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***

Softwaretechnik

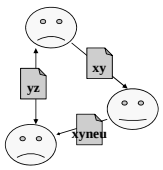
Inhalt von Kapitel 7

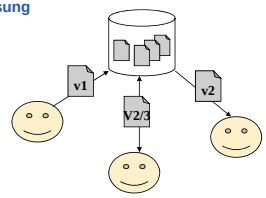
7. Management ***von Technik und Projekt***

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

Code organisieren: Versionen und Konfigurationen

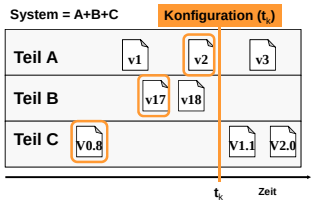
- **Ziele**
  - Dateien, die bei der Entwicklung anfallen, für Team koordinieren
  - Für alle zugreifbar
  - Kein versehentliches Löschen
  - Kein gegenseitiges Überschreiben
  - Klarheit über jeweils gültige Fassung





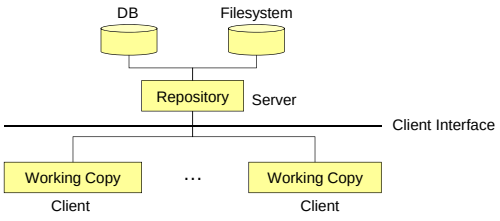
Code organisieren: Versionen und Konfigurationen

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



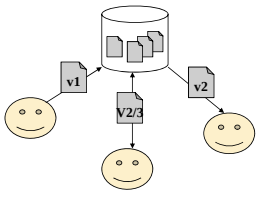
Grundlagen

- **Aufbau einer Versionsverwaltung**



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 Kollission

Pessimistisch
Sicherheitshalber darf nur einer ändern
Keine Kollission
Behindert aber Parallelarbeit

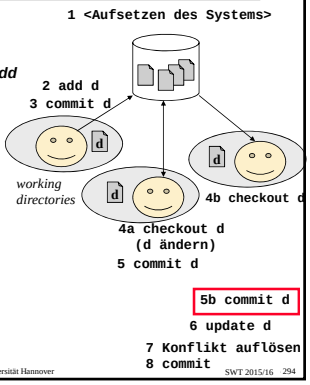
Wichtige Begriffe 2/2

- **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

Versionsmanagement

Prinzip („optimistisch“)

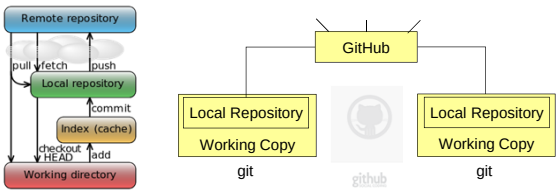
- **Grundoperationen**
  1. Aufsetzen: Pfade, Rechte etc.
  2. Datei beim System anmelden: ***add***
  3. Datei eintragen: ***commit***
  4. Datei herausholen: ***checkout***
  5. Datei zurückgeben: ***commit***
  6. Datei nochmal holen: ***update***
- **Weiterführendes**
  - Branches (Verzweigungen)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



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

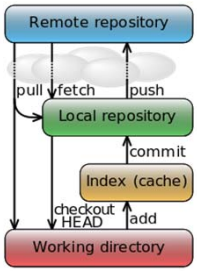


Von: Wikipedia („Git“), 4.11.12

Daten in Git

Dateneinheiten:

- **Blob:** **Nur** Dateiinhalt
- **Tree object:** Directory (Namen, Links)
- **Commit object:** Tree mit History
- **Tag object:** Link zu anderem Objekt (oft: Tree) mit Metadata



Zustände

Commits (Menge von commited items)

↑

Staged changes

↑

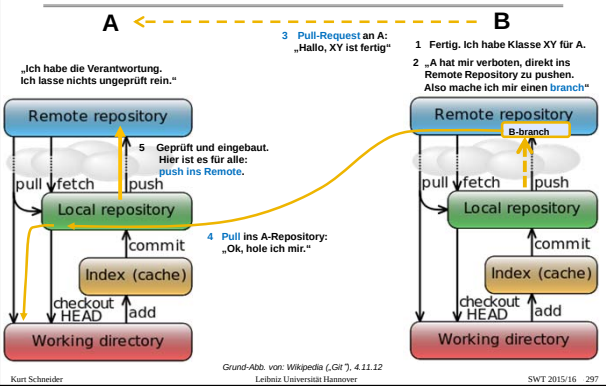
Unstaged changes

Hinweis: pull = fetch+merge

Von: Wikipedia („Git“), 4.11.12

Pull Requests

Szenario: B entwickelt für Besitzerin A



Grund-Abb. von: Wikipedia („Git“), 4.11.12

Softwaretechnik

Inhalt von Kapitel 7

7. Management ***von Technik und Projekt***

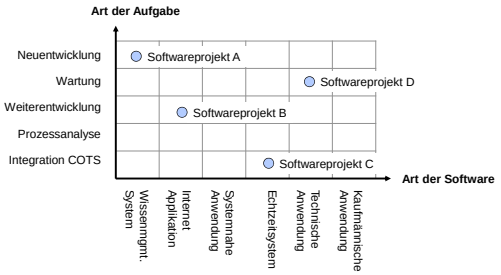
- Versions- und Konfigurationsmanagement
- Aufgaben von Projektleitern
- Aufwandsschätzungen
- Risikomanagement in Softwareprojekten
- Agile Softwareentwicklung

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

Arten von Software-Projekten

nicht alle Projekte sind gleich (... aufwändig etc.)!



Angelehnt an: Houdek, Frank (2002): Vorlesung „Management von Software-Projekten", Univ. Ulm

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

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

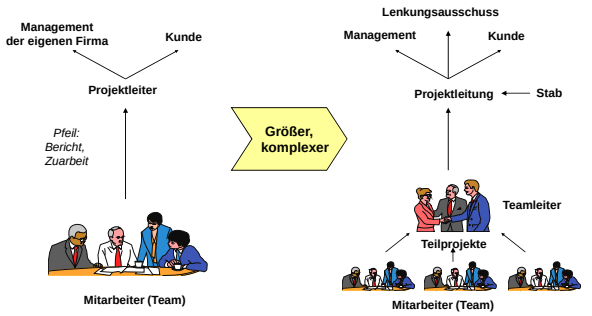
Die Weiterbildung der Mitarbeiter ist nicht zielgerichtet
Auswahl und Einsatz der Werkzeuge bzw. Methoden ist unzureichend vorbereitet

Risikomanagement wird nicht betrieben
Eine Abnahme der Phasenergebnisse erfolgt nicht

Es wird nicht systematisch bzw. unzureichend getestet
Anforderungen und Qualitätsziele werden nicht festgelegt

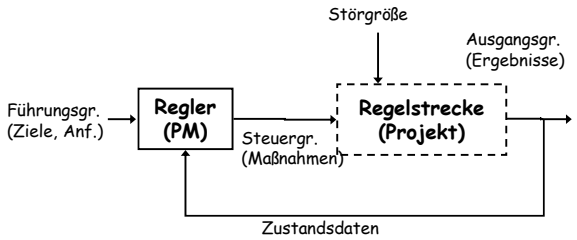
Quelle: Interne Untersuchung bei Bosch

Organisation von Projekten

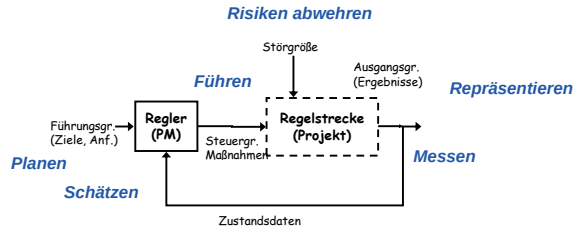


SW-Projekt als Regelstrecke

Projektmanagement als Regler



Aufgaben des Projektleiters



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 ingenieursmäßiges Vorgehen:
  - „Schätze nicht blind, sondern messe und vergleiche mit Deiner Schätzung. Auf dieser Basis schätze das nächste Mal besser.“

Welche Aspekte sind zu planen?

- Was ist zu tun? → *Lastenheft, Pflichtenheft*
- Struktur der Aufgabe → *Projektstrukturplan (PSP)*
- Organisation des Projekts → Leitung, Zuständigkeitstabelle
- Planung der Arbeitspakete → Arbeitspaketbeschreibung
- Reihenfolgen → *Netzplan*, Ablaufplan
- Kapazitäten → Kapazitätenplan
- Kosten → Kostenplan
- Verwaltung → Berichtsplan, *Projektakte*
- Risiken → *Risikolisten*

Legende
Wichtig, hier neu
An anderer Stelle besprochen

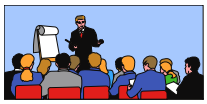
Projekt-definition | Projekt-struktur | Organi-sation | Arbeits-pakete | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken

Planungsprozess

Projektdefinition

Projekt-definition | Projekt-struktur | Organi-sation | Arbeits-pakete | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken

- Oft schon vor „Projektbeginn“ bzw. im „Vorprojekt“
- Beim Projektstart: Kickoff
  - Offizielle Vorstellung von Aufgabe und Leiter
  - Startschuss



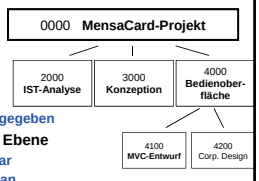
- Ergebnis der Projektdefinition:
  - Lastenheft und Pflichtenheft/Spezifikation
  - Sinn: Bezugspunkt für Entwicklung, Prüfung, Planung
- Wer definiert das Projekt?
  - Auftraggeber? Lenkungsausschuss? Mgmt. des Auftragnehmers? Projektleiter?

Projekt-Strukturplan: Was ist zu tun?

Work breakdown structure

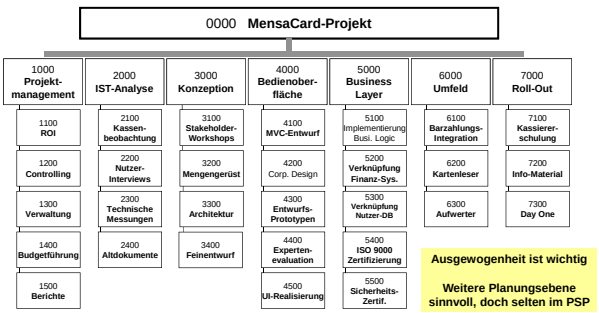
Projekt-definition | Projekt-struktur | Organi-sation | Arbeits-pakete | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken

- Hierarchische Aufgliederung
  - So fein wie nötig (meist drei Ebenen)
    - Projekt
    - Teilprojekt
    - Arbeitspaket
    - (evtl. noch: Vorgang/Aktivität)
  - Durch hierarchische Nummerierung wiedergegeben
- Schätzung und weitere Planung auf feiner Ebene
  - Kleine Teile leichter vergleich- und schätzbar
  - Wieder-zusammenrechnen ergibt Gesamtplan
- PSP ist nach ***Aufgaben*** strukturiert, diese oft analog ***Produkt***
- Die Anordnung impliziert ***nicht …***
  - Abhängigkeiten
  - Reihenfolge der Bearbeitung
  - wie oft ein Arbeitspaket angepackt wird



Projektstrukturplan

Beispiel



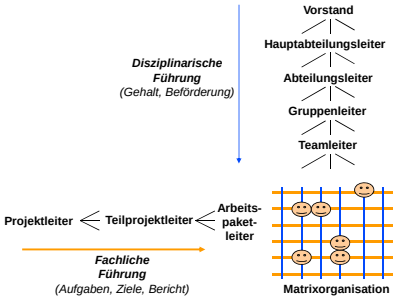
Ausgewogenheit ist wichtig

Weitere Planungsebene sinnvoll, doch selten im PSP

Projektorganisation: wie gehen wir´s an?

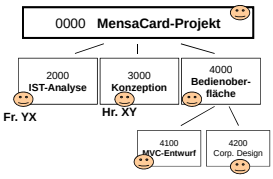
Verantwortlichkeiten regeln

Projekt-definition | Projekt-struktur | **Organi-sation** | Arbeits-pakete | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken



Zuständigkeiten: wer tut was?

Projekt-definition | Projekt-struktur | **Organi-sation** | Arbeits-pakete | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken



- **AKV** (Ausschnitt) **„Rolle Projektleiter“**
  - **Aufgaben**
    - Plant Projekt
    - Kontrolliert Durchführung
    - Berichtet Lenkungsausschuss
  - **Kompetenzen**
    - Kann MA aus Linie einfordern
    - Kann MA Aufgaben zuweisen
  - **Verantwortung**
    - Termin- u. Budgetkontrolle
    - Risikoüberwachung

Arbeitspakete: Was ist im einzelnen zu tun?

Projekt-definition | Projekt-struktur | Organi-sation | **Arbeits-pakete** | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken

- **Arbeitspaket (AP)**
  - Meist tiefste PSP-Ebene
  - Bezugspunkt für Schätzung
  - Unterteilung in „Vorgänge“ möglich
- **Zweck**
  - Klare, inhaltliche Aufgabe
  - Verantwortlicher AP-Leiter
  - Identifizierbarer Baustein
- **Form: unterschiedlich**
  - Meist Formular; **Beispiel!** →
  - Offizieller Planungsbestandteil

AP 3100.4
Stakeholder-Workshops

K.Schneid, Jan-Juli 2016 (Meilenstein MS 5)

Deliverables
- 4 Protokolle/Transkript
- Zusammenfassung

Nutzen
- Abgestimmte Reqs.
- Stakeholder ernst gen

Aufgaben
- Terminfindung (Doodle)
- Vorbereitung (Agenda)
- Moderation
- Transkribieren
- Zusammenfassung der Reqs. Im Std-Format

Kosten: 4 Pwochen
Beteiligt: X, Y, Z

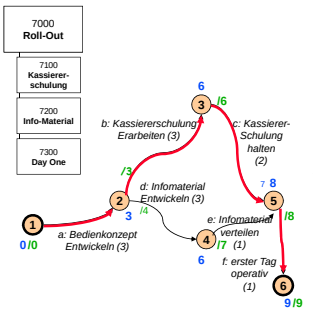
- **AP-Beschreibung enthält**
  - Titel, Verantwortlicher
  - Bearbeitungszeitraum
  - Textuelle Aufgabenbeschreibung
  - Ergebnisse/Deliverables
  - Kundennutzen
  - Interne Pläne (Meilensteine)
  - Besonderheiten

Nutzen von APs
- Erfolgsmessung
- Geht in Meilensteine ein
- „Aufwand auf APs gebucht“

Ablauf- und Terminplanung

Mit Abhängigkeiten planen

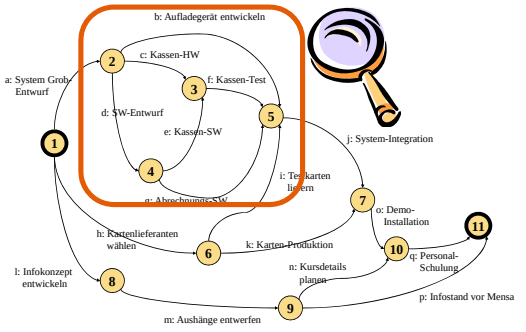
Projekt-definition | Projekt-struktur | Organi-sation | Arbeits-pakete | **Abhängigk. Zeiten** | Kapazität Kosten | Berichts-wesen | Risiken



1. **Vorwärts rechnen**
   - Start + Dauer
   - *spätester* aller eingehenden Pfeile
2. **Frühesten Endtermin ermitteln: 9**
3. **Rückwärts rechnen** (z.B. /7)
   - Spätester Beginn, um Termin zu halten?
4. **Puffer = zeitlicher Spielraum**
   - Es gibt verschiedene Pufferarten
   - Wichtig: Spätester - frühester Start
5. **„kritischer Pfad“: Aktivitäten mit Puffer=0** (z.B. 6/6)
   - Verspätung führt zu Projektverzug

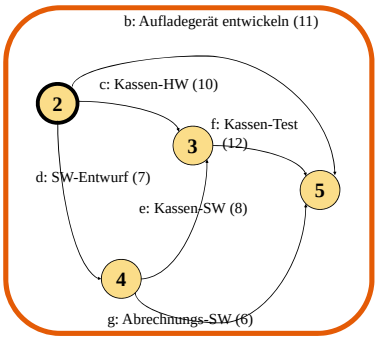
Netzpläne des MensaCard-Systems

Beispiel



Puffer und Kritischen Pfad ermitteln

Noch ein Beispiel: Ausschnitt aus MensaCard



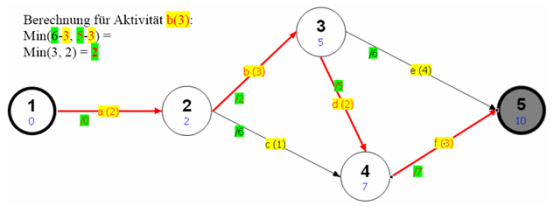
1. **Vorwärts rechnen**
   - Start + Dauer
   - Je *späteste* Aktivität
2. **Frühester Endtermin**
   - Hier Nr.5: t=27
3. **Rückwärts rechnen**
   - Spätester Beginn jeder Aktiv. für frühest. Endtermin
4. **Puffer** (oft in Tabelle)
   - Spätester-frühester Start

Aktiv.	FrStart	SpätStart	Puffer
c	0	5	5
...			
e	7	7	kritisch:0
g	7	21	14

5. **„kritischer Pfad“**
   - Hat Puffer=0
   - Jede Verspätung verzögert ganzes Projekt

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



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

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

MS 3.2
Meilenstein-Symbol in Plänen; dazu Beschreibung nötig

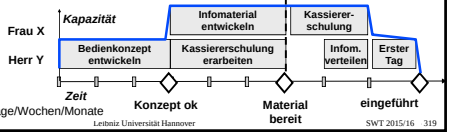
- Zweck
  - Klares, inhaltlich definiertes (komplexes) Kriterium für Projektfortschritt
  - Bewusst und spezifisch gesetzt, um Klarheit zu gewinnen
    - Innerhalb Projekt: **interner Meilenstein,** für interne Planung
    - Nach außen/zum Kunden: **externer Meilenstein**
  - An geplanten Meilenstein-Zeitpunkten 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?

Ablaufplan

Projekt-definition | Projekt-struktur | Organi-sation | Arbeits-pakete | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken

- Abfolgen festlegen
  - im Rahmen der Abhängigkeiten (Netzplan)
- Kapazitäten bestimmen möglichen Parallelitätsgrad
  - Bearbeiter und Ressourcen
- Konkrete Zeiten und Bearbeiter zuordnen

Kosten = PersKostenX*5+PersKostenY*9



Berichts- und Dokumentationsplan Risikomanagement

Projekt-definition | Projekt-struktur | Organi-sation | Arbeits-pakete | Abhängigk. Zeiten | Kapazität Kosten | Berichts-wesen | Risiken

Welche Dokumente?
Für wen?
Wozu?
Wie erzeugt?
Wie geprüft?

Risikomanagement
Wieso betreiben?
Was tun?

Softwaretechnik

Inhalt von Kapitel 7

7. Management *von Technik und Projekt*

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

Aufwandsschätzung mit COCOMO

von Barry Boehm auf Basis von 63 TRW-Projekten

- 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

- *(„parametriertes" Verfahren)*

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.

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

5 Kategorien von Op.

Anzahlen Op. schätzen

Komplexität d. Projekts ermitteln

Gewichtungs-vektor an-wenden

***Summieren, Adaptieren*:**

„Function Points“

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

** 14 Korrekturfaktoren = FP*

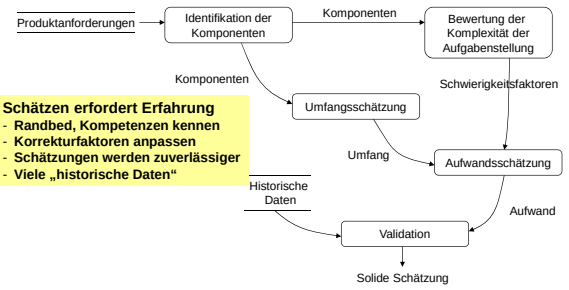
Prinzip: In Std-Komponenten zerlegen; Analogieschätzung auf diesen

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

Schätzen von Dauer und Aufwand

Datenfluss bei der Schätzung



Schätzen erfordert Erfahrung
- **Randbed, Kompetenzen kennen**
- **Korrekturfaktoren anpassen**
- **Schätzungen werden zuverlässiger**
- **Viele „historische Daten“**

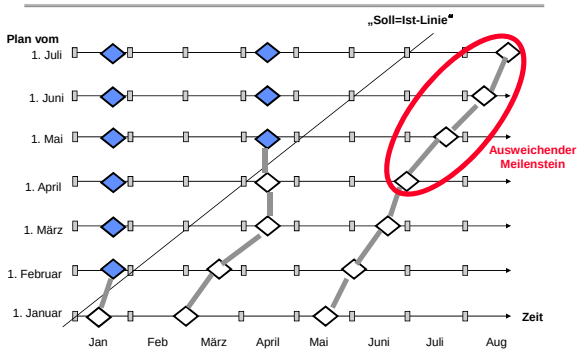
Siehe auch: Houdek, Frank (2002): Vorlesung „Management von Software-Projekten“, Univ. Ulm

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

Messen

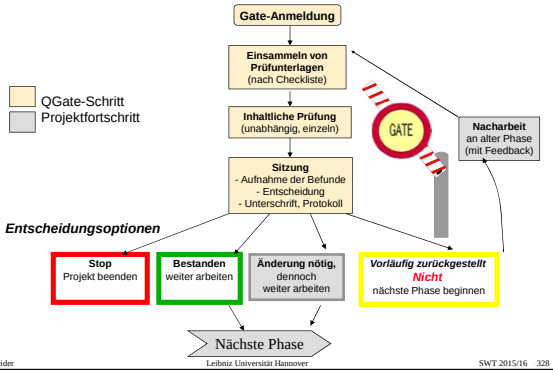
Controlling und Meilenstein-Trendanalyse



Trend der Meilensteine erkennbar: rechtzeitig erfüllt oder ausweichend?

Fortschrittsmessung mit Quality Gates

Multiprojektmanagement und Qualität



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

Softwaretechnik

Inhalt von Kapitel 7

7. Management *von Technik und Projekt*

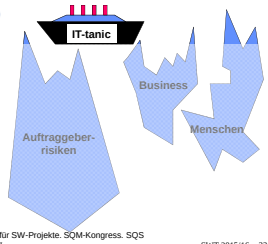
- Versions- und Konfigurationsmanagement
- Aufgaben von Projektleitern
- Aufwandsschätzungen
- Risikomanagement in Softwareprojekten
- Agile Softwareentwicklung

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



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

Begriffe

Barry Boehm

- **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):
  Risk Exposure $=_{Def}$ Probability (Outcome) * Loss (Outcome)

Was kann man mit Risiken tun?

Die fünf Kategorien möglicher Maßnahmen am Beispiel

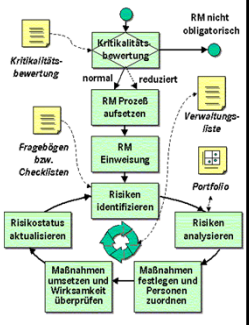
- **Situationsbeispiel**
  - Projektleiter in kleinem Softwarehaus
  - braucht Treiber-Zulieferung einer anderen Firma
  - um sein 3-Jahres-Projekt abzuschließen
  - Zwei Risiken:
    - **WENN** der Treiber nicht rechtzeitig kommt, **DANN** wird das gesamte Projekt nicht fertig und die Bezahlung verzögert 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		2		
WENN	X	DANN	Y		(5)
Speziell:	1		4		

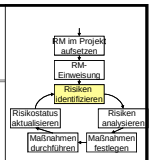
Wieso muss Risikomanagement *diszipliniert und einheitlich* ablaufen?

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



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

Risiken identifizieren



- **Wichtig**
  - **Checklisten helfen, systematisch zu suchen**
  - **Insiderkenntnisse helfen, spezifische Schwachstellen zu finden**
  - **Breitensuche ist aufwändig, im nächsten Zyklus kann man abkürzen**

CMU

Aus: CMU/SEI-93-TR-6

- **Wo lauern Risiken?**
- **Im Netz liegen Checklisten**
- **Individuelle Erfahrung bringt mehr**

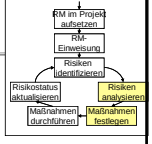
A. Product Engineering
1. Requirements
   a. Stability
   b. Completeness
   c. Clarity
   d. Validity
   e. Feasibility
   f. Precedent
   g. Scale
2. Design
   a. Functionality
   b. Difficulty
   c. Interfaces
   d. Performance
   e. Testability
   f. Hardware Constraints
   g. Non-Developmental Software
3. Code and Unit Test
   a. Feasibility
   b. Testing
   c. Coding/Implementation
4. Integration and Test
   a. Environment
   b. Product
   c. System
5. Engineering Specialties
   a. Maintainability
   b. Reliability
   c. Safety
   d. Security
   e. Human Factors
   f. Specifications

B. Development Environment
1. Development Process
   a. Formality
   b. Suitability
   c. Process Control
   d. Familiarity
   e. Product Control
2. Development System
   a. Capacity
   b. Suitability
   c. Usability
   d. Familiarity
   e. Reliability
   f. System Support
   g. Deliverability
3. Management Process
   a. Planning
   b. Project Organization
   c. Management Experience
   d. Program Interfaces
4. Management Methods
   a. Monitoring
   b. Personnel Management
   c. Quality Assurance
   d. Configuration Management
5. Work Environment
   a. Quality Attitude
   b. Cooperation
   c. Communication
   d. Morale

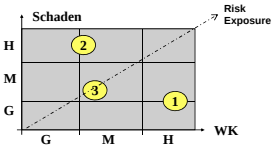
C. Program Constraints
1. Resources
   a. Schedule
   b. Staff
   c. Budget
   d. Facilities
2. Contract
   a. Type of Contract
   b. Restrictions
   c. Dependencies
3. Program Interfaces
   a. Customer
   b. Associate Contractors
   c. Subcontractors
   d. Prime Contractor
   e. Corporate Management
   f. Vendors
   g. Politics

Figure A-1 Taxonomy of Software Development Risks

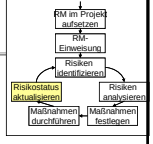
Risiko-Analyse und Maßnahmenplanung



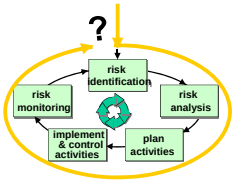
- **Worin besteht das Risiko?**
- **Wie groß ist ein Risiko?**
  - **Risk Exposure =$_{Def}$ Wahrscheinlichkeit * Schaden**
    - **Danach priorisiert man Risiken**
    - **Allerdings nochmal Plausibilität prüfen, nicht nur Rechnen**
  - **Für top-5 oder top-10 Risiken überlegt man Maßnahmen**



Risikostatus aktualisieren



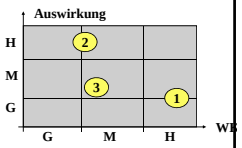
- **Regelmäßig Status der Maßnahmen abfragen**
- **Veränderte WK und Auswirkung eines Risikos vermerken**
- **Maßnahmen rejustieren oder beenden**



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



Abhilfemaßnahmen

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

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**