Cost Management

183.166 Management von Software Projekten
SS 2010

June 19, 2010

Inhaltsverzeichnis

1 Einleitung

Softwareprojekte haben den Ruf sehr häufig zu scheitern und Kosten fast permanent zu überschreiten. Schuld an diesem Ruf dürfte unter anderem der *Chaos Report* der Standish Group[9] aus dem Jahre 1994 sein, welcher nur von 16% erfolgreichen Softwareprojekten berichtet und sehr häufig zitiert wird. Die Reports der Standish Group stehen unter anderem in der Kritik wegen nicht offengelegter Methodik, schlechter Definition des Begriffs des Scheiterns u.a. Eine Studie aus dem Jahr 2008 geht von 11,5 bis 15,5% gescheiterten Projekten aus. Die vier häufigsten Gründe des Scheiterns werden wie folgt angegeben:

- Das höhere Management ist zuwenig eingebunden
- Zu viele Requirements und Änderungen des Projektfokuses
- Fehlen notwendiger Managementskills
- Überschreitung des Budgets. [5]

Diese Zahlen zeigen relativ deutlich die Wichtigkeit von adäquatem Management in Software Projekten. Fokus dieses Papers ist primär die Überschreitung des Budgets, wobei dieser Punkt nicht losgelöst von anderen Managementfaktoren gesehen werden kann. Cost Management hat primär die Aufgabe dafür zu sorgen, dass das Budget eines Projektes nicht überschritten wird. Dies beinhaltet die essentiellen Prozesse Resource Planning, Cost Estimating, Cost Budgeting und Cost Control. [12]

In diesem Paper geht es vorrangig um Softwareprojekte, da diese sich in der Regel anders als andere Projekte verhalten. Wir beginnen mit dem Thema *Cost Estimation*, welches sich mit der Schätzung der Kosten eines Projektes beschäftigt. DAbei stehen softwarebezogene Schätzmethoden wie COCOMO im Vordergrund. Mit Hilfe der hierbei ermittelten Schätzungen wird im Prozess *Cost Budgeting* ein umfassender Gesamtwert ermittelt, der alle anfallenden Kosten berücksichtigt (Cost Baseline). Schließlich wird während des laufenden Projektes der *Cost Control* Prozess iterativ durchgeführt, welcher laufend den aktuellen Status des Projektes überwacht und die vorher erstellten Dokumente aktualisiert.Ein wichtiges Artefakt dieses Prozesses ist der *Cost Management Plan*, welcher die Rahmenbedingungen des Prozesses definiert. Er stellt das Regelwerk dar welches in definierten Situationen abgearbeitet wird.

Die hier vorgestellten Methoden können dem Scheitern von Projekten wegen Kostenüberschreitung effektiv entgegenwirken. Doch selbst das Scheitern eines Projektes muss per se nicht nur negativ bewertet werden. Wenn ein gescheitertes Projekt korrekt analysiert wird um Verbesserungen für zukünftige Projekte daraus herzuleiten, kann man dem Scheitern eine positive Seite abgewinnen. Zukünftige Projekte profitieren dann auf jeden Fall von verbesserter Kostenschätzung und einem verbesserten Cost Control. [5]

Abbildung 1.1 zeigt eine Übersicht über die einzelnen Teilaspekte und Teilaufgaben des Cost Managements.

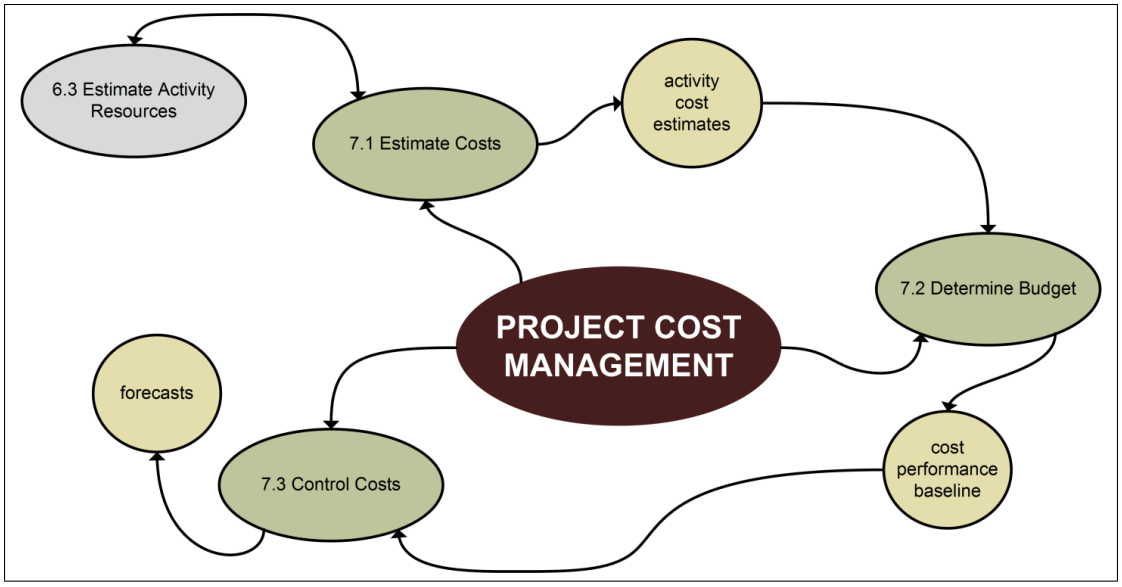

Abbildung 1.1: Cost Management Übersicht [21]

2 Cost Estimation

Cost Estimation (dt. Kostenabschätzung) ist ein zentraler Bestandteil des Cost Managements und ein nicht zu unterschätzender Punkt für eine erfolgreiche Abwicklung eines Projektes.

Die Kostenabschätzung eins Projekts ist von großer Bedeutung sowohl für den Kunden als auch für die Projektleitung. Einerseits führt eine zu geringe Abschätzung sehr schnell zu einem Überschreiten des Budgets, was in vielen Fällen den Abbruch oder die Modifikation des Projekts (Verzicht auf Funktionen) zur Folge hat. Dieses Problem der ungenauen oder fehlerhaften Kostenabschätzung spiegelt sich auch in verschiedenen Studien wieder, welche von einer durchschnittlichen Kostenüberschreitungen bei 60-80% der fertiggestellten Projekte berichten, wobei sich die Abweichungen im Durchschnitt auf 30-40% belaufen [15]. Auf der anderen Seite ist ein allzu großzügiger Kostenvoranschlag in einem freien Wettbewerb von Nachteil, da Konkurrenten günstigere Angebote machen können oder zu viele Ressourcen in das Projekt geleitet werden. Dementsprechend legen die verschiedenen Techniken Fokus auf unterschiedliche Kriterien. Als Beispiel kann hier die *Price-to-win*-Methode gesehen werden, bei welcher die Kostenabschätzung darauf ausgelegt wird, der Bestbieter in einem Wettbewerb zu sein [24, 22].

Laut [14] hat die Kostenabschätzung zwei primäre Aufgaben:

1. Feststellung der Ressourcen, welche für die Produktion, Verifizierung und Validierung des Softwareprodukts notwendig sind sowie das Management dieser Aktivitäten, und
2. Aufdeckung und Auflistung der Risiken und Unsicherheiten bei dieser Berechnung.

Daher darf die Kostenabschätzung nie unabhängig von der technischen Arbeitsaufgabe und dem Projektumfeld ablaufen und soll neben den Kosten für die Entwicklungsarbeit (welche meist den Großteil der Projektkosten ausmachen) auch Kosten für die Entwicklungsumgebung (wie Testumgebung, Softwaremanagement & Support, etc.) sowie *Nonlabor* Kosten (wie PCs, Training etc.) enthalten [14].

2.1 Größenabschätzung in Software Projekten

Viele der für die Kostenabschätzung verwendeten Methoden basieren auf einer Abschätzung, welche den zu erwarteten Aufwand für ein Projekt möglichst genau vorhersagen soll. Diese Aufwandsabschätzung (engl. Effort Estimation) kann dann in Kosten sowie Projektdauer umgerechnet werden. Das Ergebnis der meisten Abschätzungsmethoden wird in Personenmonate (PM) angegeben, welche die Arbeitsleistung einer Person in einem Monat repräsentiert. Diese Einheit hat den Vorteil, dass die relativ leicht in eine Kostenabschätzung integrierbar ist indem man die errechneten Personenmonate mit den Personalkosten der entsprechenden Berufsklasse pro Monat multipliziert. Es sollte erwähnt werden, dass, obwohl Aufwand und Kosten eng miteinander verbunden sind, diese nicht zwingend mit einfachen mathematischen Formeln umgerechnet werden können müssen. [24]

Bevor wir die einzelnen Abschätzungstechniken beschreiben machen wir einen Einschub in die Größenabschätzung von Software Projekten, da diese besonders bei modell-basierten und algorithmus-basierten Methoden benötigt wird.

Einer der wichtigsten Punkte bei vielen Modellen zur Kostenabschätzung ist die Größe des Projekts bzw. eine Abschätzung des Arbeitsaufwandes. [24] beschreibt hierfür fünf Modelle, auf welche wir nun etwas

näher eingehen. Laut [24] sind in der Praxis vor allem die ersten beiden, SLoC und Function Points, relevant. Weiters möchten wir noch auf ein zusätzliches Modell eingehen, welches in [17] beschrieben wird und im Abschätzungsmodell *Cocomo* Verwendung findet.

Source Lines of Code (SLoC) Source Lines of Code (SLoC, manchmal auch nur Lines of Code (LoC)) ist eine, von der verwendeten Programmiersprache abhängige Technik zur Größenbeschreibung eines Projekts. Dabei werden die Zeilen im ausgelieferten Programmcode gezählt, wobei natürlich Kommentare und Leerzeilen sowie, je nach Definition und Programmiersprache, Definitionen oder Kontrollkonstrukte nicht mitgezählt werden. Für eine Abschätzung der Programmgröße werden typischerweise Expertenschätzungen herangenommen, aus welchen mit Hilfe spezieller Methoden eine voraussichtliche Programmgröße berechnet wird. Zwei Beispiele für diese Methoden sind PERT (Program Evaluation and Review Technique) [24, 14] sowie eine statistische Auswertung mittels Monte-Carlo-Verfahren [14], auf welche wir hier jedoch nicht im Detail eingehen.

Function Points Function Points sind ein Versuch, den Umfang an Funktionalität eines Programms zu beschreiben. Dafür werden die einzelnen Funktionen eines Programms einer von fünf Klassen zugeordnet:

External Input Daten- oder Steuerungseingabe des Benutzers

External Output Ausgabe an den Benutzer

Internal Logical File Datenkonstrukte (Files), welche programmintern verwendet werden

External Interface File Datenkonstrukte (Files), welche ausgegeben oder an externe Programme weitergegeben werden.

External Inquiry Interaktiver Input / Input-Output-Interaktion

Zusätzlich zu diesem Typ wird für jede Funktionalität ein *complexity level* (simple, medium, complex) bestimmt, welches von der Anzahl der Datenelemente einer Funktionalität und der Anzahl der davon betroffenen Dateien abhängig ist. Mit Hilfe dieses Wertes wird dann ein *complexity weight* berechnet, wodurch am Ende durch Aufsummieren der *complexity weight* aller Funktionalitäten die *Unadjusted Function Points* berechnet werden. Dieser Wert beschreibt den Aufwand, den ein Projekt mit der gewünschten Funktionalität besitzt. [1, 3]

Dieser berechnete Wert kann direkt für Größenabschätzungen verwendet werden (z.B. bei *COCOMO*) oder als Ausgangsbasis für weitere Modifizierungen herangezogen werden. Für solch eine projektspezifische Adaption werden die *Unadjusted Function Points* mit *complexity factors* multipliziert. Der Vorteil von Function Points ist, dass sie anhand der *system requirement specification* bereits sehr früh in einem Projekt berechnet werden können. [24]

Extensions of Function Point Im Laufe der Zeit haben sich aufgrund neuer Anforderungen verschiedene Extensions zu Function Points entwickelt. Eine davon sind *Feature Points*, welche eine neue Klasse, *Algorithmen*, einführt. Diese Klasse beschreibt signifikante Berechnungsprobleme und ihre Algorithmen in einem Projekt. Dies ist vor allem für Software mit komplexen Rechenvorgängen und Algorithmen hilfreich. Andere Extensions sind *Full Function Points*, welche für Echtzeitanwendungen verwendet werden. [24]

Object Points *Object Points* verwenden einen ähnlichen Ansatz wie *Function Points* und deren Extensions, allerdings werden Objekte auf anderer Ebene erfasst. *Object Points* beziehen sich dabei nicht auf Objekte hinsichtlich objekt-orientierter Programmierung, sondern auf Objekte, welche als Basis einer Applikation dienen. Um eine Größenabschätzung durchzuführen wird zuerst die Anzahl von *Screens*, *Reports* und *3GL* (third generation programming language) Komponenten geschätzt und anhand von drei Komplexitätsgraden bewertet. *3GL* Komponenten entsprechen wiederverwendbare Softwareteilen, wie zum Beispiel Klassen, in objekt-orientierten Sprachen. Anschließend werden die bewerteten Objekte anhand einer vorgegebenen Tabelle gewichtet und die Größenabschätzung abgeschloßen. Falls die Anforderungen ausreichend detailreich sind können Größenabschätzung mittels *Object Points* eventuell einfacher und schneller als mit *Function Points* durchgeführt werden. [17]

Application Points *Application Points* sind eine Erweiterung von *Object Points* und wurden umbenannt um eine Verwechslung mit objekt-orientierter Programmierung zu vermeiden. Weiters enthalten *Application Points* mehr Informationen über Charakteristiken, welche die Produktivität beeinflussen um eine genauere Größenabschätzung zu ermöglichen. Dabei wird unter anderem die Erfahrung der Entwickler und deren Fähigkeit im Umgang mit der Entwicklungsumgebung mit einbezogen. *Application Points* werden häufig in Zusammenhang mit *Cocomo II* verwendet, auf welches wir in 2.2.1 näher eingehen. [17]

Software Science Dieses Verfahren besteht aus den zwei Metriken *Code Länge* und *Volumen*. Die *Code Länge* besteht dabei aus den Summanden *N1* und *N2*. *N1* entspricht der Anzahl der Operator-Aufrufe eines Programms, während *N2* durch die Gesamtzahl der Operanden bestimmt wird. Die dementsprechend errechnete *Code Länge* dient als Maß für den Quellcode des Programms. Das *Volumen* entspricht dem benötigten Speicherplatz und wird mit folgender Formel bestimmt:

$$V = N * log(n1 + n2)$$

N entspricht der *Code Länge* und *n1*, *n2* entsprechen der Anzahl der unterschiedlichen Operatoren bzw. der unterschiedlichen Operanden, welche im Programm Verwendung finden. Das *Software Science* Verfahren hat in den letzten Jahren aufgrund von Unstimmigkeiten bezüglich der zugrundeliegenden Theorie an Bedeutung und Unterstützung verloren. [24]

Andere Klassifikationen Andere Klassifikationen unterteilen in *size-related Metrics* und *function-related Metrics*, wobei erstere Gruppe Methoden wie LoC, die Anzahl der ausgelieferten Objekte oder Seitenanzahl der Systemdokumentation enthält. Unter function-related Metrics werden Techniken wie Function Points und Object Points klassifiziert. [22]

Die beschriebenen Metriken sind teilweise kompatibel und es existieren Umrechnungsformeln, welche eine Konvertierung zwischen den Berechnungsarten zulassen. Für das COCOMO II - Modell können z.B. sowohl Function Points als auch LoC verwendet werden, wobei am Ende die Function Points mit Hilfe der verwendeten Programmiersprache und einem dementsprechenden Umrechnungsschlüssel in LoC umgerechnet werden [3].

2.2 Abschätzungsmethoden

In der Literatur werden verschiedene Kategorisierungen für Abschätzungsmethoden verwendet. Während [14] in vier Kategorien unterteilt (*Historical Analogy*, *Expert Judgment*, *Modelle*, und *Rules-of-Thumb*)

benutzt [24] nur eine Unterteilung in *algorithmic* und *non-algorithmic* Techniken. Die letztere Einteilung beruht darauf, ob der Methode mathematischen Berechnungen (mit einem Algorithmus) zugrunde liegen. Andere Klassifizierungen wiederum sprechen von *Parameter-Modellen* (parametric models), *Experten-basierte Techniken* (expertise-based techniques), *lern-orientierten Techniken* (learning-oriented techniques), *dynamischen Modellen* (dynamics-based models), *regressions-basierten Modellen* (regression-based models) und *composite-Bayesian techniques for integrating expertisebased and regression-based models* [4].

Diese Einteilungen stehen jedoch nicht zwingend in Widerspruch zueinander. So wird in [15] *Expert consultation*, *Intuition and experience* sowie *Analogy* zu *Experten-basierte Techniken* zusammengefasst, was auch die in [14] beschriebenen Klassifizierungen *Historical Analogy* und *Expert Judgment* beinhaltet.

Da aufgrund der verschiedenen Definitionen eine genaue Klassifizierung nicht möglich ist, werden wir nun einige ausgewählte Methoden beschreiben.

2.2.1 Modellbasierte Techniken

In diesem Abschnitt möchten wir etwas näher auf mehrere ausgewählte Abschätzungsmethoden eingehen. Diese Methoden verwenden dabei ein Schätzungsmodell, welches auf mathematischen Formeln basiert, die durch Erfahrungswerte angepasst wurden. Als Erfahrungswerte wurden Informationen über frühere Projekte verwendet. Nach [4] eignen sich modellbasierte Techniken gut für projektspezifische Analysen und Finanzierungen, aufgrund deren Orientierung an bereits durchgeführten Projekten. Die größten Probleme verursachen Ereignisse und Situationen, welche in keinen vorherigen Projekten zuvor aufgetreten sind.

Cocomo *Cocomo* ist ein Modell, welches aus Erfahrungsdaten von einer großen Zahl von Projekten erstellt wurde. Dabei wurden die Erfahrungsdaten analysiert um Formeln zu erstellen, mit welchen eine Aufwandsberechnung anhand von Anforderungen ermöglicht wurde. *Cocomo* wurde bereits im Jahr 1981 veröffentlicht und nach Modifikationen in eine neue Version *Cocomo II* überführt wurde. [22]

In der ersten Version von *Cocomo* wurden drei unterschiedliche, detailreiche Ebenen verwendet um eine Kostenabschätzung durchzuführen. In der ersten Ebene wird das Projekt einer grundlegenden Abschätzung unterzogen. Die zweite Ebene modifiziert das Ergebnis anhand von einer Zahl von Projekt- und Prozess-Faktoren und die dritte Ebene beinhaltet die detailreichste Kostenabschätzung, welche eine Abschätzung für unterschiedliche Phasen im Projektverlauf ermöglichen. In der ersten Version von *Cocomo* wurde davon ausgegangen, dass das Projekt anhand eines Wasserfallprozesses unter Verwendung von imperativen Programmiersprachen entwickelt wurde. Die Softwareenwicklung hat sich allerdings in der Zwischenzeit massiv verändert, wodurch eine grundlegende Anpassung des Modells nötig wurde.

Dies wurde mit der neuen Version *Cocomo II* in [3] verwirklicht. *Cocomo II* unterstützt unterschiedliche Entwicklungsmethoden wie *Prototyping*, *Komponentenorientierte Entwicklung* und *Datenbankprogrammierung*. Weiters werden agile Softwareprozesse berücksichtigt. Im Unterschied zu den drei Ebenen aus dem ursprünglichen *Cocomo* Modell wird in *Cocomo II* zwischen folgenden vier Untermodellen differenziert:

Application-composition Model Mit dem *application-composition* Modell werden Schätzungen für Projekte durchgeführt, welche durch Verwendung von Datenbankprogrammierung, Skript-basierter Entwicklung oder wiederverwendbaren Komponenten entwickelt werden. Für die Größenabschätzung werden dabei *Application Points* verwendet. Bei der Abschätzung wird sowohl der Aufwand

der einzelnen *Application Points*, als auch Faktoren, welche die Produktivität der Entwickler beeinflußen berücksichtigt. Weiters beeinflußt die Wiederverwendbarkeit des Codes die resultierende Formel. [22]

Early Design Model Sobald die Projektanforderungen geklärt sind und genug Informationen über die Projektentwicklung vorhanden sind, können Schätzungen mit dem *early design* Modell durchgeführt werden. Da es sich hier allerdings noch um eine frühe Phase im Projektverlauf handelt, müssen häufig bei Unsicherheiten Annahmen getroffen werden. Schätzungen mit dem *early design* Modell werden daher meistens verwendet um unterschiedliche Entwicklungsmöglichkeiten zu evaluieren und zu vergleichen. Die Schätzung des Aufwands liegt folgender Basisformel zugrunde.

$$Effort = A * Size^B * M$$

Die Konstante A besitzt nach [3] den Wert *2,94*. Nachdem die *Function Points* berechnet wurden, werden diese je nach gewählter Programmiersprache in *KSLoC* (tausend SLoC) umgerechnet. Der Faktor M besteht aus weiteren sieben Faktoren, welche den Projektverlauf beeinflussen können. Diese Faktoren umschreiben unter anderem die Komplexität und Anforderungen nach Verlässlichkeit des Programms, sowie Anforderungen an die Wiederverwendbarkeit und Plattformunabhängigkeit. An dieser Stelle wird allerdings nicht mehr genauer auf die Erhebung der einzelnen Faktoren eingegangen.

Zum Berechnen von M und daher zum bestimmen der sieben Faktoren, müssen die vorgegebenen Attribute der Faktoren auf einer Skala von eins bis sechs bewertet werden. Nachdem alle Attribute bewertet wurden, wird dem Faktor anhand einer vorgegebenen Tabelle ein Wert zugewiesen. Wenn alle Faktoren einen Wert erhalten haben, wird deren Produkt berechnet, welches M entspricht.

Mit dem Exponent B werden die Einflüsse der Projekt Komplexität auf das Projekt abgebildet. Im Unterschied zur ersten Version von *Cocomo*, bei der für den Exponenten nur drei Werte möglich waren, wird in *Cocomo II* der Exponent anhand von folgenden fünf sogenannten *Scale Factors* berechnet.

Precedentedness (PREC) Dieser Faktor bestimmt in welchem Maß das Projekt Ähnlichkeit mit vorherigen Projekten besitzt.

Development flexibility (FLEX) Mit diesem Faktor wird die Flexibilität im Entwicklungsprozess bestimmt.

Architecture/risk resolution (RESL) *Architecture/risk resolution* gibt sowohl an, mit welchem Umfang eine Risikoanalyse durchgeführt wurde, als auch welches Risiko die gewählte Architektur bürgt.

Team cohesion (TEAM) Mit *team cohesion* wird die Produktivität der Entwickler als Team bewertet.

Process maturity (PMAT) Hier wird der Entwicklungsprozess des Projekts bewertet.

Die Faktoren *PREC*, *FLEX*, *RESL* und *TEAM* werden durch das Bewerten von Attributen auf einer Skala von eins bis sechs bestimmt. Beim Erfassen der Attribute, wird wiederum ein Wert zwischen eins und sechs für den jeweiligen Faktor bestimmt.

Für den Faktor *PMAT* stehen zwei verschiede Bewertungsmöglichkeiten zur Verfügung. Eine Möglichkeit verwendet dabei Ergebnisse aus einer *Capability Maturity Model* Analyse und die zweite Variante berechnet den Faktor aufgrund von 18 *Key Process Areas*. Allerdings wird hier aufgrund des Umfangs nicht näher auf die Methoden zum Berechnen des Faktors *PMAT* eingegangen.

Nachdem nun jeder *Scale Factor* jeweils eine Bewertung erhalten hat, werden die zugeordneten Werte der Faktoren nun summiert und das Ergebnis durch 100 dividiert und anschließend eine Kon-

stante C hinzuaddiert. Die Formel, die nun den Faktor B berechnet ist folgendermaßen definiert:

$$B = C + 0,01 * \sum_{i=1}^{5} SF_i$$

Die summierten Variablen SF entsprechen den bewerteten *Scale Factors* und die Konstante C besitzt nach [3] den Wert *0.91*.

Reuse Model *Cocomo II* berücksichtigt im Unterschied zur ersten Version von *Cocomo* auch die Wiederverwendbarkeit von Code. Mit dem *Reuse* Modell wird der Aufwand geschätzt, der durch die Integration des Codes zustande kommt. Laut [22] werden für die Berechnung zwei unterschiedliche Arten von Code betrachtet: *Black-box Code* und *White-box Code*. Bei *Black-box Code* handelt es sich um Code, welcher ohne Änderung oder aufwendige Integrationsarbeiten verwendet werden kann. *White-box Code* muss hingegen angepasst und integriert werden. Weiters kann mit dem *Reuse* Modell auch der Integrationsaufwand von generiertem Code geschätzt werden. [22]

Post-architecture Model Das *Post-architecture* Modell ist laut [22] das detailreichste Modell von den *Cocomo II* Modellen und wird verwendet wenn ein früher Entwurf der Architektur zur Verfügung steht. Grundsätzlich verwendet das *post-architecture Modell* die selbe Grundformel, wie das *early design* Modell, allerdings können die Faktoren genauer berechnet werden, da nun mehr Informationen zur Verfügung stehen.

Die Berechnung der *SLoC* beinhaltet nun auch eine Unterscheidung zwischen neuen Code, wiederverwendeten Code und Code, welcher aufgrund von geänderten Anforderungen erstellt wird.

Für die Berechnung des Faktors M müssen 17 Attribute bestimmt werden, welche in vier unterschiedliche Klassen eingeteilt werden. Dabei werden die Attribute unterschieden, ob sie Eigenschaften des Projekts, des Personals, der Plattform oder des Projekts sind.

Der Faktor B wird auf die selbe Weise berechnet, wie im *early design* Modell.

Nach [22] ist *Cocomo II* ein sehr komplexes und umfangreiches Modell, welches die Erfahrung der Entwickler wiederspiegelt. Es wäre wünscheswert, wenn die Parameter des Modells an die eigenen Erfahrungen aus vergangenen Projekten angepasst werden könnte. Da Informationen, welche ein solches Anpassen des Modells ermöglichen würden, nur von wenigen Betrieben gesammelt wurden, müssen die Anpassungen von den bereits vorgegebenen Parametern ausgehen. Allerdings können Anwender nur schwer nachprüfen, ob die verwendeten Informationen aus vorherigen Projekten, auch mit dem eigenen Umfeld korrelieren. Anpassungen des Modells an eigene Anforderungen sind nach [22] in Folge dessen sehr aufwendig und werden daher eher nicht durchgeführt.

Checkpoint *Checkpoint* ist ein wissensbasiertes Schätzungsmodell, welches von dem Unternehmen *Software Productivity Research* (SPR) unter Capers Jones entwickelt wurde. Bei *Checkpoint* werden Einflüsse, welche auf die Qualität der Software oder Produktivität im Projekt einwirken, in vier Bereiche unterteilt: *Technology*, *Development process*, *Environment* und *People Management*. Für die Größenabschätzung des Projektes werden *Function Points* verwendet. [4]

Das Modell bietet folgende drei Methoden, welche zur Unterstützung des Entwicklungsprozesses dienen:

Estimation *Checkpoint* ermöglicht eine Kostenabschätzung anhand von vier unterschiedlichen Detaillierungsgrade.

Measurement Mit dem Modell können Projekte auf deren Durchführbarkeit getestet werden und gute Vorgehensweisen ermittelt werden.

Assessment Projekte können mit einer Reihe von Standardprojekten, welche in der *Checkpoint* Datenbank vorhanden sind, verglichen werden und daraus Stärken und Schwächen des Entwicklungsprozesses ermittelt werden.

PRICE-S Laut [11] wurde *PRICE-S* 1977 entwickelt und als eines der ersten kommerziellen Software Estimation Modelle veröffentlicht. Für das Modell wurden zuvor viele Projekte analysiert und einzelne Charakteristiken extrahiert, welche ein Justieren der Schätzungen ermöglichen. Für die Größenabschätzung werden *SLoC* verwendet, welche entweder direkt oder anhand von *Function Points* vorgegeben werden. *PRICE-S* beinhaltet nach [4] die folgenden drei Untermodelle:

Aquisition Submodel Dieses Modell dient zur Kostenschätzung und beinhaltet bereits eine Unterstützung für aktuelle Entwicklungsmethoden, wie zum Beispiel *code generation*, *prototyping*, und weitere Methoden.

Sizing Submodel Mit diesem Modell wird die Größe des Softwareprojekts geschätzt. Hierbei können unterschiedliche Größenabschätzungen verwendet werden.

Life-cycle Cost Submodel Kosten, welche durch Wartung und Instandhaltung eines Softwareprojekts entstehen, können mit diesem Modell erfasst verwenden.

PRICE-S wird von *PRICE Systems* ständig weiterentwickelt und an neue Randbegingungen aus der Softwareentwicklung angepasst und erweitert.

SEER-SEM *SEER-SEM* (System Evaluation and Estimation of Resources - Software Estimation Model) ist nach [11] ein properitäres Modell, welches auf einer umfassenden Wissensdatenbank, bestehend aus vielen abgeschlossenen Projekten, basiert. Zum Durchführen einer Kostenschätzung werden nur sehr wenig Informationen benötigt. Dabei muss die Plattform, die Art des Programms, die Entwicklungsmethoden und die verwendeten Entwicklungsstandards angegeben werden.

Nach [4] unterstützt *SEER-SEM* eine Vielfalt von Umgebungs- und Applikationskonfigurationen. Weiters wird eine Unterstützung für eine Vielzahl von aktuellen Entwicklungsmethoden und -prozessen angeboten.

2.2.2 Erfahrungsbasierte Techniken

Erfahrungsbasierte Techniken sind Methoden, welche aus den Erfahrungen und Daten bestehender Projekte Schlüsse auf neue Aufgaben ermöglichen.

Einer der naheliegensten und einfachsten Techniken ist die Abschätzung anhand von bereits abgeschlossenen Projekten (engl. Historical Analogy [14] oder Analogy costing [24]), welche dem neuen Projekt sehr ähnlich sind. Dies kann sowohl das gesamte Projekt also auch einzelne Teilprojekte umfassen, wobei ersteres den Vorteil hat, dass alle verschiedenen Kostenstellen berücksichtigt werden während bei einem Vergleich der einzelnen Teilprojekte eine genauere Adaption der vorherigen Kosten auf das neue Projekt ermöglicht wird. Die Stärke dieser Methode ist, dass sie auf eigenen, aktuellen Projekterfahrungen beruht. Dem steht gegenüber dass eine Ähnlichkeit zwischen dem Projekt nicht zwingend einen gleichen Arbeitsaufwand bedeuten muss. [24, 22]

Abschließend sei noch erwähnt, dass erfahrungsbasierte Techniken auch in Kombination mit anderen Techniken (wie z.B. modellbasierten Techniken) eingesetzt werden können. Empfehlung dazu sowie weitere Informationen und eine Evaluierung finden sich in [20].

2.2.3 Expertenbasierte Techniken

Experten-basierte Techniken (engl. Expert judgment [22, 24, 14] oder Expertise-Based Techniques [4]) umfassen ein breites Spektrum an Methoden, welche sich weniger auf mathematische Modelle sondern auf eine Abschätzung von einem oder mehreren Experten verlassen, die wiederum andere Techniken anwenden [24] oder einfach mit ihrer Erfahrung schätzen [4]. Der große Vorteil dieser Methoden sind, dass sie sich nicht nur auf historische Daten verlassen sondern versuchen, zukünftige Entwicklungen zu berücksichtigen sowie unerwartete Risiken miteinzuberechnen. Somit eignen sich diese Methoden vor allem dann, wenn keine empirischen Vergleichsdaten zu ähnlichen Projekten existieren [4].

Delphi-Befragung Die Delphi-Technik ist nach dem griechischen Orakel in Delphi benannt und wird verwendet um mit Hilfe von mehreren Experten eine gemeinsame Vorhersage zu treffen. Als ersten Arbeitsschritt müssen die Experten unabhängig voneinander eine Abschätzung abgeben, welche gesammelt und zusammengefasst wird. Dieses Ergebnis wird dann wiederum an die Experten zurückgegeben, die unter Rücksichtnahme dieser Zusammenfassung eine neue Schätzung abgeben sollen. Die resultierenden Ergebnisse werden dann wieder zusammengefasst und an die Experten zurückgegeben. Dieser Vorgang wiederholt sich so oft, bis ein Konsens (Schätzungen innerhalb eines Toleranzbereichs) gefunden wurde. [4, 24]

Eine Modifikation dieser Prozedur ist die sogenannte *Breitband-Delphi-Methode*, bei welcher die Experten zwischen den einzelnen Iterationen in einer Gruppendiskussion aufeinander treffen und erst danach ihre Schätzung revidieren. [4, 2]

Work Breakdown Structure Work Breakdown Structure (WBS) ist eine hierarchische Strukturierung eins Projekts, wobei das Gesamtprojekt in kleinere Unterprojekte aufgeteilt wird. Für diese Unterteilung werden meist Experten herangezogen, welche sowohl bei der Aufteilung als auch bei der Abschätzung der Teilprojekte helfen. Eine Software-WBS besteht aus zwei Hierarchien - eine repräsentiert die Software selber und die andere die zur Erstellung notwendigen Teilaufgaben. Ein zusätzlicher Vorteil der WBS ist, dass die genaue Aufgliederung eine einfache Dokumentation der tatsächlichen Kosten und Arbeitszeit ermöglicht. Dies ist im Sinne des *Reportings* und *Cost Controllings* von immensen Vorteil. [4]

PERT PERT steht für *Program Evaluation and Review Technique* und ist eine Abschätzungsmethode, die sowohl für die Programmgröße [14] als auch für die Kostenabschätzung [24] verwendet werden kann. Dabei wird von einer Verteilung der Experten-Abschätzungen ausgegangen, welche in c_{least} (Minimalkosten), c_{likely} (erwartete Kosten) und c_{most} (Maximalkosten) resultieren. Die Abschätzung c_{mean} wird danach mit folgender Formel berechnet:

$$c_{mean} = \frac{c_{least} + c_{likely} * 4 + c_{most}}{6}$$

2.2.4 Andere Techniken

Parkinson's Law Parkinson's Law besagt, dass jede Aufgabe sich so ausweitet, dass sie die verfügbare Zeit komplett ausnützt. Somit entsprechen die Kosten den verfügbaren Ressourcen und keinen objektiven Kriterien. [22]

Dieses Gesetz führt auf den ersten Blick zu einem Widerspruch mit den bisher vorgestellten Abschätzungsmethoden. Man könnte nach diesem Gesetz einfach sagen, dass jegliche Aufgabe eines Projektes in der

abgeschätzten Zeit mit den budgetierten Mitteln abgeschlossen werden kann. Dies würde jedoch genaue Abschätzungsmethoden überflüssig machen.

Bei genauerer Analyse kommt man jedoch zu dem Schluss, dass hier hauptsächlich das Problem von Pufferzeiten relevant ist: Häufig ist es sinnvoll, Aufgaben mit gewissen Pufferzeiten zu planen um mögliche Probleme besser mit anderen Aufgaben des Projektes koordinieren zu können. In der Regel werden nicht alle möglichen Probleme eintreten, daher sollte diese Pufferzeit am Ende der Aufgabe frei bleiben. Parkinson's Law besagt aber, dass dies nicht der Fall ist. Ein echtes Problem für das Projekt stellt dieses Verhalten nicht dar, so lange keine andere Aufgabe mit Problemen behaftet ist, was sehr unrealistisch ist. [19]

Es ist wichtig, dass der Projektmanager dieses Phänomen kennt, um dem entgegenwirken zu können. Die freien Zeitpuffer vorhergehender Aufgaben können besser bei großen Problemen, die bei anderen Aufgaben auftreten, eingesetzt werden. Die Gründe für dieses Verhalten dürften eher psychologisch sein. Eine projektplanerische Methode um Parkinson's Law entgegenzuwirken ist die Critical Chain Methode [19], die jegliche Pufferzeiten aus der Projektplanung streicht, Abhängigkeiten dafür aber vorzieht. In der Theorie wird so der Arbeitsaufwand verringert unbenommen der Möglichkeit verspätete Tasks bei Problemen auszudehnen ohne andere Tasks zu blockieren.

Price-to-win Hierbei wird die Kostenschätzung alleine vom Kunden übernommen. Was der Kunde bereit ist zu bezahlen entspricht der Kostenschätzung des Projektes. [22] Dieser Ansatz ist hauptsächlich sinnvoll um ein Projekt während einer Ausschreibung zu gewinnen, schadet jedoch dem Projektverlauf, da hierbei zu erwarten ist, dass das Projekt nicht mit den geschätzten Kosten beendet werden kann. [24]

Bottom-up Bei der Bottom-up-Methode muss das Projekt bereits soweit geplant sein, dass ein Komponentendiagramm existiert, dessen Komponenten im wesentlichen dem fertigen System entsprechen, das Grobdesign muss also abgeschlossen werden. Der Bottom-up-Ansatz nimmt anschließend die einzelnen Komponenten und schätzt die Kosten für jede dieser Komponenten. Die einzelnen Kostenschätzungen werden anschließend aufsummiert und ergeben die Gesamtkostenschätzung. [24]

Top-down Dieser Ansatz ist das Gegenteil des Bottom-up Ansatzes. Zuerst wird die Gesamtsumme festgelegt, anschließend wird die Summe auf die einzelnen Komponenten verteilt. Die Gesamtkosten werden aus den Parametern des Projektes ermittelt. [24]

SoftCost-Ada Einen eigenständigen Ansatz für Ada-Projekte verfolgt das SoftCost-Ada Modell. Hierbei werden die Eigenheiten der Programmiersprache Ada berücksichtigt um ein möglichst akkurates Kostenschätzverfahren zu ermöglichen. Vor allem der besonders guten Software-Reuse Fähigkeit von Ada wurde hierbei Rechnung getragen. Das Modell ging aus der Analyse von 75 Ada Projekten hervor. [18]

Artificial Intelligence Bei der Betrachtung bestehender Kostenschätzverfahren auf Basis von algorithmischen Modellen ist häufig zu beobachten, dass die Schätzungen zu hoch ausfallen. Konkret 57% im Durchschnitt. Die Abweichung kann mit Hilfe des MARE (Mean Abolute Relative Error) gemessen werden, welcher eine Anzahl von Schätzungen mit den wirklichen Kosten vergleicht. Der Wert MRE berücksichtigt dabei ob die Schätzung zu hoch oder zu niedrig war. Mit diesen Kennzahlen lassen sich Schätzmethoden gut vergleichen. [7]

Künstliche Neuronale Netzwerke könnten die Nachteile bestehender Methoden kompensieren, da sie aus abgeschlossenen Projekten lernen können. Finnie und Wittig führten diesen Ansatz durch in dem

sie ein KNN mit den errechneten Daten des Tools SPQR/20 auf Basis von Zufallswerten trainierten. Die Ergebnisse fielen sehr gut aus, 75% der Schätzungen liegen innerhalb eines 25% Spektrums der wirklichen Kosten. [7]

3 Cost Bugeting

Cost Budgeting ist der Bereich des Kostenmanagement, welcher sich zwischen Cost Estimation und Cost Control befindet. Während sich Cost Estimation mit der allgemeinen Abschätzung von Kosten von Aktivitäten befasst, bringt Cost Budgeting die real-praktische Zeitkomponente des Projektes ins Spiel. Die geschätzten Kosten werden zusammengefasst und bilden somit den fundamentalen Output, die Cost Baseline. Im Vergleich dazu beschäftigt sich Cost Control mit der aktiven Beinflussung des Projektbudgets, basierend u.a. auf Erkenntnissen aus dem Cost Budgeting.

Im Folgenden werden erst die **Inputs** des Cost Budgeting erläutert. Daraufhin werden verschiedene **Werkzeuge & Techniken** zum Cost Budgeting aufgezählt und erklärt. Letzlich wird unter **Outputs** beschrieben, welche Ergebnisse das Cost Budgeting anhand der Inputs und der angewandten Werkzeuge & Techniken erzeugt. Das abschliessende Fazit des Kapitels Cost Budgeting resumiert die beschriebenen Punkte.

3.1 Inputs

Das Cost Budgeting verarbeitet Großteils die gleichen Inputs wie die Cost Estimation, nur werden diese in einem anderen Kontext betrachtet:

1. der **Projektplan (Project Schedule)** gibt vor, zu welchen Projektzeitpunkt welche Investitionen getätigt werden müssen;
2. das **Pflichtenheft (Project Scope Statement)** definiert Regeln und Einschränkungen, wie zB. dass Monatliche Budget-Limit;
3. der **WBS, WBS Dictionary, Activity Cost Estimates, Activity Cost Estimates Supporting Detail** welche ebenfalls bereits in Kapitel 2, Cost Estimation ab Seite 4 behandelt wurden;
4. der **Ressourcenkalender (Resource Calendar)** gibt vor, zu welchem Zeitpunkt welche Ressource verfügbar ist. Darunter fallen zB Mitarbeiter, Räume, Maschinen;
5. die **Verträge (Contracts)** definieren weitere Vorgaben wie verhandelte Preise und betreffende Zahlungfristen, welche einzuhalten sind;
6. der **Cost Management Plan** validiert das Cost Budgeting, siehe Kapitel 5 ab Seite 23.

3.2 Werkzeuge & Techniken

Um die verschiedenen Inputs des Cost Budgeting in die gewünschten Outputs überzuführen, werden folgende Werkzeuge & Techniken angewandt:

3.2.1 Cost Aggregation

Aggregation beschreibt im Allgemeinen die Kombination unterschiedlicher Elemente in einer Gruppe. In der Kostenrechnung sind solche Elemente zB vergangene und zukünftige Aktivitäten im Projekt. Die Aggregation fasst die Aktivitäten statistisch zusammen, wie zB der Gesamtsumme oder dem Durchschnitt der Einzelwerte. [6, S. 44]

Im konkreten Fall der Cost Aggregation werden die Kosten der einzelnen Aktivitäten der WBS entnommen und zu kumulativ zu höheren Ebenen zusammengefasst bis das Gesamtbudget errechnet werden kann. Es wird also ein "Bottom-UpVerfahren angewendet. D.h. Aktivitäten werden zu Arbeitspacketen akkumuliert, diese fliessen dann zu den entsprechenden Projektkonten zusammen und bilden gesamt die Projektkosten. Vereint mit der Sicherheitsreserve (Contingency Reserve), der Cost Baseline und der Management Reserve ergibt sich mitsamt der Projektkosten letztendlich das gesamte Kostenbudget. [10, S. 207] [13, S. 165ff]

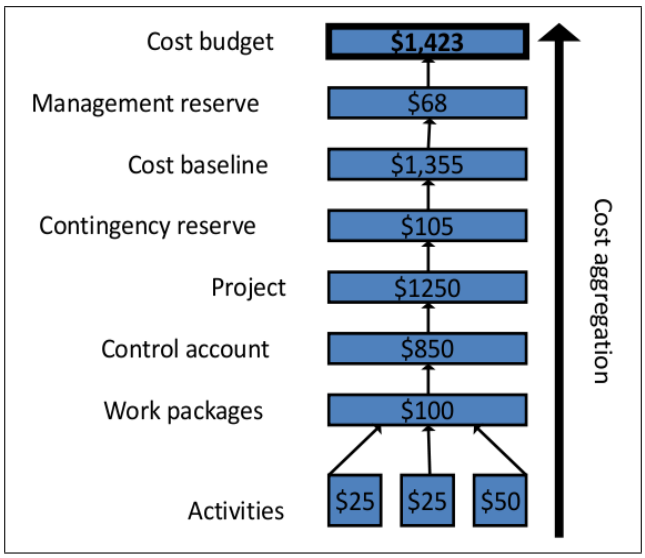

Abbildung 3.1: Cost Aggregation [13]

3.2.2 Reserve Analysis

Geschätzte Kosten und Realkosten divergieren in der Regel. Die Reserve Analysis beschäftigt sich mit aktiven Einplanung sogenannter **Known Unknowns** in das Projektbudget. Es wird versucht, möglichst genau abzuschätzen welche Art und Menge von Reserven im Projektverlauf gebraucht werden bzw. gebraucht werden könnten um die Abweichung von Real- zu Plankosten zu kompensieren. Anhand dieser Analyse werden Budget Reserven und eine sogenannte Contingengy Reserve als Teil des Projektbudgets festgelegt.

Die Abweichung zwischen Plan- und Realwerten der Kosten kann aus technischer Sicht an Risiken geknüpft werden. Man unterscheidet prinzipiell zwischen geringfügigen Schätzungsabweichungen und groben Abweichungen aufgrund des Eintreten eines Risikoereignis. Aus diesem Grund werden die folgenden 3 Techniken angewandt, um die unterschiedlichen Arten finanzieller Unsicherheit zu adressieren:

Budget Reserven für geringfügige Schätzungsabweichungen Auf Basis der Cost Estimates einzelner Aktivitäten wird individuell anhand deren Risiko ein Puffer definiert, die sogenannte Budget Allowance. Zum Beispiel ist der Preis einer geplanten Anschaffung aufgrund des Vorliegens mehrerer

Anbote von Lieferanten noch nicht gesichert, somit wird zu dem erwarteten Preis eine mögliche Varianzspanne als Budget Reserve eingeplant. [21]

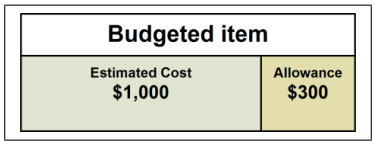

Abbildung 3.2: Budget Reserve [21, S. 197]

Contingengy Reserven für potentielle Risiken Bei jedem Projekt ist mit potentiellen Risiken zu Rechnen. Spezifisch für einzelne, identifizierte Risiken werden die Bewertungen von Wahrscheinlichkeit und Einfluss aus der Risikoanalyse herangezogen um sogenannte Contingency Reserven zu bilden. Die Summe aller Risiko-spezifischen Reserven bildet die gesamte Contingency Reserve. [21]

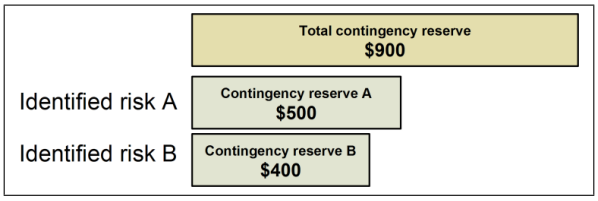

Abbildung 3.3: Contingency Reserve [21, S. 197]

Management Contingency Reserve für unvorhersehbare Risiken Die Management Contingency Reserve ist eine allgemeine, nicht dem Projektbudget zugeordnete Reserve auf Managementebene und befindet sich daher prinzipiell ausserhalb der Projektplanung. Diese Reserve wird vom Unternehmen für unvorhersehbare Risken des operativen Geschäftes gebildet (sog. **Unknown Unknowns**) und kann im Notfall vom Projekt zur Verwendung beantragt werden.

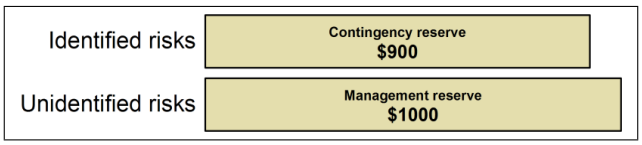

Abbildung 3.4: Management Contingency Reserve [21, S. 197]

Eine besondere Herausforderung bei der Reserve Analysis ist, dass die Reserve möglichst die Gratwanderung zwischen ausreichender Abdeckungssicherheit eventuell auftretender Risiken und Sparsamkeit optimal bewerkstelligt. [21, S. 196ff]

3.2.3 Parametric Estimation

Die Grundlagen der Parametric Estimation werden in Kapitel 2.2.1 auf Seite 7 erläutert. Die Genauigkeit der Parametric Estimation variiert anhand verschiedener Faktoren. Eine entscheidende Rolle spielt hierbei die Qualität des verwendeten, statistischen Materials, sowie die Skalierbarkeit der verwendeten Methodik auf kleine und große Projekte.

Eine weitere Herausforderung ist die unterschiedliche Quantifizierbarkeit von Parametern. Es ist zB vergleichsweise einfach, die veranschlagten Kosten pro Quadratmeter mit der realen Größe des zu nutzenden Objektes zu quantifizieren. Andererseits ist zum Beispiel die im Softwarebereich verwendete Ein-

heit Lines of Code (LOC) nur schwer vergleichbar zu skalieren, weshalb oft versucht wird, eine bessere Genauigkeit mittels Software Function Points zu erreichen. [23]

3.2.4 Funding Limit Reconciliation

Die Funding Limit Reconciliation beschreibt den dynamischen Anpassungsmechanismus, welcher die Anforderung nach Finanzierung seitens des Projektes mit den Finanzierungsvorgaben und -möglichkeiten des Unternehmens vereint. D.h. die Kostenbudgetplanung des Projektes muss laufend hinsichtlich monetärer Rahmenbedingungen adaptiert werden.

Aus Verträgen (siehe Inputs) ergeben sich verschiedene Rahmenbedingungen, welche zB definieren wieviel Geld pro Monat ausgegeben werden darf, während die Project Funding Requirements definieren, zu welchem Zeitpunkt wieviel Geld benötigt wird. Im laufenden Projektbetrieb und der mit den Requirements verbundenen laufenden Validierung können sich Konflikte zwischen Geldbedarf der geplanten Aktivitäten und der Requirements ergeben. In diesem Fall müssen geplante Aufgaben nach hinten verschoben werden, was wiederum Einfluss auf den Projektplan hat und eine besondere Herausforderung darstellt - muss ja nun zB sichergestellt werden, dass bestimmte Resourcen zu dem neudefinierten Zeitpunkt verfügbar sind und dergleichen.

3.3 Outputs

3.3.1 Cost Baseline

Die Cost Baseline visualisiert die Projektkosten über den Projektverlauf als Integral. Sie stellt aufgrund ihrer Aussagekraft den wichtigsten Output des Cost Budgeting dar.

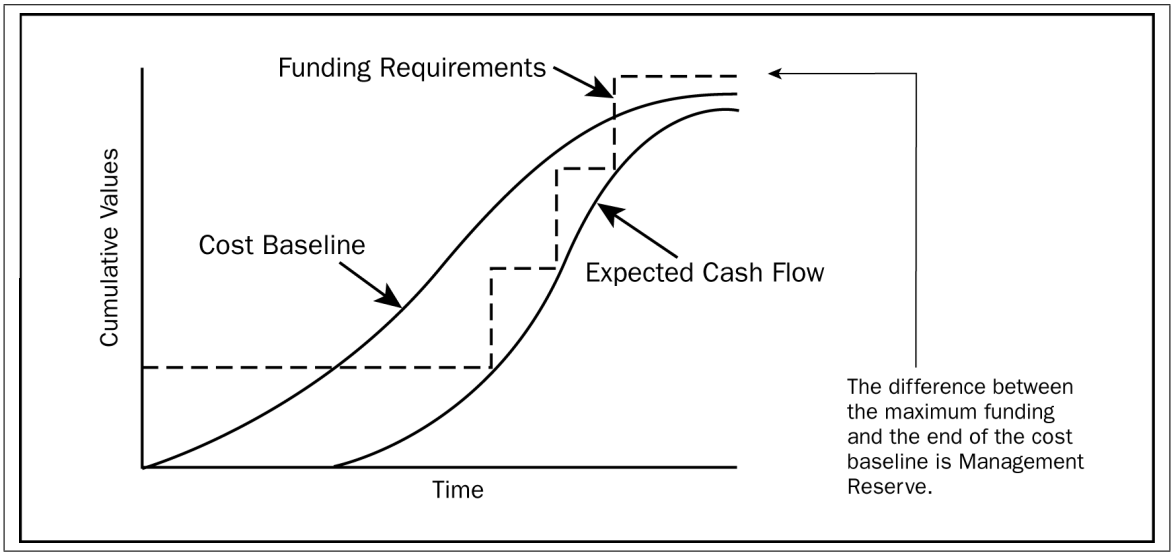

Abbildung 3.5: Cost Baseline [8, S. 13]

In der Regel weist die Cost Baseline die Form einer liegenden S-Kurve auf. Dieser Aspekt ergibt sich aus der Natur von Projekten, in unterschiedlichen Phasen unterschiedliche Kostenintensitäten aufzuweisen. Am Beginn eines Projektes schlagen Planungskosten mit geringem Gewicht zu Buche, dies erkärt die Flache Steigung zu Begin der S-Kurve. In der darauf folgenden Umsetzungsphase des Projektes steigt die zunächst Kurve immer steiler, d.h. die Kosten steigen zB aufgrund höherer Beschäftigungsraten stärker

an als zuvor. üblicherweise ergibt ein der Wendepunkt zu inmitten der Umsetzungsphase, wobei Entwicklungsressourcen sukzesive zurückgefahren werden, bis letztendlich in der Projektnachbereitung die Steigung der Kurve bis zum Projektende gegen Null sinkt.

3.3.2 Project Funding Requirements

Das Projekt definiert sogenannte **Project Funding Requirements** um dem Management zu kommunizieren, wofür wieviel Geld von nöten ist. Darunter fallen das Projektbudget sowie die veranschlagten Reserven (siehe Reserve Analysis). Je nach Anforderungen seitens des Management bzw. Größe des Projektes ergeben sich unterschiedliche Granularitätsgrade der Project Funding Requirements, wobei in Klassen von Kosten wie Personalkosten, Anmietungen, Materialkosten, Gerätschaft und dergleichen unterteilt wird.

Die Project Funding Requirements werden wie die Cost Baseline über den zeitlichen Projektverlauf aufgeschlüsselt, zB Anforderungen je Monat. Im Gegensatz zur Kurvenform der Cost Baseline visualisieren sich Project Funding Requirements über den Projektverlauf in Stufenform.

Weitere Outputs des Cost Budgeting umfassen Updates of Cost Management Plan und Requested Changes, wie bereits in Cost Estimating behandelt.

3.4 Fazit

In diesem Kapitel wurde das Cost Budgeting als Kostenmanagementbereich zwischen Cost Estimation und Cost Control beschrieben. **Inputs** wie der Projektplan, die Projektabgrenzung und die WBS werden mittels **Werkzeuge & Techniken** wie Cost Aggregation, Reserve Analysis, Parametric Estimation und Funding Limit Reconciliation zu den nützlichen **Outputs** des Cost Budgeting wie der Cost Baseline übergeführt.

Die Abgrenzung zu Cost Estimation ist vorallem bei kleineren Projekten schwierig, weshalb Cost Budgeting in solchen Fällen des öfteren innerhalb der Cost Estimation behandelt wird. In größeren Projekten macht es aber durchaus Sinn, die beiden Disziplinen klar zu trennen. Cost Estimation zur allgemeinen Bewertung von Kosten für Aktivitäten und der angewandten Berechnung und Verwaltung dieser Größen im Zusammenhang mit dem Projektplan im Rahmen des Cost Budgeting.

4 Cost Control

Cost Control ist ein wesentlicher Bestandteil des Kostenmanagements. Es überwacht die Kostenentwicklung im Bezug zu den Soll-Kosten und zeigt so frühzeitig Budgetknappheit bzw. Budgetüberschuss auf. Desweiteren wird auch der Projektfortschritt überwacht um etwaige Verzögerungen im zeitlichen Rahmen zu erkennen.

Als Input benötigt Cost Control die Projektplanung (Work-Breakdown Structure, geplante Zeit/Kosten usw.) sowie laufende Informationen über den Projektstatus. Verglichen wird im generellen Soll- und Ist-Status. Die Earned Value Technique (EVT) stellt eines der wichtigesten Werkzeuge des Cost Control dar.

Mithilfe der gewonnen Informationen können Prognosen für den weiteren Projektverlauf erstellt werden und bei Bedarf steuernd in das Projekt eingegriffen werden.

Begleitendes Beispiel
Alle im folgende vorgestellten Werkzeuge und Kennzahlen werden zur Veranschaulichung von folgendem Mini-Beispiel-Projekt begleitet:

Projekt: **'Installation eines PCs'**
Arbeitspaket(e):

Nr. 1 — Installation des Betriebssystems — Arbeitszeit 4h a 50 Euro

Nr. 2 — Installation Officepaket — Arbeitszeit 3h a 50 Euro

Bis jetzt wurde an Nr. 1 vier Stunden gearbeitet und zu 30 Prozent abgeschlossen, Nr. 2 folgt auf Nr. 1

4.1 Earned Value Technique (EVT)

Die Earned Value Methode vergleicht den Wert der bereits fertig gestellten Arbeit mit den aktuellen und geplanten Kosten. Alle Werte werden in monetärer Form ausgedrückt, sodass ein einfacher Vergleich ermöglicht wird.

Die Notwendigkeit dieser Methode wird klar, sobald man sich folgendes Beispiel vor Augen hält: Bei einem Verbrauch von 80.000 Euro bzw. 80 Prozent, des geplanten Budgets, würde man meinen, dass der Verbrauch unter dem Geplanten liegt, doch wenn der Leistungswert (Der Wert der erbrachten Arbeit) nur 70.000 Euro beträgt erkennt man sofort die Budgetknappheit.

Folgend werden benötigte Kennzahlen der EVT beschrieben.

4.1.1 Planned Cost (PC) - Plankosten

Als Input benötigt die Earend Value Technique sämtliche Arbeitspakete (Tasks) aus dem Projektstrukturplan, sowie die geplanten Kosten für die einzelnen Arbeitspakete.

So ergeben sich zu jedem Projektzeitpunkt die geplanten Kosten, welche die Ausgangsbasis darstellen

Im Beispiel
Die Plankosten sind die Kosten von Arbeitspaket Nr. 1, da vier Stunden daran gearbeitet wurde:
Nr. 1 — Installation des Betriebssystems — Arbeitszeit 4h a 50 Euro

$$PC = 4h * 50\ \texttt{EUR} = 200\ \texttt{EUR}$$

4.1.2 Budget at Completion (BAC) - Gesamtbudget

Beziffert das Gesamtbudget für das Projekt.

Im Beispiel
Das Gesamtbudget ist die Summe der Kosten aller Arbeitspakete

$$BAC = 4h * 50\ \texttt{EUR}\ + 3h * 50\ \texttt{EUR} = 350\ \texttt{EUR}$$

4.1.3 Actual Cost (AC) - Istkosten

$$AC = Arbeitskosten + Materialkosten$$

In die aktuellen Kosten laufen sämtliche, zur bisherigen Umsetzung des Tasks benötigten, angefallen Kosten. Dazu zählen sowohl Arbeits- als auch Materialaufwand.

Im Beispiel
Da wir bereits 4 Stunden an Arbeitspaket Nr. 1 gearbeitet haben, ergeben sich:

$$AC = 4h * 50\ \texttt{EUR} = 200\ \texttt{EUR}$$

4.1.4 Earned Value (EV) - Leistungswert

$$EV = Plankosten * Fortschritt$$

Der Leistungswert errechnet sich aus den Plankosten*Fortschritt in Prozent und ergibt somit wieder einen monetären Wert. Dies ist genau der Wert des Outputs den er zu diesem Zeitpunkt hat und spiegelt z.B.: den Wert für den Kunden wieder. Auch andere Metriken sind denkbar.

Mithilfe dieser drei Kennzahlen (Plankosten, Istkosten und Leistungswert) lässt sich nun ein guter Überblick über das aktuelle Projektbudget veranschaulichen.

Im Beispiel
Wir haben erst 30 Prozent Fortschritt beim aufsetzen des PCs, das ergibt einen Leistungswert von

$$EV = 200\ \texttt{EUR} * 0.3 = 60\ \texttt{EUR}$$

4.1.5 Schedule Variance (SV) - Planabweichung

$$SV = Leitsungswert - Plankosten$$

Wieviel ist die geleistete Arbeit zum aktuellen Zeitpunkt wert und wie viel sollte sie wert sein? Die Antwort zu dieser Frage liefert diese Kennzahl. Sie zeigt auf, wieviel Arbeit bereits geleistet wurde und wieviel zum aktuellen Zeitupunkt geleistet hätte werden sollen. Ein negatives Ergebnis (kleiner 0) zeigt auf, dass der akutelle Wert der geleisteten Arbeit kleiner als der geplante Wert zum aktuellen Zeipunkt ist.

Im Beispiel

$$SV = 60\ \text{EUR} - 200\ \text{EUR} = -140\ \text{EUR}$$

4.1.6 Cost Variance (CV) - Kostenabweichung

$$CV = Leistungswert - Istkosten$$

Wieviel ist die geleistete Arbeit zum aktuellen Zeitpunkt wert und wie hoch sind die Kosten welche diese Arbeit verursacht hat? Wie bei der Schedule Variance gilt auch hier, dass ein negatives Ergebnis (kleiner 0) einen negativen Projekt-Status aufzeigt.

Im Beispiel

$$CV = 60\ \text{EUR} - 200\ \text{EUR} = -140\ \text{EUR}$$

4.1.7 Schedule Performance Index (SPI) - Zeiteffizienz

$$SPI = \frac{Leistungswert}{Plankosten}$$

Spiegelt das Verhältnis zwischen abgeschlossener Arbeit und geplanter Arbeit zum aktuellen Zeitpunkt wieder. Das Ergebnis is ein prozentualer Wert, welcher Aufschluss darüber gibt ob das Projekt - zeitlich gesehen - im Verzug oder schneller als geplant erledigt ist. Sobald der Wert kleiner 1 ist, ist ein zeitlicher Verzug zu erwarten, wenn der Verzug nicht wieder aufgeholt werden kann. Ein Wert von z.B.: 0.5 - also 50 Prozent deutet darauf hin, dass sich die Projektlaufzeit um 100 Prozent erhöhen wird.

Im Beispiel

$$SPI = \frac{60\ \text{EUR}}{200\ \text{EUR}} = 0,3$$

4.1.8 Cost Performance Index (CPI) - Kosteneffizienz

$$CPI = \frac{Leistungswert}{Istkosten}$$

Spiegelt das Verhältnis zwischen dem Wert der fertiggestellten Arbeit und dem geplanten Wert zum aktuellen Zeitpunkt in Prozent wieder. Auch hier gilt wie beim Schedule Performance Index, dass ein Wert kleiner 1 eine schlechte Effizienz im Bezug zu den Kosten wiedergibt, während ein Wert größer 1 auf eine Kostenersparnis im Projekt deutet.

Im Beispiel

$$CPI = \frac{60\ \text{EUR}}{200\ \text{EUR}} = 0,3$$

4.2 Forecasting

Um Aussagen über den zukünfigen Projektverlauf treffen zu können bedient man sich dem so genannten Forecasting. Diese wird erstellt und aktualisiert sobald Informationen vorhanden sind oder diese sich ändern. Es können somit Aussagen über den zeitlichen als auch über den budgetären Verlauf des Projekts in Zukunft getätigt werden.

Alle Aussagen gehen von einer zukünftigen linearen Entwicklung aus. Das bedeutet, dass wenn die Umsetzung der Tätigkeit in der Vergangenheit doppelt so lange gedauert hat als geplant, davon ausgegangen wird, dass auch in Zukunft die Umsetzung doppelt so viel Zeit in Anspruch nehmen wird. Einze zukünftige Effizienzssteigerung ist also nicht möglich, auch wenn vorangegangene Tätigkeiten eine Investition waren und dadurch nachfolgende Tätigkeiten schneller als geplant durchgeführt werden könnten.

Als Input benötigt man aktuell verfügbaren Informationen - zumeist die Kennzahlen aus der Earned Value Technique.

4.2.1 Estimate to Complete (ETC) - Kosten zur Fertigstellung

ETC sind die Kosten welche zur Fertigstellung des Projekts noch anfallen werden, basierend auf der bisherigen Kosteneffizienz.

$$ETC = \frac{(Gesamtbudget - Leistungswert)}{Kosteneffizienz}$$

Im Beispiel

$$ETC = \frac{(350\ \text{EUR} - 60\ \text{EUR})}{0,3} = 966,66\ \text{EUR}$$

4.2.2 Estimate at Completion (EAC) - Gesamtkosten bei der Fertigstellung

EAC sind die Gesamtkosten welche am Ende des Projekts angefallen sein werden.

$$EAC = Gesamtbudget / Kosteneffizienz$$

Im Beispiel

$$EAC = \frac{350\ \text{EUR}}{0,3} = 1166,66\ \text{EUR}$$

5 Cost Management Plan

Nach der Abschätzung der Projektkosten mit Hilfe der unter **Cost Estimation** vorgestellten Methoden ist das Projektmanagement jedoch noch nicht beendet. In der Praxis werden jene theoretischen Kosten selten präzise dem entsprechen was vorher abgeschätzt wurde. Ein Projekt muss jedoch auch Kostenüberschreitungen verkraften, dies erfordert allerdings entsprechende Aktionen des Projektmanagements. Da Kostenüberschreitungen zum Zeitpunkt der Kostenschätzung nicht vorhergesehen werden können, eine zeitnahe Reaktion jedoch wichtig ist, wird ein **Cost Management Plan** erstellt, welcher die angemessenen Reaktionen auf mögliche auftretende Probleme beinhaltet.

Der Cost Management Plan beschreibt wie auf Kostenabweichungen reagiert wird (z.B. unterschiedliche Reaktionen auf schwerwiegende Probleme im Vergleich zu kleineren Problemen). Ein Cost Management Plan kann formal oder informal, sehr detailliert oder rahmenartig sein, abhängig von den Bedürfnissen der Stakeholder des Projekts. Er ist ein Teil des gesamten Projektplans. [12]

Ein praktisches Beispiel für erfasste Daten im Cost Management Plan findet sich in Softwareprojektmanagementleitfaden der US Luftwaffe[16]:

- Welche Kosten und kostenbezogenen Daten werden gesammelt und analysiert.
- Wie oft werden diese Daten gesammelt und analysiert.
- Woher stammen die kostenbezogenen Daten.
- Wie werden die Daten analysiert.
- Welche Personen und Organisationen sind in diesen Prozess miteinbezogen und welche Verantwortlichkeiten und Pflichten haben diese.
- Die Grenzen der Kostenabweichung welche noch akzeptabel sind.
- Welche Verantwortlichkeiten gibt es im Zusammenspiel von Cost Control und Change Control.
- Prozedere und Verantwortlichkeiten bei unakzeptablen Kostenabweichungen.

Diese Punkte scheinen teilweise trivial zu sein, haben jedoch ihre Berechtigung. Die ersten drei Punkte sind Grundlage um überhaupt sinnvoll messen zu können: Sie definieren was gemessen wird und wie es gemessen wird. Wichtig ist außerdem, dass es eindeutig definierte Zuständigkeiten gibt. Die Grenzen der Kostenabweichung sind ebenfalls ein wichtiger Parameter: Es muss eine Balance gefunden werden um die vorhersehbaren Kostenschwankungen in projektspezifischen Grenzen zu tolerieren, die Cost Control sollte jedenfalls nicht mit Kleinigkeiten von großen Problemen abgelenkt werden.

Der wichtigste Punkt ist das Prozedere bei unakzeptablen Kostenabweichungen, was jedoch für jedes Projekt anders aussehen dürfte und auch stark von den involvierten Stakeholdern abhängig ist.

6 Conclusions

Die hier vorgestellten Methoden sollten in Softwareprojekten zu akkurateren Kostenschätzung und besseren Einhaltung des Budgets führen. Das große Problem ist jedoch die Auswahl der passenden Schätzmethoden für ein konkretes Projekt, was ohne Erfahrungswerte mit unterschiedlichen Modellen eine schwierige Aufgabe darstellt. Empfehlenswert ist es jedenfalls mehrere Kostenschätzverfahren einzusetzen um die teilweise hohen Fehlerraten zu kompensieren. Ganz wichtig für akkurate Kostenschätzungen ist die Nachbearbeitung von abgeschlossenen Projekten. Hierbei müssen alle Faktoren dokumentiert werden um zukünftige Schätzungen noch genauer zu machen.

Trotz der negativen Studien über die hohe Abbruchrate von Softwareprojekten kann man jedoch sagen, dass das Projektmanagement und insbesondere das Kostenmanagement von Softwareprojekten deutlich besser sein dürfte als sein Ruf. Es existiert eine Vielzahl von unterschiedlichen Methoden und es kann angenommen werden, dass diese auch verwendet werden. Ob formell oder informell sei hierbei dahingestellt.

Literaturverzeichnis

[1] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Trans. Softw. Eng.*, 9(6):639–648, 1983.

[2] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[3] Barry W. Boehm, Chris Abts, Jongmoon Baik, A. Winsor Brown, Sunita Chulani, Bradford Clark, Ellis Horowitz, Ray Madachy, Don Reifer, and Bert Steece. Usc cocomo ii.2000. Technical report, The University of Southern California, 2000.

[4] Barry W. Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches – a survey. *Ann. Softw. Eng.*, 10(1-4):177–205, 2000.

[5] K. El Emam and A.G. Koru. A replicated survey of it software project failures. *Software, IEEE*, 25(5):84 –90, sept.-oct. 2008.

[6] Gerald A. Feltham. Cost aggregation: An information economic analysis. *Journal of Accounting Research*, 1977.

[7] G.R. Finnie and G.E. Wittig. Ai tools for software development effort estimation. In *Software Engineering: Education and Practice, 1996. Proceedings. International Conference*, pages 346 – 353, 24-27 1996.

[8] Trabitsch Grechenig. Management von software projekten. Slides, 2010.

[9] Standish Group. Chaos report, 1994.

[10] Kim Heldman. *PMP: Project Management Professional Exam Study Guide, 5th Edition*. Sybex, 2009.

[11] Joint Government/Industry Initiative. *Parametric Cost Estimating Handbook*. 1995.

[12] Project Management Institute. *A guide to the project management body of knowledge*. Duncan, William R., 1996.

[13] Project Management Institute. The project management knowledge areas). 2000.

[14] Karen Lum, Michael Bramble, Jairus Hihn, John Hackney, Mori Khorrami, and Erik Monson. Software cost estimation handbook. Technical report, JPL, December 2002. [last visited 30-May-2010].

[15] Kjetil Molkken and Magne Jrgensen. A review of surveys on software effort estimation. In *ISESE '03: Proceedings of the 2003 International Symposium on Empirical Software Engineering*, page 223, Washington, DC, USA, 2003. IEEE Computer Society.

[16] Department of the air force Software Technology Support Center. *Guidlines for Successful Acquisition and Management of Software-Intensive Systems - Condensed Version*. 2003.

[17] Shari L. Pfleeger, Felicia Wu, and Rosalind Lewis. *Cost Estimation and Sizing Methods: Issues and Guidelines*. RAND Corporation, 2005.

[18] Donald J. Reifer. Softcost-ada: user experiences and lessons learned at the age of three. In *TRI-Ada '90: Proceedings of the conference on TRI-ADA '90*, pages 472–482, New York, NY, USA, 1990. ACM.

[19] H. Robinson and R. Richards. Critical chain project management: Motivation and overview. In *Aerospace Conference, 2010 IEEE*, pages 1 –10, 6-13 2010.

[20] Martin Shepperd and Chris Schofield. Estimating software project effort using analogies. *IEEE Trans. Softw. Eng.*, 23(11):736–743, 1997.

[21] J. Alex Sherrer. *Project Management Road Trip For the Project Management Professional: Your Key to PMP Certification and Understanding the PMBOK Fourth Edition (Paperback)*. lulu.com, 2010.

[22] Ian Sommerville. *Software Engineering (7th Edition) (International Computer Science Series)*. Addison Wesley, May 2004.

[23] The pm411.org Project Management Podcast. Podcast episode 029: Project cost budgeting, 2009.

[24] Hareton Leung Zhang and Zhang Fan. Software cost estimation. In *Handbook of Software Engineering and Knowledge Engineering , World Scientific Pub. Co, River Edge, NJ*, 2002.