Agile Methoden II

Eine Einführung in Kanban

Agenda

- Warum Agile Entwicklung?
- Historie von Kanban
- Kanban in der Softwareentwicklung
- Scrum vs Kanban
- Fazit

Warum Agile Entwicklung?

Traditionelle Softwareentwicklung

- Beispielsweise
  - Wasserfallmodell
  - V-Modell, V-Modell XT
  - Rational Unified Process
- Starker Fokus auf Dokumentation, Planung, Tests, Verhandlungen, …
- Realistischer Anteil an „echter“ Programmierarbeit: 25-50%

Probleme traditioneller Paradigmen

- Probleme existierender Entwicklungsparadigmen
  - Starke Trennung von Rollen und Aufgaben
  - Arbeitsaufwand der Entwickler
  - Reaktionszeit auf Änderungen
  - Lange Entwicklungszyklen
  - Mangelnde Einbindung des Kunden

- Dies führte zu der Entstehung des *Agilen Manifests.*

Annahmen

- **Entwicklungskosten über Zeit**
  - Traditionelle Entwicklungsmethoden vermuten lineare oder sogar exponentielle Kosten von Entwicklung über Zeit.
  - Agile Methoden gehen von einer Kostenreduktion über die Zeit aus.
- **Änderungswünsche**
  wird es immer und jederzeit geben
- **Dokumentation**
  Mit ordentlichem Code zu arbeiten ist mindestens genauso effizient wie eine Dokumentation – eher besser.

Das Agile Manifest

Individuen und Interaktionen über Prozesse und Werkzeuge

Funktionierende Software über umfassende Dokumentation

Zusammenarbeit mit dem Kunden über Vertragsverhandlung

Reagieren auf Veränderung über das Befolgen eines Plans

Obwohl der rechte Teil eines Wertepaars ebenfalls wichtig ist, bewerten wir den linken Teil als wichtiger.

Individuen und Interaktion über Prozesse und Werkzeuge

- Traditionelle Softwareentwicklungsmethoden haben hunderte Regeln und Prozesse
- Die Werkzeuge, Regeln und Prozesse werden firmenweit vorgeschrieben – meist vom Management
- Interaktion ist langsam und indirekt durch große Hierarchien

- **Daher**:
  - Mehr Freiheit und Interaktionen zwischen Individuen
  - Das richtige Werkzeug für die richtige Situation
  - Bleibe bei keinem Prozess, der Dir im Wege steht

Funktionierende Software über umfassende Dokumentation

- Externe Dokumentation ist
  - Schwer verständlich zu schreiben und zu pflegen
  - Verfällt schnell
  - Trennt Programmierer, Dokumentationsautor und User

- **Daher**:
  - (Guter) Quelltext ist die beste Dokumentation
  - Interaktionen vor schriftlicher Dokumentation
  - Schnell funktionierende Software zum Kunden, um schnelle Rückmeldung zu bekommen

Zusammenarbeit mit Kunden über Vertragsverhandlungen

- Traditionelle Softwareentwicklungsmethoden
  - Vertragsverhandlungen passieren zwischen Management und Kunden
  - Softwareentwicklung ohne Integration des Kunden
  - Akzeptanztests erst nach langer Entwicklungszeit
  - Resultiert in unrealistischen Erwartungen und sozialer Separation von Kunde und Entwicklerfirma (wir gegen sie)

- **Daher**:
  - Kundenintegration in allen Entwicklungsphasen
  - Ein Team mit dem Kunden anstatt ein Team gegen den Kunden

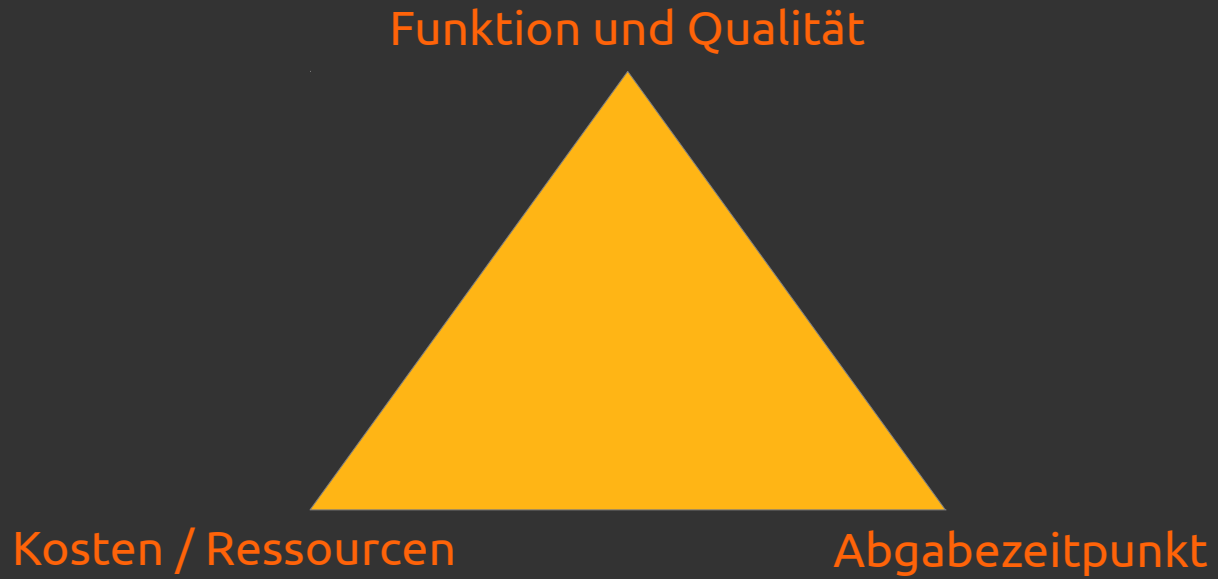
Reagieren auf Veränderungen
über das Befolgen eines Plans

- Traditionelle Softwareentwicklungsmethoden
  - Masterplan über einen kompletten und langen Entwicklungsprozess
  - Lange Entwicklungszeit, bis der Kunde Akzeptanztests durchführen kann
  - Änderungen in Anforderungen sind problematisch

- **Daher**:
  - Kurze Entwicklungszyklen / kontinuierliche Integration
  - Kunde entscheidet über nächste Entwicklungsschritte
  - Anforderungsänderungen werden erwartet und nicht gefürchtet

Das magische Dreieck des Projektmanagements

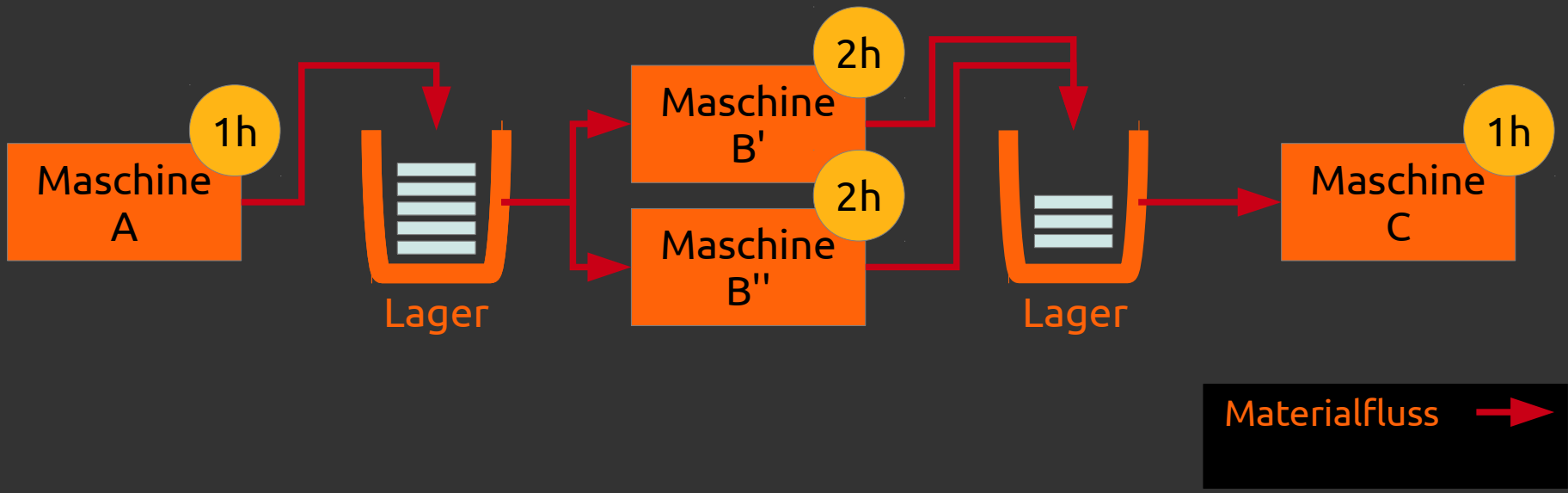
- Traditionell ist das Management und der Kunde vollständig verantwortlich (Lastenheft und Vertragsverhandlungen)
- In den agilen Methoden wird ein Teil davon an das Entwicklerteam abgegeben



Kanban

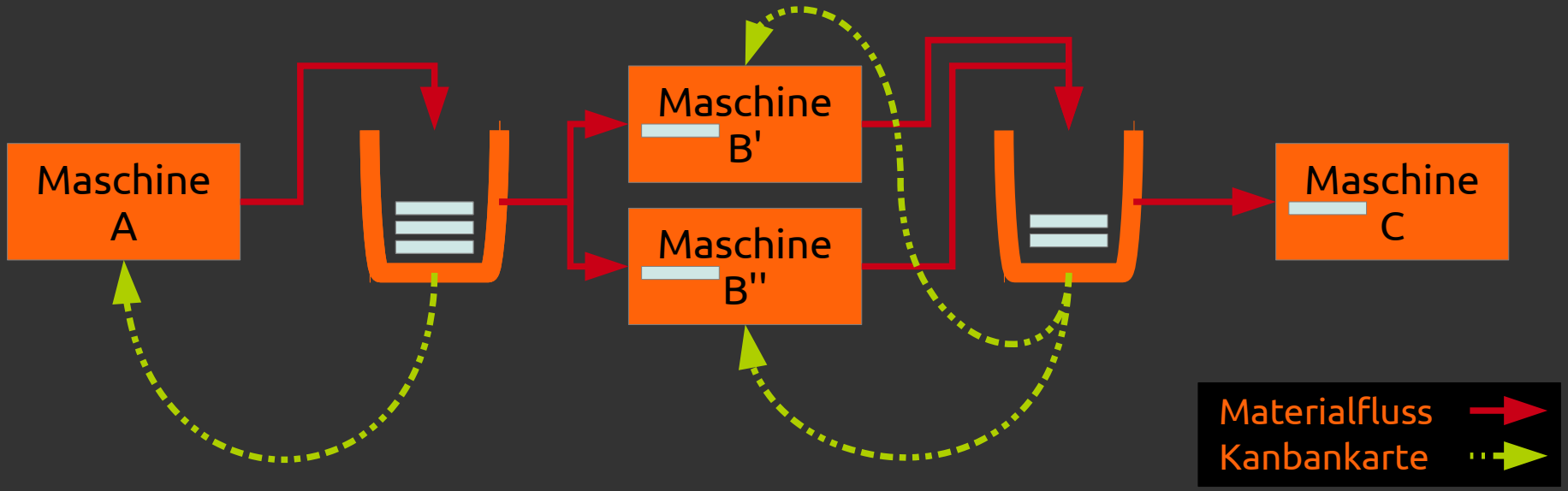
Einführung

- Kanban (japanisch, Karte / Tafel / Beleg)
- In den 70ern von Toyota entwickelt
- Kommt aus der Fertigungsindustrie
- Klassisches Pullverfahren zur Minimierung von Lagerbeständen (Just-In-Time Delivery)



Einführung

- Kanban (japanisch, Karte / Tafel / Beleg)
- In den 70ern von Toyota entwickelt
- Kommt aus der Fertigungsindustrie
- Klassisches Pullverfahren zur Minimierung von Lagerbeständen (Just-In-Time Delivery)



Kanban in der Softwareentwicklung

Kanban in der Softwareentwicklung

- Softwareentwicklung kennt auch unterschiedliche „Produktionsschritte“
  - Liste von Userstories
  - Nächste Userstories
  - Entwicklung
  - Akzeptanztests
  - Abgeschlossen (live)
- Unterschiedliche Entwicklungsschritte können auf „produzierende Ressourcen“ aufgeteilt werden
- Jeder Mitarbeiter holt sich bei Freiraum selbsttätig eine neue Userstory zum implementieren

3 Kernprinzipien

- Flow
- Limitierung der gleichzeitigen Arbeit (Work-In-Progress, WIP)
- Visualisierung

Flow

- Reibungsloser Durchlauf von Userstories, möglichst ohne Lagerzeiten und Leerlauf (also so schnell wie möglich)
  - Aufteilung der Arbeiten in kleine Pakete
  - Just-In-Time-Delivery
  - Fokus (klare Beschränkung auf möglichst wenig Parallelarbeit)
- Optimierungsmöglichkeiten
  - Bottlenecks identifizieren
  - Müll vermeiden (Probleme ansprechen und angehen)
  - Den gesamten Prozess betrachten, nicht nur die Stelle, an der es hängt

Limitierung von Parallelarbeit (WIP)

- Parallelarbeit bedeutet einen exponentiell wachsenden Overhead und Zeiteinbußen
  - Psychologisches Umschalten zwischen Themen
  - Höherer Management- und Koordinationsaufwand
  - Ständige Arbeitsunterbrechung und Neufokussierung

- Daher:
  - Limitierung der gleichzeitig abzuarbeitenden Pakete
  - Angefangene Pakete dürfen durch andere ausgetauscht werden

Visualisierung

- Nutzung eines physikalischen oder elektronischen Kanban-Boards
- Visualisiert den vereinbarten Entwicklungsprozess mit den Einzelschritten in den Spalten
- Jede Spalte sollte unterteilt werden in „in Arbeit“ und „fertig“ (letzteres bildet das „Lager“ ab)
- Kann weitere Informationen (z.B. unterschiedliche Projekte) in den Zeilen („Swimlanes“) abbilden

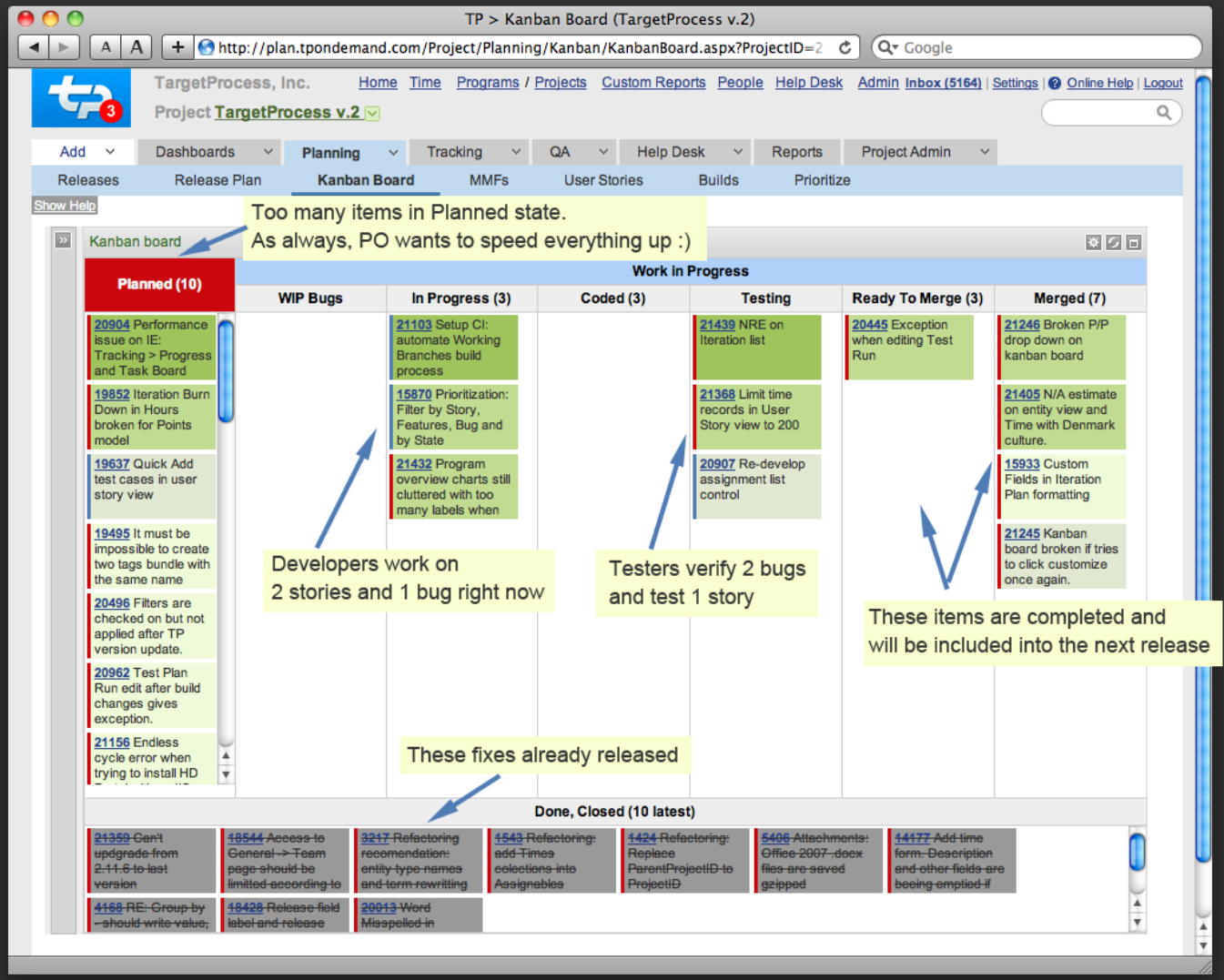
Board-Beispiel I

http://www.crisp.se/kanban/example

Board-Beispiel II

Quelle: http://multishoring.info/how-we-work/

Board-Beispiel III



Quelle: http://www.targetprocess.com/blog/2009/10/how-do-we-use-kanban-board-the-real-example.html

Board-Beispiel IV

Quelle: http://imageshack.us/photo/my-images/683/taskboardreallarge.jpg/

Anpassungen und Optimierungen

- Prozessänderungen: Mehr oder weniger Spalten?
- Änderungen der WIP
  - Innerhalb einer Spalte
  - In einer übergeordneten Spalte
- Änderung der zugeordneten Ressourcen

Scrum vs. Kanban

Wo stehen wir?

Scrum in a Nutshell

- Teams in kleine, cross-funktionale, selbstorganisierte Einheiten einteilen
- Arbeit in kleine Arbeitspakete zerlegen, schätzen und priorisieren
- Arbeitspakete in Iterationen fester Länge (~4 Wochen) einteilen – jede Iteration endet mit Produktivcode
- Involviere den Kunden als Domänenexperten zur Erstellung von Userstories und zu Priorisierung
- Plane regelmäßige Termine (Daily Standup, Retrospective, Weekly Planning Poker, etc.) mit klaren Zielen und Zeitbegrenzungen

Kanban in a Nutshell

- Visualisiere den Entwicklungsprozess
  - Pro Spalte ein Status
  - Zerlege die Arbeit in kleine Pakete, schätze die Zeit ein und hänge Sie ans Kanbanboard in den jeweiligen Status
  - Notiere die Zeit der Statusänderungen
- Limitiere die Anzahl an Karten pro Status
- Optimiere den Prozess

Scrum oder Kanban I

- Scrum ist definierter und hat mehr Regeln
- Kanban ist mehr ein Konzept als eine Entwicklungsmethode
- Erinnerung an das Agile Manifest:
  Individuen und Interaktionen über Prozesse und Werkzeuge
- Scrum und Kanban ergänzen sich hervorragend
- Nutze nichts, was Dir im Weg steht
- Mehr dazu:
  H. Kniberg & M. Skarin (2010): Kanban and Scrum

Scrum oder Kanban II

- Arbeitsbeschränkung
  - Scrum: WIP pro Iteration
  - Kanban: WIP pro Entwicklungsstatus
- Reaktion auf Änderungswünsche
  - Scrum: nur am Anfang einer Iteration
  - Kanban: auch innerhalb einer Iteration (in Austausch für ein anderes Arbeitspaket)

Scrum oder Kanban III

- Unterschiede in Boards
  - In einem Scrumboard ist die Iteration fertig, wenn alle Arbeitspakete abgearbeitet sind (also: Neues Board nach jeder Iteration)
  - Ein Kanbanboard wird nicht gelöscht, sondern kontinuierlich geändert.
- Daher: Maximale Größe einer Userstory
  - Scrum: eine Iteration
  - Kanban: auch über eine Iteration hinaus

Does it work?

- Manche unintuitive Techniken werden unterschätzt (z.B. Pairprogramming)
- Einiges an Erfolgsstories – wenige Berichte über Misserfolge (welche Firma veröffentlicht die freiwillig?)
- Sind Projekte wirklich so schlecht planbar?
- Ist Refactoring wirklich so leicht?

Fazit: Choose your weapons wisely

Ein abschließender Überblick über die Werkzeuge

Kommunikation

- Regelmäßige Meetings, mit klarem Ziel und klarer Zeitbeschränkung
  - Daily Standup
  - Weekly Planning Poker
  - Retrospective und Iteration Planning
- Räumlichkeiten evaluieren – das cross-funktionale Team sollte in einem Raum sein, nach Möglichkeit inkl. Kunde
- Regelmäßiges Feedback und Einbeziehen des Kunden sorgt für weniger Überraschungen und ein Dazugehörigkeitsgefühl

Verantwortlichkeiten bei den richtigen Personen

- Planning Poker
  - **Kunde** erstellt mit **Entwicklern** UserStories und priorisiert sie vor (kritisch, wichtig, nice-to-have)
  - **Entwickler** schätzen Zeit und planen Implementierung
  - **Kunde** und **Management** priorisieren und verhandeln über die nächsten, umzusetzenden Stories.
- Akzeptanztests werden vom **Kunden** zeitnah durchgeführt

Best Practices I

- Nutze eine gemeinsame Sprache / Metaphern
- Programmiere in Paaren
- Gemeinsamer Codebesitz und Pull-Prinzip anstatt Einzelverantwortungen und Push-Prinzip
- Nutze Test-Driven Development
- Nutze Refactoring und Design Patterns
- Keine Fixme's, Todo's, etc. - arbeite für's Jetzt

Best Practices II

- Kleine Arbeitsschritte, nichts parallel
- Stetige Code-Integration, sauberer Maintree
- Keine Überstunden
- Entwickle Codingstandards im Team
- Nutze Metriken, speichere viele Metadaten – das hilft bei der Retrospektive

Metriken

- Optimalschätzung mit Load Factor
  - Aufwandschätzungen werden nach dem Best-Case-Szenario abgegeben
  - Nach der Beendigung einer Aufgabe wird die Abweichung bestimmt
  - Die Summe aller Abweichungen bestimmt den Load-Factor, der für in die nächsten Planungen einfließt.
  - Nutze zwei unterschiedliche Load-Faktoren für wenig oder stark abweichende Schätzungen
- Nutze Planning Poker

Don't lose the passion
Fragen?

(Annotations and Blames welcome)

Quellen

[Eck09] Jutta Eckstein (2009): *Agile Softwareentwicklung mit verteilten Teams*, dpunkt.verlag, Heidelberg (Germany)

[Eck00] Jutta Eckstein (2000): *eXtreme Programming – ein leichtgewichtiger Software-Entwicklungsprozess*, http://www.jeckstein.com/papers/basProXP.pdf (checked: 2011-09-05)

[Rum01] Bernhard Rumpe (2001): *Extreme Programming – Back to Basics?*, Modellierung 2001, Workshop der Gesellschaft für Informatik e.V.(GI) 28.-30.3.2001, Bad Lippspringe. pp. 121-131

[Kni10] H. Kniberg, M. Skarin (2010): *Kanban and Scrum – making the most of both*, C4Media, http://infoq.com/minibooks/kanban-scrum-minibook (checked: 2011-09-01)