügen() fügt der Freigabe eine Freigabeplanungsentität hinzu, beispielsweise wenn ein neuer Anwendungsfall identifiziert wurde. Durch das Hinzufügen gehört die Freigabeplanungsentität zur betreffenden Freigabe, das heißt sie wird dem Freigabedeskriptor hinzugefügt. Neben der Freigabeplanungsentität selbst wird überprüft, ob andere Artefakte, die notwendig sind, um die Freigabeplanungsentität umzusetzen, existieren. Ist dies der Fall, werden diese ebenfalls dem Modell der Freigabe hinzugefügt. Beispielsweise besteht ein Anwendungsfall aus einem Ereignisfluss, der die Schritte der Akteure und des Systems beschreibt. Jacobson [Jacobson et al. 1992] [Jacobson et al. 1994] leitet aus diesen Schritten partizipierende Objekte eines Anwendungsfalls ab und stellt so eine Verbindung zwischen Anwendungsfällen und den Objektmodell her. Wird ein Anwendungsfall einer Freigabe hinzugefügt, können die partizipierenden Objekte des Anwendungsfalls dann ebenfalls gleich dem Freigabedeskriptor hinzugefügt werden.

Entfernen von Freigabeplanungsentitäten

Die Operation entfernen() entfernt eine bestehende Freigabeplanungsentität von einer Freigabe, beispielsweise wenn sie vom Kunden nicht mehr gewünscht wird. Die Freigabeplanungsentität gehört nach dem Entfernen nicht mehr zu der Freigabe, von der sie entfernt wurde, kann aber noch zu anderen Freigaben gehören. Beim Entfernen einer Freigabeplanungsentität von einer Freigabe, verschwinden auch andere Artefakte aus den Modellen, die nach dem Entfernen der Freigabeplanungsentität nicht mehr benötigt werden. Partizipiert beispielsweise ein Objekt nur an einem Anwendungsfall, der entfernt werden soll, wird dieses Objekt ebenfalls von dem Freigabemodell entfernt. Wird ein partizipierendes Objekt jedoch noch von weiteren Anwendungsfällen der Freigabe benötigt, bleibt es im Freigabemodell erhalten.

Verschieben, Splitten, Verschmelzen

Mit Hilfe der Operationen hinzufügen() und entfernen() lassen sich die Operationen verschieben(), splitten() und verschmelzen() definieren, um die Arbeit eines Freigabemanagers zu unterstützen. Verschieben() verschiebt eine bestehende Freigabeplanungsentität von einer Freigabe f1 auf eine andere Freigabe f2, beispielweise wenn die Freigabeplanungsentität aus Zeitgründen nicht mehr umgesetzt werden kann. Die Operation besteht aus dem Entfernen der Freigabeplanungsentität aus einer Freigabe f1 und dem Hinzufügen zu einer Freigabe f2. Splitten() teilt eine Freigabe in zwei Freigaben auf, hierbei werden ausgewählte Freigabeplanungsentitäten entfernt und einer leeren Freigabe hinzugefügt. Das Splitten wird beispielsweise nötig, wenn der Kunde nach einer Demo eines Systems bereits zufrieden ist und dieses verwenden möchte. Die verbleibenden Freigabeplanungsentitäten können dann in einer folgenden Freigabe umgesetzt werden. Verschmelzen() vereint zwei Freigaben f1 und f2, das heißt die Freigabeplanungsentitäten der Freigabe f2 werden der Freigabe f1 hinzugefügt und in einer gemeinsamen Freigabe geliefert.

4 Begründungsbasiertes Freigabemodell

Alternative Implementierungen einer

Im Roadmapping versucht der Freigabemanager neue Merkmale für eine Freigabe so auszuwählen, dass unter Berücksichtigung von Beschränkungen die Bedürfnisse des Kunden bestmöglich erfüllt werden. Beschränkungen können Ressourcen, aber auch Abhängigkeiten zwischen den Systemmodellen sein. [Amandeep et al. 2004], [Carlshamre et al. 2001] Da eine Freigabe diese geplanten Merkmale auf unterschiedliche Weisen implementieren kann, ist es Aufgabe des Freigabemanagers zu entscheiden, wie die Implementierung einer Freigabe konkret aussehen soll.

Begründungsbasiertes Freigabemanagement

Das begründungsbasierte Freigabemanagement unterstützt den Freigabemanager, indem es Kommunikationsmodelle und Freigabedeskriptoren als Begründungen zur Entscheidung zwischen alternativen Freigabeszenarios heranzieht. Mit Hilfe mehrerer Freigabeszenarios kommt das begründungsbasierte Freigabemodell der Anforderung von Carlshamre [Carlshamre 2002] nach und erlaubt alternative Lösungsvorschläge für eine Freigabe zu erstellen, zu erkunden, zu vergleichen und gegeneinander abzuwiegen.

Kommunikationsmodelle

Kommunikationsmodelle umfassen Begründungsmodelle zur strukturierten Kommunikation, Kommentare zur informellen Diskussion sowie ein Aufgabenmodell zur Zuweisung von Arbeit. Kommunikationsmodelle unterstützen alle Aktivitäten der Systementwicklung, indem sie die Begründungen festhalten, die zu einer Freigabe führen. Begründungsmodelle, die Ergebnisse informeller Diskussion sowie Aufgabenmodelle dienen als Begründung für Entscheidungen bei Änderungen im Freigabemanagement.

Freigabedeskriptoren

Durch Freigabedeskriptoren können Analysemodelle, Systementwurfsmodelle und Objektentwurfsmodelle zur Entscheidungsfindung herangezogen werden. In Zusammenhang mit Freigabeszenarios erlauben sie die Auswirkungen von Änderungen im Modell zu erkunden und auszuprobieren. Somit können Freigabedeskriptoren bei der Auswahl eines konkreten Freigabeszenarios als Begründungen herangezogen werden.

4.1 Freigabeszenarios

Freigabeszenarios für den Multimedia Player

Freigabeszenarios stellen eine konkrete Möglichkeit dar, die Merkmale einer Freigabe umzusetzen. Im Beispiel des Multimedia Players sieht die strategische Planung TV-Shows als neues Merkmal für die zweite Freigabe vor. Zur konkreten Umsetzung der TV-Shows muss spezifiziert werden, wie diese empfangen werden können und ob der Multimedia Player TV-Shows speichern kann. Die Alternative „digitaler Empfang“ erlaubt den Empfang über DVB-H sowie die Speicherung der TV-Shows. Ihr steht als Alternative die Freigabe „traditioneller Empfang“ mit Empfang via Antenne und Satellit, jedoch ohne Speicherung gegenüber.

Verteiltes Freigabemanagement

Freigabeszenarios können das Freigabemanagement in verteilten Softwareprojekten unterstützen, beispielsweise wenn ein Produkt an weltweit verteilte Kunden geliefert wird. Dann können regionale oder länderspezifische Anforderungen, Beschränkungen, Interessen oder Änderungen zu berücksichtigen sein. Freigabeszenarios können das verteilte Freigabemanagement unterstützen, indem jeder Standort einen Vorschlag für eine Freigabe in Form eines Freigabeszenarios erarbeitet. In einem verteilten Treffen können die Interessenvertreter die Freigabeszenarios der verschiedenen Standorte vergleichen und eine gemeinsame Lösung erarbeiten.

Um solche alternativen Implementierungen zu erkunden und zu vergleichen sowie Gründe für getroffene Entscheidungen nachvollziehen zu können, integriert diese Dissertation Freigabeszenarios in das RUSE-Begründungsmodell.

4.1.1 Integration in das Begründungsmodell

Freigabe-Fragestellung

Abb. 32 zeigt die bestehenden Klassen Fragestellung (*issue*), Vorschlag (*proposal*), Kriterium (*criterion*) und Bewertung (*assessment*) des RUSE-Begründungsmodells, wie sie in Abschnitt 2.4 beschrieben wurden. Eine Freigabe hat einen speziellen *issue*, eine Freigabe-Fragestellung: „Welche Merkmale sollen in dieser Freigabe umgesetzt werden?“ Mehrere alternative Freigabeszenarios sind Vorschläge zur Lösung dieser Fragestellung und geben mögliche Antworten auf diese Fragestellung.

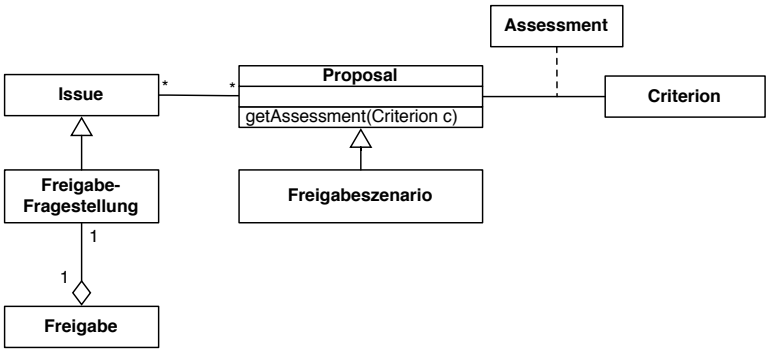


Abb. 32: Begründungsmodell für Entscheidungen im Freigabemanagement. Freigabeszenarios sind Vorschläge für die Implementierung einer Freigabe und können an Hand von Kriterien bewertet und verglichen werden.

Kriterien

Kriterien (*criterion*) sind wünschenswerte Qualitäten, die eine Freigabe haben sollte. Sie dienen dem Vergleich mehrerer Freigabeszenarios, indem die Freigabeszenarios bezüglich der Kriterien bewertet werden, wie gut sie diese erreichen. Der Vorschlag liefert mit der Methode getAssessment(Criterion c) seine Bewertung bezüglich eines Kriteriums. Kriterien können frei definiert werden und variieren je nach Projekt, Kunden oder zu betrachtendem System. Beispielsweise kann man die Alternativen für die zweite Freigabe „digitaler Empfang“ und „traditioneller Empfang“ bezüglich ihrer Innovativität, Benutzerfreundlichkeit, Mobilität bewerten.

Berechnete Kriterien

Um auch Ressourcenbedarf und das Erreichen von Geschäftszielen mit Metriken zu unterstützen, wie sie die traditionelle Freigabeplanung einsetzt, führt diese Dissertation berechnete Kriterien ein. Die Bewertung eines berechneten Kriteriums wird nicht direkt angegeben, sondern mit Hilfe einer

Formel und Parametern berechnet. Eine Formel ist eine Berechnungsvorschrift, die angibt, wie die einzelnen Bewertungen verrechnet werden sollen. Die Bewertungen erhält die Formel aus den Parametern, die wiederum Kriterien oder Formeln sein können. (siehe Abb. 33) Freigabeplanungsentitäten können direkt mit Kriterien bewertet und verglichen werden.

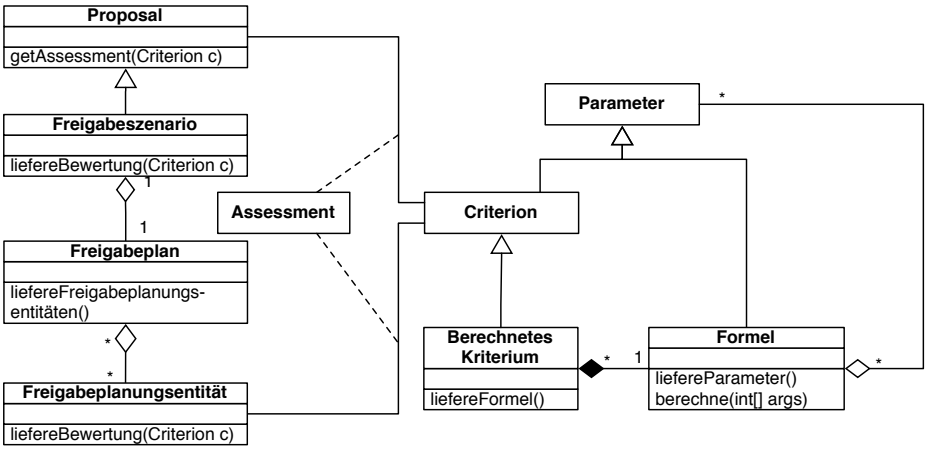


Abb. 33: Berechnete Kriterien zur Bewertung von alternativen Freigabeszenarios: Ein Freigabeszenario nützt bei der Auswertung der Bewertung eines berechneten Kriteriums die Bewertung der Freigabeplanungsentitäten bezüglich anderer Kriterien.

Formeln

Konkrete Formeln implementieren die Methode berechne(int[] args) von Formel. Abb. 34 gibt einen Überblick über Formeln: Die Summe addiert mehrere übergebene Argumente, die Division kann 2 Argumente teilen, die Differenz zieht vom ersten Argument weitere übergebene ab, die Multiplikation multipliziert die übergebenen Argumente. Minimum bzw. Maximum bestimmen den kleinsten bzw. größten Wert der übergebenen Argumente.

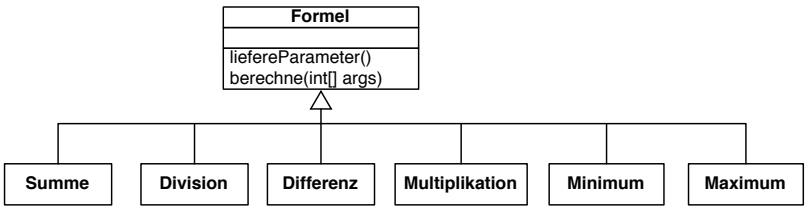


Abb. 34: Formeln im Modell zum begründungsbasierten Freigabemanagement

Vergleich 2er Freigabeszenarios des Multimedia Players

Beispielsweise können mit Hilfe berechneter Kriterien die alternativen Freigabeszenarios „digitaler Empfang“ und „traditioneller Empfang“ der zweiten Freigabe des Multimedia Players nach ihrem Gesamtaufwand in Mitarbeiterstunden verglichen werden. Dazu werden alle Freigabeplanungsentitäten der beiden Freigabeszenarios bezüglich der benötigten Mitarbeiterstunden bewertet. Jedes Freigabeszenario kann die Aufwände für seine Freigabeplanungsentitäten aufaddieren, wenn ein berechnetes Kriterium „Gesamtaufwand in Mitarbeiterstunden“ mit Summe als Formel und dem Kriterium Mitarbeiterstunden als Parameter definiert wird.

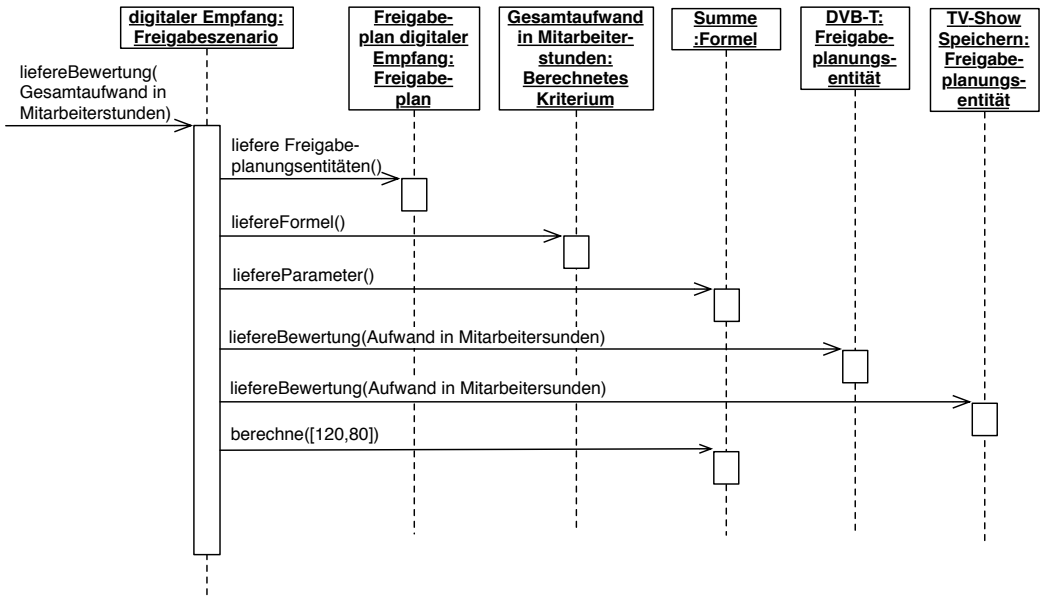


Abb. 35: Ermittlung der Bewertung eines berechneten Kriteriums: Das Freigabeszenario fragt die Bewertungen seiner Freigabeplanungsentitäten ab und lässt sie von einer Formel berechen.

Bewertung eines Freigabeszenarios

Zur Ermittlung der Bewertung eines berechneten Kriteriums wertet das Freigabeszenario in der Methode liefereBewertung(Criterion c) die Formel aus und holt sich die Bewertungen für alle seine Freigabeplanungsentitäten. Abb. 35 zeigt die Berechnung des Gesamtaufwands in Mitarbeiterstunden für das Freigabeszenario „digitaler Empfang“. Zunächst holt sich das Freigabeszenario von seinem Freigabeplan die neu umzusetzenden Freigabeplanungsentitäten DVB-T und TV-Show. Dann fragt es bei dem berechneten

Kriterium nach der Formel und erhält die Summe, welche den Aufwand in Mitarbeiterstunden als Parameter hat. Jetzt besorgt sich das Freigabeszenario von den beiden Freigabeplanungsentitäten mit liefereBewertung(Aufwand in Mitarbeiterstunden) die Bewertung, wie viele Mitarbeiterstunden für sie nötig sind. Schließlich kann die Formel diese Werte mit der Methode berechne(int[] args) summieren.

4.1.2 Freigabepläne und -deskriptoren

Freigabedeskriptoren von Freigabeszenarios

Ein Freigabeszenario besteht aus einem eigenen Freigabedeskriptor sowie einem Freigabeplan. Der Freigabedeskriptor beschreibt die Systemmodelle des Freigabeszenarios und dient - wie auch bei Freigaben - als Markierung für Knowledge Nuggets. Bei der Erstellung eines Freigabeszenarios wird zunächst ein neuer Freigabedeskriptor erzeugt, für den dann wie in Abschnitt 3.3 beschrieben eine Knowledge Nugget Fabrik die nötigen Knowledge Nuggets generiert. Diese Knowledge Nuggets erlauben gleichzeitig eigene Systemmodelle für alternative Freigabeszenarios auszuarbeiten und zu erkunden.

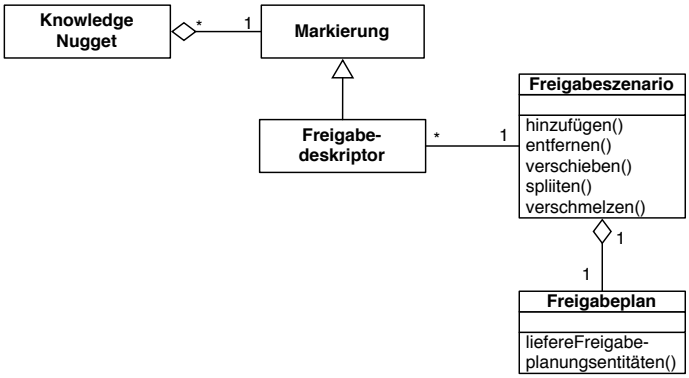


Abb. 36: Modell eines Freigabeszenarios: Ein Freigabeszenario hat einen Freigabedeskriptor und einen Freigabeplan.

Freigabeplan eines Freigabeszenarios

Der Freigabeplan eines Freigabeszenarios zeigt, welche Freigabeplanungsentitäten bei der Implementierung eines Freigabeszenarios neu erstellt werden müssen. Für die Ermittlung dieser Freigabeplanungsentitäten benötigt der Freigabeplan einen weiteren Freigabedeskriptor, zu dem er

einen Unterschied berechnet. Um einen zweiten Freigabedeskriptor zu finden, der den Zustand des Systems bei Beginn der Entwicklung des Freigabeszenarios beschreibt, verhält sich der Freigabeplan, wie wenn das Freigabeszenario eine Lösung der Freigabefragestellung wäre.

Operationen eines Freigabeszenarios

Zudem unterstützt ein Freigabeszenario wie die Freigabe auch die Operationen hinzufügen(), entfernen(), verschieben(), splitten() und verschmelzen(), um Freigabeszenarios zu modifizieren.

4.2 Begründungen aus der Kommunikation

Kommunikation

Kommunikation hat eine bedeutende Rolle in der Softwareentwicklung [Kraut & Streeter 1995]. Zum Beispiel finden während der Anforderungsanalyse Gespräche und Diskussionen zwischen Analysten und Kunden statt, um die Anwendungsdomäne oder Anforderungen zu verstehen. Im Systementwurf werden Konzepte der Lösungsdomäne, wie Architekturen, Frameworks oder Bibliotheken, diskutiert, gegeneinander abgewogen und dahingehend verglichen, wie sehr sie Entwurfsziele unterstützen.

Kommunikation bei Änderungen

Änderungen im Freigabemanagement sind eine kommunikationsintensive Aufgabe, denn es ist die Aufgabe des Freigabemanagers die Auswirkungen einer Änderung zu analysieren. Dies geschieht in Gesprächen mit betroffenen Interessenvertretern, wie Kunden, Analysten, Architekten oder Programmierern. Beispielsweise sehen agile Methoden Treffen vor, um die Auswirkungen von Änderungen auf die Erstellung der Freigabe zu analysieren.

Kommunikationsmodelle

Kommunikationsmodelle können Diskussionen und Ergebnisse von Gesprächen am Gang, per Telefon, Emailverkehr, Chats oder Diskussionsforen darstellen. Kommunikationsmodelle erlauben damit Gründe für Anforderungen oder getroffene Entscheidungen nachzuvollziehen und dienen als Begründungen bei Entscheidungen über Änderungen im Freigabemanagement. [Wolf 2007]

Annotation

Das hier vorgestellte Modell zum begründungsbasierten Freigabemanagement stimmt mit dem in [Wolf 2007] beschriebenen RUSE-Modell überein: Systemmodellelemente können durch Kommunikationsmodellelemente annotiert werden und bilden dadurch den Kontext für die Kommunikation.

Ein Kommunikationsmodellelement bildet die Kommunikation über ein bestimmtes Systemmodellelement ab. Kommunikationsmodellelemente werden direkt mit den Systemmodellelementen assoziiert, auf die sich die Kommunikation bezieht, und sind dadurch bei Änderungen des betreffenden Systemmodellelements verfügbar. Abb. 37 zeigt die Annotation von Systemmodellelementen durch die Kommunikationsmodellelemente Kommentar, Fragestellungen, Risiko, Fehlerberichte und Arbeitseinheiten.

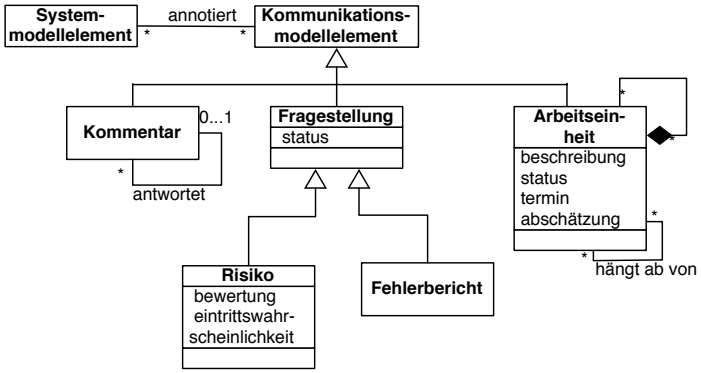


Abb. 37: Kommunikationsmodellelemente annotieren Systemmodellelemente und stehen damit als Begründungen im Freigabemanagement zur Verfügung. In Anlehnung an[Wolf 2007].

Kommentare

Kommentare ermöglichen unstrukturierte Diskussion *„to rapidly exchange knowledge and to clarify arising problems“* [Wolf 2007]. Projektteilnehmer können Kommentare für Systemmodelle erstellen oder auf bestehende Kommentare antworten. (siehe Abb. 37) Kommentare bilden Diskussionen ab, wie sie per Email, in Diskussionsforen oder in Gesprächen geführt werden können und enthalten Begründungen für das annotierte Systemmodellelement. Bei Änderungen in einer Freigabe kann der Freigabemanager Kommentare betroffener Systemmodellelemente überprüfen und feststellen, ob und wie diese Systemmodellelemente an die Änderung angepasst werden sollten.

Fragestellungen

Fragestellungen können strukturierte Diskussionen abbilden, indem sie die Inhalte einer geführten Diskussion nach alternativen Vorschlägen und Kriterien oder Argumenten für und gegen diese Vorschläge systematisch gliedern. Sie bilden den Ausgangspunkt für Begründungsmodelle, wie sie bereits in

Abschnitt 2.4 vorgestellt worden sind. Eine Fragestellung hat einen Status, der offen oder geschlossen sein kann. Offene Fragestellungen müssen noch durch weitere Diskussionen oder Verhandlungen geklärt werden, sie haben noch keine Lösung. [Wolf 2007] Offene Fragestellungen sind also aktuelle Probleme, die noch nicht gelöst sind und geben Hinweise, welche Teile des Systems nicht rechtzeitig in der gewünschten Qualität fertig gestellt werden und somit die Lieferung einer Freigabe in Gefahr bringen könnten. Im Gegensatz dazu sind geschlossene Fragestellungen gelöst und die festgehaltenen Argumente beschreiben die Begründung für die Entscheidung. Risiken und Fehlerberichte sind zwei spezielle Fragestellungen, die für das Freigabemanagement von Bedeutung sind. [Carlshamre 2002]

Risiken

Risiken sind Fragestellungen, die während der Entwicklung möglicherweise auftreten könnten. Ein Beispiel eines Risikos ist, dass eine für die Freigabe benötigte Hardware-Komponente eines anderen Herstellers nicht geliefert werden kann. Das Risikomanagement identifiziert Risiken, um diese zu vermeiden oder ihnen entgegenzuwirken. Daher ist der Status eines Risikos bei seiner Instanzierung noch geschlossen; erst bei Eintritt des Risikos wird das Risiko dann geöffnet. Risiken werden bei ihrer Identifizierung nach möglichen Auswirkungen und ihrer Eintrittswahrscheinlichkeit bewertet. Risiken mit hoher Eintrittswahrscheinlichkeit und großen Auswirkungen sind besonders kritisch und sollten vom Freigabemanager regelmäßig überwacht werden.

Fehlerberichte

Fehlerberichte sind Fragestellungen, die aus dem Testen resultieren und angeben, dass das Verhalten des Systems von der Spezifikation abweicht. Fehler können für die Herausgabe eines Systems kritisch sein, wenn sie Kernfunktionalität betreffen. Deshalb dienen Fehler als Begründungen, beispielsweise bei der Priorisierung von Arbeitseinheiten: Gerade bei kurzen Freigabezyklen sind kritische Fehler schnell zu lösen, um eine Freigabe termingerecht liefern zu können. Der Zustand eines Fehlers ist bei seiner Identifizierung offen, bis er durch geeignete Maßnahmen, beispielsweise die Fehlerbehebung, geschlossen wird.

Arbeitseinheiten

Arbeitseinheiten dienen dazu, das Projekt aktivitätsorientiert zu koordinieren und teilen Projektteilnehmern mit, was zu erledigen ist. Eine Aufgabe hat

eine Beschreibung, einen Status, einen Termin, bis zu dem sie erledigt sein muss, sowie eine Abschätzung für den Aufwand. Eine Arbeitseinheit kann von anderen Arbeitseinheiten abhängen, das heißt diese Arbeitseinheiten müssen zunächst erledigt sein, bevor die neue Arbeitseinheit endgültig erledigt werden kann. Zudem können Arbeitseinheiten hierarchisch in kleinere Teilaufgaben zerlegt werden. (siehe Abb. 37) [Wolf 2007] Offene Arbeitseinheiten können anzeigen, dass eine Systemkomponente noch nicht fertig gestellt ist, was bedeuten kann, dass eine Freigabe noch nicht zusammengestellt werden kann oder sie noch nicht allen Anforderungen entspricht. Auch dann kann es sinnvoll sein, eine Freigabe, beispielsweise für eine Demo oder einen Test mit Anwendern zu machen, wenn die zu demonstrierende Funktionalität bereits vertikal zur Verfügung steht oder Stubs noch nicht implementierte Teile des Systems simulieren.

4.3 Begründungen aus der Organisation

Organisation

Der Erfolg einer Freigabe hängt von den beteiligten Mitarbeitern und deren Leistungen ab. Daher sollte der Freigabemanager bei seinen Entscheidungen die Verfügbarkeit, Wissen und Interessen der Mitarbeiter sowie deren Zusammensetzung in Teams berücksichtigen.

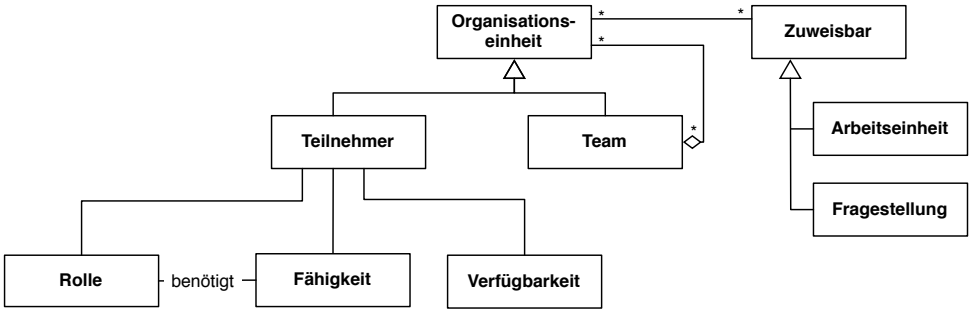


Abb. 38: Das Organisationsmodell stellt Teams zusammen mit ihren Teilnehmern, Fähigkeiten, Verfügbarkeiten und Interessen als Begründungen für das Freigabemanagement bereit. In Anlehnung an [Wolf 2007].

Fähigkeiten und Verfügbarkeit

BEEF erweitert das RUSE - Organisationsmodell [Wolf 2007] um Rollen, Fähigkeiten und Verfügbarkeiten (siehe Abb. Abb. 38). Organisationseinheiten, das heißt Teilnehmer oder Teams, sind Teile einer Organisation, die Verant-

wortungen im Projekt übernehmen. Teilnehmer können mehreren Teams angehören, die wiederum aus Organisationseinheiten bestehen. Teilnehmer haben eine beschränkte Verfügbarkeit und üben Rollen aus, für die sie bestimmte Fähigkeiten benötigen. Die Verfügbarkeit von Teilnehmern mit entsprechenden Fähigkeiten ist eine beschränkende Ressource bei der Erstellung einer Freigabe (siehe 3.2).

Zuweisbar

Organisationseinheiten erhalten Verantwortungen über die Schnittstelle Zuweisbar, die beispielsweise Arbeitseinheiten und Fragestellungen implementieren. Dieses Organisationsmodell erlaubt das Nachvollziehen von Änderungen, die von Teilnehmern ausgehen, über Arbeitseinheiten zu Systemmodellelementen. Fällt beispielsweise ein Mitarbeiter aus, kann der Freigabemanager alle betroffenen Anwendungsfälle oder Komponenten einer Freigabe ermitteln.

4.4 Freigabedeskriptoren

Modelle als Begründung

Begründungen für eine Entscheidung im Freigabemanagement hängen von der Sichtweise involvierter Interessenvertreter ab [Amandeep et al. 2004]. Beispielsweise interessieren sich Marketing und Vertrieb vor allem für wirtschaftliche Ziele, die eine Freigabe erreichen kann. Software-Architekten beurteilen Änderungen nach den Auswirkungen auf die Software-Architektur. Software-Entwickler fokussieren detaillierte Objektmodelle und die Implementierung. Während Planungsmethoden wie EVOLVE [Saliu & Ruhe 2005b] oder *„A Cost-Value Approach for Prioritizing Requirements“* [Karlsson & Ryan 1997] heute die Berücksichtigung wirtschaftlicher Faktoren bei der Auswahl von Merkmalen für eine Freigabe erlauben, bleiben Software-Architekturen in Entscheidungen weitgehend unberücksichtigt. So stellt Lehtola [Lehtola 2007] eine unzureichende Verbindung zwischen der Produktplanung und Softwareentwicklung fest. Vähäniitty [Vähäniitty et al. 2002] verbindet die Planung von Freigaben mit Komponenten der Systemarchitektur.

Freigabedeskriptor

Um die Verbindung zwischen Produktplanung, Modellen der Anforderungen und Systemarchitekturen und implementierungsnahen Modellen zu verbessern, führt diese Dissertation Freigabedeskriptoren ein. Freigabedeskripto-

ren sind eine Markierung für Knowledge Nuggets und stehen für das Systemmodell einer Freigabe. Die Systemmodellelemente, die das mit einer Freigabe zu liefernde System beschreiben, enthält dann der Knowledge Nugget (siehe Abb. 39).

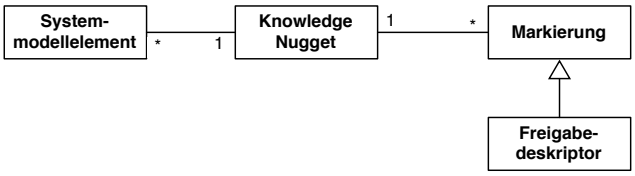


Abb. 39: Freigabedeskriptoren sind Markierungen für Knowledge Nuggets, um das Systemmodell einer Freigabe zu beschreiben.

Abstraktionsgrade

Abstraktionsgrade als zweite Markierung neben Freigabedeskriptoren geben an, wie viel Detail das Systemmodell eines Knowledge Nuggets enthält. Abstraktionsgrade blenden Details aus, die entweder unbekannt sind oder um bestimmte Aspekte zu betonen, und erlauben so den Übergang von vagen Anforderungen zu detaillierten, implementierungsnahen Modellen für aufeinanderfolgende Freigaben.

Systemmodell eines Abstraktionsgrads als Begründung

Die Beschreibungen der Systemmodelle eines Abstraktionsgrads für jede Freigabe dienen als Begründungen für Entscheidungen, wenn das Modell eines bestimmten Abstraktionsgrades die Sichtweise eines Interessenvertreters reflektiert. Beispielsweise enthält das Analysemodell eines Freigabedeskriptors nur diejenigen Anforderungen, die auch in dieser Freigabe implementiert werden sollen, und zwar ohne zu berücksichtigen, wie das System später erweitert werden könnte. Das Analysemodell eines Freigabedeskriptors blendet zum einen Details der Implementierung, zum anderen mögliche weitere Anforderungen aus, die in späteren Freigaben implementiert werden könnten; dies trifft auch dann zu, wenn Implementierungsdetails oder weitere Anforderungen bereits bekannt sind. Dieses Ausblenden von Details hält die Systemmodelle möglichst einfach und unterstützt so beispielsweise die Entwicklung mit *Extreme Programming. Extreme Programming* schlägt vor, die einfachste und schlankste Architektur für ein System zu wählen, die die benötigten *user stories* einer Freigabe umsetzt. Durch inkrementelle Verfeinerung fügt man in *Extreme Programming* der Systemarchi-

tektur schrittweise neue Klassen oder Entwurfsmuster hinzu, um neue *user stories* im System zu implementieren. [Cohn 2005], [Beck 2001]

Wahl von Abstraktionsgraden

Das Modell zum begründungsbasierten Freigabemanagement lässt offen, welche Abstraktionsgrade für die Modellierung einer Freigabe erzeugt werden. Die Lieferung des Modells auf einem bestimmten Abstraktionsgrad kann eine Pseudoanforderung des Kunden sein. Die konkrete Wahl hängt von der Art und Größe des zu entwickelnden Systems und dem gewählten Vorgehensmodell ab. Für kleine Systeme kann beispielsweise die Entwicklung eines Analysemodells und eines detaillierten Modells ausreichen, wenn Systementwurfsmodell und Objektentwurfsmodell ähnlich sind. Für große Systeme hingegen kann es sinnvoll sein vier Abstraktionsgrade zu unterscheiden, beispielsweise um die Ergebnisse der Anforderungsermittlung, Analyse, des Systementwurfs und detaillierten Entwurfs explizit in einem eigenen Modell festzuhalten. Häufig werden Analysemodell, Systementwurfsmodell und ein detailliertes Entwurfsmodell als Abstraktionsgrade eingesetzt, da sie das Ergebnis von Phasen von Vorgehensmodellen darstellen. [Jacobsen et al. 1998]

Systemmodellelemente im Modell eines Abstraktionsgrads

Die Assoziation zwischen Knowledge Nuggets und Systemmodellelement erlaubt dem Modell eines bestimmten Abstraktionsgrads beliebige Systemmodellelemente zuzuordnen. Es obliegt damit dem Freigabemanager und Systementwickler zu entscheiden, inwiefern es in einem Projekt sinnvoll ist, ein Systemmodellelement dem Modell mit einem bestimmten Abstraktionsgrad zuzuweisen. Beispielsweise sind Verträge ein Konzept des detaillierten Entwurfs, um Schnittstellen zu spezifizieren. Sie können jedoch bereits in der Analyse eingesetzt werden, wenn Kunden formale Anforderungen an bestimmte Teile eines Systems stellen.

Abhängigkeiten der Systemmodellelemente

Die folgenden Abschnitte stellen die Systemmodelle eines Freigabedeskriptors für aufeinanderfolgende Freigaben zusammen mit ihren Abhängigkeiten dar. Ihre Beschreibung ist nach den Abstraktionsgraden Analysemodell (4.4.1), Systementwurfsmodell (4.4.2) und Objektentwurfsmodell (siehe 4.4.3) gegliedert. Da Abhängigkeiten nicht nur zwischen den Systemmodellen eines Knowledge Nuggets bestehen, sondern sich auch über mehrere Know-

ledge Nuggets erstrecken können, betrachtet Abschnitt 4.4.4 Abhängigkeiten zwischen den Systemmodellen verschiedener Knowledge Nuggets.

4.4.1 Analysemodell

Analyse-modell

Das Analysemodell unterstützt die Ermittlung der Anforderungen in Gesprächen mit dem Kunden und enthält Entitäten zur Beschreibung des Systems aus Sicht des Kunden. Abb. 41 zeigt die Entitäten der Anforderungsermittlung und –analyse, die ein Systemmodell des Abstraktionsgrads Analyse enthalten könnte.

<action> the <result> <by|for|of|to>a(n)<object>

Abb. 40: Vorlage zur Beschreibung eines Merkmals. Beispielsweise „Berechne den Gesamtwert eines Einkaufs“ ein Merkmal. [Coad et al. 1999]

Merkmale

Merkmale [Coad et al. 1999], [Palmer & Felsing 2002] sind kurze prägnante Beschreibungen einer Eigenschaft des Systems, die einen Überblick über die Anforderungen und die Funktionalität geben. Die Beschreibung folgt der

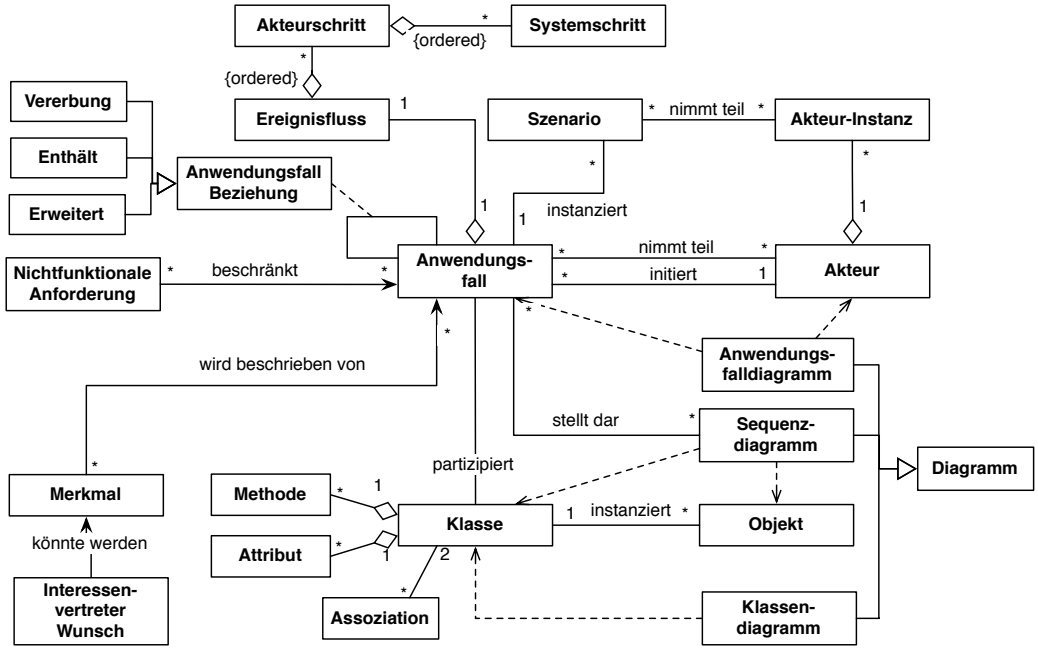


Abb. 41: Systemmodellelemente zur Anforderungsermittlung und Analyse. In Anlehnung an [Wolf 2007]

Vorlage aus Abb. 40 und ist meist nicht länger als eine Zeile. Der Ursprung von Merkmalen liegt in Wünschen von Interessenvertretern, beispielsweise der Kunden. Merkmale beschreiben das System abstrakt, während nichtfunktionale Anforderungen und Anwendungsfälle Details der beschriebenen Merkmale klären.

Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen sind Einschränkungen für das System, die für Kunden sichtbar sind. Beispiele für nichtfunktionale Anforderungen sind Anforderungen an die Benutzerfreundlichkeit, Sicherheit oder die Leistung. Nichtfunktionale Anforderungen beschränken Anwendungsfälle, indem sie Qualitäten für die Schritte des Systems definieren. Schließlich dienen nichtfunktionale Anforderungen als Ausgangspunkt für die Identifizierung von Entwurfszielen.

Anwendungsfälle

Anwendungsfälle beschreiben Merkmale, indem sie konkrete Interaktionen zwischen dem System und Akteuren beschreiben. Akteure dienen dazu die Systemgrenzen zu definieren: Sie sind Entitäten, die außerhalb des Systems liegen und mit dem System interagieren. Die Interaktion der Akteure mit dem System beschreibt der Ereignisfluss eines Anwendungsfalls. Der Ereignisfluss besteht aus Akteurschritten, die Aktionen der Akteure beschreiben. Einem Akteurschritt folgt eine Menge von Systemschritten, das heißt Reaktionen des Systems auf den Akteurschritt.

Beziehungen zwischen Anwendungsfällen

Zwischen Anwendungsfällen bestehen Erweitert-, Enthält- und Vererbungs-Beziehungen. Erweitert-Beziehungen dienen der Beschreibung von außergewöhnlichem oder optionalem Verhalten. Enthält-Beziehungen reduzieren Redundanzen, indem sie Teile des Ereignisflusses, den mehrere Anwendungsfälle benutzen, in eigene Anwendungsfälle auslagern. Durch die Vererbungs-Beziehung kann ein Anwendungsfall einen allgemeineren Fall durch das Hinzufügen von Einzelheiten spezialisieren. [Bruegge & Dutoit 2003]

Szenarios

Anwendungsfälle beschreiben mögliche Interaktionen zwischen einem Akteur und dem System in Form allgemeiner Ereignisse. Konkrete Ereignisse aus dem wirklichen Leben stellen Szenarios dar. Sie instanzieren Anwendungsfälle, um den konkreten Einsatz des Systems zu beschreiben. Ein Szenario zeigt folglich wie die Instanz eines Akteurs mit dem System interagieren könnte. [Bruegge & Dutoit 2003]

Klassen in der Analyse

Für Anwendungsfälle lassen sich, beispielsweise durch die natürliche Sprachanalyse von Abbot [Abbott 1983] partizipierende Klassen identifizieren. Partizipierende Klassen sind Klassen, die nötig sind, um den Ereignisfluss des Anwendungsfalls umzusetzen. Klassen in der Analyse repräsentieren Entitäten der Anwendungsdomäne und erstellen ein statisches Modell. Attribute beschreiben Eigenschaften von Klassen, während Methoden ihr Verhalten spezifizieren. Zudem bilden Assoziationen Beziehungen zwischen Klassen ab. Erfasst man Klassenmodelle und deren Beziehungen zu Anwendungsfällen, können bei Änderungen in den Merkmalen, betroffene Objekte der Analyse ermittelt werden oder bei Änderungen in den Objekten betroffene Merkmale bestimmt werden.

Diagramme

UML-Diagramme dienen der Visualisierung der Modelle: Anwendungsfalldiagramme zeigen Akteure, Anwendungsfälle und deren Beziehungen untereinander. Klassendiagramme zeigen die Klassen sowie ihre Assoziationen. Sequenzdiagramme zeigen die Interaktion von Objekten oder Klassen und können so den Ereignisfluss eines Szenarios oder Anwendungsfalls darstellen. [Wolf 2007], [Berenbach & Wolf 2007]

4.4.2 Systementwurfsmodell

Subsysteme

Während des Systementwurfs werden die gefundenen Analyseobjekte in kleinere Gruppen, sogenannte Subsysteme, zusammengefasst und Entscheidungen für die Architektur getroffen, beispielsweise indem Architekturmuster, wie eine Client - Server Architektur, eingesetzt werden. Im Systement wurf besteht ein System also aus einer Menge von Subsystemen, die das System in überschaubare Einheiten gliedern und große Systeme handhabbar machen, weil Subsysteme einzelnen Teams zugeordnet werden können. Durch die Dekomposition des Systems in Subsysteme und die Anwendung von Architekturmustern entwickelt sich aus dem Analysemodell ein Systementwurfsmodell, welches zusätzlich Modelle der Lösungsdomäne enthält und daher die Zusammenarbeit mit den Software-Architekten fördert. [Bruegge & Dutoit 2003]

Klassendiagramme

Klassendiagramme im Systementwurfsmodell enthalten die Dekomposition des Systems in Subsysteme, das heißt Subsysteme und ihre Klassen. Abb. 42 modelliert die Dekomposition in Subsysteme mit dem Kompositum Muster:

Subsystemelement beschreibt eine Entität, die Teil eines Subsystems sein kann. Ein Subsystem besteht aus mehreren Subsystemelementen, das heißt weiteren Subsystemen oder Klassen.

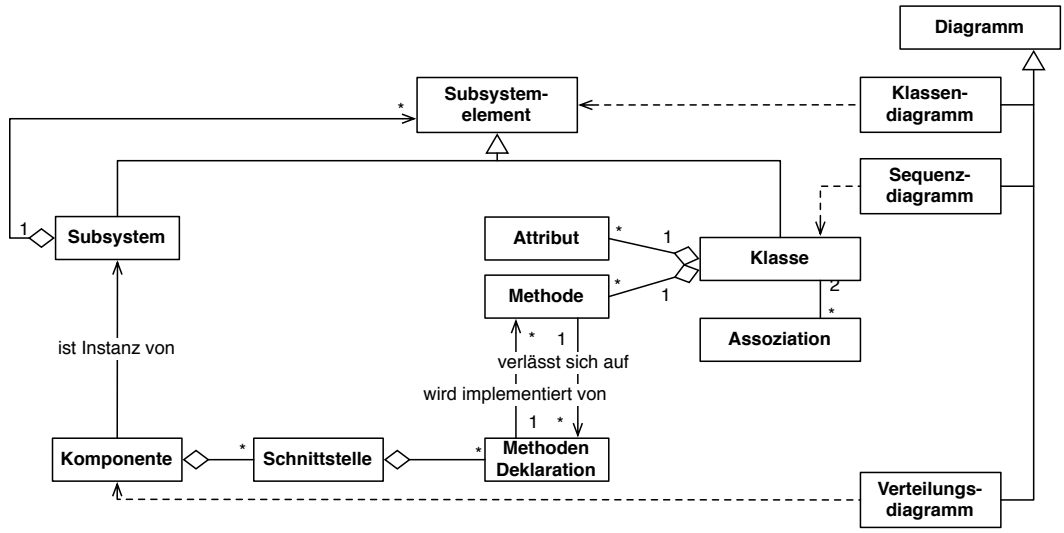


Abb. 42: Systemmodellelemente zur Unterstützung des Systementwurfs

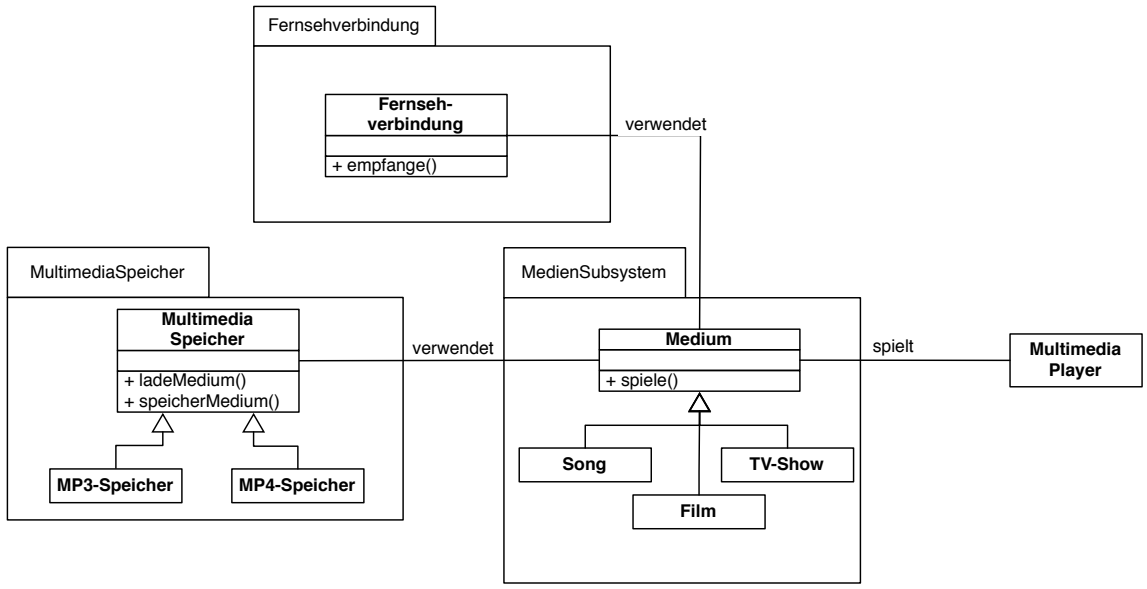


Abb. 43: Subsystem Dekomposition für den Multimedia Player: Die Klassen werden auf die Subsysteme Fernsehverbindung, Multimedia Speicher und Medien Subsystem verteilt.

Abhängigkeiten von Subsystemen

Durch den Aufruf von Methoden ergeben sich Abhängigkeiten zwischen Subsystemen. Ruft die Methode einer Klasse in Subsystem A eine Methode auf, die eine Klasse eines anderen Subsystems B anbietet, hängt das Subsystem A vom Subsystem B ab. Diese Abhängigkeit verdeutlicht beispielsweise die Subsystemdekomposition des Multimedia Players bestehend aus den Subsystemen Fernsehverbindung, Medien Subsystem sowie Multimedia Speicher, wie sie in Abb. 43 dargestellt ist: Das Medien Subsystem ist abhängig vom Fernsehverbindung Subsystem, weil die Methode spiele() der Klasse TV-Show die Methode empfange() der Klasse Fernsehverbindung aufruft.

Komponenten

Abhängigkeiten können die Grenzen des zu entwickelnden Systems überschreiten, wenn fremde Komponenten verwendet werden. Eine Komponente ist eine Instanz eines Subsystems zur Laufzeit. Über eine Schnittstelle, die mehrere Methodendeklarationen enthält, bietet sie die Dienste des instanzierten Subsystems anderen Komponenten an. Die Klassen dieses instanzierten Subsystems implementieren in öffentlichen Operationen die Schnittstelle der Komponente. Methoden anderer Komponenten können diese Schnittstelle aufrufen, wodurch die aufrufende Komponente von der anbietenden abhängig wird. Diese Abhängigkeiten müssen überprüft werden, bevor eine neue Version einer Komponente, beispielsweise einer Klassenbibliothek, verwendet wird, weil sie über geänderte Schnittstellen verfügen oder sich das Verhalten verändern kann. Denn als Folge solcher Änderungen können die Komponenten eines Systems nicht mehr zusammenpassen, wodurch Fehler auftreten. Bei einer Änderung der Schnittstelle wird der Code nicht mehr kompilieren und das System kann nicht mehr zusammengebaut werden. Wurde das Verhalten der Schnittstelle geändert, wird es zu semantischen Fehlern kommen, welche unter Umständen schwer zu identifizieren sind. [Bays 1999], [Sonatype 2008]

Verteilungsdiagramme

Gerade für die Systemintegration ist es wichtig, zu wissen, aus welchen Komponenten ein System besteht und welche Abhängigkeiten bestehen. Verteilungsdiagramme helfen hier, indem sie die Komponenten und ihre Abhängigkeiten darstellen. Sequenzdiagramme zeigen Abhängigkeiten, indem sie den globalen Kontrollfluss des Systems durch Klassen und den auf ihnen aufgerufenen Operationen, darstellen.

Randbedingungen

Anwendungsfälle im Systementwurf dienen der Beschreibung der Randbedingungen, also des Systemverhaltens beim Starten und Herunterfahren des Systems sowie administrativer Funktionalität. Verteilungsdiagramme zeigen die statische Struktur der Hardware- und Software- Komponenten für das laufende System, während Sequenzdiagramme den globalen Kontrollfluss und somit die dynamischen Aspekte des Systems beschreiben. [Bruegge & Dutoit 2003]

4.4.3 Objektentwurfsmodell

Objektentwurfsmodell

Das Objektentwurfsmodell schließt die Lücke zwischen Objekten der Anwendungsdomäne und Lösungsdomäne, indem es neue Objekte identifiziert und bestehende genauer spezifiziert. Es erlaubt Details und Probleme bei der Implementierung mit Entwicklern zu analysieren und diskutieren. (siehe Abb. 44) [Bruegge & Dutoit 2003]

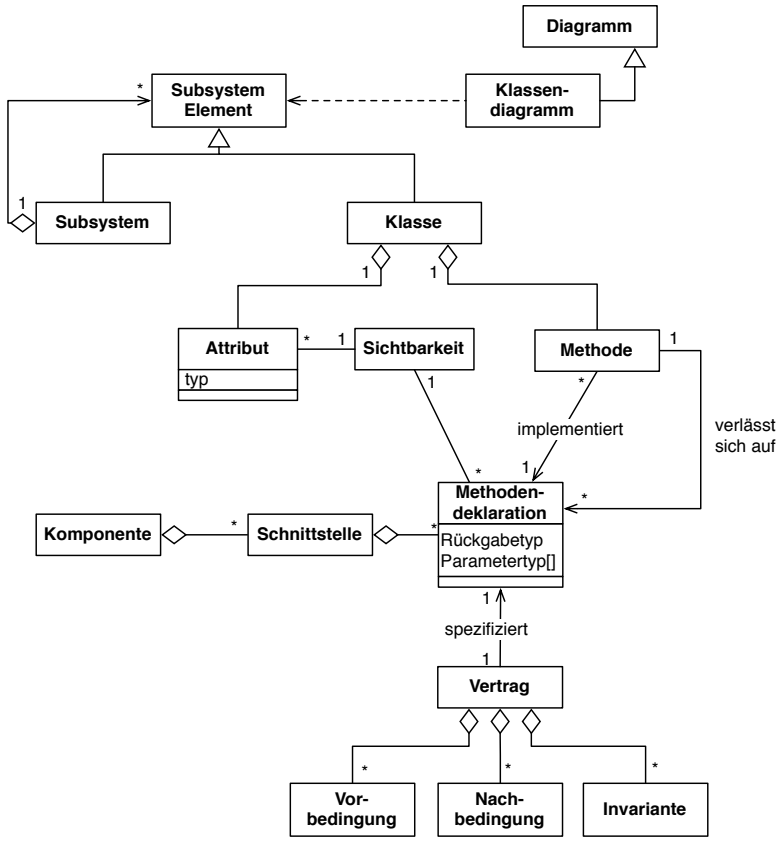


Abb. 44: Systemmodellelemente zur Unterstützung des Objektentwurfs

Klassen der Lösungsdomäne

Das Objektentwurfsmodell beschreibt die Klassen so detailliert, wie sie in Code umgesetzt werden. Der Objektentwurf entwickelt die Gliederung des Systems in Subsysteme in Bezug auf die Lösungsdomäne weiter; beispielsweise entspricht in Java [Gosling et al. 2000] ein Subsystem aus dem Objektentwurf einem Java-Package. Die Subsysteme können jetzt um weitere Klassen aus der Lösungsdomäne erweitert werden, beispielsweise durch den Einsatz von Entwurfsmustern, Standardkomponenten oder um die Performanz zu erhöhen.

Attribute und Methoden

Für alle Klassen kann der Objektentwurf spezifizieren, wie sie implementiert werden sollen, das heißt für sie werden alle Attribute und Methoden bestimmt und spezifiziert. Während man für Attribute Typen und Sichtbarkeit angibt, werden Operationen durch Methodendeklaration spezifiziert. Methodendeklarationen umfassen die Sichtbarkeit, Rückgabetypen und die Typen der Eingabeparameter. Zusätzlich kann ein Vertrag [Mitchell & McKim 2002] bestehend aus Vorbedingungen, Nachbedingungen und Invarianten das Verhalten spezifizieren. Vorbedingungen müssen vor, Nachbedingungen nach der Ausführung einer Methode gelten, während Invarianten Bedingungen angeben, die während der gesamten Ausführung der Methode gelten müssen. [Mitchell & McKim 2002]

Abhängigkeiten von Komponenten

Implementierungen von Methoden verlassen sich auf Methodendeklarationen, die zur Schnittstelle einer Standardkomponente gehören können; so können im Objektentwurf Abhängigkeiten von Komponenten definiert werden.

4.4.4 Abhängigkeiten zwischen den Systemmodellen verschiedener Knowledge Nuggets

Die vorangegangenen Abschnitte betrachten Abhängigkeiten innerhalb des Systemmodells auf einem Abstraktionsgrad, wie sie jeweils ein Knowledge Nugget beschreiben kann. Jedoch bestehen auch Abhängigkeiten zwischen den Systemmodellen zweier Knowledge Nuggets, beispielsweise zwischen dem Analysemodell und dem Systementwurfsmodell. Bei der Modellierung dieser Abhängigkeiten unterscheidet diese Dissertation Abhängigkeiten zwischen verschiedenen und identischen Entitätsklassen.

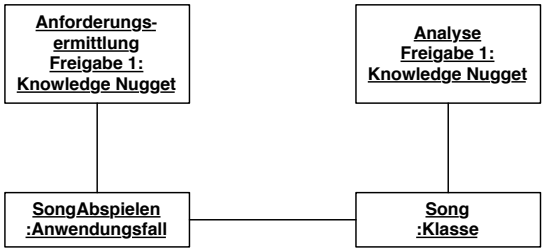


Abb. 45: Die Klasse Song ist eine partizipierende Klasse für den Anwendungsfall Song Abspielen. Die Assoziation beschreibt eine Abhängigkeit zwischen den Systemmodellen zweier Knowledge Nuggets.

Abhängigkeiten zwischen verschiedenen Entitätsklassen bilden die Modelle aus oben stehenden Abschnitten bereits ab. Beispielsweise können wie in Abb. 45 auch zwei Knowledge Nuggets für die Systemmodelle der Anforde rungsermittlung und der Analyse eingesetzt werden, wobei das

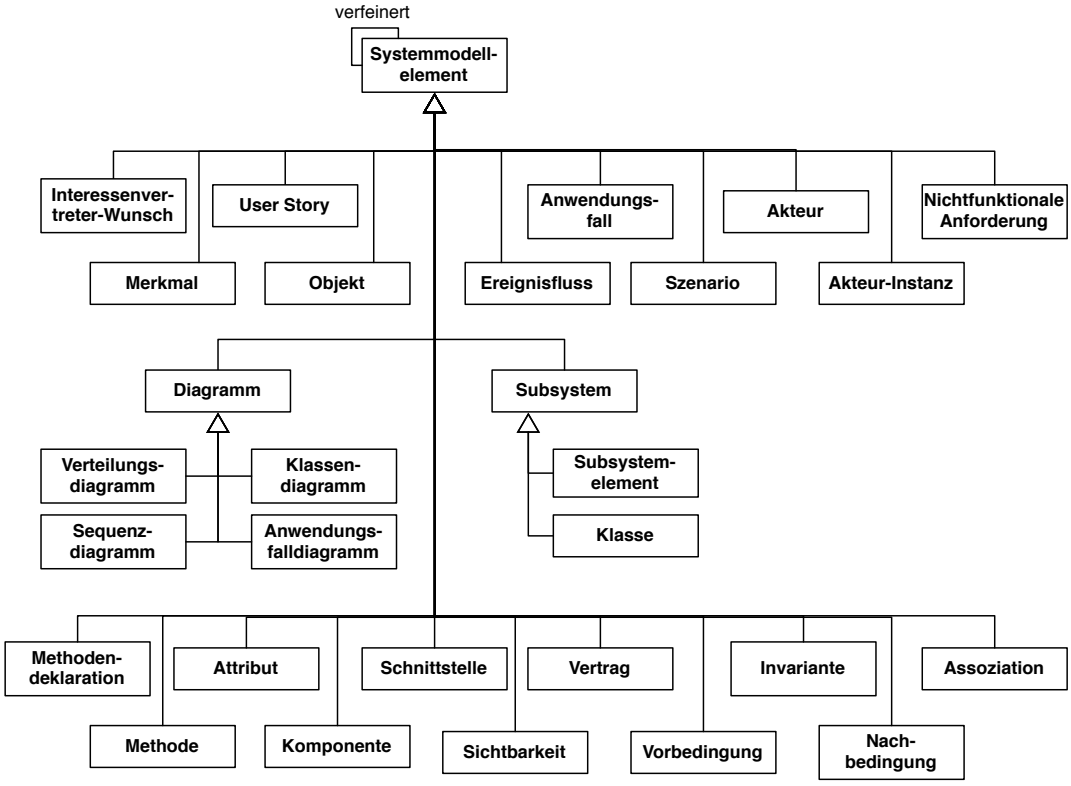


Abb. 46: Verfeinerung von Systemmodellelementen.

Abhängigkeiten zw. verschiedenen Entitätsklassen

Systemmodell der Anforderungsermittlung die Anwendungsfälle enthält und das Systemmodell der Analyse das Analyseobjektmodell. Die partizipiert-Assoziation zwischen Anwendungsfall und Klasse beschreibt dann eine Abhängigkeit zwischen den Systemmodellelementen, die zu zwei verschiedenen Knowledge Nuggets gehören. In Abb. 45 hängen der Anwendungsfall SongAbspielen und die Klasse Song voneinander ab.

Abhängigkeiten zw. gleichen Entitätsklassen

Abhängigkeiten zwischen Knowledge Nuggets für gleiche Klassen von Systemmodellen erlaubt eine Verfeinert-Assoziation zwischen Systemmodellelementen (siehe Abb. 46). Durch die Verfeinert-Assoziation kann das gleiche Systemmodellelement in zwei Knowledge Nuggets vorkommen und eine andere Beschreibung haben. Die Verfeinert-Assoziation unterstützt die Modellierungstechnik der inkrementellen Verfeinerung (siehe Abschnitt 2.3) und erlaubt Verfeinerungen zwischen Abstraktionsgraden sowie Freigaben.

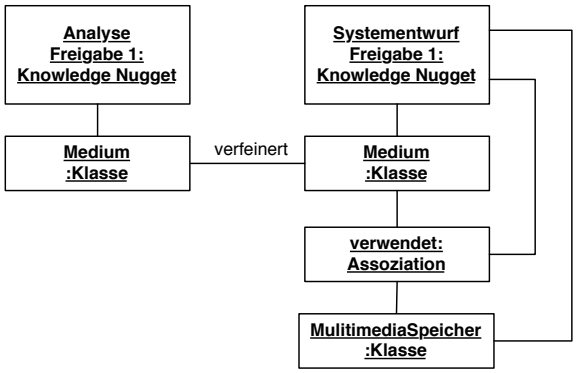


Abb. 47: Verfeinerung der Klasse Medium vom Analysemodell zum Systementwurfsmodell

Verfeinert-Assoziation für Abstraktionsgrade

Beispielsweise verwenden die Analyse, der Systementwurf und der detaillierte Entwurf Klassendiagramme um die Struktur des Systems zu beschreiben. Die Klassen entwickeln sich von der Analyse über den Systementwurf weiter zum Objektentwurf. Abb. 47 zeigt wie die Klasse Medium aus dem Analysemodell im Systementwurf verfeinert wird: Sie erhält eine Assoziation „verwendet“ zur Klasse Multimedia Speicher. Durch die Verfeinert-Assoziation sind Änderungen an Systemmodellelementen über Abstraktionsgrade hinweg nachvollziehbar, beispielsweise um die Auswirkungen der Änderungen einer Klasse des Analysemodells auf das Systementwurfsmodell zu beur-

teilen. Ebenso sind für eine Klasse des Objektentwurfs die entsprechenden Klassen des Analysemodells und des Systementwurfs leicht zu ermitteln.

Verfeinert-Assoziation für Freigaben

Knowledge Nuggets beschreiben die Systemmodelle aufeinanderfolgender Freigaben. Die Verfeinert-Assoziation unterstützt die inkrementelle Verfeinerung der Systemmodelle dieser Freigaben, denn sie erlauben, Änderungen seit der letzten Freigabe zu verfolgen. In Folge von Weiterentwicklung und Änderungen werden die Systemmodelle aufeinanderfolgender Freigaben angepasst. Unterstützt eine Freigabe neue, erweiterte oder geänderte Anforderungen als die ihr in der Roadmap vorangehende Freigabe, verfeinert ihr Systemmodell das der vorangehenden Freigabe.

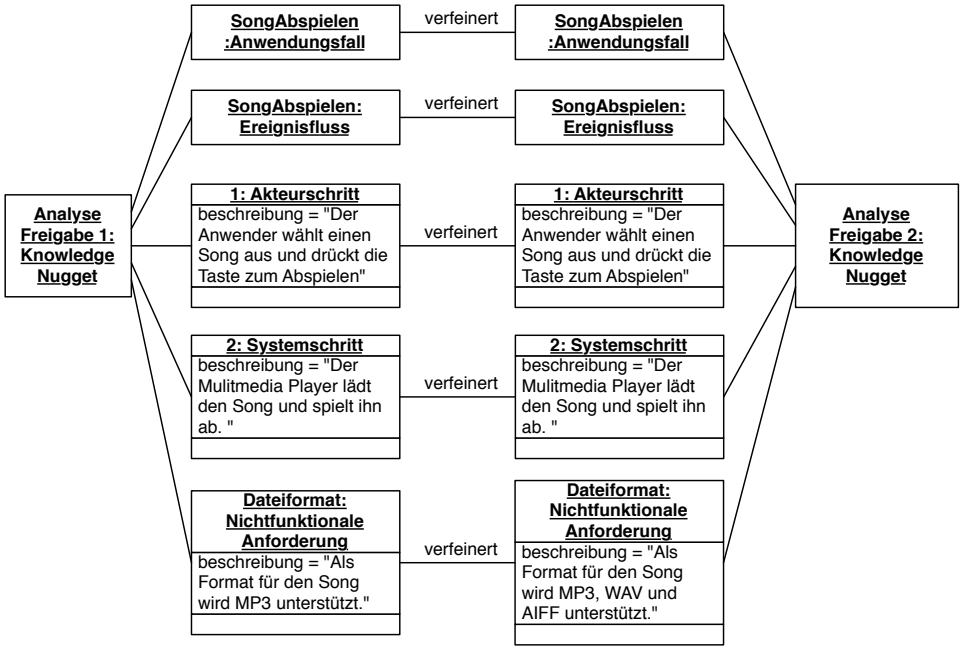


Abb. 48: Verfeinerung des Anwendungsfalls Song Abspielen: In Freigabe 2 unterstützt der Anwendungsfall SongAbspielen die Dateiformate MP3, WAV und AIFF.

Verfeinerung eines Anwendungsfalls des Multimedia Players

Abb. 48 zeigt als einfaches Beispiel die Verfeinerung des Anwendungsfalls SongAbspielen zwischen Freigabe 1 und 2 des Multimedia Players: Der Anwendungsfall SongAbspielen besteht in Freigabe 1 und 2 aus einem Ereignisfluss. Im Akteurschritt 1 initiiert der Akteur den Anwendungsfall, indem er einen Song auswählt und die Taste zum Abspielen drückt. Das System lädt

daraufhin in Systemschritt 2 den Song und spielt ihn ab. Die nichtfunktionale Anforderung Dateiformat gibt an, welche Formate für Songs unterstützt werden: Freigabe 1 unterstützt nur Songs im MP3-Format, Freigabe 2 verfeinert diese nichtfunktionale Anforderung und spezifiziert MP3, WAV und AIFF als Dateiformate.

Nachvollziehbarkeit

Abhängigkeiten zwischen verfeinerten Systemmodellelementen verschiedener Knowledge Nuggets erlauben *Post-traceability*. *Post-traceability* bezeichnet Nachvollziehbarkeit von den Anforderungen zur Implementierung und von der Implementierung zu den Anforderungen. [Jarke 1998] Beispielsweise kann der Freigabemanager die Auswirkung einer Änderung in den Anforderungen auf die Implementierung verfolgen. Ein weiteres Beispiel für *Post-traceability* ist die Bestimmung der betroffenen Merkmale, wenn in der Implementierung ein Fehler gefunden wird.

4.5 Planungsmethoden

Auswahl von Anforderungen

Planungsmethoden zeigen Vorgehensweisen oder Algorithmen auf, um die Anforderungen einer Freigabe aus einer Menge von Anforderungen so auszuwählen, dass die Auswahl bezüglich einer Menge von Kriterien optimal ist. [Bagnall et al. 2001] zeigt, dass dieses Problem NP-hart ist. NP-hart bedeutet (unter der wahrscheinlichen Vorraussetzung, dass P!=NP ist), dass eine Lösung des Problems nicht in polynomieller Zeit möglich ist. So unterscheiden sich Planungsmethoden in ihren Annäherungen an eine optimale Lösung für die nächste Freigabe, beispielsweise unter Verwendung eines Greedy Algorithmus oder mit Lokaler Suche [Bagnall et al. 2001]. EVOLVE [Greer & Ruhe 2004] verwendet einen genetischen Algorithmus und kann Alternativen für mehrere Freigaben vorschlagen.

Probleme der Planungsmethoden

Ein Problem von Planungsmethoden ist, dass Kriterien für die Entscheidungsfindung festgelegt und nicht angepasst werden können; beispielsweise berücksichtigt nur die Planungsmethode *Planning Software Evolution with Risk Management* [Greer 2004] Risiken. Für die Entscheidung wichtige Kriterien sind jedoch oft projektabhängig und können in BEEF frei definiert und angepasst werden. [Saliu & Ruhe 2005b] [Carlshamre 2002] Allen Methoden

gemeinsam ist, dass das Freigabedatum eine feste Größe in der Berechnung darstellt. Gerade eine Verschiebung des Lieferdatums stellt oft eine Lösung in der Freigabeplanung dar, wenn die geplante Funktionalität nicht bis zur Deadline geliefert werden kann und auch keine Einschränkungen in Qualität oder Umfang akzeptabel sind [Saliu & Ruhe 2005b], [Cohen 2007].

Abhängigkeiten

Die *Incremental Funding Method* [Denne & Cleland–Huang 2004] und EVOLVE berücksichtigen zwei Abhängigkeiten unter Anforderungen: Bei der *Incremental Funding Method* kann eine Vorgängerabhängigkeit für Anforderungen angegeben werden; sie gibt an, dass eine Anforderung vor einer anderen implementiert werden muss. Zusätzlich gibt es bei EVOLVE eine Gleichzeitigkeitsabhängigkeit, um auszudrücken, dass zwei Anforderungen mit der gleichen Freigabe geliefert werden müssen. Die Abhängigkeiten in einer Freigabe sind jedoch viel komplexer, wie Abschnitt 4.4 zeigt. Eine Reduzierung der Abhängigkeiten auf Vorgänger und Gleichzeitigkeitsabhängigkeit erlaubt es nicht, Beschränkungen durch das System zu berücksichtigen. Zudem reflektiert BEEF Abhängigkeiten zur Organisation, wie verfügbare Mitarbeiter und ihre Fähigkeiten, und erlaubt diese bei der Planung einer Freigabe zu berücksichtigen.

Detaillierungsgrad

Planungsmethoden legen den Detaillierungsgrad der auszuwählenden Freigabeplanungsentitäten, beispielsweise auf Merkmale, fest und berücksichtigen im Gegensatz zu BEEF keine Verfeinerung von Freigabeplanungsentitäten. Sie betrachten die Auswahl von Freigabeplanungsentitäten losgelöst von den Systemmodellen und erlauben damit keine Verbindung von Roadmapping, Freigabeplanung, der Systemarchitektur und Implementierung. [Vähäniitty et al. 2002] BEEF hingegen ist ein einheitliches Modell für die Erstellung von Roadmaps, die Planung einzelner Freigaben, die Erstellung von Systemmodellen verschiedener Abstraktionsgrade für aufeinanderfolgende Freigaben und unterstützt die Implementierung durch Fehlerberichte und *builds*.

5 Knowledge Nugget Modell

Modell der Knowledge Nuggets

Knowledge Nuggets unterstützen die Planung einer Folge von Freigaben im Freigabemodell (siehe Kapitel 3), indem sie Systemmodelle für eine Folge von Freigaben beschreiben. Im begründungsbasierten Freigabemodell (siehe Kapitel 4) dienen Knowledge Nuggets zur Verwaltung von Begründungen aus den Systemmodellen für aufeinanderfolgende Freigaben. Abb. 49 beschreibt das Modell der Knowledge Nuggets zur Verwaltung mehrerer Systemmodelle, beispielsweise für verschiedene Abstraktionsgrade oder Freigabedeskriptoren.

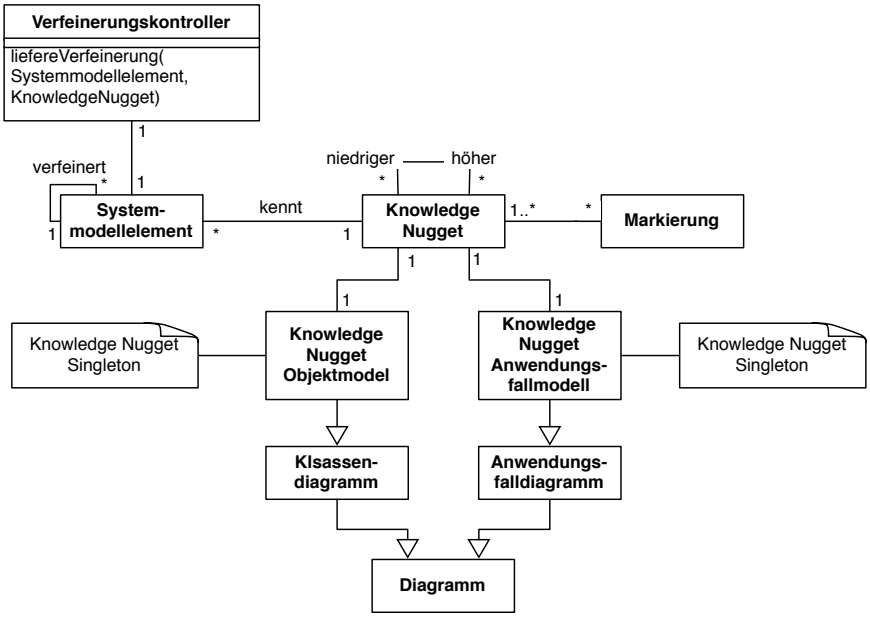


Abb. 49: Verfeinerung und Darstellung von Systemmodellen verschiedener Knowledge Nuggets

Verfeinerungskontroller

Knowledge Nuggets kennen beliebig viele Systemmodellelemente, die, wie in Abschnitt 4.4.4 beschrieben, Systemmodellelemente anderer Knowledge Nuggets verfeinern können. Durch mehrere verfeinerte Systemmodellelemente erhöht sich die Komplexität der Modellierung, weil ein Systemmodellelement, zum Beispiel eine bestimmte Klasse, in den Systemmodellen mehrerer Knowledge Nuggets vorkommen kann. Der Zustand des betroffenen Systemmodellelements kann sich in jedem Knowledge Nugget unterscheiden, wenn sich am Beispiel der Klasse ihre Beschreibung, Attribute oder Methoden weiterentwickelt haben. Ein Verfeinerungskontroller (siehe 5.1) kontrolliert Verfeinerungen eines Systemmodellelements und kann den Zustand für ein Systemmodellelement im Systemmodell eines konkreten Knowledge Nuggets bestimmen.

Beziehungen zw. Knowledge Nuggets

Ein Knowledge Nugget kann niedrigere oder höhere Knowledge Nuggets haben. Diese Beziehung drücken Abhängigkeiten zwischen Knowledge Nuggets aus, die zum Teil durch die Markierungen bedingt sind. Knowledge Nuggets nutzen diese Beziehung zusammen mit dem Verfeinerungskontroller, um Änderungen in den Systemmodellen zu berücksichtigen. (siehe Abschnitt 5.2)

Knowledge Nugget Singletons

Zur Visualisierung der Modelle werden Knowledge Nugget Objektmodelle und Knowledge Nugget Anwendungsfallmodelle eingeführt. Ein Knowledge Nugget Objektmodell ist ein Klassendiagramm, das alle Klassen des Knowledge Nuggets darstellt. Ein Knowledge Nugget Anwendungsfallmodell ist ein Anwendungsfalldiagramm mit einer Übersicht über alle Anwendungsfälle und Akteure, die in einem Knowledge Nugget beschrieben sind. Beide Diagramme sind *knowledge nugget singletons*, das heißt von ihnen existiert pro Knowledge Nugget nur eine Instanz. (siehe Abb. 49)

5.1 Verfeinerungskontroller

Verfeinerung der Klasse Medium des Multimedia Players

Ein Systemmodellelement kann in den Systemmodellen verschiedener Knowledge Nuggets vorkommen und dort verschiedene Zustände annehmen. Beispielsweise kommt die Klasse Medium im Analyse-, Systementwurfs- und Objektentwurfsmodell des Multimedia Players vor (siehe Abb. 50). In jedem Modell hat sie einen anderen Zustand: Im Analysemodell verfügt die Klasse

Medium nur über die Methode spiele(), im Systementwurf hat sie eine Assoziation zum Multimedia Speicher und im Objektentwurf ist die Methode +spiele():void durch ihre Signatur zusätzlich genau spezifiziert.

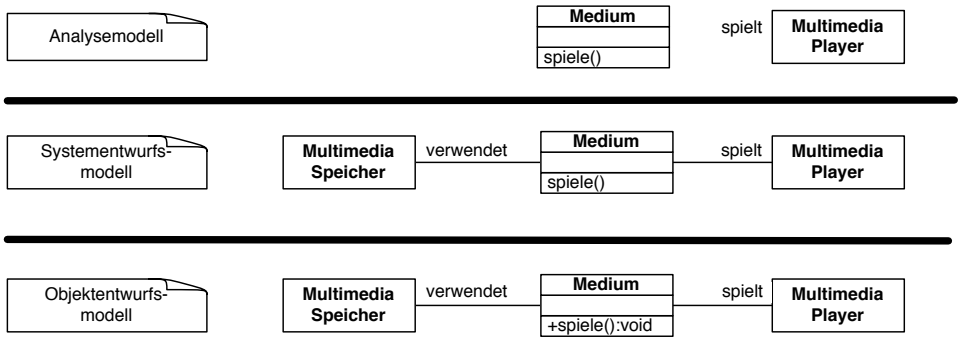


Abb. 50: Verfeinerung der Klasse Medium des Multimedia Players.

Speicherung von Änderungen

Zur Darstellung unterschiedlicher Zustände eines Objektmodells gibt es zwei grundlegend verschiedene Möglichkeiten: Die Speicherung der Unterschiede zwischen den Systemmodellelementen verschiedener Knowledge Nuggets und die Speicherung des gesamten Zustands eines Systemmodellelements. Die Speicherung von Unterschieden reduziert die zu speichernde Datenmenge, wohingegen die Berechnung des Zustandes eines Systemmodellelements für einen konkreten Knowledge Nugget Rechenzeit kostet. Dieser Ansatz ist für die Archivierung von Zuständen geeignet, beispielsweise im Konfigurationsmanagement, wenn auf alte Zustände eines Objekts nur selten zugegriffen wird.

Speicherung des gesamten Zustands

Das Freigabemanagement benötigt häufig den Zugriff auf mehrere Zustände eines Systemmodellelements, wenn Auswirkungen einer Änderung auf die Modelle eines anderen Abstraktionsgrades beurteilt werden müssen. Deshalb speichert das Knowledge Nugget Modell den gesamten Zustand eines Systemmodellelements und assoziiert ein Systemmodellelement mit genau einem Knowledge Nugget. Das Knowledge Nugget Modell erzeugt dadurch genau eine Instanz für jedes Systemmodellelement, das ein Knowledge Nugget kennt. Diese Instanz kann dann individuell und unabhängig von den anderen Instanzen, die konzeptionell das gleiche Systemmodellelement repräsentieren verändert werden. Alle diese Instanzen repräsentieren jedoch

dasselbe Objekt, nur dass es in jedem Knowledge Nugget anders modelliert sein kann; die Verfeinert-Assoziation hält solche Instanzen von konzeptionell identischen Systemmodellelementen zusammen.

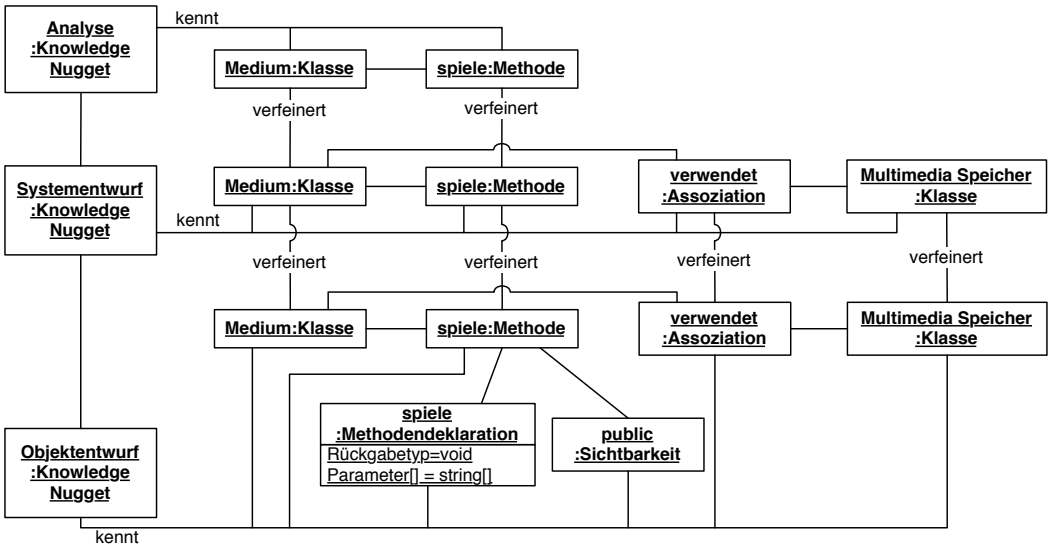


Abb. 51: Verfeinerung der Klasse Medium des Multimedia Players aus Abb. 50: In der Analyse verfügt sie nur über eine Methode spiele. Der Systementwurf verfeinert das Analysemodell durch Hinzufügen einer Assoziation zu Multimedia Speicher. Im Objektentwurf wird die Methode spiele durch eine Methodendeklaration und Sichtbarkeit weiter verfeinert.

Knowledge Nugget Modell für Klasse Medium

Abb. 51 zeigt die Instanzierung des in dieser Dissertation vorgestellten Modells für die Modellierung der Klasse Medium mit Systemmodellen auf drei Abstraktionsgraden, wie sie in Abb. 50 eingeführt worden ist. Das Beispiel verwendet drei Knowledge Nuggets für Analyse, Systementwurf und Objektentwurf, um die Klasse Medium zu verfeinern. In dem Instanzdiagramm sieht man, dass für jeden Knowledge Nugget eine Instanz jedes Systemmodellelements existiert, die durch Verfeinert-Assoziationen verbunden sind.

Verfeinerung des Multimedia Players

Abb. 52 zeigt ein weiteres einfaches Beispiel für die Verfeinerung einer Klasse Multimedia Player bei Verwendung von vier Knowledge Nuggets für zwei Freigaben und die Abstraktionsgrade Analyse und Systementwurf: Das Beispiel verfeinert die Beschreibung der Klasse von Analyse zu Systementwurf sowie von Freigabe 1 zu Freigabe 2: Die Analyse von Freigabe 1 betrachtet nur die Anforderungen, Songs und Podcast abzuspielen sowie die

Datenübertragung. Im Systementwurf wird hinzugefügt, dass die Datenübertragung auch per USB erfolgen soll. Das Analysemodell der Freigabe 2 verfeinert das Analysemodell der Freigabe 1 um das Abspielen von Videos. Im Systementwurf wird dann noch zusätzlich die Übertragung der Daten mit WLAN hinzugefügt. (siehe Abb. 52) Die Klasse Multimedia Player wird in allen Modellen verfeinert, indem für jede Verfeinerung eine eigene Instanz erzeugt wird, welche dann durch Verfeinert Assoziationen verbunden sind.

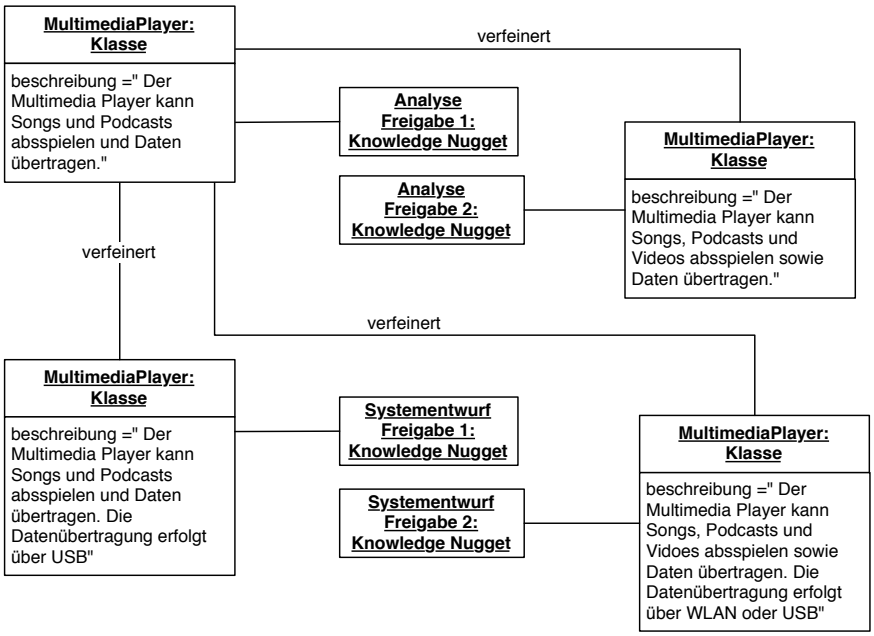


Abb. 52: Einfaches Beispiel für den Einsatz von Knowledge Nuggets: Die Klasse Multimedia Player wird in den Modellen von vier Knowledge Nuggets verfeinert. (UML-Instanzdiagramm)

Verwaltung von Verfeinerungen

Ein Verfeinerungskontroller verwaltet die Instanzen und Verfeinerungsassoziationen. Er hat eine Methode liefereVerfeinerung(Systemmodellelement systemmodellelement, Knowledge Nugget knowledge Nugget), um für ein Systemmodellelement die zu einem Knowledge Nugget passende Instanz eines Systemmodellelements zurückzuliefern. Beispielsweise liefert der Verfeinerungskontroller für eine beliebige Instanz des Multimedia Players und den Knowledge Nugget Systementwurf Freigabe 2 als Eingabe die zu dem Systementwurf der Freigabe 2 gehörende Instanz des Multimedia Players als Verfeinerung.

5.2 Änderungen an Knowledge Nuggets

Konsistenz-problem

Knowledge Nuggets erlauben eigenständige Modelle für beliebige Markierungen, beispielsweise Freigaben und Abstraktionsgrade. Da diese Modelle festgehalten und beliebig verändert werden können, entsteht der Bedarf Konsistenz zwischen diesen Modellen zu erhalten, damit durch die Weiterentwicklung des Modells eines einzelnen Knowledge Nuggets keine Widersprüche eingeführt werden. Wird beispielsweise in einer frühen Freigabe eine Anforderung hinzugefügt, so werden auch die Modelle späterer Freigaben diese Anforderung enthalten, solange bis eine erneute Änderung eintritt, die diese Anforderung betrifft. So kann nicht nur die erste Freigabe des Multimedia Players Songs abspielen, sondern auch alle folgenden Freigaben. Bei Änderungen an den Systemmodellen früherer Freigaben ist zu prüfen, ob diese auch die Modelle späterer Freigaben berühren.

Konsistenz-erhaltung

Um die Konsistenz zwischen den Modellen verschiedener Knowledge Nuggets zu erhöhen, führt das Knowledge Nugget Modell Beziehungen zwischen Knowledge Nuggets ein (siehe Abschnitt 5.2.1). Diese Beziehungen geben an, welche Knowledge Nuggets von einer Änderung betroffen sein können. Ob eine Änderung für einen anderen Knowledge Nugget relevant ist und sein Modell aktualisiert werden soll, bestimmt die Änderungspropagation (siehe Abschnitt 5.2.2). Ein Abhängigkeitskontroller (siehe Abschnitt 5.2.3) sorgt dafür, dass nicht nur das Systemmodellelement selbst, sondern auch andere von der Änderung betroffene Systemmodellelemente angepasst werden. Abschnitt 5.2.4 beschäftigt sich mit Änderungen von Knowledge Nuggets beim Einsatz von Freigabeszenarios.

5.2.1 Beziehungen zwischen Knowledge Nuggets

Beziehungen zwischen Freigaben

Knowledge Nuggets hängen voneinander ab, wenn zwischen Instanzen einer Markierung Beziehungen bestehen. Beispielsweise definiert die Roadmap eine zeitliche Ordnung auf Freigaben: Eine aktuelle Freigabe F_A wird nach einer vergangenen Freigabe F_V und vor einer zukünftigen Freigabe F_Z erstellt. Werden beispielsweise Fehler im Systemmodell einer früheren Freigabe identifiziert und behoben, können diese Korrekturen auch die Modelle nachfolgender Freigaben betreffen. Dies trifft auch dann zu, wenn die Sy-

stemmodelle mehrerer Freigaben gleichzeitig entwickelt werden: Beispielsweise während die Dokumentation der letzten Freigabe noch nicht abgeschlossen ist, wird bereits eine aktuelle Freigabe entwickelt und die nächste, künftige, Freigabe geplant. Die simultane Modellierung mehrerer Freigaben stellt die Anforderung, dass die Modelle untereinander konsistent bleiben und keine Widersprüche entstehen. Um dies zu gewährleisten, führt diese Dissertation Beziehungen zwischen Knowledge Nuggets ein, entlang derer Änderungen an Modellen weitergeleitet werden können. Für Freigaben ergibt sich die Beziehung aus der zeitlichen Ordnung, die eine Roadmap definiert.

Beziehungen zwischen Abstraktionsgraden

Zu einem Systemmodell eines bestimmten Abstraktionsgrads können Modelle mit mehr oder weniger Details existieren. Einen Abstraktionsgrad, der weniger Details zulässt, bezeichnet man als höher. Das Analysemodell ist also höher als das Systementwurfsmodell, welches wiederum höher ist als das Objektentwurfsmodell. Diese Definition geht davon aus, dass Kunden und Analysten wie ein Adler von oben herab auf das System blicken und ihnen so die tiefen Details der Implementierung verborgen bleiben. Bei Änderungen an den Systemmodellen höheren Abstraktionsgrades ist also zu überprüfen, ob sie sich auf die Systemmodelle niedrigeren Abstraktionsgrades auswirken. So kann eine Änderung in den Anforderungen, beispielsweise die Identifizierung eines neuen Analyseobjekts, eine Änderung des Objektentwurfsmodells zur Folge haben.

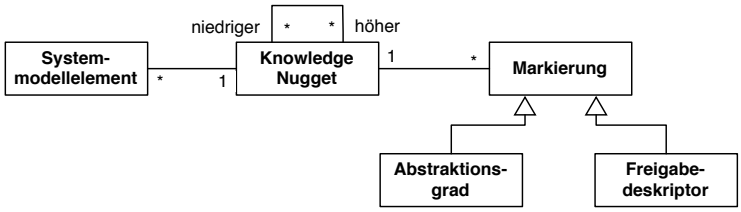


Abb. 53: Abhängigkeiten zwischen Knowledge Nuggets: Ein Knowledge Nugget kann mehrere höhere und niedrigere Knowledge Nuggets haben.

Niedriger/Höher-Assoziation

Um Änderungen an den Modellen eines Knowledge Nuggets an die Modelle anderer Knowledge Nuggets weitergeben zu können, wird eine Assoziation zwischen Knowledge Nuggets eingeführt, die angibt, dass ein Knowledge Nugget höher sein kann als andere. (siehe Abb. 53).

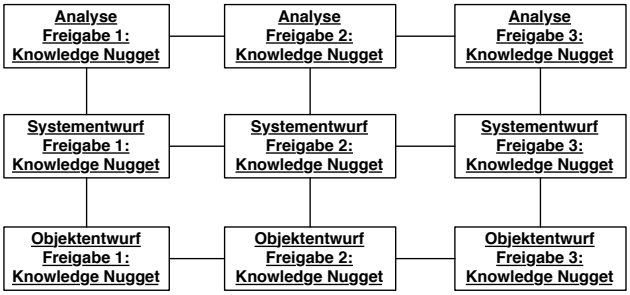


Abb. 54: Beziehungen zwischen den Knowledge Nuggets bei drei Freigabedeskriptoren und drei Abstraktionsgraden Analyse, Systementwurf und Objektentwurf (UML-Instanzdiagramm)

Beispiel

Abb. 54 zeigt ein UML-Instanzdiagramm mit Knowledge Nuggets für drei Freigaben und die Abstraktionsgrade Analyse, Systementwurf und Objektentwurf. Die Assoziationen geben die Beziehungen zwischen den Knowledge Nuggets an: Der Knowledge Nugget Systementwurf Freigabe 2 ist niedriger als Analyse Freigabe 2 und als Systementwurf Freigabe 1; er ist jedoch höher als der Objektentwurf der Freigabe 2 und als der Systementwurf der Freigabe 3.

5.2.2 Änderungspropagation

Konsistenz

Die Änderungspropagation dient dazu, die Systemmodelle verschiedener Knowledge Nuggets konsistent zu halten, das heißt sie dürfen sich nicht widersprechen. [Bruegge & Dutoit 2003] Widersprüche entstehen dadurch, dass sich das Modell eines Abstraktionsgrads weiterentwickelt, ohne die Modelle der übrigen Abstraktionsgrade zu betrachten. Die Änderungspropagation unterstützt Konsistenzerhaltung zwischen den Systemmodellen verschiedener Knowledge Nuggets, indem bestimmte Änderungen an Systemmodellen eines höheren Knowledge Nuggets an Systemmodelle niedrigerer Knowledge Nuggets weitergegeben werden.

Hinzufügen eines Anwendungsfalls

Beispielsweise entscheidet der Freigabemanager in Freigabe 2 des Multimedia Players Podcasts einzuführen: Er fügt dem Analysemodell der Freigabe 2 einen Anwendungsfall PodcastsAbspielen und eine teilnehmende Klasse Podcast als neues Medium hinzu. (siehe Abb. 55) Der neue Anwendungsfall PodcastsAbspielen ist nicht nur der Freigabe 2, sondern auch dem Analyse

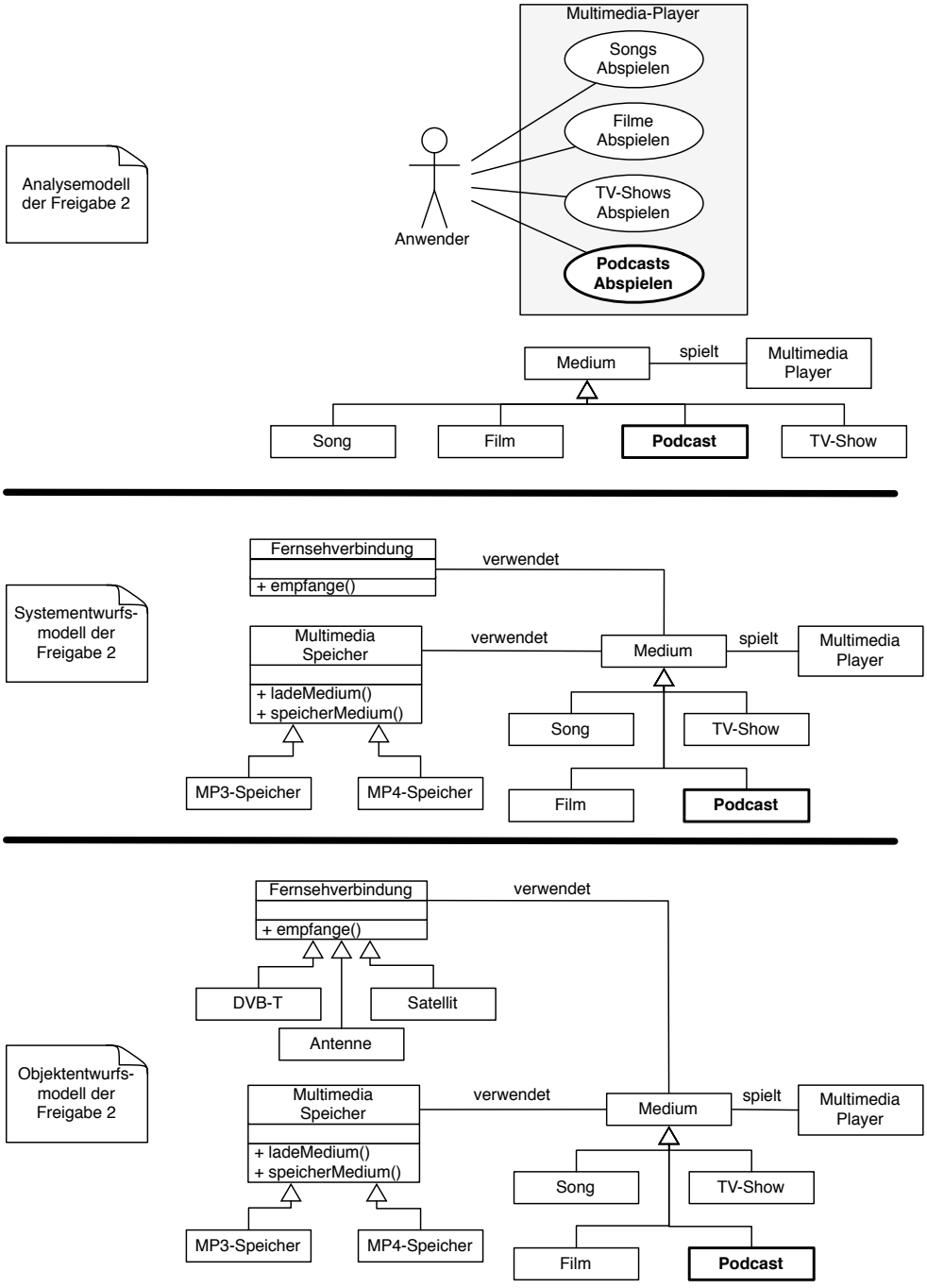


Abb. 55: Änderungspropagation bei einer Änderung des Analysemodells der Freigabe 2 des Multimedia Players: Es wird ein Anwendungsfall PodcastsAbspielen eingeführt, der als partizipierende Klasse Podcasts einführt. Diese Klasse muss nun auch dem Systementwurfs- und Objektentwurfsmodell hinzugefügt werden.

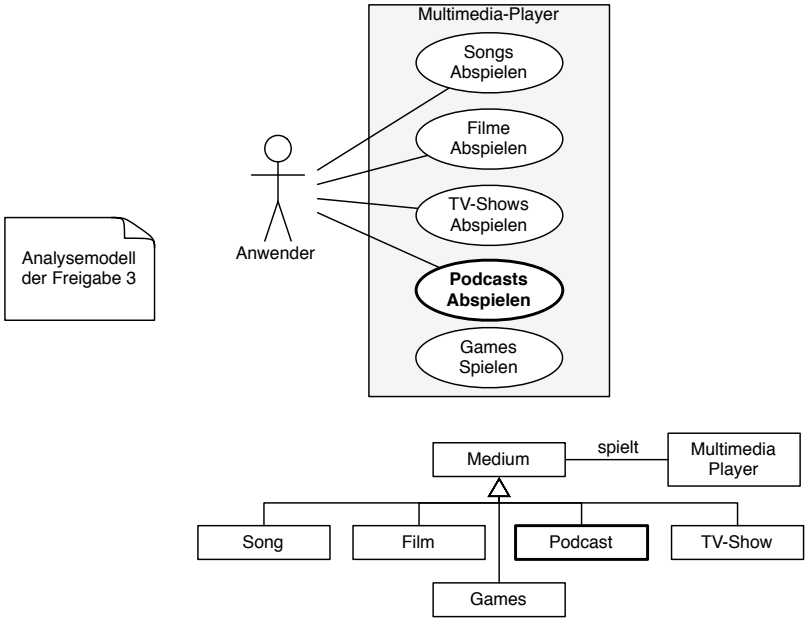


Abb. 56: Änderung des Analysemodells der Freigabe 3: Durch die Einführung von Podcasts in Freigabe 2 werden auch dem Analysemodell der Freigabe 3 der Anwendungsfall PodcastsAbspielen und die Klasse Podcast hinzugefügt.

modell der Freigabe 3 hinzuzufügen (siehe Abb. 56). Das Hinzufügen des Anwendungsfalls ändert jedoch noch nicht das Systementwurf und Objektentwurfsmodell der Freigabe 2.

Hinzufügen einer Klasse

Im Gegensatz dazu ist das Einführen einer neuen Klasse Podcast als Spezialisierung der Klasse Medium im Analyseobjektmodell auch für das Systementwurfs- und Objektentwurfsmodell der Freigabe 2 relevant. Änderungen an den Klassen sind also für die Systemmodelle niedrigeren Abstraktionsgrads relevant. (siehe Abb. 55) Die neue Klasse Podcast soll zusätzlich auch dem Analyse-, Systementwurfs- und Objektmodellen der Freigabe 3 hinzugefügt werden.

Änderungs-propagation

Verfeinerungen, Hinzufügen und Löschen von Systemmodellelementen im Modell eines höheren Knowledge Nugget können zu Inkonsistenzen in den Systemmodellen niedrigerer Knowledge Nuggets führen. Die Modelle aller Knowledge Nuggets bei jeder Änderung von Hand zu überprüfen und anzu

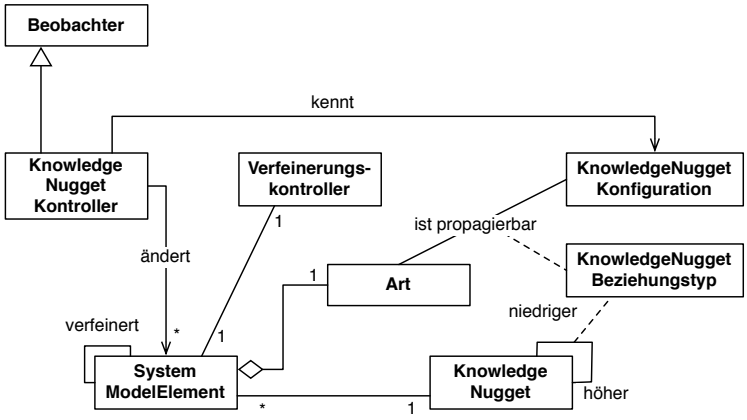


Abb. 57: Änderungspropagation zwischen den Modellen mehrerer Knowledge Nuggets: Der Knowledge Nugget Kontroller fragt bei Änderungen die Knowledge Nugget Konfiguration, ob für eine gegebene Art von Systemmodellelement und einen Knowledge Nugget Beziehungstyp eine Propagation der Änderung zu niedrigeren Knowledge Nuggets erfolgen soll.

passen, führt zu einem Mehraufwand in der Modellierung. Um diesen zu reduzieren, geben höhere Knowledge Nuggets Änderungen an niedrigere Knowledge Nuggets weiter.

Knowledge Nugget Beziehungstypen

Dabei bestimmen ein Knowledge Nugget Beziehungstyp und eine Knowledge Nugget Konfiguration (siehe Abb. 57), ob Knowledge Nuggets Änderungen an den Modellen eines Knowledge Nuggets weitergeben. Der Knowledge Nugget Beziehungstyp gibt an, welche Art von Beziehung zwischen zwei Knowledge Nuggets besteht. Beispielsweise kann ein Beziehungstyp die Beziehung zwischen den Knowledge Nuggets auf Grund von Abstraktionsgraden spezifizieren. Ein anderer Beziehungstyp kann die Beziehung zwischen Knowledge Nuggets zweier Freigabedeskriptoren bezeichnen (siehe Abb. 58).

Knowledge Nugget Konfiguration

Beziehungen zwischen den Knowledge Nuggets können also durch die Verwendung von Knowledge Nugget Beziehungstypen unterschieden werden. Für jede Art von Systemmodellelementen können in einer Knowledge Nugget Konfiguration solche Knowledge Nugget Beziehungstypen festgelegt werden, für die eine Änderung an dem Systemmodellelement an niedrigere Knowledge Nuggets weitergegeben werden soll. Beispielsweise kann das System so konfiguriert werden, dass Änderungen an Klassen sowohl zwischen den Modellen verschiedener Abstraktionsgrade als auch von Freigabe zu Freigabe

weitergegeben werden, während Änderungen an Anwendungsfällen nur von Freigabe zu Freigabe weitergegeben werden.

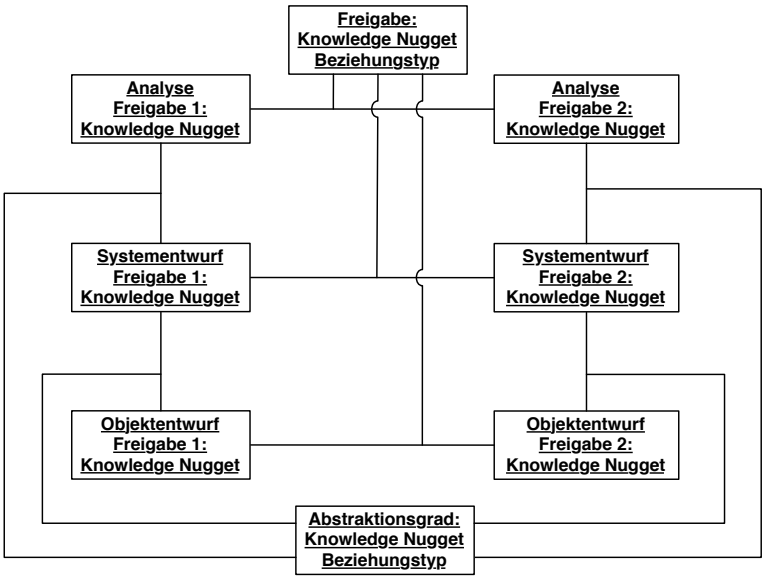


Abb. 58: Beziehungen zwischen Knowledge Nuggets mit Knowledge Nugget Beziehungstypen: Die Beziehungen zwischen Knowledge Nuggets zur Darstellung der zeitlichen Ordnung auf Freigaben sind mit einem Beziehungstyp für Freigaben spezifiziert. Die Beziehungen zwischen den Knowledge Nuggets zur Darstellung von Abstraktionsgrade sind mit einem Beziehungstyp für Abstraktionsgrade markiert.

Knowledge Nugget Kontroller

Der Knowledge Nugget Kontroller in Abb. 57 verwendet die Knowledge Nugget Konfiguration und übernimmt die Propagation von Änderungen, indem er die Modelle der Knowledge Nuggets beobachtet. Kommen neue Systemmodellelemente hinzu, werden diese gemäß der Konfiguration an die Modelle niedrigerer Knowledge Nuggets weitergegeben. Bei Änderungen an existierenden Systemmodellelementen verwendet der Knowledge Nugget Kontroller den Verfeinerungskontroller, um die zu anderen Knowledge Nuggets gehörenden Systemmodellelemente zu erhalten. Änderungen werden nur dann propagiert, wenn die geänderte Eigenschaft zuvor in den Modellen beider Knowledge Nuggets gleich war; ansonsten wird die Änderung verworfen. Beispielsweise wird eine Änderung des Klassennamens im Analysemodell der Freigabe 2 aus Abb. 55 von Medium zu Produkt in den Systementwurf und Objektentwurf der Freigaben 2 und 3 propagiert.

Stärkung der Zusammenarbeit

Dieser Mechanismus zur Änderungsverbreitung sorgt für konsistente Modelle zwischen Knowledge Nuggets, ohne dass die Anwender alle Modelle von Hand pflegen müssen. Ebenso wird dadurch die Zusammenarbeit gestärkt: Ergibt sich eine Änderung in der Anwendungsdomäne, fügt ein Analyst ein neues Analyseobjekt hinzu, welches in den Systementwurf propagiert wird. Der Architekt sieht dort das neue Objekt und kann es in seinen Entwurf integrieren, beispielsweise indem er es einem Subsystem zuordnet oder in ein Entwurfsmuster integriert.

5.2.3 Abhängigkeitskontroller

Abhängigkeiten im Systemmodell

Änderungen an einem Systemmodellelement ziehen oft Änderungen an einer Reihe von Systemmodellelementen nach sich, wie das folgende Beispiel illustriert: Die Freigabe 2 des Multimedia Players aus Abb. 55 ist um den Anwendungsfall Podcasts Abspielen erweitert worden; damit die Lieferung der Freigabe nicht gefährdet wird, beschließt der Freigabemanager den Anwendungsfall TV-Shows Abspielen auf Freigabe 3 zu verschieben. Das Analyse-Objektmodell von Freigabe 2 besteht aus den Klassen Multimedia Player, Medium, Song, Film, Podcast und TV-Show (siehe Abb. 59), während das Analyse-Objektmodell von Freigabe 3 zudem Game als weitere Spezialisierung von Medium vorsieht (siehe Abb. 60).

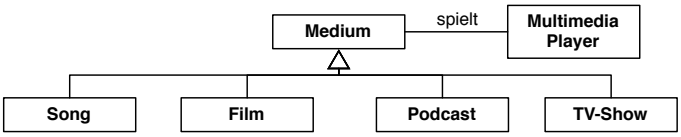


Abb. 59: Analyse-Objektmodell der Freigabe 2 des Multimedia Players vor Verschieben des Anwendungsfalls TV-Shows Abspielen auf Freigabe 3

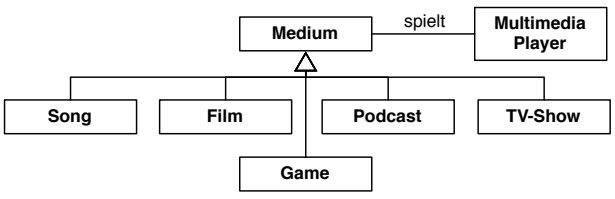


Abb. 60: Analyse-Objektmodell der Freigabe 3 des Multimedia Players

Verschieben eines Anwendungsfalls

Wird nun der Anwendungsfall TV-Shows Abspielen von Freigabe 2 auf Freigabe 3 verschoben, werden auch die Analyse-Objektmodelle der beiden Freigaben angepasst. Nicht mehr benötigte partizipierende Objekte des betroffenen Anwendungsfalls, im Beispiel die Klasse TV-Show aus dem Analyse-Objektmodell, werden aus der Freigabe 2 entfernt und der Freigabe 3 hinzugefügt. (siehe Abb. 61).

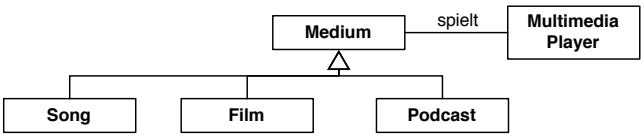


Abb. 61: Analyse-Objektmodell der Freigabe 2 eines Multimedia Players nach Verschieben des Anwendungsfalls TV-Shows Abspielen auf Freigabe 3

Abhängigkeitskontroller

Diese Aufgabe übernimmt ein Abhängigkeitskontroller. Er verwaltet die Abhängigkeiten zwischen Systemmodellelementen, indem für jedes Systemmodellelement definiert werden kann, welche anderen Arten von Systemmodellelementen bei einer Änderung auch verschoben werden sollen (Abb. 62). Beispielsweise kann man für Anwendungsfälle definieren, dass mit ihnen initiierende und partizipierende Akteure, der Ereignisfluss sowie die partizipierenden Klassen verschoben werden sollen. Für Klassen kann konfiguriert werden, dass mit ihnen Operationen und Methoden sowie verfeinerte Klassen und Assoziationen verschoben werden sollen.

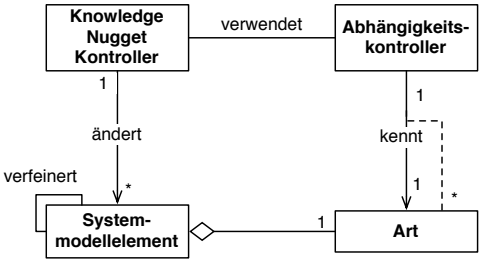


Abb. 62: Berücksichtigung von Abhängigkeiten bei Änderungen im Modell: Der Abhängigkeitskontroller kennt für jede Art von Systemmodellelementen andere Arten von Systemmodellelementen, die bei einer Änderung verschoben werden müssen.

5.2.4 Freigabeszenarios

Freigabeszenarios

Freigabeszenarios erlauben es, alternative Möglichkeiten für die Implementierung einer Freigabe zu untersuchen. Beispielsweise können sich als Kon-

sequenz einer Anforderungsänderung zwei Alternativen in ihrer Systemarchitektur unterscheiden. Das Knowledge Nugget Modell sieht eigene Systemmodelle für Freigabeszenarios vor, um diese Alternativen modellieren, erkunden, miteinander vergleichen und gegeneinander abwägen zu können.

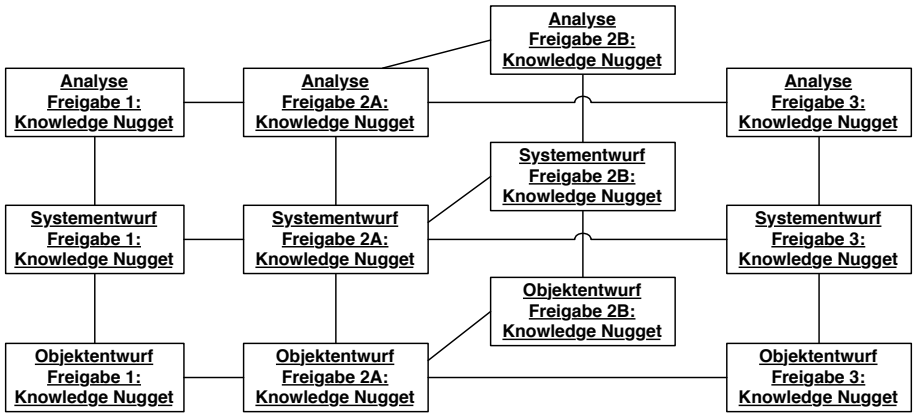


Abb. 63: Beispiel für den Einsatz von Knowledge Nuggets für die Modelle auf drei Abstraktionsgraden für drei Freigaben, wobei zwei Alternativen, Freigabe 2A und Freigabe 2B für die zweite Freigabe zur Auswahl stehen.

Entscheidung für alternatives Freigabeszenario

Ein Freigabeszenario hat wie eine Freigabe einen Freigabedeskriptor, durch den Knowledge Nuggets erzeugt werden. Um die Modelle alternativer Freigabeszenarios auch aktuell zu halten, kann zwischen Knowledge Nuggets von beschlossenen und alternativen Freigabeszenarios eine Beziehung aufgestellt werden. Entlang dieser Beziehung können Änderungen aus dem Modell des beschlossenen Freigabeszenarios zu den alternativen Freigabeszenarios propagiert werden. Abb. 63 zeigt ein UML-Instanzdiagramm für den Einsatz von Knowledge Nuggets für drei Abstraktionsgrade (Analyse, Systementwurf und Objektentwurf) und für drei Freigaben, wobei für die Freigabe 2 alternative Freigaben 2A und 2B, existieren. In Abb. 63 ist die alternative Freigabe 2A beschlossen, weshalb Änderungen aus den Knowledge Nuggets der Freigabe 1 direkt in die Freigabe 2A weitergegeben werden. Der Knowledge Nugget der Freigabe 2A propagiert die Änderungen in Freigabe 3 und in die Alternative Freigabe 2B.

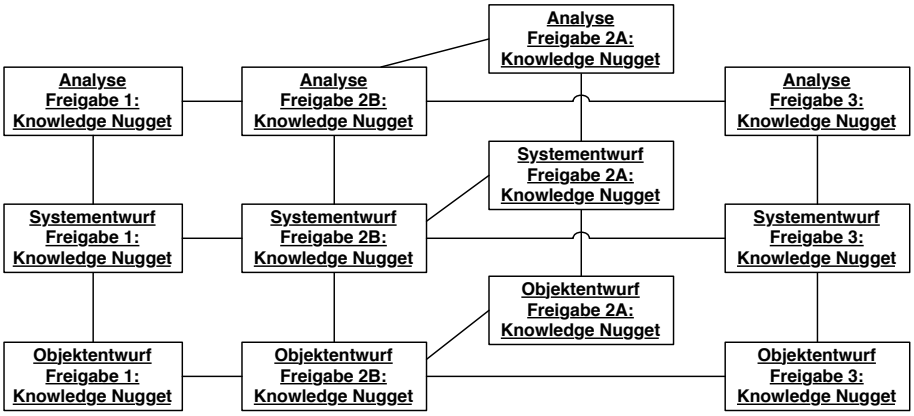


Abb. 64: Beispiel für den Einsatz von Knowledge Nuggets für die Modelle auf drei Abstraktionsgraden für drei Freigaben, wobei zwei Alternativen, Freigabe 2A und Freigabe 2B, für die zweite Freigabe zur Auswahl stehen. Im Vergleich zu Abb. 63 ist nun beschlossen worden, die Modelle der Knowledge Nuggets der Freigabe 2B umzusetzen.

Änderungspropagation zu Freigabeszenarios

Entscheidet sich der Freigabemanager für ein alternatives Freigabeszenario, im Beispiel die Freigabe 2B statt der beschlossenen Freigabe 2A, müssen die Beziehungen zwischen den Knowledge Nuggets angepasst werden. Das neu ausgewählte Freigabeszenario, im Beispiel 2B, erhält die niedrigeren und höheren Knowledge Nuggets der alten Freigabe, im Beispiel der Freigabe 2A. Zudem beschreiben die Knowledge Nuggets der zuvor beschlossenen Freigabe nun eine Alternative. Im Beispiel werden die Knowledge Nuggets der Freigabe 2A nun Alternativen zur beschlossenen Freigabe 2B (siehe Abb. 64).

5.3 Kommunikationsmodelle

Wissen aus der Kommunikation

Neben den Systemmodellen haben Kommunikationsmodelle für das Freigabemanagement Bedeutung, da sie Begründungen für Entscheidungen festhalten. Kommunikationsmodelle werden - wie in Abschnitt 4.2 eingeführt - durch Annotation von Systemmodellelementen festgehalten. Dieser Abschnitt beschäftigt sich mit der Verwaltung von Wissen aus der Kommunikation durch Knowledge Nuggets.

In Abschnitt 5.1 wird beschrieben, dass für jeden Knowledge Nugget eine eigene Instanz eines Systemmodellelements erzeugt wird, um die unterschiedlichen Zustände zu speichern. Wird nun ein Systemmodellelement mit einem

Relevanz der Kommunikationselemente

Kommunikationsmodellelement annotiert, kann das Kommunikationsmodellelement für die Instanzen mehrerer Knowledge Nuggets relevant sein. Ist bei der Analyse der Anforderungen für die Freigabe 2 des Multimedia Players beispielsweise noch nicht klar, welches Dateiformat für Podcasts verwendet werden soll, kann eine Fragestellung Welches Dateiformat wird für Podcasts verwendet? den Anwendungsfall PodcastsAbspielen und die Klasse Podcast annotieren. Die Lösung dieser Fragestellung beeinflusst nicht nur die Analyse, sondern auch die Architektur und Implementierung, weshalb die Fragestellung auch im Systementwurfs- und Objektentwurfsmodell erscheinen sollte.

Propagation von Kommunikationsmodellen

Der Knowledge Nugget Kontroller überprüft beim Hinzufügen eines neuen Kommunikationselements zu einem Systemmodellelement, ob Verfeinerungen dieses Systemmodellelements in niedrigeren Knowledge Nuggets existieren. Ist das der Fall, annotiert er auch die verfeinerten Systemmodellelemente mit dem Kommunikationselement. Abb. 65 zeigt, wie die Fragestellung Welches Dateiformat wird für Podcasts verwendet? die Klasse Podcast der Knowledge Nugget Analyse, Systementwurf und Objektentwurf annotiert: Alle Instanzen der Klassen Podcast haben eine Assoziation zu nur einer Instanz der Fragestellung. Dadurch ist eine Änderung, beispielsweise eine Lösung der Fragestellung sofort für alle Beteiligten sichtbar.

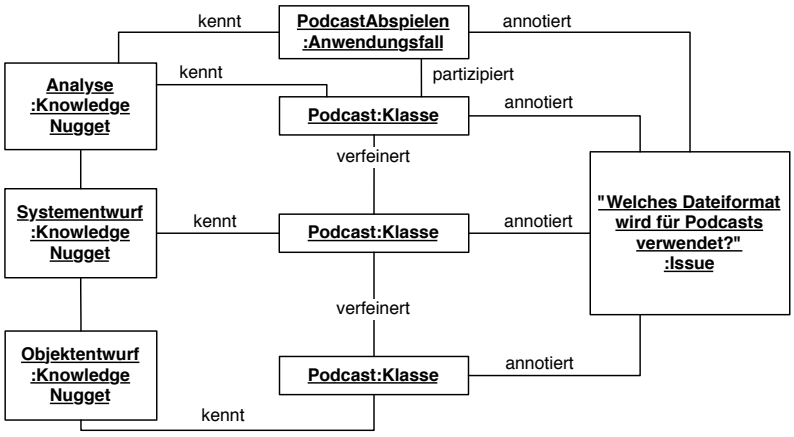


Abb. 65: Verwaltung von Kommunikationselementen durch Knowledge Nuggets: Die Fragestellung aus der Analyse „Welches Dateiformat wird für Podcasts verwendet?“ annotiert auch die Verfeinerungen der Klasse Podcast im Systementwurf und Objektentwurf

5.4 Knowledge Nugget Filter

Filter

Filter unterstützen die Suche nach Modellen, indem sie die Auswahl der Modellelemente beschränken. Ein Knowledge Nugget Filter kann eine gegebene Menge von Modellelementen nach Knowledge Nugget filtern, das heißt er liefert nur die Modellelemente zurück, die zu einem bestimmten Knowledge Nugget gehören.

Filter zur Erzeugung von Dokumenten

Beispielsweise verwendet das RUSE Modell [Wolf 2007] Filter zur strukturierten Darstellung der Modelle in Dokumenten. Ein Dokument besteht aus mehreren Abschnitten, die entweder wieder aus weiteren Abschnitten zusammengesetzt sein können oder Modelle enthalten. Diese Modelle filtern *leaf sections* nach Klassen (siehe Abb. 66), das heißt eine *leaf section* zeigt Modellelemente einer bestimmten Klasse, zum Beispiel Anwendungsfälle, an. Ein Knowledge Nugget Filter kann für *leaf sections* nicht nur nach Klassen, sondern auch nach Klassen und Knowledge Nuggets filtern, so dass für eine *leaf section* s nur die Modellelemente angezeigt werden, die zu dem gleichen Knowledge Nugget gehören, zu der auch die Sektion selbst gehört.

s.getModelElements()→forAll(m: ModelElement | verfeinerungskontroller.liefere KnowledgeNugget(m)= verfeinerungskontroller.liefereKnowledgeNugget(s))

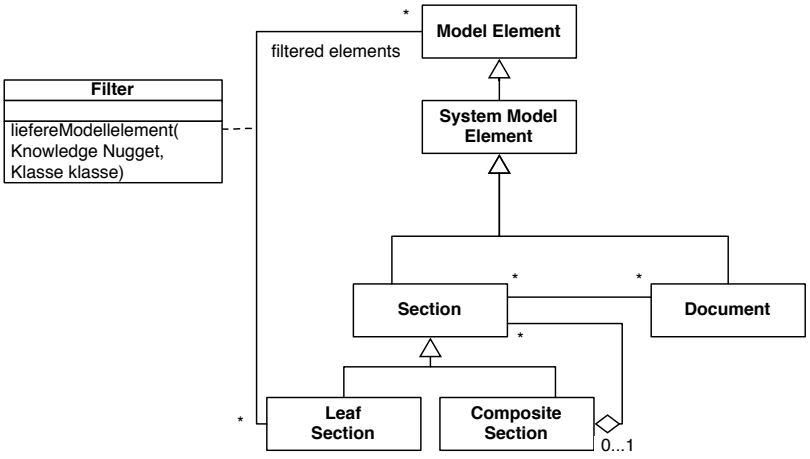


Abb. 66: Das RUSE- Dokumentenmodell [Wolf 2007]

5.5 Zweige im Konfigurationsmangement

Zweige

Ein Zweig im Konfigurationsmanagement ist eine Kopie einer Menge von Konfigurationseinheiten in einem eigenen Verzeichnis, die unabhängig voneinander bearbeitet und verändert werden kann. Durch die Verwendung von Zweigen entstehen nebenläufig unabhängige Versionen von Artefakten eines Systems, die später wieder semiautomatisch in eine einzelne Version gemischt, das heißt zusammengeführt, werden müssen, da nur eine Version des Systems an den Kunden geliefert wird. Dieses Mischen kann zu Konflikten führen, wenn in zwei Zweigen identische Stellen geändert wurden, und stellt ein Problem dar, wenn die entwickelten Versionen eines Systems zu unterschiedlich sind.

Änderungen in Zweigen u. Knowledge Nuggets

Knowledge Nuggets erlauben wie Zweige im Konfigurationsmanagement eine parallele Entwicklung von Systemmodellen. Die Änderungspropagation zwischen Knowledge Nuggets erhöht jedoch die Konsistenz zwischen diesen Modellen, indem sie Änderungen regelmäßig, automatisch an niedrigere Knowledge Nuggets weitergibt und dort mischt. Das erhöht die Bewusstheit über Änderungen zwischen Stakeholdern: Entwickler des Objektentwurfs erfahren sofort über Änderung aus der Analyse, beispielsweise wenn ein neues Analyseobjekt eingeführt wird. Entwickler nachfolgender Freigaben kennen aktuelle Entwicklungen vorangehender Freigaben und können sofort auf diese reagieren. Im Gegensatz zu Zweigen, die Entwickler vor Änderungen abschotten wollen, unterstützen Knowledge Nuggets Änderungen in der parallelen Entwicklung mehrerer Systemmodelle.

Ein weiterer Unterschied zwischen Knowledge Nuggets und Konfigurationsmanagement ist, dass das Modell eines Knowledge Nuggets mehrere Modelle als Ursprung haben kann und somit Änderungen mehrerer Modelle empfängt. Beispielsweise hat der Knowledge Nugget Freigabe 2 Systementwurf die beiden höheren Knowledge Nuggets Freigabe 2 Analyse und Freigabe 1 Systementwurf, von denen er Änderungen empfängt. Entwickler des Systementwurfs der Freigabe 2 sind also über Änderungen im Systementwurf der Freigabe 1 und der Analyse der Freigabe 2 informiert.

Auswahl zusammengehöriger Modellelemente

Zudem verschiebt der Abhängigkeitskontroller mit einem Systemmodellelement auch andere abhängige Systemmodellelemente zu anderen Knowledge Nuggets. Das Konfigurationsmanagement hingegen kopiert stets nur selektierte Entitäten zwischen Zweigen, wobei die Auswahl dem Anwender überlassen wird. Beispielsweise kann der Abhängigkeitskontroller mit einem Anwendungsfall alle partizipierenden Klassen von einem Knowledge Nugget zu einem anderen verschieben ohne dass der Anwender diese kennen muss.

Wissen aus Kommunikation und Organisation

Ein Knowledge Nugget kennt Abhängigkeiten zu Kommunikations- und Organisationsmodellen und kann für die Systemmodelle eines Knowledge Nuggets alle oder kritische Kommunikationsmodellelemente und organisatorischer Einheiten bestimmen. Beispielsweise kann ein Knowledge Nugget zur Modellierung einer Freigabe alle offenen Fragestellungen, eingetretenen Risiken, offenen Fehlerberichte sowie offene Terminaufgaben bestimmen und so dem Freigabemanager einen Überblick über den Status des Projekts geben.

6 Visualisierung

Mehrdimensionaler Entscheidungsraum

Entscheidungen im Freigabemanagement haben eine höhere Komplexität als die Entscheidungsfindung beim Entwurf eines Software-Systems: Systemmodelle auf mehreren Abstraktionsgraden, aufeinanderfolgender Freigaben oder alternativer Freigabeszenarios stellen zusammen mit Ressourcen und Begründungen aus Kommunikationsmodellen einen mehrdimensionalen Entscheidungsraum auf, den es für einen Freigabemanager bei Entscheidungen zu erforschen gilt. Als weitere Dimension kommen Versionen hinzu, die für andere oben genannte Dimensionen Änderungen im Laufe der Entwicklung festhalten.

Ressourcen

Für Ressourcen können abhängige Artefakte des Projekts angezeigt werden. Beispielsweise können für einen Mitarbeiter alle seine Arbeitseinheiten oder Anwendungsfälle, deren Umsetzung von ihm abhängt, angegeben werden.

Begründungen

Für Elemente des Begründungsmodells können betroffene Artefakte des Systemmodells ermittelt werden. Der Freigabemanager kann beispielsweise von einer offenen Fragestellung zu zugeordneten Klassen und von diesen zu Anwendungsfällen navigieren.

Freigaben

Die Dimension Freigabe fokussiert den Zustand eines Software-Systems einer bestimmten Freigabe. Durch eine Navigation von Freigabe zu Freigabe kann der Freigabemanager sehen, wie sich das Software-System über gelieferte Freigaben hinweg weiterentwickelt hat bzw. bei zukünftigen Freigaben entwickeln soll.

Alternativen

Ein Freigabeszenario stellt für jede Freigabe eine oder mehrere Alternativen dar. Um sich für und gegen solche Alternativen entscheiden zu können, navigiert der Freigabemanager von einer Freigabe zu ihren Alternativen.

Versionen

Änderungen über die Zeit hält das Konfigurationsmanagement [IEEE 1987] in Versionen fest. Versionen erlauben die Nachverfolgung von Änderungen für die übrigen Dimensionen. Die Historie der Versionen eines Software-Systems unterstützt Entscheidungen im Freigabemanagement, wenn Änderungen zurückgenommen werden, beispielsweise ein Mitarbeiter das Projekt doch nicht verlässt.

Abstraktionsgrad

Durch eine Navigation in Richtung des Abstraktionsgrads erhöht sich der Detaillierungsgrad eines Modells. Der Freigabemanager kann von abstrakten Analysemodellen zu detaillierten Objektmodellen navigieren und diese als Grundlage für seine Entscheidung heranziehen. Existierende Visualisierungen aus der Projektplanung und dem Buildmanagement betrachten nur eine der relevanten Dimensionen.

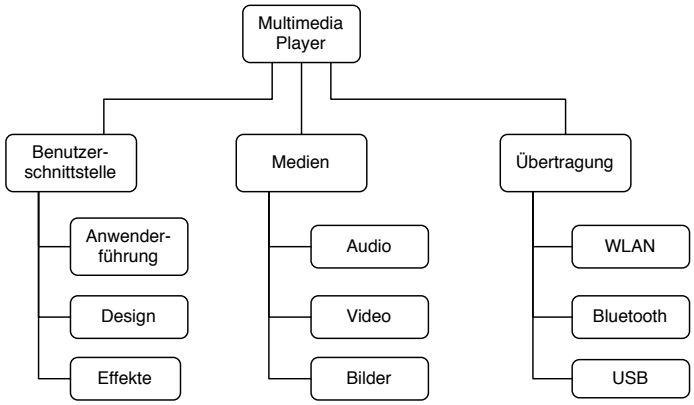


Abb. 67: Projektstrukturplan für den Multimedia Player: Die Erstellung des Multimedia Players wird in Arbeitspakete für die Benutzerschnittstelle, Medien und Übertragung zerlegt, die dann weiter zerlegt werden, um planbare und kontrollierte Arbeitspakete zu erhalten.

6.1 Visualisierungen aus der Projektplanung

Projektplanung

Bei linearen Vorgehensmodellen steht die Projektplanung im Vordergrund, die versucht Projekte in Teilaufgaben aufzuteilen und diese dann auf einer

Zeitachse zu planen. Techniken der Projektplanung sind Projektstrukturpläne, Gantt-Charts und Netzpläne.

Projektstrukturpläne

Projektstrukturpläne [Zimmermann & Stache 2001] (*work breakdown structures*) zeigen eine Aufteilung eines Projekts in immer detailliertere Teilaufgaben, um Projekte planbar und kontrollierbar zu machen. Sie fokussieren also nur eine Dimension, nämlich die Zerlegung der Systementwicklung in handhabbare Arbeitspakete. Abb. 67 zeigt als Beispiel eine Visualisierung eines Projekt Strukturplans für den Multimedia Player: Er zerlegt die Erstellung des Multimedia Players in den Entwurf einer Benutzerschnittstelle, die Verwaltung der Medien und die Übertragung der Medien auf den Multimedia Player. Im Beispiel ist das Arbeitspaket Benutzerschnittstelle in Anwenderführung, Design und Effekte aufgeteilt. Das Arbeitspaket Medien besteht aus Verwaltung und Abspielen bzw. Anzeigen von Audio, Video und Bilder. Das Arbeitspaket Übertragung garantiert die Übertragung der Mediendaten auf den Multimedia Player durch die Arbeitspakete zur Übertragung via WLAN, Bluetooth und USB. Projektstrukturpläne konzentrieren sich auf eine Visualisierung der Zerlegung der Aktivitäten ohne eine Betrachtung von Freigaben, Alternativen oder Systemmodellen.

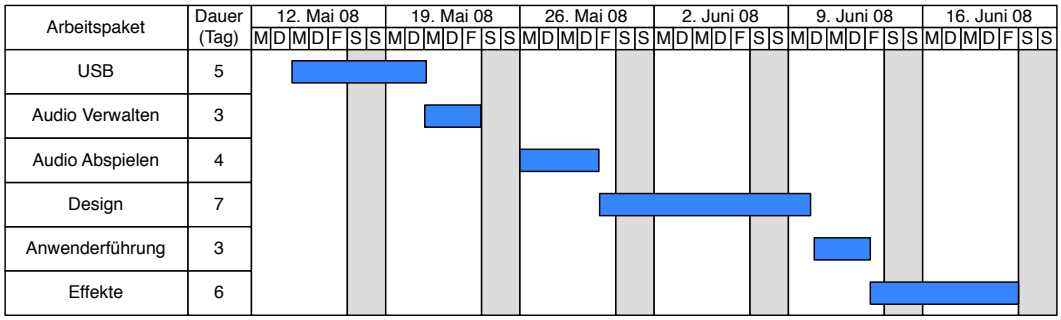


Abb. 68: Gantt-Chart für den Multimedia Player: Das Arbeitspaket USB soll von Mittwoch 14. Mai 2008 bis Dienstag 20. Mai implementiert werden, im Anschluss daran folgen nacheinander die Arbeitspakete Audio Verwalten bis Freitag 23. Mai, Audio Abspielen bis Donnerstag 29. Mai, Design der Benutzerschnittstelle bis 9. Juni, Anwenderführung bis 12. Juni sowie abschließend Effekte bis Freitag 20. Juni.

Gantt-Charts

Gantt-Charts [Gantt 1910] visualisieren die Ablaufplanung und zeigen, bis wann Arbeitspakete abgeschlossen sein sollen, indem sie für geplante Aktivitäten, beispielsweise Arbeitspakete aus den Projektstrukturplänen, Dauer und Ende angeben. Abb. 68 zeigt ein Beispiel eines Gantt-Charts für die

Arbeitspakete aus dem Projektstrukturplan in Abb. 67. In der ersten Spalte stehen die Arbeitspakete, während rechts blaue Balken die Dauer, Start und Ende der Arbeitspakete visualisieren. Gantt-Charts fokussieren also als Dimension die Ausführung von Arbeitspaketen und fixieren dazu den Detaillierungsgrad der Arbeitspakete aus den Strukturplänen.

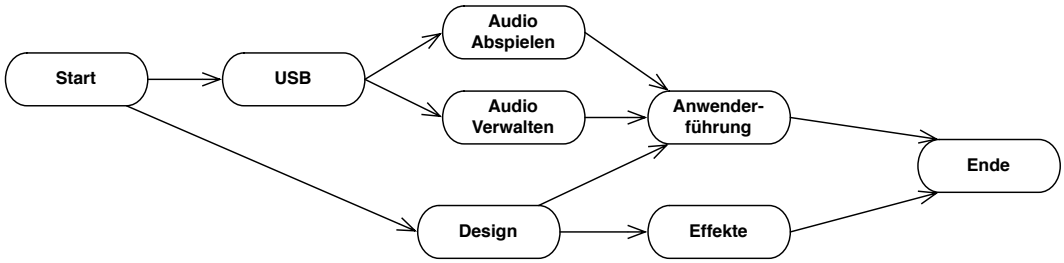


Abb. 69: Netzplan für den Multimedia Player: Der Netzplan zeigt die Abhängigkeiten zwischen den Arbeitspaketen. Zunächst muss die Übertragung der Daten per USB umgesetzt werden, dann können die Audiodaten abgespielt und verwaltet werden. Während dieser Aktivitäten, kann bereits das Design der Benutzerschnittstelle starten. Steht das Design der Benutzerschnittstelle fest und können Audiodaten verwaltet und abgespielt werden, kann die Führung der Anwender implementiert werden. Für die Entwicklung der Effekte ist der Entwurf der Benutzerschnittstelle Vorraussetzung. Sind die Arbeitspakete Anwenderführung und Effekte erledigt, ist der Plan abgearbeitet.

Netzpläne

Netzpläne [Zimmermann & Stache 2001] verwenden die Arbeitspakete des Projektstrukturplans mit demselben Detaillierungsgrad und zeigen für diese Abhängigkeiten zwischen Arbeitspaketen in einem Graphen. Die Knoten dieses Graphs stellen Arbeitspakete, die Kanten Abhängigkeiten zwischen 2 Arbeitspaketen dar. Abb. 69 zeigt als Beispiel einen Netzplan für die Arbeitspakete bei der Erstellung des Multimedia Players. Für die Zeitplanung kann der Netzplan für Arbeitspakete Abschätzungen für die erwartete Dauer sowie früheste und späteste Start- und Endzeiten zeigen.

6.2 Visualisierungen des Buildmanagements

Visualisierungen des Buildmanagements fokussieren, ob nach einer Änderung alle durch das Konfigurationsmanagement kontrollierten Teile kompilieren und ihre Testfälle fehlerfrei durchlaufen.

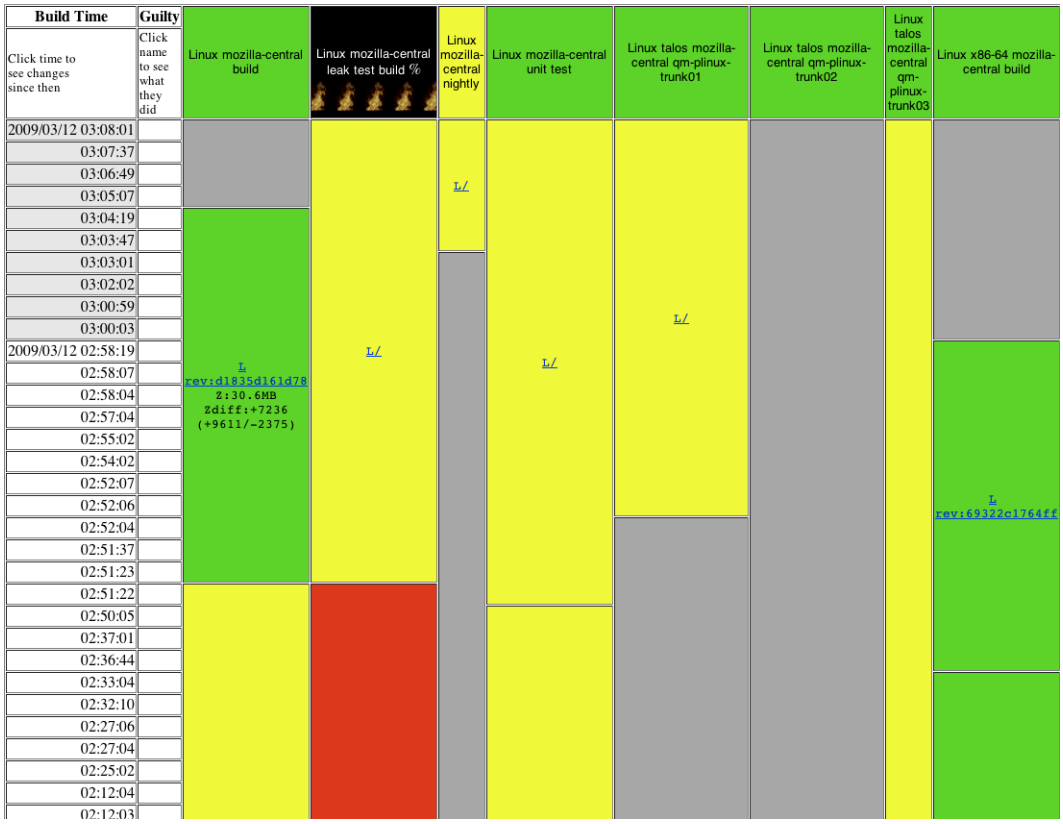


Abb. 70: Visualisierung des Buildprozesses für das Mozilla Projekt mit Tinderbox: Jede Spalte zeigt das Ergebnis des Buildprozesses für eine bestimmte Codebasis und ein festgelegtes Betriebssystem über die Zeit. Rote Balken heißen, dass Fehler beim Kompilieren aufgetreten sind, grüne Balken stehen für erfolgreiche Kompilation und Testläufe. Gelbe Balken zeigen, dass der Buildprozess gerade aktiv ist. [Tinderbox 2009][1]

Tinderbox

Abb. 70 zeigt als Beispiel eine Visualisierung des Buildprozesses mit Tinderbox [Tinderbox 2009]. Tinderbox unterhält mehrere Rechner, die regelmäßig den Quellcode mehrerer Zweige des Konfigurationsmanagements für verschiedene Betriebssysteme kompilieren und testen. Die Ergebnisse visualisiert die Bildschirmkopie aus Abb. 70: Für jede Konfiguration, das heißt Zweig und Betriebssystem, steht eine Spalte zur Verfügung. Die Zeilen zeigen die Entwicklung über die Zeit, wobei eine Version jeweils durch ein farbiges

[1] Bei schwarz-weißem Druck erscheint grün als hellgrau, rot als dunkelgrau und gelb als weiß.

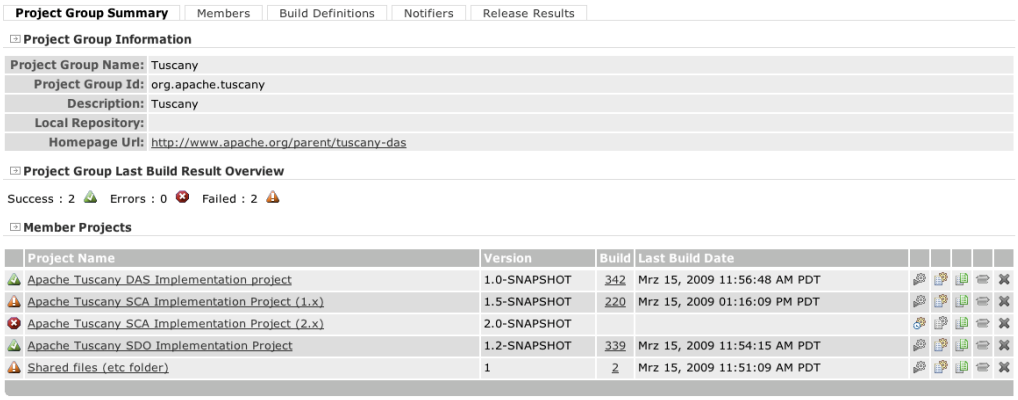


Abb. 71: Bildschirmkopie des Buildmanagements mit Continuum: Für Projekte werden Ergebnisse des Buildprozesses angezeigt. [Continuum 2008]

Rechteck markiert ist. Dieses farbige Rechteck enthält die Versionsnummer und zeigt den Zustand des *builds* an: Grün steht für ein fehlerfrei kompiliertes und getestetes System, orange bedeutet das System kompiliert, Tests schlagen jedoch fehl und rot zeigt an, dass bereits beim Kompilieren Fehler auftreten. Läuft der Buildprozess noch, ist das Rechteck gelb markiert.

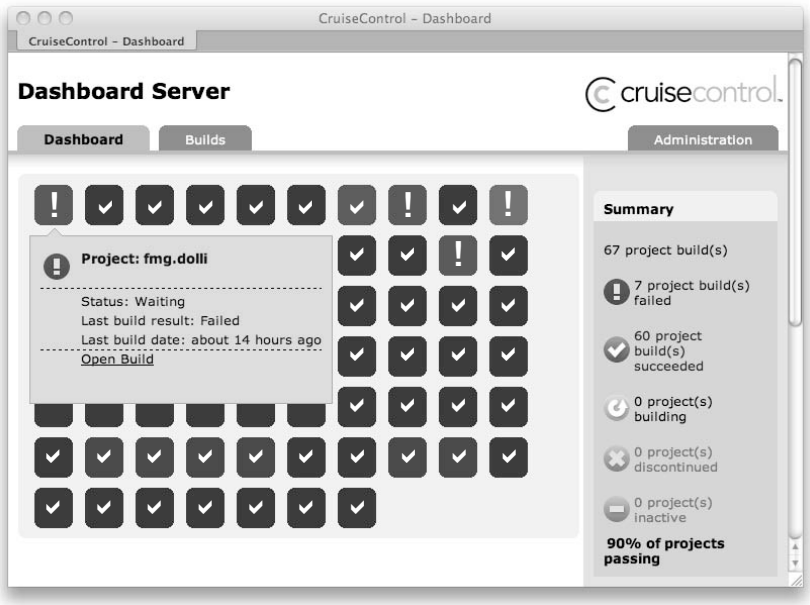


Abb. 72: Bildschirmkopie von Cruise Controll aus dem Dolli Projekt: Die Übersicht gibt an, welche *Builds* von Komponenten des Systems erfolgreich waren und welche fehlgeschlagen sind.

Continuum

Continuum [Continuum 2008] (siehe Abb. 71) und Cruise Control (siehe Abb. 72) zeigen eine Übersicht über den aktuellen Zustand des Buildprozesses für die Komponenten eines Projekts. Die Visualisierung aller drei Komponenten ist auf den Quellcode einer Komponente fokussiert und liefert als Ausgaben zusätzlich Ergebnisse des Compilers. Für das Freigabemanagement wichtige Dimensionen aus der Planung und Modellierung des Systems berücksichtigen diese Visualisierungen nicht.

Dimensionen der Visualisierungen

Visualisierungen des Buildmanagements reflektieren also lediglich den aktuellen Zustand einer Komponente unabhängig von weiteren Dimensionen des Entscheidungsraums, beispielsweise dem Entwicklungsfortschritt, Freigabeplänen und Freigabedeskriptoren. Auch die in Abschnitt 6.1 vorgestellten Visualisierungen der Projektplanung fokussieren nur eine Dimension: Projektstrukturpläne die Aufteilung eines großen Projekts in handhabbare Arbeitspakete, Gantt-Charts den zeitlichen Ablauf der Erstellung der Arbeitspakete und Netzpläne Abhängigkeiten unter den Arbeitspaketen. Diese Planungstechniken haben ihren Ursprung in Belegungsplänen für Maschinen und können mit Änderungen der übrigen Dimensionen nicht umgehen. Agile Vorgehensmodelle erlauben jedoch Änderungen und stellen den Freigabemanager vor die Entscheidung, wie mit einer Änderung umgegangen werden soll. Diese ist dann vor dem Hintergrund aller Dimensionen des beschriebenen Entscheidungsraums zu treffen. Um diesen mehrdimensionalen Entscheidungsraum zu durchsuchen, führt diese Dissertation eine dreidimensionale Sicht dieses Entscheidungsraums ein. Mit dieser dreidimensionalen Visualisierung kann der Freigabemanager drei Dimensionen frei durchlaufen, während die übrigen Dimensionen festgehalten werden.

6.3 Sicht einer Roadmap

Dreidimensionale Sicht einer Roadmap

Dieser Abschnitt beschreibt eine dreidimensionale Sicht des mehrdimensionalen Navigationsraums für dynamische Roadmaps des Freigabemodells. Das Beispiel in Abb. 73 wählt als erste Dimension die in der Roadmap definierte Folge von Freigaben, als zweite Dimension Alternativen zu den Freigaben und als dritte Dimension die Änderungen der Freigaben und Alternativen in Versionen.

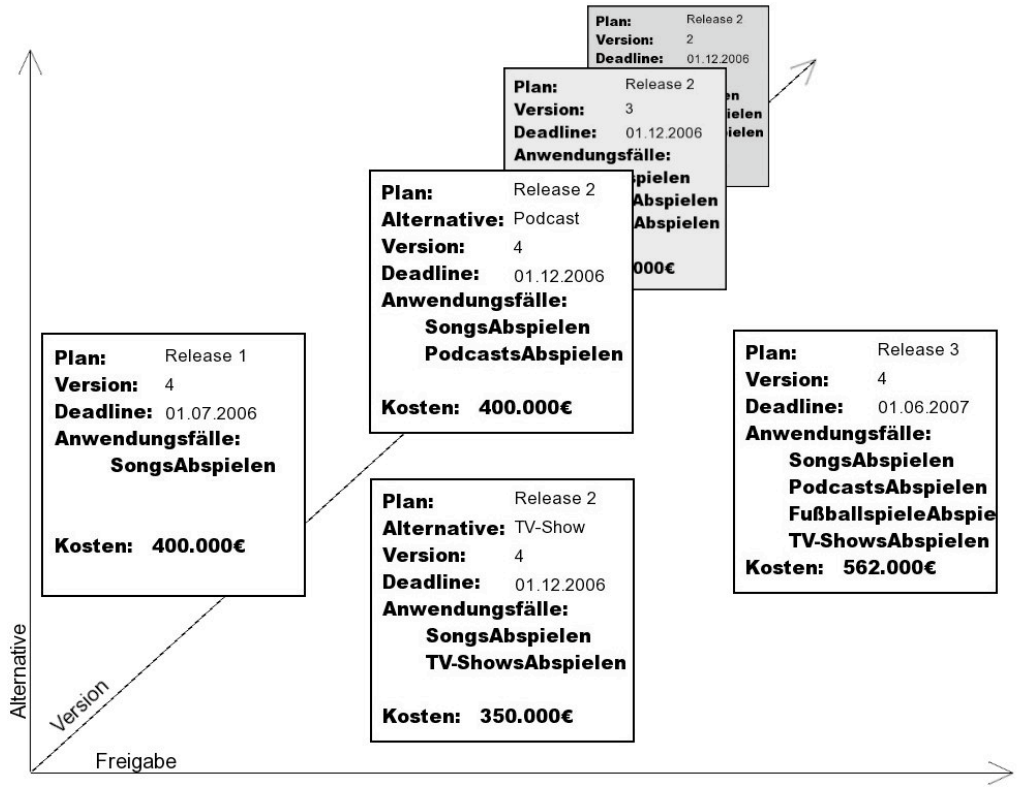


Abb. 73: Dreidimensionale Sicht einer Roadmap zur Visualisierung der Dimensionen Freigaben, Alternativen und Versionen des mehrdimensionalen Entscheidungsraums.

Dimensionen

Die Freigabe-Achse zeigt in horizontaler Richtung von links nach rechts die Folge von Freigaben an, wie sie eine Roadmap definiert. Freigaben, die sich weiter links befinden, werden vor Freigaben geliefert, die weiter rechts stehen. Stehen für eine Freigabe mehrere alternative Freigabeszenarios zur Auswahl, werden diese Freigabeszenarios auf der Achse Alternative in vertikaler Richtung angeordnet. Hat sich der Freigabemanager bereits für eine Alternative entschieden, wird diese auf gleicher Höhe mit den Freigaben angezeigt. Ansonsten ist die Anordnung der Alternativen untereinander beliebig. Änderungen an Freigaben und deren Alternativen über die Zeit verwaltet das Konfigurationsmanagement in Versionen. Die Achse Version geht in den Raum und erlaubt dem Anwender Versionen von Freigaben und Alternativen aus der Vergangenheit anzusehen.

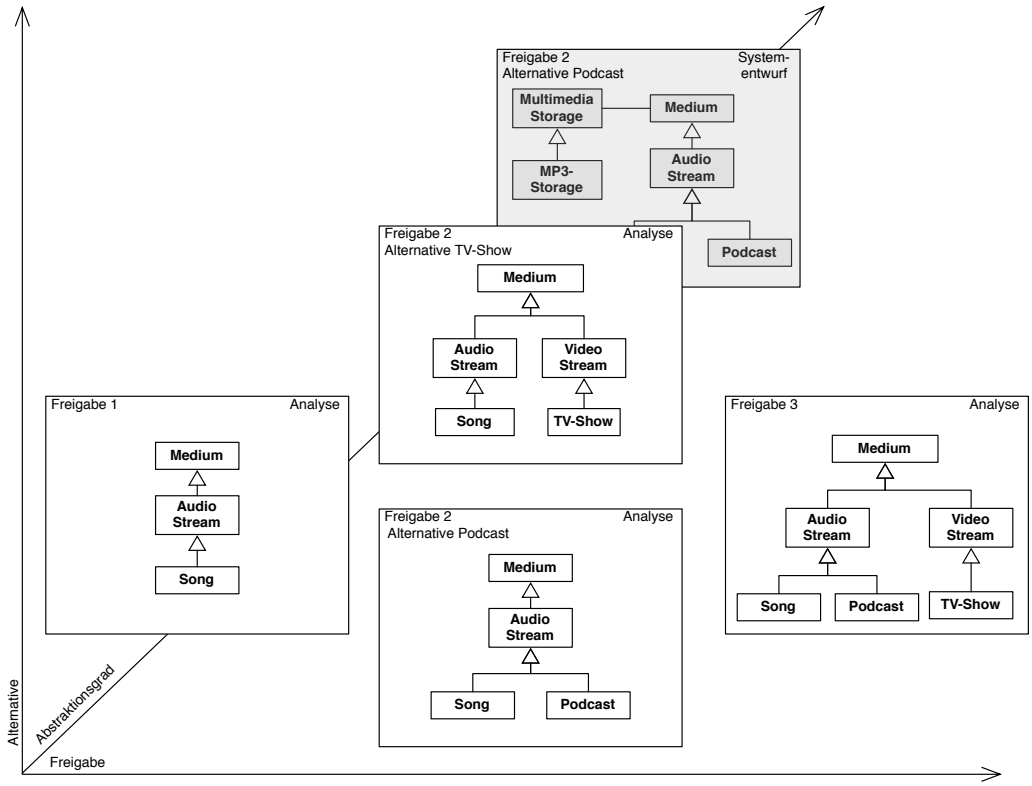


Abb. 74: Dreidimensionale Visualisierung des mehrdimensionalen Entscheidungsraums für das Objektmodell des Multimedia Players: Auf der horizontalen Achse wird die Entwicklung des Objektmodells über Freigaben hinweg angezeigt, die vertikale Achse visualisiert das Objektmodell alternativer Freigabeszenarios und durch die Navigation in den Raum werden die Objektmodelle Richtung Code verfeinert.

6.4 Sichten für Freigabedeskriptoren

Dreidimensionale Sicht eines Freigabedeskriptors

Der mehrdimensionale Entscheidungsraum enthält Freigabedeskriptoren für die Systemmodellierung aufeinanderfolgender und alternativer Freigaben. Eine dreidimensionale Sicht kann Freigabedeskriptoren visualisieren, indem die Zustände der Systemmodelle Dimensionen zugewiesen werden. Das Beispiel einer dreidimensionalen Sicht für das Objektmodell des Multimedia Players in Abb. 74 verwendet die Dimensionen Freigabe, Alternativen und Abstraktionsgrade.

Dimensionen

Die erste Dimension zeigt die Entwicklung des Objektmodells über mehrere aufeinanderfolgende Freigaben hinweg. Die Systemmodelle der Freigaben

sind gemäß der in der Roadmap definierten Ordnung so auf der horizontalen Achse angeordnet, dass die Modelle früher gelieferter Freigaben weiter links stehen. Eine Navigation in vertikaler Richtung erlaubt als zweite Dimension, Objektmodelle alternativer Freigabeszenarios für die in der Mitte fokussierte Freigabe zu vergleichen. Die dritte Dimension fokussiert die Entwicklung des Systemmodells über Abstraktionsgrade, wobei eine Navigation in den Raum den Modellen mehr Details der Implementierung hinzufügt und eine Navigation aus dem Raum heraus von den Modellen abstrahiert. Der Freigabemanager gelangt in Abb. 74 folglich durch eine Navigation in den Raum vom Analyse-Objektmodell zum Objektentwurfs-Objektmodell.

6.5 Vergleichsansicht

Vergleichsansicht

Während der mehrdimensionale Entscheidungsraum einen Überblick über die Zustände der Systemmodelle verschiedener Knowledge Nuggets gibt,

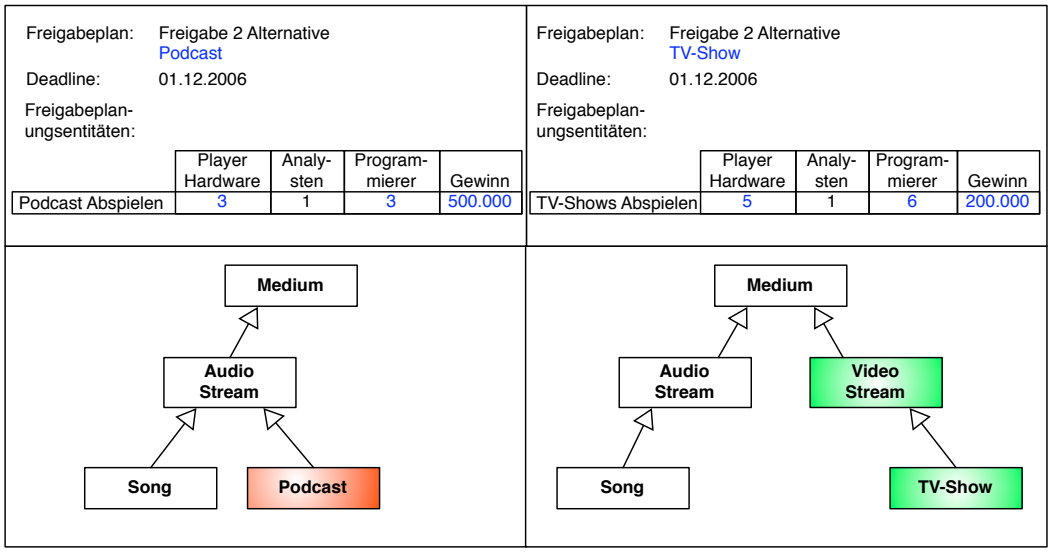


Abb. 75: Vergleichsansicht für Freigabepläne und Modelle: Ausgewählte Freigabepläne können zusammen mit ihren Freigabedeskriptoren verglichen werden, wobei Unterschiede in den Modellen und Plänen farblich hervorgehoben werden. Die Klasse Podcast ist rot markiert. Rot markierte Entitäten existieren in der linken Freigabe, aber nicht in der rechten, wohingegen grün markierte nur in der rechten Freigabe existieren. Dementsprechend sind die Klassen Video Stream und TV-Show grün markiert. Blau markierte Entitäten existieren in beiden Freigaben, haben sich jedoch verändert: So ist die benötigte Player Hardware, die Anzahl der Programmierer und der Gewinn neben dem Namen des Freigabeplans blau geschrieben.

erlaubt eine Vergleichsansicht den direkten Vergleich von Freigabeplänen und Modellen alternativer oder aufeinanderfolgender Freigaben. Die Vergleichsansicht besteht aus Reihen und Spalten. Eine Spalte stellt eine Dimension dar, beispielsweise eine Alternative, Abstraktionsgrad oder Version, und kann beliebig viele Reihen enthalten. Eine Reihe zeigt in jeder Spalte stets die gleiche Entität, beispielsweise einen Freigabeplan für alle Alternativen.

Vergleichsansicht für den Multimedia Player

Abb. 75 zeigt als Beispiel eine Vergleichsansicht für 2 alternative Freigabeszenarios der Freigabe 2 des Multimedia Players: Links steht die Alternative Podcast, rechts die alternative TV-Show. In Abb. 75 werden oben die Freigabepläne und unten die zugehörigen Modelle jeweils von links nach rechts verglichen. Die Vergleichsansicht hebt Unterschiede farblich hervor:Entitäten, die auf beiden Seiten existieren und sich unterscheiden, werden blau markiert, wie im Beispiel aus Abb. 75 die Abschätzung des Gewinns, der durch eine Freigabeplanungsentität erzielt werden soll. Rot hinterlegte Entitäten, beispielsweise die Klasse Podcast in Abb. 75, kommen nur auf der linken Seite vor und existieren nicht auf der rechten. Grün hinterlegte Entitäten sind neu im rechten Modell, die Klassen Video Stream und TV-Show in Abb. 75 sind nur bei einer Entscheidung für die Alternative TV-Show nötig.

6.6 Visualisierung einer Freigabe

Visualisierung einer Freigabe

Die Visualisierung einer Freigabe gibt dem Freigabemanager einen Überblick über Entitäten, die seine Entscheidung beeinflussen. Zudem bietet sie einen Einstieg in Systemmodelle und Begründungen aus Kommunikationsmodellen sowie eine Matrix zur Abwägung zwischen alternativen Freigabeszenarios.

Visualisierung einer Freigabe des Multimedia Players

Abb. 76 zeigt als Beispiel eine Visualisierung der zweiten Freigabe des Multimedia Players. Sie enthält als Einstieg in die Systemmodellierung Dokumente und Diagramme, die Knowledge Nuggets dieser Freigabe verwalten. Zudem stellt ein Knowledge Nugget für seine Systemmodellelemente alle Begründungen aus Kommunikationsmodellen bereit. Eine Freigabe fasst Begrün-

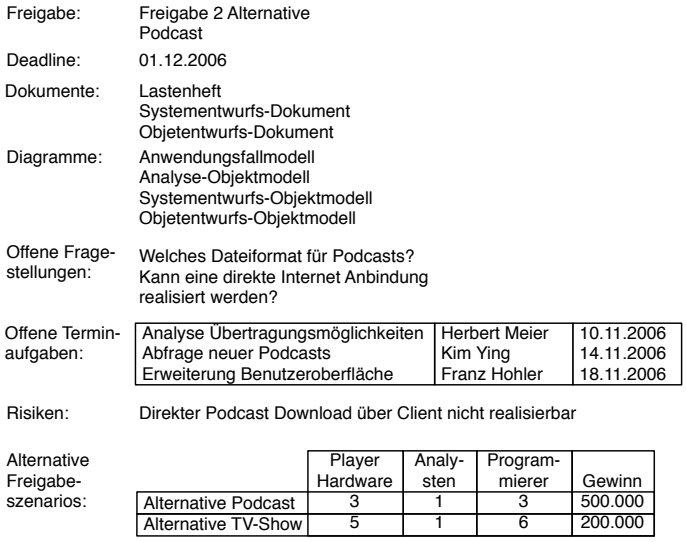


Abb. 76: Visualisierung der zweiten Freigabe des Multimedia Players: Die Freigabe zeigt Dokumente und Diagramme als Einstieg in die Systemmodelle sowie offene Fragestellungen, Terminaufgaben und Risiken als Begründungen aus Kommunikationsmodellen. Eine Matrix bewertet alternative Freigabeszenarios nach Kriterien.

ungen aus Kommunikationsmodellen ihrer Knowledge Nugget zusammen und stellt offene Fragestellungen, offene Terminaufgaben und Risiken sowie Fehlerberichte dar. Alternativen zu der gewählten Freigabe visualisiert eine Abwägungsmatrix, die in den Zeilen die Alternativen zeigt und in den Spalten diese Alternativen nach Kriterien bewertet, deren Bewertungen das begründungsbasierte Freigabemodell berechnet.

6.7 Nachvollziehbarkeits-Graph

Nachvollziehbarkeits-Graph

Die beschriebenen dreidimensionalen Sichten beschränken die dargestellten Dimensionen des mehrdimensionalen Entscheidungsraums. Der Freigabemanager kann beliebig tief in die ausgewählten Dimensionen navigieren, während andere Dimensionen ausgeblendet werden. Möchte der Freigabemanager alle Dimensionen gleichzeitig betrachten, kann er den Nachvollziehbarkeits-Graph [Wolf 2007] verwenden. Der Nachvollziehbarkeits-Graph zeigt für eine in der Mitte fokussierte Entität ungefiltert alle anderen abhängigen Entitäten.

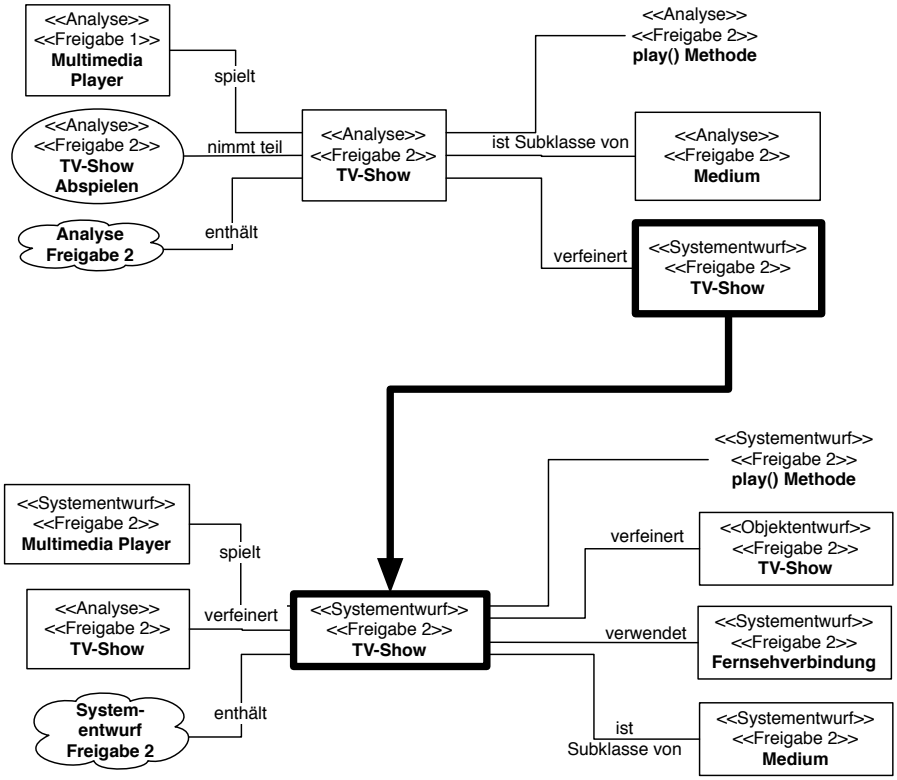


Abb. 77: Nachvollziehbarkeit im begründungsbasierten Freigabemodell: Der Knowledge Nugget der Analyse der Freigabe 2 enthält eine Klasse TV-Show. Diese verfeinert eine weitere Klasse TV-Shows, die zum Systementwurf der Freigabe 2 gehört. Im Gegensatz zur Klasse TV-Show des Analysemodells verwendet die verfeinerte Klasse im Systementwurfsmodell eine Fernsehverbindung.

Nachvollziehbarkeits-Graph für den Multimedia Player

Abb. 77 zeigt zwei Beispiele für Nachvollziehbarkeits-Graphen für den Multimedia Player. Der Freigabemanager kann von der Klasse Multimedia Player über die Abhängigkeit „spielt“, von dem Anwendungsfall TV-Show Abspielen über die Abhängigkeit „nimmt teil“ und von dem Knowledge Nugget Analyse Freigabe 2 über die Abhängigkeit enthält zu der Klasse TV-Show gelangen. TV-Show hat weitere Abhängigkeiten zu seiner Methode play(), zu einer verfeinerten Klasse TV-Show sowie zu seiner Oberklasse Medium. Die untere Visualisierung in Abb. 77 betrachtet Abhängigkeiten der verfeinerten Klasse TV-Show: Man sieht, dass der Knowledge Nugget Systementwurf Freigabe 2 die Klasse TV-Show enthält. TV-Show hat eine Methode play(), verwendet eine Fernsehverbindung und ist Subklasse der Klasse Medium. Zudem

verfeinert wiederum eine Klasse TV-Show des Objektentwurfs-Objektmodells die Klasse TV-Show des Systementwurf-Objektmodells.

7 EXPLoRE

EXPLoRE (*EXploring and PLanning REleases*) ist ein Werkzeug, das BEEF implementiert. Dieses Kapitel beschreibt die Architektur und Realisierung von EXPLoRE.

EXPLoRE

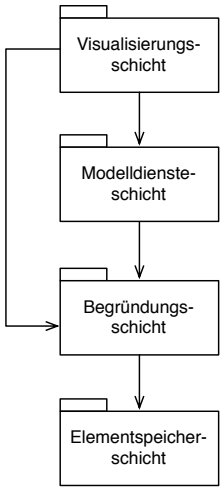


Abb. 78: EXPLoRE verwendet eine offene Schichtenarchitektur bestehend aus Visualisierungs-, Modelldienste-, Begründungs- und Elementspeicherschicht.

EXPLoRE basiert auf einer offenen Schichtenarchitektur, die aus vier Schichten besteht: Visualisierungsschicht, Modelldiensteschicht, Begründungsschicht und Elementspeicherschicht. (siehe Abb. 78) Die Visualisierungsschicht stellt in der grafischen Benutzerschnittstelle Begründungen aus BEEF zur Verfügung und interagiert mit dem Anwender. Sie verwendet für die Visualisierung aufbereitete Daten der Modelldiensteschicht und die Begründungsschicht. Die Begründungsschicht enthält Begründungen für Entscheidungen im Freigabemanagement und das begründungsbasierte

Schichten-architektur

Freigabemodell bestehend aus Knowledge Nuggets, System-, Organisations- und Kommunikationsmodellen. Die Elementspeicherschicht übernimmt die persistente Datenspeicherung der Begründungen aus der Begründungsschicht.

7.1 Visualisierungsschicht

Die Visualisierungsschicht stellt die Benutzerschnittstelle von EXPLoRE bereit, die in diesem Abschnitt vorgestellt wird.

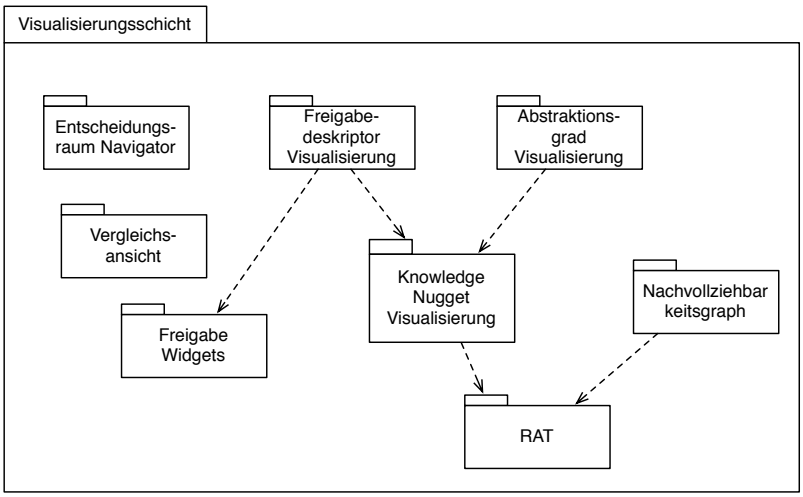
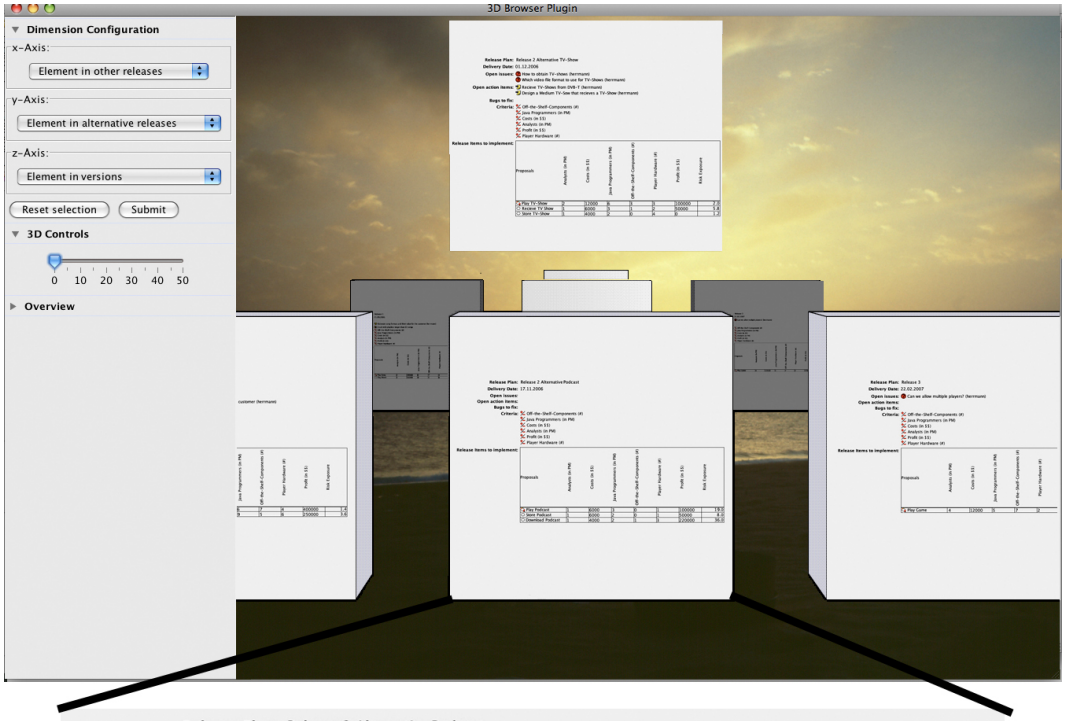


Abb. 79: Die Visualisierungsschicht besteht aus den Subsystemen Entscheidungsraum Navigator, Vergleichsansicht, Freigabe Widgets, Freigabedeskriptor, Abstraktionsgrad, Knowledge Nugget Visualisierung sowie RAT.

Subsystemzerlegung

Abb. 79 zeigt die Subsystemzerlegung der Visualisierungsschicht. Sie besteht aus dem Entscheidungsraum Navigator (siehe 7.1.1), Freigabe Widgets (siehe 7.1.2) und der Vergleichsansicht (siehe 7.1.3). Die Subsysteme Freigabedeskriptor Visualisierung, Abstraktionsgrad Visualisierung und Knowledge Nugget Visualisierung (siehe 7.1.5) stellen eine Benutzerschnittstelle für Freigabedeskriptoren und Abstraktionsgrade dar. Die Dienste der Visualisierungsschicht basieren auf dem *Rationale-based Analysis Tool* (RAT) [Wolf & Dutoit 2004].



Release Plan: Release 2 Alternative Podcast
Delivery Date: 17.11.2006
Open issues:
Open action items:
Bugs to fix:
Criteria:
- Off-the-Shelf-Components (#)
- Java Programmers (in PM)
- Costs (in $$)
- Analysts (in PM)
- Profit (in $$)
- Player Hardware (#)

Release Items to implement:

Proposals	Analysts (in PM)	Costs (in $$)	Java Programmers (in PM)	Off-the-Shelf-Components (#)	Player Hardware (#)	Profit (in $$)	Risk Exposure
Play Podcast	1	6000	3	0	1	100000	19.0
Store Podcast	1	6000	2	0	1	50000	8.0
Download Podcast	1	4000	2	1	3	220000	36.0

Abb. 80: Bildschirmkopie des Entscheidungsraum Navigators für drei Freigabepläne des Multimedia Players: Auf der x-Achse sind die Freigabe 1, 2 und 3 angeordnet, auf der y-Achse alternative Freigabeszenarios und auf der z-Achse die Zustände der Freigaben in früheren Versionen.

7.1.1 Entscheidungsraum Navigator

Der Entscheidungsraum Navigator verwendet Java 3D [Sowizral et al. 1998], um die Navigation durch den mehrdimensionalen Entscheidungsraum (siehe

Kapitel 6) zu realisieren. Die dreidimensionale Visualisierung des Entscheidungsraums basiert auf einem dreidimensionalen Gitter, das Begründungen für die drei Dimensionen enthält und vom Modellraum Subsystem in der Modelldiensteschicht generiert wird (siehe Abschnitt 7.2). Als Dimensionen kann der Anwender zwischen Version, Freigabe, alternative Freigabeszenarios und Abstraktionsgrad auswählen und mit ihnen die Achsen des Entscheidungsraum Navigators belegen.

Unterstützung des Freigabemanagers

Der Entscheidungsraum Navigator kann den Freigabemanager unterstützen, um zwischen Freigabeplänen aufeinanderfolgender und alternativer Freigaben zu wechseln und für diese Zustände frühere Versionen zu vergleichen. Abb. 80 zeigt eine Bildschirmkopie zur Navigation durch die Freigabepläne des Multimedia Players. Der Multimedia Player wird in drei aufeinanderfolgenden Freigaben entwickelt, wobei für die zweite Freigabe ein Freigabeszenario existiert, das TV-Shows und Podcasts gegeneinander erkundet. Als Dimensionen sind für die x-Achse Freigaben, die y-Achse alternative Freigabeszenarios, und in der z-Achse Versionen ausgewählt. Durch eine Navigation in horizontaler Richtung zeigt der Entscheidungsraum Navigator die Freigabepläne einer Roadmap. Eine Navigation in Richtung der y-Achse zeigt Freigabepläne alternativer Freigabeszenarios, beispielsweise liegen in Abb. 80 die Freigabepläne der Freigabe 2 Alternative Podcast und Freigabe 2 Alternative TV-Show auf der y-Achse. Durch eine Bewegung in den Raum gelangt der Freigabemanager von neueren Versionen der Freigabepläne zu älteren Versionen und kann so betrachten, wie sich die Freigabepläne während der Zeit entwickelt haben.

Unterstützung weiterer Interessenvertreter

Der Entscheidungsraum Navigator kann neben dem Freigabemanager auch andere Interessenvertreter unterstützen, beispielsweise um den gesuchten Zustand eines Modellelements auszuwählen. Abb. 81 zeigt als Beispiel für die Unterstützung eines Architekten eine Bildschirmkopie des Entscheidungsraum Navigators für die Klasse Medium des Multimedia Players, der mit Modellen der drei Abstraktionsgrade Analyse, Systementwurf und Objektentwurf entwickelt wird. Links oben in der Bildschirmkopie aus Abb. 81 ist ersichtlich, dass die x-Achse mit Freigaben, die y-Achse mit alternativen Freigabeszenarios und die z-Achse mit Abstraktionsgraden belegt ist. Der

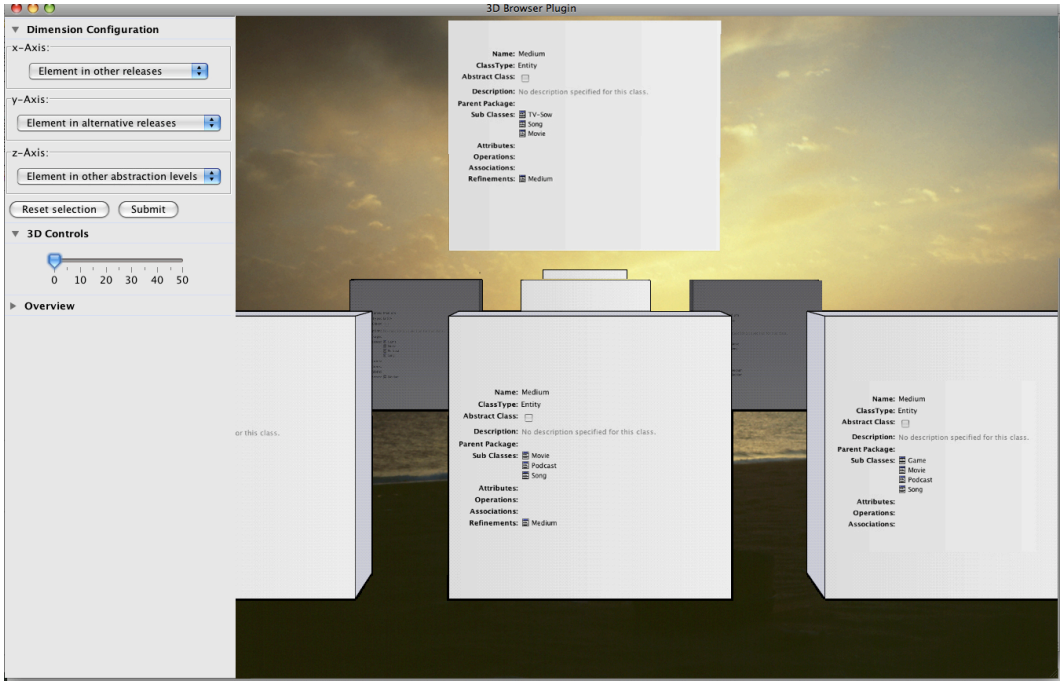


Abb. 81: Bildschirmkopie des Entscheidungsraum Navigators auf die Klasse Medium des Multimedia Players: Die Modelle dreier Abstraktionsgrade und dreier Freigaben verfeinern die Klasse Medium, die auch Bestandteil eines Freigabeszenarios für die zweite Freigabe ist.

Entscheidungsraum Navigator zeigt im Vordergrund die Klasse Medium auf dem Abstraktionsgrad Analyse, wobei in der Mitte der Zustand der Freigabe 2, mit den Subklassen Song, Movie und Podcast, dargestellt ist. Folgt man von hier der x-Achse nach rechts, ist der Zustand der Klasse Medium in der Freigabe 3 mit den Subklassen Song, Movie, Podcast und Game dargestellt, während am Ursprung (in der Abbildung verdeckt) der Zustand der Klasse Medium in der Freigabe 1 abgebildet ist. Oberhalb der Freigabe 2 ist die Klasse Medium für das Freigabeszenario TV-Shows mit den Subklassen Song, Movie und TV-Show abgebildet.

7.1.2 Freigabe-Widgets

Freigabe-Widgets

Das Freigabe-Widget Subsystem erzeugt grafische Komponenten für die Benutzeroberfläche zur Darstellung von Freigaben. Die Benutzeroberfläche gliedert die Visualisierung einer Freigabe (siehe Abschnitt 6.6) in drei Abschnitte mit allgemeinen Eigenschaften (*properties*), Modell (*model*) sowie Plan.

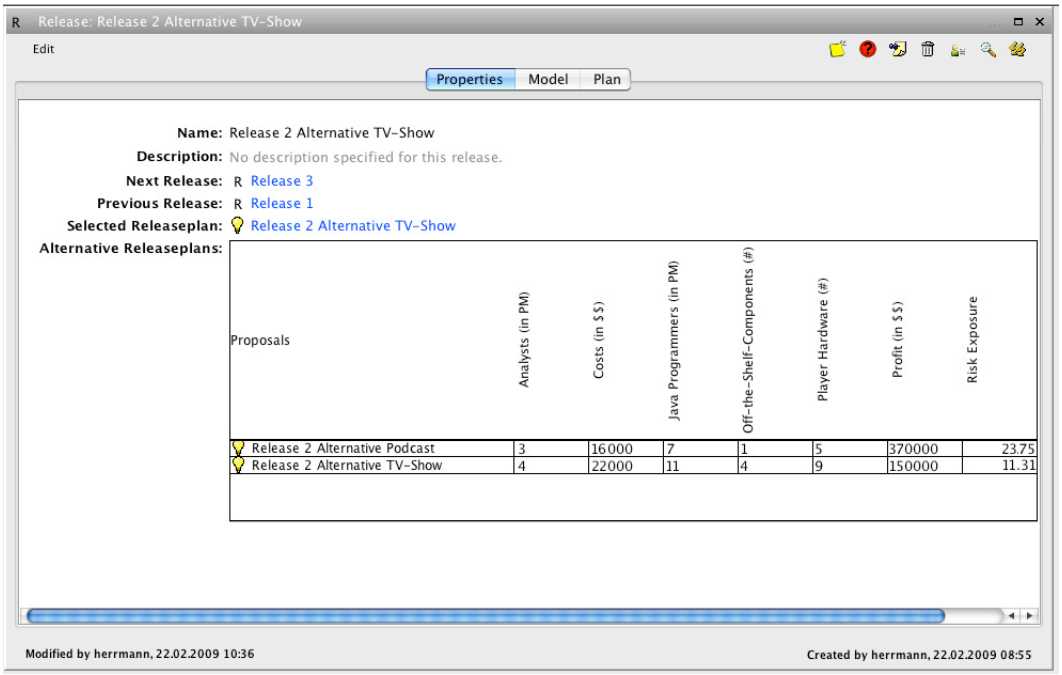


Abb. 82: Bildschirmkopie des Abschnitts Eigenschaften (*properties*) ist Teil der Visualisierung einer Freigabe durch das Widget Subsystem: Sie enthält den Namen, eine Beschreibung sowie die durch die Roadmap definierte nachfolgende und vorangehende Freigabe. Zudem zeigt die Bildschirmkopie eine Matrix zur Bewertung alternativer Freigabeszenarios mit Kriterien.

Darstellung einer Freigabe

Abb. 82 zeigt eine Bildschirmkopie für die allgemeinen Eigenschaften der zweiten Freigabe des Multimedia Players. Sie enthält den Namen der Freigabe zusammen mit einer Beschreibung. Hinzu kommen Informationen aus der in der Roadmap definierten Ordnung auf Freigaben: Die vorangehende Freigabe (*previous release*) und nachfolgende Freigabe (*next relase*). Unter Alternativen sind Freigabeszenarios aufgeführt, wobei als Darstellung eine Matrix gewählt ist, die durch das begründungsbasierte Freigabemodell berechnete Werte für die Bewertung von Freigabeszenarios darstellt.

Darstellung des Freigabedeskriptors

Der Abschnitt Modell (siehe Abb. 83) dient als Einstiegspunkt für den Freigabedeskriptor und zeigt Dokumente (*documents*), Anwendungsfälle (*use cases*) und Objektmodelle (*object models*). Dokumente beschreiben das System einer Freigabe, indem sie die Modellelemente des Freigabedeskriptors strukturiert darstellen. Zudem sind alle Anwendungsfälle zusammengefasst, die eine Freigabe beschreiben, um einen Einstieg in das funktionale Modell einer Freigabe zu geben. Objektmodelle sind Klassendiagramme und bilden

den Einstieg in die Struktur des Systems auf einem bestimmten Abstraktionsgrad.

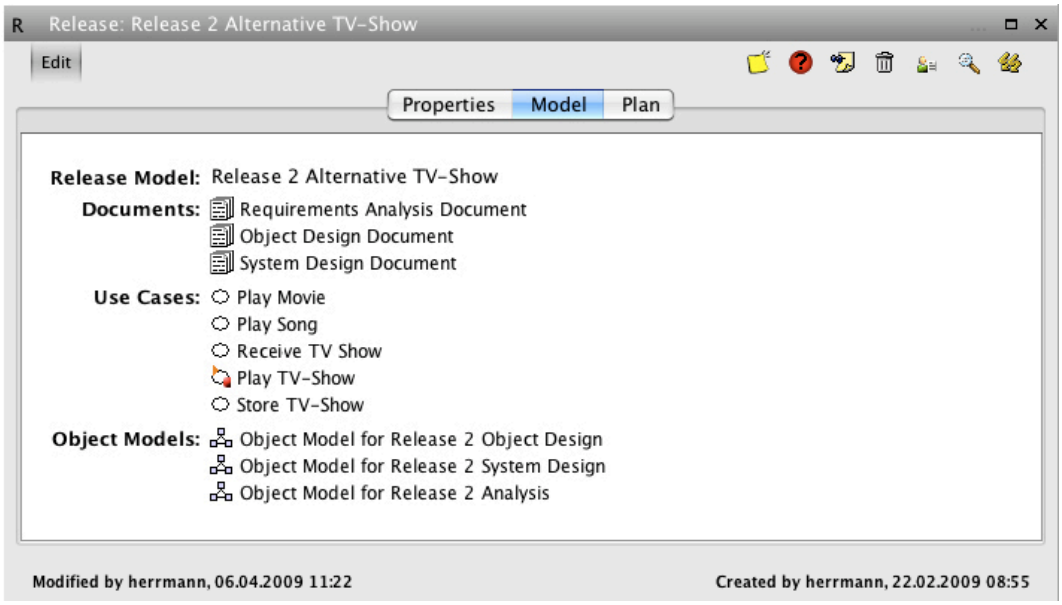


Abb. 83: Bildschirmkopie des Abschnitts Modell (*model*) einer Freigabe: Sie enthält die Dokumente der Freigabe, Objektmodelle sowie alle Anwendungsfälle, die eine Freigabe implementiert.

Darstellung des Freigabeplans

Der Abschnitt Plan in Abb. 84 illustriert den Freigabeplan für die zweite Freigabe des Multimedia Players. Er enthält das Lieferdatum (*delivery date*) für die Freigabe. Offene Fragestellungen (*open issues*), offene Arbeitseinheiten (*open action items*), nicht behobene Fehlerberichte (*bugs to fix*) geben einen Überblick über kritische und offene Punkte bei der Realisierung der Freigabe. Zudem zählt der Abschnitt Freigabeplanungsentitäten (*release items*) auf, die diese Freigabe neu implementieren. Diese Freigabeplanungsentitäten können in einer Matrix nach den unter Kriterien (*criteria*) festgelegten Kriterien bewertet werden. Die Bewertung lässt zum einen den direkten Vergleich von Freigabeplanungsentitäten zu, zum anderen dient sie als Ausgangspunkt für die Bewertung alternativer Freigabeszenarios.

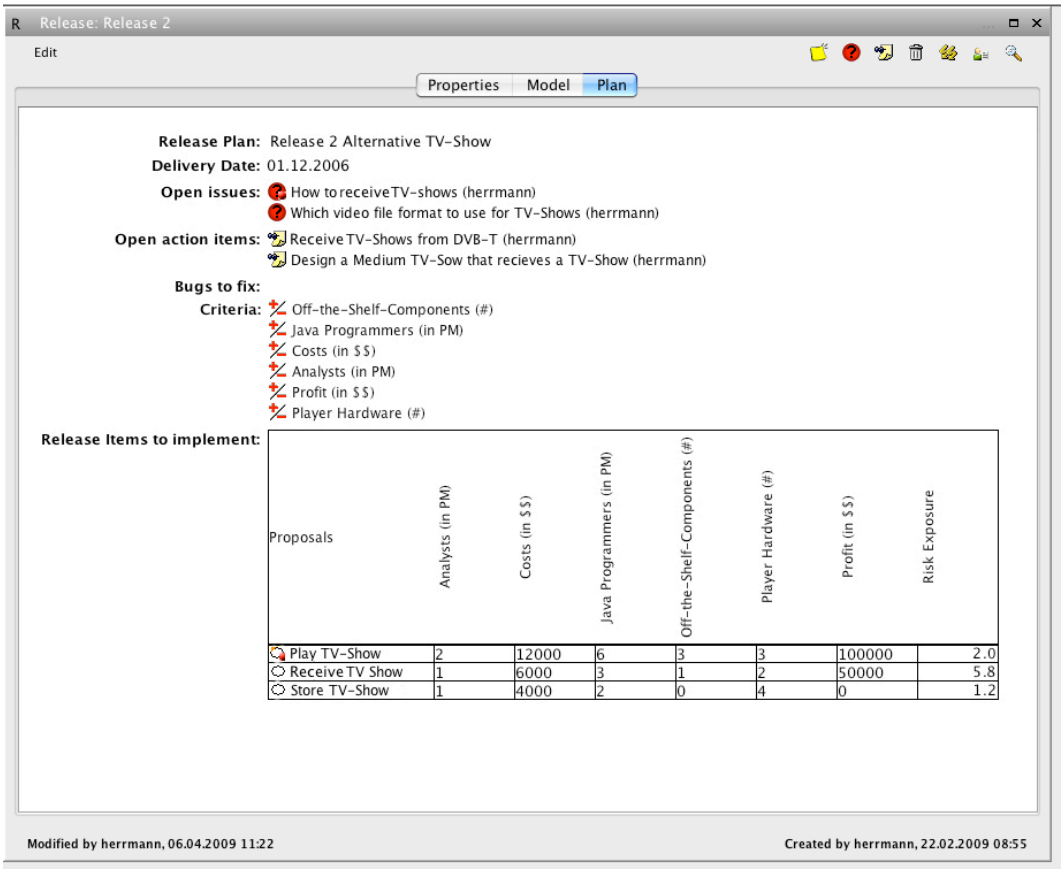


Abb. 84: Bildschirmkopie des Abschnitts Plan einer Freigabe: Sie zeigt den Freigabeplan (*release plan*) bestehend aus Lieferdatum, offenen Fragestellungen (*open issues*), offenen Arbeitseinheiten (*open action items*) sowie nicht behobenen Fehlerberichten (*bugs to fix*). Hinzu kommen Kriterien (*criteria*) zur Bewertung der Freigabeplanungsentitäten sowie eine Matrix, in der noch zu implementierende Freigabeplanungsentitäten (*release items to implement*) nach diesen Kriterien bewertet werden.

7.1.3 Vergleichsansicht

Vergleichsansicht

Die Vergleichsansicht zeigt mehrere Modellelemente in einer Paralleldarstellung mit Dehnstellen [Kelter et al. 2008]. Paralleldarstellungen ordnen die zu vergleichenden Modellelemente, beispielsweise Freigabepläne, in einem Fenster nebeneinander an, wobei sie Korrespondenzen jeweils in der gleichen Zeile anzeigen. Dazu stellt die Vergleichsansicht Modellelemente sequentiell, das heißt ohne die Verwendung von Abschnitten, dar, um in allen Fenstern die gleiche Information anzuzeigen. Damit Unterschiede ermittelt werden können, müssen die Modellelemente in einer Reihe alle von der gleichen Klasse sein.

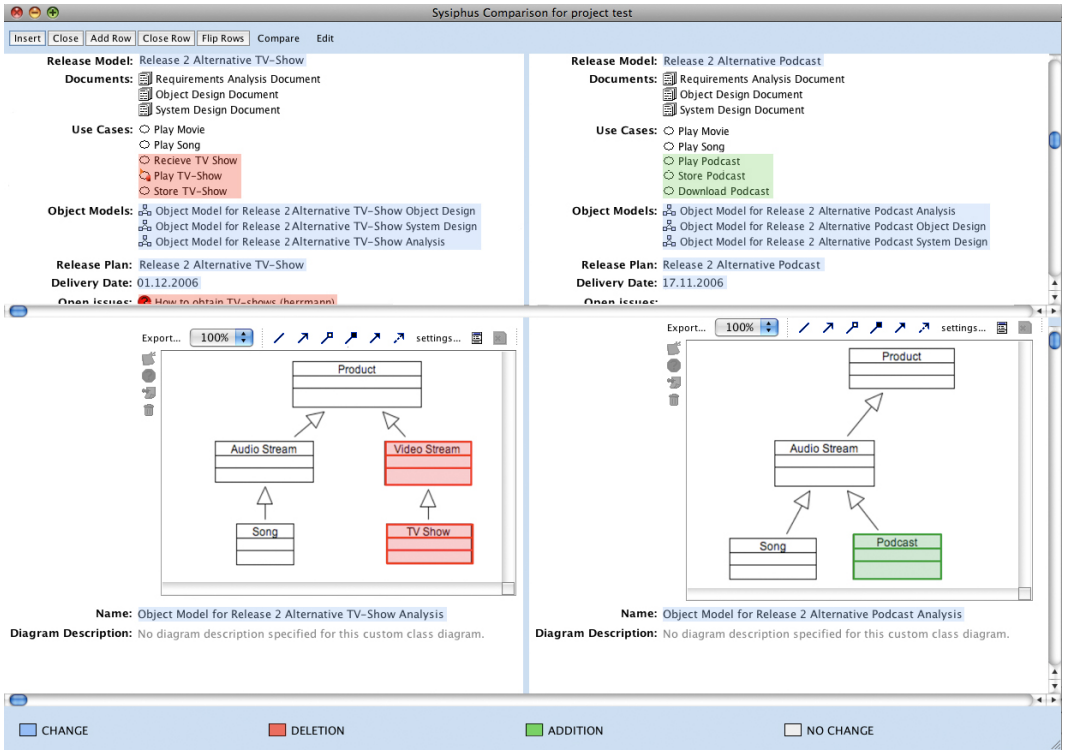


Abb. 85: Vergleichsansicht für zwei alternative Freigabeszenarios der zweiten Freigabe des Multimedia Players: Auf der linken Seite befindet sich das Freigabeszenario und das Analysemodell für die Alternative TV-Show, auf der rechten Seite für die Alternative Podcast.

Unterschiede in Freigaben

Abb. 85 zeigt eine Vergleichsansicht, die zwei Alternativen für die zweite Freigabe des Multimedia Players gegenüberstellt. Auf der linken Seite steht die Alternative TV-Shows, auf der rechten Seite die Alternative Podcast. Die oberste Reihe vergleicht Modelle und Pläne, die untere Reihe die Analyse-Objektmodelle der beiden Freigaben. Unterschiede im Modell sind durch die Farben rot, blau und grün hervorgehoben: Blau markierte Eigenschaften bezeichnen eine Änderung. Beispielsweise unterscheidet sich das Lieferdatum (*delivery date*) beider Alternativen: Die Freigabe 2 Alternative TV-Show ist bis zum 01.12.2006, die Freigabe 2 Alternative Podcast hingegen zum 17.11.2006 zu liefern, daher ist das Lieferdatum beider Freigaben blau markiert. Rot sind Löschungen, das heißt Eigenschaften des linken Modellelements, die beim rechten Modellelement nicht mehr existieren. Im Analyseobjektmodell der Alternative TV-Show aus Abb. 85 sind die Klassen Video Stream und TV-Show rot markiert, weil sie die Alternative Podcast

nicht mehr benötigt. Grün markierte Eigenschaften kommen im rechten Modell im Vergleich zum Linken hinzu: Die Anwendungsfälle Podcast Abspielen (*Play Podcast*), Podcast Herunterladen (*Download Podcast*)und Podcast Speichern (*Store Podcast*) kommen in Abb. 85 in der Freigabe 2 Alternative Podcast neu hinzu und sind deshalb grün markiert.

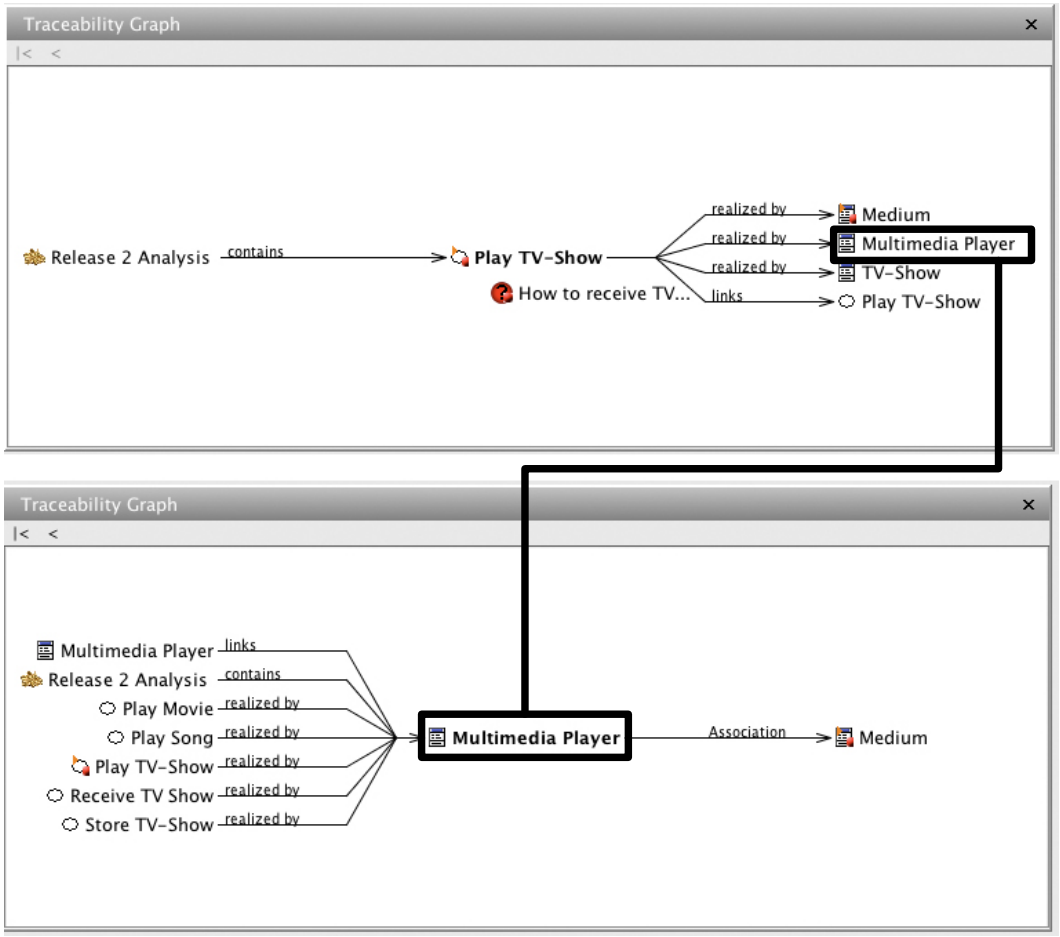


Abb. 86: Bildschirmkopien des Nachvollziehbarkeits-Graphen für den Multimedia Player: Die obere Bildschirmkopie enthält den Anwendungsfall TV-Show Abspielen (*Play TV-Show*) im Knowledge Nugget Release 2 Analyse. Er wird durch die Klassen Medium, Multimedia Player und TV-Show realisiert. Zudem wird er vom Anwendungsfall Play TV-Show, der dem Knowledge Nugget Release 3 Analyse zugeordnet ist, verfeinert. Der Freigabemanager navigiert zur Klasse Multimedia Player, deren Abhängigkeiten die untere Bildschirmkopie fokussiert: Sie gehört zum Knowledge Nugget Release 2 Analyse und ist eine Verfeinerung der Klasse Multimedia des Knowledge Nuggets Release 1 Analyse. TV-Show Abspielen (*Play TV-Show*), TV-Show Empfangen (*Receive TV-Show*) und TV-Show Speichern (*Store TV-Show*) benötigen die Klasse Multimedia Player. Multimedia Player hat eine Assoziation zur Klasse Medium.

7.1.4 Nachvollziehbarkeits-Graph

Nachvollziehbarkeits-Graph

Der Nachvollziehbarkeits-Graph [Wolf 2007] ist ein Werkzeug, um Abhängigkeiten zwischen Modellelementen des begründungsbasierten Freigabemodells zu verfolgen. Durch sie kann der Freigabemanager Systemmodelle über Abstraktionsgrade hinweg durchlaufen. Dieser Abschnitt zeigt dies am Beispiel von Bildschirmkopien des Nachvollziehbarkeits-Graphen für ein Projekt zur Entwicklung des Multimedia Players in drei Freigaben.

Abhängigkeiten im Nachvollziehbarkeits-Graph

Der Freigabemanager betrachtet in der oberen Bildschirmkopie des Nachvollziehbarkeits-Graphen (*traceability graph*) in Abb. 86 Abhängigkeiten des Anwendungsfalls TV-Show Abspielen (*Play TV-Show*), der in der Mitte der Abbildung steht. Direkt unter dem Anwendungsfall sind Kommunikationsmodellelemente abgebildet, in Abb. 86 die offene Fragestellung *„How to receive TV-Shows?“*, um über Probleme zu informieren. Der Knowledge Nugget Release 2 Analyse auf der linken Seite enthält (*contains*) den Anwendungsfall TV-Abspielen. Auf der rechten Seite des Anwendungsfalls, sieht der Freigabemanager die partizipierenden Klassen Medium, Multimedia Player und TV-Show sowie eine Verfeinerung des Anwendungsfalls TV-Show Abspielen, die zum Freigabedeskriptor der Freigabe 3 gehört.

Der Freigabemanager steigt durch die Wahl der Klasse Multimedia Player auf der rechten Seite in das Objektmodell ein. Die untere Bildschirmkopie des Nachvollziehbarkeits-Graphen in Abb. 86 betrachtet die Abhängigkeiten der in der Mitte der Bildschirmkopie dargestellten Klasse Multimedia Player: Sie gehört zum Knowledge Nugget, der die Freigabe 2 Analyse beschreibt. Die im Fokus stehende Klasse Multimedia Player ist eine Verfeinerung einer weiteren Klasse Multimedia Player, die auf der linken Seite zu finden ist und dem Knowledge Nugget der Freigabe 1 Analyse angehört. Der Freigabemanager sieht, dass die Anwendungsfälle TV-Show Abspielen (*Play TV-Show*), TV-Show Empfangen (*Receive TV-Show*) und TV-Show Speichern (*Store TV-Show*) durch die Klasse Multimedia Player realisiert werden. Das heißt, dass sich ein Fehler in der Klasse Multimedia Player auf diese Anwendungsfälle auswirken kann. Der Freigabemanager kann auf der rechten Seite zur assoziierten Klasse Medium weiternavigieren.

7.1.5 Knowledge Nugget Visualisierung

Knowledge Nugget Visualisierung

Das Subsystem Knowledge Nugget Visualisierung stellt grundlegende Komponenten für die Visualisierung von Systemmodellen bereit, die durch mehrere Knowledge Nuggets verfeinert sind. Dabei arbeitet der Anwender mit den Markierungen der Knowledge Nuggets, das heißt der Anwender betrachtet ein Modell auf dem Abstraktionsgrad Analyse einer spezifischen Freigabe. Beispielsweise gibt der Anwender beim Erzeugen eines Systemmodellelements seine Markierungen an, das heißt eine Freigabe und einen Abstraktionsgrad, um das Systemmodell zu spezifizieren, zu dem das neue Modellelement gehört.

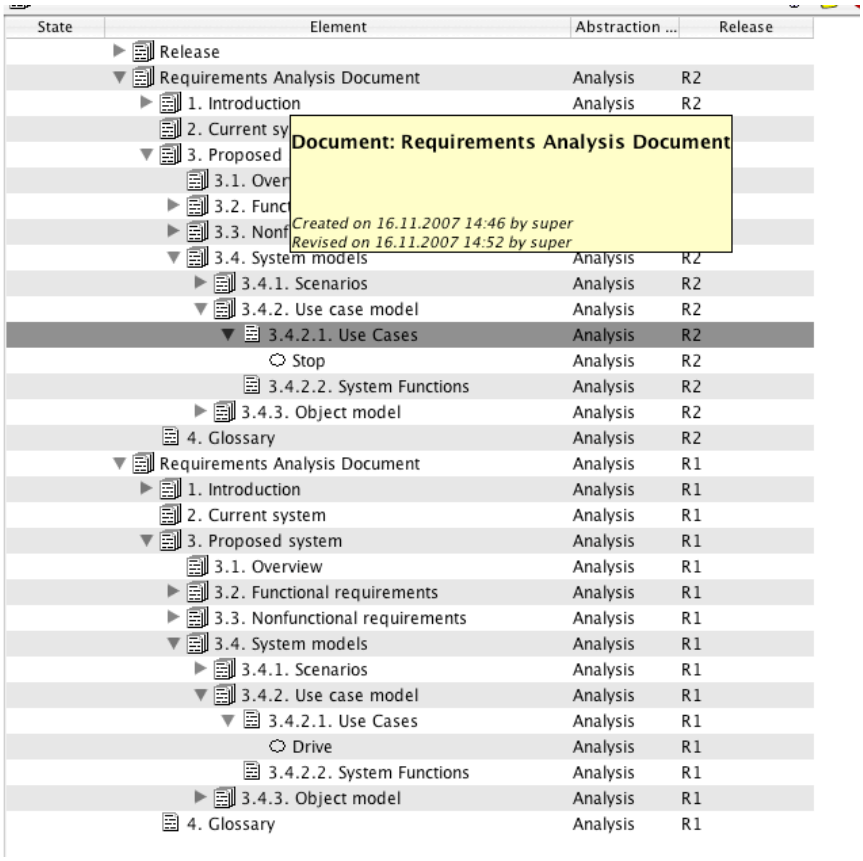


Abb. 87: Bildschirmkopie der Dokumentenansicht mit zwei Anforderungsanalyse-Dokumenten (*Requirements Analysis Document*): Das erste Dokument gehört zum Knowledge Nugget Analyse R2, das zweite Dokument zum Knowledge Nugget Analyse R1.

Dokumente mehrerer Freigaben

Zur Darstellung von Dokumenten erzeugt das Subsystem Knowledge Nugget Visualisierung eine Baumtabelle, die für jede Markierung eine Spalte anzeigt, um die Modellelemente nach ihren Knowledge Nuggets unterscheiden zu können. Existieren beispielsweise zwei Markierungen, Abstraktionsgrad und Freigabedeskriptor, werden zwei Spalten eingefügt, eine für den Abstraktionsgrad des Modellelements, die andere für die Freigabe. So können gleichnamige Modellelemente nach bekannten Konzepten, nämlich Abstraktionsgrad und Freigabe, unterschieden werden und die dahinter liegenden Knowledge Nuggets bleiben für den Anwender transparent.

Dokumentenansicht

Abb. 87 zeigt als Beispiel die Dokumentenansicht mit zwei Anforderungsanalyse-Dokumenten (*Requirements Analysis Document*). Jede Zeile stellt ein Modellelement dar, die das Dokument baumartig strukturiert: In der ersten Zeile steht der Name des Dokuments, gefolgt von geschachtelten Sektionen, wobei nach einer *leaf sections* die gefilterten Modellelemente erscheinen. Die erste Spalte zeigt den Namen des Modellelements einer Zeile an, die zweite den Abstraktionsgrad des Modellelements und die dritte die Freigabe des Modellelements an.

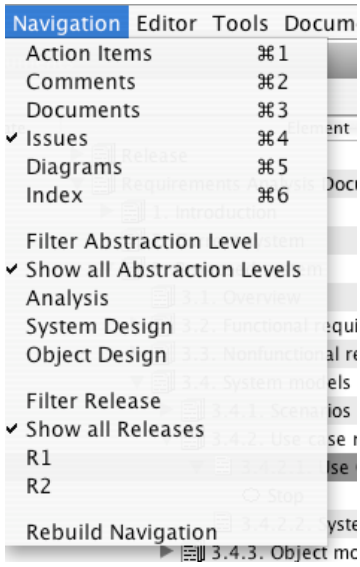


Abb. 88: Bildschirmkopie eines Menüs zum Einschränken der angezeigten Modelle.

Filter

Ein zusätzlicher Filter kann durch die Auswahl von Markierungen die in EXPLoRE angezeigten Modellelemente auf die Modellelemente eines oder mehrerer Knowledge Nuggets einschränken. Beispielsweise kann der Anwender nur die Modellelemente einer Freigabe anzeigen und so die Entwicklung der aktuellen Freigabe unterstützen.

Beispiele für Anwendung des Filters

Während Entscheidungen des Freigabemanagers mehrere Freigaben und Modelle verschiedener Abstraktionsgrade berücksichtigen, konzentrieren sich andere Interessenvertreter auf das Modell einer konkreten Freigabe und bzw. oder eines konkreten Abstraktionsgrades. Beispielsweise bearbeiten Analysten Analysemodelle für eine oder mehrere Freigaben. Um sie zu unterstützen, kann der Interessenvertreter die in der Anwendung angezeigten Modelle durch Festlegen der Markierungen beschränken, wie die Bildschirmkopie in Abb. 88 zeigt. Erstens kann sich ein Interessenvertreter in der Dokumentenansicht nur die Dokumente einer konkreten Freigabe anzeigen lassen, während die Dokumente aller anderen Freigaben ausgeblendet werden. Zweitens kann der Interessenvertreter die Dokumente eines bestimmten Abstraktionsgrads, beispielsweise die Analysemodelle, aller Freigaben auswählen, wodurch ihm Dokumente mit einem anderen Abstraktionsgrad verborgen bleiben. Drittens ist es möglich, den Fokus auf ein ganz konkretes Modell zu legen, indem alle Markierungen festgelegt werden. So kann beispielsweise ein Architekt, der gerade die zweite Freigabe des Multimedia Players entwickelt, das Systementwurfsmodell der Freigabe 2 auswählen.

Abstraktionsgrad Visualisierung Subsystem

Die Dienste des Subsystems Knowledge Nugget Visualisierung nutzen sowohl das Abstraktionsgrad Visualisierung Subsystem als auch das Freigabedeskriptor Visualisierung Subsystem. Das Abstraktionsgrad Visualisierung Subsystem erlaubt Anwendern Systemmodelle auf mehreren Abstraktionsgraden zu erstellen. In einer administrativen Konfigurationsoberfläche kann der Benutzer Abstraktionsgrade und Beziehungen angelegen sowie die Propagation von Modellelementen konfigurieren. Dazu verwendet das Abstraktionsgrad Visualisierung Subsystem das Abstraktionsgrad Subsystem der Begründungsschicht (siehe Abschnitt 7.2).

Das Freigabedeskriptor Visualisierung Subsystem erlaubt die Systemmodellierung aufeinanderfolgender Freigaben, die Erstellung von Roadmaps mit

Freigaben, Freigabeplänen sowie alternativen Freigabeszenarios. Dazu verwendet das Freigabedeskriptor Visualisierung Subsystem die Benutzeroberflächen des Freigabe-Widget Subsystems (siehe 7.1.2) und die Modelle des Freigabemanagement Subsystems der Begründungsschicht (siehe Abschnitt 7.2).

Freigabedeskriptor Visualisierung Subsystem

7.2 Modelldiensteschicht

Die Modelldiensteschicht enthält Subsysteme, die vorbereitende Dienste für die Visualisierung der Informationen aus der Begründungsschicht bereitstellen und besteht aus dem Modellraum und dem Vergleich Subsystem (siehe Abb. 89).

Modelldiensteschicht

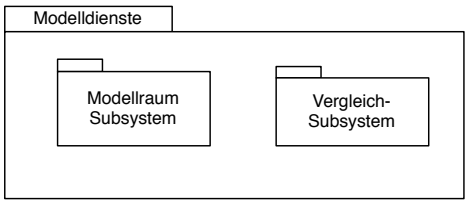


Abb. 89: Das Modelldienste Subsystem besteht aus zwei Subsystemen: Dem Modellraum Subsystem und dem Vergleich Subsystem.

Das Modellraum Subsystem leitet aus Modellelementen, ihren Abhängigkeiten sowie Beziehungen zwischen den Knowledge Nuggets Dimensionen ab. Für drei gegebene Dimensionen kann ein Dimension Kontroller ein dreidimensionales Gitter mit Modellelementen erzeugen, indem er das Gitter mit Hilfe einer Schnittstelle Eindimensional befüllt. Eindimensional verfügt über Methoden, um für ein gegebenes Modellelement nachfolgende und vorangehende Modellelemente in dieser Dimension zu liefern. Dieses Gitter enthält dann die Modellelemente für alle Positionen des Entscheidungsraums und dient als Basis für die dreidimensionale Visualisierung. Die Berechnung des kompletten Gitters ist eine zeitintensive Aufgabe und kann bei Aufruf des Entscheidungsraums Navigators gleich komplett erfolgen, was längere Startzeiten mit sich bringt. Diese vermeidet eine inkrementelle Berechnung des Gitters während der Navigation, die stets nur die nötigen Werte berechnet. Diese inkrementelle Berechnung führt jedoch vor allem anfangs zu Verzöge-

Dienste des Modellraum Subsystems

rungen während der Navigation, die den Anwender irritieren kann. Deshalb wurde ein hybrider Algorithmus zur Berechnung des Gitters implementiert, der zunächst ausgehend vom Startelement die nächsten 5 Werte für alle Dimensionen berechnet und anzeigt und dann Ruhezeiten nützt, um weitere Werte in einem eigenen Prozess zu ermitteln.

Dienste des Vergleich Subsystems

Das Vergleich Subsystem berechnet Unterschiede zwischen Modellelementen der gleichen Klasse, indem es für definierte Attribute bestimmt, ob sie bei zwei Modellelementen neu, gleich, geändert oder gelöscht sind. Zu jedem Modellelement kann ein Modell Element Diffing Result den Unterschied zu einem anderen Modellelement speichern. Ein Modell Element Diffing Result besteht aus mehreren Attribute Diffing Results, die den Unterschied des Attributes eines Modellelements zu einem anderen Modellelement festhalten. Die Durchführung des Vergleichs in der Modelldiensteschicht macht den Vergleich und die Visualisierung der Unterschiede unabhängig voneinander. Die Visualisierung greift beim Aufbau der Benutzeroberfläche für ein Attribut einfach auf die Ergebnisse der Unterschiedsberechnung zu.

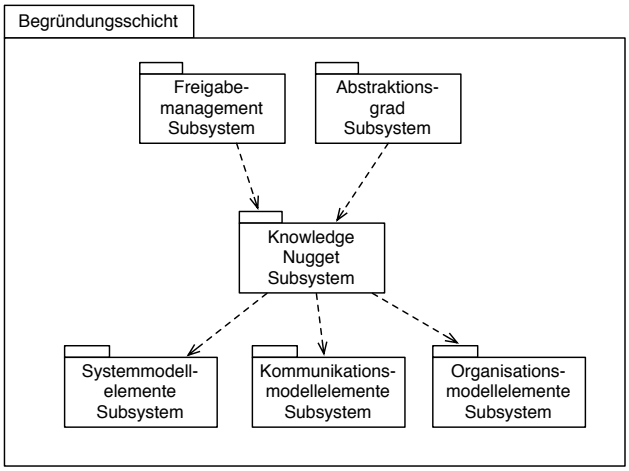


Abb. 90: Subsysteme von BEEF in der Modellschicht

7.3 Begründungsschicht

Begründungs schicht

Die Begründungsschicht von EXPLoRE enthält die Begründungen aus dem begründungsbasierten Freigabemodell und enthält das Knowledge Nugget Subsystem, das Abstraktionsgrad Subsystem und das Freigabemanagement

Subsystem sowie das System-, Kommunikations- und Organisationsmodellelemente Subsystem (siehe Abb. 90).

Dienste des Knowledge Nugget Subsystems

Das Knowledge Nugget Subsystem enthält Knowledge Nuggets und erlaubt Markierungen für Knowledge Nuggets zu erzeugen. Das Subsystem nimmt erzeugte Markierungen als Grundlage für die Erzeugung notwendiger Knowledge Nuggets und ihrer Beziehungen. Das Knowledge Nugget Subsystem hört auf Änderungen und propagiert diese an Modellelemente niedrigerer Knowledge Nuggets. Die beschriebenen Dienste des Knowledge Nugget Subsystems verwenden das Abstraktionsgrad Subsystem und Freigabemanagement Subsystem.

Dienste des Abstraktionsgrad Subsytsems

Das Abstraktionsgrad Subsystem hängt vom Knowledge Nugget Subsystem ab, denn es verwendet Knowledge Nuggets für die Modellierung eines Systems auf Abstraktionsgraden. Das Abstraktionsgrad Subsystem erlaubt beliebige Abstraktionsgrade für die Systemmodellierung zu erstellen und Beziehungen zwischen diesen zu definieren. Systemmodellelemente können mehreren Abstraktionsgraden zugewiesen und in den Modellen eines Abstraktionsgrads unterschieden und verfeinert werden.

Dienste des Freigabemanagement Subsystems

Das Freigabemanagement Subsystem verwendet Knowledge Nuggets zur Erstellung von Systemmodellen für aufeinanderfolgende Freigaben und alternativer Freigabeszenarios. Mit dem Freigabemanagement Subsystem kann der Freigabemanager Freigabeplanungsentitäten und alternative Freigabeszenarios bewerten und deren Modelle erkunden.

Systemmodellelemente Subsystem

Das Systemmodellelemente Subsystem enthält Modellelemente für die Systemmodellierung, wie sie die Abschnitte 4.4.1 bis 4.4.3 beschreiben. Die Implementierung greift hier auf die Modellelemente und Beziehungen des RUSE-Modells zurück. Das Kommunikationsmodelle Subsystem enthält die Kommunikationsmodelle des RUSE-Modells und umfasst ein Begründungsmanagement Subsystem (siehe Abschnitt 4.2). Das Organisationsmodelle Subsystem besteht aus dem RUSE-Organisationsmodell (siehe Abschnitt 4.3).

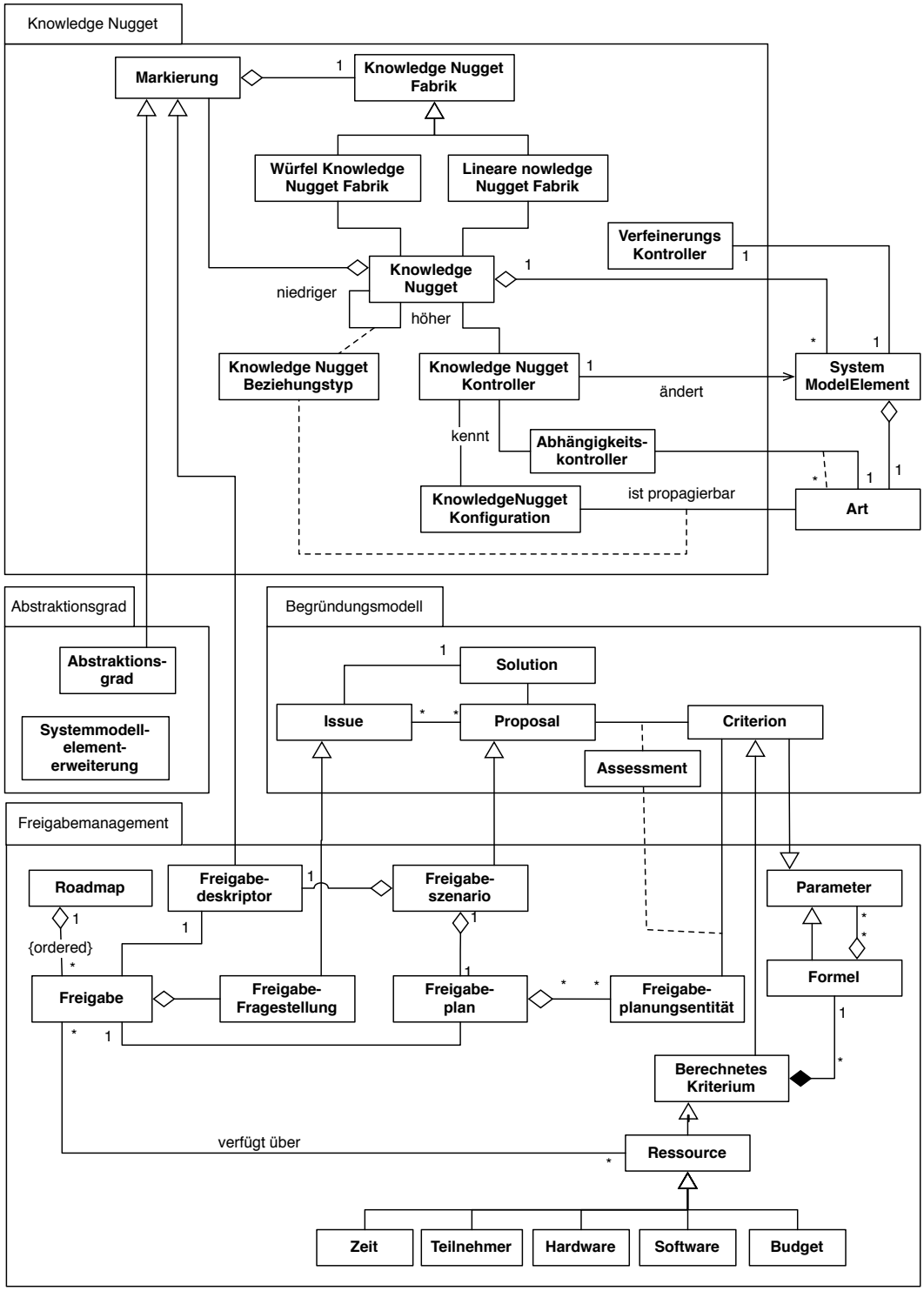


Abb. 91: Subsystemdekomposition von BEEF in EXPLoRE.

Klassen im Knowledge Nugget Subsystem

Abb. 91 zeigt die Klassen der Subsysteme Knowledge Nugget, Freigabemanagement, Abstraktionsgrad und Begründungsmanagement. Das Knowledge Nugget Subsystem enthält die Klassen des Knowledge Nugget Modells, wie sie in Kapitel 5 beschrieben worden sind sowie die Knowledge Nugget Fabrik zum Erzeugen von Knowledge Nuggets für gegebene Markierungen. Die Klasse Knowledge Nugget hat eine Assoziation zu der Klasse Systemmodellelement und stellt somit eine Verbindung zum Systemmodellelemente Subsystem her.

Klassen im Abstraktionsgrad Subsystem

Das Subsystem Abstraktionsgrad enthält die Markierung Abstraktionsgrad, die als einzige Klasse nötig ist, um das Knowledge Nugget Subsystem für Abstraktionsgrade zu verwenden. Hinzu kommt und eine Modellelementerweiterung von Systemmodellelement, die eine Assoziation zu einem Modellelement hat und über eine Methode verfügt, um den Abstraktionsgrad des assoziierten Modellelements zu bestimmen.

Klassen im Freigabemanagement Subsystem

Freigabemanagement enthält die Klassen zur Freigabeplanung: Roadmap, Freigabe, Freigabeplan, Freigabeplanungsentität, Freigabeszenario, Freigabefragestellung Freigabedeskriptor, Ressource, Zeit, Teilnehmer, Hardware, Software, berechnetes Kriterium, Formel und Parameter. Die Klasse Freigabedeskriptor stellt als Subklasse von Markierung eine Verbindung zum Knowledge Nuggets Subsystem her.

Klassen im Begründungs management Subsystem

Das Begründungsmanagement Subsystem ist Teil des Kommunikationsmodellelemente Subsystems und enthält die Klassen des RUSE- Begründungsmanagements: Fragestellung (*issue*), Vorschlag (*proposal*) Lösung (*solution*), Kriterium (*criterion*) und Bewertung (*assessment*), wie sie Abschnitt 2.4 beschreibt. Das Freigabemanagement Subsystem hängt vom Begründungsmanagement Subsystem ab, weil die Klassen Freigabefragestellung eine Spezialisierung von Fragestellung, Freigabeszenario eine Spezialisierung von Vorschlag und berechnetes Kriterium eine Spezialisierungen von Kriterium ist. Zudem hat Freigabeplanungsentität eine Assoziation zu Kriterium (*criterion*).

7.4 Elementspeicherschicht

Elementspeicherschicht

Die Elementspeicherschicht von EXPLoRE speichert Begründungen aus BEEF mit Hilfe des RUSE-Metamodells (siehe Abschnitt 7.4.1). Abschnitt 7.4.2 beschreibt die Realisierung der Änderungspropagation.

7.4.1 RUSE-Metamodell

RUSE-Metamodell

Das generische RUSE-Metamodell kann heterogene Artefakte unabhängig von ihrer Art und Beziehungen zwischen diesen darstellen. Dazu stellt das RUSE-Metamodell einheitliche Klassen bereit, um Modelle eines Softwareprojekts unabhängig von ihrer Art zu speichern. Für diese Modelle stellt das RUSE-Metamodell eine Versions- und Zugriffskontrolle bereit.

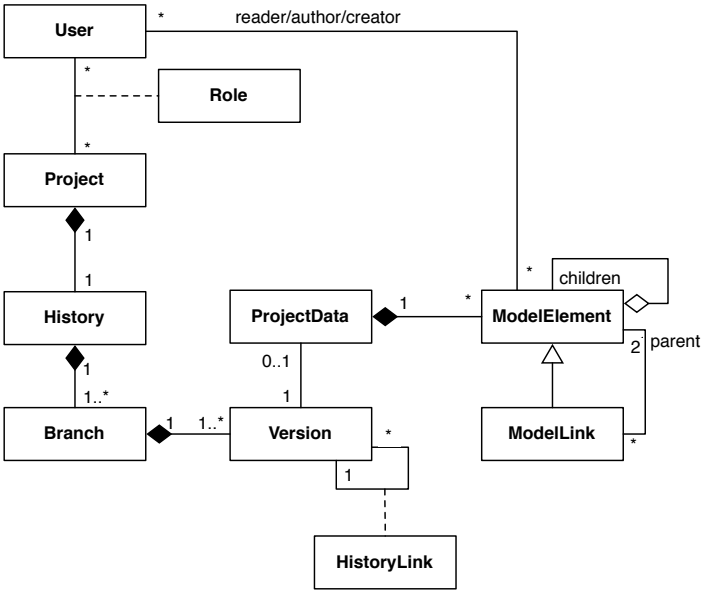


Abb. 92: RUSE-Metamodell [Wolf 2007]

Versionskontrolle

Die Klasse Projekt stellt ein Softwareprojekt dar und erlaubt Zugriff auf die Projektdaten. Die Klassen History, Branch, Version und HistoryLink garantieren eine Versionskontrolle über Projektdaten: Für ein Projekt können mehrere Branches existieren, in denen alle Projektdaten Versionen haben. (siehe Abb. 92) [Wolf 2007]

Modell-element und Modelllink

Die Projektdaten werden in Modellelementen und Modelllinks gespeichert. Ein Modellelement repräsentiert ein beliebiges Software Engineering Konzept. Ein Modellelement bietet generische Methoden zum Speichern von Attributen an. Es können einfache Werte in Feldern, Listen oder Tabellen abgelegt werden. Für die Beschreibung komplexer Artefakte und Abhängigkeiten stehen Subelemente und Modelllinks zur Verfügung. Ein Modellelement kann mehrere Subelemente haben. Ein Subelement ist ein Modellelement, das Teil genau eines Eltern-Elements ist. Modelllinks sind Modellelemente, die zwei Modellelemente verbinden. (siehe Abb. 92) Durch die Verwendung von Modellelementen und Modelllinks entsteht ein Projektgraph, der Artefakte des Software Engineerings sowie deren Abhängigkeiten abbildet. Die ProjectData Klasse verfügt über Methoden, um auf Modellelemente im Projektgraph zuzugreifen, den Projektgraphen zu traversieren, Modellelemente zu suchen oder zu filtern. Die Speicherung persistenter Daten geschieht durch generische Methoden der Klasse Modellelement. [Wolf 2007]

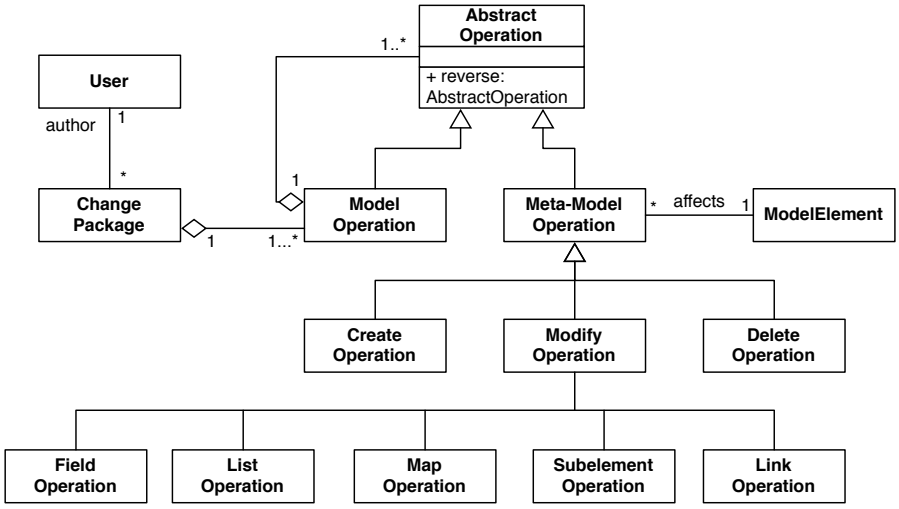


Abb. 93: Änderungen im RUSE-Metamodell [Wolf 2007]

Operationen

Veränderungen am Metamodell werden durch Operationen festgehalten und kommuniziert. Die Versionskontrolle verwendet diese Operationen, um Änderungen am Modell zu speichern und nachzuvollziehen; alle Änderungen können mit Hilfe der Operationen rückgängig gemacht werden. Ein Anwen-

der ändert ein Modellelement, wodurch ein Änderungspaket erzeugt wird, das aus einer Menge von Modelloperationen besteht. Modelloperationen dienen dazu zusammengehörige Operationen zu kapseln und stellen damit eine atomare Einheit dar, das heißt sie sind stets als ganzes oder gar nicht zu behandeln. Eine Modelloperation kann wieder aus mehreren Modelloperationen bestehen, wobei sie letzen Endes mehrere Metamodelloperationen enthält, die Veränderungen an Modellelementen beschreiben. Das RUSE-Metamodell unterscheidet Metamodelloperationen für das Erzeugen, Löschen und Ändern von Modellelementen. Es gibt Metamodelloperationen für Änderungen an den Feldern, Listen, Tabellen, Subelementen oder Links, die angeben, ob ein Wert hinzugekommen, entfernt oder geändert worden ist. (Abb. 93) [Wolf 2007]

Rollen für User

User können auf mehrere Projekte zugreifen. Für jedes Projekt kann ihnen eine bestimmte Rolle zugewiesen werden, aus der sich die Zugriffsrechte für einen User und ein Projekt ergeben. (siehe Abb. 92) Beispiele für Rollen sind Projektmanager oder Analyst. Gemäß ihren Zugriffsrechten dürfen User Modellelemente lesen oder erzeugen. [Wolf 2007] Benutzer können mit Teilnehmern aus dem RUSE-Organisationsmodell (siehe 4.3) assoziiert werden: Nicht jeder Teilnehmer muss ein Benutzer sein und auf das Modell zugreifen können; es können also auch Teilnehmer modelliert werden, die das System nicht benutzen. Umgekehrt muss jeder Benutzer ein Teilnehmer des Projekts sein.

7.4.2 Realisierung der Änderungspropagation

Modelloperationen durch Änderungspropagation

Die Änderungspropagation verwendet die RUSE Meta-Modell Operationen, um Änderungen an verfeinerten Modellelementen vorzunehmen: Die Elementspeicherschicht veröffentlicht Änderungen im RUSE-Metamodell und benachrichtigt andere Subsysteme über Änderungen, indem sie durchgeführte Modell- und Metamodelloperationen (siehe Abb. 93) verschickt. Der Knowledge Nugget Kontroller hört auf diese Operationen und ändert als Konsequenz andere Modellelemente. Dadurch erzeugt er neue Metamodelloperationen, die auf Modellebene mit den ursprünglichen Änderungen zusammengehören und eine atomare Operation darstellen: Soll die ursprüngliche Änderung eines Modellelements, beispielsweise eines Anwenders, durch die Ver-

sionskontrolle zurückgesetzt werden, sollen auch die in der Folge durch den Knowledge Nugget Kontroller durchgeführten Änderungen an den Modellen anderer Knowledge Nuggets zurückgesetzt werden. Alle Änderungen, das heißt die des Anwenders und die des Knowledge Nugget Kontrollers, sollen zusammen in einer Operation festgehalten werden.

Vorab-Beobachter

Um die bereits erstellten Modelloperationen des Anwenders um weitere vom Knowledge Nugget erzeugte Modelloperationen erweitern zu können, verwendet die Elementspeicherschicht ein Beobachter Muster: Ein Veröffentlicher teilt Änderungen mit, bevor sie an das übrige System verschickt werden. Beobachter, die sich an diesem Veröffentlicher registrieren, werden in EXPLoRE als Vorab-Beobachter bezeichnet, da sie Änderungen vor den übrigen Beobachtern erhalten und somit weitere Änderungen zu Operationen hinzufügen können, bevor alle Änderungen als atomare Einheit an die übrigen Beobachter weitergegeben werden. Der Knowledge Nugget Kontroller ist ein

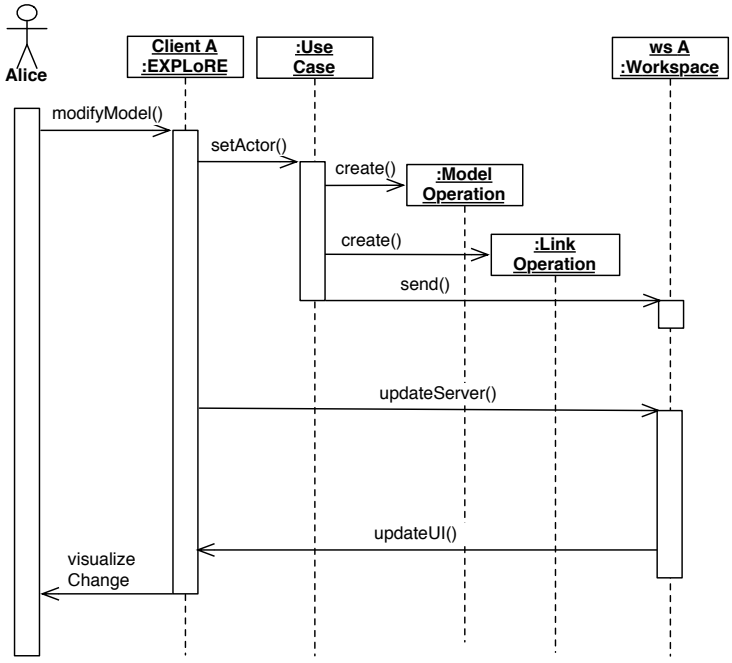


Abb. 94: Kontrollfluss zum Einsatz von Knowledge Nuggets (I)

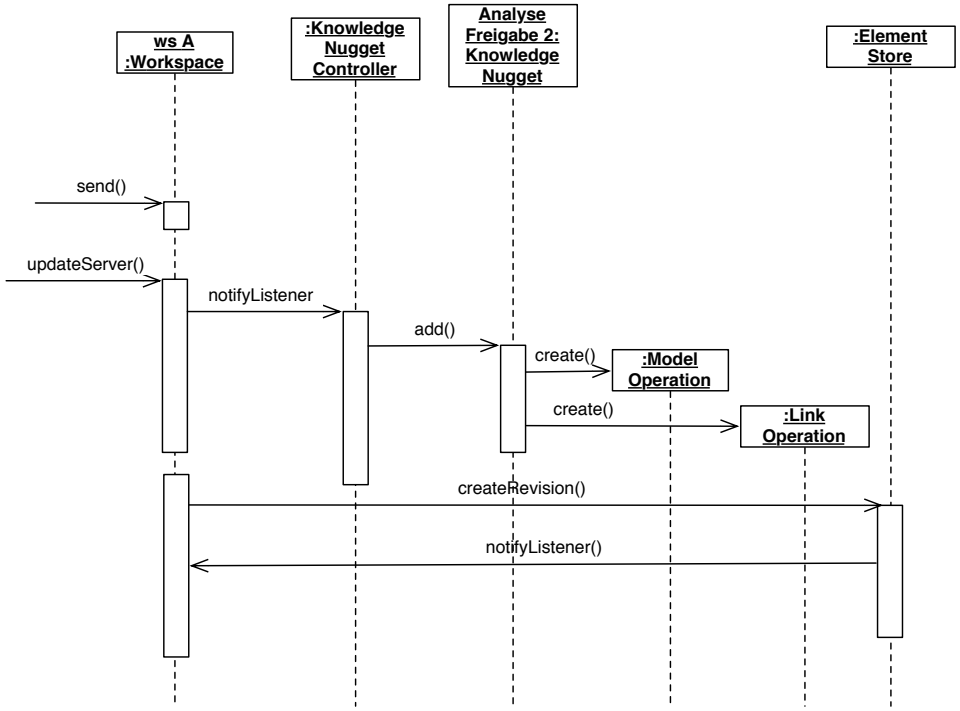


Abb. 95: Kontrollfluss zum Einsatz von Knowledge Nuggets (II)

Beispiel für einen Vorab-Beobachter, der vor dem übrigen System über Änderungen informiert wird. Er nützt dies, um neu erzeugte Modellelemente Knowledge Nuggets zuzuordnen oder um Änderungen an Modellelementen an die verfeinerten Modellelemente anderer Knowledge Nuggets weiterzuverbreiten.

Kontrollfluss bei Änderung

Abb. 94 bis Abb. 96 zeigen den Kontrollfluss bei einer Änderung im Systemmodell. Das Ende einer Änderung wird vom Klient durch den Aufruf der Methode updateServer() angezeigt, in Abb. 94 ruft der EXPLoRE Klient A updateServer() auf dem Workpace A auf. Nun beginnt der Workspace A seine Beobachter zu informieren und zwar zunächst, die Vorab-Beobachter und dann die übrigen Beobachter. In Abbildung Abb. 95 ist der Knowledge Nugget Kontroller ein Vorab-Beobachter und wird über die Änderung des initiierenden Aktors des Anwendungsfalls informiert. Er fügt den Aktor auch dem Anwendungsfall desjenigen Knowledge Nuggets hinzu, der das Analysemodell der Freigabe 2 beschreibt, wodurch dem Änderungspaket neue Modell- und Metamodelloperationen hinzugefügt werden. Im Anschluss er-

zeugt der Workspace A eine Revision und überträgt sie an den Element Store (siehe Abb. 95), welcher die übrigen beobachtenden Klienten aufruft und ihnen nun jedoch die gesamten Änderungen, also die des Anwenders und die des Knowledge Nugget Kontrollers, mitteilt (siehe Abb. 96). Als Folge werden die Modelle und Benutzeroberflächen der Klientanwendungen aktualisiert.

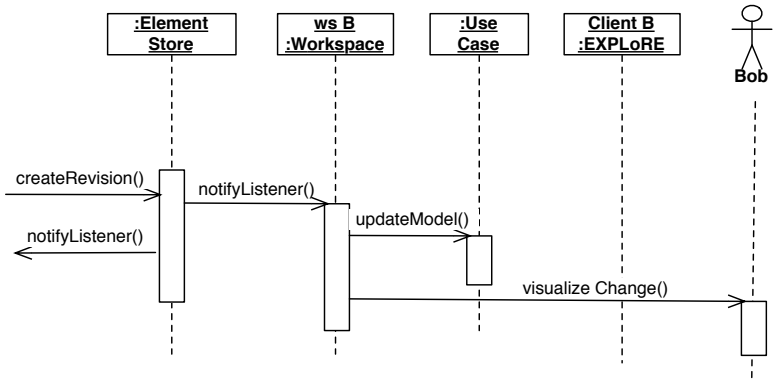


Abb. 96: Kontrollfluss zum Einsatz von Knowledge Nuggets (III)

7.4.3 Propagation von Modellelementen

Begründung für Kopien

Während sich der vorangehende Abschnitt mit der Propagation von Änderungen zwischen bestehenden Modellelementen beschäftigt hat, beschreibt dieser Abschnitt, wie ein Systemmodellelement zu mehreren Knowledge Nuggets hinzugefügt werden kann. Um Unterschiede im Zustand eines Systemmodellelements darzustellen, erzeugen Knowledge Nuggets eine eigene Instanz für jedes Systemmodellelement, sobald ein Systemmodellelement einem Knowledge Nugget hinzugefügt wird (siehe Abschnitt 5.1). Diese neu erzeugten Instanzen sollten zunächst dieselben Eigenschaften haben, wie das Systemmodellelement, das dem Knowledge Nugget hinzugefügt worden ist. Deshalb bietet es sich an, beim Hinzufügen eines Systemmodellelements zu einem Knowledge Nugget eine Kopie des Systemmodellelements zu erzeugen, die dann weiter verfeinert werden kann, ohne dass die Instanzen des Systemmodellelements anderer Knowledge Nuggets betroffen sind.

Copy-Methode in Modellelement

Diese Kopie eines Modellelements erzeugt eine neue Instanz eines Modellelements mit identischen Eigenschaften und kann durch eine Modifikation der Klasse Modellelement des RUSE-Metamodells erstellt werden. Die Erzeu-

gung einer Kopie in der Superklasse Modellelement hat den Vorteil, dass damit Kopien für alle Spezialisierungen von Modellelementen erzeugt werden können. Allerdings ist die Klasse Modellelement als Verallgemeinerung aller Modellelemente unabhängig von ihren Spezialisierungen und kennt folglich deren Eigenschaften nicht. Ein Modellelement muss also allgemein ohne Anbetracht eines konkreten Modellelements eine Kopie von sich erzeugen und dennoch alle relevanten Informationen enthalten.

Kopieren einfacher Werte und Subelemente

Als Grundlage für die Erstellung der Kopie dienen die Attribute eines Modellelements, das heißt einfache Werte, die in Feldern, Listen oder Tabellen abgelegt sind, Subelemente und andere verlinkte Modellelemente. (siehe Abschnitt 7.4.1) Einfache Werte der Quelle werden kopiert und auch wieder als einfache Werte der Kopie des Modellelements hinzugefügt. Subelemente sind Modellelemente, die Teil eines Eltern-Modellelements sind und damit fest zum Eltern-Modellelement gehören, weshalb die Methode copy() auch Subelemente mit ihren Eltern-Modellelementen mitkopiert und der Kopie wieder als Subelemente hinzufügt.

Kopieren verlinkter Modellelemente

Anders stellt sich die Situation bei verlinkten Modellelementen dar: Alle verlinkten Modellelemente eines Modellelements zu kopieren, führt dazu, dass beim Kopieren eines einzelnen Modellelements die gesamte Zusammenhangskomponente des Projektgraphen, dem das Modellelement angehört, kopiert wird. Dies kann dazu führen, dass durch die Kopie eines einzelnen Modellelements nahezu der gesamte Projektgraph dupliziert wird. Tatsächlich hängt es vom konkreten Modellelement und der Anwendung ab, wie wichtig ein Modelllink für das kopierte Modellelement ist, und kann daher nur auf Modellebene entschieden werden. Wird ein Anwendungsfall kopiert, so kann es sinnvoll sein, Links zu initiierenden und partizipierenden Akteuren mitzukopieren, da sie einen wesentlichen Bestandteil eines Anwendungsfalls ausmachen. Bei der Kopie einer Klasse ist es hingegen nicht immer wünschenswert alle anderen Klassen, die durch eine Assoziation verbunden sind, mitzukopieren. Deshalb verzichtet die Implementierung der Methode copy() in Modellelement auf die Kopie von Modelllinks im Metamodell. Modelllinks müssen dann von Anwendungen der Methode copy() auf Modellebene eingefügt werden. Beispielsweise verwaltet im Knowledge Nugget

Modell der Abhängigkeitskontroller (siehe Abschnitt 5.2.3) Abhängigkeiten zu verlinkten Modellelementen.

7.5 Existierende Werkzeuge

Release-Planner

Dieser Abschnitt kontrastiert existierende Werkzeuge zur Freigabeplanung mit EXPLoRE. Einige Werkzeuge zur Freigabeplanung realisieren eine Planungsmethode und unterliegen damit den in Abschnitt 4.5 beschriebenen Schwächen der Methode. Beispielsweise verwendet der ReleasePlanner [Ruhe 2005] die Planungsmethode EVOLVE, um Freigabeplanungsentitäten auf eine Folge von Freigaben zu verteilen. Jedoch kann pro Projekt im ReleasePlanner nur eine Art von Freigabeplanungsentitäten gewählt werden, es können also beispielsweise nicht wie in EXPLoRE Fehlerberichte und Merkmale zur Planung einer Freigabe herangezogen werden. Da eine Freigabe eine vorangegangene Freigabe in der Regel nicht nur um Anforderungen erweitert, sondern auch Verbesserungen enthält, wenn Fehler behoben werden, sollten auch Fehlerberichte in die Freigabeplanung mit einfließen. Ebenso können Freigabeplanungsentitäten nicht verfeinert werden, wodurch der ReleasePlanner entweder nur für die strategische oder nur für die operationelle Planung eingesetzt werden kann. EXPLoRE hingegen erlaubt eine Verbindung der strategischen und operationellen Freigabeplanung.

ID	Requirement	Alternative 1	Alternative 2	Alternative 3	Alternative 4	Alternative 5
1	PlayAudioStream	1	1	1	2	2
3	PlaySongs	1	2	1	2	2
4	PlayVideo	2	2	3	3	3
2	PlayVideoStream	2	1	2	2	1
7	ShowFotos	3	3	2	1	2
5	UploadFoto	2	1	2	1	2
6	UploadSong	1	2	1	2	1

Analyze Solution Set

Abb. 97: Alternative Freigabepläne für einen einfachen Multimedia Player in ReleasePlanner [Ruhe 2005].

Der ReleasePlanner erzeugt verschiedene alternative Freigabepläne, die angeben in welcher Freigabe Anforderungen umgesetzt werden könnten. Abb.

97 zeigt alternative Freigabepläne zur Umsetzung von Anforderungen eines einfachen Multimedia Players. Hierbei handelt es sich um eine Matrix, die für jede Kombination aus Alternative und Anforderung angibt, in welcher Freigabe die Anforderung realisiert werden soll. Jedoch können diese nicht weiter evaluiert werden, da das Werkzeug notwendige Systemmodelle oder Abhängigkeiten nicht erfasst.

Accept-Planner

AcceptPlanner [AcceptSoftware 2008] unterstützt Portfoliomanagement, strategische Produktplanung sowie Anforderungsmanagement. Das Portfoliomanagement kann Kundenwünsche, Unternehmensziele, Strategien sowie Risiken und Konkurrenten am Markt erfassen und strukturiert darstellen und analysieren. Diese Informationen sollen den Entscheidungsträger unterstützen, bestimmte Produkte im Portfolio zu priorisieren. Die strategische Produktplanung im AcceptPlanner plant mehrere aufeinanderfolgende Freigaben mit Anforderungen auf hohem Niveau unter Berücksichtigung ihres strategischen Nutzens und Aufwandsschätzungen. Die Freigabeplanung basiert auf betriebswirtschaftlichen Kriterien, wie Kosten, Nutzen oder Kundenzufriedenheit. Alternative Entscheidungsszenarios können erstellt werden und bezüglich erfasster Kriterien verglichen werden. Anforderungen können erfasst, priorisiert und Freigaben zugeordnet werden. Zudem stellt das Werkzeug Kommunikationsmechanismen zur Diskussion von Anforderungen bereit. Wie beim ReleasePlanner können für Anforderungen Gleichzeitigkeit- und Vorgängerabhängigkeit spezifiziert werden. [AcceptSoftware 2008], [Kaarlas & Vähäniitty 2005]

AcceptPlanner [AcceptSoftware 2008] ist ein Werkzeug, das sowohl Portfolio- und Produktmanagements als auch Anforderungsmanagement erlaubt. Entscheidungen berücksichtigen lediglich wirtschaftliche und marktstrategische Kriterien. Änderungen aus der Systementwicklung, die beispielsweise Fragestellungen oder Fehlerberichte hervorrufen, erfasst AcceptPlanner ebenso wenig wie die Systemmodelle, Kommunikationsmodelle und Organisationsmodelle und deren Abhängigkeiten.

VersionOne, InStep

Die agilen Projektmanagement Werkzeuge VersionOne [VersionOne 2008] und In-Step [InStep 2008] ignorieren das Problem von Abhängigkeiten zwischen Anforderungen und unterstützen nur die Zuweisung von Merkmalen

zu Freigaben basierend auf Ressourcenabschätzungen. Die Auswirkungen einer Änderung in detaillierten Modellen der Systemmodellierung oder in den Organisationsmodellen auf eine Freigabe können in VersionOne und In-Step nicht als Entscheidungsgrundlage herangezogen werden.

MS-Project

MS-Project kann für Projekte unabhängig von ihrer Art eingesetzt werden und unterstützt allgemien die Projektorganisation, eine aktivitätsorientierte Projektplanung durch die Erstellung von Zeitplänen und Berücksichtigung von Ressourcen. Eine Verwaltung von Artefakten der Softwareentwicklung ist in MS-Project nicht möglich: Beispielsweise können keine Anforderungen oder Abhängigkeiten zwischen diesen erfasst werden. [Chatfield & Johnson 2007]

MS-Ecxel

In der Praxis verwendet man für die Planung von Freigaben häufig Kalkulationstabellen, beispielsweise in Microsoft Excel [Walkenbach 2007], wobei die Tabellen für jedes Projekt individuell gestaltet werden müssen. Roadmaps und Freigabepläne können in eigenen Tabellen verwendet werden.. Die Darstellung von Roadmaps und Freigabeplänen in eigenen Kalkulationstabellen ist die einfachste Art, zu planen ohne Anbindung an die übrigen Werkzeuge der Software Entwicklung. Kalkulationstabellen sind nicht oder nur ungenügend geeignet, um die heterogenen Artefakte der Software-Entwicklung zusammen mit ihren Abhängigkeiten abzubilden. [Kaarlas & Vähäniitty 2005]

Abgrenzung zu existierenden Werkzeugen

Existierende Werkzeuge, wie sie oben beschrieben worden sind, adressieren nur einen Teil der beschriebenen Problematik und Anforderungen an das Freigabemanagement. Daher werden für eine fundierte Entscheidungsfindung Modelle aus mehreren verschiedenen Werkzeugen benötigt. Beispielsweise könnte der ReleasePlanner zusammen mit Doors [Doors 2008] zur Anforderungsanalyse und Rationale Rose [Quatrani 2003] zur Erstellung von UML-Modellen für den Systementwurf und Objektentwurf eingesetzt werden. Jedes dieser Werkzeuge jedoch verwendet eine eigene Datenbasis, was zu Inkonsistenzen in den Daten der eingesetzten Werkzeuge führt, wenn Artefakte von mehreren Werkzeugen verwaltet werden. Zudem können Abhängigkeiten nicht oder nur ungenügend werkzeugübergreifend verfolgt und analysiert werden. In der Folge treffen Projektmanager falsche Entscheidungen auf Grund von nicht vollständigen, veralterten oder nicht konsistenten Daten.

8 Evaluierung

Mehrfachfall-studie

Die empirische Evaluierung von BEEF erfolgte durch eine eingebettete Mehrfachfallstudie [Lee 2003], [Yin 2002], die kausale Zusammenhänge des Freigabemanagements erforscht hat. Die Fallstudie wurde zweimal in verschiedenen Projekten durchgeführt und verwendet die direkten Techniken Beobachtung und Befragung, um Daten über Entscheidungen im Freigabemanagement und den Einsatz von EXPLoRE zu erhalten.

8.1 Entwurf der Fallstudie

Kernfragen

Bei Entscheidungen im Freigabemanagement müssen die der Freigabe zu Grunde liegenden Modelle unabhängig von ihrer Art sowie ihrer Abhängigkeiten berücksichtigt werden. Diese Hypothese sollen die folgenden Kernfragen der Fallstudie bestätigen:

> Wie und warum unterstützt das begründungsbasierte Freigabemodell die Entscheidungsfindung bei Änderungen, wenn Freigaben in kurzen Zyklen aufeinanderfolgen?

Behauptungen

Um die Kernfragen näher zu erforschen, werden aus ihnen mehrere Behauptungen abgeleitet:

> Das begründungsbasierte Freigabemodell macht Begründungen für Entscheidungen im Freigabemanagement explizit und reduziert damit die Herausforderungen im Freigabemanagement

B_1) durch Freigabedeskriptoren mit den erfassten Systemmodellen, Kommunikations- und Organisationsmodellen sowie ihren Abhängigkeiten,

B_2) durch die explizite Modellierung alternativer Freigabeszenarios,

B_3) durch Freigabeoperationen, welche die Freigabedeskriptoren modifizieren.

Analyseeinheit und Fälle

Zur Untersuchung der Behauptungen B_1 bis B_3 verwendet die Mehrfachfallstudie Änderungen als Analyseeinheit, das heißt Experten beobachten den Umgang mit auftretenden Änderungen im Projekt. Zur Untersuchung einer Änderung betrachtet diese Fallstudie folgende drei Fälle: a) Abhängigkeiten zwischen und innerhalb von Freigabedeskriptoren, b) alternative Freigabeszenarios und c) Freigabeoperationen.

Fall Abhängigkeiten

a) Abhängigkeiten werden als Fall gewählt, weil sie darlegen, wie viele Modellelemente von einer Änderung betroffen sind und somit Aufschluss über die Komplexität der Änderung geben. Die Komplexität einer Änderung zeigt, ob kritische Auswirkungen ohne die Verwendung von BEEF übersehen werden könnten, und wird durch die Abhängigkeiten zwischen geänderten und betroffenen Modellelementen ermittelt. Die Technik der Beobachtung kann die Anzahl der von einer Änderung betroffenen Modellelemente ermitteln, indem ein Experte notiert, welche Modellelemente der Freigabemanager in seine Entscheidung einbezieht. Dabei ist für ein betroffenes Modellelement von Interesse, wie lang der Weg im Projektgraph zwischen ihm und dem geänderten Modellelement ist, das heißt wie viele Modellelemente den beiden Modellelementen liegen. Zur Bestimmung der Komplexität der Abhängigkeit, wird der längste Weg mit der Anzahl betroffener Modellelemente multipliziert. Ist die Komplexität der Abhängigkeit größer oder gleich fünf, so kann die Änderung als sehr komplex angesehen werden. Eine Änderung ist in diesem Sinne sehr komplex, entweder wenn es mindestens fünf direkt betroffene Modellelemente gibt oder wenn nur ein Modellelement betroffen ist, zu dem man über mindestens fünf andere Modellelemente gelangt. [Miller 1956], [Cowan 2001] Ist die Komplexität der Abhängigkeit kleiner als 2, ist die Änderung nicht komplex,

dazwischen komplex. Bereits das Auftreten weniger sehr komplexer Abhängigkeiten kann zeigen, dass das Modell eine Unterstützung bei der Entscheidungsfindung darstellt. Denn diese Fälle sind zu unübersichtlich, um ohne ein Werkzeug behandelt zu werden. Können hingegen betroffene Modellelemente einfach ermittelt werden, ohne das Modell zu verwenden, stellt ein derartiges Modell keine Unterstützung in der Entscheidungsfindung, sondern Mehraufwand durch die Erstellung der Modelle dar.

Fall Freigabeszenarios

b) Alternative Freigabeszenarios zeigen, dass eine Änderung zu Diskussionen führt, weil nicht klar ist, wie mit der Änderung umgegangen werden soll und deshalb mittels alternativer Entscheidungsszenarios Modelle verglichen werden können. Um die Bedeutung alternativer Freigabeszenarios für die Freigabeplanung zu bestimmen, können die Modelle der Alternativen analysiert und verglichen werden, wobei die Analyse vorwiegend Unterschiede der Modelle fokussiert. Durch die Modelle von BEEF kann die Anzahl der unterschiedlichen Modellelemente bestimmt werden. Eine Modellierung alternativer Freigabeszenarios ist nur bei wichtigen Änderungen zu erwarten, weshalb die Anzahl alternativer Freigabeszenarios gering sein wird. Diese wenigen Freigabeszenarios sollten sich jedoch stark in ihren Modellen unterscheiden, um die Behauptung zu bestärken.

Fall Freigabeoperationen

c) Freigabeoperationen erlauben die Anpassung einer Freigabe als Reaktion auf eine Änderung und sind als Fall von Interesse, da sie zeigen, dass die eingetretene Änderung zu einer Änderung der Roadmap geführt hat. Schließlich belegen Freigabeoperationen, ob und wie eine Änderung letztlich eine Freigabe verändert hat. Hier ist interessant, ob der Freigabemanager die Möglichkeit zur Planung einer Folge von Freigaben in einer Roadmap nutzt oder nur die aktuelle Freigabe bei der Planung fokussiert. Experten sollen in der Fallstudie beobachten, ob zukünftige Freigaben bei einer Änderung bereits geplant werden, oder ob verschobene bzw. neue Freigabeplanungsentitäten zufällig auf nachfolgende Freigaben verteilt und erst später bei der Realisierung genau geplant werden. Betrachtet der Freigabemanager bei neuen oder zu verschiebenden Freigabeplanungsentitäten Beschränkungen und Ziele nachfolgender Freigaben genauer, zeigt dies, dass die Probanden die Möglichkeiten zur Planung mehrerer Freigaben benötigen.

Art der Änderung

Diese Beobachtungen und eine Analyse der Artefakte sollen zudem die Art der Änderung festhalten. Die Fallstudie differenziert Änderungen in den Anforderungen, Technologien, Organisation und Fehler, um die Bedeutung der einzelnen Modelle zu erforschen. Beispielsweise zeigen Änderungen in der Organisation, die eine Analyse der Abhängigkeiten und Freigabeoperation auslösen, dass Organisationsmodelle das Freigabemanagement unterstützen.

Befragung

Schließlich soll eine Befragung der Probanden durch einen Fragebogen feststellen, wie sich die Probanden bei der Entscheidungsfindung durch die erfassten Abhängigkeiten oder Freigabeszenarios unterstützt fühlten. Zudem sollen sie angeben, wie sich die Modellierung mehrerer aufeinanderfolgender Freigaben auf das Freigabemanagement ausgewirkt hat.

8.2 Durchführung und Ergebnisse

Umgebung der Fallstudie

Die Fallstudie wurde im Rahmen eines Seminars an der Technischen Universität München mit Studenten als Probanden durchgeführt, indem Softwareprojekte mit mehreren Freigaben simuliert wurden. In diesen Softwareprojekten übernahmen die Probanden eine der folgenden Rollen im Projekt: Kunden, Analysten, Architekten, Systementwickler, Tester, Projektmanager und Freigabemanager. Zur Unterstützung bekam jede Rolle eine Liste mit vorgeschlagenen Handlungsanweisungen zu ihrem Verhalten im Projekt; beispielsweise bekamen Kunden eine Liste möglicher Anforderungen für das zu modellierende System. Die Probanden wurden zudem explizit aufgefordert, sich in ihre Rolle zu versetzen und diese aktiv selbst zu gestalten. Während der Fallstudie setzten die Probanden EXPLoRE ein, um Freigaben begründungsbasiert zu managen.

Kenntnisse der Probanden

Alle Probanden verfügen über Grundkenntnisse im Software Engineering, die sie in ihrem Studium auch bereits praktisch eingesetzt haben. Im Freigabemanagement sind die Probanden als Anfänger einzustufen. Um sie an das Thema heranzuführen, wurden im Seminar theoretische Ansätze und Werkzeuge für das Freigabemanagement vorgestellt. Als weitere Vorbereitung auf die Fallstudie führten die Probanden Übungen mit EXPLoRE durch, um sich mit der Bedienung und Funktionsweise des Werkzeugs vertraut zu machen.

Diese Übungen waren in zwei Teile aufgeteilt: Der erste Teil fokussierte die Systemmodellierung, Kommunikationsmodelle sowie die Modelle der Organisation. Im zweiten Teil setzten die Probanden das begründungsbasierte Freigabemodell für Freigabemanagement und Abstraktionsgrade ein.

Verteilung der Probanden

Während des Projekts waren die Probanden in zwei Räume aufgeteilt, um die Kommunikation durch EXPLoRE zu fördern: Kunden, Analysten und Projektmanager waren in einem Raum, während der Freigabemanager zusammen mit Architekten, Systementwickler und Testern in einem anderen Raum arbeitete. Jeweils ein Experte beobachtete die Arbeit der Probanden in jedem Raum und half bei Problemen mit der Bedienung des Werkzeugs.

Durchführung der Fallstudie in zwei Projekten

Dieselben Probanden führten die Fallstudie zwei Mal hintereinander für zwei Projekte durch, um mehr Daten sammeln zu können und dadurch die Aussagekraft der Fallstudie zu erhöhen. Außerdem konnten die Probanden im zweiten Projekt bereits auf eine größere Erfahrung im Freigabemanagement sowie der Funktionalität und Bedienung von EXPLoRE zurückgreifen. Die Rollen wurden nach der ersten Fallstudie getauscht, um die erfassten Daten und Beobachtungen möglichst unabhängig von den Probanden zu halten. Nach beiden Projekten erfolgte eine Befragung der Probanden. In beiden Projekten arbeiteten die Probanden motiviert mit und ergänzten die initialen Vorschläge für die Ausführung ihrer Rollen mit eigenen Ideen.

Verteilung der Änderungen

Im ersten Projekt entwickelten die Probanden fünf Freigaben eines Multimedia-Mobiltelefons, im zweiten Projekt vier Freigaben eines Infotainment-Systems für ein Fahrzeug. Bei der Entwicklung des Multimedia-Mobiltelefons wurden insgesamt über 100 Änderungen beobachtet. Von diesen resultierten etwa 47% aus Fehlern, 40% aus sich ändernden Anforderungen, 7% aus der Systemarchitektur oder detailliertem Entwurf sowie 3% aus organisatorischen Gründen. Während der Entwicklung des Infotainment-Systems für ein Fahrzeug beobachteten die Experten 83 Änderungen: 37% der Änderungen kamen aus den Anforderungen, 25% der Änderungen aus der Systemarchitektur oder dem detaillierten Entwurf, 5% aus der Organisation und 33% aus Fehlern. (siehe Abb. 98)

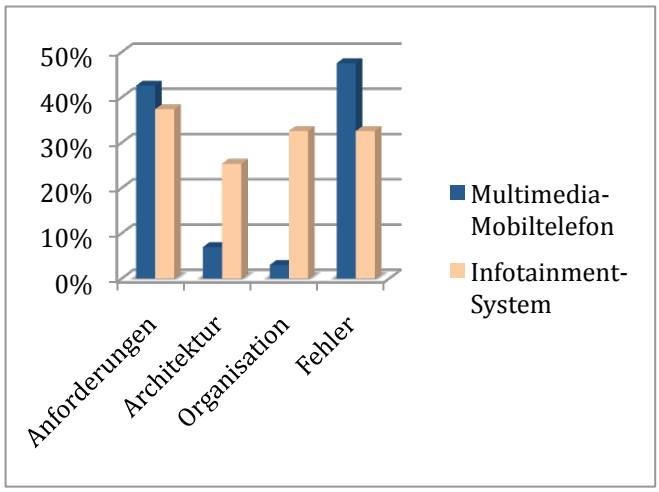


Abb. 98: Beobachtete Arten von Änderungen in der Fallstudie für die Projekte Multimedia-Mobiltelefon und Infotainment System.

8.2.1 Komplexität der Abhängigkeiten

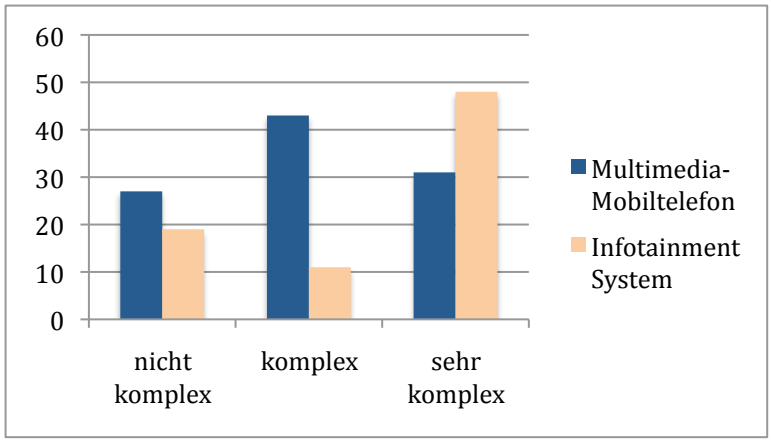


Abb. 99: Häufigkeiten für die Komplexität der Abhängigkeiten bei einer Änderung.

Häufigkeits-verteilung

Abb. 99 zeigt die Häufigkeiten für die Komplexität der Abhängigkeiten: Im Projekt des Multimedia-Mobiltelefons waren bei 27 Änderungen die Abhängigkeiten nicht komplex, bei 43 Änderungen die Abhängigkeiten komplex und bei 31 Änderungen sehr komplex. Der Mittelwert liegt bei 5,1 mit einer Standardabweichung von 4,5. Im Projekt des Infotainment-Systems wurden bei 19 Änderungen keine komplexen Abhängigkeiten analysiert, bei 11 Änderungen komplexe Abhängigkeiten und bei 48 Änderungen sehr komplexe

Abhängigkeiten. Der Mittelwert ist hier 8,2 mit einer Standardabweichung von 10,8.

Mittelwert und Standard-abweichung

Die Mittelwerte für die Komplexität der Abhängigkeiten liegen in beiden Projekten im Bereich der sehr komplexen Abhängigkeiten. Die mit 4,5 bzw. 10,2 hohen Standardabweichungen relativieren die Mittelwerte, denn einige Abhängigkeiten werden dadurch sogar nicht komplex sein. Allerdings unterscheidet die Standardabweichung nicht, in welche Richtung die Werte abweichen. Da das Maß für die Komplexität nach unten bei 0 beschränkt, jedoch nach oben offen ist, beeinflusst eine besonders hohe Abweichung, beispielsweise eine Komplexität von 52 beim Infotainment-System, die Standardabweichung stärker als nicht komplexe Abhängigkeiten. Gerade derart komplexe Abhängigkeiten bedürfen der Unterstützung durch ein Werkzeug, das die Abhängigkeiten abbildet und den Anwender visualisiert.

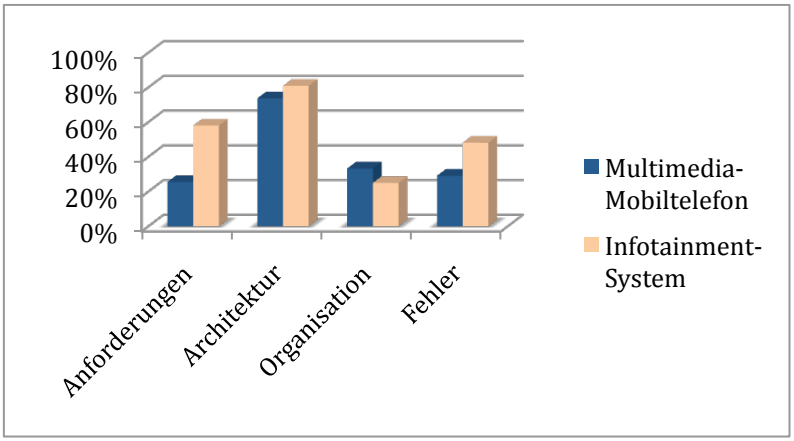


Abb. 100: Änderungen, bei denen komplexe Abhängigkeiten berücksichtigt werden müssen.

Verteilung auf Änderungsarten

Sehr komplexe Abhängigkeiten im Systemmodell wurden bei etwa 30% der Änderungen im Projekt des Multimedia-Mobiltelefons identifiziert: 25% der Änderungen aus den Anforderungen, 60% der Änderungen aus der Architektur, etwa 30% der Änderungen aus der Organisation sowie 30% der Änderungen wegen Fehlern wiesen sehr komplexe Abhängigkeiten auf. Im Projekt des Infotainment-Systems wurden bei knapp 60% der Änderungen komplexe Abhängigkeiten im Systemmodell berücksichtigt: 58% der Änderungen aus den Anforderungen, 81% der Änderungen aus der Systemarchitektur oder detaillierten Modellen, 25% der Änderungen aus der Orga-

nisation sowie 48% der Änderungen durch Fehlerberichte hatten komplexe Abhängigkeiten (siehe Abb. 100).

Im Projekt des Multimedia-Mobiltelefons war die Anzahl der komplexen Abhängigkeiten höher als erwartet. Die Beobachtungen zeigten, dass die erfassten Systemmodelle mit ihren Abhängigkeiten die Probanden gut unterstützten. Die Abhängigkeiten des Multimedia-Mobiltelefons waren so komplex, dass sie von einzelnen Personen nicht mehr überschaut werden konnten. Ohne die Unterstützung von EXPLoRE bei der Abhängigkeitsanalyse werden betroffene Modellelemente übersehen und somit Auswirkungen nicht berücksichtigt.

Die Durchführung der Fallstudie mit dem Infotainment-System wies darauf hin, dass vor allem bei Änderungen aus der Systemarchitektur oder dem Objektentwurfsmodell komplexe Abhängigkeiten zu berücksichtigen sind, deren Auswirkungen Personen nicht einfach bestimmen können. Die einer Freigabe zugrundeliegenden Systemmodelle auf verschiedenen Abstraktionsgraden unterstützen das Auffinden abhängiger Modellelemente.

8.2.2 Freigabeoperationen

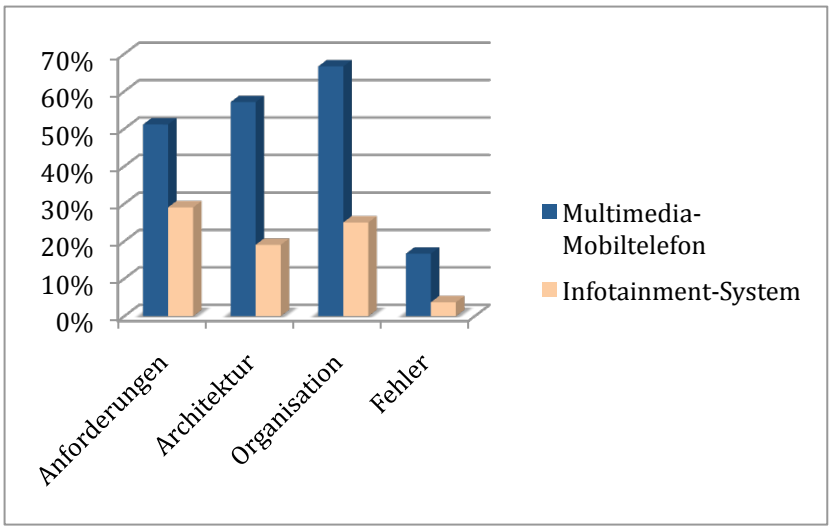


Abb. 101: Änderungen nach ihren Arten, die zu einer Änderung der Roadmap durch Freigabeoperationen führten.

In beiden Projekten führte etwa ein Drittel aller Änderungen zu einer Anpassung der Roadmap durch Freigabeoperationen. Wie Abb. 101 illustriert, hatten beim Multimedia Player etwa 50% der Änderungen in den Anforderungen, 50% Änderung im Systementwurf, 66% der organisatorischen Änderungen und 15% der Fehler eine Freigabeoperation zur Folge. Beim Infotainment-System waren es 29% der Änderungen in den Anforderungen, 33 % der Änderungen in der Systemarchitektur, 33% der Änderungen in der Organisation und 4 % der Änderungen durch Fehlerberichte.

Vorkommen von Freigabeoperationen

Änderungen führten unabhängig von ihrer Art zu einer Anpassung der Roadmap durch Freigabeoperationen, wobei Fehlerberichte nur in wenigen Fällen, die übrigen Änderungen häufiger Freigabeoperationen zur Folge hatten. Beobachtungen zeigten, dass die Probanden nach Ausführung von Freigabeoperationen mit den durchgeführten Änderungen an Freigabedeskriptoren zufrieden waren.

8.2.3 Freigabeszenarios

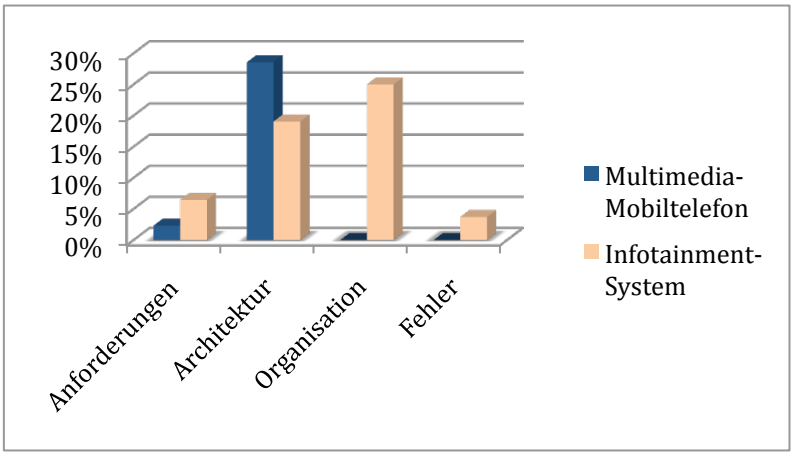


Abb. 102: Änderungen nach ihren Arten, die zu einem Freigabeszenario führten

Freigabeszenarios wurden im Projekt des Multimedia Mobiltelefons drei Mal eingesetzt, einmal wegen einer Änderung in den Anforderungen, zwei Mal wegen einer Änderung in der Technologie. In zwei der drei Fällen wiesen die beobachteten Systemmodelle der Freigabeszenarios bedeutende Unterschiede auf und dienten so als Grundlage für die Entscheidung, weil echte Al-

Multimedia Mobiltelefon

ternativen verglichen wurden. Im dritten Fall versuchte der Proband, wie die Erstellung des Freigabeszenarios funktioniert und welche Möglichkeiten es ihm bietet.

Infotainment- System

Für das Infotainment-System wurden Freigabeszenarios in 10% der Änderungen erstellt. 6% der Änderungen in den Anforderungen, 19% der Änderungen in der Architektur, 25% der Änderungen in der Organisation und 4% der Fehler führten zu einem Freigabeszenario. Diese setzten die Probanden ein, um die Auswirkungen von Änderungen in der Architektur oder dem Objektentwurf mit Alternativen zu modellieren und zu vergleichen.

In beiden Projekten verwendeten die Probanden Freigabeszenarios, um die Auswirkungen von Änderungen zu analysieren, Unsicherheiten zu entgegnen und Alternativen zu diskutieren. Die Probanden erstellten Modelle für Alternativen, um diese zu erkunden und miteinander zu vergleichen. Die Tatsache, dass im zweiten Projekt mehr Freigabeszenarios erstellt wurden, ist auf die gewonnenen Erfahrungen im Umgang mit EXPLoRE zurückzuführen.

8.2.4 Befragung

Fragebogen

Die Befragung der Probanden durch einen Fragebogen zielt darauf ab, eine direkte Rückmeldung über das begründungsbasierte Freigabemanagement zu erhalten. Konkret erkundete der Fragebogen, wie nützlich die Probanden die erfassten Systemmodelle und Abhängigkeiten sowie die Systemmodellierung auf mehreren Abstraktionsgraden im Hinblick auf die Herausforderungen im Freigabemanagement finden. Die Probanden beurteilten den Einfluss der Änderungspropagation auf die Konsistenzerhaltung und die Förderung der Zusammenarbeit. Sie gaben an, wie hilfreich Freigabeoperationen und Freigabeszenarios in den durchgeführten Projekten waren.

Interpretation leerer Angaben

Die folgende Auswertung der Befragung zeigt, dass sich die Befragten oft keine Meinung zu bestimmten Fragen bilden konnten. Das ist so zu erklären, dass jeder Proband pro Projekt nur eine Rolle ausführte und somit nicht mit allen Konzepten des begründungsbasierten Freigabemanagements in Berührung kam. Beispielsweise kann ein Kunde den Einfluss der Änderungspropagation auf die Konsistenz der Modelle nicht beurteilen.

Systemmodellierung

Die Befragung der Probanden ergibt, dass die Erfassung der Systemmodelle mit ihren Abhängigkeiten für 80% der Probanden im Hinblick auf die Herausforderungen im Freigabemanagement nützlich ist. 10% geben an, dass sie dadurch nicht unterstützt werden; weitere 10% können die Frage nicht beurteilen. Auf die Frage nach dem Vorteil der Systemmodellierung auf mehreren Abstraktionsgraden antworten 70%, dass sie für ihre Tätigkeiten nützlich ist; 10% finden sie nicht nützlich und weitere 20% beantworten die Frage nicht. 20% der Probanden geben zudem an, dass Abstraktionsgrade für Systemmodelle die Kommunikation mit anderen Interessenvertretern sehr stark unterstützt hat und 10% fühlen sich in der Kommunikation durch diese Modelle gut unterstützt, während 20% sie weniger unterstützend fanden. Die übrigen Probanden machten zu dieser Frage keine Angabe.

Änderungspropagation

Zwei weitere Fragen zielen auf eine Beurteilung der Änderungspropagation ab: 60% der Probanden finden, dass dieser Mechanismus einen positiven (10%) oder sehr positiven (50%) Einfluss auf die Erhaltung der Konsistenz der Modelle aufeinanderfolgender Freigaben und verschiedener Abstraktionsgrade hat. Die übrigen Teilnehmer können die Frage auf Grund ihrer Rollen in der Fallstudie nicht beurteilen. Die zweite Frage erforscht die Förderung der Zusammenarbeit durch die Änderungspropagation: 40% geben an, über Änderungen anderer Projektteilnehmer sehr gut informiert gewesen zu sein, 30% waren gut informiert. Weitere 30 % beantworten die Frage nicht.

Roadmaps und Freigabeszenarios

70% der befragten Probanden geben an, dass die Modellierung aufeinanderfolgender Freigaben zusammen mit den angebotenen Freigabeoperationen zum Verschieben von Freigabeplanungsentitäten für sie hilfreich oder sehr hilfreich ist. Die verbleibenden 30% können die Frage nicht beurteilen. Die Befragung der Probanden zeigt, dass Freigabeszenarios mehr als 20% der Befragten sehr unterstützen und mehr als 60% unterstützen. 10% der Probanden gaben an, dass Freigabeszenarios für sie keine Unterstützung darstellen und die übrigen können die Frage nicht beurteilen. Die Qualität der Entscheidungen des Freigabemanagers beurteilen 80% der befragten Projektteilnehmer mit gut und 20% mit ausreichend.

8.2.5 Validität

Ergebnisse

Die Evaluierung bestand aus einer eingebetteten Mehrfachfallstudie, die für zwei Softwareprojekte ausgeführt worden ist. Der Fokus der Fallstudien lag auf den Auswirkungen von Änderungen in einer Freigabe; insgesamt wurden 184 Änderungen als Fälle untersucht, um die Aussagekraft zu erhöhen. Die Fallstudie beobachtete, dass die Erfassung von Systemmodellen und deren Abhängigkeiten, Freigabeoperationen und Freigabeszenarios als Begründungen im Freigabemanagement dienen und den Freigabemanager bei Entscheidungen unterstützen. Das Modell zum begründungsbasierten Freigabemanagement erfasst Abhängigkeiten, die für Entscheidungen im Freigabemanagement wichtig sind. Der Freigabemanager kann so die Auswirkungen einer Änderung beurteilen, Freigabeszenarios zur Abwägung erstellen und dann eventuell Freigabeoperationen ausführen, um die Freigabepläne und Freigabedeskriptoren anzupassen.

Bewertung der Projekte

Da es sich in den Fallstudien um Studentenprojekte handelte, kann die Verteilung der Arten der Änderungen von denen eines Industrieprojekts anweichen. Dies beeinflusst jedoch die Ergebnisse nicht, da der Fokus der Interpretation darauf liegt, wie sich eine konkrete Änderung auf die zu untersuchenden Behauptungen auswirkt. So zeigt die Evaluierung, dass eine Änderung in der Architektur des Systems sehr häufig komplexe Abhängigkeiten hervorruft. Die Erfassung von Systemmodell und deren Abhängigkeiten unterstützt das Freigabemanagement hier, weil Projektteilnehmer oder Teams nicht alle kritischen Abhängigkeiten eines Software-Systems überblicken können.

9 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse der Dissertation zusammen und gibt einen Ausblick auf weitere Forschungsfragen im Freigabemanagement.

9.1 Zusammenfassung

Freigabemodell

Ein Beitrag dieser Dissertation ist das Freigabemodell, das Entscheidungen des Freigabemanagers bei häufigen Änderungen im Freigabemanagement erheblich verbessert. Diese Verbesserung erreicht das Freigabemodell, indem es die Zusammenhänge zwischen Roadmaps, Freigabeplänen, Konfigurations- und Buildmanagement sowie Systemmodellen darstellt. Der Vorteil dieser Darstellung ist die Nachvollziehbarkeit von Roadmaps und Freigaben über Versionen und *builds* zu Systemmodellen. Roadmaps sind als Folge von Freigaben definiert, von denen jede für sich mit Knowledge Nuggets beschrieben wird. Durch diese Knowledge Nuggets können Freigabepläne generiert und bei Änderungen auch automatisch neu erstellt werden, wodurch sie stets die aktuellen Änderungen in den Systemmodellen widerspiegeln. Änderungen aus den Systemmodellen führen also unmittelbar zu einer Änderung des Freigabeplans und informieren den Freigabemanager über Änderungen in der Systementwicklung. Dies erhöht die Qualität von Entscheidungen des Freigabemanagers, denn diese basieren nicht mehr wie beim Einsatz mehrerer Werkzeuge auf veralterten, inkonsistenten Daten.

Freigabeoperationen

Das Freigabemodell erleichtert die Arbeit des Freigabemanagers durch besondere Freigabeoperationen zur Anpassung von Freigaben bei Änderungen. Diese Freigabeoperationen verwenden Nachvollziehbarkeit, um bei Änderungen die Knowledge Nuggets mehrerer Freigaben anzupassen, wodurch die Systementwickler über Änderungen des Freigabemanagers informiert sind. Das Freigabemodell ändert also automatisch, das heißt ohne Betrachtung jeder einzelnen Abhängigkeit, die Knowledge Nuggets von Freigaben. Der Vorteil dieses Vorgehens ist, dass die Arbeit des Freigabemanagers und der Systementwickler auf den gleichen, konsistenten Modellen beruht.

Begründungsbasiertes Freigabemodell

Das begründungsbasierte Freigabemodell führt Freigabeszenarios zur Beschreibung von alternativen Lösungsvorschlägen für eine Freigabe ein. Das begründungsbasierte Freigabemodell verbessert die Entscheidungsfindung für ein bestimmtes Freigabeszenario durch Kriterien, System-, Kommunikations- und Organisationsmodelle. Kriterien sind nicht durch das Modell festgelegt, sondern können nach den Bedürfnissen von Projekten, Systemen oder Interessenvertretern frei definiert werden, um Freigabeplanungsentitäten beispielsweise nach betriebswirtschaftlichen Gesichtspunkten zu bewerten. Systemmodelle zeigen die Struktur des Systems und machen Abhängigkeiten zwischen den Modellelementen explizit. Kommunikationsmodelle fassen Diskussionen und Begründungen für Entscheidungen und reflektieren somit wichtige Elemente eines Softwareprojekts. Organisationsmodelle können Abhängigkeiten zwischen Mitarbeitern und dem System abbilden und auf mögliche personelle Engpässe hinweisen.

Knowledge Nuggets

Die Beschreibungen einer Freigabe oder eines Freigabeszenarios verwalten Knowledge Nuggets: Knowledge Nuggets erlauben die Erstellung einer individuellen Beschreibung sowohl jeder Freigabe als auch jedes Freigabeszenarios eines Software-Systems, indem sie die Systemmodelle einer Freigabe zusammenhalten. Der Einsatz mehrerer Knowledge Nuggets pro Freigabe ermöglicht darüber hinaus die Freigabemodellierung auf verschiedenen Abstraktionsgraden und macht damit Begründungen aus Modellen der Analyse, Systemarchitektur oder der Implementierung explizit, die für die Entscheidungsfindung wichtig sind. Neben Systemmodellen fassen Knowledge Nuggets Kommunikations- und Organisationsmodelle einer Frei-

gabe zusammen und ziehen somit neben Systemmodellen auch Wissen aus Kommunikation oder Beschränkungen der Organisation in die Entscheidungsfindung mit ein und führen so zu besseren Entscheidungen.

Konsistenzerhaltung

Knowledge Nuggets verfügen über zwei Mechanismen zur Konsistenzerhaltung zwischen den Beschreibungen aufeinanderfolgender Freigaben: Zum einen propagieren sie anhand einer definierten Ordnung auf Knowledge Nuggets relevante Änderungen zu den Modellen niedrigeren Abstraktionsgrads oder späterer Freigaben. Änderungen müssen daher nicht in das Modell jeder Freigabe einzeln, sondern nur einmal eingearbeitet werden. Zum anderen unterstützen Knowledge Nuggets das Verschieben von Freigabeplanungsentitäten zwischen Freigaben, wenn Änderungen eintreten, indem sie zusammen mit einer Freigabeplanungsentität andere Modellelemente, die von ihr abhängig sind, zu den Knowledge Nuggets der neuen Freigabe verschieben. Knowledge Nuggets für Roadmaps, Freigaben, alternative Freigabeszenarios und Modelle unterschiedlicher Abstraktionsgrade stellen zusammen mit Versionen, Kriterien, System-, Kommunikations- und Organisationsmodellen machen die Dimensionen der Entscheidungsfindung explizit.

Visualisierungen in EXPLoRE

Das im Rahmen dieser Dissertation entwickelte Werkzeug EXPLoRE erlaubt im Gegensatz zu existierenden Werkzeugen die Berücksichtigung von alternativen Systemmodellen im Freigabemanagement. EXPLoRE visualisiert den mehrdimensionalen Entscheidungsraum mit Hilfe einer dreidimensionalen Visualisierung, so dass der Freigabemanager durch die Begründungen navigieren kann. Zudem hebt eine direkte Vergleichsansicht dem Freigabemanager Unterschiede zwischen den Modellen alternativer oder aufeinanderfolgender Freigaben farblich hervor. EXPLoRE stellt die zur Entscheidungsfindung notwendigen Begründungen damit klar und strukturiert dar und reduziert damit die Herausforderungen im Freigabemanagement.

Verbesserte Entscheidungsfindung in Fallstudie

Dies bestätigte der Einsatz von EXPLoRE in einer Fallstudie; in den durchgeführten Projekten wurden die Auswirkungen von 184 Änderungen beobachtet. Bei 80 Änderungen trug EXPLoRE zu einer verbesserten Entscheidungsfindung bei, weil die Teilnehmer komplexe Abhängigkeiten

verfolgten. Die Teilnehmer setzten Freigabeszenarios bei kritischen Fragestellungen aktiv ein, um alternative Lösungsvorschläge zu modellieren. In der Fallstudie führten 61 der 183 Änderungen nach der Analyse der Abhängigkeiten oder Erkundung der Freigabeszenarios zu einer Anpassung der Roadmap, welche nach Aussagen der Teilnehmer dazu beitrugen, dass die Konsistenz zwischen Freigabeplänen und Systemmodellen erhöht wurde.

9.2 Ausblick

Berücksichtigung des Quellcodes

Das begründungsbasierte Freigabemodell sieht Systemmodelle auf mehreren Abstraktionsgraden vor, um Entscheidungen im Freigabemanagement zu unterstützen. Gerade kurz vor der Lieferung besteht die Tendenz, dass Modelle an Bedeutung verlieren und der Quellcode das System dokumentiert. Durch die kurzen Lieferzyklen agiler Prozesse steht das Entwicklungsteam stets vor der Herausgabe der nächsten Freigabe, so dass man untersuchen könnte, inwieweit die Konsistenz zwischen Systemmodellen verschiedener Knowledge Nuggets und dem Quellcode verbessert werden kann. Mögliche Ansätze sind die Anbindung eines *Reverse Engineering* Werkzeugs an EXPLoRE, um die Konsistenz zwischen Quellcode und Modellen zu pflegen oder die Vereinung von Knowledge Nuggets und *Model Driven Architecture*.

Knowledge Nuggets für Builds

Konsistenz zwischen Quellcode und Systemmodellen wie oben beschrieben, könnte dafür eingesetzt werden, einzelne *builds* zu dokumentieren. Denn BEEF beschreibt bislang jede Freigabe durch eigene Systemmodelle, um Versionen, die an den Kunden geliefert werden zu planen und zu dokumentieren. Die kontinuierliche Integration erstellt jedoch für jede Version im Konfigurationsmanagement einen *build*, der getestet und intern freigegeben wird. Knowledge Nuggets zusammen mit einer Anbindung an den Quellcode können ein Modell für jeden *build* zu erzeugen.

Knowledge Nuggets für Produktlinien

Ein weiteres Forschungsproblem ist die Untersuchung der Einsetzbarkeit von Knowledge Nuggets für Produktlinien. Eine Produktlinie [Pohl et al. 2005] ist eine Variante eines Softwaresystems; Produktlinien haben weitgehende Gemeinsamkeiten, beispielsweise basieren sie auf der gleichen Plattform, unterscheiden sich jedoch in entscheidenden, für den Anwender

bedeutenden, Eigenschaften. Ein Beispiel für Produktlinien sind Basis, Standard und Luxus Varianten eines Software-Systems. Produktlinien bilden einen weiteren Einsatzbereich von Knowledge Nuggets: Sie erlauben die Darstellung eines einzelnen Systemmodells für jede Variante, wenn Knowledge Nuggets Änderungen an gemeinsamen Systemteilen propagieren. Dadurch stünde für die Entwicklung, Qualitätssicherung und Dokumentation ein konkretes Systemmodell für jede Variante bereit.

Knowledge Nuggets für Demos

Ein weiteres Einsatzgebiet für Knowledge Nuggets ist das Management von Demos. Demos sind interne oder externe Freigaben und haben das Ziel den Projektfortschritt zu demonstrieren und früh Rückmeldungen über Änderungen an dem demonstrierten System zu erhalten. Neben der Demonstration des lauffähigen Systems informieren Demos die Interessenvertreter auch über Systemmodelle, die das System darstellen. Diese Systemmodelle zeigen abhängig vom Ziel und Interessenvertreter vereinfachte Darstellungen des Analyse-, Systementwurfs- oder Objektentwurfsmodells. Ein Knowledge Nugget kann die Systemmodelle für eine Demo verwalten ohne die Beschreibung einer Freigabe ändern zu müssen. Das begründungsbasierte Freigabemodell kann die Entwicklung einer Demo unterstützen, da Demos letztlich eine Freigabe eines Systems darstellen. Freigaben für Demos verwenden häufig Stubs und Driver, um noch nicht entwickelte Teile des Systems zu simulieren. Das Demomanagement muss also mit Stubs und Drivers umgehen können und wissen, welche Teile eines zu demonstrierenden Szenarios schon implementiert sind oder welche noch simuliert werden.

Literaturverzeichnis

[Abbott 1983] R. Abbott: „Program design by informal English descriptions." In: *Communications of the ACM*, Vol. 26, No. 11, 1983.

[AcceptSoftware 2008] AcceptSoftware 2008: *„Accept 360°"*, 2008. Verfügbar unter: http://www.acceptsoftware.com/pdf/whitepapers/ExcellenceWhitePaper08.pdf [30.04.2008]

[Amandeep et al. 2004] A. Amandeep, G. Ruhe, M. Stanford: „Intelligent Support for Software Release Planning". In: Frank Bomarius, Hajimu Iida (Hrsg.): *„Product Focused Software Process Improvement: 5th International Conference, PROFES 2004, Kansai Science City, Japan, April 5-8, 2004, Proceedings "*, Lecture Notes in Computer Science Vol. 3009, Springer, Kansai Science City, 2004.

[Balzert 2001] Helmu Balzert: „Lehrbuch der Softwaretechnik – Software-Entwicklung". 2. Auflage, 1. Nachdruck, Springer, Berlin 2001.

[Bagnall et al. 2001] A.J. Bagnall, V.J. Rayward–Smith, I.M. Whittley: "The Next Release Problem". In: *„Information and Software Technology"*, 43(14), pp. 883–890, 2001.

[Bays 1999] Michael E. Bays: Software Release Methodology. Prentice Hall, Upper Saddle River (NJ), 1999.

[Beck 2001] Kent Beck: „Extreme Programming Explained", Addison Wesley, 2001.

[Berenbach & Wolf 2007] Brian Berenbach, T. Wolf: “A unified requirements model; integrating features, use cases, requirements, requirements analysis and hazard analysis“. In: „*Proceedings of the Second IEEE International Conference on Global Software Engineering (ICGSE'07)*“, München, August 2007.

[Berliner 1990] B. Berliner: „CVS II: Parallelizing software development“. In: Proceedings of the 1990 USENIX Conference, Washington, DC, pp. 22-26, 1990.

[Boehm 1988] B. W. Boehm: „ A spiral model of software development and enhancement“. In: *„IEEE Computer“*, Vol. 21, S. 61-72, May 1988.

[Bonsai 2008] Bonsai Projekt-Homepage. http://bonsai.mozilla.org/ [23.09.2008]

[Bruegge & Dutoit 2003] Bernd Bruegge and Allen H. Dutoit: „Object-Oriented Software Engineering Using UML, Patterns, and Java.“ Prentice Hall, Englewood Cliffs, NJ, 2. Auflage, 2003.

[Bruegge et al. 2005] Bernd Bruegge, Korbinian Herrmann, Axel Rauschmayer, Patrick Renner: „Situational Requirements Engineering Gets Distributed!“. In: „Proceedings of the International Workshop on Distributed Software Development“, 2005.

[Bruegge et al. 2007] B. Bruegge, K. Herrmann, I. Bonev, F. Schneider: “Entscheidungsunterstützungswerkzeug für Release Management”. Technischer Bericht TUM-I0713, München, 2007.

[Buxton & Randell 1970] J. N. Buxton and B. Randell (Hrsg.): „Software Engineering Techniques - Report on a conference sponsored by the Nato Science Commitee“. Rom, 1970.

[Carlshamre et al. 2001] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, J. Natt och Dag: “An Industrial Survey of Requirements Interdependencies in Software Release Planning”. In Proc. of Fifth IEEE Int. Symposium on Requirements Engineering (RE´01), Aug. 27-31, Toronto, Canada, 2001.

[Carlshamre 2002] P. Carlshamre: “Release Planning in Market-driven Software Product Development - Provoking an Understanding”, In: “*Requirements Engineering Journal*”, 7(3), pp.139-151, London, September 2002.

[Chatfield & Johnson 2007] Carl Chatfield, Timothy Johnson: "Microsoft Office Project 2007 Step by Step“. Microsoft Press, Redmond (WA), 2007.

[Coad et al. 1999] Peter Coad, Eric Lefebvre, Jeff de Luca: „Java Modeling in Color with UML – Enterprise Components and Process“, Prentice Hall, Upper Saddle River, 1999.

[Cohen 2007] Peter Cohen: „Apple delays Leopard until October because of iPhone“. Verfügbar unter: http://www.macworld.com/article/57321/2007/04/leoparddelay.html [12.4.2007]

[Cohn 2005] Mike Cohn: „Agile Estimating and Planning“. Prentice Hall, 2005.

[Collins-Sussman et al. 2004] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato,: „ Version Control with Subversion“. O'Reilly, Sebastopol, CA, 2004.

[Continuum 2008] Continuum-Hompage. http://continuum.apache.org/ [01.05.2008]

[Cowan 2001] N. Cowan: The magical number 4 in short-term memory: A reconsideration of mental storage capacity. Behavioral and Brain Sciences, 24, 87-185. 2001.

[Cruise Control 2008] Cruise Control-Homepage. http://cruisecontrol.sourceforge.net/ [01.10.2008]

[Cusumano & Selby 1997] M. A. Cusumano, R. W. Selby: "How Microsoft Builds Software". Communications of the ACM, Vol. 40, No. 6, pp. 53–61, 1997.

[Demarco 1979] Tom Demarco: „Structured Analysis and System Specification.“ Yourdon Press, Prentice Hall, 1979.

[Denne & Cleland–Huang 2004] M. Denne, J. Cleland–Huang: „The Incremental Funding method: Data Driven Software Development", 21(3), pp. 39–47, 2004.

[Doors 2006] Doors-Homepage. http://www.telelogic.com/products/doors/index.cfm [02.11.2006]

[Dutoit et al. 2006] Allen H. Dutoit, Raymond McCall, Ivan Mistrík, Barbara Paech: „Rationale Management in Software Engineering – Concepts and Techniques“. In: Allen H. Dutoit, Raymond McCall, Ivan Mistrík, Barbara Paech: „*Rationale Management in Software Engineering*“, Springer, Heidelberg, 2006.

[Duvall et al. 2007] Paul Duvall, Steve Matayas, Andrew Glover : Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley Professional, 2007.

[Easterbrook et al. 2008] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, Daniela Damian: „Selecting Empirical Methods for Software Engineering Research“. In: Forrest Shull, Janice Singer, Dag I. K. Sjoberg (Hrsg.): „Guide to Advanced Empirical Software Engineering“, Springer, London 2008.

[Elsässer 2006] Wolfgang Elsässer: “ITIL einführen und umsetzen - Leitfaden für effizientes IT-Management durch Prozessorientierung“. Hanser-Verlag, München, 2006.

[Fowler 2004] Martin Fowler: „UML distilled : A Brief Guide to the Standard Object Modeling Language“. 3. Auflage, Addison-Wesley, 2004.

[Fowler 2006] Martin Fowler: „Continuous Integration“. Verfügbar unter: http://www.martinfowler.com/articles/continuousIntegration.html [01.05.2006].

[Gamma et al. 2004] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: „Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software“, Addison-Wesley, München, 2004.

[Gantt 1910] Henry L. Gantt: „Work, wages, and profits.“ In: „*The Engineering Magazine“*, New York, 1910.

[Gold 2004]: Virginia Gold: „ACM Honors Creator of Landmark Software Tool“. New York, 2004. Verfuegbar unter: http://www.acm.org/announcements/softwaresystemaward.3-24-04.html

[Gosling et al. 2000] James Gosling, Bill Joy, Guy Steele, Gilad Bracha: „The Java language specification“, 2. Auflage, Addison-Wesley, New York, 2000.

[Greer 2004] D. Greer: “Decision Support for Planning Software Evolution with Risk Management”, In: “*Proc. 16th International Conference on Software Engineering and Knowledge Engineering”*, pp. 503-507, June, 2004.

[Greer & Ruhe 2004] D. Greer, G. Ruhe: "Software Release Planning: An Evolutionary and Iterative Approach". In: „Information and Software Technology", Vol. 46, pp. 243-253, 2004.

[Grimm 1998] Todd Grimm :„The Human Condition: A Justification for Rapid Prototyping." Time Compression Technologies, vol. 3 no. 3. Accelerated Technologies, Inc. May 1998

[Gürsoy 2007] Halil-Cem Gürsoy: „Kontinuierlich zum Ziel - Eine Einführung in Maven Continuum". In: Java Magazin, Frankfurt, 2007.

[Herrmann & Bruegge 2006] Korbinian Herrmann, Bernd Bruegge: „Visualization of Release Planing." In: *"Proceedings of 1st International Workshop on Requirements Engineering Visualization" (REV'06), IEEE Requirements Engineering*, Minneapolis-St.Paul, Minnesota, 2006.

[Herrmann 2007] Korbinian Herrmann: „Release Planning Gets Distributed" In: *„Proceedings of 1st International Global Requirements Engineering Workshop (GREW)"*, München, 2007.

[Herrmann 2007b] Korbinian Herrmann: „Roundtrip Roadmapping: From the Roadmap to the Model and Back Again". In: Anread Herrmann, Maya Daneva, Christof Ebert, Chris Rupp (Hrsg): „Proceedings of the MEREP'07: Workshop on Measuring Requirements for Project and Product Success, Palma de Mallorca, 2007.

[Hernandez 2008] Tara Hernandez: „The Bonsai Project". Verfügbar unter: http://www.mozilla.org/projects/bonsai/ [23.09.2008]

[Holzner 2005] Steve Holzner: „Ant: The Definitive Guide". O'Reilly, 2. Auflage, Sebastopol, 2005.

[IEEE 1987] IEEE Guide to Software Configuration Management, IEEE Standards Board, September 1987 (Reaffirmed, Dezember 1993). In: IEEE: „IEEE Standards Collection Software Engineering." Piscataway (NJ), 1997.

[InStep 2008] InStep-Homepage. http://www.microtool.de/inStep/ [16.05.2008]

[Jacobson et al. 1992] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Övergaard: „Object-Oriented Software Engineering – A Use Case Driven Approach." Addison-Wesley, New York, 1992.

[Jacobson et al. 1994] Ivar Jacobson, Maria Ericsson, Agneta Jacobson: „The Object Advantage – Business Process Reengineering With Object Technology“. Addison-Wesley, New York, 1994.

[Jacobsen et al. 1998] E. E. Jacobsen, B. B. Kristensen, & P. Nowack: Models, Domains, and Abstractions in Software Development. In: „Proceedings of International Conference on Technology of Object-Oriented Languages and Systems“, Bejing, China, 1998.

[Jarke 1998] Matthias Jarke: „Requirements tracing“. In: *„Communications of the ACM“,* 41(12), S. 32–36, December 1998.

[Jensen & Tonies, 1979] R. W. Jensen & C. C. Tonies: “Software Engineering“. Prentice Hall, Englewood Cliffs, NJ, 1979.

[Kaarlas & Vähäniitty 2005] J. Kaarlas, Jarno Vähäniitty: „Tool support for software product and release planning - requirements and current solutions“, 2005. Verfügbar unter: http://www.soberit.hut.fi/~jvahanii/documents/Kaarlas_Vahaniitty_ToolSupport_WorkingPaper.pdf [07.07.2008].

[Karlsson & Ryan 1997] J. Karlsson, K. Ryan: *“A cost–value approach for Prioritising Requirements”*, IEEE Software, 14(5), pp.64–74, 1997.

[Kelter et al. 2008] Udo Kelter, Maik Schmidt, Sven Wenzel: „Architekturen von Differenzwerkzeugen“. In: Korbinian Herrmann (Hrsg.), Bernd Bruegge: “Software Engineering 2008 – Proceedings der Fachtachgung des GI-Fachbereichs Softwaretechnik“, Kölln Verlag , Bonn, 2008.

[Kraut & Streeter 1995] Robert E. Kraut & Lynn A. Streeter: *„Coordination in software development“*, Communications of the ACM, 38(3), Mar 1995.

[Kunz & Rittel 1970] W. Kunz & H. Rittel, “*Issues as elements of information systems*.” Working Paper No. 131, Institut für Grundlagen der Plannung, Universität Stuttgart, 1970.

[Larman 2004] Craig Larman: „Agile & iterative Development – A Manager’s Guide“. Addison-Wesley, Boston (MA), 2004.

[Lee 2003] Seok-Won Lee: „Software Evaluation Research: Case Study Methodology Designed Research, 2003. Verfügbar unter: http://www.sis.uncc.edu/~seoklee/Projects/CSM.htm

[Lehtola et al. 2007] L. Lehtola, M. Kauppinen, J. Vähäniitty: "Strengthening the link from business decisions to requirements engineering: Long-term product planning in software product companies". In: *"Proceedings of the 15th IEEE International Requirements Engineering Conference (RE'07)"*, New Delhi, India, Oct 2007.

[MacLean et al. 1991] Allan MacLean, Richard M. Young, Victoria M.E. Bellotti, Thomas P. Moran. „Questions, options, and criteria: Elements of design space analysis." HCI, 6(3-4):201–250, 1991.

[Matzke 2003] Bernd Matzke: „ Ant – Das Java Build Tool in der Praxis". Addison Wesley, München, 2003.

[Miller 1956] Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information Psychological Review, 63, 81-97.

[Mitchell & McKim 2002] Richard Mitchell, Jim McKim: „Desing By Contract, By Example". Addison-Wesley, Indianapolis, 2002.

[Momoh & Ruhe 2006] J. Momoh, G. Ruhe: "Release planning process improvement - an industrial case study", In: *„Software Process: Improvement and Practice"*, Volume 11, Issue3, pp 295-307, 2006.

[Mynatt 1990] Barbee Teasley Mynatt: „Softwrae Engineering with student-project guidance", Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[Naur & Randell 1969] Peter Naur, Brian Randell (Hrsg.): *„Software Engineering - Report on a conference sponsored by the NATO Science Committee"*, Garmisch, 1969.

[O'Brien et al. 2008] Tim O'Brien, John Casey, Brian Fox, Bruce Snyder, Jason Van Zyl, Eric Redmond: Maven. The Definitive Guide. O'Reilly Media, USA, 2008.

[Palmer & Felsing 2002] Stephen R. Palmer, John M. Felsing: „A practical Guide to Feature Driven Development", Prentice-Hall, Upper Saddle River (NJ), 2002.

[Penny 2002] D. A. Penny: „An Estimation-Based Management Framework for Enhancive Maintenance in Commercial Software Products". In: „Proceedings of International Conference on Software Maintenance (ICSM), pp. 122-130, 2002.

[Pews 2006] Gerhard Pews: „IT Projektmanagement – Planung II“, Karlsruhe, 2006. Verfügbar unter: http://wwwagse.informatik.uni-kl.de/teaching/pm/ws2006/material/2006-11-24%20-%2008%20-%20Planung%20II.pdf [16.05.2008]

[Pohl et al. 2005] Klaus Pohl, Günter Böckle, Frank van der Linden: „Software Product Line Engineering - Foundations, Principles and Techniques“, Springer, Heidelberg, 2005.

[Quatrani 2003] Terry Quatrani: „Visual modeling with Rational Rose 2002 and UML“, 3. Auflage, Addison-Wesley, 2003

[Ray 2003] Erik T. Ray: „ Learning XML“ 2. Auflage, O'Reilly, 2003.

[Royce 1970] W. W. Royce, “Managing the development of large software systems,” in Tutorial: Software Engineering Project Management, IEEE Computer Society, Washington, DC, pp. 118–127, 1970.

[Royce 1998] Walker Royce: „Software Project Management – A Unified Framework“. Addison-Wesley, Reading, MA, 1998.

[Ruhe 2005] Günther Ruhe: „Software Release Planning“. In: „Handbook Software Engineering and Knowledge Management.“ Vol.3, pp 365-394, (S.K. Chang, Ed.), World Scientific 2005.

[Saliu & Ruhe 2005a] O. Saliu, G. Ruhe: “Software Release Planning for Evolving Systems”. In: „Innovations in Systems and Software Engineering - A NASA Journal“. Vol. 1, Issue 2, pp.189-204, 2005.

[Saliu & Ruhe 2005b] O. Saliu, G. Ruhe: “Supporting software release planning decisions for evolving systems”. In: “Proceedings of 29th IEEE/NASA software engineering workshop“, Greenbelt, MD, USA, 6–7 April, 2005.

[Schwaber & Beedle 2002] Ken Schwaber, Martin Beedle: „Agile Software Development with SCRUM“. Prentice Hall, Upper Saddle River, NJ, 2002.

[Shubin 2002] Sean Shubin: *„Test First Guidlines“,* 2002. Verfügbar unter: http://www.xprogramming.com/xpmag/testFirstGuidelines.htm [25.01.2002]

[Singer et al 2008] Janice Singer, Susan E. Sim, Timothy C. Lethbridge: "Software Engineering Data Collection for Field Studies". In: Forrest Shull, Janice Singer, Dag I. K. Sjoberg (Hrsg.): „Guide to Advanced Empirical Software Engineering", Springer, London 2008.

[Smart 2006] John Ferguson Smart: „ Which open source CI tool is best suited for your application's environment? - A comparative review of four open source continuous integration tools". In: JavaWorld, 2006. Verfügbar unter: http://www.javaworld.com/javaworld/jw-11-2006/jw-1101-ci.html?page=2 [1.10.2008]

[Sonatype 2008] Sonatype Company: „Maven: The Definitive Guide". O'Reilly, 2008.

[Sowizral et al. 1998] Henry Sowizral, Kevin Rushforth, Michael Deering: „The Java 3D API Specification", Addison-Wesley, New York, 1998.

[Tinderbox 2009] Tinderbox Homepage: „Tinderbox Firefox Tree". http://tinderbox.mozilla.org/showbuilds.cgi?tree=Firefox [13.03.2009]

[Vähäniitty et al. 2002] J. Vähäniitty, C. Lassenius, K. Rautiaine: "An Approach to Product Roadmapping in Small Software Product Businesses". In: „Conference Notes of Quality Connection - 7th European Conference on Software Quality (ECSQ2002)". Center for Excellence Finland, Helsinki, Finland, 2002.

[Victor & Günther 2005] Frank Victor, Holger Günther: „Optimiertes IT-Management mit ITIL - So steigern sie die Leistung ihrer IT-organisation-Einführung, Vorgehen, Beispiele". 2. Auflage, Vieweg Verlag, Wiesbaden, 2005.

[VersionOne 2008] Version One Homepage. Verfügbar unter: http://www.versionone.com/ [16.05.2008]

[Walkenbach 2007] John Walkenbach „Microsoft Office Excel 2007". Wiley Publishing Inc., Indianapolis (IN), 2007.

[Wolf & Dutoit 2004] Timo Wolf & Allen Dutoit: „A Rationale-based Analysis Tool". In: „Proceedings of 13th International Conference on Intelligent & Adaptive Systems and Software Engineering", July 1-3, Nice, France, 2004.

[Wolf 2007] Timo Wolf: „Rationale-based Unified Software-Engineering Model", Dissertation, TU-München, 2007.

[Yin 2002] Robert K. Yin: „Case Study Research – Design and Methods“. Applied Social Research Methods Series Volume 5. 3. Auflage, Sage Publications, Thousand Oaks, CA, 2002.

[Zimmermann & Stache 2001] Werner Zimmermann, Ulrich Stache: „Operations Research - Quantitative Methoden zur Entscheidungsvorbereitung “, 10. Auflage, Oldenbourg Wissenschaftsverlag, 2001