



Master Thesis

**Metrikenkonzepte von CASE-Tools
am Beispiel von Together**

von

Hashem Yazbek

Betreuer: Prof. Dr.-Ing. habil. R. Dumke

Magdeburg, im März 2007

Inhaltsverzeichnis

1. Einleitung	3
2. Einführung	5
2.1 Software Engineering	5
2.2 Software-Ressourcen	7
2.3 Software-Produkte	7
2.4 Der Software-Lebenszyklus	8
2.5 Fluggesellschaftssystem-Fallstudie	11
2.6 Management der Softwareentwicklung	12
2.7 Methoden & Technologien	14
2.8 Web Engineering	15
3. CASE-Tools	16
3.1 Die Klassifikation der CASE-Tools	17
3.2 Beispiele für CASE-Tools	19
3.3 CASE-Tools beim Web Engineering	22
4. Softwaremessung und –bewertung	24
4.1 Was sind Metriken?	25
4.2 Klassifikationen von Metriken.....	25
4.3 Beispiele für Metriken	28
4.4 Skalentypen	30
4.5 Darstellungsformen von Messwerten	33
4.6 Software-Messtools	34
4.7 Probleme bei der Auswahl und dem Einsatz eines Messtools	36
5. Metriken in Together	37
5.1 Das Dialogfeld Metriken	37
5.2 Kategorisierung der Together-Metriken	38
5.3 Wie werden Metriken verwendet?	39
5.4 Metrik-Ergebnisse	40
5.5 Eigenschaften und Charakteristiken der Metrik-Ergebnisse	41
5.5.1 Ergebnistabelle sortieren	41
5.5.2. Speichern und Laden der Metrik-Ergebnisse	41
5.5.3 Vergleichen von Metrik-Ergebnissen	41
5.5.4. Grafische Ansichten von Metrik-Ergebnisse	42
5.5.5 Dokumentation der Metrik-Ergebnisse	45
5.6 Metriken für die Verletzungen in Audits-Regeln	47
5.7 Eigene Metriken definieren	48
6. Bewertung des Messkonzeptes in Together	49
6.1 Allgemeine Aspekte der Metrikenanwendung in Together	49
6.2 Unterstützungsformen der Software-Entwicklung	51
7. Zusammenfassung und Ausblick	52
8. Literatur	53

Das industrielle Erstellen großer Software-Systeme ist ohne Computerunterstützung nicht mehr möglich.

Helmut Balzert/1998

1 Einleitung:

Software-Entwicklung, im Vergleich zu anderen Ingenieurdisziplinen, ist eine relativ junge Dienstleistung, die noch nicht in ausreichendem Maße über genügend Erfahrung verfügt, einen optimalen Prozess bei der Herstellung von Software-Produkten garantieren zu können. Anders als in anderen industriellen Disziplinen ist man hier von einem Konsens bezüglich der durchzuführenden Maßnahmen, ein qualitativ hochwertiges Produkt zu erstellen, weit entfernt.

Kunde und Hersteller stellen aber hohe Ansprüche an die Ergebnisse des Herstellungsprozesses. Der Kunde fordert ein funktionierendes Produkt, welches seine Bedürfnisse hinsichtlich einer automatisierten Abarbeitung seiner Betriebsabläufe in adäquater Weise erfüllt. Der Hersteller seinerseits versucht, das Produkt mit hoher Qualität in einem gegebenen Zeit- und Kostenrahmen zu liefern.

Die wesentlichen Komponenten dieser Definition sind **Qualität**, **Zeit** und **Geld**. Um innerhalb eines bestimmten **Kostenrahmens** zu bleiben, muss ein Manager schätzen können, welche Ressourcen benötigt werden, und sie dann in Form von Projektbeteiligten, Training und Werkzeugen bereitstellen. Um innerhalb des gesteckten **Zeitplans** bleiben zu können, muss der Manager zeitlichen Aufwand abschätzen und den Projektstatus kontrollieren können. Zum Erreichen der gewünschten **Qualität** braucht der Manager Mechanismen zur Berichterstattung von Fehlern und Techniken für das Risikomanagement. Es werden Komponenten ausgewählt und gemessen und Werte dann errechnet. Dies umfasst normalerweise das Durcharbeiten der Komponentendarstellung (Entwurf, Programmcode usw.) mit Hilfe eines automatischen Werkzeugs für die Datenerfassung. Diese könnte speziell erstellt werden oder bereits in die in der Organisation verwendeten CASE-Tools integriert sein.

Es ist ein wesentliches Ziel, das die Phasen des SE-Prozesses (Analyse, Entwurf, Implementierung, Test, Wartung) immer in Einheit mit dem Projektmanagement und Qualitätsmanagement gesehen werden.

Anliegen und Fokus der vorliegenden Arbeit sind die angewandten Metriken in CASE-Tools. Dabei dient die ISO 9000-3 zur Darlegung der Software-Qualitätssicherung. Ich habe ein CASE-Tools ausgewählt, nämlich das **Borland Together ControlCenter 6.2** (im Folgenden *nur als Together bezeichnet*), und habe eine kurze Einführung in Together gegeben und dann die von Together angebotenen Metriken beschrieben und bewertet. Diese Arbeit konzentriert sich auf die Darstellung der Metriken mit dem Schwerpunkt in der Anwendung der Metriken, und nicht in der Beschreibung der Bedienung des Tools.

Um dieses Ziel zu erreichen, besteht die Arbeit aus 5 Teilen. **Im zweiten Kapitel** wird eine Einführung in Software Engineering mit den Aspekten (Lebenszyklus, Ressourcen,

Management, Methoden) vermittelt. Hier habe ich auf das Wasserfall-Modell zur Darstellung des Lebenszyklusses beschränkt. Ich habe auch hier das Web Engineering, welches als Gebiet aus dem Software Engineering hervorgegangen ist, betrachtet.

Im dritten Kapitel habe ich den Begriff CASE mit seinen Aspekten (Methoden, Vorgehen, Tools) erläutert und Klassifikationen von CASE-Tools gegeben und dann vier auf dem Markt aktuellen CASE-Tools vorgestellt.

Im vierten Kapitel werden ebenfalls die Begriffe Softwaremessung und Metriken erläutert und dann einen Überblick über bekannte und weit akzeptierte Klassifikationen von Metriken gegeben. Hier habe ich auch drei Messtools vorgestellt und habe dann einige bei dem Auswahl und Einsatz eines Messtools bestehende Probleme angedeutet.

Das fünfte Kapitel stellt schließlich ausführlich alle von Together angebotenen Metriken und ihre Verwendung vor.

Im sechsten Kapitel werden diese Metriken bewertet bzw. die Anwendung von Softwaremetriken beim Together-CASE-Tool noch einmal allgemein charakterisiert.

Zur Veranschaulichung der Metriken und ihrer Ergebnissen habe ich an dieser Stelle ein *Fluggesellschaftssystem (FGS)* entwickelt. Das Projekt ist ein sehr stark vereinfachtes Beispiel, das trotzdem alle wichtigen Phasen und Strukturelemente eines realen Projektes enthält. Daher gibt es natürlich noch Lücken. Zum besseren Verständnis der Metriken wird das Projekt (FGS) diese Arbeit bis zum Ende begleiten. Der Projektplan und die aufbereiteten Projektdaten, die bei den Berechnungen und Grafiken als Grundlage verwendet wurden, sind im Anhang C zu finden.

2. Einführung:

Softwaresysteme sind komplexe Gebilde. Der Begriff *Software Engineering* wurde 1968 geprägt. Er war die Antwort auf den damals trostlosen Zustand bei der Entwicklung zuverlässiger Software in einem vernünftigen Zeitraum und innerhalb eines gesetzten Kostenrahmens. Betrachte ich einmal folgende Beispiele [Neumann 95]:

Verspätet und kostenüberschreitend

2002 überwache das Swanick Air Traffic Control-System den gesamten Überflugreiseverkehr über England und Wales. Das System war mit erheblicher Kostenüberschreitung (Kosten 623 Millionen Pfund, ursprünglich geplant waren 350 Millionen) und sechs Jahre zu spät geliefert worden. Zwei wesentliche Überarbeitungen des Systems wurden notwendig, nachdem das Training für Fluglotsen bereits begonnen hatte.

Rechtzeitige Lieferung

Nach 18 Monaten Entwicklungszeit wurde im Jahre 1984 einer Krankenversicherung in Wisconsin ein System für 200 Millionen Dollar geliefert. Aber das System arbeitet nicht korrekt und stellte mehr als 60 Millionen Dollar Überzahlungen aus. Die Verbesserung des Systems brauchte noch einmal drei Jahre.

Jeder der hier beschriebenen Fehler resultierte aus einem softwarebezogenen Problem. In diesen Fällen ergaben sich Systemfehler durch Managementfehler (späte und kostenüberschreitende Auslieferung, rechtzeitige Auslieferung eines falschen Systems). Es gibt natürlich noch andere Systemfehler. In einigen Fällen sahen Entwickler selten auftretende Situationen (jemand lebt länger als 100 Jahre, Schaltjahre wirken sich bei Verfallsdaten aus) nicht voraus. In anderen Fällen ahnten die Entwickler nicht, dass der Benutzer das System tatsächlich falsch benutzen konnte (Drücken eines Knopfes, Ausnutzen eines Fehlers in einem Softwareretzwerk). Und noch mehr. Was sind nun die wirklichen Probleme?

Im nächsten Abschnitt betrachte ich das Software Engineering von einer höhern Ebene aus. In Abschnitt 2.2 gebe ich einen Überblick über Software-Ressourcen und in Abschnitt 2.3 einen Überblick über das Software-Produkt. In Abschnitt 2.4 stelle ich ausführlich die Entwicklungsaktivitäten in der Software-Entwicklung vor. In Abschnitt 2.5 stelle ich eine Fallstudie vor, um die Entwicklungsaktivitäten in einem Zusammenhang mit einem Software-Projekt zu bringen. In Abschnitt 2.6 konzentriere ich mich auf das Planen und Steuern von Projekten und in Abschnitt 2.7 gebe ich einen Überblick über die Methoden der Software-Entwicklung.

2.1 Software Engineering:

Es existiert eine ganz Menge an Definitionen zum Begriff Software Engineering. An dieser Stelle möchte ich nur einige wenige stellvertretend für viele andere zitieren. Eine sehr ausführliche Definition bietet Helmut Balzert für den Begriff Softwaretechnik als Synonym für Software Engineering an [Balzert 98]:

„Softwaretechnik“: Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden, Konzepten, Notation und

Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresysteme. Zielorientiert bedeutet dies die Berücksichtigung z.B. von Kosten, Zeit und Qualität.

Das IEEE (Institute of Electrical and Electronic Engineers) schlägt eine sehr knappe, aber exakte Definition vor [IEEE 90]:

Software Engineering: “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”

Nach diesen Definitionen sind also folgende Aspekte bei der Software-Entwicklung und -Anwendung zu betrachten:

- **Methoden:** Richtlinien, Strategien und Technologien für eine systematische, d.h. phasen- oder schrittweise Entwicklung von Software,
- **Werkzeuge:** rechnergestützte Hilfsmittel zur Software-Entwicklung und -Anwendung,
- **Maßsystem:** Menge von Software-Maßen zur Bewertung und Messung der Eigenschaften der zu entwickelnden Software hinsichtlich Eignung, Qualität und speziell des Leistungsverhaltens,
- **Standards:** Menge von Richtlinien für die einheitliche und abgestimmte Form der Software-Entwicklung und des zu entwickelnden Software-Systems,
- **Erfahrungen:** (quantifizierte) Kenntnisse über die Entwicklung der Software sowie dem Entwicklungsergebnis selbst hinsichtlich deren Einsatz, der Qualität und des Nutzens (als Ingenieurwissen).

Diese Prinzipien schließen auch die Personifizierung (als **Software-Ingenieur**) und die Integration in eine Software-Ingenieurgemeinschaft (**Community**) ein. Die folgende Abbildung visualisiert die grundlegenden Aspekte beim Software Engineering.

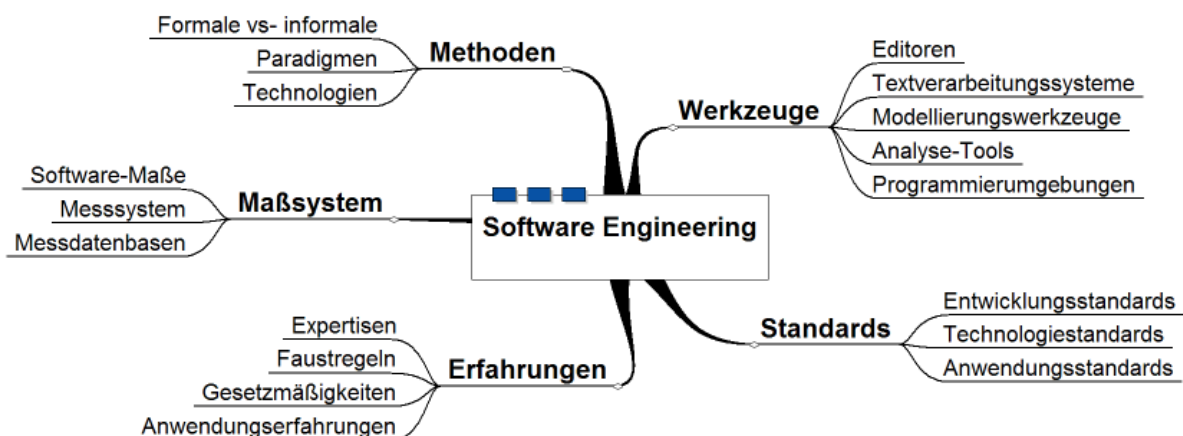


Abb.1 Aspekte des Software Engineering [Dumke,03]

Ein *Projekt* zur Entwicklung eines Softwaresystems setzt sich aus einer Reihe von Aktivitäten zusammen. Jede *Aktivität* wiederum setzt sich aus einer Reihe von Aufgaben zusammen. Eine *Aufgabe* verbraucht Ressourcen und erzeugt ein Produkt. Ein *Produkt* kann entweder ein

System, ein Modell oder ein Dokument sein. *Ressourcen* sind entweder Teilnehmer oder Ausrüstung.

Auf all diese Konzepte möchte ich jetzt kurz eingehen.

2.2 Software-Ressourcen:

Software-Ressourcen (kurz Ressourcen) sind Vermögenswerte, die zur Bewältigung der Arbeit verwendet werden. Ressourcen umfassen Mitarbeiter, Soft- und Hardware-Mittel. Nach [Dumke 03] gilt:

*„Als **Ressourcen** (resources) fassen wir alle für die Software-Entwicklung aufgewendeten bzw. aufzuwendenden personellen, Software- und Hardware-Mittel zusammen.“*

Die *personellen Ressourcen* sind alle am Projekt beteiligten Personen (*Teilnehmer* auch genannt), beispielsweise Kunde, Benutzer, Entwickler, Analysator, Designer, Programmierer oder Manager. Jeder Teilnehmer hat eine bestimmte Rolle (*Aufgabe*). Der Kunde bestellt und bezahlt das System. Die Entwickler konstruieren das System. Der Projektmanager plant und budgetiert das Projekt und koordiniert die Entwickler und den Kunden. Die Endbenutzer werden vom System unterstützt. Derselbe Teilnehmer kann natürlich verschiedene Rollen gleichzeitig wahrnehmen.

Die *Software-Ressourcen* werden als **CASE-Tools** bezeichnet, die ich im nächsten Abschnitt ausführlicher behandeln will.

Die *Hardware-Ressourcen* werden allgemein als **Plattform** bezeichnet und bestimmen die zugrunde liegende (hardwarenahe) **Systemsoftware**, wie das Betriebssystem, die Netz- oder die Steuerungssoftware.

Bei der Planung eines Projekts zerlegt ein Manager die Arbeit in Aufgaben und teilt diesen Ressourcen zu.

2.3 Software-Produkte:

Ein Software-Produkt ist nicht nur eine Menge von Programmen oder ein einzelnes sehr umfangreiches ausführbares Programm. Es gehören eigentlich alle Dokumente der Software-Erstellung dazu, die für den Entwickler, den Betreiber bzw. den eigentlichen Nutzer erstellt wurden.

Definiert wird das Software-Produkt wie folgt [Dumke 03]:

*„Ein **Software-Produkt** ist die Gesamtheit von Softwarekomponenten (Programmen, Dokumentationen usw.), die als Ganzes entwickelt, vertrieben, angewendet und gewartet werden.“*

Ein Software-Produkt hat im Allgemeinen folgende Bestandteile oder Komponenten

- die *Anwendungsbeschreibung (User manual)*, die die Handhabung der Software einschließlich der organisatorischen und technischen Voraussetzungen beschreibt. Umfangreiche Produkte verfügen weiterhin über ein so genanntes *Referenz-Handbuch (reference manual)*, welches eine Übersicht über die Komponenten in lexikografischer Form gibt, bereitgestellt bzw. angefertigt,
- die eigentlichen *Programme*, die zur Umsetzung der Funktionalität in verschiedenen Formen vorhanden sind (als unmittelbar ausführbaren Code (*object code*), als so genannten Quellcode (*source code*), als Masken (*forms*), als so genannte Ablaufkommandos (*scripts*) oder als im Allgemeinen externe Ad-hoc-Komponenten (*plug ins*),
- das *Installationsprogramm (Setup)* zur Einrichtung aller code-basierten Produktbestandteile auf einem konkreten Computer bzw. konkreten Plattform,
- die *Entwicklerdokumentation*, die alle im Verlauf der Produktentwicklung entstandenen Dokumente, wie Skizzen, Diagramme, Prüflisten, Testdateien, Hilfsprogramme, Qualitätsberichte, Änderungsanleitungen usw., zusammenfasst,
- ein *Demonstrationsprogramm*, das den potentiellen Anwender über den Leistungsumfang und die Handhabung informiert.

Ein Beispiel für ein Software-Produkt und deren Komponenten bzw. Bestandteile soll im Folgenden skizziert werden. Das Software-Entwicklungswerkzeug ***Borland Together***:

- *Anwendungsbeschreibung*: Durch die schnelle Verbreitung dieser Technik existieren auch hierbei bereits eine Reihe von Büchern sowie die Anwendungsmöglichkeit einer Online-Dokumentation bei der Tool-Anwendung.
- *Programme*: Die dafür erforderlichen Programme und Dokumentations-Files sind auf der Homepage von Borland als Demo-Version „herunterladbar“. Diese Demo-Version ist zeitlich begrenzt nutzbar und besitzt Einschränkungen hinsichtlich des Leistungsumfangs. Natürlich ist auch die „Vollversion“ allerdings bei entsprechender Bezahlung erhältlich.
- *Installationsprogramm*: Die Installation wird mittels des Setups vorgenommen.
- *Demonstrationsprogramm*: Insbesondere aus Marketing-Gründen wurden zahlreiche Multimediasysteme zur Erläuterung und Motivation für die UML-Technik im Allgemeinen und das Together-Tool in Besonderen entwickelt.

2.4 Der Software-Lebenszyklus:

Wie alle anderen technischen Artefakte unterliegt auch Software einem ***Software-Lebenszyklus (software lifecycle)***. Sie wird geplant, entworfen, hergestellt, in Betrieb genommen, benutzt und gepflegt, und schließlich abgelöst und außer Dienst gestellt. Abbildung 2 zeigt schematisch den Software-Lebenszyklus.

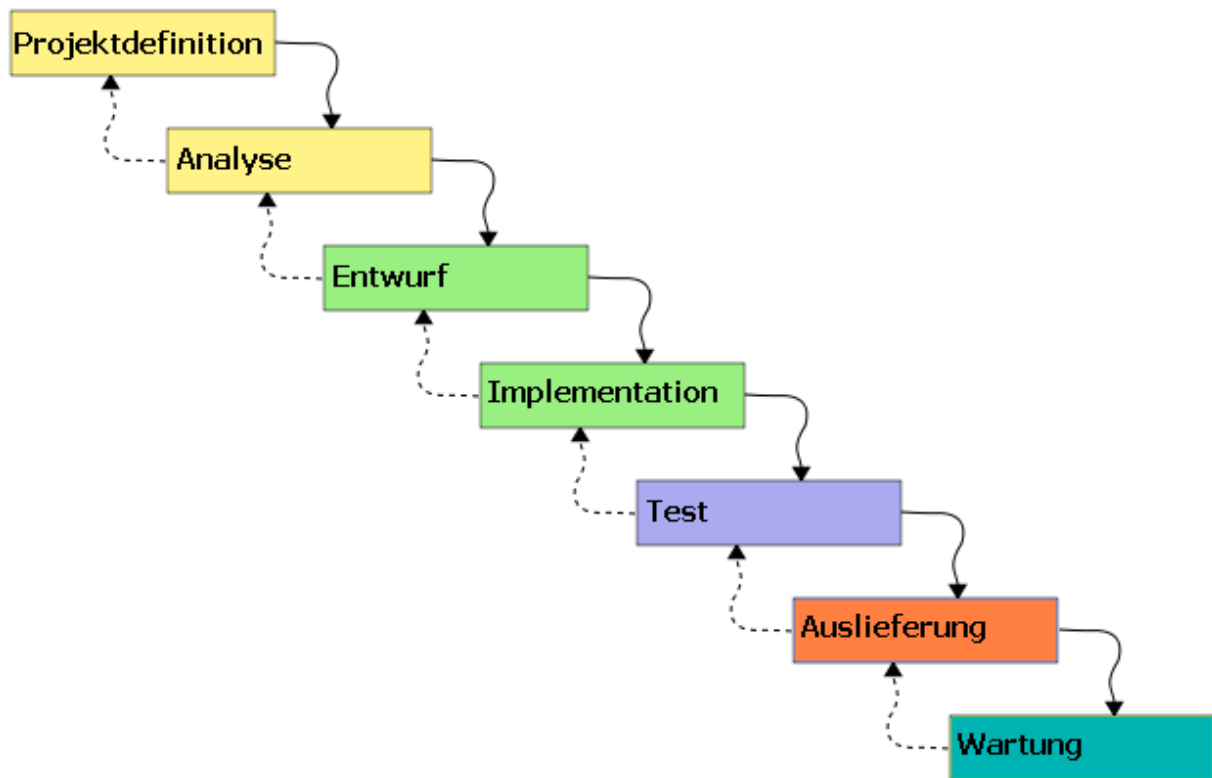


Abb.2 Phasen im Software-Lebenszyklus

Dieser Lebenszyklus ist z.B. in der Norm [ISO 12207 (1995)] definiert und heute in dieser Form weitgehend akzeptiert. In [Dumke 03] wird der Software-Lebenszyklus wie folgt definiert:

*„Der **Software-Lebenszyklus** (software life cycle) ist der Prozess der Entwicklung von Software-Produkten und kennzeichnet alle Phasen und Stadien dieser Produkte von ihrer Entwicklung, Einführung und Wartung bis zu ihrer Ablösung oder Beseitigung.“*

Die Vorgehensweise zur Herstellung von Software nennt man Entwicklungsprozess oder Softwareprozess, der in folgender Weise definiert wird [Dumke 03]:

*„Der **Software-Entwicklungsprozess** (software development process) ist der gesamte Prozess der Aufgabenstellung, Planung, Realisierung und Bewertung einer Software-/Hardware-Anwendung einschließlich der verwendeten Hilfsmittel und Methoden und dem erforderlichen Personal.“*

Die Abschnitte des Lebenszyklus bzw. einer Softwareprozess nennt man häufig **Phasen**, ein Begriff der ursprünglich auf das sog. Wasserfall-Modell zurückgeht. Daher lautet eine Definition beispielsweise nach [Dumke 03] wie folgt:

*„Eine **Lebenszyklus-Phase** (life cycle phase) innerhalb des Software-Entwicklungs-, Anwendungs- und Wartungsprozesses ist ein zeitlich begrenzter Abschnitt mit relativ eigenständigen Ressourcen, für den eine Anfangssituation und ein bewertbarer Endzustand bestimmt werden können.“*

Zum besseren Verständnis wird zunächst eine statische Sicht auf die Entwicklung eines Software-Produktes zugrunde gelegt und werden diese Phasen in allgemeiner und chronologischer Form beschrieben:

- **Problemdefinition:** Zuerst werden in Gesprächen mit dem Kunden die Anforderungen an das System ermittelt.
- **Anforderungsanalyse:** Zunächst werden die Anforderungen hinsichtlich ihrer Realisierbarkeit untersucht und in einem Systemmodell beschrieben. Diese Phase dient zur Vorbereitung aller nachfolgenden Phasen und schließt mit einer groben Projektplanung für die Realisierung ab.
- **Spezifikation:** Hier wird das zu implementierende Produkt oder System in seiner Funktionsweise beschrieben. Im Allgemeinen werden Anforderungsanalyse und Spezifikation innerhalb einer Phase realisiert.
- **Entwurf:** Während des Entwurfs werden alle technischen Vorbereitungen und Planungen getroffen, welche für eine fehlerfreie und rasche Implementierung des Systems notwendig sind. Parallel zum Entwurf wird auch der Testplan erstellt, welcher einerseits die Erfüllung der Anforderungen und andererseits die korrekte Funktionsweise der Implementierung überprüfen soll.
- **Implementation (implementation):** Die Implementierung dient zur Umsetzung des Entwurfs in einer konkreten Programmiersprache und der Erstellung eines lauffähigen und lieferbaren Systems.
- **Test (auch Akzeptanztest):** Zum Abschluss werden die angestrebte Korrektheit des Systems in Bezug auf Anforderungen und Technik sichergestellt und gefundene Fehler vor der Auslieferung an den Kunden beseitigt.
- Die **Auslieferung** schließt die Entwicklung mit der Übergabe aller anwendungsspezifischen Produktkomponenten an den oder die Auftraggeber bzw. Nutzer ab.
- **Wartung:** Die Software wird in Betrieb genommen. Dabei werden Fehler und Lücken in den ursprünglichen Anforderungen entdeckt. Es tauchen Programm- und Entwurfsfehler auf und der Bedarf an neuen Funktionen wird festgestellt. Das System muss sich also weiterentwickeln, um nützlich zu bleiben. Das kann zur Folge haben, dass einige oder alle der vorherigen Prozessphasen wiederholt werden müssen.

Jede Phase sollte nicht beginnen, bevor nicht die vorherige abgeschlossen wurde. In der Praxis überlappen sich die Phasen und tauschen Informationen untereinander aus. Während des Entwurfs werden Probleme mit den Anforderungen entdeckt, während des Programmierens fallen Fehler im Entwurf auf usw. Der Softwareprozess ist nicht einfach ein linearer Prozess, sondern besteht aus einer Reihe sich wiederholender Entwicklungsaktivitäten.

Neben den oben beschriebenen Software-Entwicklungs-Phasen müssen noch zwei weitere Tätigkeiten hervorgehoben werden, und zwar die **Dokumentation** und die **Qualitätssicherung**. Die dazu notwendigen Tätigkeiten bilden keine eigentlichen Projektphasen, sondern müssen projektbegleitend, d.h. in allen Phasen durchgeführt werden.

Die **Dokumentation** soll während der Entwicklungsphasen die Kommunikation zwischen den an der Entwicklung beteiligten Personen ermöglichen und nach Abschluss der

Entwicklungsphasen den Einsatz und die Wartung von Softwareprodukten unterstützen. Sie soll außerdem den Projektverlauf zum Zwecke der Kalkulation der Herstellungskosten und zur besseren Planung zukünftiger Projekte dokumentieren.

Die *Qualitätssicherung* umfasst analytische, konstruktive und organisatorische Maßnahmen zur Qualitätsplanung und zur Erreichung von Qualitätsmerkmalen wie Korrektheit, Zuverlässigkeit, Benutzerfreundlichkeit, Wartungsfreundlichkeit, Effizienz und Portabilität.

2.5 Fluggesellschaftssystem-Fallstudie

Die oben genannten Lebenszyklusphasen werden jetzt an einem ganz einfachen Beispiel kurz erläutert. Es soll jetzt das eben erwähnte *Fluggesellschaftssystem (FGS)* entwickelt werden. Damit ergibt sich ganz kurz skizziert die folgenden möglichen Phaseninhalte gruppiert nach Entwicklung, Wartung und Anwendung.

Software-Entwicklung:

- *Problemdefinition:* Die Problemdefinition ist bereits teilweise durch den Kunden erfolgt. Wie schon beschrieben, bei FGS geht es um die Buchungsnachfrage, die eigentliche Vermittlungstätigkeit und die Flugorganisation. Die erste Aufgabe im FGS ist es, den Systemkontext zu beschreiben, also das Umfeld mit den bestehenden und fehlenden teilen, die Beteiligten bzw. Betroffenen. Die Arbeitsergebnisse dieser Aufgabe dienen zum einen der persönlichen Orientierung, zum anderen auch der Kommunikation sowohl innerhalb des Teams als auch mit dem Kunden, die so früh wie möglich eingebunden werden sollten.
- *Anforderungsanalyse:* Es geht hier darum festzustellen, was im Detail eigentlich die Aufgabe ist. Es ist außerdem zu prüfen, welche ressourcenbedingten Einschränkungen ein derartiger Computer für ein FGS mit sich bringt (Speicherung, Codeeingabe und -anzeige). Es wird weitere Informationen über ähnliche Projekte gesammelt, z.B. durch Interviews mit erfahrenen Anwendern. Die Mitschriften solcher Interviews werden als strukturierten Text aufbereitet. Aufgrund dieser Unterlagen werden die Geschäftsprozesse identifiziert und jeder einzelne Geschäftsprozess im Detail beschrieben. Die Prozesse werden zu Nutzfällen verfeinert, die ebenfalls in Inventaren und Tabellen beschrieben werden können. Hier ist auch die Hauptinformationsquelle wieder der Kunde.
- *Spezifikation:* Hier wird die Funktionalität des gesamten FGS ausformuliert und die korrekte Umsetzung mit dem Auftraggeber abgestimmt. Eine Anwenderdokumentation wird hier ebenfalls entworfen.
- *Entwurf:* Hier wird die Software-Architektur geklärt und wird dann festgelegt, welche Subsysteme und Komponenten das System haben soll. Die Aufgaben werden dann auf die Teilprojekte, Auftragnehmer und gegebenenfalls auf Partnerfirmen verteilt. Außerdem werden die Schnittstellen und Verbinder zwischen den Subsystemen und Komponenten und zu Nachbarsystemen festgelegt. Eine spezielle Schnittstelle ist die Schnittstelle zum Benutzer, insbesondere die Abläufe von GUI-Dialogen. Schließlich wird der Feinaufbau der Komponenten beschrieben.
- *Implementation:* Alle fachlichen Fragen sind jetzt (zumindest theoretisch) geklärt, sodass man sich auf die technischen Details der Umsetzung konzentrieren kann. Hier wird programmiert und die einzelnen Systemfunktionen getestet. Heute können Code

ganz oder teilweise automatisch erzeugt werden. Die Entwicklungsdokumentation wird auch in dieser Phase gebildet.

- *Test*: Die Implementierten Subsysteme von FGS werden integriert, also zu lauffähigen Einheiten zusammengefügt. Das FGS wird dann von einem potentiellen Nutzer angewandt und bewertet. Mängel werden unmittelbar behoben.
- *Auslieferung*: Die zu installierenden Programme und die Anwenderdokumentation werden an die Nutzer bzw. Auftraggeber geschickt.
- *Wartung*: Die Erstellung von FGS ist abgeschlossen und beginnt nun die Nutzung der Applikation. Der Entwickler verfolgt jetzt das FGS in seiner Anwendung (Beim Nutzer) und wartet es. Aus dem laufenden Betrieb des FGS ergeben sich dann ständig neue Anforderungen, die umgesetzt werden. Es werden zum Beispiel Fehler gefunden, die behoben werden sollen, und gesetzliche Änderungen erfordern neue Anpassungen. Die Veränderungen am Markt machen auch neue Funktionen nötig. Allerdings kann auch die Änderung der Computer-Plattform ein Grund für derartige Änderungen sein.

Nachdem die Änderungen, Verbesserungen und Umstellungen am FGS realisiert wurden, sind diese Änderungen dem Auftraggeber zur Verfügung zu stellen, damit er das sich auf einer Plattform befindliche FGS aktualisieren bzw. korrigieren kann.

Beim Einsatz und der Wartung von FGS sollte auch auf Folgendes geachtet werden:

- Das Erstellen von **Dokumentationen** und **Schulungsunterlagen** sowie die Schulung der Benutzer des FGS müssen frühzeitig geplant und vorbereitet werden.
- Die **Ablösung** eines Altsystems, inklusive der **Migration** von Bestandteilen muss organisiert werden.
- Die **Beschaffung** und **Installation** von Hardware und Software, die **Verteilung** der Applikation auf der Zielpattform und gegebenenfalls die Eingliederung in einen laufenden Rechenzentrumsbetrieb erfordern erhebliche Vorbereitungen.

2.6 Management der Softwareentwicklung

In diesem Abschnitt beschreibe ich kurz die Aktivitäten, die beim Verwalten eines Software-Projekts auftreten. Managementaktivitäten konzentrieren sich auf das Planen des Projekts, das Überwachen seines Status, das Verfolgen von Änderungen und das Koordinieren der Ressourcen, sodass ein qualitativ hochwertiges Produkt zur rechten Zeit und innerhalb des Kostenrahmens geliefert wird. Managementaktivitäten involvieren nicht nur die Manager, sondern auch die meisten anderen Projektteilnehmer.

Es gibt keine allgemeingültige Beschreibung der Managementaktivitäten. Die Tätigkeiten verändern sich in Abhängigkeit von dem Unternehmen und dem zu entwickelnden Software-Produkt nämlich ganz erheblich. Hier beschreibe ich nur einige Tätigkeiten, die für meine Arbeit wichtig sind. Sie umfassen:

- **Projektplanung**: Die Projektplanung konzentriert sich auf die Definition des Problems, die Planung einer Lösung und die Abschätzung von Ressourcen. Ein Plan muss erstellt werden, um die Entwicklung zum Projektziel zu führen. Der Plan sollte während des Projekts überarbeitet werden, wenn das Projekt fortschreitet und bessere Informationen auftauchen.

- **Zeitplanung:** Es wird ein Zeitplan für das Projekt erstellt. Manager schätzen die benötigte Zeit und die benötigten Ressourcen zum Abschließen der Aufgaben ab und bringen diese in eine zusammenhängende Abfolge. Zeitpläne müssen auch aktualisiert werden, wenn bessere Informationen über den Fortschritt verfügbar werden.
- **Personalmanagement:** Projektmanager müssen normalerweise die fähigen Menschen auswählen, die an dem Projekt arbeiten sollen, und sie dann in Arbeitsgruppen organisieren. Projektmanager müssen ihre Mitarbeiter motivieren, deren Arbeit planen und organisieren sowie sicherstellen, dass die Arbeit gut gemacht wird.
- **Aufwand- und Kostenschätzung:** Diese Schätzungen sind möglicherweise zum Festlegen eines Projektbudgets oder zur Ermittlung des Softwarepreises für einen Kunden erforderlich. Diese Schätzungen sollten ebenfalls regelmäßig im Laufe des Projekts aktualisiert werden. Die zu berücksichtigenden Faktoren sind z.B. Hardware-, Software-, Reise-, Schulungs-, Personalkosten. Es gibt keinen einfachen Weg, die Kosten genau zu schätzen. Aber das COCOMO-Modell wurde dazu empfohlen [Sommerville 07].
- **Qualitätsmanagement:** Der Ablauf der Softwareentwicklung wird von Manager überwacht, um zu gewährleisten, dass ein qualitativ hochwertiges Produkt ausgeliefert wird, denn es ist nicht akzeptabel, Probleme und Mängel nach der Auslieferung an den Kunden durch Reparaturen zu beheben. Dazu werden z.B. Tests, Reviews, Audits und Messen eingesetzt. Ihre Durchführung ist nicht auf einer bestimmten Entwicklungsphase beschränkt, sondern in allen Phasen des Projektverlaufs. Ich erörtere das Thema Messen in Kapitel 4 genauer.
- **Konfigurationsmanagement:** Die Änderungen in den Software-Produkten werden von Manager überwacht und steuert. Änderungen durchziehen die gesamte Softwareentwicklung. Anforderungen ändern sich, wenn der Kunde neue Fähigkeiten verlangt und oft auch, wenn die Entwickler ihr Verständnis des Anwendungsbereichs verbessert haben. Die Hardware- und Softwareplattform, auf der das Produkt aufgebaut ist, kann sich ändern, wenn neue Technologien verfügbar werden. Das Produkt kann sich ändern, wenn während der Testphase Fehler entdeckt und verbessert werden. Dabei wird das System als eine Anzahl von Konfigurationselementen repräsentiert, die unabhängig voneinander überarbeitet werden. Für jedes Konfigurationselement wird die Entwicklung als eine Reihe von Versionen verfolgt. Das Auswählen einer bestimmten Version gestattet es dem Entwickler, zu einem wohldefinierten Status des Systems zurückzugehen, wenn eine Änderung fehlschlägt.
- **Wartungsmanagement:** Manager versuchen vorherzusagen, welche Systemänderungen vermutlich beantragt, welche Teile des Systems dem Wartungspersonal wahrscheinlich die meisten Schwierigkeiten bereiten und wie hoch die Wartungskosten für ein System innerhalb eines bestimmten Zeitraum sein werden. Die Wartungsaktivitäten werden dann geplant, organisiert, und ihre Durchführung kontrolliert.

Ausführliche Informationen über Aktivitäten und Methoden beim Management sind in [Dumke 03] und [Sommerville 07] zu finden.

2.7 Methoden & Technologien

Methoden sind organisierte Vorgehensweisen, um kostengünstige und qualitativ hochwertige Software herzustellen. Eine Methode soll allgemein Personen helfen, bestimmte Tätigkeiten zu verrichten. Grundsätzlich fordert man von einer Methode, dass sie die Dokumentation der Arbeitsergebnisse vereinheitlichen, eine systematische und zielstrebige Vorgehensweise bietet und nach einer Anzahl planbarer Schritte zum Ergebnis der Aufgabe führt.

Alle Methoden beruhen auf der Vorstellung, Modelle eines Systems zu entwickeln, die sich grafisch darstellen lassen, und diese Modelle als Systemspezifikation oder -entwurf zu verwenden. Eine Methode umfasst:

- **Modelle**, um die Konzepte und inhaltliches Wissen über ein Problem und dessen Lösung sammeln zu können.
- spezielle **Notationen** aus Symbolen, Komponenten oder mathematischen Kalkülen, um die Modelle geeignet darstellen zu können.
- Ein iteratives **Vorgehen** für die Konstruktion der Modelle und deren Implementierung.
- **Vorschläge** und **Regeln** für die Erstellung der Modelle.

Eine Auflistung verschiedener Software-Entwicklungsmethoden zeigt die Abbildung 3.

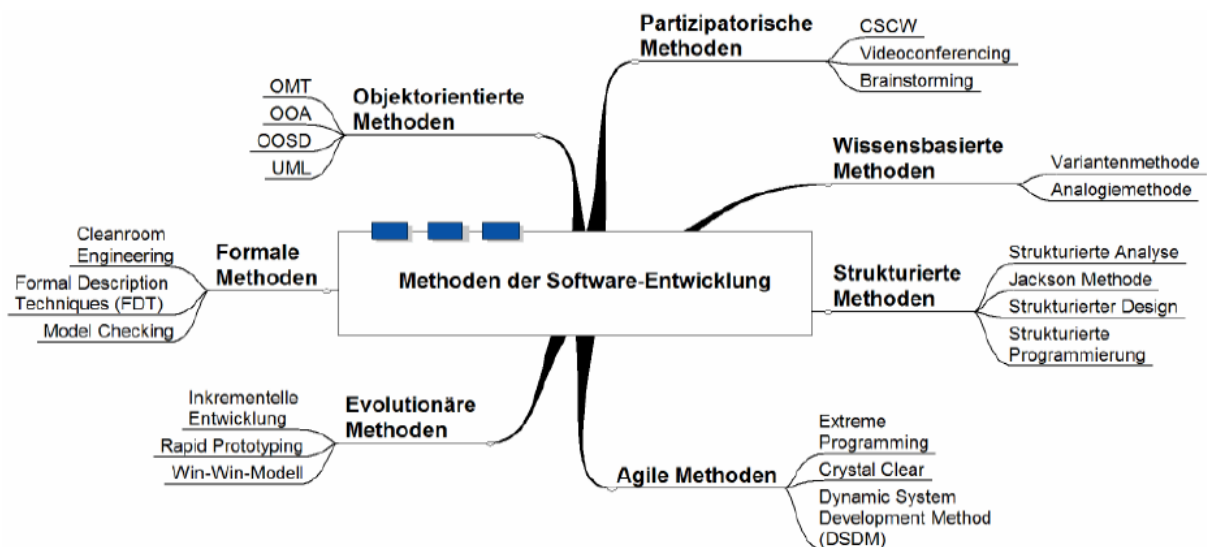


Abb. 3 Ausgewählte Methoden der Software-Entwicklung [Dumke,03]

Ich stelle im Rahmen dieser Arbeit nur eine Methode kurz vor, und zwar die objektorientierte Methode. Mitte der 80er Jahre begann man objektorientierte Methoden zu entwickeln und einzusetzen. Die ersten objektorientierten Methoden entstanden aus dem Bedarf der konzeptionellen Unterstützung der objektorientierten Programmierung. Nach und nach wurden zahlreiche objektorientierte Methoden entwickelt, wie z.B. CRC, OOSA, HOOD, OOSD, OMT und OOSE. Trotz der unterschiedlichen Notationen und Vorgehen basieren alle objektorientierten Methoden auf denselben Konzepten. Die **UML (Unified Modeling Language)** wurde dann entstanden und definiert eine Standardnotation, die für alle objektorientierten Methoden benutzt werden kann.

2.8 Web Engineering

Ein relativ neues, aus dem Software Engineering hervorgegangenes Gebiet ist das **Web Engineering**, das auf den Erkenntnissen von Software Engineering aufbaut und sich über den gesamten Lebenszyklus eines Websystems erstreckt. In [DLWZ 03] ist Web Engineering wie folgt definiert:

„Web Engineering ist die methodenbasierte, werkzeugunterstützte, quantifizierte, standardgerechte, erfahrungsnutzende und Community-bezogene Entwicklung und Wartung von webbasierten Softwaresystemen.“

Web Engineering überträgt die **Methoden** des Software Engineering auf die Entwicklung von webbasierten Softwaresystemen und erstreckt sich über den gesamten **Lebenszyklus** eines Websystems. Es werden standardisierte ingenieurmäßige **Prozesse** entwickelt, die die Unterschiede zur klassischen Software Engineering berücksichtigen.

In der Regel ist Web Engineering nicht nur die Entwicklung, sondern auch die Fortentwicklung und Erweiterung von vormals erstellten Websites.

Web Engineering unterscheidet sich in einigen Aspekten von Software Engineering, wie die Nutzung von Hypertexte, TCP/IP-Protokoll, HTTP, HTML, CSS, JavaScript, XML, SQL u.a. Die Entwicklung von webbasierten Softwaresystemen hat im Vergleich zu der Entwicklung von Software-Produkte einen relativ kürzeren **Lebenszyklus**.

3. CASE-Tools

Hier möchte ich drauf hinweisen, dass ich Tool und Werkzeug als Synonyme verwende, was von manchen Quellen anders interpretiert wird.

Die Abkürzung *CASE* steht für *Computer-Aided Software Engineering* (computer-unterstütztes Software Engineering) und bedeutet wörtlich, dass ein Computer bei der ingenieurmäßigen Planung, Entwicklung und Wartung von Software-Produkten hilft. Der Begriff selbst ist zwar erst zu Beginn der zweiten Hälfte der 80er Jahren entstanden, jedoch Hilfsmittel zur Erstellung von Software gibt es bereits seit Beginn der elektronischen Datenverarbeitung, wie z.B. Assembler, Linker und Lader.

CASE besteht im Zusammenspiel der drei Komponenten *Vorgehen*, *Methoden* und *Werkzeuge (CASE-Tools)*.

Unter *Vorgehen* versteht man einen Standardablauf, der an die Projektart und -größe angepasst werden kann. Die Vorgehensbeschreibung ist somit eine Beschreibung des Softwareentwicklungsprozesses, eine Vorgabe, die die wesentlichen Schritte und Ergebnisse beschreibt, die zur Erstellung eines Systems notwendig sind. Sie dient hauptsächlich dem Projektmanagement und der Ausbildung der Entwickler, und sie ist die Grundlage für die stetige Verbesserung des Entwicklungsprozesses.

Eine *Methode* ist eine planmäßig angewandte, begründete Vorgehensanweisungen zur Erreichung eines festgelegten Ziels (z.B. bessere Qualität). Methoden zeigen dem Anwender den Weg auf, wie er sein Ziel erreichen kann, und vermeiden nutzloses Herumprobieren. Im weitern dienen sie auch der Erfahrungssicherung: Sie enthalten das Wissen und die Erfahrungen, die bei der früheren Erarbeitung der gleichen Ergebnisse gewonnen wurden.

Werkzeuge (CASE-Tools) bezeichnen Software-Systeme, also Programme, die benutzt werden, um die Herstellung, Prüfung, Wartung und Dokumentation von Software-Produkte zu vereinfachen und beschleunigen oder in ihrer Qualität zu verbessern. Zu den CASE-Tools zählen daher Modellierungsprogramme, Entwurfseditoren, Data Dictionaries, Compiler, Debugger, Tools für die Systemerstellung usw. Die meisten Methoden können nur mit Hilfe von Werkzeugen zur Lösung echter Probleme eingesetzt werden; ohne die Werkzeuge funktionieren sie nur für kleine Schulbeispiele.

Erst diese drei Komponenten zusammen ermöglichen Computer Aided Software Engineering. Lässt man eine Komponente weg, kann CASE nicht funktionieren. Insbesondere können Werkzeuge das Vorgehen oder die Methoden nicht ersetzen!

CASE-Tools helfen die Qualität insofern zu verbessern, indem sie einerseits das Entstehen von Fehlern unterbinden und andererseits die Anwendung der Standards, Methoden und Notationen vereinfachen und unterstützen. CASE-Tools automatisieren einige Abläufe im Software-Entwicklungsprozess, wie beispielsweise:

- Die Entwicklung grafischer Systemmodelle als Teil des Softwareentwurfs.
- Die Benutzung eines *Data Dictionary* mit Informationen über die Objekte und Beziehungen in einem Entwurf, um das Verständnis des Entwurfs zu verbessern.
- Die Erzeugung von Benutzeroberflächen durch eine grafische Oberflächenbeschreibung, die interaktiv vom Benutzer erstellt wird.

- Die Behebung von Programmfehlern mit Hilfe von dargestellten Informationen über ein laufendes Programm.
- Die automatische Übersetzung von Programmen von einer alten Version einer Programmiersprache in eine neuere Version.

Die CASE-Tools können auch einen Codegenerator enthalten, der automatisch Code aus dem Systemmodell erzeugt, sowie eine gewisse Anleitung zum Vorgehen, die dem Softwareentwickler vorschlägt, was als Nächstes zu tun ist.

Es gibt eine große Anzahl von CASE-Tools auf dem Markt und sie werden von einer ganzen Anzahl von Herstellern angeboten. Hier möchte ich einen Überblick mit Beispielen von einigen Klassifikationen für CASE-Tools geben.

3.1 Die Klassifikation der CASE-Tools

CASE-Klassifizierungen helfen uns, die verschiedenen Typen von CASE-Tools und ihre Rolle bei der Unterstützung von Aktivitäten der Software-Entwicklung zu verstehen. Es gibt verschiedene Möglichkeiten zur Klassifizierung von CASE-Tools, wobei jede diese Tools von einem anderen Standpunkt aus betrachtet [Sommerville,07]. In diesem Abschnitt werde ich CASE-Tools danach unterteilen, welche Phasen der Software-Entwicklung sie unterstützen. Ich werde sie in zwei Kategorien unterteilen, *Phasenbezogene* und *Projektbegleitende* CASE-Tools.

Phasenbezogene CASE-Tools

- Analyse (Text- und Graphik-Editoren, ...)
- Entwurf (Modellierungs-Tools, ...)
- Implementierung (Programmierungsumgebungen, Prototyping, Generatoren, ...)
- Test (Testdatengeneratoren, Analysatoren, ...)
- Wartung (Performance-Tools, Re-Engineering-Tools (*CARE-Tools*), ...)

Projektbegleitende CASE-Tools

- Projektmanagement (Schätzung, Planung, ...)
- Qualitätssicherung (Debugger, Compiler, *CAME-Tools*, ...)
- Dokumentation
- Konfigurationsmanagement (Projektbibliotheken, Versionskontrolle, ...)

Die phasenbezogenen CASE-Tools für die so genannten frühen Phasen der Software-Entwicklung, also von der Problemdefinition bis zum Entwurf, werden als *Upper CASE-Tools* bezeichnet. Die phasenbezogenen Werkzeuge für die unteren bzw. „späten“ Entwicklungsphasen, also die Implementation mit den verschiedenen Testformen, werden als *Lower CASE-Tools* bezeichnet. Die CASE-Tools können dabei innerhalb einer Entwicklungsphase angewendet werden bzw. alle Aufgaben einer Entwicklungsphase abdecken (Abbildung 3-1).

Ein typisches Beispiel für ein Upper CASE-Tool wäre ein *Textverarbeitungssystem*, das die Problemstellung bzw. weitere Entwicklungsdokumente erfasst. Ein Beispiel zum Lower CASE-Tool ist eine *Java-Programmierungsumgebung*.

Realisieren phasenbezogene CASE-Tools die Aufgaben mehrerer Entwicklungsphasen, so werden sie integrierte CASE-Tools, oder kurz *I-CASE-Tool* genannt. Sie umfassen im Allgemeinen verschiedene Werkzeuge, die auf eine bestimmte Art integriert sind. Sie stellen die Infrastruktur zur Integration von Daten, Steuerung und Darstellung bereit.

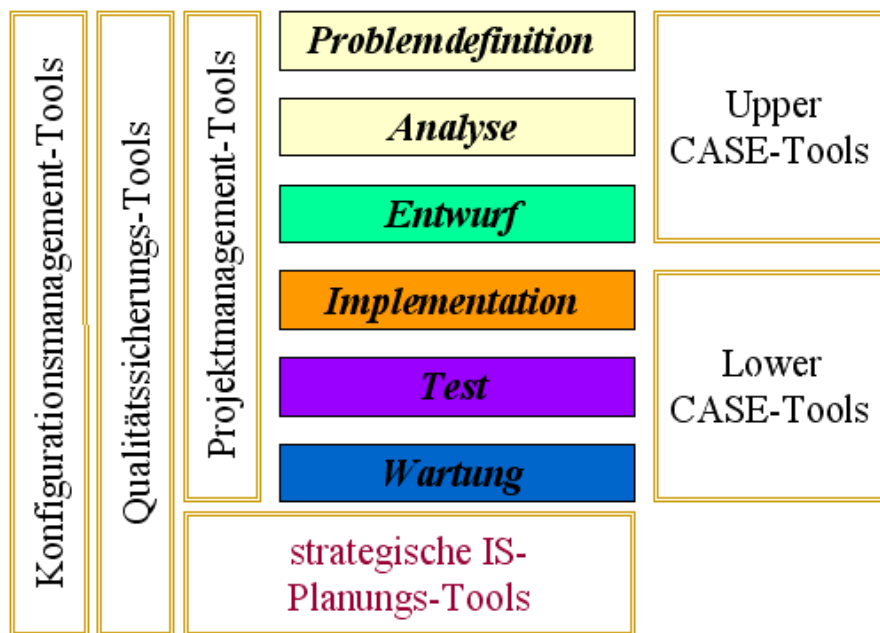


Abb. 3-1 CASE-Tools bei der Software-Entwicklung

Die *strategischen Informationssystem-Planungswerkzeuge* dienen zur unternehmerischen Führung der Informatik und der unternehmensweiten Planung und Verwaltung von Informationssystemen im erweiterten Sinne. Sie werden nur für diese Klasse von Applikationen verwendet [Wallmüller 01]. Eine kleine Gruppe von Spezialisten benutzt sie für die Geschäftsplanung (inklusive der Unternehmenszielplanung), Ressourcenplanung, Technologieplanung und deren Übersichtsverwaltung, sowie Informationsarchitekturplanung und -verwaltung. Ein Beispiel für diese Kategorie von Werkzeugen sind die U5 Tools [U5Tools 2006].

Eine weitere Klassifikation zu CASE-Tools ergibt sich aus ihrer Leistungsfähigkeit, die sich im Laufe der Zeit ergeben hat, und zwar als *CASE-Tool-Generationen* [Dumke 03]. So spricht man beispielsweise von CASE-Tools erster Generation, wenn diese nur jeweils eine Entwicklungsphase abdecken, sonst nennt man sie CASE-Tools zweiter Generation.

Re-Engineering-Tools sind speziell für die Wartung charakterisiert. Sie leiten aus dem Programmcode den dazugehörigen Entwurf bzw. die Dokumentation ab. Re-Engineering-Tools werden deshalb auch als *CARE-Tools* (*computer-aided reengineering tools*) bezeichnet.

CASE-Tools, die wiederum der Messung und Bewertung von Software-Komponenten bzw. zur Bewertung der Software-Entwicklung selbst dienen, sind unter der Bezeichnung *CAME-Tools* (*computer-assisted measurement and evaluation tools*) zusammengefasst [DFKW 96].

CASE-Tools, die die Analyse, Entwicklung und Wartung von webbasierten Softwaresystemen unterstützen, werden als *C*AWE*-Tools* (*Computer Aided Web Engineering*) bezeichnet.

3.2 Beispiele für CASE-Tools

Beispiele über Phasenbezogene CASE-Tools findet man in [Dumke 03] und [Pomberger 93]. Im Folgenden werde ich Beispiele für aktuelle I-CASE-Tools angeben. Diese CASE-Tools sollen den ganzen oder zumindest einen großen Teil des Software-Lebenszyklusses unterstützen. Sie sollen helfen, Fehler frühzeitig zu erkennen und zu korrigieren, die Dokumentation der in den einzelnen Phasen entstandenen Produkte zu unterstützen und den Herstellungsprozess zu rationalisieren.

Visual Paradigm Suite for UML v4.1 (VP-UML)

VP-UML ist ein führender Hersteller von Modellierungs- und Entwicklungswerkzeugen. Es ist ein UML CASE-Tool und unterstützt alle Phasen der Softwareentwicklung von der Analyse bis zur Implementation. Das VP-UML wurde für verschiedenen Anwender gedacht, wie Software Ingenieure, System Analysten, Business Analysten, Systemarchitekten und andere, die ihre Systeme mit objektorientierter Methode entwickeln. Die Anwender können durch Drag & Drop aller verfügbaren UML-Diagramme erstellen.

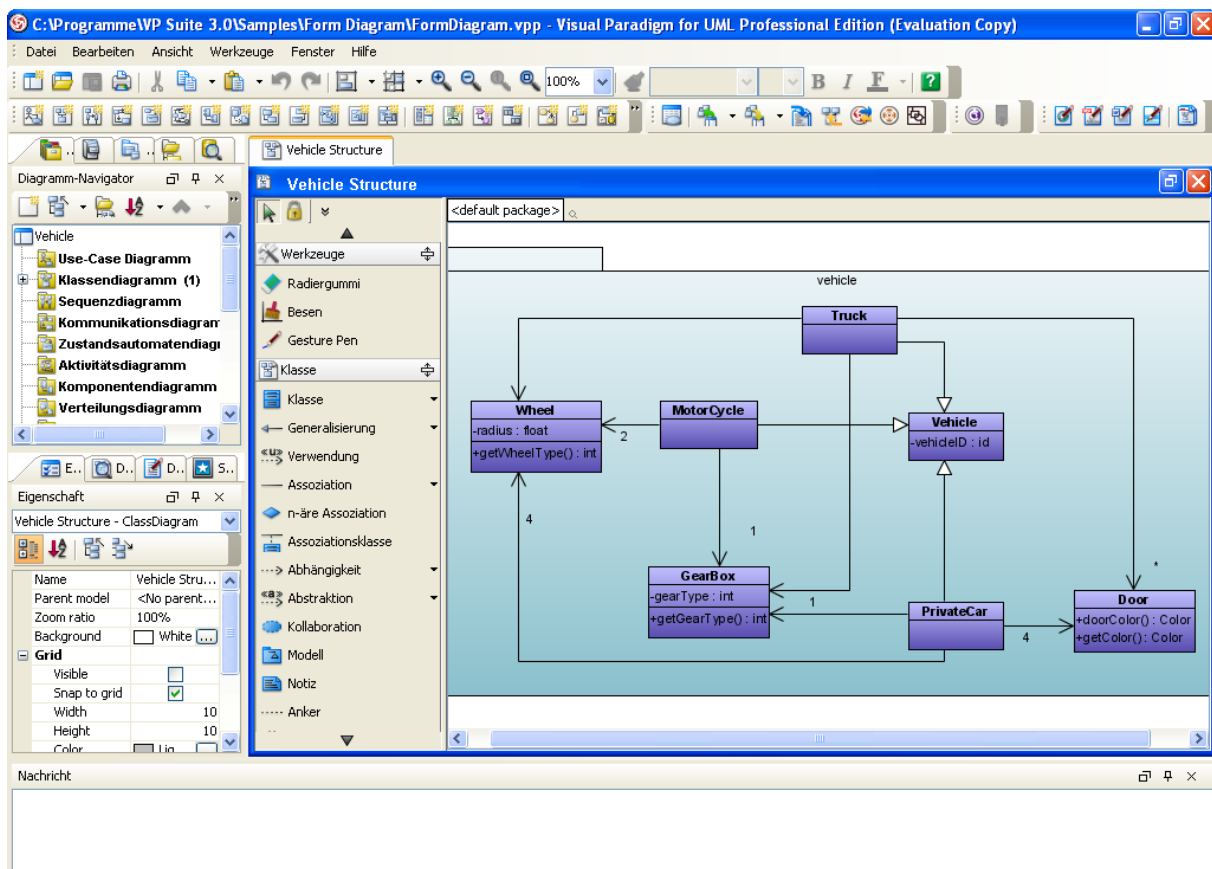


Abb.3-2 Visual Paradigm Suite for UML Layout

VP-UML verbindet UML-Konformität mit wichtigen Arbeitstechniken wie textueller Analyse oder der Modellierung relationaler Datenbanken. Um das Zusammenwirken mit anderen Anwendungen zu unterstützen, wird der Export/Import im XML-Format unterstützt. Anwender sind so in der Lage mit minimalen Aufwand Daten in ihre Anwendungen zu übernehmen.

VP-UML unterstützt die aktuellen Standards der UML-Notation und von Java. Es bietet den vollständigen "Round-Trip-Engineering"-Ansatz durch die Unterstützung von Code-Generierung und Reverse Engineering für Java. VP-UML ist auf Windows, Linux/Unix und Mac verfügbar und ist auch auf Deutsch erhältlich. Mehr Informationen über VP-UML findet man in [VPUML 2006].

Metamill v4.2

Ein CASE-Tool für die UML-basierte Modellierung. Es wurde in C++ geschrieben und unterstützt das Round-Trip-Engineering für Java, C++, ANSI C und C#. Das Tool unterstützt alle UML 2.0 Diagramme, einschließlich der Verteilungsdiagramme. Klassen und Beziehungen können durch Drag & Drop erzeugt werden.

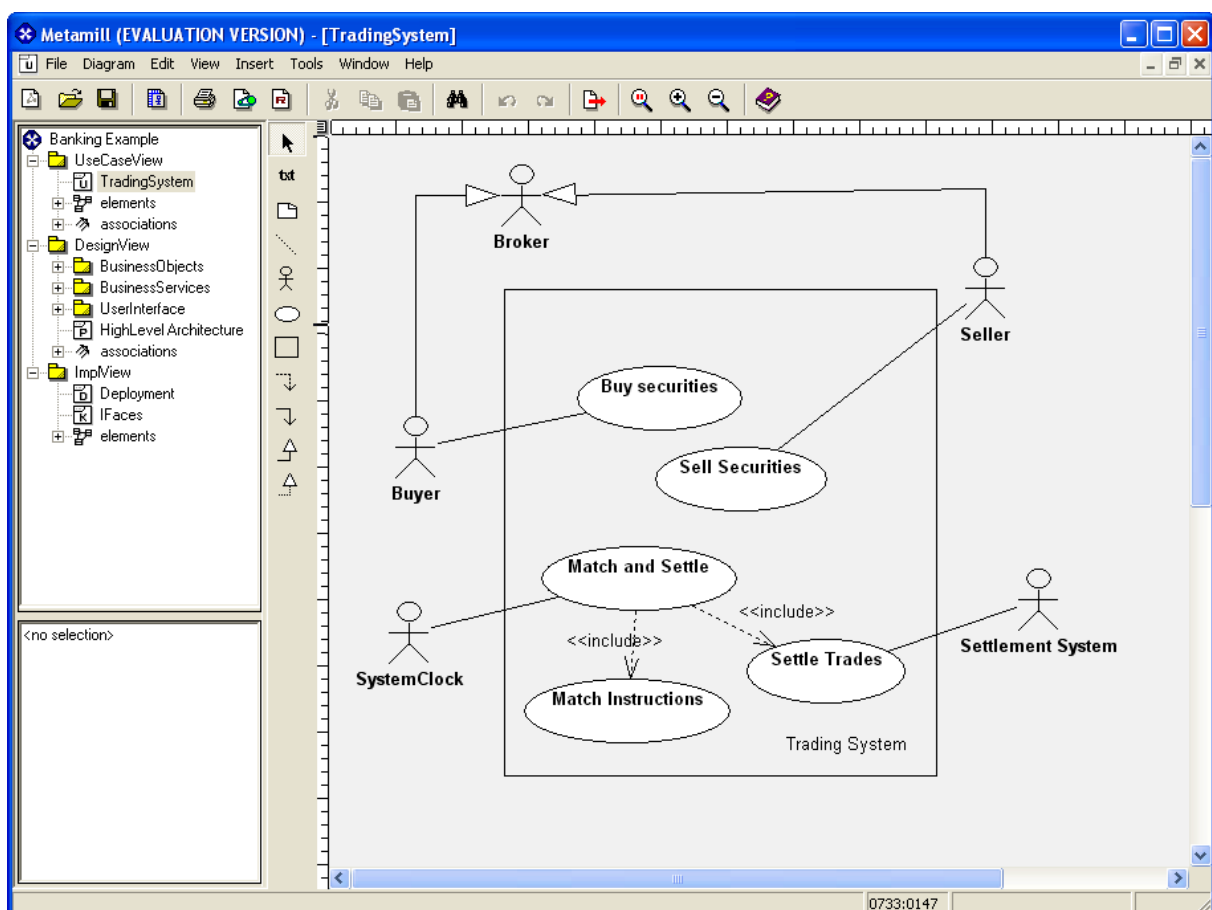


Abb.3-3 Metamill Layout

HTML-Dokumente können aus den Modellen erzeugt werden. Paketen können als XMI-Dateien importiert/ exportiert werden, sowie Diagramme, die in anderem Tool erzeugt werden, können hier als .MDL File importiert werden. Mehr Informationen über Metamill findet man in [Metamill 2006].

Together

Together ist ein integriertes CASE-Tool, was für die kompletten Phasen der Software-Entwicklung und -Wartung entwickelt wurde. Dazu stehen zahlreiche Funktionen zum erstellen von UML Diagrammen, Editieren & Kompilieren von Quellcode, Debugger, GUI-Designer, Audits, Metriken, Versionskontrolle, Workflowmanagement und andere Funktionalitäten zur Verfügung. In Together werden auch die Dokumentationen für den Entwickler und den Anwender abgeschlossen.

Together zählt wahrscheinlich zu den führenden CASE-Tools, die das Round-Trip-Engineering unterstützt. Round-Trip-Engineering ist eine Technik, was selbst aus zwei Funktionen besteht:

- Forward Engineering: automatische Erzeugung von Quellcode aus dem Model
- Reverse Engineering: automatische Generierung vom Model aus Quellcode

Wenn ein Tool beide Funktionen bietet, dann spricht man von Round-Trip-Engineering.

In Together wird in Projekten gearbeitet. Beim Erstellen eines neuen Projektes hat man die Möglichkeit, den Quellcode in einer aus sechs verschiedenen Programmiersprachen zu schreiben, nämlich aus *Java*, *C ++*, *CORBA IDL*, *VisualBasic.Net*, *VisualBasic 6* und *C #*. Zunächst erscheint die Together-GUI, die auch Workspace genannt wird (Abbildung 3-4).

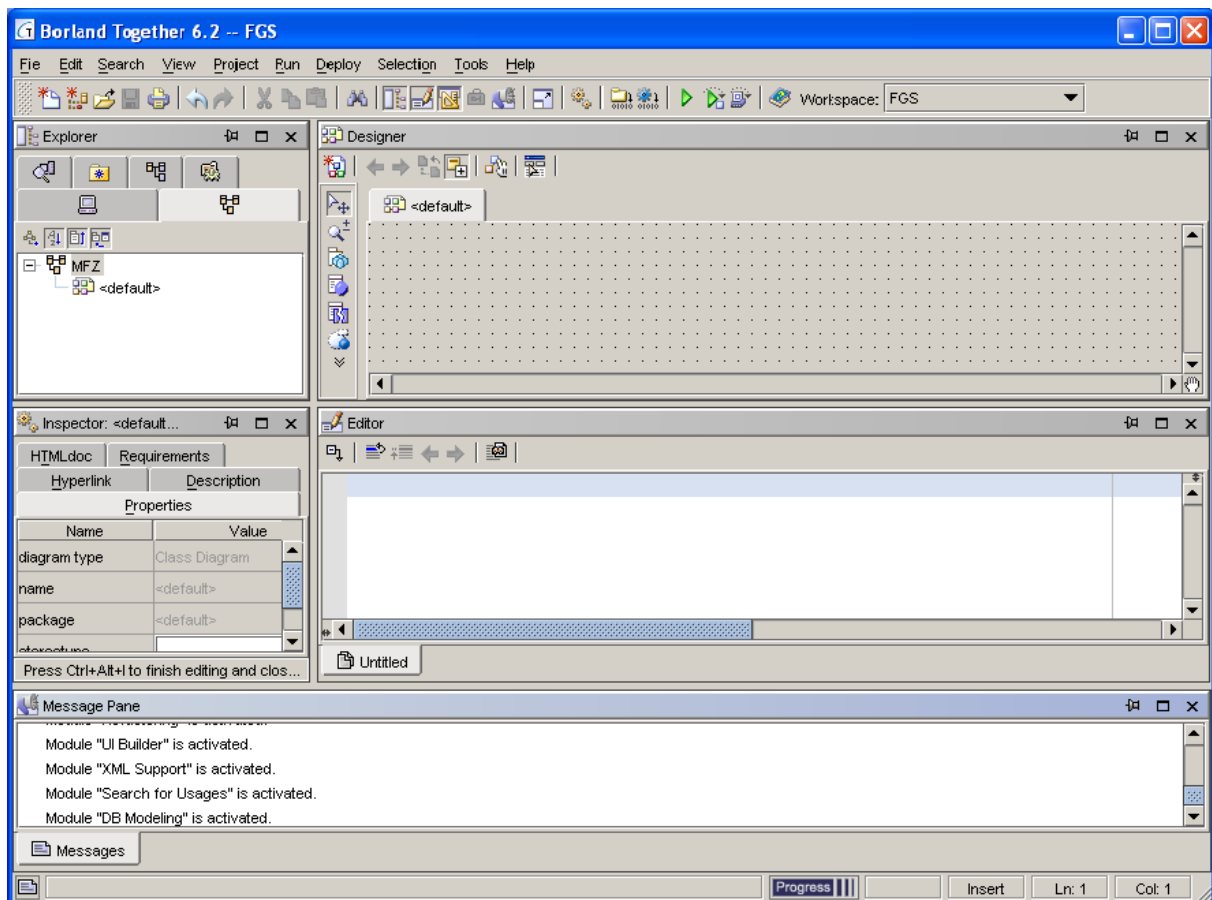


Abb.3-4 Together-Workspace

Wie Abbildung 3-4 zu entnehmen ist, befindet sich fünf Fenster unter der Hauptmenüleiste und der Hauptsymbolleiste:

- *Explorer*-Fenster: dient dazu, das Navigieren im System und im Projekt zu erleichtern.
- *Design*-Fenster: dient dazu, Diagramme zu erstellen.
- *Inspector*-Fenster: zeigt die Eigenschaften des selektierten Design-Elements an und gibt die Möglichkeit, sie zu ändern.
- *Editor*-Fenster: dient dazu, das Quellcode anzuzeigen und zu editieren.
- *Message*-Fenster: dient dazu, Fehler im Quellcode hinzuweisen und die Metrik-Ergebnisse anzuzeigen.

Together verfügt außerdem über einen GUI-Designer, der auch beim Arbeiten als Fenster angezeigt wird [Together 2006].

3.3 CASE-Tools beim Web Engineering

Für eine Websystem-Entwicklung benötigt man im Allgemeinen eine Reihe von CASE-Tools, die ich bereits als *CAWE-Tools* erwähnt habe. Die Anzahl und Vielfältigkeit von CAWE-Tools ist sehr umfangreich (einige Beispiele sind in [Dumke et al, 03a] und unter <http://www.w3.org/Tools> zu finden.

Eine Klassifikation für CAWE-Tools kann wie folgt angegeben werden [DLWZ 03]:

- CASE für dokumentenbasierte Websysteme
 - Websystem-Entwicklungs-Tools,
 - Websystem-Analyse-Tools;
- CASE für operationale Websysteme.

Ich werde mich hier nur auf ein aktuelles Beispiel für ein Websystem-Entwicklungs-Tool beschränken.

Delphi for PHP (D4PHP)

Es ist ein Web-Entwicklungsumgebung von der Borlandtochter CodeGear. Delphi für PHP ist ein I-CASE-Tools und enthält ein Designer, ein PHP-Debugger und ein Code-Editor.

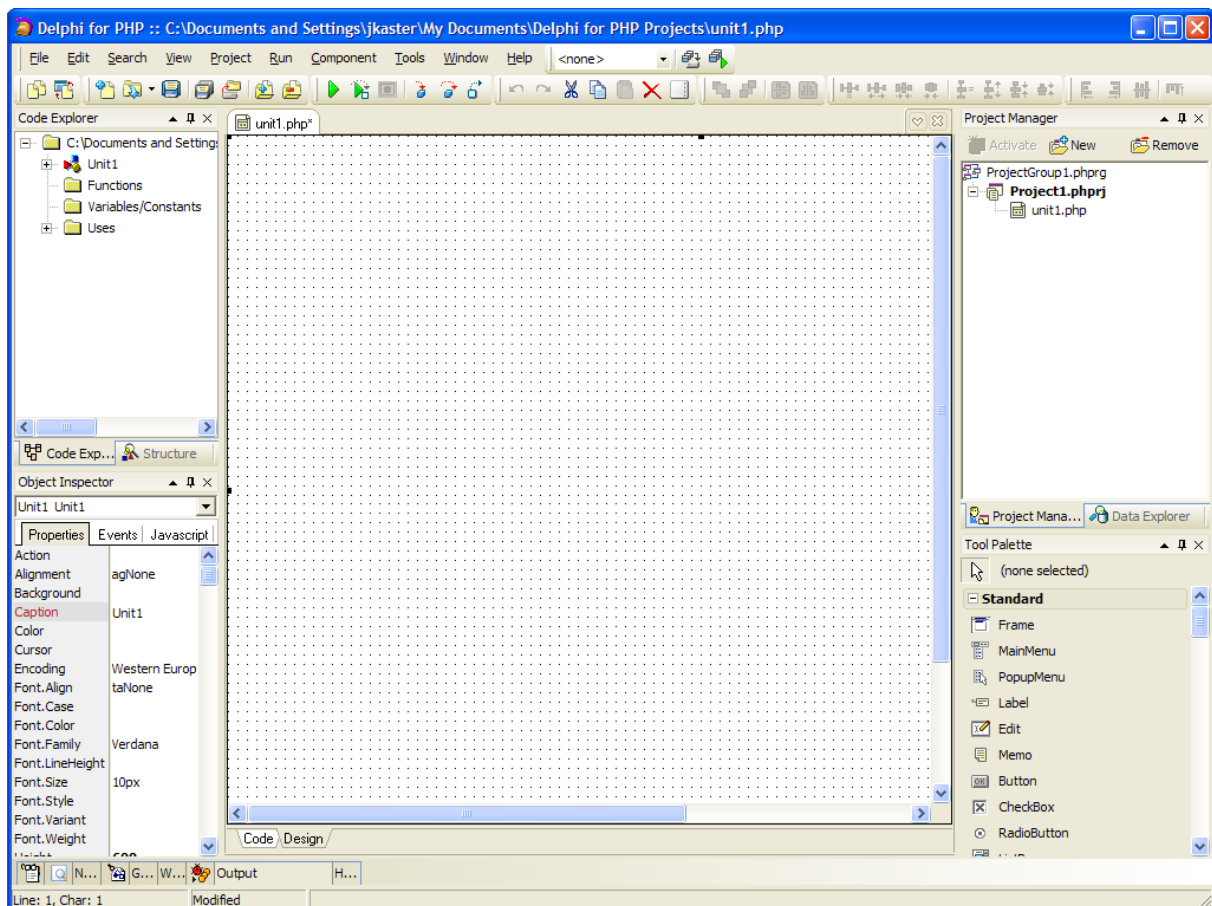


Abb.3-5 Delphi for PHP Layout

Es ist ein benutzerfreundliches Layout. Es stellt dem Anwender eine WYSIWYG-Umgebung zur Verfügung. Man klickt in der Toolbox (rechts unten) auf ein Tool (Frame, Button, Edit, Label,...) und zieht es dann in das Design-Fenster (Mitte). Doppelklickt man auf dieses Tool, öffnet sich das Editor-Fenster, wo man seine PHP-Code für seine Anwendung eingeben kann [Codegear 2006].

Schließlich möchte ich noch mal darauf hinweisen, dass CASE-Tools keine Zauberstäbe sind, mit denen allein das lauffähige Software-System erzeugt werden kann. Zuerst müssen Vorgehen und Methoden zur Software-Entwicklung eingeführt werden, dann erst können Werkzeuge sinnvoll zur Durchführung eingesetzt werden. CASE-Tools können die erfolgreiche Anwendung von Software-Engineering unterstützen, sie können aber in keinen Fall ein ingenieurmäßiges Vorgehen ersetzen.

M. Vetter [Vetter,95] hat die Problematik der CASE-Tool-Anwendung sehr prägnant formuliert: „Die besten CASE-Tools sind wertlos, wenn sie nicht im Rahmen eines methodischen Vorgehens zur Anwendung gelangen. Dieses methodische Vorgehen ist zu beherrschen, genauso wie die zur Herstellung von Anwendungen einzusetzenden Techniken. CASE-Tools vermögen zwar den Wirkungsgrad besagter Techniken zu erhöhen, keinesfalls aber- gewissermaßen per Knopfdruck - Anwendungen aus dem Nichts zu erzeugen.“

4. Softwaremessung und -bewertung:

Bei der Softwaremessung geht es darum, für einige Merkmale eines Softwareprodukts oder -prozesses einen numerischen Wert abzuleiten. Durch den Vergleich dieser Werte untereinander sowie den Vergleich mit den organisationsweit geltenden Standards ist es möglich, Schlussfolgerungen über Qualität der Software oder des Softwareprozesses zu ziehen.

Für eine ingenieurmäßige Software-Entwicklung benötigt der Ingenieur ein entsprechendes *Maßsystem*, welches wie folgt definiert werden kann [DFKW 96]:

*„Ein **Maßsystem** (system of measures) ist eine Menge von **Software-Maßen**, die sich auf alle wesentlichen Aspekte der Software-Entwicklung bezieht und die Bewertung bzw. die damit verbundene Prüfung der Einhaltung von Vorgabekriterien gewährleistet.“*

Software-Maße (*software measures*) beziehen sich auf Attribute von ein oder mehrere Objekte bzw. Komponenten der Software-Entwicklung.

Ein Messobjekt (*measurement object*) kann ein Quellprogramm, eine grafische Nutzeroberfläche oder aber auch der Software-Entwickler selbst sein.

Die Anwendung eines Software-Maßes bezeichnet man als Software-Messung und definiert sie in Anlehnung an [DFKW 96] in folgender Weise:

*„Die **Software-Messung** (software measurement) ist der Prozess der **Quantifizierung** von Attributen der Objekte bzw. Komponenten des Software Engineering mit der Ausrichtung auf spezielle **Messziele** (measurement goals) und der ggf. Einbeziehung von **Messwerkzeugen** (measurement tools).“*

Die Messziele können dabei sehr unterschiedlicher Art sein und sich auf alle Komponenten der Software-Entwicklung beziehen. Beispiele derartiger Messziele sind:

- die Bestimmung der Größe eines Programms,
- die Bewertung der Benutzerfreundlichkeit eines Software-System,
- die Ermittlung der durchschnittlichen Antwortzeit eines Rechensystems,
- die Bestimmung der Zuverlässigkeit einer software-basierten Robotersteuerung,
- die Abschätzung der Bearbeitungszeit und der -kosten einer Systementwicklung auf der Grundlage einer vorgegebenen Problemstellung,
- die Bewertung der Programmiererfahrung eines ausgewählten Entwicklungsteams,
- die Bewertung der Korrektheit eines Programmiersystems,
- die Einschätzung der Tauglichkeit von einzusetzender Standardsoftware im zu entwickelnden Gesamtsystem,
- die Bestimmung der Produktivität unterschiedlicher Entwicklungsteams.

Die Messziele sind bei der Software-Entwicklung im Allgemeinen in den qualitativen und prozessspezifischen Anforderungen explizit oder implizit festgelegt. Gegebenenfalls können sie auch mit den systembezogenen Anforderungen verknüpft sein.

4.1 Was sind Metriken?

In den Literaturen gibt es verschiedene Definitionen von der Software-Metrik. Hier sind einige:

Dumke definiert Software-Metrik wie folgt [Dumke,03]:

*„Eine **Software-Metrik** (software metric) ist gemäß der **Maßtheorie** (measure theory) eine **Abstandsfunktion** (distance), die Attributen von Software-Komponenten Zahlen (-bereiche) zuordnet.“*

Die Abstandsfunktion ist dabei in folgender Weise definiert: Für eine Metrik m als funktionale Abbildung von Elementen a einer Menge A auf die Menge der positiven reellen Zahlen \mathbb{R} gilt

- | | |
|--|-----------------|
| 1. $\forall a_1, a_2 \in A$ gilt $m(a_1, a_2) \in \mathbb{R}$, | Wertebereich, |
| 2. $\forall a_1, a_2 \in A$ gilt $m(a_1, a_2) = 0$ für $a_1 = a_2$, | Identität, |
| 3. $\forall a_1, a_2 \in A$ gilt $m(a_1 \cup a_2) \leq m(a_1) + m(a_2)$, | Additivität, |
| 4. $\forall a_1, a_2 \in A$ gilt $m(a_1, a_2) = m(a_2, a_1)$, | Kommutativität, |
| 5. $\forall a_1, a_2, a_3 \in A$ gilt $m(a_1, (a_2, a_3)) = m((a_1, a_2), a_3)$, | Assoziativität, |
| 6. $\forall a_1, a_2, a_3 \in A$ gilt $m(a_1, a_3) \leq m(a_1, a_2) + m(a_2, a_3)$, | Transitivität. |

Ian Sommerville versteht unter eine Software-Metrik folgendermaßen [Sommerville 07]:

*„Eine **Softwaremetrik** ist jede Art von Messung, die sich auf ein Softwaresystem, einen Prozess oder die dazugehörige Dokumentation bezieht.“*

IEEE Standard 1061 (1992):

*„Eine **Softwarequalitätsmetrik** ist eine Funktion, die eine Software-Einheit in einen Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit.“*

ISO/IEC 9126:

*„Eine **Software-Metrik** ist eine quantitative Skala und eine Methode, mit welcher der Wert ermittelt werden kann, den ein Indikator für ein bestimmtes Softwareprodukt oder ein bestimmtes Softwareprojekt aufweist.“*

Wir haben zu verstehen, dass eine **Softwaremetrik** eine Art von Messung ist, die bei der Analyse eines Projektes hilft und wichtige Informationen über es, wie beispielsweise seine Größe anhand der Anzahl der Codezeilen oder seine Komplexität, liefert.

4.2 Klassifikationen von Metriken

In der Literatur findet man zahlreiche Ansätze von Metriken (ca. 1500 Metriken [Zuse 98]) und eine Vielzahl von Klassifikationsversuche. In diesem Kapitel wird anhand häufig zu

findender Einteilungen versucht, einen großen Überblick über die verschiedenen Kategorien von Software-Metriken zu geben, wobei auch Mischformen möglich sind, d.h. dass sich Metriken nicht eindeutig einer dieser Kategorien zuordnen lassen.

Die bekannteste und weit akzeptierte Klassifikation ist die von Fenton. Fenton hat die Software-Metriken in drei Gruppen eingeteilt [Fenton,Pfleeger 97]: Produkt-, Prozess- und Ressourcenmetriken.

- **Produktmetriken** betreffen die Ermittlung und Analyse von Maßzahlen für das Software-Produkt. Unter Produktmetriken versteht man Größen bzw. Umfangsmetriken: Anzahl der Codezeilen, Anzahl der Function Points, die Halstead-Metriken, usw. Ebenfalls gehören zu den Produktmetriken die Komplexitätsmetriken wie die zyklomatische Zahl oder Fan-in/Fan-out.
- **Prozessmetriken** versuchen, die während der Software-Entwicklung ablaufenden Prozesse zu charakterisieren und zu quantifizieren. Hierunter fallen hauptsächlich Maturity- und Management-Metriken, wie z.B. Organisationsniveau, Technologieniveau, Meilensteinmetriken, Risikometriken, Produktivitätsmetriken, aber auch die Kostenschätzverfahren, beispielsweise COCOMO I/II. Die Life-Cycle-Metriken werden ebenfalls den Prozessmetriken zugeordnet, da auch hier Aussagen zum Prozess gemacht werden.
- **Ressourcenmetriken** wie z.B. die Programmiererfahrung, Kommunikationsniveau, Größe des Entwicklungsteams, Produktivität, Leistungsmerkmale, Verfügbarkeit, Zuverlässigkeit.

Da ich in dieser Ausarbeitung nur auf Produktmetriken fokussiere, die man aus dem Quellcode erheben kann, werden weitere Unterscheidungsmerkmale bei Prozess- und Ressourcenmetriken nicht weiter erläutern. Die Produktmetriken werden weiterhin nach dem verwendeten Strukturierungsparadigma unterteilt und stellen somit ebenfalls eine historische Unterteilung dar.

- *Traditionelle Metriken:* Traditionelle Metriken wurden ursprünglich dafür entworfen, den Anforderungen imperativer Programmiersprachen gerecht zu werden. Kritiker sehen in ihnen für den Einsatz in objektorientierten Umgebungen keine ausreichende Berücksichtigung von objektorientierten Konzepten.
- *Objektorientierte Metriken:* Mit zunehmender Verbreitung der Objektorientierung entstanden vermehrt objektorientierte Metriken. Diese Metriken verfolgen das Ziel, auf die Besonderheiten von objektorientierten Strukturen einzugehen. Allerdings existiert auch in diesem Bereich keine eindeutige Definition, was traditionelle von objektorientierten Metriken unterscheidet. Ebenso ist unklar, welche traditionellen Metriken sinnvoll in objektorientierten Umgebung eingesetzt werden können.
- *Aspektororientierte Metriken:* Ein aktueller Trend ist die aspektororientierte Softwareentwicklung, bei der zur Zeit der Kompilierung zentral verwaltete Aspekte in Quellcode eingewebt werden. Metriken für diese Sprachart werden zurzeit entwickelt.

Eine weitere Trennung erfolgt nach der Art, wie die Daten der Metriken erhoben werden. Balzert benutzt folgende Kriterien, die teilweise orthogonal zu einander stehen, für eine Klassifikation von Metriken.

Objektiv/Subjektiv

- Zu *objektiven Metriken* gehören die Metriken, die leicht quantifizierbar und messbar sind, z.B. Programmumfang, verbrauchte Zeit oder Fehleranzahl.
- Die *subjektiven Metriken* erfordern menschliche Einschätzung. Ein Beispiel für eine solche Metrik ist Kundenzufriedenheit. Daten für solche Metriken können Antwortklassen zugeordnet werden, z.B. ausgezeichnet, gut, befriedigend, schlecht. Diese Klassen sollten durch Referenzpunkte auf einer Skala definiert werden.

Absolut/Relativ

- *Absolute Metriken* sind invariant gegenüber der Hinzufügung neuer Elemente. Der Programmumfang ist beispielsweise absolut und unabhängig vom Umfang anderer Programme.
- *Relative Metriken* ändern sich abhängig von anderen Elementen, z.B. der Durchschnitt. Objektive Metriken sind oft absolut, wobei die subjektiven meistens relativ sind.

Direkte/Abgeleitet

- *Direkte Metriken*, auch Basis genannt, werden direkt ermittelt. Ein Beispiel für direkte Metrik ist Ausfallanzahl,
- *abgeleitete Metriken*, auch berechnete Metriken genannt, von anderen direkten oder abgeleiteten Metriken berechnet werden. Ein Beispiel für abgeleitete Metrik ist Fehleranzahl geteilt durch LOC.

Dynamisch/Statisch

- *Dynamische Metriken* besitzen eine Zeitdimension, z.B. gefundene Fehler pro Tag. Die Werte von solchen Metriken ändern sich abhängig davon, wann die Messung durchgeführt wurde.
- *Statische Metriken* bleiben invariant, z.B. die Anzahl von gefundenen Fehlern in einer ganzen Entwicklung.

Vorhersage/Bewertung

- *Vorhersagende Metriken*, oder Abschätzungen können im Voraus ermittelt oder generiert werden, während die
- *Bewertungsmetriken* hinterher ermittelt werden.

Global/Speziell

- *Globalen Metriken* umfassen meistens mehrere Phasen eines Entwicklungsprozesses, z.B. Umfang, Produkt- und Prozessqualität.
- *Spezielle Metriken* sind Indikatoren für jeweils eine spezielle Phase im Entwicklungsprozess.

Ian Sommerville unterteilt Metriken in Steuer- und Vorhersagemetriken [Sommerville,07].

- *Steuermetriken* sind normalerweise mit den Softwareprozessen verbunden. Als Beispiel für diese Metriken sind der durchschnittliche Aufwand und die für die Beseitigung festgestellter Fehler erforderliche Zeit zu nennen.
- *Vorhersagemetriken* sind mit Softwareprodukten verbunden. Beispiele für diese Metriken sind die zyklomatische Komplexität eines Moduls, die durchschnittliche Länge von Bezeichnern in einem Programm und die Anzahl der für die Objekte eines Entwurfs verbundenen Attribute und Operationen.

Eine erfolgreiche Methode Metriken zu klassifizieren ist die Klassifikation nach dem Software Engineering Entwicklungsprozess. Eine übersichtliche Darstellung davon mit typischen Kenngrößen ist in der folgenden Abbildung 4-1 präsentiert.

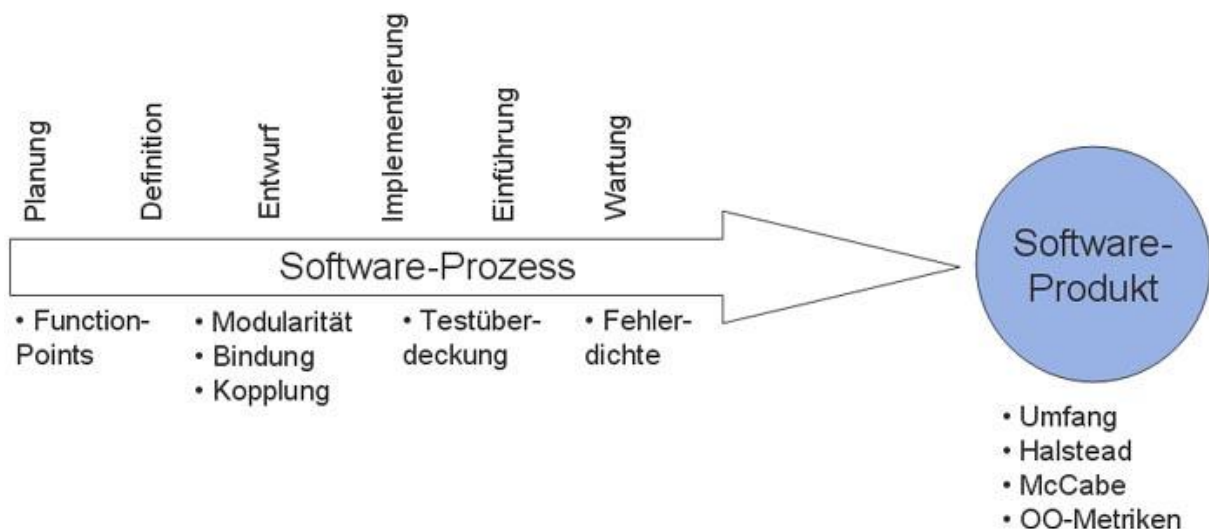


Abb. 4-1 die Klassifikation nach dem Software Engineering Entwicklungsprozess

Weitere Klassifikationen findet man in [Ebert 04], [Kitchenham 96], [Möller,Paulish 93] und [Munson 03].

4.3 Beispiele für Metriken:

Metriken gibt es heutzutage wie Sand am Meer. Die im Folgenden aufgeführten Metriken stammen von Chidamber/Kemerer, Lorenz/Kidd u.a. [Lorenz 94].

LOC (Lines of Code):

Bei dieser Metrik handelt es sich um eine der ältesten und bekanntesten Metriken, bei der die Anzahl der Quellcodezeilen gezählt wird. LOC-Metrik ist einfach zu messen und zu verstehen. Die Messung ist dabei mit Problemen wie beispielsweise der Sprachabhängigkeit (eventuell in unterschiedlichen Sprachen unterschiedliche Anzahl von Zeilen notwendig) verbunden. Allerdings berücksichtigt diese Metrik nicht das "Layout" und den Informationsgehalt des Codes.

WMC (Weighted Methods per Class)

Summe der Komplexitäten aller Methoden der Klasse ohne geerbte Methoden. Hier wird häufig die zyklomatische Komplexität nach McCabe verwendet, was allerdings den vollständigen Code erfordert; damit ist es keine Entwurfsmetrik mehr. Daher wird bisweilen als Gewicht die Anzahl der Parameter oder der konstante Wert 1 verwendet. Es wird angenommen, dass eine hohe Methodenkomplexität einen hohen Entwicklungsaufwand erfordert, und dass eine hohe Zahl von Methoden auf eine hohe anwendungsspezifische Spezialisierung hinweist; die zweite Interpretation ist allerdings nur sinnvoll, wenn die gesamte Vererbungshierarchie mit bewertet wird. Im Folgenden werden WMC1 (McCabe), WMC2 (Parameterzahl) und WMC0 (Einheitsgewicht) unterschieden.

LCOM (Lack in Cohesion Of Methods)

Nach unterschiedlichen Rechenverfahren wird aus der Nutzung der Attribute durch die Methoden einer Klasse ein Wert für den Kohäsionsmangel ermittelt. Es wird angenommen, dass eine geringe Kohäsion auf mehrere unterschiedliche Aufgaben zurückzuführen ist. Allerdings setzen alle Rechenverfahren den vollständigen Code voraus (ist also keine Entwurfsmetrik).

DIT (Depth in Inheritance Tree)

Länge des längsten Pfades von der Klasse zur Wurzel des Vererbungsbaumes ohne Interfaces. Es wird vermutet, dass eine große Vererbungstiefe zu schwer vorhersagbarem Verhalten, zu einer höheren Entwurfskomplexität und zu einer starken Wiederverwendung der geerbten Methoden und damit eingeschränkter Flexibilität der Oberklassen führt.

NOC (Number Of immediate Children)

Anzahl der unmittelbaren Nachfolgeklassen. Es wird vermutet, dass eine große Zahl von Nachfolgeklassen zu einer hohen Wiederverwendung der geerbten Methoden und zu einem großem Einfluss der Klasse auf den Gesamtentwurf führt; außerdem sei es wahrscheinlich, dass die Vererbung missbraucht wurde.

NAM (Number of Added Methods)

Anzahl der Methoden, die im letzten Vererbungsschritt hinzugefügt wurden. Eine hohe Anzahl lässt eine sehr starke Spezialisierung vermuten.

NOM (Number of Overridden Methods)

Anzahl der Methoden, die im letzten Vererbungsschritt überschrieben wurden. Eine hohe Anzahl lässt vermuten, dass die Vererbung missbraucht wurde.

CBO (Coupling Between Objects)

Anzahl von anderen Klassen (nicht Objekten), die der betrachteten Klasse in irgendeiner Weise bekannt sind. Es wird vermutet, dass eine hohe Kopplung die Wiederverwendung erschwert und die Empfindlichkeit gegen Änderungen an anderen Stellen des Entwurfs erhöht.

FI (Fan-in)

Anzahl der public Methoden der Klasse plus Anzahl der anderen Klassen, die darauf zugreifen. Es wird vermutet, dass ein hohes Fan-in zu einer hohen Empfindlichkeit des restlichen Entwurfs gegen Änderungen der Klasse führt.

RFC (Response For a Class)

Anzahl der public Methoden der Klasse plus der Methoden aus anderen Klassen, die von den eigenen Methoden aufgerufen werden. Es wird vermutet, dass eine große Antwortmenge auf eine hohe Komplexität der Klasse hinweist; außerdem wird über die Anzahl der aufgerufenen Methoden wiederum die Intensität der Kopplung erfasst. Grundsätzlich ist dafür der Code erforderlich; die UML 2.0 sieht jedoch eine Spezifikation der Import-Schnittstelle von Klassen, so dass RFC künftig auch ohne Code ermittelt werden kann.

4.4 Skalentypen:

Die Zuordnung von Zahlen (Messwerten) zu Objekten gemäß festgelegter Regeln nennt man *Messen*.

reale Welt → Zahlenbereich

Die Messwerte müssen zueinander eine Beziehung aufweisen und diese müssen den Beziehungen der gemessenen Objekte entsprechen. Das führt uns zu den Software-Maßen, die in [Dumke 03] wie folgt definiert sind:

*„Ein **Software-Maß** (software measure) ist gemäß der **Messtheorie** (measurement theory) eine mit einer **Maßeinheit** (unit) versehene **Skala** (scale), die in dieser Form ein Software-Attribut bewertet bzw. messbar macht.“*

In der Praxis allerdings werden die Begriffe Maße und Metriken aus Unwissenheit häufig synonym verwendet. Die gemessenen Ergebnisse (*Messwerte*) werden also auf einer *Skala* dargestellt. Auf die Menge der Messwerte (Skalenwerte) lassen sich formale Operationen anwenden. Jedoch nicht jede formal korrekte Operation hat eine entsprechend sinnvolle Interpretation in der realen Welt. In der Messtheorie unterscheidet man daher eine Hierarchie

von *Skalentypen*. Fenton und Pfleeger (1997) unterscheiden die folgenden Skalentypen [Fenton, Pfleeger,97]: *Nominalskala*, *Ordinalskala*, *Intervallskala*, *Rationalskala* und *Absolutskala*.

Nominalskala:

Für ein Software-Maß ist eine Nominalskala anzusetzen, wenn sich die Messwerte (Skalenwerte) in keine Reihenfolge bringen lassen oder miteinander vergleichen lassen. Die Nominalskala lässt nur die Prüfung auf Gleichheit zu. Es kann unterschieden werden, ob zwei Elemente gleich sind oder ungleich. Die Nominalskala ist die Skala mit dem niedrigsten empirischen Gehalt. Es sind außerdem keine Operationen erlaubt.

Beispiel: Die Nummern auf dem Rücken von Fußballspielern dienen zur Identifikation der Spieler. Diese Zahlen sagen nichts über deren Leistungsfähigkeit aus. Deshalb können wir auch nicht sagen, dass der Spieler mit der Nummer 11 besser als der mit der Nummer 3 spielt, es sei denn, der Trainer hätte die Zahlen nach der Leistungsfähigkeit der Fußballspieler vergeben. Es ist auch sinnlos diese Zahlen zu addieren.

Noch ein Beispiel: Die Zuordnung von Zahlen, z.B. zu verschiedenen Programmiersprachen, ist eine Nominalskala. Beispielsweise erhält ADA=3 und PASCAL =4. Dies bedeutet nicht, dass PASCAL einen höheren Wert hat, es heißt nur, dass PASCAL mit 4 und ADA mit 3 identifiziert werden. Ebenso gut könnte ADA mit 111 identifiziert werden. Die empirische Aussage würde sich nicht ändern.

Ordinalskala:

Für ein Software-Maß ist eine Ordinalskala anzusetzen, wenn die Messwerte (Skalenwerte) in eine Reihenfolge gebracht werden können, aber Aussagen bzgl. des Abstands zwischen den Messwerten nicht sinnvoll interpretiert werden können. Hier sind ebenfalls keine Operationen möglich. Eine Ordinalskala lässt im Vergleich zur Nominalskala nicht nur eine Unterscheidung der Messwerte zu, sondern bringt diese in eine Reihenfolge. Allerdings ist es nicht sinnvoll, über den jeweiligen "Abstand" zu reden.

Beispiel: Soll die Verstehbarkeit von Quelltext untersucht werden, lässt sich dies etwa durch Befragung von Versuchspersonen ermitteln. Diese werden gebeten, die Verstehbarkeit auf folgender Ordinalskala einzuordnen: sehr gut, befriedigend, schlecht.

Intervallskala:

Für ein Software-Maß ist eine Intervallskala anzusetzen, wenn die Messwerte (Skalenwerte) zwischen je zwei Messwerte ein Abstand identifiziert werden kann und die Abstände bedeutungstragend sind. Allerdings existiert kein Nullpunkt, Aussagen über das Verhältnis von Messwerte sind daher sinnlos. Eine Intervallskala lässt im Vergleich zur Ordinalskala eine Quantifizierung des Abstands zwischen Messwerte zu, ohne jedoch einen Nullpunkt zu implizieren. Auf der Intervallskala sind die Addition und Subtraktion erlaubt, womit es u.a. möglich ist, einen Durchschnitt zu bilden. Die Multiplikation ist auf dieser Skala aufgrund des fehlenden Nullpunkts nicht sinnvoll. Des Weiteren sind lineare Transformationen der Art $y=ax+b$ zulässig.

Beispiel: Bei der Messung der Temperatur kann gut zwischen wärmer und kälter unterschieden werden und auch den Temperaturunterschied zwischen zwei Temperaturen lässt sich sinnvoll bestimmen. Jedoch kann der Messwert Null nicht als Abwesenheit der Eigenschaft "Temperatur" gedeutet werden.

Rationalskala (Verhältnisskala):

Für ein Software-Maß ist eine Rationalskala anzusetzen, wenn es Sinn macht, über Vielfache von Messwerte zu sprechen, und wenn das Nichtvorhandensein einer Ausprägung interpretierbar ist. Bei einer Rationalskala lässt sich im Vergleich zur Intervallskala der Messwert Null interpretieren. Damit sind die Operationen Addition, Subtraktion, Multiplikation und Division möglich. Das Verhältnis von Abständen zwischen Messwerte ist relevant. Nun sind multiplikative Transformation der Art $y=ax$ erlaubt.

Beispiel: Soll beispielsweise die Länge einer Wegstrecke gemessen werden, kann dies in Metern ausgedrückt werden. Eine andere Skala ist die Messung in Kilometern. Welche der Skalen verwendet wird ist gleich, solange das Verhältnis aller Messwerte im Vergleich zur Eigenschaft "Länge" gewahrt wird.

Absolutskala:

Für ein Software-Maß ist eine Absolutskala anzusetzen, wenn die Messwerte festen, nicht transformierbaren Zahlen sind. Die Absolutskala ist mit einem natürlichen Nullpunkt und einer natürlichen Einheit und erlaubt nur Transformation der Art $y=x$. Umrechnungen, wie zum Beispiel Meter in Zoll, sind hier also nicht möglich.

Beispiel: Bei der Angabe der Anzahl der Kinder einer Mutter oder der Zahl der Tage mit Gewitter innerhalb eines Jahres für einen bestimmten Ort ist keine Datentransformation zulässig.

Tabelle 4-1 zeigt einen allgemeinen Überblick über die Eigenschaften der Skalen und die Unterschiede.

Merkmalsart	Qualitative Merkmale		Quantitative Merkmale (kontinuierliche oder diskrete)		
	Nominalmerkmal	Ordinalmerkmal			
Skalentyp	Topologische Skalen		Kardinalskalen (Metrische Skalen)		
	Nominalskala	Ordinalskala	Intervallskala	Rationalskala	Absolutskala
Definierte Beziehungen	= , ≠	= , ≠ < , >	= , ≠ < , > + , -	= , ≠ < , > + , - * , /	= , ≠ < , > + , - * , /
Interpretation der hinzukommenden Beziehungen	Unterscheidung gleich/ungleich möglich	Unterscheidung kleiner/größer möglich	Differenzen haben empirischen Sinn	Verhältnisse haben empirischen Sinn	–
Zugelassene Transformationen	umkehrbar eindeutige (bijektive)	monoton steigende (isotone)	lineare ($y=ax+b, a>0$)	Ähnlichkeits-transform. ($y=ax, a>0$)	Identität ($y=x$)

Beispiele für Merkmale	Postleitzahlen Autokennzeichen Artikelnummern Symbole Fehlerursachen (IEEE Std. 1044-1993, S. 9)	Schulnoten Militärische Dienstgrade Mercallische Erdbebenskala Windstärke Beaufort Prozessreifegrad (CMM)	Celsius-Temperatur Kalenderdatum zyklomatische Komplexität (McCabe, 1976)	Kelvin-Temperatur Einkommen Richtersche Erdbebenskala Windgeschw. m/s Projektdauer	Teamgröße Fehlerzahl Lines of Code
Beispiele für statistische Kennwerte	Modalwert Häufigkeiten	Quantile (Median, Quartile, ...)	Arithmetischer Mittelwert Standardabweichung	geometrischer Mittelwert Variationskoeffizient	wie Rationalskala, wenn der Wertebereich in die rationalen Zahlen eingebettet wird
Statistische Verfahren	nichtparametrische		parametrische, unter Beachtung der Modellvoraussetzungen		
Informationsinhalt	gering ----->hoch				
Empfindlichkeit gegenüber Ergebnisabweichungen	gering ----->hoch				

Tabelle 4-1: Übersicht über die Skalentypen (nach DIN 55350, Teil 12, S. 11, erweitert um die Absolutskala)

Die fundamentale Aufgabe des Messens besteht nun darin, für die zu beobachtenden Software-Maß die Art der Skala festzulegen.

4.5 Darstellungsformen von Messwerten

Ziel von Software-Messung ist es also nicht, einfach zu messen, was messbar scheint, wie Größe und Zeit, sondern auch solche Merkmale, wie Benutzerfreundlichkeit, Zuverlässigkeit, Tauglichkeit, Programmiererfahrung und Korrektheit. Derartige Charakterisierungen werden **empirische Merkmale** (*empirical aspects oder viewpoints*) genannt, da es sich hierbei um Bewertungsaspekte handelt, die auf die Erfahrung bzw. dem Wissenshintergrund der Software-Entwicklung und -Anwendung beruhen. Die Metriken leiten von diesen Merkmalen Zahlen (Messwerte) ab. Diese Messwerte können z.B. in **Tabellen** ermittelt werden. Aber ein Hauptproblem beim Messen ist auch die Interpretation von diesen Messwerten.

Demeyer hat in seinem Buch [Demeyer 99] geschrieben: „*a picture tells more than thousand words*“. So ist denn eine gute Methode, die uns beim Verstehen der Messwerte hilft, ist die Messwerte zu visualisieren.

Eine unmittelbare Darstellungsform von (empirischen) Messungen sind **Diagramme** (*diagrams*), wie zum Beispiel als Kurvenverlauf in einem Koordinatensystem, Histogramme, das Pareto-Diagramm, das Ursache-Wirkungsdiagramm, usw. Abbildung 4-2 zeigt einige

grundlegende Formen derartiger Diagramme. Damit kann einerseits ein Trend oder allgemeine Verhältnisse von Bewertungsgrößen oder aber die genaue Werte- bzw. Ergebnisbestimmung dargestellt werden.

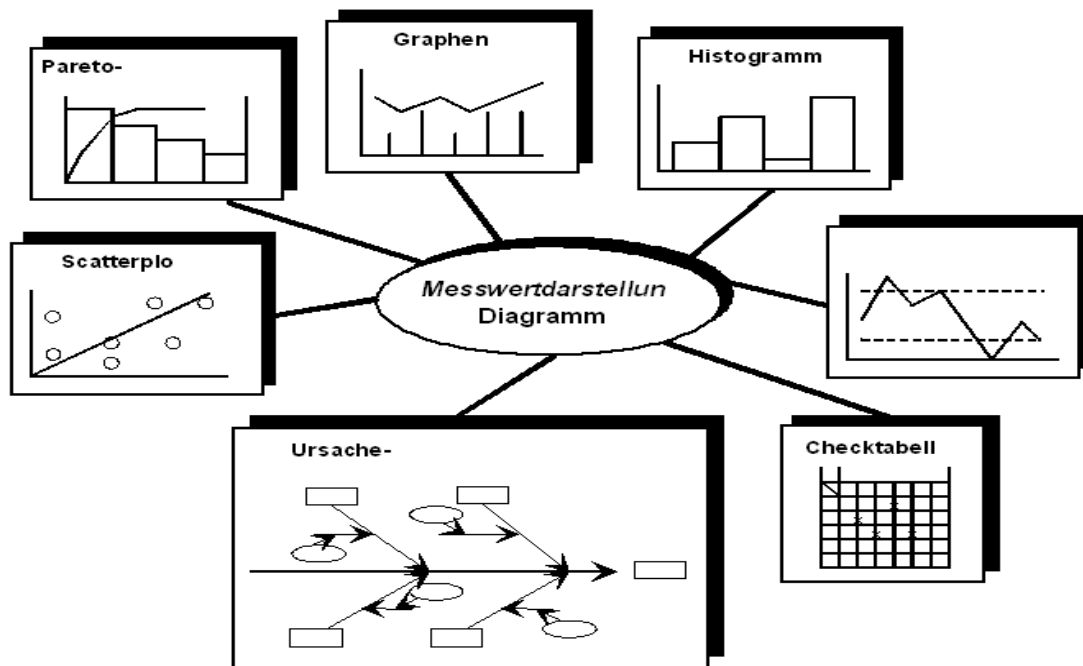


Abb. 4-2 Diagrammformen zur Darstellung von Messwerten

4.6 Software-Messtools

Schon bei der relativ bescheidenen Zahl der Metriken fragt man sich natürlich, ob für diese Metriken Werkzeuge vorhanden sind. Bereits bei einem Projekt relativ bescheidenen Umfangs fallen so viele Messungen an und sind so viele Auswertungen durchzuführen, dass die Arbeit ohne Unterstützung durch Werkzeuge kaum zu bewältigen ist. Z.B. ist Für LOC ein Werkzeug notwendig.

Die Verwendung von Werkzeugen zur Sammlung, Speicherung und Analyse von Messwerten kann wesentlich zur Vereinfachung des Messprozesses beitragen. Es gibt dazu eine Reihe von Werkzeugen, die zum Berechnen vordefinierter Metriken dienen. Für die Tools zur Software-Messung habe ich bereits den Begriff *CAME-Tools* eingeführt.

Eine gute Zusammenfassung von aktuellen Messtools, die in der Praxis verwendet werden, kann man im Internet finden, z.B. auf den Seiten des Testing-FAQs (<http://testingfaqs.org/t-static.html>), den Seiten der Firma VerifySoft (http://www.verifysoft.com/de_overview.html).

Hier möchte ich nur einige aktuelle Messwerkzeuge vorstellen.

Metrics Eclipse Plugin:

Bei diesem Messtool handelt sich um ein Plugin für die eclipse IDE, welches von <http://metrics.sourceforge.net/> bezogen werden kann. Das Tool stellt eine Reihe von Metriken zur Verfügung, mit deren Hilfe es möglich sein soll, Eclipse-Projekte auf Qualitätseigenschaften zu untersuchen. Es werden sowohl einfache als auch komplexere

Metriken berechnet. Vertreter von einfachen Metriken sind Lines of Code, Anzahl aller Klassen und Interface oder Anzahl der Instanzvariablen. Eine Auflistung aller verfügbaren Metriken zeigt Tabelle 4-2.

Number of Classes	Number of Children
Number of Interfaces	Depth of Inheritance Tree (DIT)
Number of Overridden Methods (NORM)	Number of Methods (NOM)
Number of Fields	Lines of Code
Specialization Index	McCabe Cyclomatic Complexity
Weighted Methods per Class (WMC)	Lack of Cohesion of Methods (LCOM)
Afferent Coupling (Ca)	Efferent Coupling (Ce)
Instability (I)	Abstractness (A)
Normalized Distance from Main Sequence	

Tabelle 4-2: verwendete Metriken in Metrics Eclipse Plugin

Die Ergebnisse werden dann tabellarisch in einer Baumstruktur dargestellt, wie Abb. 4.3

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Packages	16					
Number of Methods (avg/max per type)	1310	6.65	8.553	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
tgsrc	489	7.191	11.544	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
src	761	6.238	6.553	45	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.core.sources	108	15.429	12.129	45	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui	77	9.625	10.111	33	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.core	198	6.6	7.093	27	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui.preferences	52	6.5	7.467	26	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui.dependencies	95	5.588	3.727	15	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.persistence	18	4.5	4.33	12	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.prevayler.implementa...	54	5.4	2.871	10	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.xml	41	4.1	2.022	9	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.calculators	79	4.158	2.254	8	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.propagators	31	5.167	1.067	7	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.tests	8	2.667	1.886	4	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.prevayler	0	0	0			
classycle	60	8.571	2.556	13	/net.sourceforge.metrics/classycle/classycle/g...	
Lines of Code (avg/max per type)	6593	33.467	49.02	339	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
Number of Interfaces (avg/max per packageFragment)	16	1	1.414	4	/net.sourceforge.metrics/src/net/sourceforge/...	
Lines of Code (avg/max per method)	6593	4.812	7.355	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
classycle	324	5.4	9.94	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
tgsrc	2321	4.661	8.278	59	/net.sourceforge.metrics/tgsrc/com/touchgrap...	scrollSelectPanel
src	3948	4.862	6.473	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
net.sourceforge.metrics.ui	544	6.8	8.707	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
MetricsTable.java	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
MetricsTable	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
setMetrics	52					

Abb. 4.3 Ergebnissicht beim Metrics Eclipse Plugin

Die Anzeigereihenfolge der Metriken, sowie die Farbgebung der Tabellenwerte (inkl. Schwellwerte), können in den Einstellungen des Plugins festgelegt werden.

Weiterhin besteht die Möglichkeit Klassen- bzw. Pakethängigkeiten grafisch darzustellen, wie Abb. Auf der Projekt-Webseite wird genau beschrieben, wie dieser Graph zur Verminderung von Abhängigkeiten verwendet und somit zur Verbesserung der Projektqualität genutzt werden kann.

CCCC:

Dieses Messtool wurde in C geschrieben, das unter [CCCC 2006] bezogen werden kann. Es analysiert C, C++ und Java Quellcode. Reports sind in XML und HTML. Die verwendete Metriken sind LOC, CCN, WMC, DIT, NOC, CBO und andere.

CMTJava Metrikentool

CMTJava Complexity Measures Tool ist ein Werkzeug zur Messung der statischen Komplexität in Java implementierter Software. CMTJava berechnet die folgenden Softwaremetriken: McCabes Cyclomatic Number, Lines-of-code-Metriken, Halsteads Metriken und Maintainability Index [CMTJava 2006].

Es sollte noch darauf hingewiesen werden, dass eine Messung nur so gut wie das verwendete Werkzeug sein kann. In einem realen Softwareprojekt werden Metriken programmunterstützt erhoben, da dieser Vorgang sehr gut automatisiert werden kann. Ist das Messwerkzeug fehlerfrei, schleichen sich keine Fehler bei der Metrikerhebung ein. Dann ist die Analyse der Ergebnisse bezüglich der zu erreichenden Qualitätsziele der eigentliche Knackpunkt. Wenn das Messwerkzeug schon falsche Daten erhebt bzw. die Interpretation von Metriken zwischen verschiedenen Messwerkzeugen nicht vergleichbar ist, wird die Analyse der Messwerte nicht fehlerfrei verhalten.

4.7 Probleme bei der Auswahl und dem Einsatz eines Messtools

Die Notwendigkeit der Anwendung von Software-Messtools ist wohl unumstritten. Dennoch gibt es dabei häufig Probleme, die beim Auswahl und der Einführung von Messtools entstehen. Ich werde sie hier kurz beschreiben:

- Allerdings ist das Nutzen von Messtools noch relativ *zeitaufwändig*. Der Zeitaufwand betrifft nicht nur den Messaufwand mit dem Werkzeug an sich, sondern auch den eigentlichen Messvorbereitungsaufwand. Das Tool muss außerdem die verwendete Programmiersprache unterstützen, sodass dann Marktbeobachtungen zu entsprechenden Messwerkzeugen durchgeführt und dies dann auch getestet werden müssen.
- Ein großes Problem sind die *Systemschnittstellen* zwischen dem Entwicklungs- und Messtool. Einmal sind sie nicht kombinierbar, weil ihre Systemschnittstellen unverträglich sind. Das andere Mal rühren die Schwierigkeiten daher, dass die Werkzeuge auf unterschiedlichen Methoden beruhen, deren Ergebnisse nicht kompatibel sind.
- Falls mehr als ein Messtool gebraucht werden, dann sind wahrscheinlich die ausgewählten Messtools *nicht kombinierbar*, weil sie ihre Daten in verschiedenen Typen speichern.
- Ein anderes Problem ist die *arbeitsteilige Unterstützung* von mehreren Entwicklern mit der Anforderung, dass die Arbeitsergebnisse konsistent bleiben müssen. Die Mängel von Einzelwerkzeugen betreffen ihre Handhabung und ihre Erlernbarkeit.
- Ein weiteres Problem ist die *langfristige Abhängigkeit* von Werkzeugherstellern, insbesondere dann, wenn diese vom Markt verschwinden.

Eine Lösung dieses Dilemmas besteht darin, einheitliche Software-Entwicklung und -Messung einzusetzen, in denen Entwicklungs- und Mess-Werkzeuge integriert sind (Kapitel 5). Zudem gewinnt man über diesen Weg auch Zeit, um sich am Markt nach geeigneten Messwerkzeug umzusehen, deren Entwicklung derzeit kontinuierlich weiter läuft.

5. Metriken in Together

Together ist ein CASE-Tool, das alle Phasen im Software-Lebenszyklus unterstützt [Together 2006]. Es stellt dem Anwender UML Diagramme, GUI-Designer, Editor, Debugger, Audits und noch andere Funktionalitäten einschließlich Metriken zur Verfügung (Abschnitt 3-2).

Ich habe das *Fluggesellschaftssystem (FGS)* durch Together entwickelt und werde die Metriken an dieses Projekt anwenden, um die Metriken und ihre Ergebnisse zu beschreiben.

5.1 Das Dialogfeld Metriken

Nach Aufruf der Metriken erscheint ein Dialogfeld, das alle verfügbare Metriken anzeigt (Abbildung 5-1):

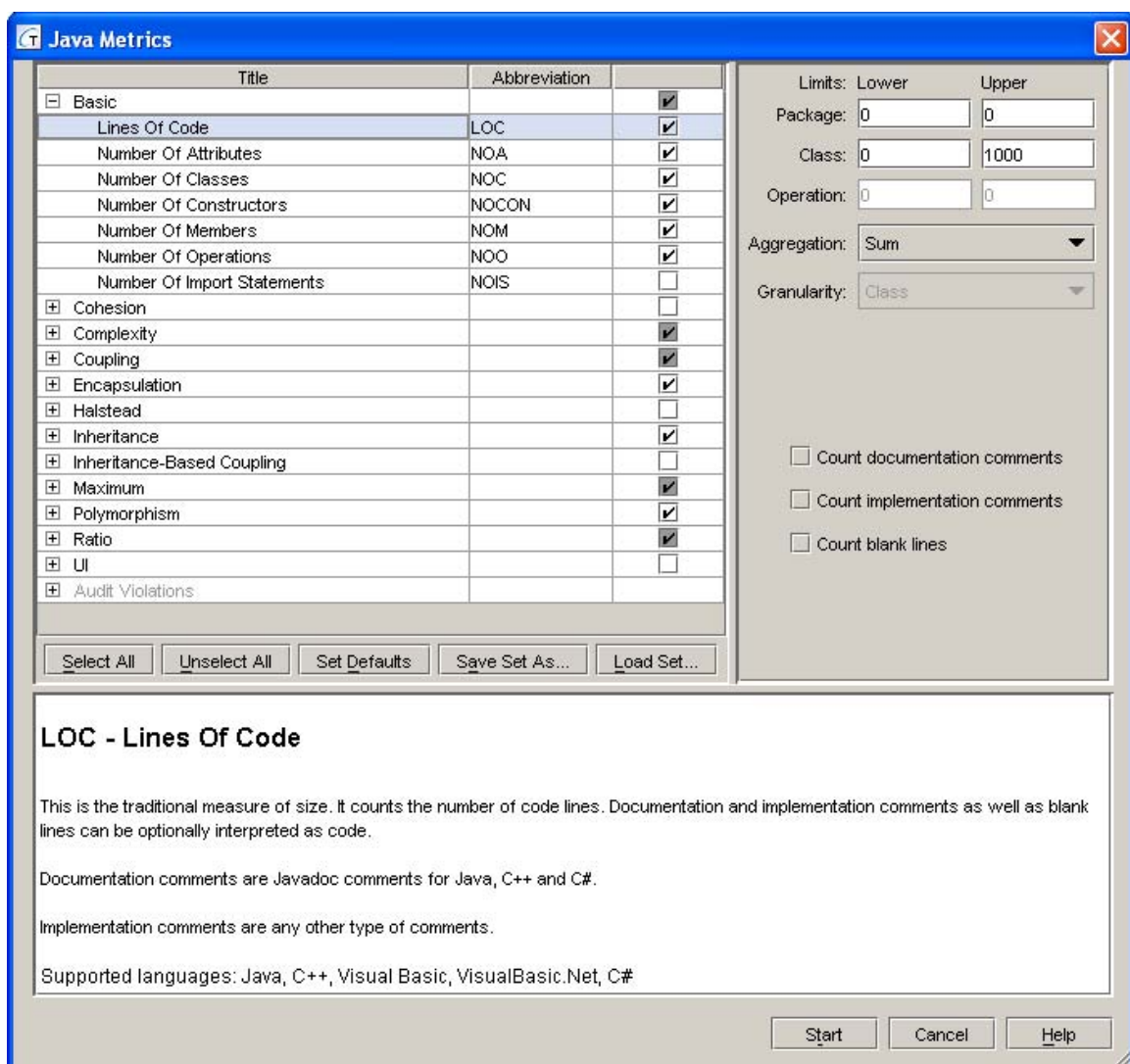


Abb. 5-1 Das Dialogfeld Metriken

Überschrift des Dialogfeldes ändert sich entsprechend der ausgewählten Programmiersprache für das aktuelle Projekt. *Java Metrics* ist beispielsweise die Dialogfeldüberschrift für Java Projekte.

Das Dialogfeld besteht aus drei Bereichen:

- Der **Auswahlbereich** befindet sich im linken Teil des Dialogfeldes. Da kann man Metriken auswählen, die man ausführen lassen möchte. Jede Metrik hat ihre eigene Abkürzung. Unter diesem Bereich befinden sich auch Schaltflächen, die bei der Auswahl der Metriken helfen, z.B. *Select All*, *Unselect All*, ...
- Der **Beschreibungsbereich** befindet sich im unteren Teil des Dialogfeldes. Da findet man eine Beschreibung der ausgewählten Metrik.
- Der **Optionsbereich** befindet sich im rechten Teil des Dialogfeldes. Da kann man für jede Metrik bestimmte Optionen einstellen bzw. ändern, z. B. Schwellwerte (*limits*). Daher kann man von Flexibilität der Together-Metriken sprechen. Die Optionen ändern sich je nach der ausgewählten Metrik, und sind gegebenenfalls auch im Beschreibungsbereich erklärt. Die folgenden Optionen sind bei den meisten Metriken anwendbar:
 - *Lower limit* und *Upper limit*: Sie sind die akzeptable Unter- und Obergrenze in den Metrik-Ergebnissen, und sind möglicherweise unterschiedlich für die Pakete, Klassen und Methoden. Die angezeigten Schwellwerte sind nur als Empfehlung, der Anwender darf natürlich seine eigene eingeben. Hat eine Metrik keine Schwellwerte, wird dann 0 angezeigt.

Hinweis: Diese Angaben werden nicht für die Berechnung der Metrik verwendet, sie helfen nur bei der Analyse der Ergebnisse. Die Ergebnisse, die nicht im akzeptablen Bereich liegen, werden in der Ergebnistabelle und den Graphen gekennzeichnet.
 - *Aggregation*: Passt die Zählmethode (Summe, Durchschnitt, ...) an.
 - *Granularity*: Passt die Stufe der Analyse (Projekt, Klasse, ...) an.

Die Metriken sind im Together gut dokumentiert. Together verfügt über eine ausführliche Hilfe im User Guide zu Metriken [Tog]. Das Dialogfeld und seine Komponenten werden in der Online-Hilfe [Tog] ausführlicher erklärt.

5.2 Kategorisierung der Together-Metriken

Together bietet eine Vielzahl von Metriken, dessen Anzahl hängt von der ausgewählten Programmiersprache für das erstellte Projekt ab. Java Projekte haben die größte Anzahl von Metriken, andere Sprachen unterstützen eine kleinere Auswahl von Metriken, die speziell für diese Sprachen entwickelt wurden.

Für Java stellt Together 55 Metriken zur Verfügung, die in 12 logischen Einheiten gruppiert sind, um die Übersichtlichkeit zu verbessern:

- Basic (7 Metriken)
- Complexity (6)
- Encapsulation (2)
- Inheritance (4)
- Maximum (3)
- Ratio (6)
- Cohesion (3)
- Coupling (7)
- Halstead (9)
- Inheritance-Based Coupling (4)
- Polymorphism (3)
- UI (1)

Um diese Metriken verwenden zu können, sollten man zuerst wissen, welche Metriken dies sind und was sie messen können. Deshalb befindet sich eine gute Beschreibung der ausgewählten Metrik im **Beschreibungsbereich**.

Together beinhaltet also neben den Basismethoden, die z.B. die Anzahl der Codezeilen (LOC), auch Metriken für die Kopplung (Coupling) und für den Zusammenhalt (Cohesion). Es gibt auch Metriken für die Vererbung und die Polymorphie. In Bezug auf die Komplexität gibt es ebenfalls Kennzahlen, beispielsweise die zyklomatische Komplexität nach McCabe. Nicht fehlen darf natürlich die Halstead-Metrik in verschiedenen Varianten.

Bemerkung: wie aus der Abbildung 2 ersichtlich gibt es eine dreizehnte Gruppe unter *Audit Violations* und sie erscheint nicht aktiv. Diese Gruppe wird im Abschnitt 5.6 diskutiert.

5.3 Wie werden Metriken verwendet?

Die Metriken können auf das ganze Projekt angewandt oder nur auf bestimmte Klassen, Pakete oder Diagramme eingegrenzt werden. Die erfolgreiche Anwendung der Metriken setzt einen syntaktisch korrekten Quellcode voraus. Ansonsten führen die Metriken nicht zu den gewünschten Ergebnissen.

Wenn der Anwender eine Metrik im Dialogfeld aktiviert, zeigen sich empfohlene Schwellwerte (Unter- und Obergrenze) der Metrik in Eingabefeldern im rechten Bereich des Dialogfeldes (Abbildung 5-2). Diese empfohlenen Schwellwerte stammen nicht alle aus einer bestimmten Literaturangabe, viele davon sind aus den langen Erfahrungen der Together-Hersteller hergekommen. Diese empfohlenen Schwellwerte sollten also nicht allzu als Regel gehalten und angewendet werden. Denn die Schwellwerte können unter bestimmten Umständen auch überschritten werden, wobei der Quellcode völlig in Ordnung ist. Die Schwellwerte können deshalb vom Anwender geändert werden. Hat eine Metrik keinen Schwellwert, wird dann 0 angezeigt bzw. eingegeben.

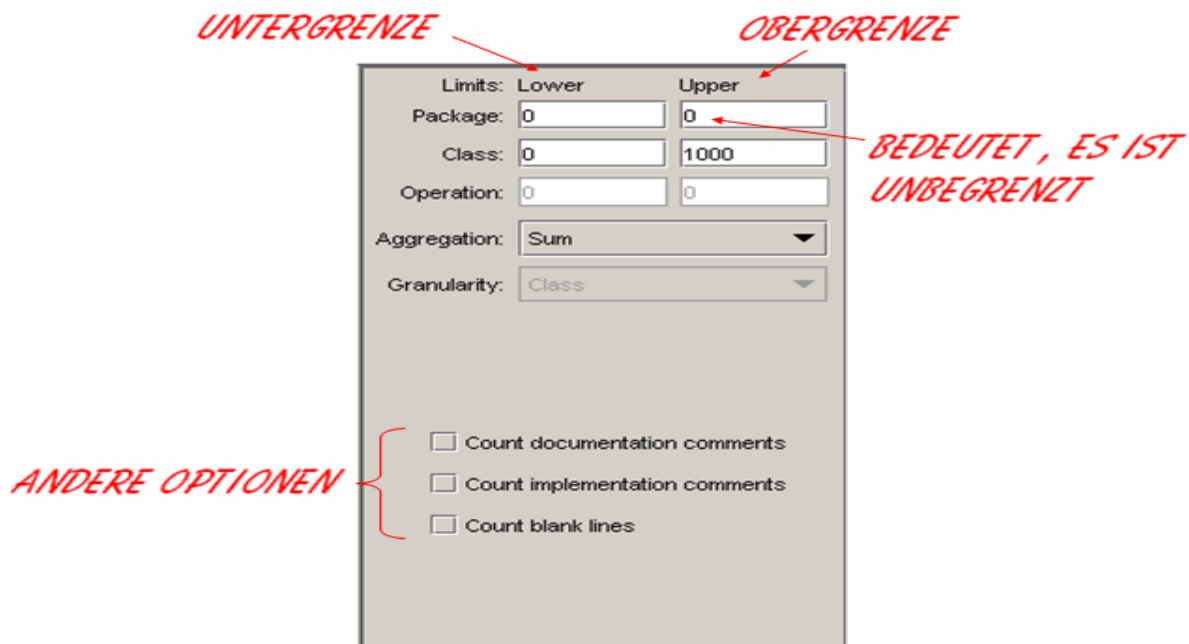


Abbildung 5-2 Schwellwerte und andere Optionen für LOC

Wenn das Dialogfeld Metriken geöffnet wird, wird automatisch eine Default-Menge von den verfügbaren Metriken ausgewählt. Diese Default-Menge enthält alle meist verwendeten Metriken (auf Basis von Erfahrungen des Herstellers). Es liegt aber an dem Anwender,

Metriken als sinnvoll für sein Projekt zu halten und auszuwählen. Der Anwender sollte deshalb alle verfügbaren Metriken vorher kennen und genau wissen, was sie messen können – Together bietet Tipps für alle Metriken und ihre Verwendung als Online-Hilfe. Der Anwender wählt in der Regel nicht alle verfügbare Metriken, sondern nur eine bestimmte Teilmenge davon, und bestimmt ihre Schwellwerte und die andere Optionen. Wenn er dann auf *Start* klickt, werden alle ausgewählte Metriken ausgeführt und die Ergebnisse angezeigt. Es liegt dann an ihm, die Ergebnisse zu interpretieren.

Wie die Metrik-Ergebnisse aussehen, das werde ich im nächsten Kapitel erklären. Mit der Schaltfläche *Set Defaults* im Dialogfeld Metriken lässt sich das Default-Menge jederzeit wiederherstellen.

5.4 Metrik-Ergebnisse

Metriken zeigen ihre Ergebnisse im *Message*-Fenster als eine Tabelle an (Abbildung 5-3).

Item	AHF	AIF	CBO	CC	CF	CR	DAC	DOIH	FO	HDiff	HEff	HPLen	LOC	LOCOM3	MHF
<default>	100	0	5	5	12	8	3	4	5	18	1571	104	123	50	0
Benutzeroberflaeche			1	1		27	1	1	1	0	0	0	7		
Flugplan			1	1		27	1	1	1	0	0	0	7		
FGSProblemDomain			5	5		8	3	4	5	18	1571	104	116	50	
Buchungsausnahme			0	2		8	0	4	0	1	1	2	8		
EconomyKlasse			0	1		36	0	2	0	1	1	2	6		
ErsteKlasse			0	1		36	0	2	0	1	1	2	6		
Flug			5	4		38	3	1	5	4	651	36	25		
Flugbeschreibung			1	2		17	1	1	1	2	36	7	14	0	
ICreateTickets			1	1		64	0	1	1	0	0	0	4		
INamed			0	1		13	0	1	0	0	0	0	4		
Mitarbeiter			1	1		40	1	1	1	0	0	0	7		
Reservierung			2	5		24	1	1	2	4	549	35	26	50	
Ticket			0	1		63	0	1	0	0	0	0	5		
TicketMaster			1	3		35	0	1	1	5	332	20	11		

Abbildung 5-3 Metrik-Ergebnisse

Die Zeilen enthalten die analysierten Klassen, Pakete oder Diagramme, die Spalten zeigen die ausgewählten Metriken und ihre entsprechenden Messwerte. Die Messwerte in der Tabelle werden wie folgt farblich gekennzeichnet:

- Schwarze Ziffern stehen für akzeptable Werte.
- Blaue Ziffern stehen für Werte, die die vorgegeben Untergrenze unterschreiten.
- Rote Ziffern stehen für Werte, die die vorgegeben Obergrenze überschreiten.

Die Ergebnisse stehen in enger Beziehung zum Quellcode. Der Anwender kann durch einen Doppelklick auf jede Zeile in der Ergebnistabelle zur entsprechenden Stelle im Diagramm oder im Quellcode wechseln. Metrik-Ergebnisse können auch Teile vom Quellcode markieren, die geändert werden sollen.

5.5 Eigenschaften und Charakteristiken der Metrik-Ergebnisse

Wenn sich der Anwender die Ergebnisse anzeigen lassen, möchte er wahrscheinlich die Tabelleneinträge vergleichen, sortieren, visualisieren oder vielleicht auch drucken. Together hat noch mehr Möglichkeiten, die dem Anwender bietet.

5.5.1 Ergebnistabelle sortieren

Um die Einträge in der Tabelle leichter zu vergleichen und zu organisieren, stehen die Möglichkeiten Sortieren nach einer Spalte, Sortieren nach mehreren Spalten, Aufsteigende Sortierung (*Sort Ascending*) oder absteigende Sortierung (*Sort Descending*) zur Verfügung.

5.5.2. Speichern und Laden der Metrik-Ergebnisse

Nach der Ausführung von Metriken gibt es die Möglichkeit, die Ergebnisse zu speichern und später dann unabhängig vom Projekt als Tabelle zu laden und anzuzeigen. So hat der Anwender die Möglichkeit, die Ergebnisse anderen Anwendern vorzulegen oder sie über einen längeren Zeitraum zu vergleichen. Gespeicherte Metrik-Ergebnisse haben die Erweiterung *.mtbl*.

5.5.3 Vergleichen von Metrik-Ergebnissen

Diese Eigenschaft ist vorhanden, nur wenn zwei oder mehr Ergebnistabellen geöffnet sind, damit können beispielsweise aktuelle Messergebnisse mit einem gespeicherten Messergebnis oder zwei gespeicherten Ergebnistabellen verglichen werden.

Ergebnistabellen können in zwei Weisen verglichen werden:

- Beim Vergleichen von zwei Tabellen werden unterschiedliche Ergebnisse für dieselbe Metrik mit verschiedenen Farben hervorgehoben (Abbildung 5-4).
 - Niedrigere Werte haben hellblauen Hintergrund.
 - Höhere Werte werden rosa dargestellt.
 - Nicht verglichene Werte werden grau dargestellt.
 - Gleiche Werte werden weiß dargestellt.

NIEDRIGER ALS DENSELBE METRIK-WERT
IN DER ANDEREN TABELLE

HÖHER ALS DENSELBE METRIK-WERT
IN DER ANDEREN TABELLE

Item	AHF	AIF	CBO	CC	CF	CR	DAC	DOIH	FO	HDiff	HEff	HPLen	LOC	LOCO...	MHF	Mif
<default>	89	0	42	55	33	4	18	7	30	152	717657	4511	1820	95	23	1
Requirements			1	4		25	1	1	1	3	73	15	59	0		
data_management			7	11		24	3	2	7	12	22720	296	107	72		
problem_domain			13	44		31	6	5	11	32	50197	619	473	92		
server			9	55		4	2	1	8	42	159313	1118	420	67		
user_interface			42	45		13	18	7	30	50	480493	2384	700	95		
util			0	13		23	0	1	0	13	4861	79	61	31		

NICHT GEFUNDEN IN
DER ANDEREN TABELLE

WEISSER HINTERGRUND FÜR DENSELBE
MESSWERT IN DER ANDEREN TABELLE

Abbildung 5-4 Vergleichen von Tabellen mit Hintergrundfarben

- Beim Vergleich von zwei oder mehreren Tabellen zeigt sich ein Kurzhinweis über jeder Zelle in den Tabellen an, er enthält eine Liste von den entsprechenden Zellen in den anderen Tabellen (Abbildung 5-5).

DAC	DOIH	FO	HDiff	HEff	HPLen
18	7	30	152	717657	4511
<default>	DAC	1	3	73	15
3	7	7	12	22720	296
6	3	1	32	50197	619
2	Untitled3	0	8	42	159313
18	Untitled5	2	0	50	480493
0	Untitled6	1	0	13	4861
	Untitled7	18			

Abbildung 5-5 Kurzhinweis mit den entsprechenden Zellen in den anderen Tabellen

Mit dem Befehl *Go to Object in Table* im Kontextmenü kann bequem von einem Objekt in der aktuellen Tabelle zum gleichen Objekt in einer anderen Tabelle gewechselt werden.

5.5.4. Grafische Ansichten von Metrik-Ergebnisse:

Die Metrik-Ergebnisse können auch visualisiert werden. Es gibt drei grafische Ansichten, den *Kiviat Graph*, den *Bar Graph* und den *Distribution Graph*.

Kiviat Graph

Der Kiviat-Graph zeigt alle Metrik-Ergebnisse für eine bestimmte Klasse oder ein bestimmtes Paket an, berücksichtigt dabei nur Metriken mit vordefinierten Schwellwerten.

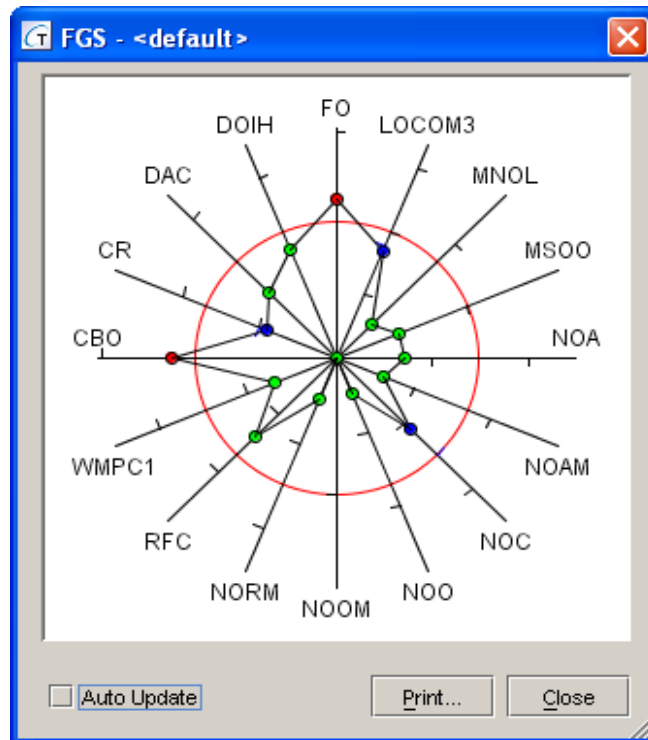


Abbildung 5-6 Kiviat Graph

Die Metriken werden als Strahlen dargestellt, deren Ursprünge im Mittelpunkt des Graphen treffen. Die Strahlen derart sind logarithmisch skaliert ([Tog] S.584), dass die Obergrenzen einen Kreis bilden. Die Messwerte und Schwellwerte werden wie folgt angezeigt:

- Ein roter Kreis stellt die Obergrenzen aller Metriken dar.
- Die Untergrenzen werden als kleine blaue Striche dargestellt, jeder Strich wird im rechten Winkel auf den einzelnen Strahlen gegen den Uhrzeigersinn angezeichnet. Untergrenze von 0 oder 1 werden nicht angezeichnet.
- Die Strahlen werden durch kleine Striche im rechten Winkel im Uhrzeigersinn auf den einzelnen Strahlen skaliert. Jeder Strahl hat seine eigene Maßeinheit.
- Alle Messwerte werden dann als Punkte dargestellt:
 - Grüne Punkte stehen für akzeptable Werte.
 - Blaue Punkte stehen für Werte, die die Untergrenze unterschreiten.
 - Rote Punkte stehen für Werte, die die Obergrenze überschreiten.
- Die Messwerte bilden dann einen Stern.

Bar Graph

Der Bar Graph (Balkengraph) zeigt die Ergebnisse einer bestimmten Metrik für alle Pakete, Klassen und/oder Methoden an.

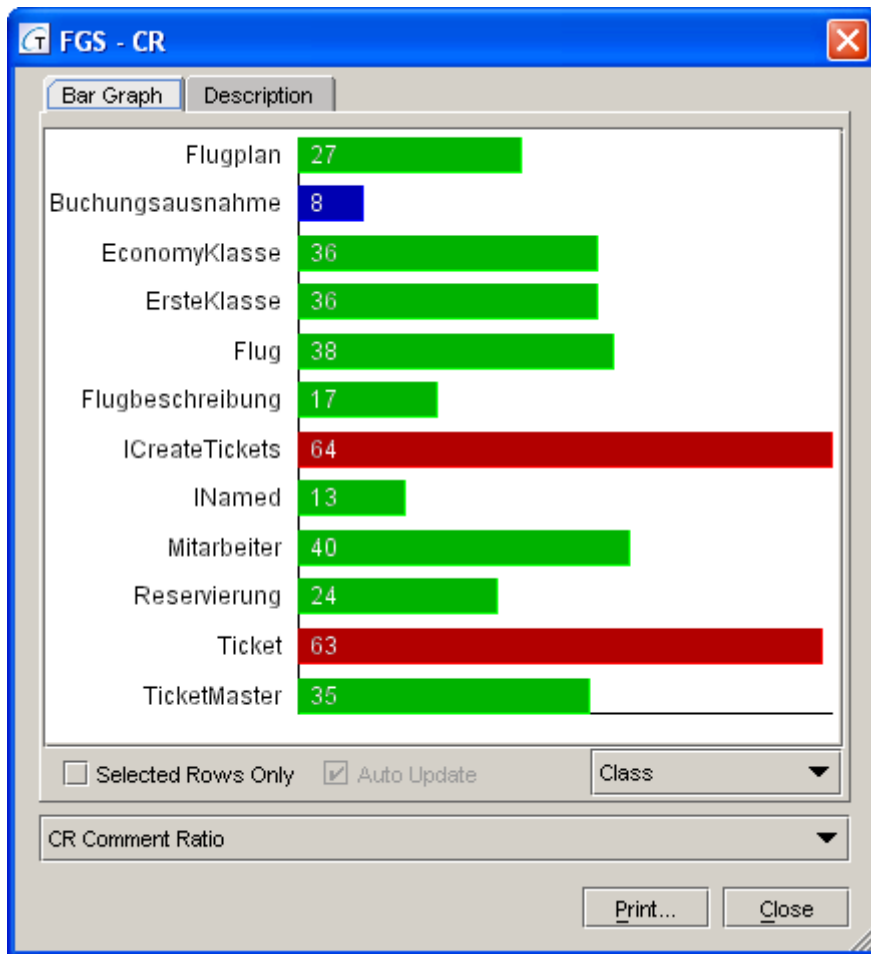


Abbildung 5-7 Bar Graph

Die Farbe des Balkens verdeutlichen, wie sich die betreffende Metrik zum festgelegten Schwellwerte verhält:

- Ein grüner Balken steht für akzeptable Werte.
- Ein roter Balken steht für Werte, die die Obergrenze überschreiten.
- Ein blauer Balken steht für Werte, die Untergrenze unterschreiten.

Durch Balkengraph kann einfach die Messwerte aller Objekte verglichen werden.

Mit einem Doppelklick auf einem Balken wird zum entsprechenden Quellcode im *Editor*-Fenster oder entsprechenden Diagramm im *Designer*-Fenster gewechselt.

Distribution Graph

Wie im Balkengraph zeigt der Distributionsgraph die Verteilung der Ergebnisse einer bestimmten Metrik für alle Pakete, Klassen und/oder Methoden an. Der Graph gibt einen Gesamtüberblick über das ganze Projekt an.

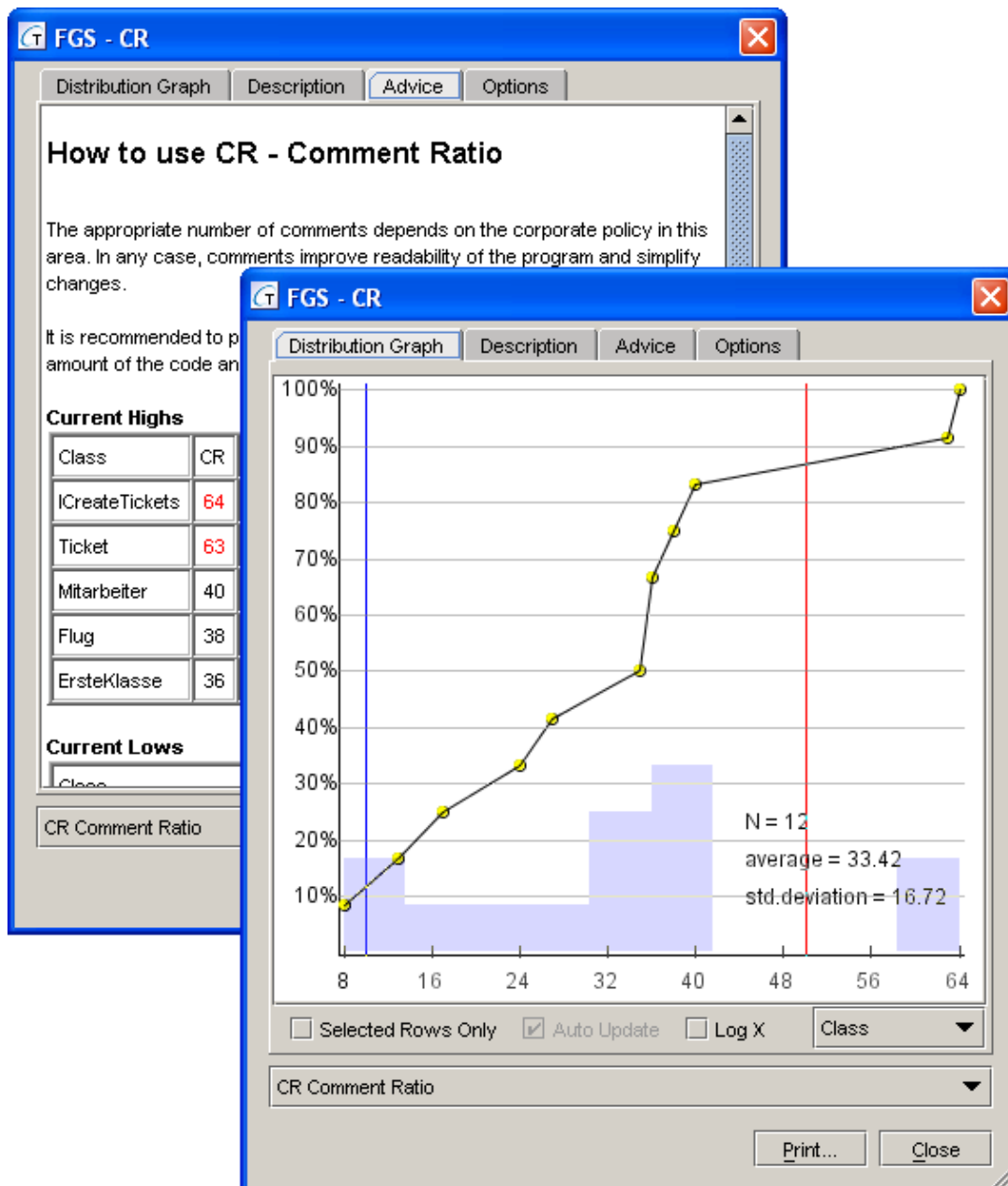


Abbildung 5-8 Distribution Graph

Eine dünne vertikale rote Linie stellt die Obergrenze dar, eine dünne vertikale blaue Linie die Untergrenze. In der Registerkarte *Advice* befindet sich Tipps dafür, wie die Ergebnisse interpretiert und verbessert werden.

5.5.5 Dokumentation der Metrik-Ergebnisse:

Der Anwender kann eine umfangreiche Dokumentation von den Metrik-Ergebnissen erstellen, um sie dann unabgänglich von Together vorzuzeigen und überprüfen.

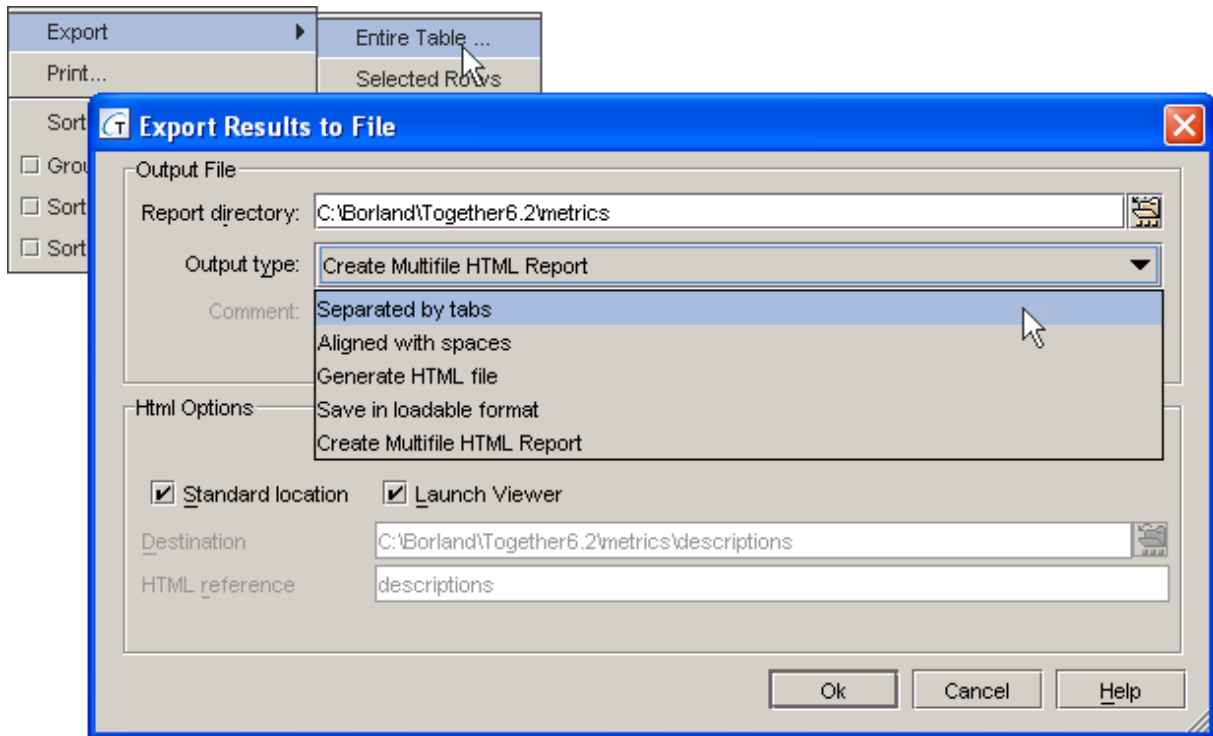


Abbildung 5-9 Die Möglichkeiten bei der Dokumentation

Die Ausgabe-Datei enthält alle an der Ergebnistabelle beteiligten Pakete, Klassen, Metriken und die gefärbten Messwerte. Sie wird entweder in eine Text-Datei, in der die Spalten durch Tabulatorzeichen oder Leerzeichen getrennt sind, oder in eine HTML-Datei, die Hyperlinks für das Navigieren durch den Projektbaum enthält (Abbildung 5-10), geschrieben.

Top level All classes All packages

Package <default> *UM EINE BESCHREIBUNG DER METRIK ZU ERHALTEN*

Item	AHF	AIF	CBO	CC	CF	CR	DAC	DOIH	FO	HDiff	HEff	HPLen	LOC	LOCOM3	MHF	MIF	MINOL	MSOO	NOAN	NOAN
<default>	100	0	5	5	12	8	3	4	5	18	1571	104	123	50	0	11	2	3	5	
Benutzeroberflaeche	1	1	27	1	1	1	1	1	0	0	0	0	7				0	1	1	
FGSProblemDomain			5	5		8	3	4	5	18	1571	104	116	50			2	3	5	

Top level All classes All packages *UM DURCH DEN PROJEKTBAUM ZU NAVIGIEREN*

Abbildung 5-10 HTML-Datei von den Metrik-Ergebnissen

Die HTML-Datei kann in allen Browsern geöffnet werden, mit Hilfe der Verknüpfungen können alle Pakete und Klassen angezeigt werden, und außerdem kann mit der Metrik-Abkürzung eine Web-Seite geöffnet werden, die den vollständigen Namen und die Beschreibung der Metrik enthält.

Zu der Dokumentation gehört auch das Drucken. Alle Metrik-Ergebnisse können gedruckt werden. Gedruckt werden können die ganze Ergebnistabelle oder nur bestimmte Metriken, Pakete oder Klassen. Außerdem können alle verfügbaren grafischen Graphen gedruckt werden.

Im Allgemeinen ist Together ganz abgedeckt dokumentiert, z.B. durch Online-Hilfe. Aber Als Kritikpunkt bei der Metriken merkt man, dass auf allen Visualisierungsgraphen keine Schaltfläche für Hilfe steht. Was soll z.B. LogX auf dem Distribution Graph bedeuten (Abbildung 5-8)? Dafür muss man im UserGuide nachschlagen.

5.6 Metriken für die Verletzungen in Audits-Regeln:

Audits bieten die Möglichkeit, die Software-Qualität zu messen, um die Schwachstellen im Code zu identifizieren und dann zu beheben. Mit Hilfe von Audits werden Überprüfungen von Dokumentation durchgeführt, und der spezifische Programmierstil einzelner Programmierer an den Programmierstandard des Unternehmens angepasst. Audits bieten tatsächlich viele Möglichkeiten, den Code zu verbessern.

Together bietet über 110 Audits, die in 10 Gruppen unterteilt sind. Together-Metriken können auch auf die Audit-Ergebnisse angewandt werden. Wie schon oben erwähnt wurde, gibt es eine Gruppe von Metriken unter *Audit Violations* im Dialogfeld Metriken, und enthält alle vorhandenen Audits mit ihrer Abkürzung, die Abkürzungen haben nur dem Präfix *av* (Abbildung 5-11). Diese Gruppe von Metriken wird aktiviert, erst wenn Audits vorher ausgeführt wurden.

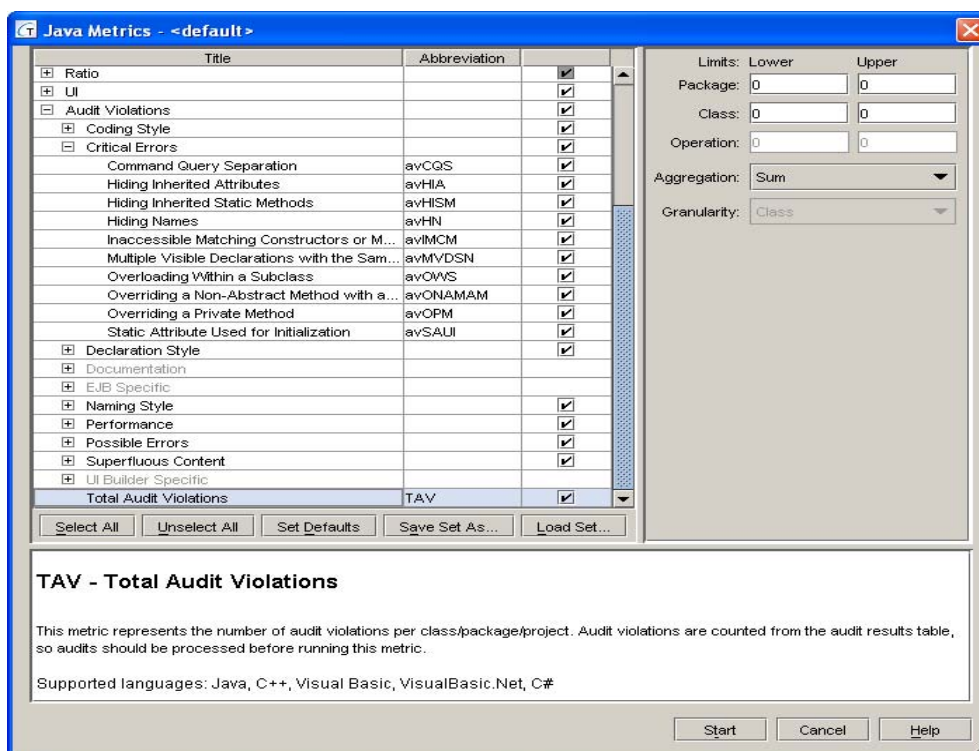


Abbildung 5-11 Metriken für *Audit Violations*

Die Metriken für *Audit Violations* ermitteln die Zahl der Verletzungen in jeder Klasse für die vorgewählte Audit. Die Gesamtanzahl der Verletzungen kann dann berechnet werden. Diese Gesamtanzahl wird auch durch die Metrik TAV (*Total Audit Violations*) ermittelt.

The screenshot shows a window titled 'Message Pane' with a sub-window 'Message Pane'. It displays a table of audit violation metrics for a project named 'FGS' in a file named 'Untitled16'. The table has 13 columns representing different metrics and 4 rows representing different items. The bottom of the window has three tabs: 'Messages', 'Metrics', and 'Java Audit'.

Item	TAV	avAFP	avEBWB	avILC	avNEC	avUC	avUCVN	avUIOE	avULVFP	avOIM	avUIMM	avUPCM
<default>	24	0	3	1	1	0	3	0	4	0	0	10
Benutzeroberflaeche	2	0	0	0	0	0	0	0	1	0	0	1
Datenverwaltung	0	0	0	0	0	0	0	0	0	0	0	0
FGS	0	0	0	0	0	0	0	0	0	0	0	0
FGSProblemDomain	22	0	3	1	1	0	3	0	3	0	0	9

Abbildung 5-12 Metrik-Ergebnisse für *Audits Violations*

5.7 Eigene Metriken definieren:

Die Metriken bei Together sind sehr flexibel und haben verschiedene Optionen, sodass sie alle Anforderungen des Anwenders erfüllen. Jedoch wenn der Anwender spezifische Anforderungen hat, kann er seine eigene Metriken herstellen, und sie dann in das QA Modul im Together einfügen - QA steht für *Quality Assurance*.

Das QA Modul befindet sich im `C:\Borland\Together6.2\modules\com\togethersoftware\modules\qa`. In diesem Ordner befinden sich auch Beispiele über die Weise, wie Metriken in Java geschrieben werden können. Sollte eine Beschreibung einer Metrik, ihre Interpretation oder Tipps dazu auf dem Distribution Graph angezeigt werden, sollten diese Informationen als HTML-Datei geschrieben und in denselben eben erwähnten Ordner beigefügt werden.

6. Bewertung des Messkonzeptes in Together

6.1 Allgemeine Aspekte der Metrikenanwendung in Together

Together bietet die Möglichkeit objektorientierte Metriken auf den Quellcode anzuwenden. Hinsichtlich der **Anzahl der Metriken** ergibt sich bei Together das folgende Bild:

- 55 Metriken für Java
- 54 für C#
- 52 für Visual Basic .NET
- 52 für C++
- 17 für Visual Basic 6

Da einige Metriken durchaus komplexer Natur sind, sollten sie im Tool möglichst ausführlich angelehnt an die folgenden **Beschreibungskriterien** dokumentiert sein.

Name und Abkürzung

Der eindeutige Name und die Abkürzung einer Metrik.

Bedeutung

Eine Beschreibung, wie die resultierende Kenngröße zu interpretieren ist.

Randbedingungen

Eine Liste von Faktoren, die das Ergebnis der Messung beeinflussen können.

Verwandte Metriken

Ähnliche Metriken oder Metriken, die häufig zusammen mit dieser genutzt werden.

Schwellwerte

Werte der Kenngröße, die bei Über- oder Unterschreitung Anomalien anzeigen.

Handlungshinweise

Hinweise darauf, welche Handlungen bei festgestellten Anomalien durchgeführt werden können, um diese zu beseitigen.

Eine Kurzbeschreibung sollte im Dialog der Auswahl der anzuwendenden Metriken angezeigt werden. Teil dieses Auswahlprozesses sollte auch die **Definition von Schwellwerten** sein, um gezielt Anomalien im Quellcode aufspüren zu können. Bei Together sind für die Metriken jeweils Schwellwerte vorgegeben.

Gerade bei Metriken, die auf eine Intervall- oder Rationalskala abbilden, kann eine grafische Darstellung der Ergebnisse aussagekräftiger sein als eine rein tabellarische. **Verschiedene Diagrammarten** zur Darstellung wie Balkendiagramme oder spezielle Diagramme zur Visualisierung von Distanzen sind denkbar. Daher gibt es bei Together auch diese verschiedenen Darstellungsformen.

Aus einer tabellarischen Darstellung heraus sollte es möglich sein mit einer Auswahlaktion – bspw. einem Doppelklick auf den entsprechenden Eintrag – die Komponente, die der Kenngröße zugrunde liegt, in Form des korrespondierenden Modellelements bzw. dessen Quellcodes in der Anzeige zu zentrieren, um die **Messung zu verifizieren** oder eventuelle Anomalien zu korrigieren.

Der **Export der Messergebnisse** in einem standardisierten Format ermöglicht die Weiterverwendung der Werte außerhalb des CASE-Tools zum Zweck der Einbindung in

einen Bericht, eine Präsentation oder der Analyse durch statistische Anwendungsprogramme. Bei Together besteht dieser Messwerteexport nur im eingeschränkten Maße.

Die meisten Messtools unterstützen nicht wirklich einen zielorientierten Ansatz, sondern dienen zum Berechnen vordefinierter Metriken. Da sich der Prozess für jedes Unternehmen unterscheidet, kann es im Tool dazu auch keine verfügbaren Metriken geben. Daher besitzt Together die Möglichkeit, auf der Grundlage ausgewählter Quellcodeelemente **Metriken selbst zu definieren** und anzuwenden.

6.2 Unterstützungsformen der Software-Entwicklung

Die Entwicklung der Software steht naturgemäß im Vordergrund des Interesse, deshalb müssen Metriken immer aus einer Vielzahl möglicher Messungen und Kennzahlen ausgewählt werden, unter Berücksichtigung der Tatsache, dass viele durch Metriken ermittelte Kenngrößen erst in Kombination mit anderen Messergebnissen ausreichend zuverlässige Aussagen über gewisse Sachverhalte zulassen. Daher sollte ein CASE-Tool eine möglichst große Zahl an unterschiedlichen Metriken anbieten. Hinsichtlich des Lifecycles bietet Together die folgenden Mess- bzw. Metrikenunterstützung.

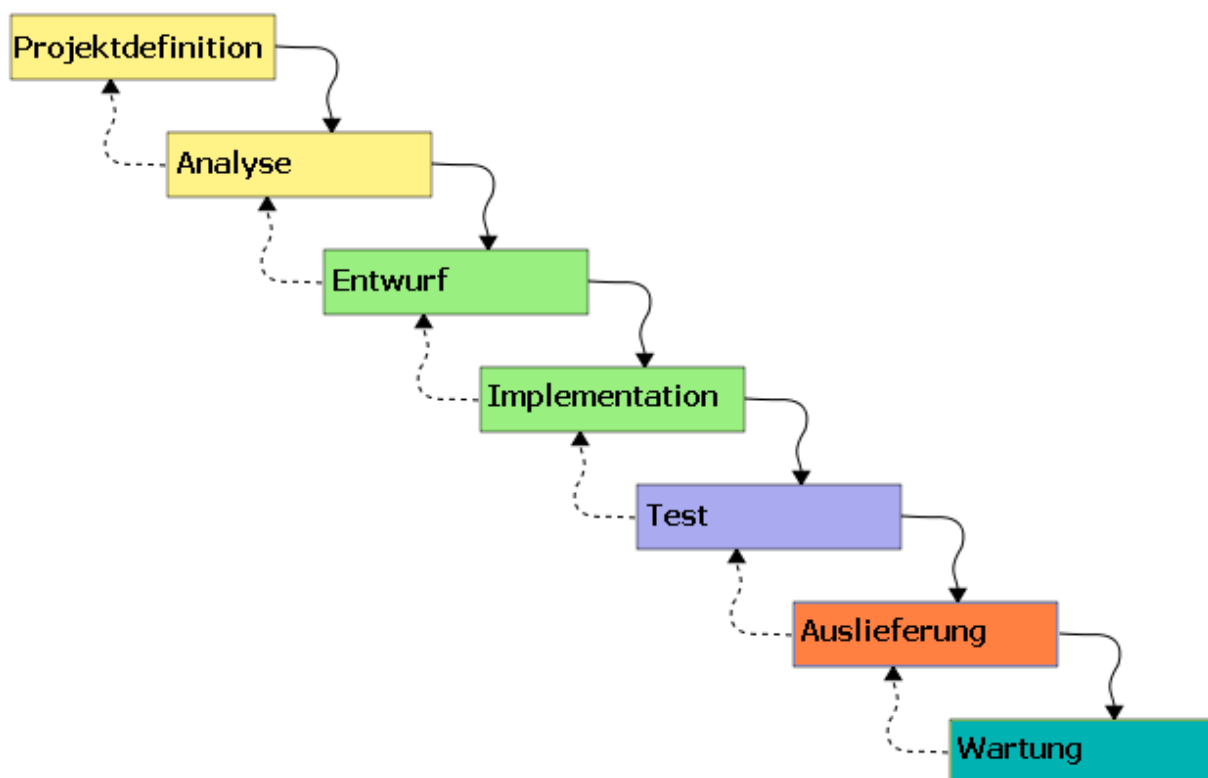


Abb.6-2 Phasen im Software-Lebenszyklus

Die Metriken können entsprechend den oben beschriebenen Kategorien (Prozess-, Ressourcen- oder Produkt-Metriken) geordnet sein, sowie können auf unterschiedlicher Skalen abbildende Metriken angeboten werden. Die Metriken von Together ordnen sich nach dieser Klassifikation wie folgt ein:

Produkt:

Prozess:

Ressourcen:

Hinsichtlich der grundlegenden Messprozesse (Reviews, Aufwandsschätzung, Codemessung usw.) bietet Together die folgenden Möglichkeiten:

Quellcode-Audits:

...

7. Zusammenfassung und Ausblick

Die vorliegende Arbeit dient der Darstellung und Bewertung von Softwaremessprozessen in CASE-Tools. Zunächst wurden einige Grundlagen zur Software-Entwicklung, den Methoden, Werkzeugen, Vorgehensmodellen und Prozessen beschrieben. Dann wurden die Software-Entwicklungswerkzeuge mit ihren generellen Merkmalen dargestellt und architekturelle Grundlagen zur Metrikenanwendung skizziert.

Die allgemeine Beschreibung der Softwaremessung und der Arten und Anwendungsformen der Metriken dienen der besseren Einschätzung der Metrikenanwendung innerhalb des ausgewählten CASE-Tools: Borland Together.

Die abschließende Bewertung der Metrikenanwendung im Together zeigten die bereits hilfreichen Unterstützungsformen aber auch mögliche Probleme bei der Metrikenanwendung.

Weitere Arbeiten sollten auf die Untersuchung der Einhaltung von Messstandards im Bereich der CASE-Tools gerichtet sein um damit das Niveau der Qualitätssicherung besser einschätzen zu können.

8. Literatur

- [Balzert 98] Balzert, H.: *Lehrbuch der Softwasretechnik*. Spektrum Verlag, 1998
- [CCCC 2006] CCCC-Toolbeschreibung, <http://sourceforge.net/projects/cccc> (5.3.2007)
- [CMTJava 2006] CMTJava-Dokumentation, <http://www.testwell.fi/cmtjdesc.html>, (5.3.2007)
- [Codegear 2006] Codegear-Homepage, <http://dn.codegear.com/php/> (5.3.2007)
- [Demeyer,99] Demeyer, *Object-Oriented Reengineering: The FAMOOS Experience*, 1999
- [Dumke 03] Dumke, R.: *Software Engineering*. Vieweg Verlag, 2003
- [DFKW 96] Dumke, R.; Foltin, E.; Koeppe, R.; Winkler, A.: *Software-Qualität durch Messtools – Assessment, Messung und instrumentiertes ISO 9000*. Vieweg Verlag, Wiesbaden Braunschweig, 1996
- [DLWZ 03] Dumke, R.; Lothar, M.; Wille, C.; Zbrog, F.: *Web Engineering*. Pearson Education Publ., 2003
- [Ebert 04] Ebert, C.; Dumke, R.; Bundschuh, M.; Schmietendorf, A.: *Best Practices in Software Measurement*. Springer Verlag, 2004
- [Fenton,Pfleeger 97] Fenton, N. E.; Pfleeger, S. L.: *Software Metrics – A rigorous and practical approach*. Thomson-Verlag, 1996
- [IEEE 90] *IEEE Standard in Software Engineering*, IEEE Press, 1990
- [ISO 12207 (1995)] *IEEE Standard of Software Processes*, IEEE Press, 1995
- [Kitchenham 96] Kitchenham, B.: *Software Metrics – Measurement for Software Process Improvement*. NCC Blackwell Verlag, Oxford, 1996
- [Lorenz 94] Lorenz, M.; Kidd, J.: *Object-Oriented Software Metrics*. Prentice Hall Verlag, 1994
- [Metamill 2006] *Metamill-Toolbeschreibung*, <http://www.metamill.com>. (5.3.2007)
- [Möller,Paulish 93] Möller, K. H.; Paulish, D. J.: *Software-Metriken in der Praxis*. Oldenbourg-Verlag, München, 1993
- [Munson 03] Munson, J.: *Software Engineering Measurement*. Auerbach Publ., 2003
- [Neumann 95] Neumann, P.G.: *Computer Related Risks*. Addison-Wesley Verlag, 1995
- [Pomberger 93] Pomberger, G.; Blaschek, G.: *Software Engineering – Prototyping und objektorientierte Software-Entwicklung*. Carl Hanser Verlag, 1993
- [Sommerville 07] Sommerville, I.: *Software Engineering*. Pearson Education Publ., 2007
- [Together 2006] *Together-Toolbeschreibung*, <http://www.borland.com/de/products/together> (5.3.2007)
- [U5Tools 2006] *U5-Toolübersicht*, <http://www.u5tools.ch> (5.3.2007)
- [Vetter 95] Vetter, P.G.: *Progress: a Toolkit for Interactive Program Steering*. 1995
- [VPUML 2006] *VP-Entwicklungstoolumgebung*, www.visual-paradigm.com (5.3.2007)
- [Wallmüller 01] Wallmüller, E.: *Softwarequalitätsmanagement*. Carl Hanser Verlag, München Wien, 2001
- [Zuse 98] Zuse, H.: *A Framework of Software Measurement*. De Gruyter Verlag, Berlin New York, 1998

Selbständigkeitserklärung:

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

.....
HashemYazbek

Magdeburg im März 2007

Anhang