Teil I

Einleitung und Grundlagen

I.1 Einleitung

> More than a discipline or a body of knowledge, engineering is a verb, an action word, a way of approaching a problem.
>
> Scott Whitmire

Inhalt

Das Streben nach hoher Qualität ist in allen Lebensbereichen ein essentieller Punkt. Vor allem in der Wirtschaft ist es für die Wettbwerbsfähigkeit und letztendlich für gewinnbringende Einnahmen ein nicht zu vernachlässigender Faktor. In Ingenieursdisziplinen, wie dem Maschinenbau und der Elektrotechnik, kann man Qualität meist anhand materieller Eigenschaften messen. Im Bereich der IT sieht dieses bedauerlicherweise anders aus. Obwohl das Streben nach einer hohen Qualität hier ebenso existiert, ist Software ein immaterielles Gut, welches nicht manuell „gefertigt“ sondern „nur [virtuell] entwickelt“ wird [LL07] und somit nicht so simpel vermessbar ist, wie beispielsweise eine Maschine.

Aus diesem Grund sind andere Maßstäbe anzusetzen. Es müssen Kennzahlen erschaffen werden, mit denen ein Software-Produkt, wie auch das gesamte Entwicklungsprojekt, objektiv evaluiert und beurteilt werden kann. Anhand dieser Kennzahlen sollen die Projektverantwortlichen die Möglichkeit erhalten, das Projekt zu steuern und somit eine hohe Qualität anzustreben.

Im Laufe der Zeit wurden viele Metriken entwickelt, die Informationen zur Produkt- und Prozessqualität liefern (siehe beispielsweise „Software Metrics: A Rigourous Approach“ von Norman E. Fenton [Fen91]). Zusätzlich existieren diverse Normen, Richtlinien und Vorgänge, welche sich mit der Herleitung dieser Metriken durch Software-Messprozesse und Software-Qualitätsmodelle beschäftigen (z.B. *ISO/IEC15939:2007* [Ins09], *ISO/IEC 9126-1:2001(E)* [ISO01], *GQM-Ansatz* [OB92]).

Die Herausforderung in der Anwendung von Projekt- und Produktmetriken besteht darin, ein auf das Projekt bzw. Produkt passendes Qualitätsmodell zu erstellen, daraus die entsprechenden Informationsbedürfnisse abzuleiten und letztendlich die benötigten Daten aus dem Projekt zu sammeln. Diese Datensammlung alleine reicht jedoch nicht

aus um effizient die Qualität des Projekts zu bestimmen und es diesbezüglich zu steuern. Folglich müssen die erhobenen Daten auf eine simple und verständliche Weise dargestellt werden, so dass der Projektstatus ohne großen Aufwand auf einen Blick erkennbar ist.

I.1.1 Ziele der Arbeit

An dieser Stelle setzt die vorliegende Arbeit an. Das Lehr- und Forschungsgebiet Softwarekonstruktion der RWTH Aachen hat ein System entwickelt, welches Projektdaten, die mit unterschiedlichen Werkzeugen erhoben werden können, zusammenfasst, evaluiert und visuell darstellt. Die visuelle Darstellung geschieht durch ein Dashboard, welches neben der Darstellung der Kennzahlen auch Konfigurations- und Interpretationshilfen bietet, um es an jedes Projekt und jede Projektrolle individuell anzupassen.

Ziel dieser Arbeit ist die Erstellung und Evaluation eines Konzepts zur fachliche Integration dieses Systems in bestehende Organisationsstrukturen. So soll im Laufe dieser Arbeit festgestellt werden, welche Vorgehensweisen bei der Integration von Nöten sind und wie sowohl das (Software-)Projekt an das System, als auch das System an das Projekt und den vorhandenen Prozess angepasst werden muss.

Des weiteren wird im Laufe dieser Arbeit eine Vorlage mit typischen Kennzahlen der Projektleiter entwickelt, welche die Benutzung der genannten Dashboard-Komponente vereinfachen soll. Wie zuvor erwähnt existiert eine Vielzahl an Metriken, welche für einen Neuling im Bereich von Software-Messungen unüberschaubar wirkt. Diese Vorlagen sollen eine Hilfestellung zur Erstellung und Anpassung eines Dashboards an die Bedürfnisse der Projektrollen darstellen, ohne großen Konfigurationsaufwand für die Benutzer zu verursachen.

I.1.2 Aufbau der Arbeit

Bevor der Aufbau der Arbeit beschrieben wird, soll an dieser angemerkt werden, dass die männliche Bezeichnung von Projektrollen oder ähnlichem keinen Rückschluss auf das Geschlecht zulassen. Der Einfachheit halber wird in dieser Ausarbeitung nur der männliche Bezeichner gewählt, die Intention der Aussage gilt jedoch dem Menschen hinter dieser Rolle, nicht dem Geschlecht.

I.1.2.1 Teil 1: Grundlagen

Im ersten Teil der Arbeit werden die wichtigsten Grundlagen für das Verständnis vermittelt. So wird als Ausgangspunkt die Basis des Software-Projektmanagements erläutert (Kapitel I.2) und weiterhin ein Fundament an Informationen bzgl. Software-Qualitätssicherung und Metriken erstellt (Kapitel I.3). Zu guter Letzt wird das oben erwähnte System, die *EMI*, mit ihrer Dashboard-Komponente *SCREEN* genauer erläutert.

I.1.2.2 Teil 2: Konzept zur fachlichen Integration von Messsystemen

Der zweite Teil der Arbeit stellt ein Konzept der Integration von Kennzahl-Messsystemen (wie beispielsweise dem EMI-System) in bestehende Organisationsstrukturen vor. Neben den Anforderungen an das Konzept, den Messkunden und das Messsystem (Kapitel II.2), werden die einzelnen konzeptionellen Schritte des Vorgehens detailliert erläutert (Kapitel II.3 bis Kapitel II.7). Zusätzlich werden verwandte Arbeiten aus diesem Bereich, wie auch die Abgrenzung zu dieser Diplomarbeit vorgestellt (Kapitel II.8).

I.1.2.3 Teil 3-5: Feldstudien

In den folgenden drei Teilen werden Feldstudien aus Forschung und Wirtschaft präsentiert. Aufgrund dieser Feldstudien konnten Metrik-Vorlagen für Projektleiter erstellt und der konzeptionelle Teil der Integration evaluiert werden. Teilnehmend waren sowohl das Forschungsprojekt der Software-Entwicklungsplattform *SSELab* (Teil III), als auch die beiden Unternehmen *Generali Deutschland Informatik Services GmbH (GDIS)* (Teil IV) und *i-nex GmbH* (Teil V).

I.1.2.4 Teil 6: Evaluation, Zusammenfassung und Ausblick

Im letzten Teil der Arbeit wird das erstellte Konzept, wie auch die fachliche Integration der Feldstudien evaluiert (Kapitel VI.1). Weiterhin werden die wichtigsten Punkte dieser Arbeit zusammengefasst (Kapitel VI.2) und ein Ausblick auf weitere Möglichkeiten und Arbeiten gegeben (Kapitel VI.3).

I.2 Software-Projektmanagement

> It takes less time to do a thing right than explain why you did it wrong.
>
> Henry Wadsworth Longfellow

Inhalt

Wie eingangs erwähnt, spielt Qualität in der Softwareentwicklung eine große Rolle. Bevor der Qualitätsbegriff im kommenden Kapitel I.3 genauer definiert wird, erläutert dieses Kapitel die Begrifflichkeit des *Software-Projektmanagements* mit seiner Definition, seinen Aufgaben und Prozessen.

I.2.1 Projekt

Vor der Thematisierung des (Software-)Projektmanagements, soll die Terminologie des *Projekts* geklärt werden. Im umgangssprachlichen Wortschatz wird dieser Begriff oftmals für eine Reihe von Tätigkeiten verwendet, die sowohl Planung, als auch größeren Aufwand beinhalten. So werden im Alltag beispielsweise der Wohnungsputz, die Facharbeit des Schulkindes oder die Planung des Urlaubs als Projekt bezeichnet. Die große Gemeinsamkeit dieser Tätigkeiten ist, dass sie ein bestimmtes Ziel verfolgen und von gewisser Dauer sind.

Im Umfang dieser Arbeit werden Projekten vor allem im Rahmen der Softwareentwicklung betrachtet. So definiert der IEEE-Standard 1490-2003: *A Guide to the Project Management Body of Knowledge* [Ins04] ein Projekt als

> „A temporary endeavor undertaken to create a unique product or service."

Folglich liegt in dieser Definition der Fokus auf der nicht langanhaltenden Bemühung ein Produkt bzw. einen Service bereit zu stellen. Bergmann und Garrecht zitieren in ihrem Buch *Organisation und Projektmanagement* [BG08] hingegen die Definition der DIN-Norm 69901 [Deu09], wonach ein Projekt beschrieben wird als „ein Vorhaben, das

im Wesentlichen durch **Einmaligkeit der Bedingungen** in ihrer Gesamtheit gekennzeichnet ist, z.B.

- spezielle, einmalige **Zielvorgabe**,
- zeitliche, finanzielle, personelle oder andere **Begrenzungen**,
- **Abgrenzung** gegenüber anderen Vorhaben,
- projektspezifische **Organisation**“

Auch Patzak und Rattay [PR98] bauen auf dieser Definition auf und schreiben Projekten typische Merkmale wie *Neuartigkeit*, *Zielorientierung*, *Dynamik und Komplexität*, *Interdisziplinarität* und *Bedeutsamkeit* zu. Ludewig und Lichter [LL07] definieren Projekte weitergehend als eine zweckgebundene Anreihung von Aktionen, dessen Laufzeit beschränkt ist und welches „Menschen, Resultate (*Zwischen- und Endprodukte*) und Hilfsmittel (*Ressourcen*)“ miteinander verbindet.

Aus diesen unterschiedlichen Sichtweisen, wird für diese Arbeit folgende Definition des Begriffs angenommen:

Definition I.2.1 (Projekt) *Ein (Software-)Projekt ist eine Kette von Aktivitäten, die als Ziel die Erstellung eines (komplexen) Produktes hat. Seine Entwicklungslaufzeit, wie auch die vorhandenen Ressourcen (Menschen, Hilfsmittel, Resultate) sind begrenzt und werden in einer projektspezifischen Umgebung interdisziplinär verwendet.*

An der steuernden Spitze eines Projekts steht eine leitende Person, welche das Projekt organisiert, steuert und mithilfe weiterer Mitarbeiter durchführt. Im Rahmen dieser Arbeit wird der Begriff des Projektleiters synonym mit dem des Projektmanagers verwendet und deutet auf keinen fachlichen Unterschied hin.

I.2.1.1 Projektarten

Die alleinige Begriffsdefinition ist für das Verständnis von Projekten nicht ausreichend. Projekte können eine Vielzahl von unterschiedlichen Formen annehmen, so dass auch die Betrachtung des Projekttyps von Interesse ist. Die folgende Auflistung gibt einen groben Überblick über die Mannigfaltigkeit von Projektarten (vgl. [PR98], [KHL+11]):

Forschungsprojekte sind typische Projekte im universitären Bereich und dienen der Erforschung eines neuen Themengebietes. Oftmals spielt die Qualität des Projektziels keine immense Rolle, da diese Projekte der Unterstützung von wissenschaftlichen Thesen oder Modellen dienen und nicht für den wirtschaftlichen Markt gedacht sind.

Produktentwicklungsprojekte führen zu der Entwicklung eines materiellen Produkts, wie beispielsweise einem Auto oder einer Maschine. In diesen Projekten kann die typische ingenieurmäßige Herangehensweise gewählt werden, die ein bestimmtes Produktionsvorgehen vorgibt.

IT-Projekte sind Produktentwicklungsprojekte, die ein immaterielles Software-Produkt als Ergebnis liefern. Der Bereich des *Software Engineerings* existiert nicht so lang wie die Ingenieursdisziplinen, so dass ein klares strukturiertes Vorgehen in der Entwicklung nur bedingt vorhanden ist.

Infrastrukturprojekte dienen der Verbesserung bzw. dem Ausbau der vorhandenen Infrastruktur. So können sie sowohl im öffentlichen Bereich den Ausbau der Verkehrsinfrastruktur umfassen, als auch innerhalb von Organisationen die Verbesserung der Unternehmensinfrastruktur (beispielsweise Erneuerung der Hardware).

Organisationsprojekte führen die Verbesserung des Organisationsablauf beziehungsweise -prozesses mit sich. Sie sind immaterielle Projekte, welche innerhalb einer Organisation entwickelt werden.

Investitionsprojekte beinhalten hohen Investitionsaufwand in die Entwicklung neuer Produkte, Technologien oder Infrastrukturen. Aus diesem Grund liegt der Schwerpunkt dieser Projekte bei der Planung und Budgetierung, wie auch der Einhaltung des Terminplans.

Im Zuge dieser Ausarbeitung werden hauptsächlich Forschungs- und IT-Projekte betrachtet, dessen Fokus auf der Softwareentwicklung liegt. Auch andere Projekte könnten Kennzahlen-Messsysteme, wie das *EMI*-System, verwenden, jedoch liegt auf diesen in der vorliegenden Arbeit nicht der Schwerpunkt.

I.2.2 Software-Projektmanagement

Für die Durchführung eines komplexen (Software-)Projektes, ist eine dezidierte Projektplanung von Nöten. Neben der Ablaufplanung, fallen auch Entscheidungen bzgl. der Projektorganisation, -kontrolle, wie auch der Ressourcenverteilung an, welche sich mit dem Begriff des *Projektmanagements* zusammenfassen lassen [PR98].

Der IEEE-Guide bzgl. des *Projekt Management Book of Knowledge* (*kurz: PMBoK*) [Ins04] definiert Projektmanagement als

> „the application of knowledge, skills, tools, and techniques to project activities to meet project requirements.“

Neben der Anwendung von Wissen, Techniken und Werkzeugen, liefert das *PMBoK* weitere Wissensgebiete, die das Projektmanagement umfassen. Folglich werden diese kurz erläutert.

Integrationsmanagement dient der Koordination diverser Projektelemente, wie der Projektplanung, -ausführung und des integrierten Änderungsmanagements.

Inhalts- und Umfangsmanagement umfasst alle Prozesse, die zur Durchführung der benötigten Arbeiten für einen Projekterfolg führen. Dazugehörige Prozesse sind unter anderem Umfangsplanung, -definition und -änderungskontrolle.

Zeitmanagement stellt die zeitliche Fertigstellung des Projekts sicher und beinhaltet Aktivitätenplanungen (inklusive Definition, Sequenzierung und Zeitschätzungen), wie auch die Erstellung und Kontrolle des Zeitplans.

Kostenmanagement dient der Sicherstellung des Kostenrahmens, in dem sich das Projekt befindet. Hierbei spielt vor allem die Ressourcenplanung, Kostenschätzung und -budgetierung, wie auch die Kostenkontrolle eine große Rolle.

Qualitätsmanagement soll sicherstellen, dass die entsprechenden Projektbedürfnisse (wie beispielsweise Anforderungen) ausreichend erfüllt werden. Hierfür wird ein Qualitätsplan erstellt, welcher kontrolliert wird und bei dem die Durchführung der einzelnen Qualitätsprüfungen sicherzustellen sind.

Personalmanagement ist ein Wissensgebiet, welches sich mit der organisatorischen Planung, der Projektmitarbeiter-Akquise und der Team-Entwicklung beschäftigt. Der essentielle Punkt des Personalmanagements ist die Sicherstellung einer möglichst effektiven Nutzung der Mitarbeiter innerhalb des Projekts.

Kommunikationsmanagement beschreibt die Kommunikation von Wissen und Informationen innerhalb des Projekts. Hierbei werden unter anderem Prozesse zur Kommunikationsplanung, Informationsverteilung und Fortschrittsberichterstattung bereitgestellt beziehungsweise entwickelt.

Risikomangement ist der systematische Prozess der Identifikation, Analyse und Reaktion auf Projektrisiken. Er beinhaltet Prozesse zur Planung und Identifikation von Risiken, so wie Prozesse zur Reaktion, Steuerung und Kontrolle dieser.

Beschaffungsmanagement dient der Beschaffung von Gütern und Dienstleistungen, der Erreichung des Projektumfangs und dem Lieferantenmanagement. Es beinhaltet zusätzlich die Angebotseinholung, wie auch alle Prozesse im Bereich der Vertragsabschlüsse.

Das in dieser Arbeit betrachtete Kennzahlen-System *EMI* dient der Kontrolle und Steuerung dieser diversen Projektmanagement-Wissensgebiete. Anhand diesem, wie auch dem zugehörigen Kennzahlen-Dashboard *SCREEN* (siehe auch Kapitel I.4), soll die Kontrolle über Aspekte wie beispielsweise Projektkosten, Termineinhaltung, Produktqualität bewahrt und vereinfacht werden.

I.2.3 Software-Entwicklungsprozess

„Jedem Projekt liegt unvermeidlich ein Prozess zu Grunde“ [LL07], der das geplante Entwicklungsvorgehen beschreibt. Ein Software-Entwicklungsprozess beinhaltet somit

eine Reihe von Aktivitäten, die zu der Erstellung eines Software-Produktes führen [Som11]. In der Literatur wird zwischen reinen Vorgehensmodellen (die nur die einzelnen Entwicklungsschritte beschreiben) und Prozessmodellen (die neben dem Vorgehen auch Informationen zur Organisation, Qualitätssicherung, Konfiguration und dem Projektmanagement liefern) unterschieden [LL07].

Die unterschiedlichen Vorgehensmodelle, wie beispielsweise der Software-Lebenszyklus oder das Wasserfall-Modell, sind im Bereich des Software Engineerings wohlbekannt und werden mit dem Verweis auf entsprechende Standardliteratur, wie [LL07], [Som11] oder [Pre01], in dieser Arbeit nicht weiter erläutert.

I.2.3.1 Prozessmodelle

Um die unterschiedlichen Bedürfnisse, wie auch diverse Optionen der Projektsteuerung in Software-Projekten zu erfassen, ist das Verständnis der verschiedenen Möglichkeiten, welche durch Prozessmodelle gegeben sind, essenziell. Aus diesem Grund wird innerhalb dieses Abschnitts eine kurze Zusammenfassung der wichtigsten, in der Wirtschaft verwendeten Prozessmodelle präsentiert. Dieser Überblick dient nicht der vollständigen Prozessbeschreibung, sondern ist nur eine schemenhafte Übersicht der Unterschiede innerhalb der beschriebenen Modelle. Für weiterführende Informationen wird an dieser Stelle auf entsprechende Literatur wie beispielsweise [LL07], [Som11] oder [Pre01] verwiesen.

I.2.3.1.1 Phasenmodell

Das *Phasenmodell* erweitert den typischen Software-Lebenszyklus um Projektphasen, die einen festen Zeitrahmen umfassen. Ihr Ende wird anhand eines Meilensteins („markanter Punkt mit bekannter Position“ [LL07]) markiert, welcher den Beginn einer neuen Phase einläutet. Die Dauer der einzelnen Phasen wird bei der Projektplanung geschätzt und gegebenenfalls korrigiert. Alternativ kann bei der Feststellung einer Fehlschätzung, wie beispielsweise einer zu geringe Restzeit für übrige Aufgaben, die Arbeitsleistung erhöht werden, um den Meilenstein dennoch einhalten zu können. Innerhalb des Phasenmodells kann der Projektfortschritt durch die Erreichung eines Meilensteins ermittelt werden [LL07].

I.2.3.1.2 V-Modell

Das *V-Modell* ist der deutsche Entwicklungsstandard für IT-Projekte, der von öffentlichen Auftraggebern (beispielsweise Bund und Länder) vorgeschrieben wird. Es ist ein aktivitätsorientiertes Phasenmodell, welches die Vorgehensschritte des Wasserfallmodells um „projektbegleitende Tätigkeiten“ [LL07], wie dem Projekt-, Qualitäts- und Konfigurationsmanagement, als auch dem Problem- und Änderungsmanagement, erweitert. Es trägt seinen Namen nicht nur aufgrund der Tatsache, dass es ein *Vorgehens*modell ist, sondern auch aufgrund des V-förmigen Ablauf innerhalb der Systementwicklung (siehe

Abbildung I.2.1).

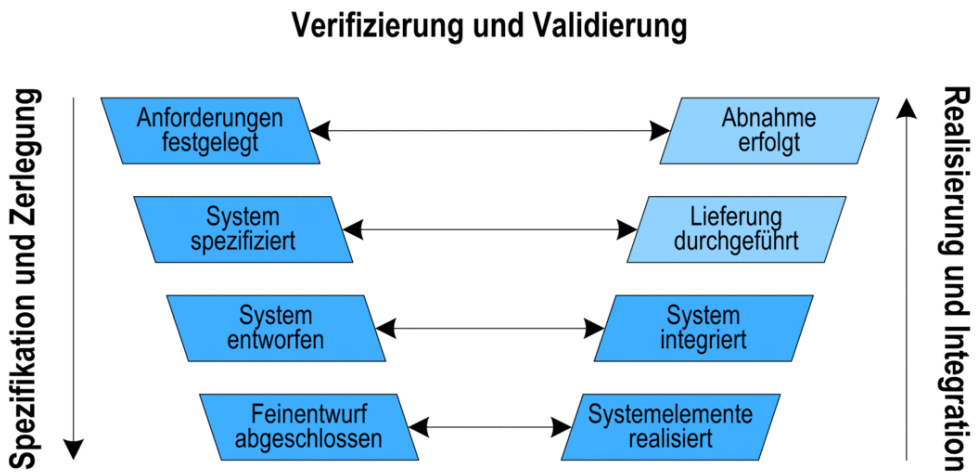

Abbildung I.2.1: Struktur der Systementwicklung nach dem V-Modell (Abb. aus *Teil 1: Grundlagen des V-Modells*, Kap. 5.2 [VMo12])

2004 wurde das V-Modell um Aspekte der Anpassbarkeit zu dem *V-Modell XT* (mit XT = *extreme Tailoring*) erweitert. Das V-Modell XT unterstützt sowohl die inkrementelle und komponentenbasierte, als auch die agile Entwicklung und lässt sich somit für diverse Projektarten verwenden.

Innerhalb dieses Modells haben sich die Bezeichnungen zum Teil geändert, so dass Meilensteine beispielsweise als Entscheidungspunkte definiert wurden, von denen aus der Beginn des nächsten Projektabschnittes entschieden wird. Weitere Begrifflichkeiten des Modells sind *Aktivitäten*, *Produkte* und *Rollen*. So führen Aktivitäten zur Entwicklung von Produkten, die Zwischenergebnissen und Endergebnissen entsprechen. Jedem Produkt ist mindestens eine Rolle (Verantwortlicher der Aufgabe), wie etwa der Qualitätsbeauftragte, zugewiesen. Das V-Modell XT beinhaltet mehr als 90 Aktivitäten, über 100 Produkte und 31 Rollen, so dass es sehr variabel eingesetzt werden kann. [LL07]

I.2.3.1.3 Rational Unified Process

Der Rational Unified Process (RUP) wurde 1998 von Booch, Jacobson, Kruchten und Rumbaugh entwickelt und ist ein hybrides Prozessmodell, welches sowohl die inkrementelle, als auch die iterative Entwicklung in Phasen vereint [Som11], [LL07]. Dies hat den Vorteil, dass Projektplanung und Projektmanagement u.a. durch frühzeitige Erkennung und Beseitigung von Risiken vereinfacht wird [LL07].

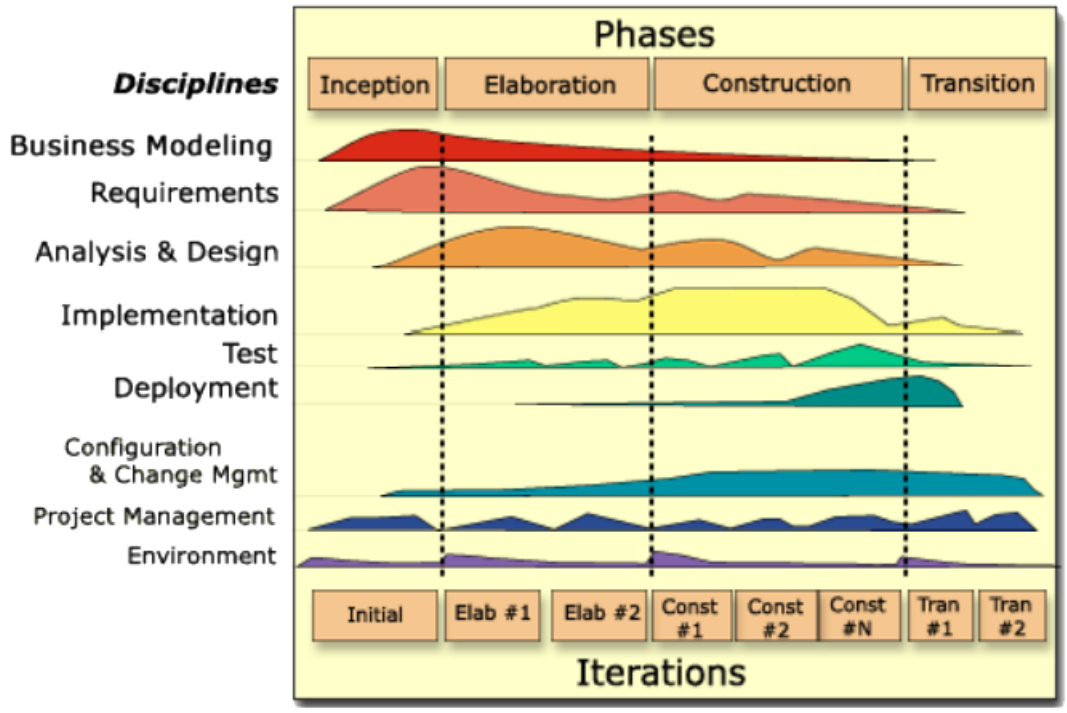

Abbildung I.2.2: Verteilung des Disziplinaufwandes bzgl. der RUP-Phasen (Abb. aus *Rational Unified Process* Version 2003.06.15 [IBM03])

Der RUP beinhaltet *vier Entwicklungsphasen* (bzw. Iterationsschritte), in denen Aktivitäten aus *neun Disziplinen* inkrementell ausgeführt und daraus folgende Artefakte entwickelt werden. Die Aufwandsverteilung der Disziplinen innerhalb der einzelnen Phasen wird in Abbildung I.2.2 verdeutlicht. So wird auch ersichtlich, dass innerhalb einer Iteration eine Weiterentwicklung sämtlicher Disziplinen stattfindet, sich jedoch der Entwicklungsschwerpunkt jeder Phase innerhalb dieser verschiebt. Folglich werden die vier Iterationsschritte des RUP genauer betrachtet.

Inception: Die Phase der *Konzeption*, in der die ersten Anforderungen und Architekturpläne erstellt werden. Weiterhin dient sie der Risiko- und Durchführbarkeitsanalyse.

Elaboration: Die *Entwurfsphase* der kompletten Systemarchitektur und der Vervollständigung der Anforderungen. In diesem Schritt werden zusätzlich Prototypen erstellt, um Architekturentscheidungen und Funktionalitäten abzubilden.

Construction: Die Phase der tatsächlichen *Produktrealisierung*. In diesem Iterationsschritt wird das Produkt zusätzlich getestet und weitere Dokumente, wie beispielsweise das Benutzerhandbuch, gefertigt.

Transition: Die *Einführungsphase*, in der das Produkt getestet, verbessert und auf die Integration und Auslieferung an den Kunden vorbereitet wird.

Innerhalb dieser vier Phasen werden die neun unterschiedlichen Disziplinen (auch *Workflows* genannt), jeweils inkrementell weiterentwickelt. Der Aufwand der Entwicklungstätigkeit pro Disziplin variiert dadurch je nach Bedarf. Im Folgenden werden die Workflows mit ihren Eingenschaften genauer erläutert. Hierbei wird zwischen „Ingenieurstätigkeiten“ (den ersten sechs genannten) und unterstützenden Disziplinen (den letzten dreien) unterschieden.

Business Modelling: Hierbei wird ein Workflow, dessen Resultat eine Art Business-Plan ist, abgearbeitet. Zusätzlich werden Business Use Cases erstellt, die dazu dienen, allen Stakeholder des Projekts das gleiche Verständnis der Prozesse, Anforderungen und des Systems zu vermitteln.

Requirements: In dieser Disziplin werden Anforderungen an das zu entwickelnde System erarbeitet, welche die von Kunden und Entwicklern akzeptiere Ausgangsbasis für den weiteren Projektverlauf darstellt. Innerhalb des RUP werden sie als Use Case dargestellt.

Analysis and Design: In dieser Disziplin das Entwurfsmodell und die System-Architektur ausgearbeitet. Diese Arbeitsergebnisse dienen als Startpunkt der tatsächlichen Implementation.

Implementation: Sie umfasst die Erstellung des geplanten Systems. Innerhalb dieser Disziplin kann in einer frühen Phase ein Prototyp zur Funktions- und Risikoabsicherung erstellt werden, wohingegen in einer späteren Phase das gesamte System anhand der zuvor entworfenen Architektur entwickelt wird.

Testing: Es dient innerhalb des RUPs zur Sicherstellung der Interaktionen zwischen den Objekten, der Integration der entwickelten Komponenten, der richtige Implementierung der Anforderungen, so wie der Fehlerhandhabung innerhalb der Entwicklung.

Deployment: Innerhalb dieser Disziplin wird das System gepackt, veröffentlicht und an den Kunden übergeben. Ebenso ist die Installation des Systems ein wichtiger Teil des Deployment-Workflows.

Configuration and Change Management : Zur Verwaltung entstandener Arbeitsergebnisse (ebenfalls *Artefakte* genannt), wie auch der Dokumentation nachträglicher Änderungen oder hinzukommender Ergebnissen wird dieser Workflow genutzt. Er

ist einer der drei unterstützenden RUP-Disziplinen und ein wichtiger Teil der Kontrolle und Überwachung der Arbeitsergebnisse, wie auch des Nachvollziehens getroffener Entscheidungen.

Project Management: Diese Disziplin beinhaltet die essentiellen Aufgaben der Planung und Kontrolle des gesamten Projekts. So fallen in diesem Workflow beispielsweise die Zusammenstellung der Projektteams, die Planung, Überwachung und Vermeidung der vorhanden Risiken, wie auch die allgemeine Projektkontrolle bezüglich Aufwand, Kosten und Qualität der Projektdurchführung an.

Environment: Diese Disziplin dient der Sicherstellung der Arbeitsumgebung. Ziel dieses Workflows ist sowohl die Bereitstellung entsprechender Arbeitswerkzeuge (Hardware, Software, Regelwerke etc.), als auch der entsprechenden Entwicklungsprozesse.

Der Rational Unified Process ist somit ein Prozess-Framework, welches das Vorgehen innerhalb eines Entwicklungsprojekts vorgibt. Er liefert eine Menge an Arbeitsergebnissen, die zu einer guten Dokumentation des Projekts führen und definiert diverse Projektrollen inklusive ihrer Projektaufgaben. IBM liefert mit dem *IBM Rational Unified Process* (siehe [Kru04]) ein Werkzeug, welches direkt nutzbar ist, jedoch auch individuell an die Organisation angepasst werden kann. Für weitere Informationen existiert eine von Rational veröffentliche Ausarbeitung mit *Best Practices* für Softwareentwicklungsteams [Rat03], wie auch eine ausführliche Anleitung des RUP von Philippe Kruchten [Kru04].

I.2.3.1.4 Agile Prozesse

Aufgrund der zum Teil gegebenen Prozessstarrheit bzw. des Bürokratieaufwandes einiger Prozessmodelle, entstand Ende der Neunzigerjahre eine Gegenbewegung von „Anti-Prozessen“ [LL07]. Während Prozess-Frameworks wie das V-Modell oder der Rational Unified Process durch großen Dokumentationsaufwand, viele Artefakte und Auflagen das Vorgehen künstlich um viele Arbeitsschritte vergrößern und Inflexibilität fördern, ist das Ziel der agilen Prozesse ein Augenmerk auf das tatsächliche Ziel - dem zu entwickelnden System - und die beteiligten Personen zu richten.

2001 wurde von den Gründern der agilen Bewegung ein Agiles Manifest (*Agile Manifesto*) verfasst, welches vier Grundregeln umfasst [24]:

> „Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:
>
> **Individuen und Interaktionen** mehr als Prozesse und Werkzeuge
>
> **Funktionierende Software** mehr als umfassende Dokumentation
>
> **Zusammenarbeit mit dem Kunden** mehr als Vertragsverhandlung
>
> **Reagieren auf Veränderung** mehr als das Befolgen eines Plans

> Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.“

Hinter diesem Manifest stehen weitere zwölf Prinzipien, welche die Grundzüge der agilen Entwicklung genauer definieren. Die Begründer der agilen Bewegung betonen die flexible, kundenorientierte Entwicklung, die zu schnellen, einfachen Ergebnissen führt und welche durch selbst-organisierte Teams erreicht werden kann. Die genauen Prinzipien sind im Anhang A bzw. in [25] zu finden.

Aufbauend auf das Agile Manifesto, wurden agile Prozessmodelle bzw. Rahmenwerke entwickelt, die eine unterschiedlichen Auslegungsgrad beinhalten. Neben der alleinigen Betrachtung dieser Modelle, existiert zusätzlich die Möglichkeit innerhalb eines Projekts, mittels der Mischung verschiedener agilen Techniken, einen eigenen, angepassten agilen Prozess zu erstellen. Zwei der bekanntesten agilen Prozessmodelle sind *Extreme Programming* und *Scrum*, welche an dieser Stelle nicht genauer erläutert, sondern innerhalb vorhandener Literatur wie beispielsweise [WB11], [Wol12] oder [Pic08] genauer definiert werden.

I.2.3.2 Unterstützung durch CASE-Tools

Nahezu jedes Software-Entwicklungsprojekt arbeitet heutzutage mit diversen *Computer-Aided Software Engineering (CASE)* Tools, um bestimmte Teile des Entwicklungsprozesses zu unterstützen. Die Nutzungsintensivität dieser Werkzeuge hängt von dem verwendeten Entwicklungsprozess, der Organisationsmentalität und dem Kenntnisstand zur Nutzung ab, jedoch trifft die allgemeingültige Aussage zu, dass innerhalb jedes Software-Entwicklungsprojektes mindestens ein unterstützendes Werkzeug (beispielsweise ein Entwicklungsframework) verwendet wird.

Eine große Bandbreite an unterschiedlichen CASE-Tools ist für die Softwareentwicklung vorhanden. So existieren Werkzeuge, welche die reine Produktentwicklung unterstützen und andere die wiederum den Prozess und die Planung vereinfachen. Folglich wird eine für die Ausarbeitung relevante Auswahl der von Pressman [Pre01] dargestellten Kategorien erörtert.

Tools zur Prozessmodellierung und dem Prozessmanagement dienen der Anpassung und Weiterentwicklung des verwendeten Prozesses, wie auch dem Überblick der Prozessschritte oder Artefakte. Der *IBM Rational Unified Process*® [23] ist beispielsweise ein solches Werkzeug. Er bietet dem Anwender sowohl die ausführliche Beschreibung des Prozesses und der Artefakte, als auch Hilfestellungen bezüglich dessen Anwendung.

Tools zur Unterstützung des Projektmanagements und der Projektplanung sind hilfreich für die Erstellung und Überwachung des Projektplans mit dem beinhalteten Zeit- und Kostenbudget. Beispiele für solche Tools bieten *Microsoft Project* [30] oder *JIRA* von *Atlassian* [7].

Tools zur Produktentwicklung wie Entwicklungsframeworks oder Editoren werden in jedem Software-Entwicklungsprozess genutzt. So kann ein Entwickler zur Erstellung des Quellcodes entweder einen schlichten Editor benutzen oder gar ein Framework, wie beispielsweise *Eclipse* [11], welches zusätzliche Funktionen und Add-Ons, wie beispielsweise ein integriertes Code-Formatierungssystem oder gar Integrationsmöglichkeiten anderer CASE-Tools, liefert.

Tools zur Qualitätssicherung können sowohl Testumgebungen, Code Checker (beispielsweise zur Findung von Verletzungen der Kodierungsrichtlinien), Continuous Integration Systeme, oder auch Metrik- und Messtools beinhalten. So ist *JUnit* [3] ein Test-Framework für Java-Entwicklungen, *Jenkins* [2] bzw. *Hudson* [12] liefern einen Continuous-Integration-Server und Ticket-Systeme bzw. Bug-Tracker wie *Trac* [13] dienen der Kontrolle von Fehlern und Aufgaben. Das in dieser Arbeit verwendete System *EMI* (siehe auch Kapitel I.4) ist ein Qualitätssicherungswerkzeug zur Erhebung und Visualisierung von Projektkennzahlen.

Weitere Tools zur Unterstützung des Software-Entwicklungsprozesses, wie beispielsweise Dokumentationstools, Programmiertools, Editoren, Analysetools oder Tools zur Unterstützung des Architekturaufbaus existieren ebenfalls, werden innerhalb dieser Arbeit jedoch nicht tiefgründiger betrachtet.

Diese Werkzeuge unterstützen nicht nur den Software-Entwicklungsprozess, sondern liefern auch eine Menge an Daten, die zur Projektanalyse genutzt werden können. So liefern Ticketsysteme Informationen bezüglich der Projekt-Fehler oder der erstellten Tickets, Hilfsmittel für das Projektmanagement wiederum Aufwands- und Projektdaten. All diese Informationen können durch ein Kennzahlen-Messystem wie *EMI* gesammelt, ausgewertet und visualisiert werden, so dass sie zur Kontrolle und Steuerung des Projekts dienen.

I.3 Software-und Prozess-Metriken

> Software metrics let you know
> when to laugh and when to cry.
>
> Tom Gilb

Inhalt

Der erste Teil dieses Kapitels stellt den *Qualitätsbegriff*, wie auch die Auffassung von *Modellen zur Qualitätssicherung* in Projekten vor. Im zweiten Teil wird die Bedeutung der *Metriken* definiert, Beispiel-Metriken genauer erläutert und in Kontext dieer Arbeit gesetzt.

I.3.1 Qualitätsbegriffe

Sucht man eine allgemeine Definition des Begriffs *Qualität*, so beschreibt die Online-Begriffsdefinition des Dudens [10] diese als die „Gesamtheit der charakteristischen Eigenschaften“, die „Beschaffenheit“ oder auch die „Güte“ von Materialien, Sachen und Personen.

Patzak und Rattay definieren in ihrem Projekt Management Leitfaden [PR98]:

> „**Qualität** ist allgemein formuliert die Gesamtheit von Eigenschaften bzw. **Merkmalen eines Produkts** (Güter, Dienstleistungen, das Ergebnis eines Prozesses) und des **zugehörigen Prozesses**, bezogen auf die Einigung zur Erfüllung vorgegebener Anforderungen bzw. Erwartungen.“

Betrachtet man Qualität nun aus dem Kontext von (Software-)Projekten, so fasst der Standard ISO/IEC/IEEE 24765:2010(E) [ISO10] Begriffsdefinitionen diverser ISO/IEC bzw. IEEE Standards zusammen. Aus diesem Standard sind vor allem folgende Teil-Definitionen für den weiteren Gebrauch passend:

„**quality**
[. . .]
2. ability of a product, service, system, component, or process to meet customer or user needs, expectations, or requirements.
[. . .]
5. the degree to which a set of inherent characteristics fulfils requirements.
A Guide to the Project Management Body of Knowledge (PMBOK® Guide) - Fourth Edition.“

Aus den oben beschriebenen Qualitätsbegriffen kann zusammenfassend die für diese Arbeit geltende Definition von *Qualität* beschrieben werden:

Definition I.3.1 (Qualität) *In einem (Software-)Projekt, beschreibt die Qualität das Maß der Qualitätszieltreue, wobei das Qualitätsziel entweder den Anforderungen an das zu entwickelnde Produkt oder dem Entwicklungsprozess und der daraus entstehenden Artefakte entspricht.*

I.3.1.1 Produkt- vs. Prozessqualität

Wie Ludewig und Lichter in ihrem Buch „Software Engineering: Grundlagen, Menschen, Prozesse, Techniken“ [LL07] schreiben, befasst sich Software-Engineering „sowohl mit der Qualität des Produkts (*Produktqualität*) als auch mit der Qualität des Projekts, in dem das Produkt hergestellt wird (Projektqualität).“ Aus diesem Grund muss der Qualitätsbegriff weiter unterteilt werden. Das in dieser Arbeit dargestellte Kennzahlen-Messsystem *EMI* nutzt und berechnet Kennzahlen, die sowohl zu der Ermittlung der Produkt-, als auch der Prozessqualität dienen. Innerhalb dieser Arbeit wird ein großer Augenmerk auf die Prozessqualität gelegt, da das Konzept exemplarisch auf die Informationsbedürfnisse der Projektleiter angewandt wird, die das Dashboard *SCREEN* (ein Teil des *EMI*-Systems) zur Projektsteuerung nutzen können. Nach Ludewig und Lichter [LL07] kann man die beiden Qualitätsarten wie folgt definieren:

Definition I.3.2 (Produktqualität) *Bezüglich der Projektqualität kann zwischen „Gebrauchs- und Wartungsqualität“ [LL07] unterschieden werden. Zum einen wird die Qualität des Produkts aus Sicht des Benutzers betrachtet (z.B. Bedienbarkeit) und zum anderen aus Sicht des Entwicklers (z.B. Änderbarkeit).*

Definition I.3.3 (Prozessqualität) *Die Prozessqualität kann die Produktqualität beeinflussen, es besteht jedoch keine zwingende Abhängigkeit zwischen den beiden Begriffen. Sie trifft Aussagen über Kosten- und Termineinhaltung, Projektaufwände, Wissensstand und Wiederverwendbarkeit des Projektes, wie auch über das Betriebsklima.*

Das dazugehörige Qualitätsmodell von Ludewig und Lichter wird in Abschnitt I.3.3 erläutert und der daraus folgende Qualitätenbaum kann in Abbildung I.3.3 betrachtet werden.

I.3.2 Qualitätssziele

Qualitätsziele definieren „Spezifikationen von Mindestwerten von Qualitätsmerkmalen“[Kel07], welche für ein Projekt oder auch die ganze Organisation erstellt werden können. Anhand dieser Qualitätsziele sollen sich Qualitätsbeauftragte, Projektleiter und andere Projektmitarbeiter orientieren, um die Projekt- und Prozessgüte auf einem entsprechenden und akzeptablen Niveau zu halten.

Qualitätsziele können sowohl auf technischer Ebene (beispielsweise schnelle Laufzeit, Speichereffizienz, leichte Anpassbarkeit) als auch auf Prozessebene (wie schneller Fortschritt, wenig Fehler, geringe Kosten, etc.) in einem Qualitätsmodell definiert werden. Der folgende Abschnitt erläutert die Definition und den Aufbau von Qualitätsmodellen.

I.3.3 Qualitätsmodelle

Wie in Abschnitt I.3.2 bereits erwähnt, beschreibt ein Qualitätsmodell die Qualitätsziele eines Projekts beziehungsweise einer Organisation. Hierbei werden Faktoren definiert, welche bei der Qualitätserhebung, wie auch der Verbesserung des Produkts und der Prozesses eine große Rolle spielen.

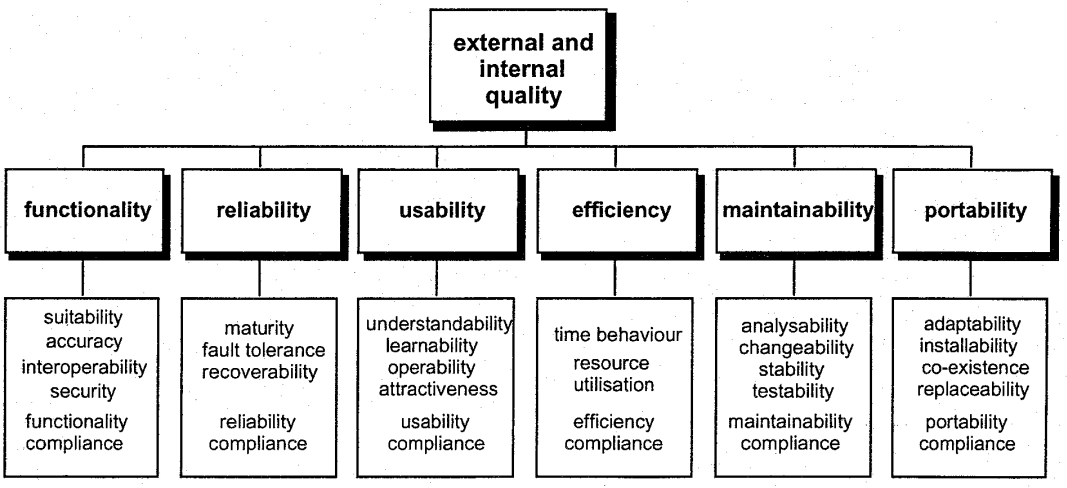

Abbildung I.3.1: Qualitätsmodell der internen und externen Qualitätsanforderungen (Abb. entnommen aus ISO/IEC 9126-1:2001(E), S.7)

Ein Beispiel eines produktbasierten Qualitätsmodells ist im Standard ISO/IEC 9126-1:2001(E) [ISO01] beschrieben. Hierbei wird die Produktqualität im Rahmen des Software-Lebenszyklus betrachtet, welche zwischen internen und externen Qualitätsanforderungen, wie auch Anforderungen bezüglich der Produktnutzung unterscheidet.

Einen groben Überblick dieser unterschiedlichen Bedürfnisse geben Abbildung I.3.1 und Abbildung I.3.2, für weitere Informationen der Begrifflichkeiten wird jedoch an dieser Stelle auf den Standard selbst verwiesen [ISO01].

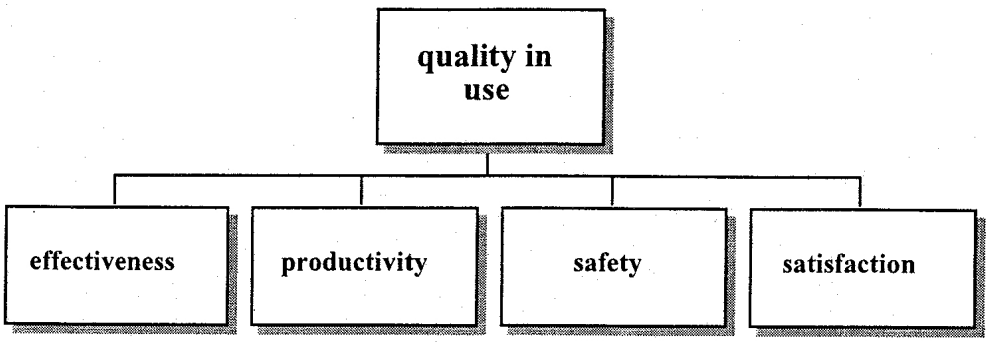

Abbildung I.3.2: Qualitätsmodell der Qualitätsanforderungen in der Produktnutzung (Abb. entnommen aus ISO/IEC 9126-1:2001(E), S.12)

Ein weiteres Qualitätsmodell wird von Ludewig und Lichter in ihrem Buch „Software Engineering: Grundlagen, Menschen, Prozesse und Techniken“ [LL07] beschrieben. Die Autoren unterscheiden hierbei, im Gegensatz zum ISO/IEC 9126-1 Standard, zwischen Produkt- und Prozessmerkmalen (siehe Abbildung I.3.3). Die Produktmerkmale ähneln denen des oben genannten Standards, hinzu kommen jedoch Merkmale zur Qualitätsbewertung von Projekt-Leistung, Planungssicherheit und innerer Prozessqualität. Genau diese Qualitätsmerkmale sind für die Qualitätssicherung ganzer Prozesse bezüglich der Projekte von Nöten. In den späteren Feldstudien (Teil III - Teil V) wird pro Organisation ein eigenes Qualitätsmodell aufgebaut, welches das Modell von Ludewig und Lichter als Grundlage benutzt. Aus diesem Grund werden die im Qualitätsbaum dargestellten Merkmale (Abbildung I.3.3) erst im Bereich der Konzeptvorstellung (Kapitel II.4), wie auch den entsprechenden Kapiteln der Feldstudien (Kapitel III.2, Kapitel IV.2 und Kapitel V.2) genauer definiert.

I.3.4 Qualitätssicherung in Projekten

„Qualitätssicherung muss im gesamten Projektmanagement stattfinden um zum einen Projektergebnisse hoher Qualität zu erzeugen und zum anderen eine hohe Qualität in der Entwicklung dieser Ergebnisse zu beinhalten“, so schreiben es Patzak und Rattay in ihrem Buch über Projekt Management [PR98]. Es stellt sich jedoch die Frage, was dieses genau bedeutet.

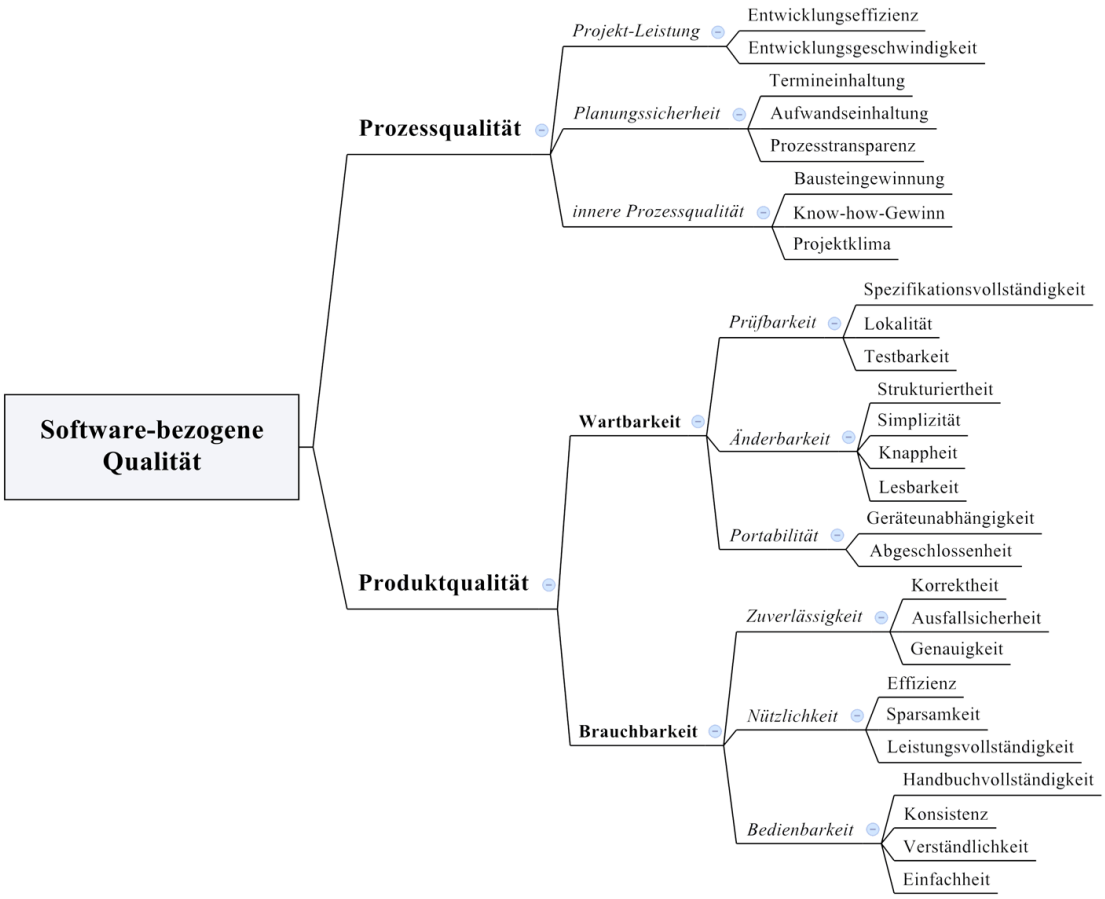

Abbildung I.3.3: Qualitätsmodell nach Ludewig und Lichter, [LL07]

Wie Ludewig und Lichter in ihrem Buch [LL07] feststellen, hat die Qualitätssicherung (QS) in Software-Projekten „die Aufgabe, alle qualitätsrelevanten Aktivitäten und Prozesse zu gestalten, zu organisieren, abzustimmen und zu überwachen.“ Man kann sie demnach wie folgt definieren:

Definition I.3.4 (Qualitätssicherung) *(Software-) Qualitätssicherung hat die Sicherstellung einer hohen Produkt-, wie auch Prozessqualität zum Ziel und nutzt resultierend daraus diverse Methoden innerhalb des Entwicklungsprozesses, die zur Kontrolle und zur Sicherstellung dieser dienen.*

Qualitätssicherung kann somit innerhalb eines Projektes auf diverse Arten und Weisen stattfinden. Angefangen von der Wahl des entsprechenden Vorgehensmodells oder Entwicklungsprozesses, bis hin zu den verwendeten Methoden, die im Weiteren genauer erläutert werden.

I.3.4.1 Wahl des Entwicklungsprozesses

Im ersten Schritt der Qualitätssicherung innerhalb eines Projekts, steht die Wahl eines geeigneten Entwicklungsprozesses an. Die in Abschnitt I.2.3.1 beschriebenen Prozessmodelle unterscheiden sich nicht nur vom Vorgehen innerhalb der Entwicklung, sondern auch von den unterschiedlichen Methoden der Qualitätssicherung. So betrachtet der *RUP* das *Testen als gesonderte Disziplin* und stellt anhand der unterstützenden Disziplinen (Projektmanagement, Änderungsmanagement, etc.) auch die Einhaltung einer hohen Qualität bezüglich der Projektartefakte sicher. Im *Extreme Programming*, einer agilen Entwicklungsmethode, existiert hingegen der *Test-First-Ansatz*, welcher die Qualitätssicherung durch das Testen in den Vordergrund der Entwicklung stellt.

Es liegt somit an der Organisation beziehungsweise dem Projektverantwortlichen einen geeigneten Entwicklungsprozess zu wählen, welcher die Qualitätsziele optimal unterstützt. Innerhalb dieses Prozesses werden weitere Maßnahmen zur Qualitätssicherung getroffen, die in Abschnitt I.3.4.2 beschrieben werden.

I.3.4.1.1 Bewertung des Entwicklungsprozesses

Um das Maß beziehungsweise die Maßnahmen der Qualitätssicherung auch nach außen, beispielsweise für den Kunden oder Mitbewerber, sichtbar zu machen, gibt es Bewertungskriterien, welche die Prozessreife des Unternehmens beurteilen. Nur ein Beispiel hierfür ist die *Capability Maturity Model Integration (CMMI)*, welche ein Unternehmen innerhalb einer Stufenbewertung von fünf Reifegraden kategorisiert. So werden für die Erreichung einer höheren Stufe, bestimmte Prozessbereiche mit Inhalten und Zielen beschrieben, die durch vorgegebene Vorgehensweisen erreicht werden müssen. Ab dem zweiten Reifegrad spielt beispielsweise die Möglichkeit, Messungen innerhalb der Organisation durchzuführen oder die Option des Einblicks in die Qualität von Arbeitsergebnissen, eine große Rolle. Weitere Einblicke in die Methoden der CMMI liefert Kapitel 11 des Buches „Software Engineering“ von Ludewig und Lichter [LL07], wie auch das ausführliche Werk zu CMMI von Kneuper [Kne06].

I.3.4.2 Maßnahmen innerhalb des Entwicklungsprozesses

Die in Abschnitt I.3.4.1 beschriebene Wahl des geeigneten Entwicklungsprozesses ist für die Sicherstellung der Projektqualität nicht ausreichend. Auch innerhalb des Prozesses müssen Maßnahmen zum Erhalt, wie auch zur Verbesserung der Qualität durchgeführt werden.

Testen

Beginnend von der Sicht der Produktqualität stellt die Möglichkeit des Testens eine Art der Qualitätssicherung dar. Neben der Nutzung manueller Tests, empfiehlt sich innerhalb der Produktentwicklung die Einführung automatisierter Tests, welche anhand regelmäßiger Durchführungen zu eine dauerhafte Qualitätskontrolle führen. Mittels der

ermittelbaren Testabdeckung kann zusätzlich festgestellt werden, in welchem Umfang und Maße die vorhandenen, automatisierten Tests, die gesamte Produktqualität widerspiegeln.

Kodierrichtlinien

Als ein weiteres Mittel der Qualitätssicherung können organisationsweite Kodierrichtlinien eingeführt werden. Diese dienen nicht der funktionalen Produktqualität, sondern führen beispielsweise zur Erreichung von Qualitätszielen wie der *einfachen Änderbarkeit* und der *Lesbarkeit*. Innerhalb diverser Entwicklungsumgebungen (wie beispielsweise Eclipse [11]) können erstellte Vorlagen, Formatierungsschemata oder andere Teile der Richtlinien automatisch integrieren.

Nutzung von Framesworks

Entwicklungsframeworks können die Erhöhung der Produktqualität ebenfalls deutlich unterstützen. Sie geben Entwicklungsrichtlinien, wie auch die Strukturierung der Implementierung vor. Somit können nicht nur Qualitätsziele wie die *Wartbarkeit* und *Lesbarkeit*, sondern auch die der *Strukturiertheit* und *Simplizität* erfüllt werden. Ein Beispiel für ein nützliches Framework in der Java-Entwicklung ist *JUnit* [3], welches der Entwicklung automatisierter Tests in Java dient.

Einhaltung externer Richtlinien und Referenzmodelle

Auch externe Referenzmodelle mit klaren Richtlinien fördern die Produkt- und Prozessqualität. So gibt beispielsweise das *CMMI-Referenzmodell* Richtlinien zur Einhaltung eines gewünschten Reifegrades des Entwicklungsprozesses oder die *Information Technology Infrastructure Library (ITIL)* Richtlinien bzw. Praktiken im Bereich des IT-Service-Managements vor (siehe auch [Köh07]).

Kennzahl-Erhebungen

Eine fortgeschrittene Maßnahme zur Qualitätssicherung ist die Erhebung von Projektkennzahlen. Sie dient der Überwachung des Projektstatus und kann zur Verbesserung der Projekt- wie auch der Produktqualität führen, durch den direkten Eingriff bei möglichen Verschlechterungen. Um Projektkennzahlen nutzen zu können, müssen Metriken entwickelt werden, welche auf vorhandenen Projektdaten Berechnungen ausführen und die daraus erhaltenen Kennzahlen im besten Fall simpel und verständlich visualisieren.

Innerhalb dieser Arbeit ist die Kennzahlerhebung und die Darstellung dieser durch ein Projekt-Dashboard, die in den Vordergrund gestellte Maßnahme zur Qualitätssicherung. Bevor in Teil II ein Konzept zur Einführung von Kennzahlerhebungen mittels Kennzahlen-Messsysteme (im Fall dieser Arbeit dem System EMI, siehe auch Kapitel I.4) vorgestellt wird, soll im nachfolgenden Abschnitt I.3.5 die Begrifflichkeit der *Metriken* definiert und genauer erläutert werden.

I.3.5 Metriken

Der Begriff der Metrik stammt aus dem griechischen und bedeutet „'aus dem Meter' oder 'auf dem Meter basierend'“ [37]. Anhand des Wikipedia-Eintrages dieses Begriffs wird deutlich, dass Metriken in vielen Bereichen des Lebens, wie beispielsweise der Mathematik, Physik, Literatur oder Softwareentwicklung, eine Rolle spielen. Innerhalb dieser Ausarbeitung zählt die Definition der Metrik im Bereich der Softwareentwicklung, auf die folglich eingegangen wird.

Im ISO/IEC/IEEE Standard 24765:2010(E), dem Vokabular der System- und Software Engineering Begriffe [ISO10], wird der Metrikbegriff unter Punkt 3.1767 wie folgt definiert:

> „**metric**
> **1.** a combination of two or more measures or attributes. *ISO/IEC 20926:2003, Software engineering - IFPUG 4.1 Unadjusted functional size measurement method - Counting practices manual*
> **2.** a quantitative measure of the degree to which a system, component, or process possesses a given attribute.
> **3.** the defined measurement method and the measurement scale. *ISO/IEC 14598-1:1999, Information technology - Software product evaluation - Part 1: General overview.4.20* “

Zusammengefasst besagen diese Definitionen, dass eine Metrik sowohl ein Maß oder eine Zusammenfassung von Maßzahlen ist, als auch die Methodik beziehungsweise Skala zur Erfassung dieser Maßzahlen liefert. Wikipedia nennt eine (Software-)Metrik „eine (meist mathematische) Funktion, die eine Eigenschaft von Software in einen Zahlenwert, auch Maßzahl genannt, abbildet“ [14]. Aus diesem Grund muss vor der Definition der Metrik die Begrifflichkeiten der Maßzahl (von nun an auch *Kennzahl* genannt) und der Messdaten bezüglich dieser Ausarbeitung erläutert werden.

Definition I.3.5 (Messdaten) *Messdaten sind Rohdaten, die von Datenquellen (wie beispielsweise BI-Systemen, Excel-Dokumenten, Datenbanken) an das Kennzahl-Messsystem übermittelt werden, auf dessen Basis Metriken zu bilden und Kennzahlen zu berechnen sind.*

Definition I.3.6 (Kennzahlen) *Kennzahlen sind die Ergebnisse der Metrik-Berechnungen auf den Messdaten und dienen zur Interpretation des Projektstatus.*

Der Begriff der Metrik wird innerhalb dieser Arbeit, ähnlich wie in Punkt drei der obigen Definition des System- und Software Engineering Vokabular, nicht nur als Berechnungsvorschrift, sondern als auch als Methode, Skala und Visualisierung betrachtet.

Definition I.3.7 (Metrik) *Eine Metrik umfasst neben der Berechnung von Kennzahlen aus gegebenen Messdaten auch die Visualisierung dieser. Zusätzlich liefert eine Metrik eine Interpretationshilfe zum Verständnis der Visualisierung, der Kennzahlen und der Anwendung dieser. Innerhalb dieser Arbeit und vor allem bezüglich des Messsystems EMI, entspricht eine Metrik dem Dashboard-Widget des Kennzahlen-Dashboard SCREEN.*

Betrachtet man Metriken in Bezug auf Ihre Berechnungsvorschriften, so kann weiterhin zwischen Basismetriken und Pseudometriken unterschieden werden (siehe auch Kapitel 14.2 „Objektive Metriken, Messung“ bzw. 14.4 „Pseudometriken“ in [LL07]):

Definition I.3.8 (Basismetrik) *Basismetriken oder auch Basiskennzahlen basieren auf ermittelten Messdaten. Sie spiegeln objektive Werte wider und führen Berechnungen rein auf den gegebenen Messwerten aus.*

Definition I.3.9 (Pseudometriken) *Im Gegensatz zu Basismetriken, sind Pseudometriken nicht direkt messbar, sondern werden aufgrund gemessener Daten und Basismetriken berechnet. Sie spiegeln komplexe, aber auch subjektive Werte wider, welche meist einen projektspezifischen Interpretationsraum für ihre Auswertung benötigen.*

Im folgenden Teil werden die Definitionen der Basis- und Pseudometriken anhand einer kleinen Auswahl an typischen Software- und Prozessmetriken verdeutlicht.

I.3.5.1 Softwaremetriken

Softwaremetriken dienen der Vermessung und Informationsableitung des Softwareprodukts. Da die Anzahl der typischen Softwaremetriken sehr groß ist, werden an dieser Stelle nur drei beispielhaft erläutert.

Lines of Code

Die wohl typischste und am schnellsten assoziierte Software-Metrik ist *Lines of Code (LOC)*. Diese liefert den Umfang der Codezeilen, jedoch ist eine allgemeingültige Definition dieser Metrik nicht vorhanden. So muss innerhalb der Organisation entschieden werden, wie LOC definiert wird. Beispielsweise können alle Zeilen des Programms (inklusive der Leerzeilen) betrachtet werden (LOC_{tot}) oder nur die nichtleeren Programmzeilen (LOC_{ne}). Wichtig hierbei ist die einheitliche Verwendung dieser Definition, da andernfalls weitere Aussagen und Berechnungen nicht vergleichbar sind. LOC ist eine typische Basismetrik, dessen Aussage alleine nicht viel Ausdrucksstärke besitzt. So können beispielsweise Programme unterschiedlicher Sprache nicht verglichen werden, da der Umfang für die selbe Funktion je Sprache unterschiedlich sein kann. Sie dient jedoch zu weiteren Berechnungen wie beispielsweise der Komplexität.

Lack of Cohesion in Methods

Eine weitere Basismetrik ist *Lack of Cohesion in Methods (LCOM)*, welche den Zusammenhalt einer objektorientierten Klasse beschreibt. So ist der Zusammenhalt innerhalb einer Klasse hoch, sofern „ihre Attribute und Methoden einen einzigen zusammengehörenden Aspekt implementieren“ [LL07]. Die mathematische Definition der Klasse sieht wie folgt aus:

$$\text{LCOM} = \begin{cases} |P| - |Q|, \text{ falls} |P| > |Q| \\ 0, \text{ sonst} \end{cases}$$

mit P, Q Mengen an Methodenpaaren, die kein ($\hat{=}$ P) bzw. mind. ein ($\hat{=}$ Q) Attribut benutzen

Komplexität nach McCabe

Eine typische Pseudometrik im Bereich der Software Entwicklung ist die *zyklomatische Komplexität* nach McCabe. Sie beruht auf der Graphentheorie, wird anhand des Flussgraphen des Programms berechnet und gibt die Komplexität anhand der Verschachtelungstiefe der Bedingungen an. Simple ausgedrückt erhöht sich die Komplexität eines Programms aufgrund der Anwendung gewisser Programmelemente (beispielsweise IF-Anweisungen, Schleifen) um mindestens den Wert 1. Je höher dieser Wert, desto komplexer ist das Programm. Jedoch ist die Aussage dieser Zahl nicht allgemeingültig. So werden beispielsweise komplexe Datenstrukturen bei der Berechnung nicht berücksichtigt, wodurch die Kennzahl zwar niedrig gehalten wird, die Komplexität dieses Programms jedoch deutlich höher ist. Für die mathematische Definition dieser und weiterer Metriken wird auf das Standardwerk „Software Metrics“ von Norman E. Fenton [Fen91] verwiesen.

I.3.5.2 Prozessmetriken

Neben den oben erwähnten Software-Metriken, können innerhalb der Entwicklung auch Metriken verwendet werden, welche die Qualität des Entwicklungsprozesses widerspiegeln. Diese dienen der Evaluation von Kosten, Zeit und Aufwand innerhalb des Projekts. Folglich werden beispielhaft die Kennzahlen der Leistungswertanalyse (*Earned Value*) erläutert, weitere Prozessmetriken werden in den einzelnen Fallstudien (Teil III-Teil V) dargestellt.

Earned Value Analyse

Die *Earned Value Analyse* ist eine Analyse der Projektleistung bezüglich Kosten, Zeit und Qualität. Für die Durchführung ist ein strukturiertes Projekt von Nöten, dessen Arbeitsaufwände vor der Analyse geschätzt werden müssen. Hierfür empfiehlt sich die Schätzung einzelner Arbeitspakete. Mit Hilfe dieses Kalküls kann der *geplante Aufwand* (*Planned Value (PV)*) ermittelt und anhand der *aktuellen Kosten* (*Actual Cost (AC)*) der *Soll-Aufwand* (*Earned Value (EV)*) für bereits abgeschlossene Arbeitspakete zum Zeitpunkt der Erhebung berechnet werden. Dieser Soll-Aufwand gibt den geplanten

Aufwand zum Erhebungszeitpunkt unter Berücksichtigung des geplanten Projektaufwandes (sowohl Kosten, als auch Zeit) an. Abbildung I.3.4 zeigt eine typischen Earned-Value-Graphen.

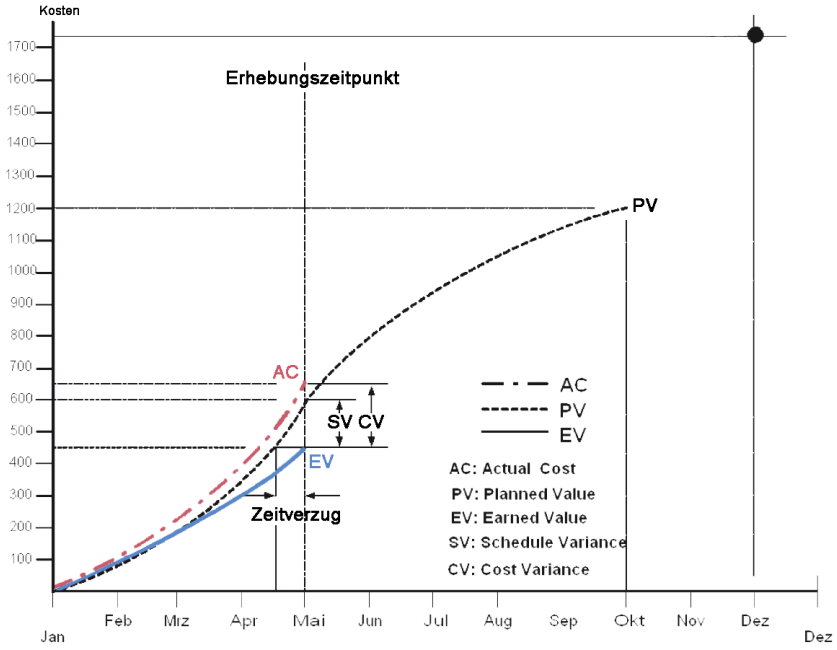

Abbildung I.3.4: Typischer Graph der Earned Value Analyse (abgeänderte Abb. aus [DH07], S.234)

Anhand eines Vergleichs tatsächlich aufgebrachten Kosten (AC) mit dem Soll-Wert (EV) kann eine mögliche *Kostenabweichung* (Cost Variance (CV)) erkannt und interpretiert werden. Diese Abweichung kann sowohl auf eine einmalige Abweichung innerhalb des Projekts, als auch auf eine Fehlschätzung des Aufwandes hinweisen. Des weiteren ist auch die *Zeitabweichung* (*Schedule Variance (SV)*) anhand des Soll- und des Plan-Werts ermittelbar, die einen möglichen Zeitverzug andeuten kann. Die Berechnung dieser Werte wird Tabelle I.3.1 angegeben.

Wie ebenfalls in Tabelle I.3.1 ersichtlich, existieren zwei weitere Earned-Value-Werte: der *Cost Performance Index (CPI)* und der Schedule Performance Index (SPI). Diese Werte geben einen Performance-Indikator für die Abweichung von Kosten und Zeit an. So ist der Idealwert dieses Indikators 1. Ein Wert < 1 deutet auf erhöhte Projektkosten, beziehungsweise Zeitverzug hin, ein Wert > 1 macht einen geringeren Kosten- beziehungsweise Zeitaufwand als geplant deutlich. Eine Ermittlung der Projektleistung anhand der beiden Werte CPI und SPI ist, vor allem bei Großprojekten, schwierig. So scheint ein SPI von 0,99 nicht schlecht zu sein, da er nahe dem Idealwert ist. Hat ein Großprojekt jedoch 20.000 PT, so bedeutet der Geschwindigkeitsverlust von nur

einem Prozent eine Erhöhung der Projektlaufzeit um 200 PT. Aus diesem Grund ist es empfehlenswert die Projektbewertung anhand der Abweichungen (CV, SV) anstatt der Performance-Indizes vorzunehmen, da dadurch die reellen Werte und nicht nur die Verhältnisse mit schwacher Aussagekraft abgebildet werden. Weitere Informationen zu der Earned Value Analyse sind auch in diverser Literatur, wie [DH07], [KHL+11], [Pre01], [36] und [34] zu finden.

Tabelle I.3.1: Berechnungen der Earned-Value-Werte, wobei angenommen wird, dass Kosten = Aufwand (in Personentage)

Bezeichnung	Beschreibung	Berechnung
PV (Planned Value)	Geschätzter Arbeitsaufwand aller Arbeitspakete	
AC (Actual Value)	tatsächlich verbrauchter Arbeitsaufwand	
EV (Earned Value)	Soll-Aufwand an bestimmten Ergebnugszeitpunkt	EV = Projektbudget * prozentualer Arbeitsfortschritt[1]
CV (Cost Variance)	Kostenabweichung	$C = EV - AC$
SV (Schedule Variance)	Zeitplanabweichung	$SV = EV - PV$
CPI (Cost Performance Index)	Kostenindikator	$CPI = \frac{EV}{AC}$
SPI (Schedule Performance Index)	Zeitindikator	$SPI = \frac{EV}{PV}$

I.3.6 Goal-Question-Metric-Methode

Um passende Metriken innerhalb einer Organisation zu finden, kann eine Methode angewandt werden, die *Goal-Question-Metric (GQM)* genannt wird. Sie ist ein Top-Down-Mechanismus, der von den abstrakten Zielen Fragestellungen ableitet, welche zu detaillierten Metriken führen. Folglich werden die drei Schritte der Metrik-Findung nach Basili, Caldiera und Rombach [BCR94] erläutert, welche im GQM-Modell Abbildung I.3.5 visuell dargestellt werden.

Stufe 1: Findung der Ziele

In der ersten Stufe müssen Qualitätsziele innerhalb der Organisation definiert werden. Dieser konzeptionelle Schritt kann für Prozesse, Produkte und Ressourcen erfolgen und

[1] Wird anhand der abgeschlossenen Arbeitspakete ermittelt, weitere Informationen auch in [34] Abschnitt 1.4

dient einem gemeinsamen Verständnis des Qualitätsbegriffs. Mögliche Qualitätsziele können beispielsweise die fehlereffiziente Entwicklung, geringer Aufwand oder leichte Änderbarkeit des Quellcodes sein. Diese Ziele werden anhand von vier Betrachtungswinkeln (Zweck, Qualitätsaspekt, Betrachtungsgegenstand und Perspektive) kategorisiert. Für das Qualitätsziel einer fehlereffizienten Entwicklung wären diese „Zielfacetten“ [Wer12] folgende:

- **Zweck**: Verbesserung
- **Qualitätsaspekt**: Reife
- **Betrachtungsgegenstand**: Quellcode
- **Perspektive**: Entwickler, Projektleiter

Stufe 2: Erstellung des Fragenkatalogs

Der nächste operationale Schritt ist die Findung von Fragestellungen bezüglich der erhobenen Qualitätsziele. Diese sollen die Messobjekte nach bestimmten Gesichtspunkten (wie Produkt, Prozess und Ressourcen [BCR94]) charakterisieren und deren Qualität anhand dieser Punkte bestimmen. So könnten beispielsweise folgende Fragen für das Ziel einer fehlereffizienten Entwicklung gestellt werden:

- Wie hoch ist die Fehlerrate im Projekt?
- Wie verläuft die Fehlerentwicklung während der Projektlaufzeit?
- Welche kritischen Fehler existieren?
- Wie schnell verläuft die Fehlerabarbeitung?
- Treten bestimmte Fehler(typen) regelmäßig auf?

Stufe 3: Definition der Metriken

Die letzte Stufe umschreibt den quantitativen Schritt der Definition von Projekt-, Prozess- und Produktmetriken. Hier können Basis- und Pseudometriken (siehe Abschnitt I.3.5) anhand der gegebenen Datenquellen definiert werden. Für das Beispiel des Ziels der fehlereffizienzen Entwicklung wären das folgende Metriken:

- Anzahl der offenen Fehler
- Anzahl der Fehler innerhalb des gesamten Projektverlaufs
- Durchschnittliche Fehler-Abarbeitungsdauer
- Entwicklung der Fehler im Vergleich zur Quellcode-Entwicklung

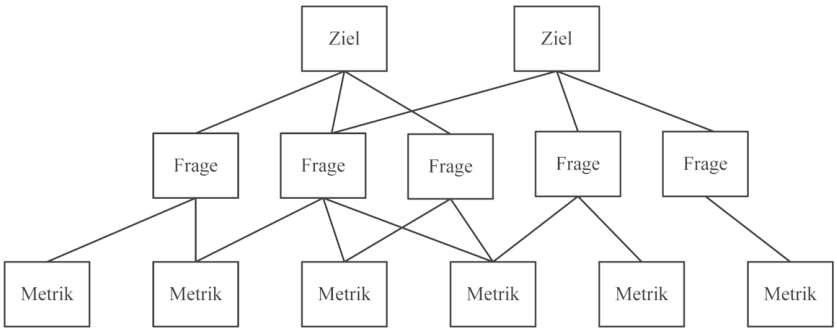

Abbildung I.3.5: GQM-Modell, welches mögliche Abhängigkeiten der Ziele, Fragen und Metriken darstellt

I.3.6.1 GQM-Modell

Innerhalb des GQM-Ansatzes wird ein hierarchisches GQM-Modell aufgebaut, wie zuvor in Abbildung I.3.5 verdeutlicht wurde. Dieses Modell kann mithilfe von *Abstraction Sheets* dokumentiert werden. Diese Darstellungsform bietet die Möglichkeit einer systematischen GQM-Vorgehensweise und liefert als Ergebnis die Dokumentation des GQM-Modells. Abbildung I.3.6 stellt ein solches *Abstraction Sheets* dar. Einen Editor zur vereinfachten Anwendung der GQM-Methodik wurde am Fachgebiet Software Engineering des Instituts für Praktische Informatik der Leibniz Universität Hannover durch die Bachelorarbeit von Florian Werner [Wer12] erstellt. Weitere Informationen zu diesem Editor findet sich ebenfalls in dem Paper von Raphael Pham [PS13].

Zweck	**Qualitätsaspekt**	**Betrachtungsgegenstand**	**Perspektive und Umgebung**
Qualitätsfaktoren Welche Faktoren definieren den Qualitätsaspekt?		**Einflussfaktoren** Was hat Einfluss auf die Qualitätsfaktoren?	
Ausgangshypothese Momentane Erwartungen über die Qualitätsfaktoren?		**Einflusshypothese** Was beeinflusst wie die Qualitätsfaktoren?	

Abbildung I.3.6: Abstraction Sheet der GQM-Analyse (angelehnt an [Wer12], Abb.5)

Innerhalb dieser Diplomarbeit wurde eine abgeschwächte, nicht-strikte Variante des GQM-Ansatzes gewählt. Zwar wurde die Herleitung der Metriken in drei Stufen eingehalten, jedoch keine Abstraction Sheets verwendet, sondern Ziele, Fragen und Metriken anhand der Mind-Mapping-Technik [38] zusammengetragen. Ausführliche Informationen zu dem empfohlenen GQM-Ansatz dieser Arbeit sind in Kapitel II.3 der konzeptionellen Anforderungsanalyse beziehungsweise in den jeweiligen Kapiteln innerhalb der Fallstudien (Kapitel III.2, Kapitel IV.2, Kapitel V.2) zu finden.

Nach der Klärung des Qualitätsbegriffs schließt das folgende Kapitel den Teil der Grundlagen mit der Erläuterung des Kennzahl-Messsystems *EMI*, seiner Dashboard-Komponente *SCREEN* und weiteren Arbeiten aus diesem Bereich ab.

I.4 Das Metrik-System *EMI*

> The basic rule of software status reporting can be summarized in a single phrase: "No surprises!".
>
> Capers Jones

Inhalt

Innerhalb dieses Kapitels wird das vom Lehr- und Forschungsgebiet Software-Konstruktion der RWTH Aachen entwickelte Kennzahlen-Messsystem *EMI* und die dazugehörige Dashboard-Komponente *SCREEN* vorgestellt. Vor allem das Dashboard *SCREEN*, welches der Frontend-Zugriff der Benutzer ist, stellt einen wesentlichen Bestandteil dieser Arbeit dar. In früheren Arbeiten zu diesem System wurde an statt der Begriffe *EMI* und *SCREEN* vom*MeDIC*-System gesprochen. Diese Bezeichnung wird innerhalb dieser Arbeit nicht mehr in Bezug auf dieses System verwendet.

In den Feldstudien (Teil III-Teil V) werden sowohl Anforderungsanalyse und prototypische Entwicklung einer Dashboard-Lösung, als auch die fachliche Integration dieses Systems genauer erläutert. Folglich wird das *EMI*-System in kurzen Auszügen vorgestellt, sowie auf die Nutzung von Dashboards im Bereich der Projektsteuerung und der speziellen Dashboard-Lösung *SCREEN* detailliert eingegangen.

I.4.1 Das *EMI*-System

Die „*föderalistische Enterprice Measurement Infrastructure*“ [VLS] ist ein Messsystem, welches von Matthias Vianden am Lehr- und Forschungsgebiet für Software-Konstruktion der RWTH Aachen innerhalb einer Forschungsarbeit konzipiert und mithilfe diverser Abschlussarbeiten realisiert wurde. Es dient der Messung von Produkt- und Prozessmetriken innerhalb Software-Entwicklungsprojekte. Der größte Unterschied zu vorhandenen zentralisierten Messsystemen wie beispielsweise der IBM-Lösung *Rational Insight* [20] (siehe auch Kapitel 6.3 in [Ste13]) ist das föderalistische Prinzip. So können mittels EMI Informationen diverser heterogenen Datenquellen wie BI-Systeme (*Business Intelligence*), CRM-Werkzeugen (*Change-Request-Management*), Datenbanken, Code-Analyse-Tools oder gar Dateien wie Excel-Dokumente ausgelesen, evaluiert,

berechnet und visualisiert werden. In zentralisierten Systemen ist dies jedoch nur bedingt möglich, da sie oftmals auf der Nutzung ihrer eigener System-Landschaften beruhen, welche externe Integrationen erschweren, beziehungsweise bestimmte Daten-Schemata vorschreiben, die gegebenenfalls umständlich oder gar nicht von externen Datenquellen abgebildet werden können. Das EMI-System bietet durch den komponentenbasierten Aufbau, welcher in der Diplomarbeit von Andreas Steffens [Ste13] ausführlich dargestellt wird, eine individuelle Anpassung verschiedener Datenquellen. Im Folgenden werden die einzelnen Komponenten des Systems schemenhaft erläutert.

I.4.1.1 *EMI*-Systemkomponenten

Die *Enterprice Measurement Infrastructure* basiert auf dem ISO 15939 Messprozess, welcher Aufgaben und Aktivitäten zur Verbesserung der Vermessung von Projekten und organisationsinterner Messstrukturen bereithält [Ins09]. Dieser Standard wird auch in der Ausarbeitung von Andreas Steffens [Ste13] in Kapitel 2.1 genauer beschrieben. Die Architektur des *EMI*-Systems baut auf einer Schichtenarchitektur auf, welche Frontend, Backend, Datentransport und -verarbeitung, wie auch eine operationale Ebene trennt. Dieser Aufbau wird in Abbildung I.4.1 verdeutlicht und im Folgenden genauer erläutert.

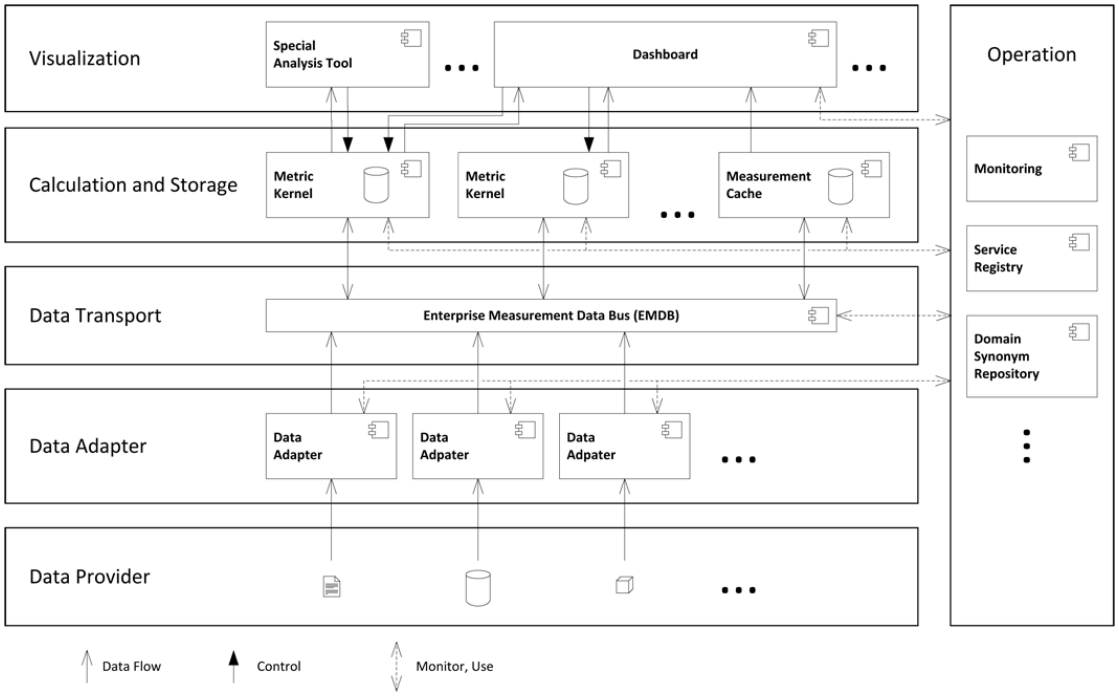

Abbildung I.4.1: *EMI*-Systemarchitektur (erstellt durch Matthias Vianden)

Externe Datenquellen, wie beispielsweise Business Intelligence (BI)- und Change Request (CR)-Systeme, Datenbanken oder Dateien liefern Messdaten, welche im System verarbeitet und dem Messkunden visualisiert dargestellt werden.

Datenadapter dienen der Anpassung externer Datenquellen an das *EMI*-System. An dieser Stelle zeigt sich die Heterogenität des Systems, da Datenadapter für beliebige Datenquellen erstellt und erweitert werden können.

Datentransport durch den *Enterprise Measurement Data Bus (EMDB)* stellt eine weitere Architekturschicht dar. Dieser Datenbus regelt die Weiterleitung der eingelesenen Daten in Berechnungskerne oder Speicherungskomponenten. Er wurde im Rahmen der Diplomarbeit von Andreas Steffens entwickelt [Ste13].

Berechnungs- und Speicherungsschicht enthält diverse Metrik-Berechnungskerne, welche Berechnungen auf den eingespeisten Messdaten ausführen können. Zusätzlich können diese auch innerhalb des Systems abgespeichert werden, so dass keine externe Speicherung von Nöten ist.

Visualisierungsschicht ist die Schicht, die dem Benutzer zur Verfügung gestellt wird. Die Hauptkomponente dieser Schicht ist die Dashboard-Lösung *SCREEN*, mit welcher dem Benutzer die vorhandenen und berechneten Metriken visualisiert dargestellt werden können. Zusätzlich befinden sich weitere Komponenten wie beispielsweise ein Dashboard-Prototyping-Werkzeug [Gor13] oder ein Analysewerkzeug für Ticket-Flüsse innerhalb von Projekten [Chr13] in Aufbau.

Betriebsschicht dient der Kontrolle des Systems. Diese Schicht befindet sich zum Zeitpunkt dieser Arbeit noch in Aufbau, jedoch sollen operationale Komponenten, wie beispielsweise Überwachungswerkzeuge für den Datenbus oder die Benutzung des Dashboards erstellt werden.

Da diese Arbeit die fachliche Integration aus Sicht des Messkundens in den Vordergrund stellt, wird die zuvor erwähnte Dashboard-Lösung *SCREEN*, wie auch allgemeine Informationen bezüglich der Benutzung von Dashboards zur Projektsteuerung im folgenden Abschnitt erörtert.

I.4.2 Die Komponente *SCREEN*

Vor der Erläuterung der Dashboard-Komponente des EMI-Systems, wird an dieser Stelle der Begriff des *Dashboards* definiert und wichtige Eigenschaften nach Stephan Few [Few06] erläutert.

I.4.2.1 Dashboards im Allgemeinen

Eine für den Benutzer wichtige Komponente eines Kennzahlen-Messsystems, ist das Metrik-Dashboard. Mithilfe dieses Werkzeugs kann der Projektstatus durch die Visualisierung diverser Kennzahlen einfach und auf einen Blick erfasst werden. In der Welt

der Kennzahl-Visualisierungen, gibt es unterschiedliche Bezeichnungen wie beispielsweise *Cockpits* oder *Scoreboards*, welche oftmals synonym verwendet werden. Die Unterschiede dieser Begrifflichkeiten werden in der Masterarbeit von Frederic Evers [Eve12] detailliert erläutert und sollen an dieser Stelle nicht von weiterem Belangen sein. Stattdessen stellt sich die Frage, was unter einem Dashboard letztendlich zu verstehen ist. Diese Frage kann durch die folgende Definition des Begriffs durch Stephen Few [Few04] genauer beantwortet werden. In seinem Buch *Information Dashboard Design: The Effective Visual Communication of Data* [Few06] erläutert er neben der Definition und den unterschiedlichen Einsatzzielen eines Dashboards auch Prinzipien zur optimalen Gestaltung, typische Anforderungen und Design-Fehler bei der Entwicklung von Dashboards. Diese Aspekte werden bei der Erstellung von des *SCREEN*-Dasboards berücksichtigt. Die wichtigsten Prinzipien werden in Abschnitt I.4.2.1.1 kurz erläutert. Zuvor soll der Begriff des Dasbhoards für die Verwendung innerhalb dieser Arbeit definiert werden. Hierfür spiegelt die Definition nach Stephen Few [Few04] die wichtigsten Merkmale eines Dashboards wider, so dass sie frei übersetzt auch für diese Ausarbeitung angenommen wird.

Definition I.4.1 (Dashboard) *Ein Dashboard ist ein visuelles Display, welches die wichtigsten Informationen zur Erreichung eines oder mehrerer Ziele darstellt. Diese Informationen werden zusammengefasst und auf einem einzelnen Bildschirm angeordnet dargestellt, so dass sie auf einen Blick überwacht werden können.*

Die genannten Ziele, für die ein solches Dashboard genutzt werden kann, können sowohl strategisch, analytisch oder auch operational sein. So kann ein Dashboard zur reinen Wiedergabe von Projektinformationen genutzt werden, um anhand dieser Entscheidungen für den weiteren Projektverlauf zu treffen. Hierfür reichen oftmals „high-level“ Kennzahlen bezüglich der Projektleistung zu bestimmten Zeitabschnitten innerhalb des Projekts aus (beispielsweise einer monatlichen oder wöchentlichen Datenerhebung), mithilfe dessen man unter anderem Prognosen des weiteren Vorgehens erstellen kann [Few06]. Ein weiteres Ziel kann der analytische Nutzen des Dashboards sein. In diesem Fall werden keine oberflächlichen Kennzahlen, sondern komplexe Datenauswertungen visuell dargestellt, die tiefgründigere Informationen zu bestimmten Daten liefern. Auch in diesem Fall reicht eine statische Datenergebung oftmals aus. Die operationale Nutzung ist die komplexeste Art einer Dashboard-Verwendung. Sie geht davon aus, dass Echtzeitdaten zur Verfügung stehen und nutzt das Dashboard als Überwachungsmonitor der Daten, auf die bei Veränderungen direkte Aktivitäten folgen können. Aufgrund dieser Komplexität stellt das *EMI*-Dashboard *SCREEN* ein operationales Dashboard dar, welches jedoch ebenso für strategische oder analytische Ziele eingesetzt werden kann.

I.4.2.1.1 Prinzipien des Dashboard-Designs

Um ein Dashboard für eine ideale Nutzung zu gestalten, müssen die Prinzipien der menschlichen Wahrnehmung beachtet werden. Durch die Ausnutzung dieser können Informationen einfach und effektiv innerhalb eines visuellen Displays vermittelt werden. Durch die Benutzung der gestaltungspsychologischen Gruppierungsgesetze können Daten

einfach organisiert werden. Visuelle Kodierungsrichtlinien dienen zusätzlich der Hervorhebung wichtiger Informationen und auch die Kapazität des menschlichen Kurzzeitgedächtnisses ist für die Aufnahme wichtiger Informationen nicht zu vernachlässigen. Im folgenden Abschnitt werden diese Aspekte erläutert.

Gruppierungsgesetze der Gestaltpsychologie

Ein erster essentieller Punkt der Informationsaufnahme ist die Gruppierung von Komponenten innerhalb des Dashboards. So existieren für die menschliche Wahrnehmung Gruppierungsgesetze der Gestaltpsychologie, welche Zusammengehörigkeit von Elementen definieren. Sollen Elemente in Kontext zueinander gesetzt werden, so sollten sie mindestens eine der folgenden Eigenschaften aufweisen:

- räumliche Nähe zueinander (Gesetz der Nähe)
- ähnliches Erscheinungsbild (Gesetz der Ähnlichkeit)
- gleichgerichtete und zueinander abgestimmte Charakteristika (Gesetz der Kontinuität, siehe Abbildung I.4.2)
- offene Strukturen, die als zusammenhängendes Objekt wahrgenommen werden (Gesetz der Geschlossenheit, siehe Abbildung I.4.3)
- umgrenzt durch einen Rahmen oder eine farbliche Fläche sein (Gesetz der Umrandung)
- durch eine Linie verbunden sein (Gesetz der Verbindung)

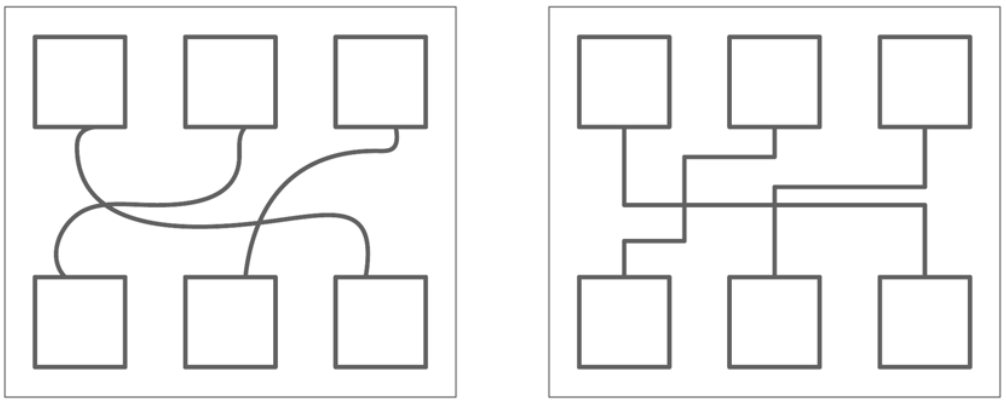

Abbildung I.4.2: *Gesetz der Kontinuität* - die linke Darstellung beachtet das Gesetz der Kontinuität, eine Verbindung der oberen und unteren Quadrate ist eindeutig erkennbar. Die rechte Darstellung verdeutlicht, dass diese Kontinuität durch die Verwendung von Kanten nicht mehr eindeutig ersichtlich ist. So kommt bei der Betrachtung der Verbindungen auch eine alternative Abzweigung an einem Linienkreuzungspunkt in Frage.

Abbildung I.4.3: *Gesetz der Geschlossenheit* - obwohl die Linien nicht abschließen, werden die sie als Formen (Rechteck, Ellipse) wahrgenommen

Weitere Informationen aus dem Bereich der Wahrnehmungs- und Aufmerksamkeitstheorie sind auch im Werk *Attention: Theory and Practice* von Johnson und Proctor [JP04] oder in Kapitel 4 des Buches *Information Dashboard Design* [Few06] zu finden.

Visuelle Kodierung von Daten

Neben den Gruppierungsgesetzen, kann auch die Kodierung anhand von Farben, Formen, Positionen und Bewegungen eine große Rolle im Bereich des Dashboard-Designs spielen. So können wichtige Elemente aufgrund der Nutzung unterschiedlicher Kodierungen in den Betrachtungsfokus des Benutzers gezogen beziehungsweise unwichtige Elemente in den Hintergrund gebracht werden.

- Farbe: Sowohl der Farbton, als auch die Farbintensität können für die visuelle Wahrnehmung von Elementen eine Rolle spielen. So kann ein Item durch prägnante Farben (sattes Rot) hervorgehoben oder in den Hintergrund (schlichtes Grau) gerückt werden.

- Position: Die Positionierung von Komponenten dient ebenfalls der Wahrnehmung und Hervorhebung von Informationen. So gilt in westlichen Ländern, dass im Zentrum des visuellen Blickfeldes Informationen zuerst und danach in der Reihenfolge von links oben nach rechts unten wahrgenommen werden.

- Form: Auch die Form von Elementen ist nicht zu vernachlässigen. So können sowohl die Länge, als auch die Breite von Linien Informationen kodieren. Formen, Größen und Ausrichtungen von Elementen können ebenfalls zu einer schnelleren Wahrnehmung dieser führen.

- Bewegung: Durch Blinken beziehungsweise Flackern von Objekten, können diese in den Fokus der Betrachtung gezogen werden.

Kapazität des menschlichen Kurzeitgedächtnisses

Des Weiteren spielt das menschliche Kurzzeitgedächtnis bei der Verarbeitung von Informationen eine große Rolle. So hat es eine Kapazitätsspanne von 7 ± 2 Chunks. Dies bedeutet, dass zwischen fünf bis neun Einheiten beziehungsweise Items zeitgleich wahrgenommen und verarbeitet werden können. Für den Bereich des Dashboard-Designs ergibt sich hierdurch die Regel nicht zu viele visuelle Reize auf einmal zu setzen, beispielsweise nicht mehr als neun unterschiedliche Dashboard-Widgets darzustellen.

I.4.2.1.2 Wahl der Kennzahl-Darstellungen

Neben den Prinzipien der menschlichen Wahrnehmung ist auch die Wahl der richtigen Visualisierungsform wichtig. Gewisse Diagrammearten sind mehr oder weniger geeignet für die Vermittlung von Informationen.

Sinnvolle Darstellungsformen

Für die Darstellung von Kennzahlen in Dashboards existieren eine Reihe von Graphen und anderen Möglichkeiten, welche sinnvoll zur Informationsvermittlung eingesetzt werden können. Sie beruhen auf der einfachen und verständlichen Wahrnehmung durch den Benutzer. Abbildung I.4.4 visualisiert diese möglichen Darstellungsformen.

- Graphen: Komplexe Verhältnisse zwischen unterschiedlichen Werten können mittels der Verwendung von Graphen dargestellt werden.
  - *Punktdiagramme*: Individuelle Werte können innerhalb eines Graphen durch Punkte dargestellt werden. Durch die Verwendung von Punktgraphen kann die Verteilung einer größeren Menge an Werten innerhalb eines Graphen dargestellt werden. Durch Verdichtungen (auch *Punktwolken* genannt) können die quanititativen Schwerpunkte der Werte erkannt werden.
  - *Liniendiagramme*: Durch die Verbindung einzelner Punkte innerhalb eines Graphen, erhält man ein Liniendiagramm, welches mehrere Werte inklusive ihrem Trend durch nur einen Blick interpretieren lässt. Es besteht die Möglichkeit der Kombination von Punkt- und Liniendiagrammen um sowohl die Verteilung, als auch den Trend bestimmter Werte zu betrachten.
  - *Säulen- und Balkendiagramme*: Sie dienen dem quantitativen Vergleich einiger weniger Werte durch die Länge des Balkens/ der Säule. Durch die zweite Dimension (der Breite) werden die Verhältnisse prägnanter vermittelt. Jedoch eignen sich diese Art von Diagrammen nur für einen groben und keinen Detailwert-Vergleich.
  - *Boxdiagramme*: Ähnlich wie Säulendiagramme werden Boxdiagramme durch Rechtecke dargestellt. Im Gegensatz zu diesen, werden statt einzelner Werte ganze Wertbereiche durch Boxen präsentiert, welche zusätzlich einen Mittelwert (beispielsweise durch den Median) kodieren können.

- Boxplots: Eine Erweiterung der einfachen Boxdiagramme ist der Boxplot. Er stellt ebenfalls einen Wertebereich dar, welcher auf den verteilten Werte die Extremwerte, die beiden 25%- und 75%-Quantile und den Median angibt. Abbildung I.4.4 (S. 43), Bild 5 verdeutlicht diese Darstellung.

- Bulletgraphen: Der Bulletgraph ist eine von Stephen Few entwickelte Visualisierungsform, welches einen speziellen Wert innerhalb einer quantitativen Skala qualitativ bewertet und zusätzlich einen Vergleichswert, wie beispielsweise den Wert einer vorigen Messung oder einen vorgegebenen Richtwert, angibt. Dieser Graph eignet sich hervorragend dazu Werte anhand ihrer Zielerreichung und Einhaltungen bestimmter Qualitätsbereiche zu überprüfen. Bild 6 in Abbildung I.4.4 (S. 43) erläutert den Bulletgraph etwas genauer, weitere Erklärungen zu dieser Visualisierungsart finden sich auch in Kapitel 2.3.1 der Ausarbeitung von Frederic Evers [Eve12].

- Tabellen: Die Visualisierung einer Reihe von detaillierten, individuellen Werten ist mittels Graph nicht möglich. An dieser Stelle können Tabellen eingesetzt werden, um Verhältnisse zwischen diesen Werten darzustellen.

Ungeeignete Darstellungsformen

Neben den oben genannten Darstellungsarten, existieren auch Informationsdarstellungen, die weniger für die Verwendung innerhalb von Dashboards verwendet werden, folglich werden die in Abbildung I.4.5 (S. 44) dargestellten schlechten Diagrammtypen erläutert.

- Tachometer oder Thermometer: Diese Visualisierungsarten dienen der Darstellung von Werten innerhalb eines qualitativen Bereichs. Jedoch sind sie sehr schwer interpretierbar, wie das Beispiel des Thermometers (Bild 1 in Abbildung I.4.5) zeigt: werden die Werte entsprechend eines Thermometers angeordnet, so steigt der Wert in den roten (=warmen Bereich an). Jedoch wird innerhalb von Dashboards die Farbe Rot mit einer Warnung assoziiert, was die falsche Aussage zu dem steigenden Wert liefert. Tachometer haben neben dem Problem der Informationsdarstellung in Winkelflächen und der daraus folgenden schwierigeren Interpretation, ebenso einen hohen Platzverbrauch und könnten oftmals durch klar strukturierte Bulletgraphen platzsparend und leicht verständlich dargestellt werden.

- Kreisdiagramme: Neben Säulen- oder Balkendiagrammen, existiert eine weitere Möglichkeit Daten durch zwei Dimensionen zu codieren. Das Kreisdiagramm ist jedoch keine geeignete Darstellungsform für Werte innerhalb eines Dashboards, da die Interpretation der Werte sehr schwer ist. Der Mensch kann Winkelinformationen sehr schlecht verarbeiten, so dass zwar eine Verteilung von Werten erkannt wird, jedoch nicht ihr tatsächlicher Wert, da eine Umrechnung von der Wertmenge

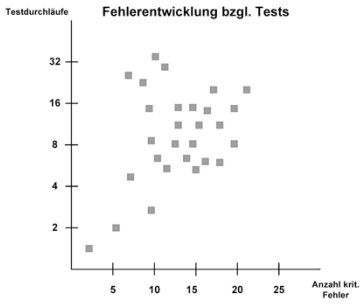

Bild 1: Punktdiagramm

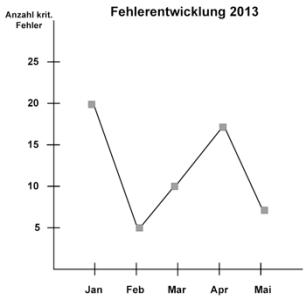

Bild 2: Liniendiagramm

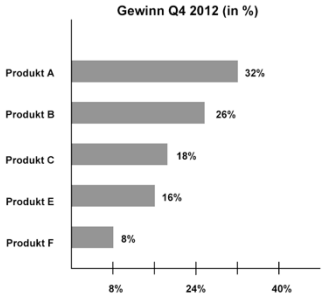

Bild 3: Balkendiagramm

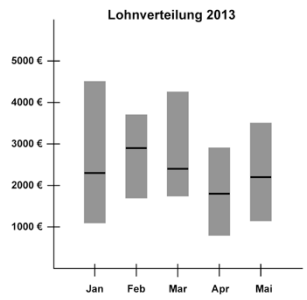

Bild 4: Boxdiagramm

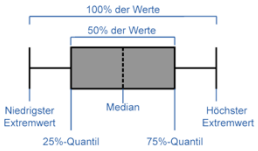

Bild 5: Boxplot

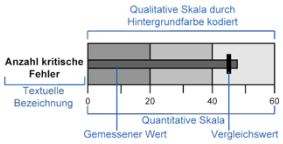

Bild 6: Bulletgraph

Abbildung I.4.4: Übersicht einer Auswahl geeigneter Dashboard-Widgets

in Winkelmengen geschehen muss. Oftmals lassen sich Daten, welche in Kreisdiagrammen dargestellt werden sollen besser in Balkendiagrammen präsentieren.

- 3D-Diagramme: Wie bei der Interpretation von Winkelinformationen, fällt es dem Menschen im Allgemeinen schwer Informationen, welche in drei Dimensionen kodiert wurden, zu interpretieren. So stellt sich die Frage, welche Bedeutung die einzelnen Dimensionen haben oder wie die einzelnen Werte anhand solcher Graphen abgelesen werden können.

Detaillierte Informationen zu den unterschiedlichen Diagrammtypen, wie auch dem richtigen Designansatz sind in *Show Me the Numbers: Designing Tables and Graphs to Enlighten* von Stephen Few [Few12] zu finden.

Bild 1: Thermometer

Bild 2: Tachometer

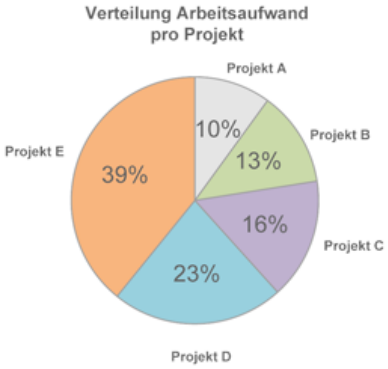

Bild 3: Kreisdiagramm

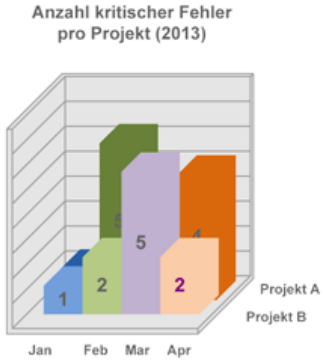

Bild 4: 3D-Säulendiagramm

Abbildung I.4.5: Übersicht ungeeigneter Dashboard-Widgets

I.4.2.1.3 Regeln des Dashboard-Designs

Anhand der Prinzipien der menschlichen Wahrnehmung, der Kapazität des Kurzzeitgedächtnisses und der optimalen Darstellungsformen stellte Stephen Few 13 Regeln für das Dashboard-Design auf, mithilfe dessen typische Fehler in der Gestaltung und der daraus folgenden Unbenutzbarkeit von Dashboards vermieden werden sollen. Die einzelnen Regeln werden in seinem Werk *Information Dashboard Design* [Few06], wie auch in der Ausarbeitung von Frederic Evers zum Thema *Konzeptionelle Erweiterung von Projektdashboards für unerfahrene Anwender* [Eve12] detailliert erläutert. An dieser Stelle wird eine Auswahl der wichtigsten Fakten kompakt dargestellt.

Prägnante Informationsvermittlung des Projekt- oder Prozessstatus ist die Hauptaufgabe eines Kennzahlen-Dashboards. Aus diesem Grund ist es wichtig alle Informationen *auf einem Bildschirm ohne weitere Navigationsmöglichkeiten*, wie beispielsweise dem Scrollen innerhalb der Ansicht oder dem Wechsel unterschiedlicher Ansichten, zu vereinen. Nur so können die Informationen auf einen Blick wahrgenommen und verglichen werden.

Ein niedriger Detaillierungsgrad fördert ebenfalls die direkte Erfassung der Informationen. So können zu viele beziehungsweise zu detaillierte Informationen den Zeitaufwand des Benutzers, alle Kennzahlen zu erfassen, deutlich erhöhen, was gegen das Prinzip des Dashboards (eine schnelle, effektive Möglichkeit den Projektstatus zu überblicken) wirkt.

Die Wahl der richtigen Dashboard-Widgets spielt ebenso eine große Rolle im Bereich des guten Dashboard-Designs. So soll bei der Konfiguration darauf geachtet werden, dass sowohl die richtigen Kennzahlen zu den entsprechenden Informationsbedürfnissen ermittelt werden, als auch die richtige Darstellungsformen (siehe Abschnitt I.4.2.1.2) gewählt werden. Hilfestellungen hierzu liefert das Buch *Show Me the Numbers: Designing Tables and Graphs to Enlighten* von Stephen Few [Few12].

Vermeidung von Dekorationen ist ein weiterer Punkt auf den der Dashboard-Designer achten sollte. So scheint ein Dashboard ohne Farben, Bilder und Logos auf den ersten Blick trist zu wirken, jedoch wird sich dies bei der täglichen Arbeit als Vorteil heraus stellen. Bilder, Rahmen, Logos und eine große Auswahl an Farben lenken den Fokus von den tatsächlichen Informationen ab und stören den täglichen Arbeitsfluss.

Die bedachte Auswahl an Farben sollte nicht vernachlässigt werden. Sie dienen in der Regel zur Markierung wichtiger Werte. Aus diesem Grund sollten sie äußerst sparsam verwendet werden. Müssen farbliche Abstufungen zur Unterscheidung (beispielsweise unterschiedlicher Balken innerhalb eines Diagramms) getätigt werden, so empfiehlt sich an dieser Stelle unterschiedliche Grautöne zu verwenden. Ist dies nicht möglich, so sollten Farben einer schwachen Intensität (helles grün, blau, braun etc.) verwendet werden. Vor allem die Kennzeichnung von Qualität durch die Ampelfarben Rot und Grün sollte aufgrund möglicher Benutzer mit Rot-Grün-Schwäche vermieden werden. Kräftige Farben (vor allem Rot) sollten nur zur Hervorhebung wichtiger Informationen, beispielsweise als Warnung für eine schlechte Kennzahlentwicklung, Verwendung finden. Positive Werte sollten hingehen nicht hervorgehoben werden, so stechen mögliche Probleme bei der ersten Betrachtung des Dashboards direkt ins Auge.

Die Anordnung der Dashboard-Items nach Wichtigkeit ist ein weiterer Punkt, der innerhalb des guten Designs eines Dashboards eine große Rolle spielt. So ist der erste Fokus des Betrachters meist die Mitte des Bildschirms, in der westlichen Welt orientiert sich die Reihenfolge der Wichtigkeit zusätzlich von links oben nach rechts unten (entsprechend der Lese- und Schreibrichtung). Unwichtigere Daten (wie Firmenlogos, nicht so relevante Dashboard-Widgets) sollten nicht an diese Positionen dargestellt werden, sondern unterhalb der Mittelposition beziehungsweise innerhalb der rechten Bildschirmseite auftreten.

Ein ansprechendes Design ist für die alltägliche Nutzung des Dashboards ebenfalls von Nöten. So werden „unattraktive graphische Displays“ [Few06] seltener genutzt als

ansprechende. Die obigen Regeln dienen der Erstellung ansprechender Dashboards und sollten aus diesem Grund berücksichtigt werden. Scheint ein Dashboard uninteressant für den Benutzer zu wirken, so sollten die dargestellten Informationen auf ihren Inhalt und ihre visuelle Darstellung überprüft werden, anstatt Dekorationen, Farben oder ähnliches hinzu zu fügen.

Die hier genannten Regeln spiegeln nur die für diese Arbeit wichtigsten Aspekte des Dashboard-Designs wider. Weitere Informationen im Bereich des Display-Designs sind in dem Standardwerk von Stephen Few [Few06], als auch in *Engineering Psychology and Human Performance* [WH00] von Wickens und Hollands zu finden.

I.4.2.2 *SCREEN*

Nachdem die Begrifflichkeiten um den Begriff des Dashboards geklärt wurden, soll an dieser Stelle das systemeigene Dashboard *SCREEN* der *Enterprice Measurement Infrastructure* vorgestellt werden.

Im Rahmen der Forschungsarbeit bezüglich des Metrik-System *EMI* wurden wie in Abschnitt I.4.1.1 vorgestellt diverse Komponenten entwickelt. Das Dashboard *SCREEN* dient der Visualisierung durch das System berechneter Kennzahlen und wurde nach den Design-Prinzipien von Stephen Few entwickelt (siehe Abschnitt I.4.2.1.3). Abbildung I.4.6 zeigt die prototypische Ansicht eines mit Widgets gefüllten Dashboards. So werden die Kennzahlen durch Dashboard-Items auf der linken Seite dargestellt. Die rechte Seite beinhaltet den Baum an Informationsbedürfnissen, welche die vorhandenen Widgets beantworten. Zusätzlich besteht die Möglichkeit die Ansicht (und somit auch die dargestellten Items) zu verändern und aus diesem Dashboard heraus vorhandene Daten beziehungsweise Graphiken innerhalb eines Reports zu exportieren.

Die in Abbildung I.4.6 dargestellte Ansicht ist ein Teil des *SCREEN* Dashboards. Ein weitere Teil ist die Konfiguration durch den Benutzer. Die Vorteile, die *SCREEN* bietet, sind unter anderem die Tatsache, dass das Dashboard beliebig durch einen Metrikexperten an die Organisation und die Bedürfnisse der Benutzer angepasst werden kann. So liefert es dem Benutzer einen Konfigurationsassistenzen, welcher anhand gegebener Rollenspezifikationen und Informationsbedürfnissen, den Benutzer durch die individuelle Konfiguration leitet. Dieser Konfigurationsassistent ist bislang prototypisch angedacht und wird mithilfe dieser Arbeit fachlich ausgearbeitet. Weitere Informationen hierzu sind auch im SSELab Kapitel des Prototyps zu entnehmen.

I.4.3 Weitere Arbeiten im Bereich des *EMI*-Systems

Wie bereits erwähnt stellt diese Arbeit ein fachliches Konzept der Integration eines Kennzahlen-Messsystems in bestehende Organisationsstrukturen dar. Weiterhin werden Vorlagen exemplarisch für Projektleiter ausgearbeitet, welche die Konfiguration eines für Kennzahlen-Dashboards vereinfachen sollen. Zeitgleich zu dieser Arbeit, wie auch bereits

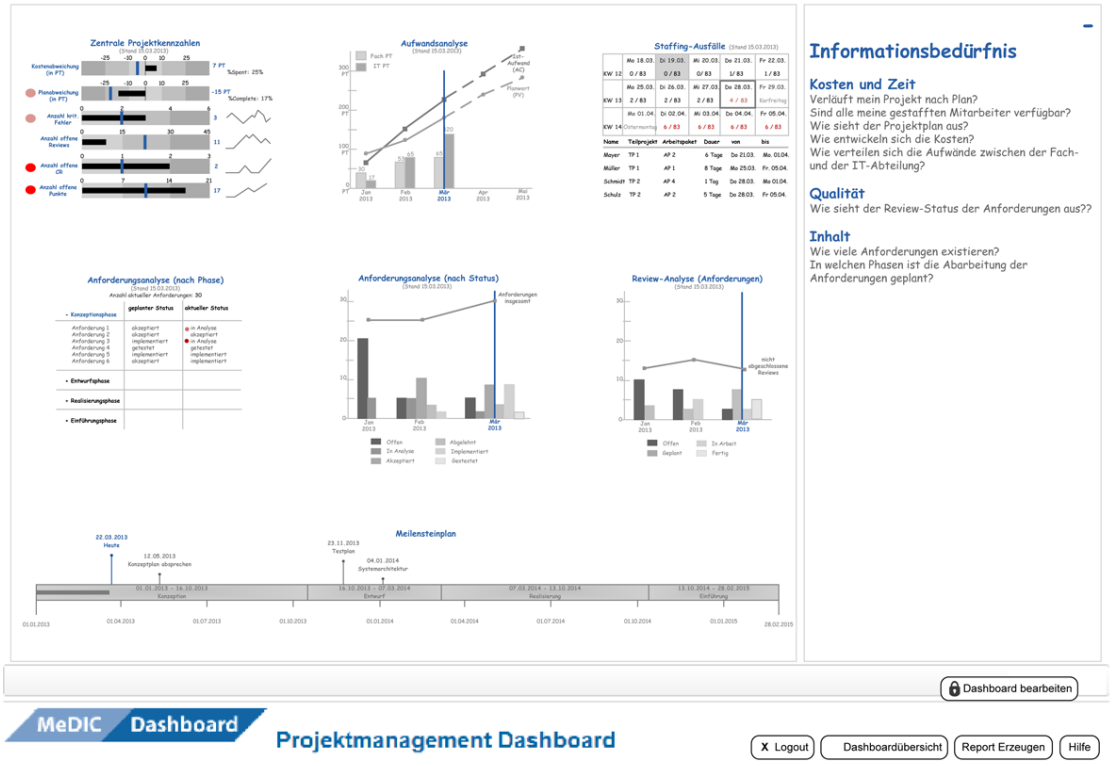

Abbildung I.4.6: Darstellung einer prototypischen Ansicht des *SCREEN*-Dashboards

davor, wurden einige weitere Arbeiten im Bereich des *EMI*-Systems fertiggestellt, von denen ein Auszug an dieser Stelle kompakt erläutert wird.

I.4.3.1 Prototyping Tool

Matthias Gora erstellte in seiner Bachelorarbeit [Gor13] ein Werkzeug, mit dessen Hilfe ein prototypisches Dashboard zum Abgleich mit dem Messkunden realisiert werden kann. Vor- und Nachteile eines solchen Werkzeugs, wird in Kapitel II.5 genauer evaluiert.

I.4.3.2 Variabilität

In seiner Diplomarbeit *Variabilität von Metriken und Dashboard-Items im Umfeld von MeDIC* [Mäd13] ergänzt Martin Mädler das *EMI*-System beziehungsweise -Dashboard um Konfigurationsmöglichkeiten bezüglich Variabilitäten. So können mithilfe dieser Ergänzung unterschiedliche Zeitintervalle, Datenmengen etc. durch den Benutzer eingestellt werden.

I.4.3.3 Folien-/Metrik-Export aus DB

Paul Tokarev erweitert das *SCREEN*-Dashboard um eine Exportkomponente, welche die vorhandene Kennzahl-Graphiken für den Einsatz in Teammeetings, Vorstandspräsentationen oder ähnliches in Präsentationsformate zur Weiterverwendung exportiert [Tok13].

I.4.3.4 Technische Integration

Im Anschluss an diese Ausarbeitung, wird Arthur Otto innerhalb seiner Bachelorarbeit [Ott13] das entwickelte Konzept mittels der technischen Integration in das SSELab (siehe Teil III) austesten und erweitern.