Softwaretechnik

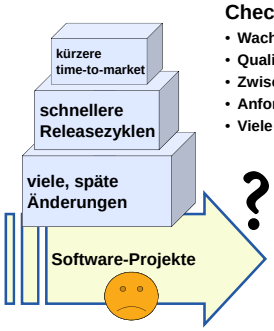
Inhalt von Kapitel 7

7. Management *von Technik und Projekt*

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

Softwareprojekte unter Druck

Motivation



Checkliste: Agil erwägen?

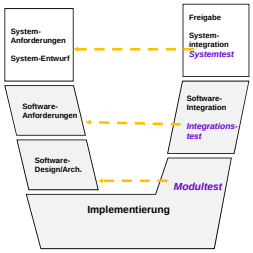
- Wachsender Zeitdruck
- Qualität ist wichtig
- Zwischenlieferungen erwartet
- Anforderungen vage oder unklar
- Viele späte Änderungen

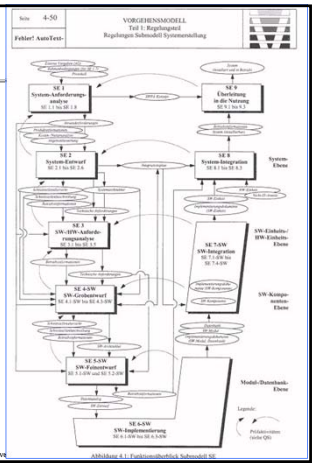
?

Qualität und Wartbarkeit

Systematische Entwicklung

Oft ideal, manchmal nicht angemessen





Traditionelle Softwareentwicklung

Stärken +

- Systematisch, nachvollziehbar
- Gut dokumentiert
- Saubere Spezifikation
- UML-Modelle
- Projektziele früh klar
- Detaillierte Planung, Vorgaben, Abläufe

Probleme -

- Relativ langsam
- Viel Aufwand
- Dokumente veralten oft, bevor man sie braucht
- Anforderungen oft missverstanden
- Änderungen aufwändig

Manifesto for Agile Software Development

www.agilemanifesto.org

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions *over* processes and tools
Working software *over* comprehensive documentation
Customer collaboration *over* contract negotiation
Responding to change *over* following a plan

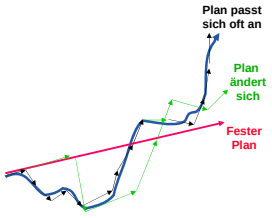
That is, while there is value in the items on the right, we value the items on the left more. "

Kent Beck, Alistair Cockburn
Martin Fowler, Jim Highsmith, Ron Jeffries
Ken Schwaber und 11 andere

Zusammenfassung

Angemessener Planungshorizont

- Grober Langfristplan; nur auf kurze Frist genau planen
- Defer Commitment: ständig anpassen
- Lernen und Verbessern



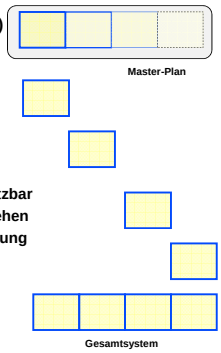
- Gleichmäßiger Durchfluss
- Und geringe Aufgabenzahl
- Führt zu gutem Durchsatz, großer Geschwindigkeit

Agile Prinzipien
Auswahl

- **Einfachheit ist wichtigstes Konstruktionsprinzip**
  - Möglichst wenig unnötige Arbeit, nicht auf Vorrat arbeiten
  - → ***Inkrementelle Entwicklung, kurze Releases, wenig Dokum.***
- **Höchste Priorität: Kundenzufriedenheit**
  - durch frühe und häufige Auslieferung laufender Software
  - → ***Enge Kundeneinbindung, Planungssitzung***
- **Auch späte Änderungen werden willkommen geheißen**
  - Änderungen bedeuten Verbesserungen für den Kunden
  - → ***Ständig testen, integrieren, vereinfachen; Pair Programming***

Inkrementelles Entwicklungsmodell
Prinzip Einfachheit

- **Entwicklung in Bausteinen (Inkrementen)**
  - **Grobe Vision (Master-Plan)**
  - **Spätere Inkremente nicht detailliert**
  - **Jeweils ein Inkrement genauer**
  - **Kurze Inkremente, zusammengesetzt**
- **Versetzte Entwicklungsstränge**
  - **Jedes Inkrement für sich vollwertig einsetzbar**
  - **Von vornherein als Ausbaustufen vorgesehen**
  - **Relativ unabhängig während der Entwicklung**
- **Time Boxing: immer im gleichen Zeittakt**

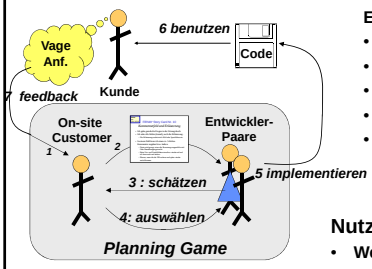


Kurze Release-Zyklen
Prinzip Einfachheit

- **Ziel: Häufiges Feedback durch den Kunden**
  - Operative Nutzung der Inkremente (nicht nur Prototypen!)
  - Aufteilung in Teillieferungen scheint schwierig – geht aber meist
- **Begriffsklärung**
  - „*Release*“ = „Inkrement“ = Auslieferung zur operativen Nutzung
  - Dazwischen *Iterationen*: auch lauffähig, gehen nicht an Kunden
- **„Saubere Spezifikation für alles ist besser“ – stimmt das?**
  - ist aber oft nicht zu erreichen
  - Kostet so viel Aufwand, dass keine Zeit für Umsetzung bleibt
  - Veraltet, bevor Projekt zu Ende ist
- **Selbst bei Projekt-Abbruch ist schon Nützliches entstanden**

… wenn man das Wichtigste zuerst macht!

Planungssitzung
Prinzip Kundeneinbindung



Embracing Change
- Änderungen als Chance
- Weniger Pläne und Prozesse
- Weniger Dokumente
- Direkte Kommunikation
- Direkt zur Software

Nutzen
- Weniger Overhead
- Schneller, einfacher, flexibler
- Kunde oft zufriedener
- Oft der *einzige* mögliche Weg

350

Informationen zur Planungssitzung
Prinzip Kundeneinbindung

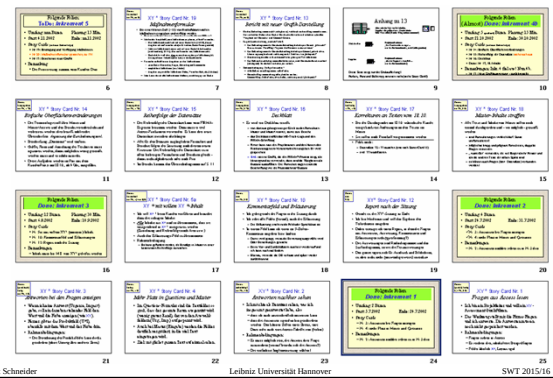
- **Anforderungen werden auf Story Cards gesammelt**
  - vom Kunden umgangssprachlich geschrieben
  - Informell, auf A5 od. A4-Karten, evtl. mit kleinen Skizzen
- **Entwickler schätzen Aufwand für jede Story Card**
  - abstraktes Maß (z.B. 1-5 Punkte)
  - Schätzung durch Erfahrung
  - Kunden, Entwickler interagieren dabei
  - Neue Erkenntnisse auf Story Cards notieren
- **Kunde wählt Story Cards für nächste Release(s) aus**
  - (Meistens, möglichst!) nach Nutzen priorisiert
  - Bisherige Produktivität des Entwicklungsteams ergibt Limit
- **Story Cards sind Grundlage für Akzeptanz-Tests**
- **Es gibt keine gewohnte, „ordentliche Spezifikation“**

Geld abheben
7.6.13, D.Autor

Anforderung
1. *Authentifizieren*
2. *Betrag eingeben*
3. *Prüfen*
4. *Auszahlen*

Aufwand,
Datum der Erledigung
geschätzt: 6 Pkt./ 1 Wo

Story Cards mehrerer Iterationen
Prinzip Kundeneinbindung



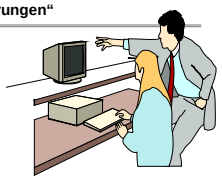
On-site Customer / Kunde vor Ort

Prinzip Kundeneinbindung

- **Kunde ist über die gesamte Projekt-Laufzeit vor Ort**
  - zumindest leicht und jederzeit erreichbar
- **Schreibt Story Cards (Entwickler dürfen helfen)**
  - Ständige Weiterentwicklung der Anforderungen
- **Beantwortet Fragen der Entwickler, auch unter Ungewissheit**
  - Kein Zeitverlust bei Entscheidungen
  - Keine Fehlentscheidungen durch Entwickler
- **Voraussetzung: Kompetenz und Entscheidungsgewalt**
  - Kunden müssen nach Nutzen sortieren (können und wollen)

Pair Programming

Prinzip „offen für Änderungen“



- **Metapher: Fahrer und Beifahrer**
- **Jedes Paar arbeitet an einer Story Card**
  - oder Task Card (Teile einer Story Card)
  - häufiges Abwechseln der Rollen
  - möglichst tägliches Tauschen der Paare
- **Ständiges Review**
  - Vermeidet vor allem Flüchtigkeitsfehler
  - Eine(r) schreibt, zweite(r) denkt und kontrolliert
  - Vier-Augen-Prinzip
- **Ständiges Lernen voneinander**
  - Hebt Entwickler kontinuierlich auf ein einheitliches Niveau
- **Ist effektiv und effizient**
  - Belegen einige Studien; andere belegen das Gegenteil
  - es gibt verschiedene Erfahrungen

Kontinuierliche Integration

Prinzip „offen für Änderungen“

- **Integration mehrfach am Tag, mit optimistischer „Sperre“**
  - Jeder hat immer aktuellen Code
  - Doppelte Arbeit wird vermieden
- **Ist Voraussetzung für gemeinsame Code-Verantwortung**
- **Jederzeit lauffähige, getestete Version**
  - Dadurch stets Feedback vom Kunden
  - Basis für Entscheidungen
  - Rückfallposition
  - Refactorings werden kleiner

Prinzip von TestFirst: ein Dialog

Prinzip „offen für Änderungen“

Aufgabe: Java-Methode len(int) gibt zurück, wie viele Stellen eine int-Zahl hat.

Test beginnt
„len(5) soll 1 sein!“
`assertEquals(len(5),1);`

JUnit
COMPILER-FEHLER!
Was soll „len“ bedeuten?

Programm: Das ist einfach:
public int len (int zahl) { return 1; }

JUnit: ok. Testfall erfüllt.

Test: Na, warte!
„len(321) soll 3 sein!“
`assertEquals(len(321),3)`

JUnit: Fehler! 1 statt 3

Programm: auch nicht so schwer …
if zahl<10 then return 1 else return 3

JUnit: ok.

Test: Ich glaub es nicht!
„len(12345678) soll 8 sein!“
`assertEquals(len(12345678),8);`

JUnit: Fehler! 3 statt 8

Programm: … ok, ich sehe das Muster:
eine Schleife: for (i=…

SCRUM

auf einen Blick

Andere

Product Owner

SCRUM Master

SCRUM team 7+/-2

SCRUM Room

Increment

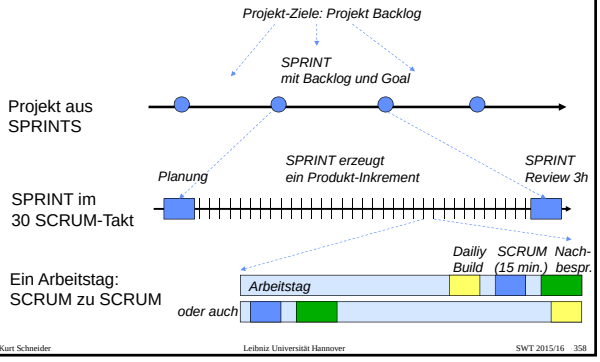
Product Backlog (priorisiert)

SPRINT Backlog

SPRINT: 30 days

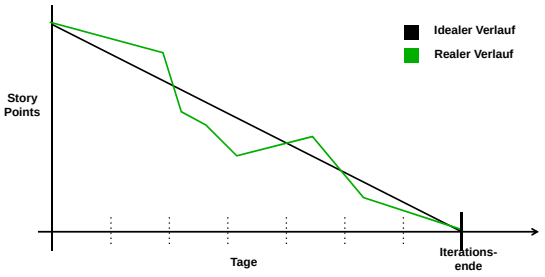
Daily SCRUM 15 min.

Detaillierungsebenen in SCRUM



Fortschrittsmessung

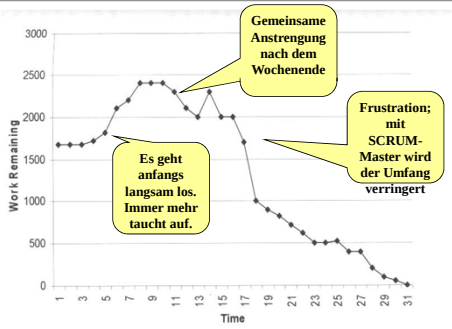
Burndown Charts



Olga Liskin: Labor "Agile Software Entwicklung" 359

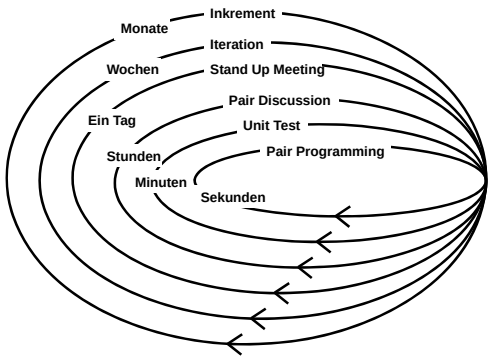
Sprint-Fortschritt schätzen

ein typischer Verlauf für neues SCRUM-Team



Nach Schwaber, Ken; Beedle, Mike (2002): Agile Software Development with Scrum. Prentice Hall.

Was agile Methoden stabil laufen lässt



Basiert auf Folien von Stefan Roock, it-wps, Hamburg

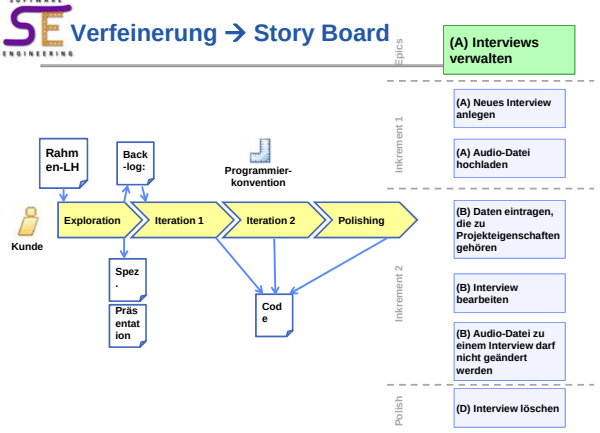
Ausblick: Software-Projekt (SWP)

- **Idee: Software entwickeln im Team**
  - Mit Anforderungserhebung und Kunden
  - Mit Entwurf, Implementierung und Test
  - Mit Besprechungen, Reibereien und Lösungen
  - Mit wirklich einsetzbarer Ziel-Software
  - Mit echtem Feedback
- **Früher: Strenges V-Modell; heute: Mischform** Spez. + Stories
- **Heute: Extra angepasstes Vorgehensmodell, wie in Praxis**
  - Klassische Spezifikation
  - Auflösung in Story Cards für Priorisierung und Änderung
  - SCRUM-Meetings pro Woche

Granularität, von Use Case ausgehend

Interviews verwalten	
Use Case Nr.	02
...	...
Auslöser	Interviewübersicht wird geöffnet.
Beschreibung	1. System zeigt Interviewübersicht. 2. Forscher erstellt ein neues Interview. 3. System öffnet Interview-Bearbeiten-Dialog. 4. Forscher lädt eine Audio-Datei hoch. 5. Forscher klickt auf Speichern. 6. System speichert die Änderungen und schließt den Dialog. 7. Forscher klickt auf den Bearbeiten-Button von einem Interview. 8. System öffnet Interview-Bearbeiten-Dialog. 9. Forscher ändert den Wert einer Interview-Eigenschaft. 10. Forscher klickt auf Speichern. 11. System speichert die Änderungen und schließt den Dialog. 12. Forscher löscht ein Interview.
Erweiterung	5a. WENN der Forscher auf Abbrechen klickt, DANN werden die Änderungen nicht gespeichert. 10a. WENN der Forscher auf Abbrechen klickt, DANN werden die Änderungen nicht gespeichert.

Verfeinerung → Story Board

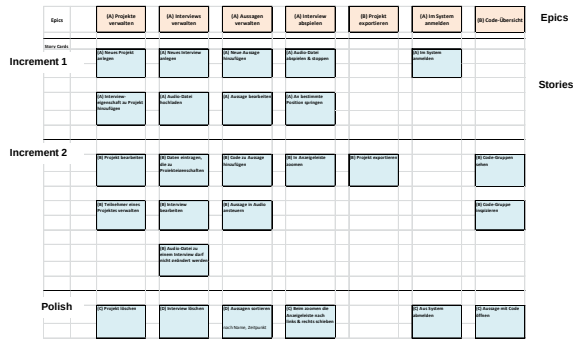


Abbildung

Interviews verwalten	
Use Case Nr.	02
...	...
Auslöser	Interviewübersicht wird geöffnet.
Beschreibung	1. System zeigt Interviewübersicht. 2. Forscher erstellt ein neues Interview. 3. System öffnet Interview-Bearbeiten-Dialog. 4. Forscher lädt eine Audio-Datei hoch. 5. Forscher klickt auf Speichern. 6. System speichert die Änderungen und schließt den Dialog. 7. Forscher klickt auf den Bearbeiten-Button von einem Interview. 8. System öffnet Interview-Bearbeiten-Dialog. 9. Forscher ändert den Wert einer Interview-Eigenschaft. 10. Forscher klickt auf Speichern. 11. System speichert die Änderungen und schließt den Dialog. 12. Forscher löscht ein Interview.
Erweiterung	5a. WENN der Forscher auf Abbrechen klickt, DANN werden die Änderungen nicht gespeichert. 10a. WENN der Forscher auf Abbrechen klickt, DANN werden die Änderungen nicht gespeichert.

- **Zerlegung von Use Case in Story Board-Elemente**

Story Board with Epics



Planung der Inkremente

Olga Boruszewski
FG Software Engineering

Use Case zeigt das gewünschte End-Verhalten

UC01	Ressourcen verwalten (Beamer, Rechner, Räume)
...	...
Hauptszenario	1. Mitarbeitet wählt „Ressourcen anzeigen“ 2. System zeigt alle Ressourcen in einer Liste an 3. Mitarbeiter wählt „Neue Ressource anlegen“ 4. System zeigt leeres Formular an 5. Mitarbeiter gibt ID und Name der Ressource an 6. Mitarbeiter wählt die passende Kategorie für die Ressource aus (Beamer, Rechner, Raum, ...) 7. Mitarbeiter bestätigt die Eingabe 8. System trägt die neue Ressource ein und zeigt die Liste mit allen Ressourcen an
Erweiterungen	6b. WENN es die passende Kategorie noch nicht gibt, DANN wählt Mitarbeiter „neue Kategorie erstellen“, gibt Kategorienamen an und bestätigt. Zurück zu Schritt 6. 8b. WENN es bereits eine Ressource mit derselben ID gibt, warnt das System den Nutzer. Weiter bei Schritt 5

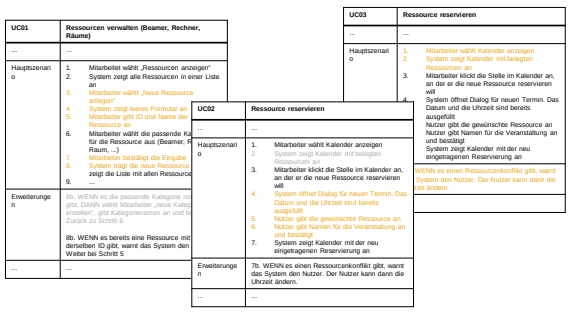
... aber nicht jeder Schritt ist gleich wichtig

UC01	Ressourcen verwalten (Beamer, Rechner, Räume)
...	...
Hauptszenario	1. Mitarbeitet wählt „Ressourcen anzeigen“ 2. System zeigt alle Ressourcen in einer Liste an 3. Mitarbeiter wählt „Neue Ressource anlegen“ 4. System zeigt leeres Formular an 5. Mitarbeiter gibt ID und Name der Ressource an 6. Mitarbeiter wählt die passende Kategorie für die Ressource aus (Beamer, Rechner, Raum, ...) 7. Mitarbeiter bestätigt die Eingabe 8. System trägt die neue Ressource ein und zeigt die Liste mit allen Ressourcen an
Erweiterungen	6b. WENN es die passende Kategorie noch nicht gibt, DANN wählt Mitarbeiter „neue Kategorie erstellen“, gibt Kategorienamen an und bestätigt. Zurück zu Schritt 6. 8b. WENN es bereits eine Ressource mit derselben ID gibt, warnt das System den Nutzer. Weiter bei Schritt 5

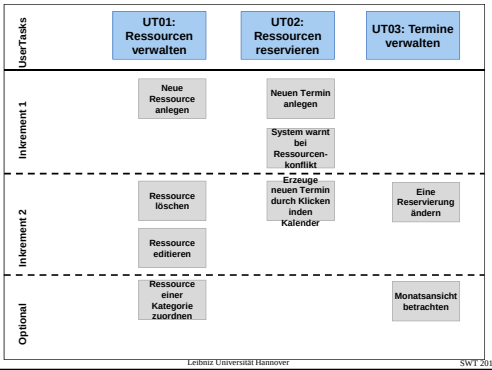
Besonders wichtige Schritte

Besonders unwichtige Schritte

Man verliert schnell den Überblick

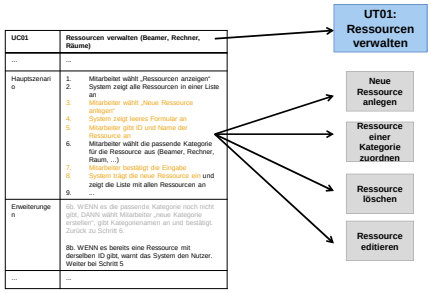


Agile Methoden helfen bei Planung der inkrementellen Entwicklung



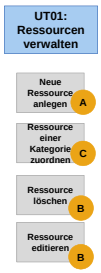
Story Map aus Use Cases ableiten

1. Lege UserTask zu Use Case an und erstelle Story Cards



Story Map aus Use Cases ableiten

2. Priorisiere Story Cards



A. **Wichtigste Funktionen. Ohne diese Story macht das ganze Produkt gar keinen Sinn.**

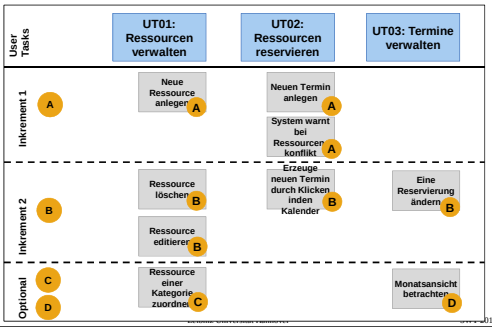
B. **Story ist wichtiger Bestandteil des Produktes. Ohne sie kann man aber auch schon sinnvolle Dinge mit dem Produkt machen.**

C. **Story fügt kleinere Funktionen hinzu oder macht Interaktion mit System angenehmer/hübscher, hat aber keinen großen Einfluss darauf, ob Nutzer das Hauptziel erreicht**

D. **Optionale Stories**

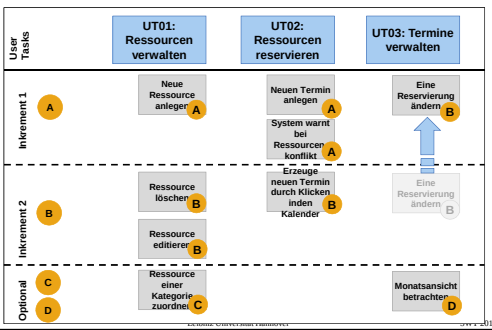
Story Map aus Use Cases ableiten

3. Ordne Story Cards zu Inkrementen zu



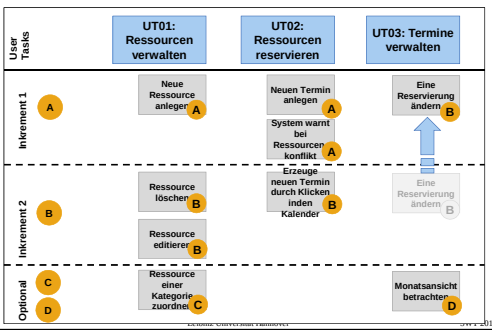
Story Map aus Use Cases ableiten

4. Gleiche Aufwand aus

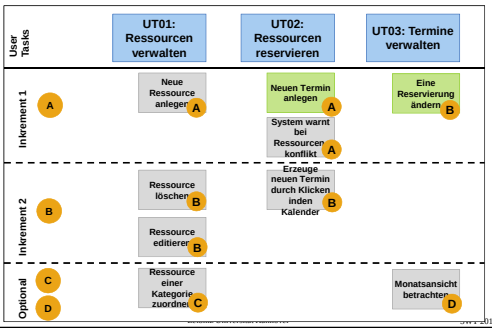


Story Map aus Use Cases ableiten

5. Bespreche Planung mit Kunden

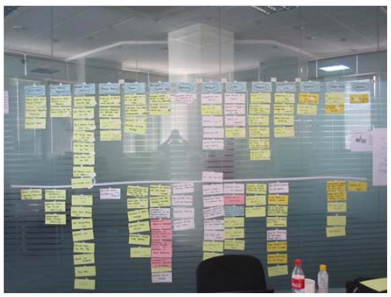


Verwende Story Map für Kommunikation mit Kunden



Oft im Einsatz bei agilen Teams: Stift und Papier

Leicht Anforderungen:
- neu hinzufügen
- verschieben
- austauschen
- Bilder dazu malen
- markieren
- ...



Bei uns: Flipchart und PostIts

KANBAN: Visualisierung des Workflow

- **Konkrete Umsetzung von LEAN-Prinzipien**
- **KANBAN ist sehr konkret**
  - Setzt auf existierendem Workflow auf, visualisiert u. verbessert ihn
- **Im Zentrum steht eine Visualisierung des Workflow**
  - Im Prinzip jede Prozessvisualisierung möglich
  - Meist aber: KANBAN-Board mit Signalkarten (=„kanban“)
- **Grundideen hinter dem KANBAN-Board**
  - Spalten zeigen Aufgaben und Wartepositionen
  - Aufgaben (Story Cards, Change Requests) „fließen“ durchs Sys.
  - Kapazitäten sind beschränkt, um WIP zu senken
  - Pull statt Push: Wenn Platz frei ist, wird neue Aufgabe „gezogen“
  - Gleichmäßiger Fluss durch ähnliche Aufgaben-Größe

Beispiel eines KANBAN-Boards

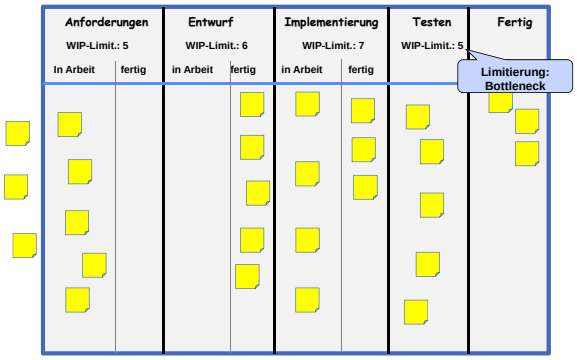
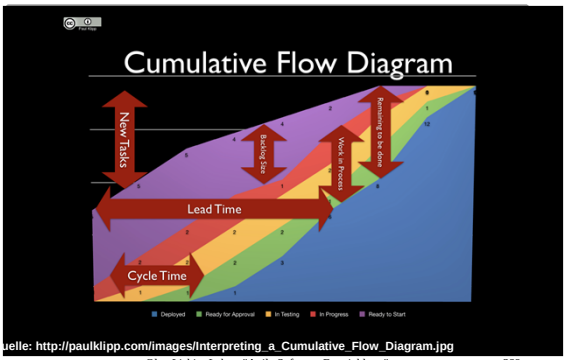


Abbildung 10.7 Eine einfache Prozessvisualisierung, auch für Kanban geeignet.ZWEI SUBSPALTEN

Flow messen & steuern



uelle: http://paullklipp.com/images/Interpreting_a_Cumulative_Flow_Diagram.jpg

Olga Liskin: Labor "Agile Software Entwicklung" 382

Fazit: Agil

- **Agile Methoden sind heute allgemein anerkannt**
  - **Nicht nur „weniger Dokumentieren“**
  - **Auch: Angemessener Planungshorizont**
  - **Flexibilität durch abgestimmte Maßnahmen**
- **Auch nicht-agile Projekte können profitieren**
  - **Time-Boxing, enger Kundenkontakt**
  - **inkrementelle Entwicklung**
  - **Anforderungspakete nach Nutzen sortieren**
- **SCRUM ist derzeit sehr populär**
- **Viele Unternehmen mischen und kombinieren**

Wir freuen uns auf Sie!

- **Vorlesung Software-Qualität**
- **Software-Projekt *SWP***
- **Bachelor- und Masterarbeiten (an Uni, mit Unternehmen, …)**
- **… und viele Labore, Projekte, Vorlesungen zu aktuellen Themen.**

- **Für qualifizierte und engagierte Studierende**
  - **Studentische Hilfskraft- oder Tutorentätigkeit**
  - **Vermittlung von Auslandsaufenthalten**
  - **Gutachten für Stipendien**

  Sprechen Sie uns an!