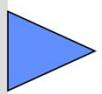


Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Softwaretechnik



Kapitel 7



1. Wieso Software Engineering?
2. Vom Einzelkämpfer zum Großprojekt

Systematische Softwareentwicklung

3. Anforderungen und Test: *Basis des Projekts*
4. Entwurf: *Strukturen und nicht-funktionale Eigenschaften*
5. Entwürfe notieren mit UML: *Modelle im SE*
6. Design Patterns: *Entwurfserfahrungen nutzen*
7. Management von *Technik und Projekt*

Leibniz Universität Hannover

SWT 2015/16 · 287

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Inhalt von Kapitel 7

7. Management von Technik und Projekt

- Versions- und Konfigurationsmanagement
- Aufgaben von Projektleitern
- Aufwandsschätzungen
- Risikomanagement in Softwareprojekten
- Agile Softwareentwicklung

Leibniz Universität Hannover SWT 2015/16 · 288

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**Software
Engineering**

Code organisieren: Versionen und Konfigurationen

- **Ziele**
 - Dateien, die bei der Entwicklung anfallen, für Team koordinieren
 - Für alle zugreifbar
 - Kein versehentliches Löschen
 - Kein gegenseitiges Überschreiben
 - Klarheit über jeweils gültige Fassung

Auch das ist wichtig: Aktuelle Versionen aufbewahren, in Arbeitsgruppe koordiniert austauschen.

Dieses Prinzip finden Sie in den verschiedensten Tools. Es geht hier nicht um Details eines Produkts, sondern um das Prinzip.

Das Prinzip sehen Sie hier an den allgemeinen Begriffen. Die Kommandos von SVN heißen ganz genauso!

Hier sehen Sie also eigentlich den derzeitigen Stand von SVN. Der ist relativ einfach und schon sehr gut geeignet auch für große Projekte.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**Code organisieren:
Versionen und Konfigurationen**

- Grundbegriffe
 - Version =_{Def} Definierter Stand eines Dokuments
 - Konfiguration =_{Def} Menge der Versionen aller Bestandteile eines Systems, die zu einem definierten Stand gehören

System = A+B+C Konfiguration (t_k)

Teil A	Teil B	Teil C
v1	v17	v0.8
v2	v18	
v3		v1.1 v2.0

Zeit

t_k

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 290

Jedes Dokument (Code, Spezifikation etc.) hat verschiedene Versionen.

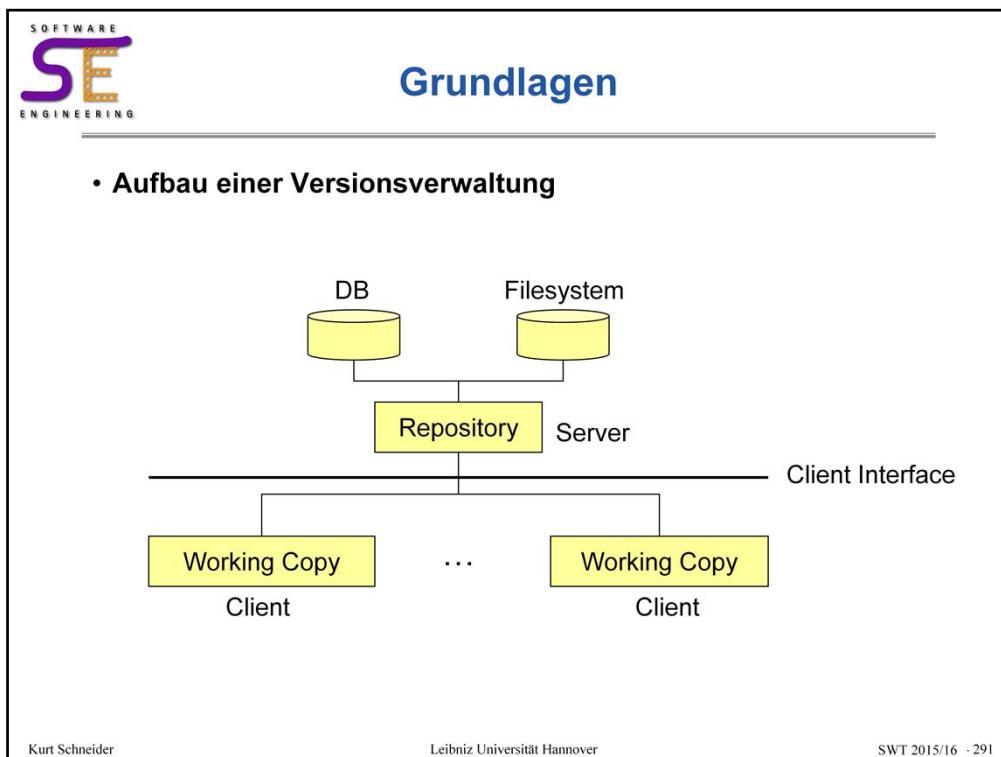
Das System besteht aus mehreren solchen Teilen.

Eine Konfiguration hat für jedes Teil genau eine definierte Version.

Das muss nicht die neueste sein (wie oben B v17 statt v18), aber natürlich können nur vorhandene in die Konfiguration eingebunden werden (z.B. nicht v1.1)

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Das Tool versteckt vor Ihnen (Information Hiding, gut!), ob Sie die Einträge in Dateien, eine Datei, oder eine Datenbank stecken.

Hauptsache, Sie können mit den gezeigten Operationen darauf arbeiten.

Das „Repository“ ist der Überbegriff über die Speicherformen, Sie selbst haben „Working Copies“ auf Ihren eigenen Arbeitsrechnern.

Wie kann man nun zwischen verschiedenen Working Copies mit Hilfe des Repositories abstimmen?

Das ist die Aufgabe.

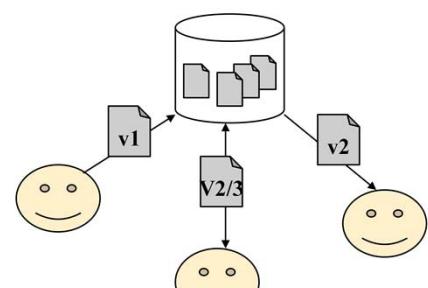
Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Wichtige Begriffe 1/2

- **Repository**
 - Zentraler Datenspeicher
- **Working Copy**
 - Lokale Arbeitskopie
- **Checkout**
 - Arbeitskopie aus dem Repository erstellen
- **Update**
 - Arbeitskopie mit Änderungen im Repository aktualisieren
- **Add**
 - Neue Datei/Verzeichnis dem Repository hinzufügen
- **Checkin/Commit**
 - Änderungen der Arbeitskopie ins Repository schreiben



Optimistisch

Mehrere haben Zugriff auf ein Dokument
Falls beide ändern, gibt es Kollision

Pessimistisch

Sicherheitshalber darf nur einer ändern
Keine Kollision
Behindert aber Parallelarbeit

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 292

So (in SVN) oder so ähnlich (in evtl. zukünftigen Tools) heißen die wichtigsten Kommandos. Das ist hier also das Prinzip und gleichzeitig heißen die SVN-Kommandos genau so.

Natürlich dürfen nicht zwei Personen die gleiche Datei an der gleichen Stelle ändern, das führt zu Inkonsistenzen.

Beim optimistischen Ansatz hofft man, dass schon kein Konflikt auftreten wird und gibt dieselbe Datei auch mehrfach zum Schreiben aus.

Nur bei tatsächlichen Konflikten, die man beim Zurückschreiben (commit) bemerkt, muss der zweite den Konflikt zuerst beheben.

(Wer zuerst kommt, hat Glück. Daher wird man bei optimistischen Strategien versuchen, möglichst rasch Änderungen zurückzumelden.)

Im pessimistischen Fall lässt man jeweils nur eine Person schreibenden Zugriff gewinnen.

Dann kann es zwar keine Konflikte geben, aber die Parallelarbeit ist auch erheblich eingeschränkt.

Beide Strategien sind üblich; in den modernen (auch „agilen“) SW-Entwicklungsmethoden ist die optimistische Variante häufiger.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Wichtige Begriffe 2/2

- Conflict
 - Gleichzeitige Änderung der selben Stelle einer Datei
 - Diff
 - Differenzen zweier Dateien anzeigen
 - Merge
 - Zusammenführen von zwei in Konflikt stehenden Dateien
 - Tag
 - Spezieller Name einer Version (z.B.: Release 1.4)
 - Branch
 - Separater Zweig eines Projekts, an dem unabhängig vom Hauptzweig weitergearbeitet wird
 - Z.B.: Hauptzweig: Firefox 1.5.x, Branch: Firefox 2.0.x

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 - 293

Immer noch: Prinzip und SVN

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Versionsmanagement
Prinzip („optimistisch“)

- **Grundoperationen**
 1. Aufsetzen: Pfade, Rechte etc.
 2. Datei beim System anmelden: **add**
 3. Datei eintragen: **commit**
 4. Datei herausholen: **checkout**
 5. Datei zurückgeben: **commit**
 6. Datei nochmal holen: **update**
- **Weiterführendes**
 - Branches (Verzweigungen)
 - Abspalten: **branch**
 - Verbinden: **merge**
- **Konflikte**
 - Nie beim Auslesen: „optimistisch“
 - Falls beim Zurückgeben:
 - Merge versuchen
 - Sonst Warnung, manuell

Die Zahlen zeigen die Reihenfolge.

Bei Schritt 5b kommt es zum Konflikt, bzw. wird der schon bestehende Konflikt hier bemerkt.

Um ihn zu reparieren, muss sich der zweite Schreiber den aktuellen Zustand holen (6 update).

Die Konflikte aufzulösen, kann schwierig sein. Wichtig ist dafür ein Dateivergleicher, mit dem man die eigenen Änderungen mit der (anderweitig geänderten) Version vergleichen und abstimmen kann. Wenn bis dahin nicht wieder jemand ein commit abgegeben hat, kann der zweite Schreiber dann selbst mit commit die Version zurückspielen, in der beide Änderungen (des ersten und des zweiten Schreibers) berücksichtigt sind.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Variante: Verteiltes Versionsmanagement
Git git-scm.com und GitHub

- **Git: Versionsmanagement-System auf GitHub**
 - Initiator: Linus Torvalds (Linux)
 - Gestartet als „Anti-CVS (bzw.: Anti-SVN)“
- **GitHub: Repositories (Hosting, Vernetzung, Social Network)**
 - Open source: kostenlos; Privat/exklusiv: gegen Bezahlung

Remote repository
Local repository
Index (cache)
Working directory
pull fetch push
commit add
checkout HEAD
Von: Wikipedia („Git“), 4.11.12
Kurt Schneider

GitHub
Local Repository Working Copy git
github
Local Repository Working Copy git
Leibniz Universität Hannover
SWT 2015/16 · 295

Eine andere Gruppe von Konfigurations- und Versionsmanagement-Werkzeugen arbeitet noch verteilter; denn auch SVN kann man ja auf einem Server laufen lassen und von überall her zugreifen. Jeder Entwickler „cloned“ sich ein Local Repository, das alles enthält. Aber natürlich braucht man eine Verbindung zwischen ihnen, das Remote Repository.

Derzeit bekanntester Vertreter ist Git, mit GitHub, als dem dazugehörigen Service zum Hosten von Repositories und dazugehörigem Sozialen Entwicklernetzwerk.

Schon lokal hat man in Git ein Repository (Local). Es gibt also eine Stufe mehr: Eigene Dateien ohne Git; dann das eigene Local Repository, in dem man über add und commit Dateien hat, das funktioniert genau so wie das Prinzip und SVN. Aber dann erst kommt die Verteilung, die war in SVN schon oben mit drin, denn dort gibt es nur ein Repository.

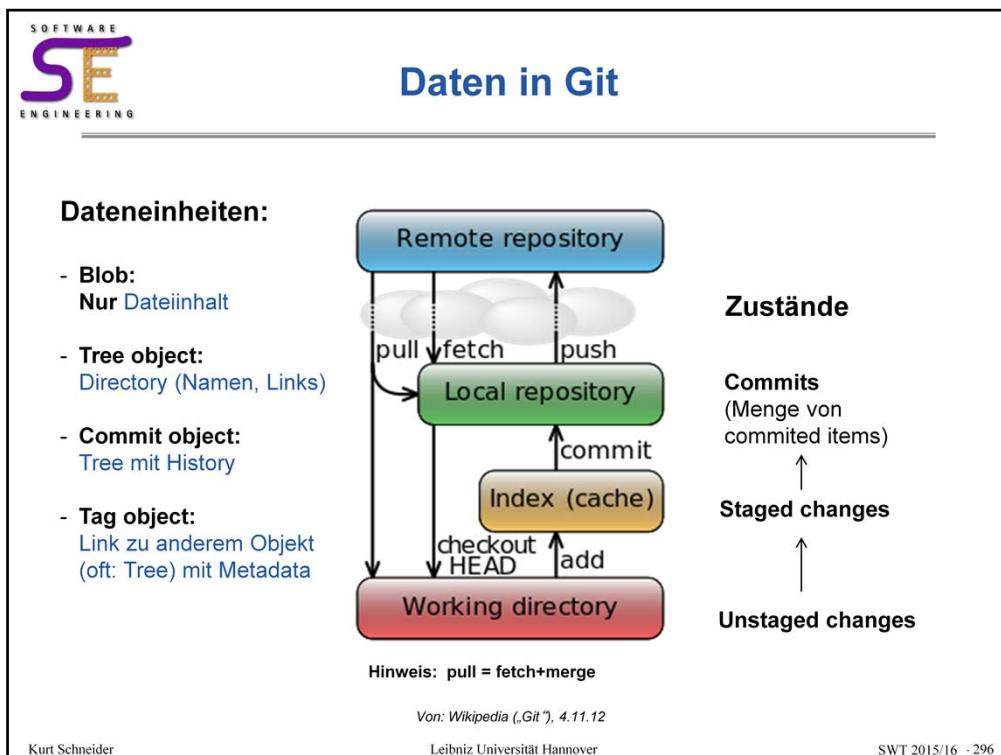
Über push kann man in Git Repositories mit anderen teilen, mit pull und fetch kann man von anderen, mit denen man zusammenarbeitet, etwas holen.

Fetch liest nur. Pull liest (fetch) und merged es noch mit den eigenen Dateien.

Praktisches Kommando: git status zeigt an, wie es derzeit steht, was also in welchem Status ist – was muss ich noch commiten usw.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Links die Terminologie für Dateneinheiten in Git.

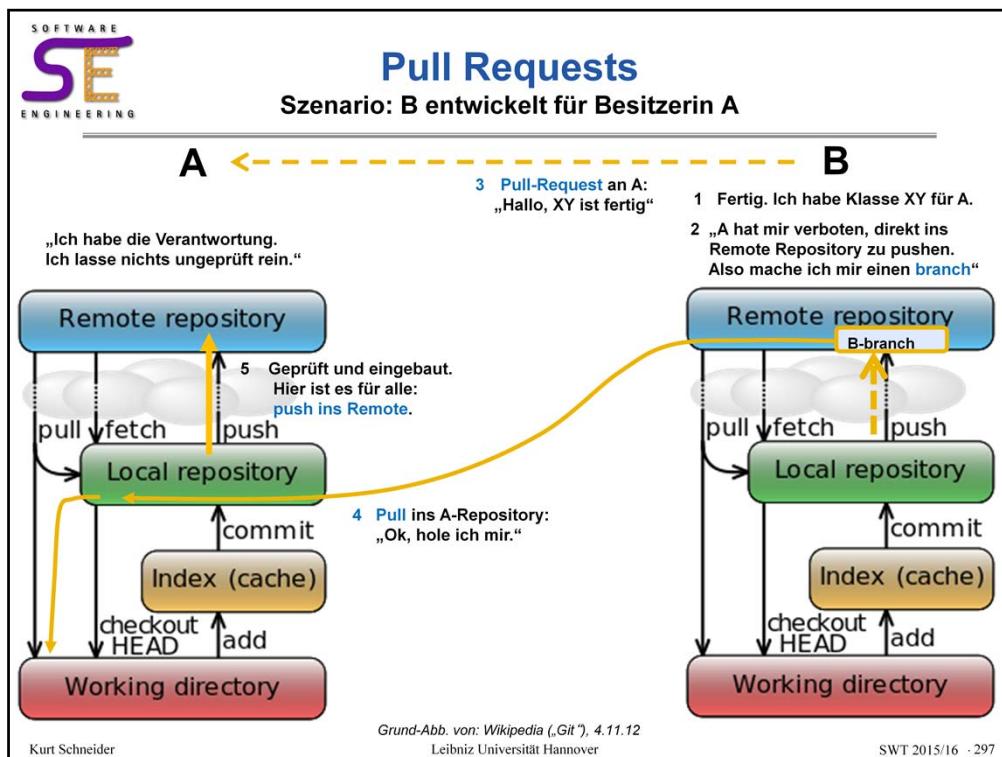
Rechts die Bezeichnung der Zustände einer Datei, von unten nach oben zu lesen.

Interessant ist, dass man mit dem Kommando commit in der Regel nicht nur eine Datei sondern alle im Index vorhandenen („staged“) Änderungen bestätigt. Die Menge der auf einen Schwung bestätigten Dateien sind „das Commit (Object)“, das unbedingt kommentiert werden sollte: was soll mit dem Commit erreicht werden?

Bis hierher kann man einfach herumprobieren, die Commits in verschiedene Branches stecken und auch einfach wieder unbemerkt löschen. Erst wer mit per pull ins Netz geht, kommt mit anderen zusammen. Ob es dann gleich alle zu sehen bekommen, hängt von der Einstellung ab.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Die verteilte Zusammenarbeit funktioniert so wie die Zahlen angeben:

- A hat ein Projekt
- B macht etwas dafür uns pusht es. Aber noch nicht ins Remote Repository des Projekts, nur in einen Branch
- A muss das erst übernehmen. A erfährt vom neuen XY durch eine Art Email; holt (pull) es sich dann, sieht es durch, korrigiert es ggf. und schiebt es dann mit push ins allgemeine Remote Repository. A darf das, weil sie Eigentümerin des Projekts ist.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Inhalt von Kapitel 7

7. Management von Technik und Projekt

- Versions- und Konfigurationsmanagement
- Aufgaben von Projektleitern
- Aufwandsschätzungen
- Risikomanagement in Softwareprojekten
- Agile Softwareentwicklung

Leibniz Universität Hannover SWT 2015/16 · 298

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The logo for Software Engineering (SE) features the letters 'S' and 'E' in a stylized font. The 'S' is purple with a black outline, and the 'E' is yellow with a black outline. Below the letters, the word 'ENGINEERING' is written in a smaller, black, sans-serif font.

SOFTWARE
SE
ENGINEERING

Was ist ein Projekt?

- Inhaltlich abgeschlossenes Vorhaben
 - Keine ständige Aufgabe
 - Vorhaben komplexer Natur
- Zeitlich begrenzt
 - Definierten Anfangs- und Endtermin
- Einmaligkeit der Bedingungen (nicht immer das Gleiche: **Risiken**)
 - Projektziele
 - Neuartige oder unbekannte Probleme zu lösen
 - Abgrenzung der Aufgabe
 - Mitwirkende Organisationen
 - Projektteam: Teilnehmende Personen
 - Budget und Ressourcen

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 299

Heute spricht man ständig von Projekten, ohne sich so recht klar zu machen, was damit gemeint ist.

Das Ziel, die zeitliche und inhaltliche Begrenzung und die „Einmaligkeit“ machen ein Projekt aus.

Damit liegt kein Projekt vor, wenn ...

Eine ständig fortzusetzende Tätigkeit beschrieben wird. Eine Verwaltungsabteilung bearbeitet bei der Reisekostenabrechnung kein Projekt, weil diese Tätigkeit kein inhaltlich abgeschlossenes Vorhaben darstellt, sondern immer weiter geht.

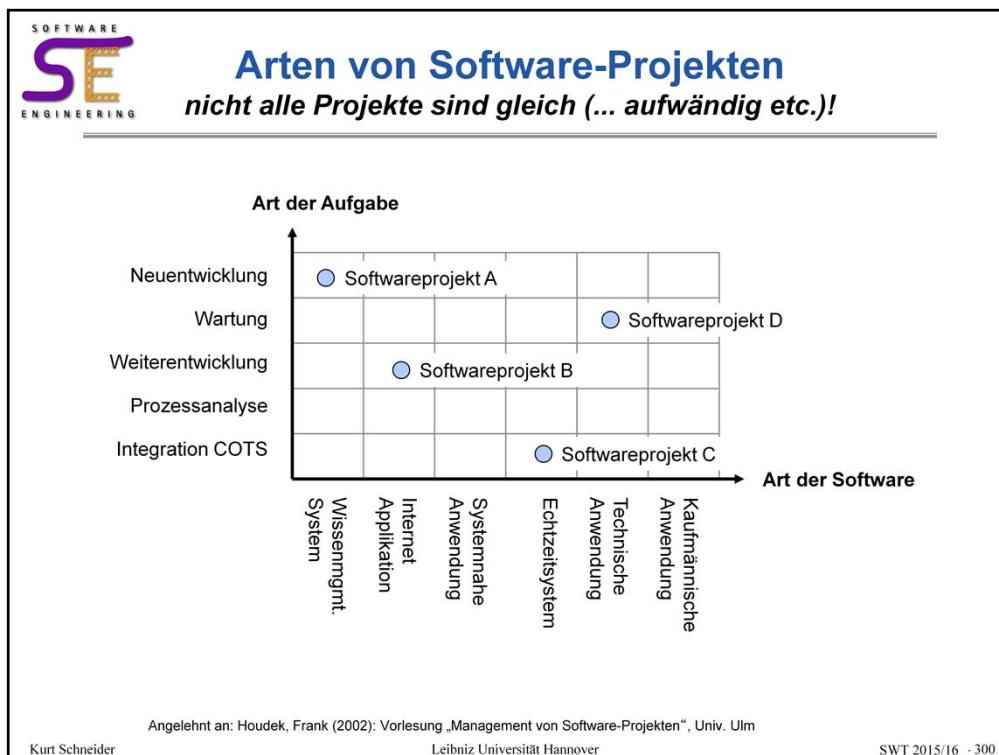
Der Betrieb einer Firma ist kein Projekt, weil es nicht auf einen begrenzten Zeitraum, sondern (zunächst) unbegrenzt angelegt ist.

Eine Routinetätigkeit ist kein Projekt, weil sie das Originalitätskriterium nicht erfüllt.

Alles zusammen genommen sind Projekte meist interessante Aufgaben, die besonders herausfordernd sind und – im Gegensatz zu „normalen Aufgaben“ eben auch scheitern können.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Nicht alle SW-Projekte sind gleich und damit auch nicht direkt vergleichbar. Die bewährten Schätzverfahren sehen Anpassungsfaktoren vor, die es einer Unternehmung erlauben, ihre eigene Situation zu berücksichtigen.

Sie sehen oben zwei Dimensionen, in denen sich SW-Projekte unterscheiden: Die inhaltliche Dimension (x-Achse) führt zu ganz unterschiedlichen Projekten. Aber auch die y-Achse ist wichtig, denn die Integration eines fertigen Programm Pakets COTS hat natürlich ganz andere Charakteristika als eine Neuentwicklung. Dies muss man in Schätzungen berücksichtigen, dazu braucht man die Korrekturfaktoren.

Eine reife Organisation wird Analogie-Daten für mehrere Felder der obigen Matrix bereit halten, um die Vergleichbarkeit zu verbessern.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The logo for Software Engineering (SE) features the word "SOFTWARE" in small black capital letters above the letters "SE" in large, bold, yellow capital letters. The letter "E" has a purple outline and is decorated with a series of small yellow lightbulbs or stars arranged in a curved pattern along its top edge.

SOFTWARE
SE
ENGINEERING

Gutes Projekt-Management

- Erfolgsfaktoren für Projekte

- Einbeziehung der Benutzer (aller Stakeholder, 16%)
- Unterstützung durch das Management (14%)
- Klar beschriebene Anforderungen (13%)
- Geeignete Planung (10%)
- Realistische Erwartungen (8%)
- Ausreichend feine Meilensteine (8%)
- Gut ausgebildete Mitarbeiter (7%)
- Ownership (5%)
- Klare Vision und Ziele (3%)
- Motivierte, hart arbeitende Mitarbeiter (2%)
- Sonstiges (13%)

Chaos Report, The Standish Group, 1995: Ein Klassiker

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 301

Dieser Report der Standish Group wird überall zitiert.

Er soll zeigen: Die meisten SW-Projekte laufen und enden nicht befriedigend. Sie führen nie zu einem einsetzbaren Ergebnis oder wenn, dann viel zu spät und mit zu hohen Kosten.

Die Gründe dafür liegen zum großen Teil in den frühen Phasen, also bei den Anforderungen zu suchen. Deswegen war Kapitel 1 auch so ausführlich.

Manche Projekte sind aber auch erfolgreich.

Hier sind dann auch die Hauptgrnde aufgelistet, die befragte erfolgreiche Projekte fr ihren Erfolg nannten (Erfolgsfaktor). Die obersten drei haben mit Benutzern, Anforderungen und Management zu tun. In der Liste kommen technologische Faktoren gar nicht vor.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Was soll Projektmanagement verhindern?

- Klassische **Fehler** in der Software-Entwicklung:
 - ✓ Es wird direkt mit der Codierung angefangen
 - ✓ Das Vorgehensmodell fehlt bzw. wird nicht befolgt
 - ✓ Die Terminvorgaben sind unrealistisch
 - ✓ Die Weiterbildung der Mitarbeiter ist nicht zielgerichtet
 - ✓ Auswahl und Einsatz der Werkzeuge bzw. Methoden ist unzureichend vorbereitet
 - ✓ Risikomanagement wird nicht betrieben
 - ✓ Eine Abnahme der Phasenergebnisse erfolgt nicht
 - ✓ Es wird nicht systematisch bzw. unzureichend getestet
 - ✓ Anforderungen und Qualitätsziele werden nicht festgelegt

Quelle: Interne Untersuchung bei Bosch

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 302

Bei den übelsten Fehlern kommt Technik dagegen durchaus vor:

Ein Grund für dieses Phänomen mag sein: Bei Erfolgen wird funktionierende Technik als selbstverständlich vorausgesetzt. Scheitert dagegen die Hard- oder Softwaretechnik (z.B. Methoden und Werkzeuge), so fällt dies als Misserfolgsfaktor auf.

Die roten Haken stehen für besonders schlimme Fehler,

Die gelben sind nicht ganz so schlimm.

Beachten Sie, wie viele Fehler auf dem Gebiet des Managements liegen; aber auch die mangelnde Prozesstreue und –reife (zu früh codiert, kein Vorgehensmodell, keine Abnahme) ist bemerkenswert.

Sie können die beiden Listen als Checklisten verwenden, wenn Sie plötzlich zum Projektmanager berufen werden. Die Vorlesung bietet einen Einstieg in die wichtigsten Dinge, die Sie dann berücksichtigen müssen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Wie kompliziert die Organisation von Projekten ist, hängt natürlich von deren Größe und Schwierigkeit ab.

Das Grundmuster ist einfach:

Der Projektleiter spielt eine Schlüsselrolle. In kleinen Projekten ist er auf sich alleine gestellt, in großen steht ihm eine Assistenz und vielleicht sogar ein ganzer Stab (Risikomanager, Budgetverwalter) für Teilaufgaben zur Verfügung. Dann spricht man von der Projektleitung.

In einem kleinen Projekt ist ihr direkt das Team unterstellt. Die Mitarbeiter arbeiten dem Projektleiter zu, man sagt, sie „berichten ihm“.

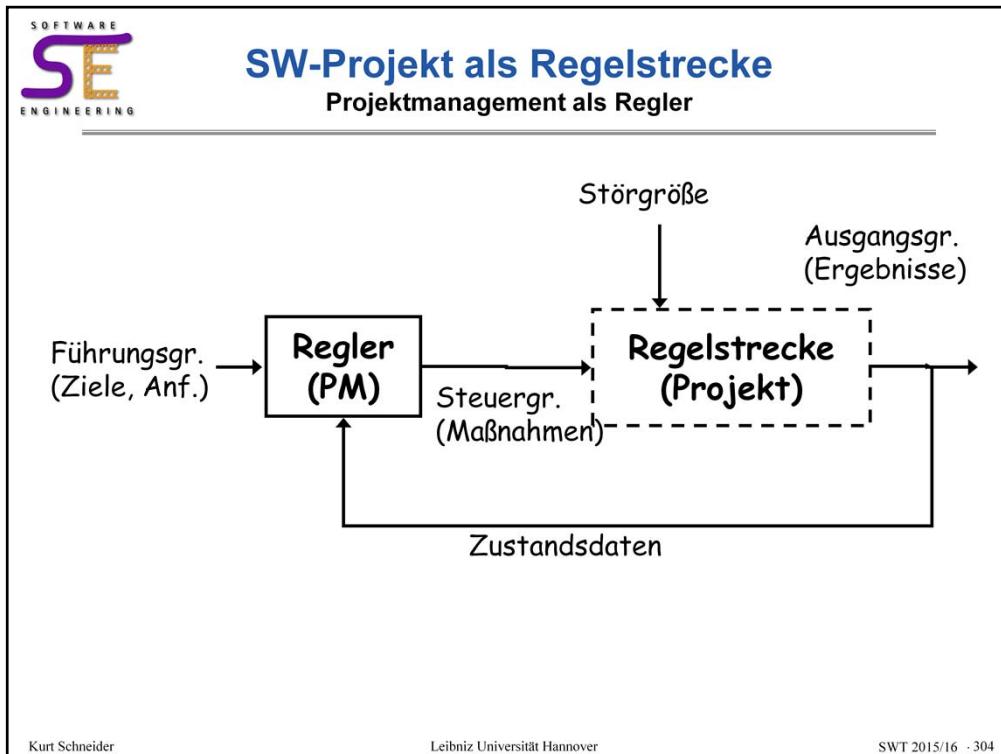
Natürlich arbeitet der Projektleiter für den Kunden. Der Kunde kann selbst wieder kompliziert zusammengesetzt sein aus verschiedenen Geschäftsbereichen oder Interessengruppen.

Öfter wird das eigene Management in diesem Bild vergessen. Das Projekt dient natürlich nicht nur dem Kunden, sondern eigentlich zuerst der eigenen Firma, Gewinn zu machen. Aus dieser doppelten Zuordnung ergeben sich öfter Probleme. Der Projektleiter muss diesen Spagat aushalten.

In großen Projekten ist „das Team“ noch komplizierter aufgebaut und meist hierarchisch in Teilteams bzw. Teilprojekt mit Untergruppen organisiert. Aber auch auf oberster Ebene kommt dann meist ein neues Konstrukt dazu, das z.B. „Steuerkreis“ oder „Lenkungsausschuss“ heißt. Das ist ein Gremium, das aus hochrangigen Vertretern des Kunden und der Entwicklungsorganisation besteht. Die Projektleitung ist dann zunächst diesem (einen) Gremien verantwortlich, was eine Erleichterung sein kann, weil die Doppelverantwortung wegfällt. Andererseits kann ein starker Lenkungsausschuss, der sich nur selten trifft und entscheidungsarm agiert, ein großes Projektrisiko darstellen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



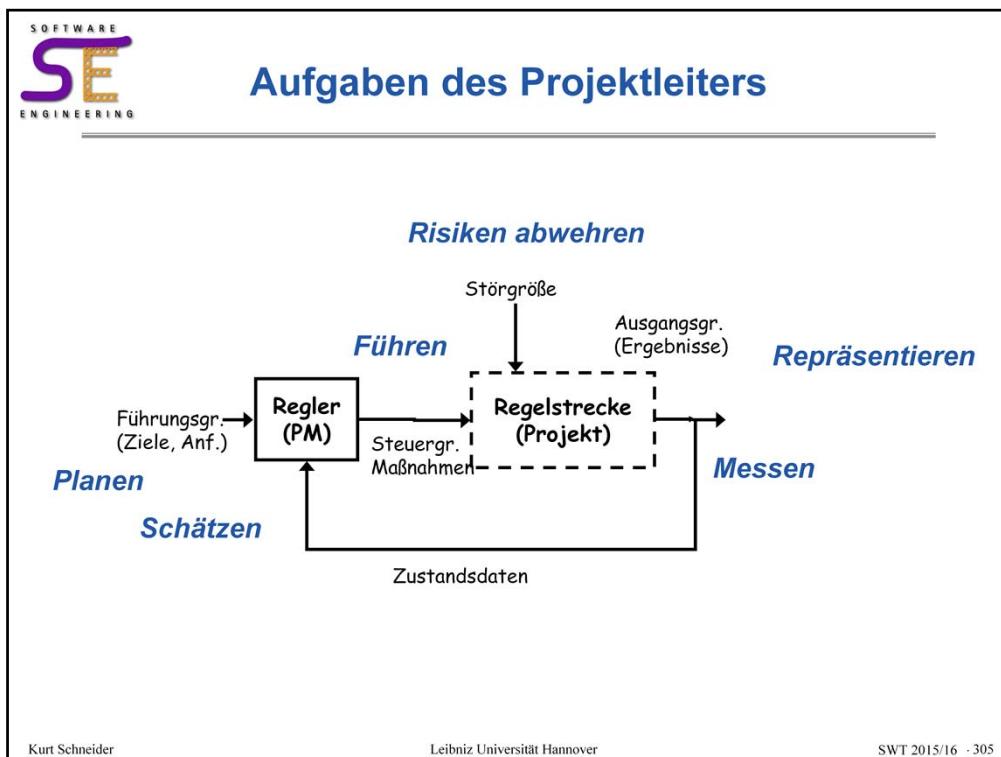
Grob kann man die Aufgabe eines PL auch mit einem Regler vergleichen:

- Ziele und Anforderungen geben die Richtung vor.
- Über Maßnahmen kann der PL das Projekt beeinflussen.
- Daneben wirken aber auch Störgrößen (Budgetkürzungen, Krankheit von Mitarbeitern, Anforderungsänderungen) auf das Projekt ein.
- Es erzeugt auf diese Weise Ergebnisse, die von den Anforderungen abweichen können.
- Dies bemerkt ein guter PL schon vor dem Projektende, indem die Zustandsdaten aus der Mitte des Projekts abgegriffen werden.

Damit gehören zu den wichtigsten PL-Aufgaben: Schätzen, planen und messen. Nur so kann der Stand erkannt und auch in veränderlichen Rahmenbedingungen weiter angepeilt werden.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Hier sind die Hauptaufgaben der Projektleitung noch einmal explizit den Teilen des Regelkreises zugeordnet:

Der Projektleiter plant, gibt damit ein Ziel vor.

Dann schätzt er aufgrund des Plans und ergreift entsprechende Maßnahmen, um das Projekt zum Ziel zu führen.

Dabei treten Störungen auf.

Sie sind erst zu erkennen, wenn durch Messen der wahre Zustand des Projekts ermittelt wird.

Er kann auf die Plan- und Schätzgrößen zurück abgebildet werden. Dadurch werden Abweichungen erkennbar, denen gegenzusteuern ist. Das geschieht mit neuen Steuergrößen (Führungsmaßnahmen).

Schließlich muss und darf der Projektleiter die Resultate des Projekts intern und extern repräsentieren.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Planen

- „Geistige Vorwegnahme des Kommenden“
- Ein Projekt muss geplant werden
 - Was erreicht werden soll
 - Welche Aufwände und Zeit es brauchen darf
 - Wann was getan werden muss
 - Wer zuständig ist, wie geprüft wird
- Planung ist ein Kompromiss
 - Schätzen, wie lange es dauert/wie viel man braucht
 - Berücksichtigen, wie viel der Kunde und die eigene Firma geben kann
 - Entscheiden, was die Vorgabe an das Projekt ist
 - Plan liegt meist zwischen Wunsch und Schätzung
- Forderung an ingenieurmäßiges Vorgehen:
 - Schätze nicht blind, sondern messe und vergleiche mit Deiner Schätzung.
Auf dieser Basis schätze das nächste Mal besser.“

Wichtig: Die Schätzung ist eine erste Basis.

Oft zeigt sie, dass das Projekt zu lang dauern oder zu viel kosten würde.

Konsequenz muss sein: Abstriche an Anforderungen machen. Dies kann man von Kunden umso seriöser verlangen, je klarer man anhand von Vergleichsdaten macht, dass der gewünschte Umfang nicht im gesteckten Rahmen zu erbringen ist.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Auf den nächsten Folien werden die Teilaufgaben der Projektplanung angesprochen. Hier zunächst der Überblick.

Links sind die Aspekte zu sehen, die der Reihe nach (von oben nach unten) geplant werden müssen. Rechts stehen die Ergebnisse der jeweiligen Planungsschritte.

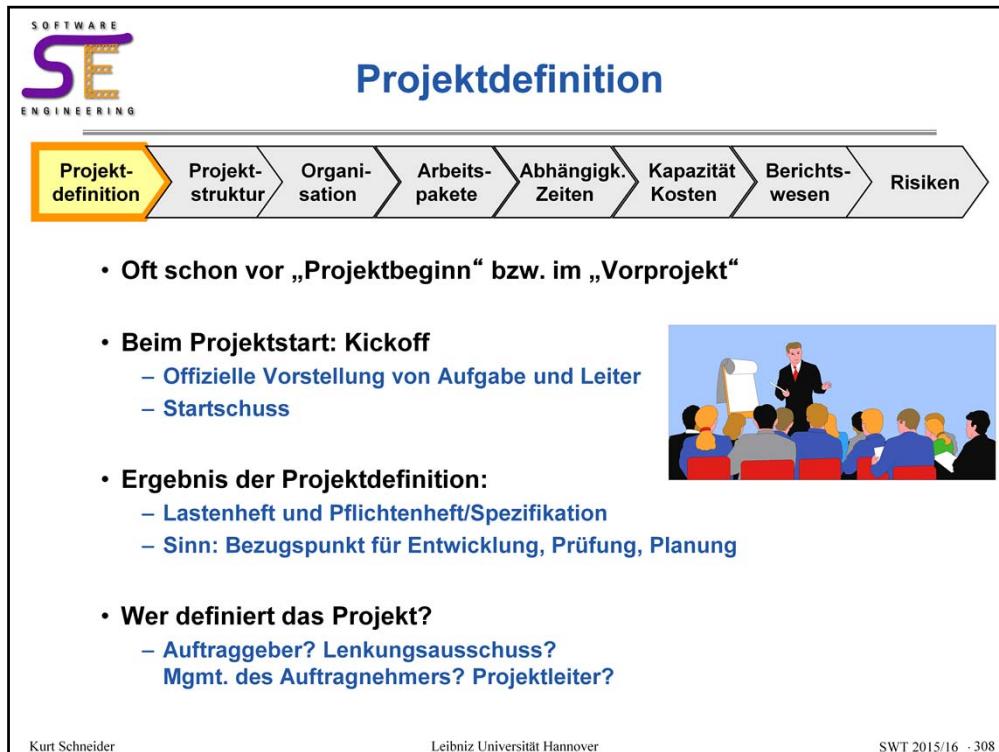
Einige davon sind orange, diese wurden in der Vorlesung schon besprochen. Andere sind blau, das sind die Ergebnisarten, die hinten noch genauer betrachtet werden.

Die schwarzen Ergebnisse werden nur kurz erklärt.

Im unteren Teil der gesamte Planungsprozess, der nun schritt für Schritt durchgegangen wird.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



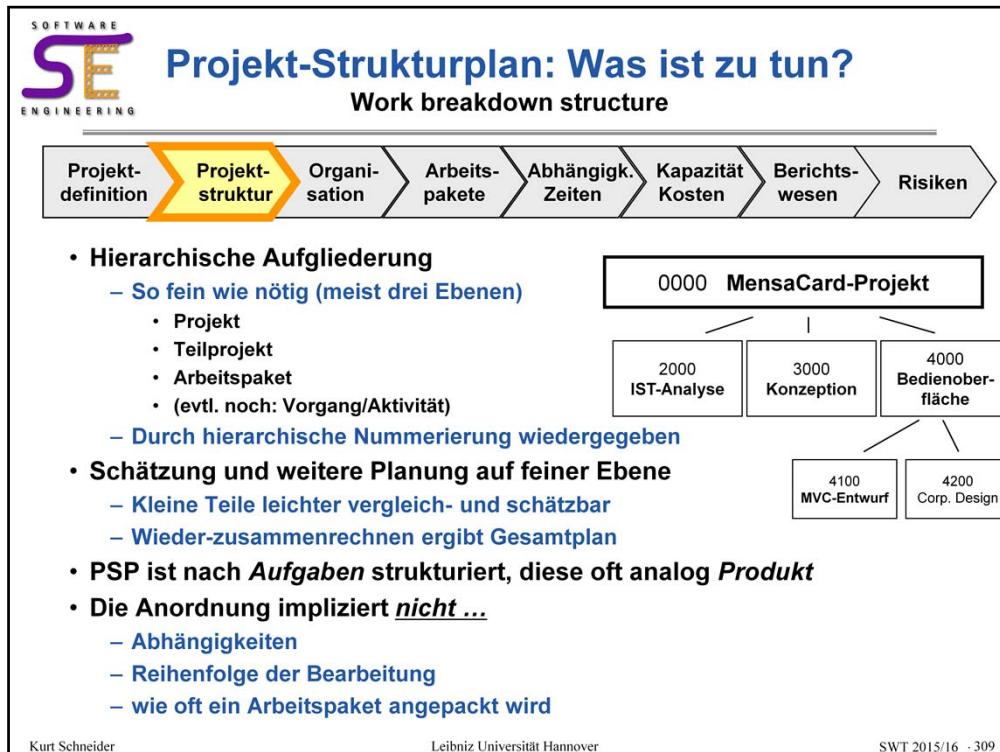
Bei größeren Projekten ist nicht einfach zu sagen, wann sie eigentlich beginnen. Das ist aber wichtig, weil sie dann organisatorische, finanzielle und Führungs-Konsequenzen auslösen.

Daher werden die Aufgaben des Projekts oft schon vor seinem Beginn geklärt. Das führt zu Lasten- und Pflichtenheften, zumindest in der klassischen Entwicklung, die solche Dokumente ausführlich fordert.

Im Kickoff wird das Projekt offiziell gestartet. War die Vorbereitung sehr aufwändig, so wird sie oft einem eigenen kleinen Projekt, dem „Vorprojekt“ übertragen. Das Vorprojekt soll die Planung und Vorbereitung des Hauptprojekts leisten.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Projektstrukturpläne oder Work-Breakdown-Structures sind weit verbreitet und sehr nützlich.

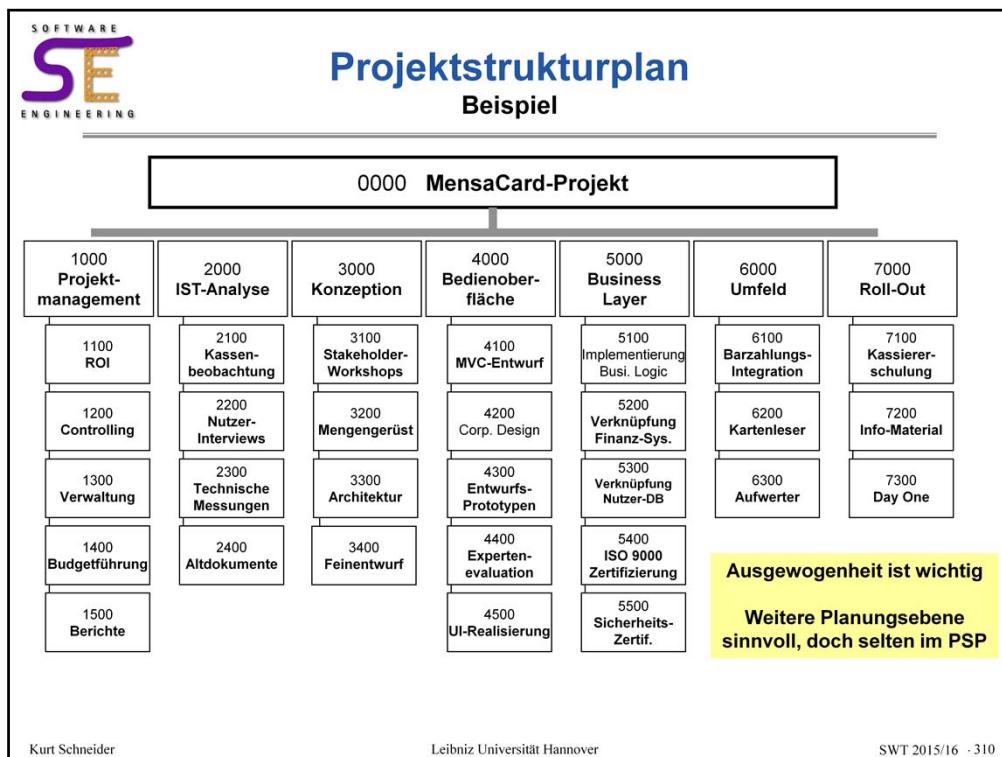
Die Diagramm könnten aussehen wie ein Baum, die Darstellung auf der nächsten Folie ist aber gebräuchlicher.

Wichtig ist, die Intention dieses Modells zu verstehen. Man möchte hiermit klären, welche Aufgaben zu erledigen sind. Dazu die hierarchische Teilung bis auf ein handhabbares Granulat. Drei Ebenen (wie hier, nur vollständiger, siehe nächste Seite) ist üblich.

Wenn mehrere Arbeitspakete (APs) unter einem Teilprojekt stecken, bedeutet das nur, dass alle diese Aufgaben zu erfüllen sind, um die Aufgaben und Ziele des Teilprojekts zu erreichen. Eine Reihenfolge der Teilschritte scheint oft impliziert zu sein, sie ist aber nicht vorgesehen: Der PSP sagt nur, was zu tun ist, aber nicht wann, von wem, oder in welcher Reihenfolge.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Dies ist ein Beispiel für einen weit entwickelten PSP. Noch viel ausführlicher sollte ein PSP dann schon nicht mehr werden.

Eine weitere Granularitätsebene („Vorgänge“) würde das Bild schwer lesbar machen.

Die vierst gigen Zahlen sind eigentlich vierstufige Hierarchiekennzeichen. Auf der ersten Ebene (Projekt) steht die Nummer 0. Darunter wird die erste Stelle weitergez hlt, wieder darunter die zweite Stelle und so weiter.

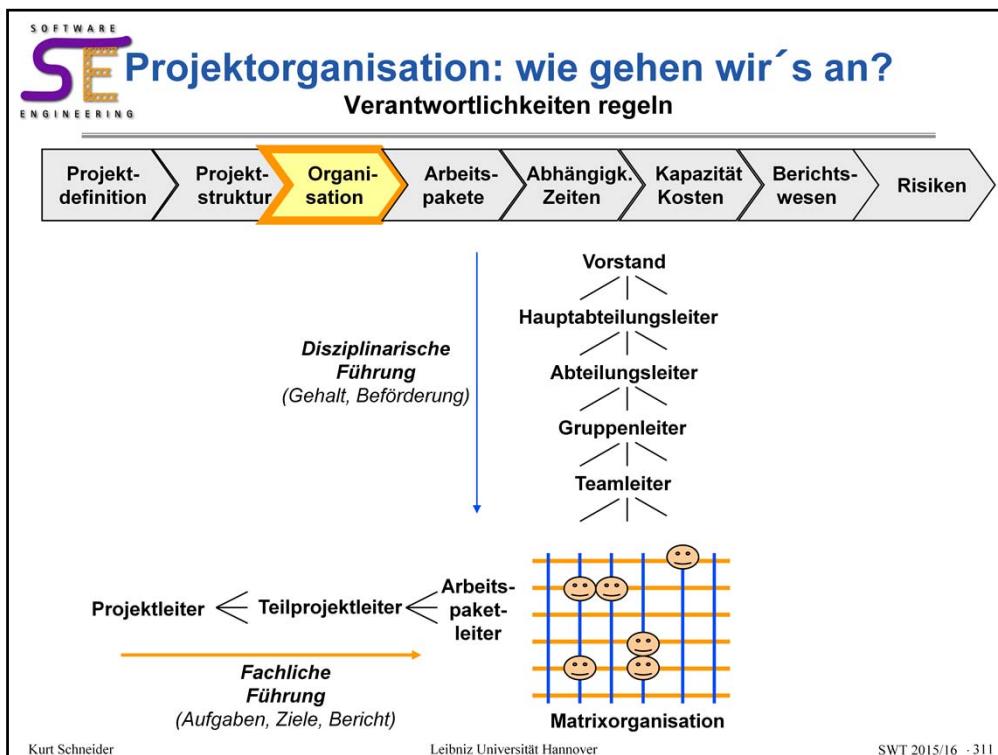
Durch Angabe einer vierstelligen Nummer weiß man sofort, wo im PSP der Vorgang, das Arbeitspaket usw. steht.

Es kann übrigens durchaus sinnvoll sein, das obige PSP zwar noch um eine Ebene zu verfeinern, diese Ebene aber nicht mehr grafisch aufzumalen.

Dennoch ist dann klar, dass 5210 der Vorgang „10“ innerhalb von AP 5200 ist.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Zunächst einmal gibt es die Linienorganisation. Sie folgt der disziplinarischen Führung.

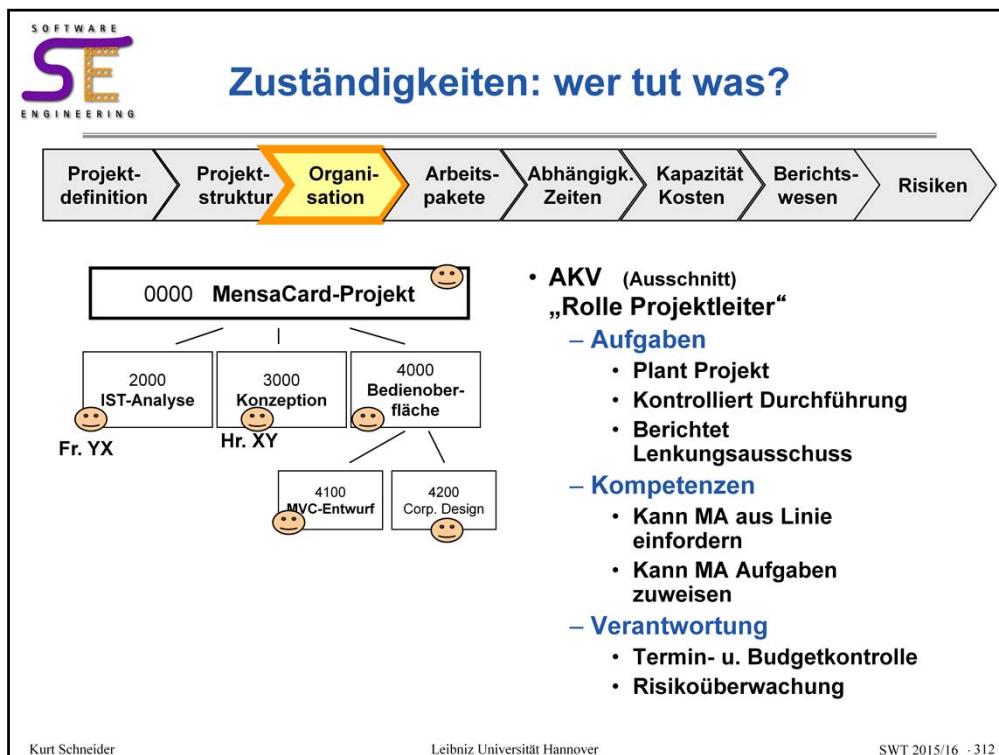
So sind die meisten Firmen (primär) organisiert. Solange kein Projekt anliegt, stecken alle Mitarbeiter an ihrer Stelle in der Linienorganisation.

Manche Firmen sind nur nach Projekten organisiert, aber das ist eher selten. Denn auch die Zeit zwischen den Projekten muss ja abgedeckt sein.

Häufig und beliebt, aber nicht unproblematisch ist die Kombination der beiden Organisationsformen, die sogenannte Matrixorganisation. Wie oben dargestellt hat nun jeder Mitarbeiter seinen Platz in der Linie, wird aber während des Projekts an dieses „entliehen“ bzw. dem Projekt zugeordnet. Während der Projektlaufzeit überwiegt die fachliche Zuordnung zum Projekt. Dennoch hat der Mitarbeiter auch die andere, disziplinarische Seite über sich.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Sobald man weiß, was zu tun ist und wer dafür in Frage kommt (Team), müssen die einzelnen Personen konkreten Aufgaben (Vorgängen, APs usw.) zugewiesen werden.

Im Überblick können die Namen an den PSP geschrieben werden.

Schließlich braucht man aber eine genauere Aufgabenbeschreibung. Dazu haben sich Techniken wie die AKV etabliert. Eine Kompetenz ist darin etwas, das man tun darf (nicht eine geistige Fähigkeit).

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

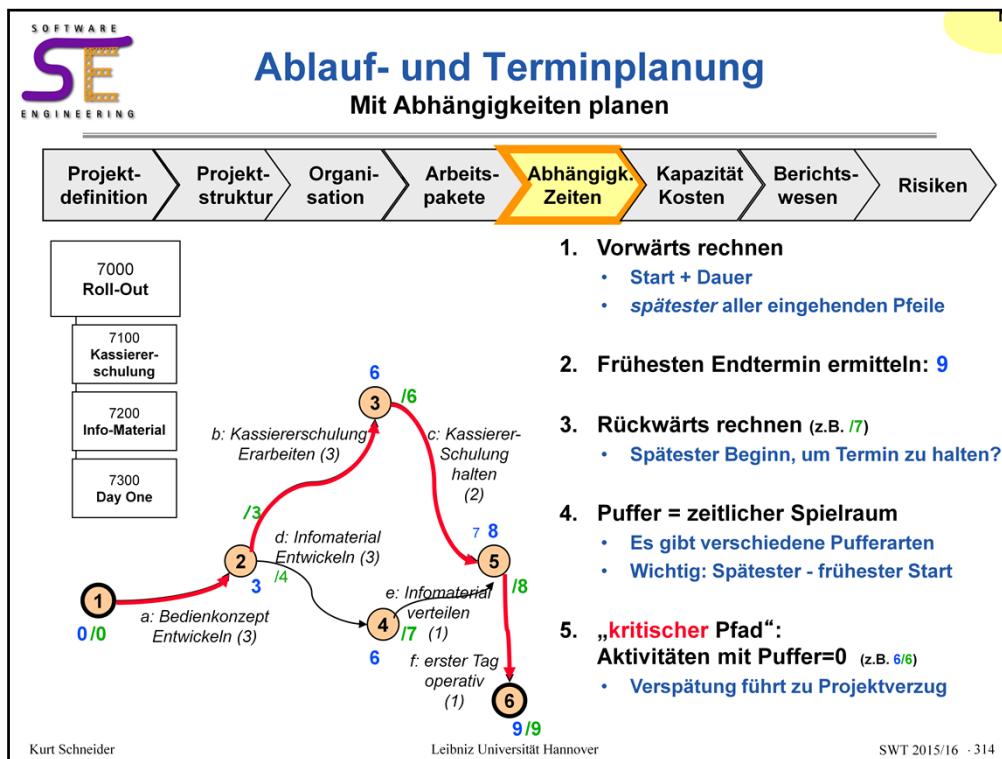
Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Arbeitspakete sind der Bezugspunkt für Schätzungen, auch oft für finanzielle Fragen. Daher sind APs ziemlich genau zu beschreiben. Oben steht, was wirklich beschrieben werden sollte. Das sieht nach viel aus und ist auch einige Arbeit. Aber in klassischen Organisationen (nicht gerade in agilen) wird dieser Aufwand in der Regel getrieben.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Nun werden die Aufgaben auf Abhängigkeiten untersucht. Danach kann man einen Zeitplan erstellen.

Der Netzplan besteht aus Aktivitäten (Pfeilen) und Ereignissen bzw. Zuständen (Kreisen). Dazu kommen quantitative Angaben.

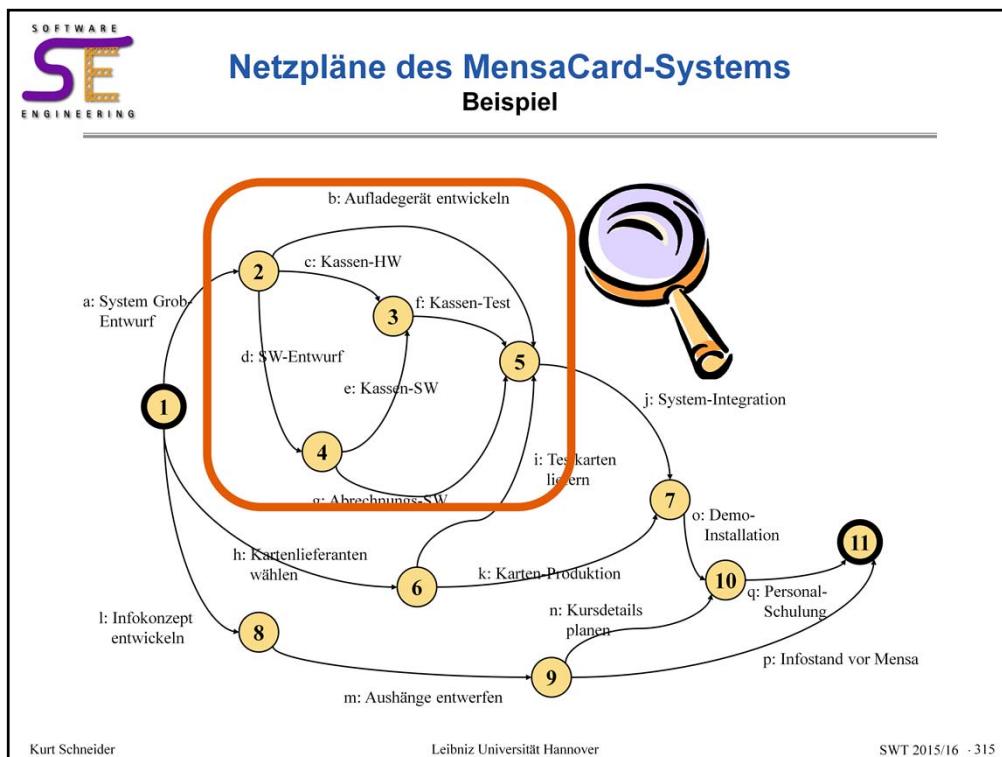
Der Netzplan wird zunächst strukturell/qualitativ aufgebaut. Was muss fertig sein, damit etwas anderes beginnen kann? Alle in einen Kreis eingehenden Tätigkeiten müssen beendet sein, bevor (alle) daraus ausgehenden Tätigkeiten beginnen können. Aus bekannten Abhängigkeiten erstellt man den Graphen. Beispielsweise können in den Arbeitspaketbeschreibungen auch Abhängigkeiten formuliert sein: Man schreibt, welche Dokumente man braucht, und schon ergibt sich eine Abhängigkeit.

Nun kommt die Berechnung, wann man frühestens fertig sein kann und wann man dazu spätestens anfangen muss. Der Algorithmus steht rechts und ist durch die Zahlen am Beispiel nachvollzogen.

Der kritische Pfad ist rot eingezeichnet: hier gibt es keinen Spielraum; wer hier zu spät beginnt, verzögert das ganze Projekt. Im Beispiel kann man aber den unteren Ast (über Zustand 4) noch hin oder herschieben, weil es hier Puffer gibt.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Die Netzplantechnik ist ein Planungsinstrument, das keinesfalls auf SW-Projekte beschränkt ist, aber auch ihnen besonders gut tut.

Immer dann, wenn Sie ein Projekt in eine recht kompliziert voneinander abhängige Menge von Teiltätigkeiten aufspalten können, bietet sich die Netzplantechnik an.

Es gibt verschiedene Varianten, oben finden Sie die sogenannte „Critical Path Method CPM“. Dabei sind die Kreise „Ereignisse“, die hier nur zur Identifikation durchnummeriert sind. Das wichtigere Element sind die Tätigkeiten, die als Pfeile zwischen Ereignissen verlaufen.

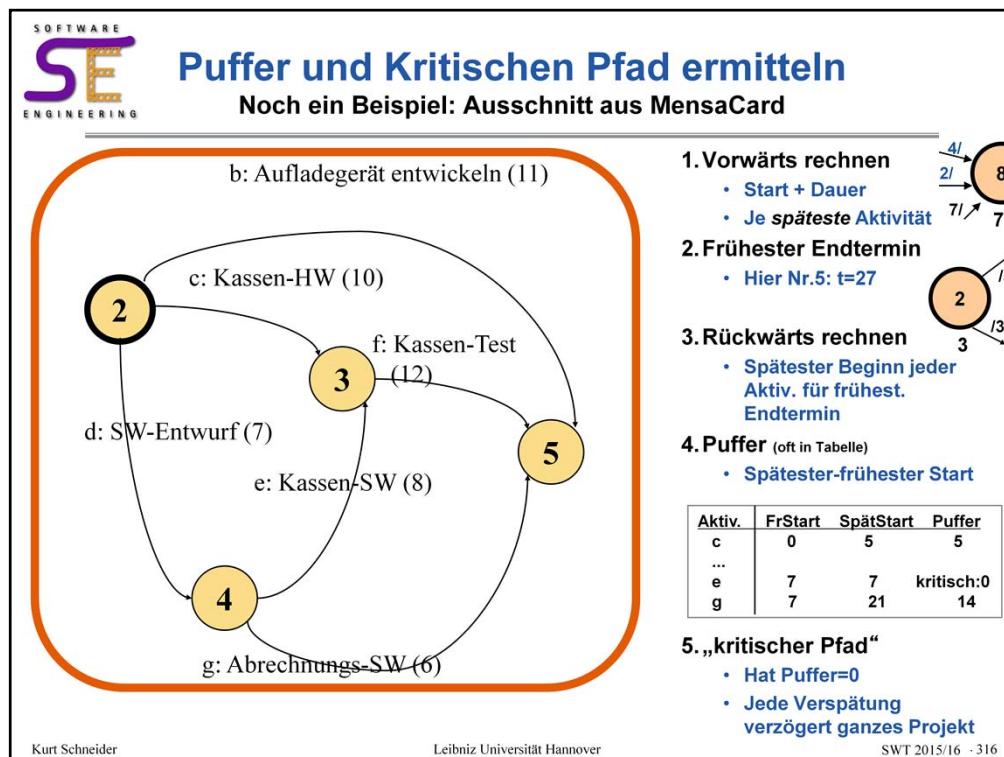
Die Tätigkeiten werden hier durch einen Buchstaben identifiziert und durch eine kurze Benennung beschrieben. Die Semantik lautet: Wenn mehrere Tätigkeiten in einen Ereigniskreis eingehen und andere herauslaufen, dann müssen alle eintreffenden abgeschlossen sein, bevor eine der auslaufenden beginnen kann. Es gibt ein Start- und ein Endereignis, die fetteren Kreise.

Zyklen sind verboten.

Im folgenden wird ein Teilnetz näher betrachtet, um schneller zum Punkt zu kommen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Schon der reine Netzplan hilft, die Abhangigkeiten explizit zu machen und im Auge zu behalten. Mit Netzplanen kann man aber auch rechnen, und auf den Rechnungen konnen vielfaltige Planungen aufsetzen.

Zunächst ermittelt oder schätzt man für jede der Tätigkeiten ihre Dauer, hier in Klammern angegeben. Es gibt kompliziertere Varianten, in denen man sogar ganze Wahrscheinlichkeitsverteilungen für die Dauer angeben kann (z.B. PERT). Unsere CPM kann das nicht.

Nun rechnet man vorwärts:

- Jede Tätigkeit vom Startereignis ab hat als frühesten Startzeitpunkt die „0“ bzw. den Starttermin des Projekts. Man schreibt diesen Termin an alle diese Tätigkeitspfeile.
 - Nun folgt man jedem dieser Pfeilen vorwärts, addiert ihre Dauer und schreibt das Ergebnis an ihre Zielereignisse.
 - Für jedes (Ziel-)ereignis lässt man nur die höchste Zahl, also den spätesten Startzeitpunkt stehen und löscht alle anderen. Es müssen ja alle, also auch die letzte einlaufende Tätigkeit fertig sein, bevor die auslaufenden Tätigkeiten beginnen können.
 - Der resultierende Zeitpunkt ist der früheste Startzeitpunkt aller auslaufenden Tätigkeiten. Dieses Prinzip setzt sich bis zum Endknoten fort.

Nun rechnet man rückwärts, vom frühesten Endzeitpunkt aus, den man gerade errechnet hat.

- Entlang aller Tätigkeiten, die in ein Ereignis einlaufen, subtrahiert man deren Dauer und erhält damit den spätesten Zeitpunkt, an dem sie starten dürfen (so dass doch noch der früheste Endzeitpunkt erreicht wird).
 - Dabei gilt für jedes Ereignis: es ist der minimale Wert, also der früheste (von allen spätestmöglichen) Startzeitpunkten zu wählen (damit auch der letzte noch fertig wird)
 - Alle Tätigkeiten, bei denen frühester und spätester Startzeitpunkt identisch sind, heißen „kritisch“, sie bilden den oder die „kritischen Pfad(e)“. Verspätet sich eine Tätigkeit auf einem kritischen Pfad, so wird sich das gesamte Projekt verspätzen. Der PL muss darauf achten, dass das nicht passiert.
 - Die nicht-kritischen Aktivitäten haben einen „Puffer“ zwischen dem frühesten und dem spätesten Startzeitpunkt. Der PL kann ihren Start verschieben, um Personal- und Ressourceneinsatz gleichmäßiger zu verteilen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

ProFLOW für Netzpläne

- Es gibt mehrere Werkzeuge für Netzpläne
- hier implementiert in ProFLOW von FG SE
- Zum Zeichnen und zum Berechnen → Tabelle der Zeiten u. Puffer

Berechnung für Aktivität b(3):
Min($\text{E}_\text{F}(2), \text{E}_\text{F}(3)$) =
Min(3, 2) = 2

```
graph LR; 1((1)) -- "a(2)" --> 2((2)); 2 -- "b(3)" --> 3((3)); 2 -- "c(1)" --> 4((4)); 3 -- "d(2)" --> 4; 4 -- "e(4)" --> 5((5));
```

Aus der Bachelorarbeit von Michael Gross (2009) am FG SE

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 317

Ein Beispiel aus einem Werkzeug. Es errechnet die Zeiten und die Puffer.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Meilensteine

- Geplanter Zustand des Projekts zu best. Zeitpunkt

- Geplanter Zeitpunkt
- Geplanter Entwicklungsstand
 - Zustand von Dokumenten, Code, Prüfungen
 - Jeder Aspekt möglich (z.B. technisch, organisatorisch)
 - Entscheidungs-relevante Bündelung von Teil-Zuständen
- Erreichungskriterien klar definiert

Meilenstein-Symbol
in Plänen; dazu
Beschreibung nötig

- Zweck

- Klares, inhaltlich definiertes (komplexes) Kriterium für Projektfortschritt
- Bewusst und spezifisch gesetzt, um Klarheit zu gewinnen
 - Innerhalb Projekt: **interner Meilenstein**, für interne Planung
 - Nach außen/zum Kunden: **externer Meilenstein**
- An geplanten Meilenstein-Zeitpunkten wird entschieden
 - Ist Meilenstein erreicht?
 - d.h.: geplanter Entwicklungsstand genügt den definierten Kriterien
 - Dagegen reicht NICHT: der geplante Zeitpunkt ist erreicht
 - Und damit: Fortschritt wie geplant?
 - Sind Änderungen im Plan nötig?

Kurt Schneider

Leibniz Universität Hannover

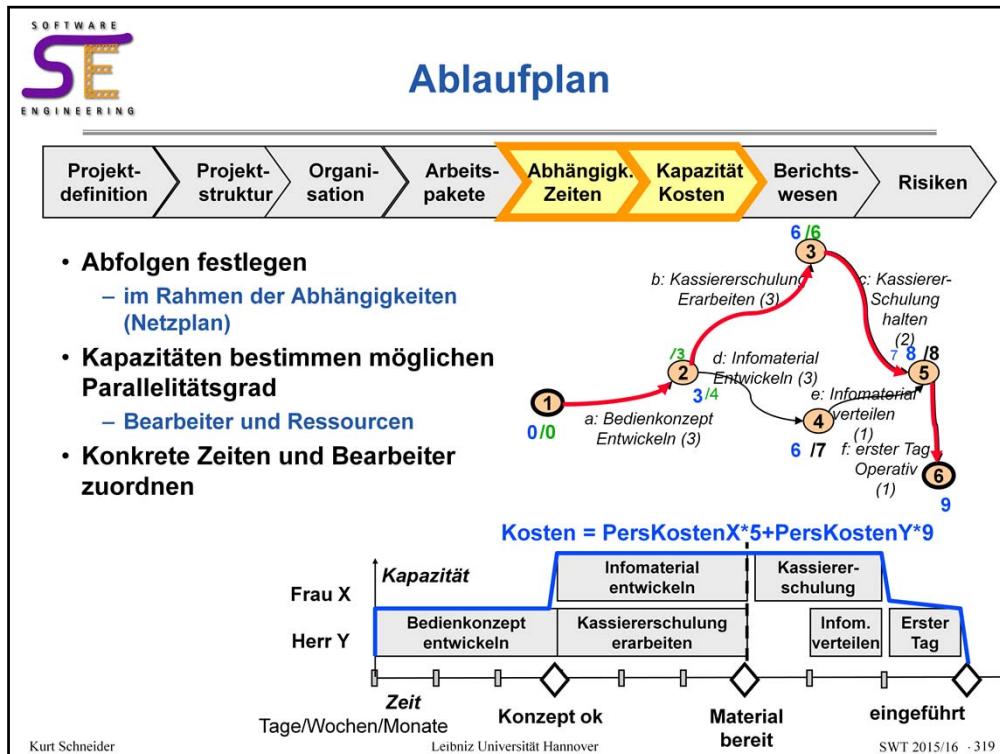
SWT 2015/16 · 318

Meilensteine sind Hilfen für die Planung.

Sie fassen zusammengehörige Aspekte zusammen und erlauben später, einfach den Fortschritt zu „messen“ (Kriterien, anhand derer man feststellen kann, ob der Meilenstein erreicht ist oder nicht).

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Nun werden die nötigen Aktivitäten in eine Reihenfolge gebracht, Bearbeitern und Ressourcen zugeordnet und ihre Kosten werden abgeschätzt.

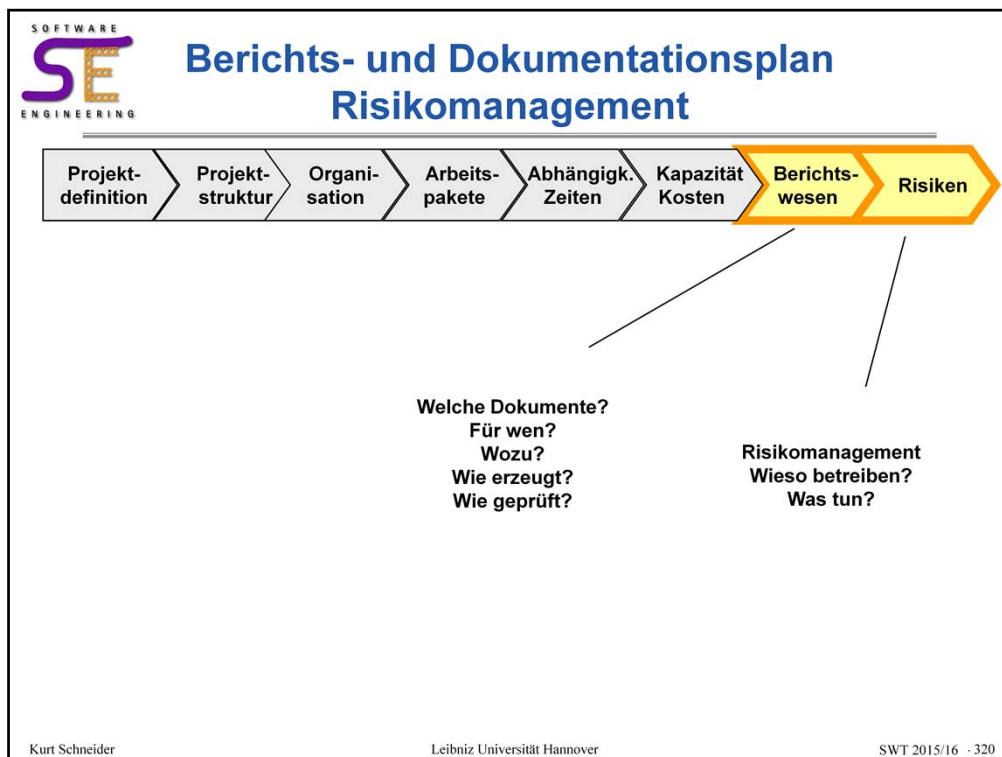
Die Aktivitäten aus dem Netzplan sind hier in einer Weise angeordnet, die die Abhängigkeiten berücksichtigt, aber auch die vorhandenen zwei Bearbeiter nutzt: Die Aktivitäten werden ausgetauscht, eine Unterbrechung bei Hr. Y (Urlaub?) wird berücksichtigt.

Drei Meilensteine an wichtigen Punkten stellen fest, ob das Projekt inhaltlich schon so weit ist. Der zweite Meilenstein verlangt, dass zwei Ergebnisse vorliegen. Das kann man hier nicht sehen, nur vermuten. Die Meilensteinbeschreibung gibt Gewissheit.

Kosten entstehen, wenn Ressourcen bezahlt werden müssen – egal, ob und was sie arbeiten. Oben ist es gut gelöst, nur Hr. Y wird kurz bezahlt, obwohl er nicht arbeitet. Nicht immer gelingt es aber, Mitarbeiter nur exakt für den Wunschzeitraum ins Projekt zu nehmen. Dann kostet das Personal mehr als die reinen Tätigkeiten vermuten lassen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Softwaretechnik

Inhalt von Kapitel 7



7. Management von Technik und Projekt

- Versions- und Konfigurationsmanagement
- Aufgaben von Projektleitern
- Aufwandsschätzungen
- Risikomanagement in Softwareprojekten
- Agile Softwareentwicklung

Leibniz Universität Hannover SWT 2015/16 · 321

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The slide is titled "Aufwandsschätzung mit COCOMO" and is based on Barry Boehm's work from 63 TRW projects. It features the Software Engineering logo (SE) and includes a table of parameters for Organic, Semidetached, and Embedded projects.

Boehm hat Formeln ermittelt:

- Ausgangsbasis: **Schätzung f. KLOC**
- $Aufwand = a * KLOC^b$
- $Dauer = c * Aufwand^d$
- $Personal = Aufwand/Dauer$

Parameter je nach Projektart:

	Organic	Semidetached	Embedded
a	2,4	3,0	3,6
b	1,05	1,12	1,2
c	2,5	2,5	2,5
d	0,38	0,35	0,32

Beispielrechnungen:

- Kleine Projekte** (z.B. im Studium)
3 KLOC, organic
7,6 PM Aufwand; 5,4 Monate; 1,4 Pers.
- 6 KLOC, organic
15,7 PM Aufwand; 7,1 Monate; 2,2 Pers.
- Größere Projekte** (Schätzung * 100)
600 KLOC, organic
1983 PM Aufwand; 45 Monate; 44 Pers.
- 600 KLOC, semidetached
3878 PM Aufwand; 45 Monate; 86 Pers.

(„parametrisiertes“ Verfahren)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 322

COCOMO von B. Boehm geht von einer Umfangsschätzung aus (in tausend Lines of Code, kLOC bzw. KDSI). Dann gibt es Formeln, die – nach Anwendung von Korrekturfaktoren – Aufwand, Dauer und Personal schätzen.

Die kleine Tabelle zeigt, welche Werte die Formel annimmt. Durch den Exponenten leicht über 1,0 ergibt sich ein schwach-exponentielles Wachstum: Der Aufwand für ein doppelt so umfangreiches SW-Projekt ist damit etwas mehr als doppelt so hoch (Verwaltungs- und Management-Overhead).

Die Beispielsrechnungen rechts zeigen dieses nicht-lineare Verhalten:

Zunächst ein kleines Projekt, dann eines mit doppeltem geschätztem Umfang.

Und dann davon das Hundertfache: führt zu weit über hundertfachem Aufwand, aber nicht zu hundertfacher Dauer (weil man ja mehr Leute beschäftigt).

Wenn man dann noch von einem etwas schwierigeren Projekt (semidetached) ausgeht, verdoppelt sich der Aufwand noch einmal, aber die Dauer bleibt (praktisch) gleich: alles kann durch mehr Personal abgefangen werden.

Natürlich kann man und muss man oft von den COCOMO-Schätzungen abweichen. In der Regel kann man nicht exakt 2,2 Personen beschäftigen, zumal dies ja nur ein Mittelwert ist. Aber laut Boehm ist die Schätzung ein guter Anhaltspunkt, jede Abweichung erst einmal negativ.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Häufig verwendet: Function Points

Details in der Originalveröffentlichung: Albrecht, A.J. (1979): Measuring Application Development Productivity. Proc. IBM Applic. Dev. Symposium, Monterey, CA, pp 83-92

Measurement parameter	Count	Weighting factor			=	Function Points
		Simple	Average	Complex		
Number of user inputs	12	x 3	4	6	=	48
Number of user outputs	20	x 4	5	7	=	100
Number of user inquiries	5	x 3	4	6	=	20
Number of files	3	x 7	10	15	=	30
Number of external interfaces	2	x 5	7	10	=	14
Count - total					→	212

* 14 Korrekturfaktoren = FP

Prinzip: In Std-Komponenten zerlegen; Analogieschätzung auf diesen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 323

Einen anderen Weg geht die sehr bekannte Function Point–Methode:

Es handelt sich um eine Analogieschätzung auf Komponenten, jedoch nicht auf anwendungsspezifischen, relativ großen Komponenten, sondern auf recht kleinen Standard-Operationen (die fünf links benannten).

Man zählt die vermutlich benötigten Operationen der fünf Arten, multipliziert sie mit einem Vektor von Komplexitätsfaktoren. Die Summe dieser Punkte wird noch einmal mit Korrekturfaktoren multipliziert (zur Anpassung an die eigene Situation; Faktor schwankt um 1). Das Endresultat sind die „Function Points“, und dafür gibt es nun viele Erfahrungswerte:

Wenn man also weiß, wie viel man für einen FP im Schnitt braucht, kann man Aufwand, Umfang usw. eines Projekts abschätzen, dessen FP-Zahl man kennt.

Es gibt Varianten des Verfahrens, z.B. Feature Points.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The logo for Software Engineering (SE) features the word "SOFTWARE" in small black capital letters above the letters "SE" in large, stylized, yellow and purple block letters. Below "SE" is the word "ENGINEERING" in small black capital letters.

SOFTWARE
SE
ENGINEERING

Ansätze für Aufwandsschätzung

- Analogie-Methode (Basis: Umfangsschätzung)
 - Software-Umfang schätzen
 - Vergleich mit abgeschlossenen Projekten (Dreisatz über Umfang)
- Prozentsatz-Methode (Basis: Erfahrungswerte für Phasen)
 - Basis: Bereits erbrachte Aufwände für Systemanalyse
 - Hochrechnen mit Systemanalyseanteil anderer Projekte
 - Beispiel:
 - Wir haben 6 PM Systemanalyse getrieben
 - Bei ähnlichen Projekten macht Systemanalyse 20% aus
 - Wenn es hier wieder 20% sind, wird Gesamtaufwand also 30 PM sein
- Expertenschätzungen (z.B. Delphi)
- Parametrierte Verfahren (COCOMO, Function Points)
 - Ermittlung von Aufwand, Dauer und Personalstärke über Formeln

Die Analogiemethode wurde schon angesprochen: Man vergleicht mit ähnlichen Projekten oder ähnlichen Komponenten.

Bei der Prozentsatzmethode geht man von der Hypothese aus, dass die Verteilung des Aufwands z.B. über die Phasen immer ähnlich ist. Demnach kann man messen, wie viel Aufwand bis zum Ende einer Phase P aufgelaufen ist. Weiß man aus Erfahrung, dass bis zum Ende von P meist 35% des Gesamtaufwands gebraucht werden, kann man sich ausrechnen, dass man auch im aktuellen Projekt nach Phase P noch rund zwei Drittel des Aufwands vor sich hat. Man kann solche Erfahrungswerte mit Vorteil auch an erfolgreichen Quality Gates (statt an Phasen) festmachen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Eine solide Schätzung geht aus von einer

- Analogie zwischen ganzen Projekten; aber das ist selten, so sehr gleichen sich zwei Projekte kaum
- Oder von der Zerlegung des aktuellen Projekts in Komponenten, die dann noch gewichtet werden und einzeln geschätzt.

Auf der Basis der Umfangsschätzung setzt die Aufwandsschätzung (und ggf. die Ressourcenplanung) auf. Es empfiehlt sich, zum Abschluss des mehrstufigen Verfahrens noch einmal Plausibilitäts-Validierungen gegen Projektdaten zu machen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Messen

- Ziel: Fortschritt (Funktion u. Qualität) feststellen
- „Einfaches“ Beispiel: was misst „Lines of Code“
 - Funktionsumfang?
 - Leistung der Programmierer?
 - Komplexität des Programms?
 - ... oder einfach die Länge des Programms?
- Was kann man aus den Ergebnissen schließen?
 - Auf welcher Skala liegen die Messungen?
 - Was tut man damit?
- Generell: Vorsicht beim Interpretieren!
 - Gefährlicher Ansatz: „was können wir denn leicht messen?“
 - Ganz anderes Prinzip: Goal-Question-Metric

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 326

Schon bei der Diskussion der Modelle war aber klar geworden:

Die wenigsten Metriken messen direkt den (Qualitäts-) Aspekt, zu dessen Beurteilung sie herangezogen werden.

Vielmehr liegt eine mehr oder weniger komplizierte Modellabbildung dazwischen. Im Fall der zyklomatischen Komplexität ist sie explizit angegeben – und die Modellabbildung liegt durchaus nicht auf der Hand.

Noch schlimmer ist, wenn die Modellabbildung implizit oder unklar bleibt.

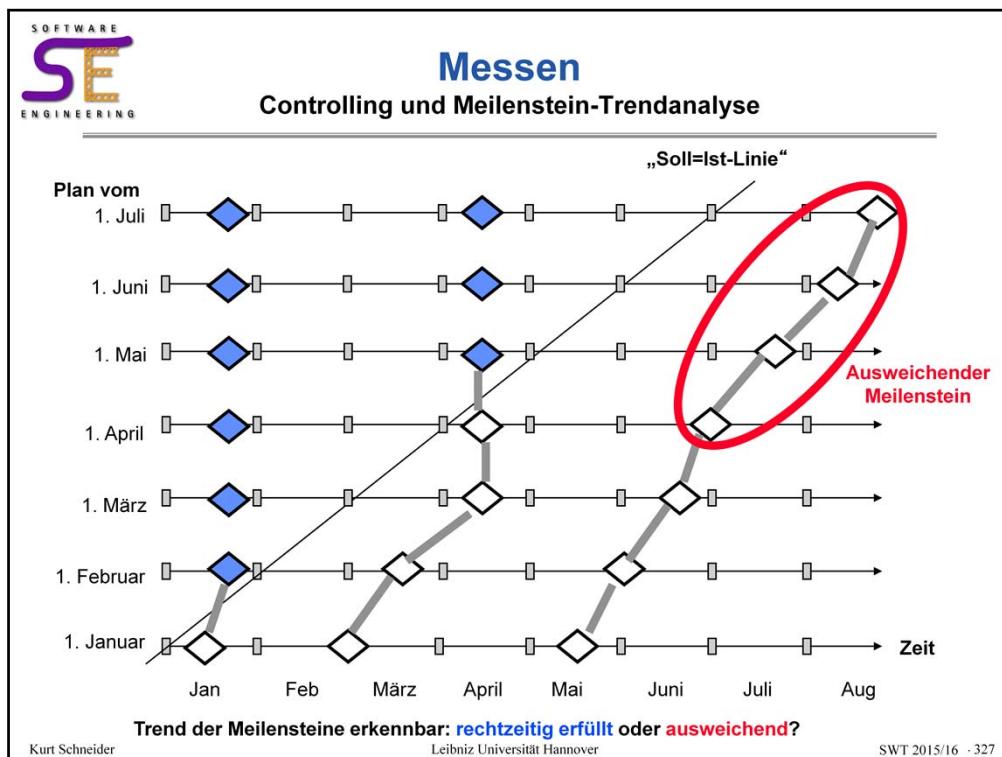
Nicht selten werden die Modellattribute (also z.B. die Resultate einer Metrik) in unzulässiger Weise verwendet: so können Ordinalzahlen nur die Reihenfolge von Alternativen angeben (1 ist besser als 2), aber nicht den Abstand bewerten (2 ist nicht doppelt so schlecht wie 1).

Metriken haben eindeutig Modellcharakter.

Wie bei Modellen immer, so ist auch bei Metriken genau zu klären, worin die Modellabbildung besteht, die ihnen zugrunde liegt. Das wird oft versäumt. Die Methode GQM dagegen macht diesen Fehler nicht. Sie ist aber sehr aufwändig.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Die Meilensteintrendanalyse misst, ob Meilensteine pünktlich erfüllt werden, oder ob sie sich ständig verschieben.

Man beginnt mit dem ursprünglichen Plan, dem untersten Zeitstrahl.

Bei der ersten Überprüfung stellt man fest, ob Meilensteine erfüllt sind (volle Raute) oder nicht. Auch können neuen Zeitpunkte eingeplant werden. Hier ist das für alle Meilensteine geschehen, der erste wurde aber dennoch inzwischen erreicht. Um den muss man sich keine Sorgen mehr machen, er bleibt, wo er war, als er erfüllt wurde.

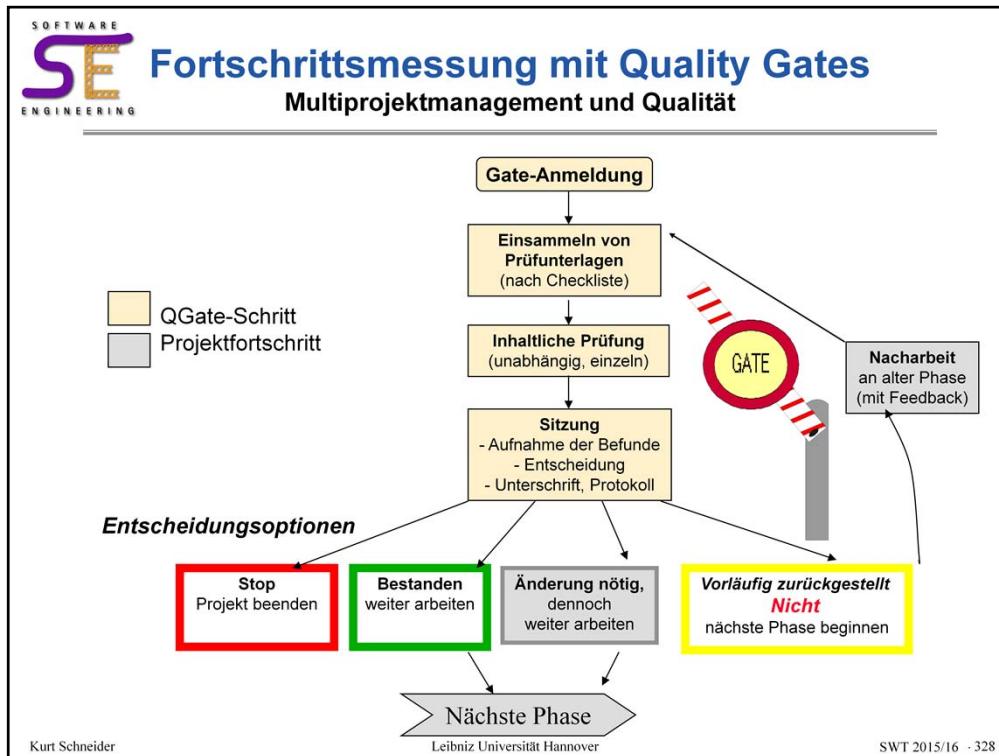
Beim zweiten Meilenstein zeigt der Trend in Richtung Verzögerung. Erst im Mai ist der für April/Mai geplante Meilenstein endlich erreicht. Ursprünglich sollte er im März erreicht werden.

Beim dritten Meilenstein ist noch gar nicht abzusehen, wann und ob er erfüllt wird. Bisher ist er durch Verschiebung immer weiter ausgewichen.

Die diagonale Linie zeigt, wann ein Planzeitpunkt erreicht ist. Meilensteine links oben müssen erfüllt sein – einen nicht erfüllten Meilenstein würde man ja nicht in der Vergangenheit ansiedeln. Dagegen könnten Meilensteine auch früher erfüllt sein als geplant.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Das führen wir in unseren Softwareprojekten durch; Quality Gates sind auch in der Industrie weit verbreitet. Sie sind keine akademische Erfindung, sondern der Praxis entnommen. Daher gibt es wenige Publikationen darüber, aber viele Firmen vertrauen darauf.

Es gibt natürlich verschiedene Ausprägungen von Quality Gate-Ansätzen.

In der Regel wird aber in der QG-Sitzung eine echte, harte Entscheidung getroffen. Das ist ein Sicherheitscheck, um ganz schlechte Qualität auszusieben.

Sie kann insbesondere auch bedeuten, das Projekt zu beenden. Das ist nicht immer auf ein Versagen der Projektmitarbeiter zurückzuführen. Stellen Sie sich vor, ein innovatives Projekt stellt in einem Quality Gate fest, dass sein angepeilter Markt inzwischen eingebrochen ist (z.B. wegen Konkurrenzprodukt oder Wechselkurs). Dann sollte man ein nicht mehr aussichtsreiches Projekt vielleicht lieber beenden – und niemand in der eigenen Firma ist „schuld“.

Ein Quality Gate ist wie ein Meilenstein (bedeutender Projektzustand), aber Quality Gates dienen zur Steuerung und Kontrolle *vieler* Projekte durch das Management. Immer an der gleichen Stelle im Projekt werden die gleichen, meist ziemlich formalen Kriterien angelegt, und nur wenn die erfüllt sind, kann weiterentwickelt werden. Meilensteine sind dagegen Projekt-spezifisch und damit nicht übertragbar.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The logo for Software Engineering (SE) features the word "SOFTWARE" in a small, black, sans-serif font at the top. Below it, "SE" is written in large, bold letters where the "S" is purple and the "E" is yellow, both with a glowing effect. At the bottom, the word "ENGINEERING" is written in a smaller, black, sans-serif font.

Häufige Planungsfehler

- **Unrealistische Pläne und Wünsche**
 - **Unrealistische Wunschtermine**
 - Schätzung u. Planung werden vom Management „überstimmt“/ignoriert
 - Schätzungen ohne Erfahrungsbasis
 - Basieren auf unklaren/unzureichenden Daten
- **Pläne nicht mit Betroffenen abgestimmt**
 - Ist das eingeplante Personal dann wirklich verfügbar?
 - Wer plant noch mit diesen Ressourcen?
 - Auf „Selbstverständlichkeiten“ verlassen
- **Kosten nicht (ausreichend) geplant**
 - Entwicklung nicht so weit vorhersehbar

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 329

Klingt wieder einmal banal.

Aber das sind die häufigen, teuren Fehler.

Achten Sie zumindest darauf, wo sie in Ihrem Umfeld gemacht werden (sie werden stndig gemacht), dann werden Sie bald die Bedeutung einschtzen knnen - und vielleicht Ihrerseits diese Fehler vermeiden.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Inhalt von Kapitel 7

7. Management von Technik und Projekt

- Versions- und Konfigurationsmanagement
- Aufgaben von Projektleitern
- Aufwandsschätzungen
- Risikomanagement in Softwareprojekten
- Agile Softwareentwicklung

Leibniz Universität Hannover SWT 2015/16 · 330

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**SOFTWARE
SE
ENGINEERING**

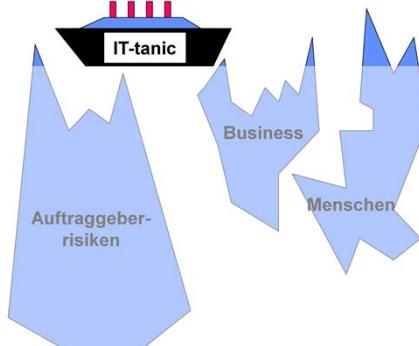
Wieso IT-Risiken managen?

Ziel: „Eisberge“ (Risiken)...
– kennen ([Wetterbericht fragen](#))
– vermeiden ([Umweg; Eisbrecher](#))
– Folgen verringern ([Aufprallwinkel](#))
– auf Folgen vorbereiten ([Rettungsboote](#))

– oder: „... *das riskieren wir!*“ (nichts tun)

Viele IT-Risiken sind verborgen!

Vorteile durch Risikomanagement
Proaktiv entscheiden:
– Überraschungen und Probleme rechtzeitig vermeiden
– Schlechte Folgen mindern
– Projektfortgang bestimmen ...
– ... statt von Problemen getrieben sein



Das Diagramm zeigt einen Eisberg im Wasser. Ein kleiner Teil des Berges ist über Wasser sichtbar und beschriftet mit 'IT-tanic'. Der größere, untergetauchte Teil ist als 'Auftraggeber-risiken' beschriftet. Rechts neben dem Eisberg sind zwei weitere Blöcke dargestellt: einer als 'Business' und einer als 'Menschen'.

Egerter, Heidi; Schneider, Kurt (2002): Risikomanagement für SW-Projekte. SQM-Kongress. SQS
Leibniz Universität Hannover

Kurt Schneider

SWT 2015/16 · 331

Ein Risiko ist umso schlimmer, je unbekannter es ist.

Ein Eisberg, der erkannt und vermessen ist, kann umfahren werden. Ein Risiko, das man kennt, kann man in den Griff bekommen. Die Folie zeigt beispielhaft verschiedene Dinge, die man tun kann, wenn sich ein Eisberg/Risiko zeigt. Nachher werden die Möglichkeiten noch einmal systematisch sortiert.

Man kann auch bewusst *nichts* tun. Solange das bewusst geschieht, ist es eine legitime Entscheidung. Manche Risiken muss man auf sich zukommen lassen. Gegen viele andere sollte man sich aber wappnen. Darauf kann Risikomanagement immer wieder hinweisen.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Begriffe

- **Risiko** =_{Def} *potenzielles Problem (verbunden mit einem Verlust), das eintreten kann, aber nicht sicher eintritt.*
 - Ein **Problem** ist damit kein Risiko mehr (es ist ja schon eingetreten).
 - Eine **Gefahr** ist ein noch nicht erkanntes (unbewusstes) Risiko.
- **Risikomanagement (RM)**
 - **systematische Vorgehensweise**, um
 - Projektrisiken (aller Art) frühzeitig zu erkennen und
 - während der Projektlaufzeit **fortlaufend** zu überwachen, um **mögliche Nachteile und Verluste** zu verringern.
- **Risk Exposure** (nach B. Boehm):
$$\text{Risk Exposure} = \text{Probability (Outcome)} * \text{Loss (Outcome)}$$



Barry Boehm

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 332

Diese Definitionen sind besonders prüfungsrelevant, da auf ihnen das ganze Gebiet des Risikomanagement und alle Aktivitäten dazu ruhen.

Viele formalere Verfahren beruhen auf Begriffen wie der Risk Exposure. Sie klingt einleuchtend, ist aber nicht immer einfach zu bestimmen. Während man den möglichen Schaden noch quantifizieren kann, gelingt das bei der Wahrscheinlichkeit erfahrungsgemäß nicht mehr so gut.

Daher ist es auch sinnvoll, anstelle der Wahrscheinlichkeit ein anderes Maß einzuführen. Bewährt haben sich: die Dringlichkeit des Handlungsbedarfs, die Unbekanntheit des Risikos (je unbekannter desto schlimmer!)

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Was kann man mit Risiken tun?

Die fünf Kategorien möglicher Maßnahmen am Beispiel

- Situationsbeispiel
 - Projektleiter in kleinem Softwarehaus
 - braucht Treiber-Zulieferung einer anderen Firma
 - um sein 3-Jahres-Projekt abzuschließen
- Zwei Risiken:
 - WENN der Treiber nicht rechtzeitig kommt, DANN wird das gesamte Projekt nicht fertig und die Bezahlung verzögert sich
 - WENN der Treiber nicht funktioniert DANN wird der Kunde nicht abnehmen und die Bezahlung verzögert sich

1. Verhindern
 - Lösung mit Standard-Treiber anstreben
2. Schaden verringern
 - Ähnlichen Treiber/Vorversion bereithalten
3. Wahrscheinlichkeit verkleinern
 - Zulieferer auf Bedeutung des Termins hinweisen/Vertrag
4. Vorbereiten auf Schaden
 - Kunden langsam vorbereiten, dass er evtl. vorerst ohne diese Funktion auskommen muss
5. Hinnehmen
 - „Wird schon klappen; Kunde merkt es auch nicht so schnell“

Allg.: 3 Speziell: 1 2 4 (5)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 333

An dem kleinen Beispiel soll gezeigt werden, welche Kategorien/Arten von Maßnahmen überhaupt in Frage kommen. Es sind nur fünf.

Davon sind zwei eher genereller Art (Schäden verringern und Wahrscheinlichkeit verkleinern). Sie greifen an den beiden Parametern an, die ein Risiko ausmachen: der Bedingung und der Folge.

Etwas spezifischer sind Maßnahmen 1 und 4: vollständiges Verhindern eines Risiko ist eine drastische Verringerung der Wahrscheinlichkeit. Sich auf den Schaden vorzubereiten ist ein Weg, ihn zu verkleinern.

Der Vollständigkeit halber wird wieder die Nicht-Maßnahme (5) erwähnt: Sie ist legitim, wenn der Aufwand für eine Maßnahme in keinem sinnvollen Verhältnis zum verhinderten Risiko steht.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Wieso muss Risikomanagement diszipliniert und einheitlich ablaufen?

- Häufiger Einwand:
„Intuitiv machen wir das schon!“
- Aber intuitives Vorgehen reicht nicht!
– versagt genau, wenn man es braucht:
unter Druck

Das Diagramm zeigt den Prozessfluss von oben nach unten. Es beginnt mit einer Kritikaltätsbewertung, die in normal oder reduziert unterteilt ist. Von 'normal' führt der Pfeil zu 'RM Prozeß aufsetzen', dann zu 'RM Einweisung', dann zu 'Risiken identifizieren'. Von 'reduziert' führt der Pfeil zu 'Risiken analysieren'. Von 'Risiken identifizieren' führt der Pfeil zu 'Risikostatus aktualisieren'. Von 'Risiken analysieren' führt der Pfeil zu 'Maßnahmen festlegen und Personen zuordnen', dann zu 'Maßnahmen umsetzen und Wirksamkeit überprüfen', und schließlich zurück zu 'Risikostatus aktualisieren'. Links neben dem Flussdiagramm steht 'Kritikaltätsbewertung' und 'Fragebögen bzw. Checklisten'. Rechts stehen 'RM nicht obligatorisch', 'Verwaltungsliste' und 'Portfolio'. Am oberen Rand steht 'Kritikaltätsbewertung' und 'RM nicht obligatorisch'.

Egerter, Heidi; Schneider, Kurt (2002): Risikomanagement für SW-Projekte. SQM-Kongress. SQS
Leibniz Universität Hannover

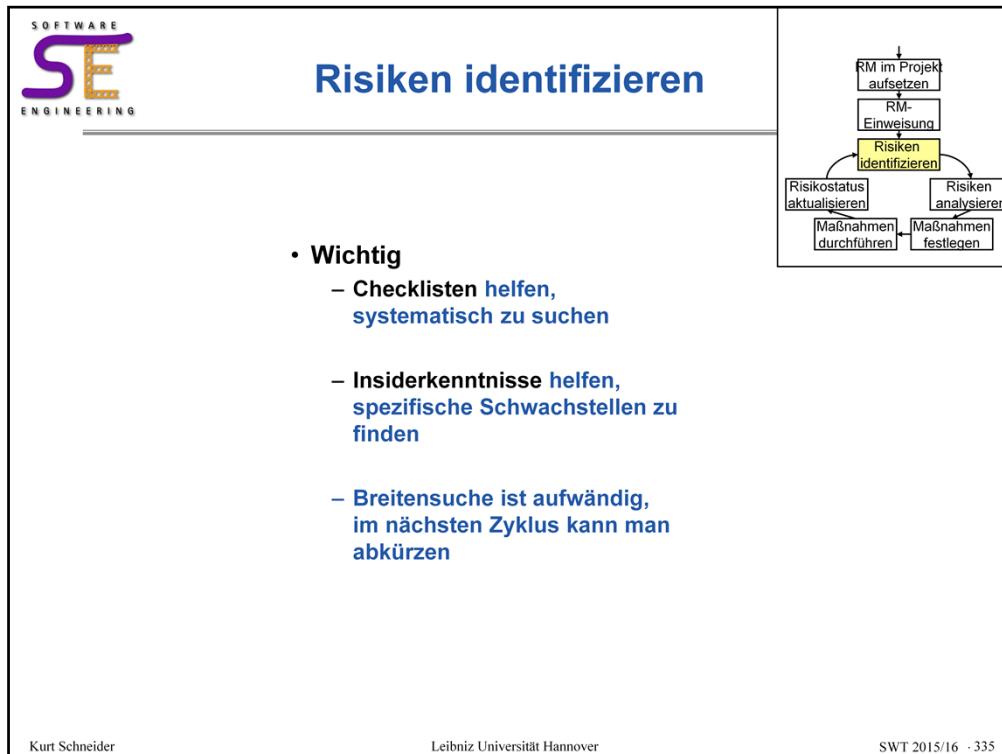
Kurt Schneider

SWT 2015/16 · 334

So sieht ein typischer RM-Prozess aus. Dieser Prozess läuft neben dem normalen Entwicklungsprozess her und ist idealerweise mit dem Entwicklungsprozess verbunden: so legt man beim „RM Prozeß aufsetzen“ fest, wann die Identifikations-, Analyse- und weiteren Sitzungen im Projektablauf stattfinden sollen. Am besten geschieht das in der Regelkommunikation oder angelehnt an ähnliche Aktivitäten.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Man kann sich eine Liste typischer Risiken aufstellen (oder sogar kaufen), aber am wichtigsten ist die eigene Überlegung: was sieht hier komisch aus, wer ist uns nicht so wohlgesonnen, welcher Termin scheint eng zu sein, wer könnte unzuverlässig sein? Das ergibt die wichtigsten Risiken.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Das Inhaltsverzeichnis der CMU-Risikoliste; auch als Checkliste verwendbar.
Diese Liste ist auf dem Internet verfügbar.

Sie ist umfangreich, aber natürlich nicht sehr spezifisch. Das ist beides eher schlecht: denn wer hat schon Lust, sich durch seitenweise möglicher, auf englisch eher vage beschriebener Risiken durchzukämpfen? Da wären einem die „top-10 Risiken“ der eigenen Organisation viel lieber. Risikomanagement kann durch Erfahrungen stark profitieren.

Und auch wenn man ein Risiko als relevant identifiziert hat, ist man damit noch nicht fertig: Man muss genau beschreiben, in wiefern sich dieses Risiko hier im Projekt manifestiert. Also: genaue Namen, exakte Beschreibung.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Risiko-Analyse und Maßnahmenplanung

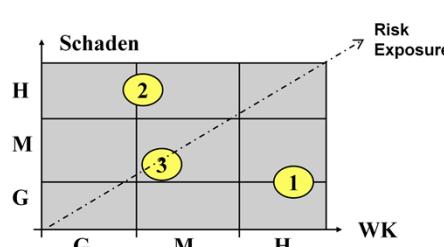
- Worin besteht das Risiko?
- Wie groß ist ein Risiko?

– **Risk Exposure =_{Def} Wahrscheinlichkeit * Schaden**

- Danach priorisiert man Risiken
- Allerdings nochmal Plausibilität prüfen, nicht nur Rechnen

– Für top-5 oder top-10 Risiken überlegt man Maßnahmen

```
graph TD; A[RM im Projekt aufsetzen] --> B[RM-Einweisung]; B --> C[Risiken identifizieren]; C --> D[Risikostatus aktualisieren]; D --> E[Maßnahmen durchführen]; E --> F[Maßnahmen festlegen]; F --> G[Risiken analysieren]; G --> C;
```



			Schaden			
			H	M	G	
			2	3	1	
			G M H			
			WK			

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 337

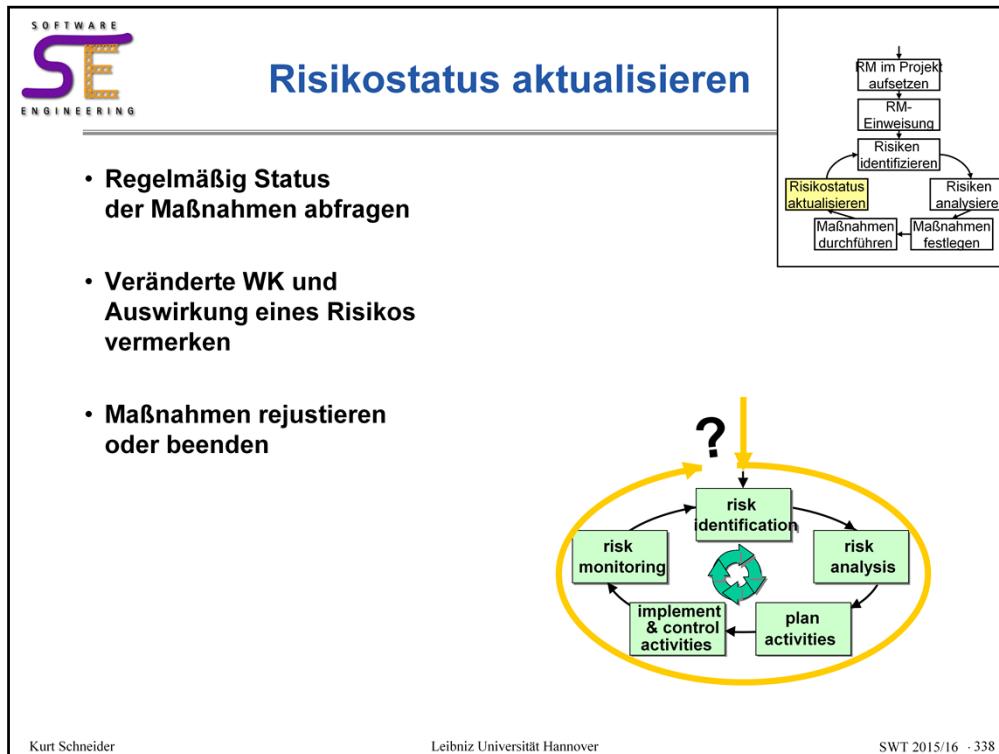
Die Risikoanalyse schließt alle Tätigkeiten ein, um das Risiko besser zu verstehen.

Dazu kann es nötig sein, eine Klärungsmaßnahme einzuplanen. Sie hat nur die Aufgabe, einen Aspekt des Risikos zu konkretisieren oder vielleicht auch herauszufinden, dass es doch kein Risiko darstellt.

Beispielsweise hat man herausgefunden, dass ein Vorstand sich abfällig über das Projekt geäußert hat. Dann könnte die Klärung darin liegen, dem nachzugehen und genauer zu beschreiben, ob und was dahinter steckt. Vielleicht war er auch nur schlecht gelaunt, dann ist das Risiko keines. Wie man aber so etwas herausfindet, erfordert einige Fantasie. Die Maßnahmenplanung ist nicht immer einfach.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Irgendwann muss man dann nachsehen, ob die Maßnahmen gegriffen haben. Der Status aller Risiken ist also zu aktualisieren.

Und dann sollte es von vorne beginnen.

Leider bricht RM an dieser Stelle oft ab: Man mag nicht noch einmal durch die dicken Checklisten gehen, die identifizierten Risiken sind scheinbar in Bearbeitung.

Das mag stimmen, aber man muss hier prüfen, ob nicht doch noch etwas ganz neues hinzugekommen ist oder ob sich die Risiken tatsächlich verschärft haben.

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

Risiken aufschreiben
Praktisches Beispiel für gutes Format

1. **WENN** der Kunde spät noch weitere Optionen verlangt,
DANN müssten wir den Entwurf noch deutlich modifizieren,
dadurch verzögert sich das Projektende und die Architektur verliert ihre Klarheit
(Wartbarkeit nimmt ab, geforderter Qualitätsaspekt)
2. **WENN** die biometrischen Sensoren nicht fein genug auflösen
DANN kann unsere Software die Gesichter nicht unterscheiden und die
Hauptfunktion der Zugangssteuerung funktioniert nicht;
das Projekt ist vorerst gescheitert
3. **WENN** die Nachbarabteilung nicht
motiviert ist, mitzuarbeiten
DANN werden wir den engen Zeitplan nicht
halten und als die Verursacher dastehen

Auswirkung

	G	M	H
H		2	
M		3	
G			1

Abhilfemaßnahmen

Zu 1: Design-Pattern *Strategy* einsetzen, um flexibler zu bleiben
Zu 2: Vorabtest-Resultate vom Hersteller einfordern,
bei ersten Anzeichen weitere Maßnahmen planen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 339

Das hier vorgeschlagene WENN-DANN-Format betont den Risikocharakter: Nur WENN die Bedingung eintritt, ergibt sich auch die Folge und daraus eine mehr oder weniger schlimme Konsequenz.

Dabei sollte im DANN-Teil tatsächlich auch die schlimme Konsequenz stehen, denn sonst ist es schwer, den Grad der Auswirkung abzuschätzen.

Diese beiden Parameter werden genutzt, um die Risiken relativ zueinander in einem zweidimensionalen Portfolio zu platzieren.

In den meisten Lehrbüchern finden Sie den Begriff der „Risk Exposure“, eines Erwartungswerts für den befürchteten Schaden. Er wird als möglicher Schaden mal Wahrscheinlichkeit errechnet. Das ist plausibel, aber die Ermittlung präziser Schadensaufkommen oder Wahrscheinlichkeiten ist nicht einfach. Dagegen ist ein Portfolio mit 3x3 Feldern (je hohe, mittlere, geringe Wahrscheinlichkeit und Grad der Auswirkung) in der Praxis hilfreicher.

Am schlimmsten sind die Risiken, die wahrscheinlich eintreten werden und hohe Auswirkungen haben (oben rechts). Die beiden Fälle 1 und 2 sind auf den ersten Blicke ähnlich schlimm, aber aus unterschiedlichen Gründen. Bei genauerem Hinsehen überwiegt aber der sehr große Schaden, der durch Risiko 1 hervorgerufen werden könnte.

Daher wären die angegebenen Abhilfemaßnahmen wohl angemessen, zumal der Assistent auch ohne das Problem der kranken Projektleiterin schon nützliche Aufgaben wahrnimmt (Dokumentation).

Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The logo for Software Engineering (SE) features the word "SOFTWARE" in small black capital letters above the letters "SE". The letter "S" is purple and stylized with a wavy line, while "E" is yellow with a zigzag pattern. Below "SE" is the word "ENGINEERING" in small black capital letters.

**Software
Engineering**

Tipps und Tricks

Zusammenfassung aus Erfahrung

Einstellung

- Offenheit und Kultur sind wichtig
- Risiken nicht Personen anlasten

Organisation

- Nicht zu viele Risiken verfolgen, die aber richtig
- Jedes Risiko bekommt einen Beauftragten
- Risikomanagement in die Regelkommunikation einfügen
- Regelmäßig kontrollieren, auch wenn nichts los ist
- Disziplin und Einheitlichkeit sind besonders wichtig!

Darstellung des Nutzens

- Bewältigte Risiken herausstellen
- Worst Case nennen: zeigt, welcher Schaden verhindert wurde

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 340