# Software Engineering – Blatt 10

Rasmus Diederichsen Felix Breuninger
{rdiederichse, fbreunin}@uos.de

20. Januar 2015

### Aufgabe 10.1: Evolutionäre / inkrementelle Modelle

#### a) Inkrementelle Modelle

Der inkrementelle Entwicklungsprozess ist durch folgende Aspekte gekennzeichnet:

- Software im Allgemeinen zunächst als Kernprodukt abgeliefert
- Nach und nach werden Features hinzugefügt
- Jedes Feature durchläuft einen kompletten Entwichklungszyklus (z.B. wie im Wasserfallmodell)
- Frühere Releases sind abgespeckte Versionen der späteren.
- Inkremente werden anhand der Benutzung des vorherigen Releases durch den Kunden durchgeplant unt entwickelt.

| Vorteile | Nachteile |
|---|---|
| Schneller Weg zur fertigen Kernfunktionalität | Unklar, wann es “fertig” ist |
| Es werden keine Releases weggeworfen, Qualitätsarbeit in jeder Iteraion | Inkremente v. A. nach User-Feedback → schwer planbar |
| Entwicklung kann bei Einhaltung on Restriktionen dennoch stets fertiges Produkt liefern (Ressourcenknappheit, Deadlines, Warten auf Technologien, etc.) | |
| Beliebige Erweiterbarkeit (potenziell langlebig) | |
| Kurze Iterationen machen Fehler weniger teuer | |
| Nachträgliche Ergänzungen einfach einzubauen | |

### b) Evolutionäre Modelle

Das evolutionäre Vorgehen ist charakterisiert durch

- Anforderungen, Pläne, Designs, Aufwandsschätzungen etc. entwickeln sich über die Zeit
- Entwicklung verläuft iterativ, indem die einzelnen Phasen nach Anpassung der Spezifikationen wiederholt werden
- In jeder Iteration werden die Bestandteile der Phasen (s.o.) verfeinert
- im Spiralmodell schreitet die Entwicklung spiralförmig fort, indem aufbauend auf dem Produkt der vorherigen Iteration immer alle Phasen neu durchlaufen werden.
- Anders als beim rein inkrementellen Vorgehen durchlaufen nicht einzelne Features den Entwicklungsprozess und werden “angetackert”, sondern die Gesamtsofware wird überarbeitet.

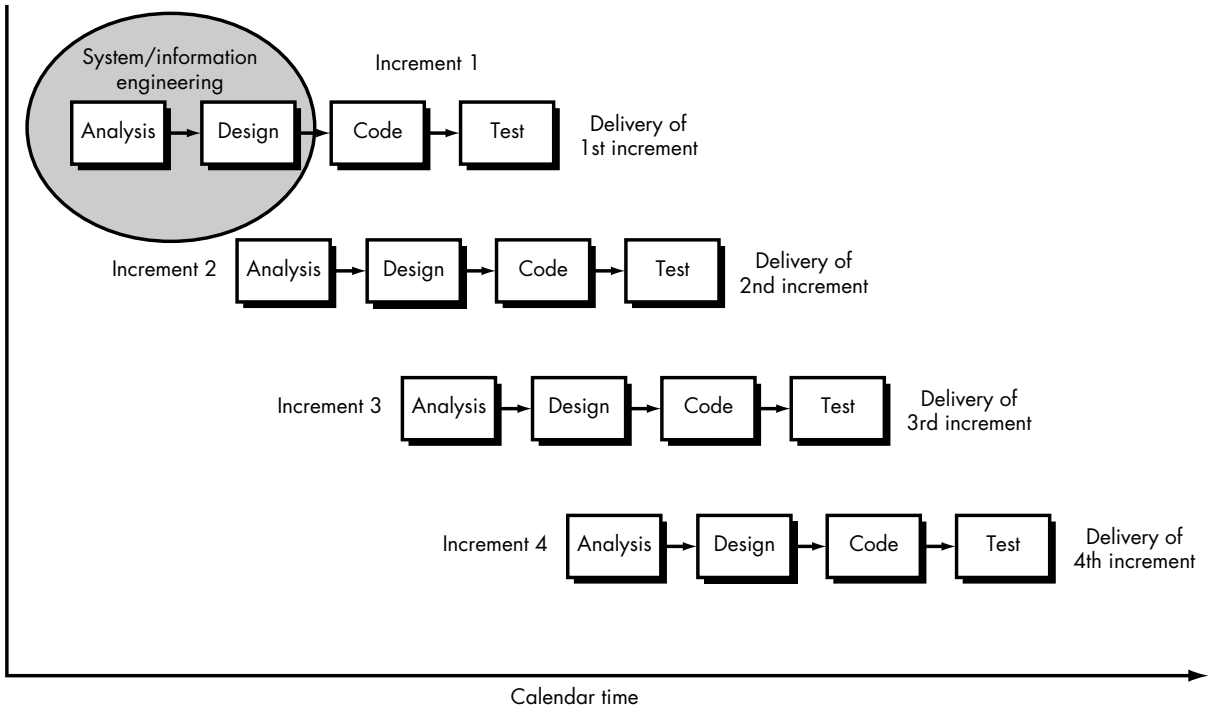


Abbildung 1: Iteratives Vorgehensmodell

| **Vorteile** | **Nachteile** |
|---|---|
| Hohe Anpassbarkeit an Kundenvorstellungen | Schwer planbar, da Anforderungen nicht zu Anfang bekannt |
| Schneller Prototyp | Versionen werden verworfen |
| Anforderungen müssen nicht alle zu Beginn bekannt sein | Häufige Änderungen sind schwer zu überblicken und können zu schlechterem Code führen |
| Sämtliche Projektdokumente werden verbessert, alles ist konsistent | |
| Prozess hat kein Ende, kann gesamte Lebensdauer abdecken | Kein klares Ende |
| Zu keinem Zeitpunkt im Lebenszyklus werden Aspekte vernachlässigt. | |

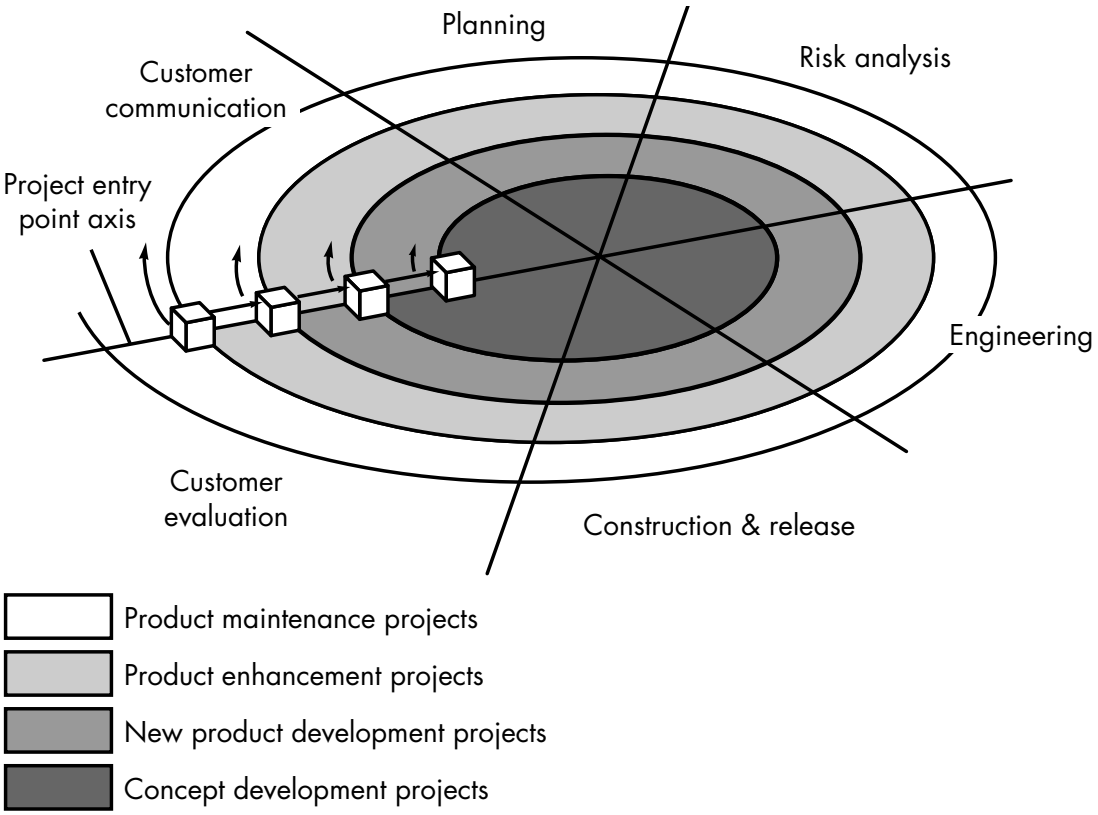


Abbildung 2: Spiralmodell; Evolutionäres Vorgehen mit definierten Task Sets

**Quellen**:

1. *Pressman, R.: Software engineering: a practitioner's approach, 5th ed.*
2. *Larman, C.: Agile and Iterative Development: A Manage's Guide*

### Aufgabe 10.2: Vorgehensmodelle

1. Da die Anforderungen bekannt sind und ein strikter Zeitplan möglich ist, bietes sich ein lineares Vorgehen wie Wasserfall oder V-Modell an. Es ist nicht erforderlich, möglichst schnell etwas präsentieren zu können, also muss man nicht agil vorgehen.
2. Rapides Prototyping erlaubt, das Problem notdürftig zu hotfixen, der Fix kann dann verbessert und stabilisiert werden. Möglich wäre auch ein Code-and-Fix-Ansatz, da hier höchste Dringlichkeit gegeben ist.
3. Evolutionär vorzugehen, würde es erlauben, die zwangsläufig auftretenden Änderungen miteinzubeziehen.
4. Das V-Modell expliziert die Notwendigkeit und Zuständigkeit verschiedener Tests. Dank der vorliegenden älteren Version kann das Projekt gut von vorn bis hinten durchgeplant werden, bevor man anfängt.
5. Code-and-Fix wäre hier ausreichend, da Dr. Blair kein Softwareingenieur ist und nur flott etwas demonstrieren muss. Die Software wird weder wirklich benötigt, noch wird sie länger in Betrieb sein.

### Aufgabe 10.3: Scrum

#### a)

Zu Beginn des Projektes definiert der Kunde eine Menge von Funktionalitäten und einen Zeitrahmen; die Funktionalitäten werden priorisiert und in das vom *Product Owner* gepflegte *Product Backlog* eingefügt. Dies kann in Form von User Stories mit assoziierten Story Points (Wichtigkeit, Aufwand) o.Ä. geschehen. In einem *Release Plan* wird dann die Dauer der Sprints und die grobe Verteilung der Features festgehalten.

Anhand des Zeitrahmens kann die Zahl an Features, die im nächsten *Sprint* abgearbeitet werden sollen, sowie die Länge der Sprints (typisch 1-4 Wochen), festgelegt werden. Für jeden Sprint werden die wichtigsten Features durch den Product Owner aus dem Product Backlog in das *Sprint Backlog* eingefügt und in dem für einen Sprint allokierten Zeitrahmen abgearbeitet. Das Entwicklerteam hält täglich ein *Daily Scrum* Meeting, in dem jedes Mitglied folgende Fragen beantwortet:

1. Was habe ich seit dem letzten Meeting erreicht?
2. Welche Probleme traten dabei auf?
3. Was werde ich bis zum nächsten Meeting schaffen?

Anhand der Antworten können der *Scrum Master*, der auch die korrekte Einhaltung des Vorgehens überwacht, und das Management z.B. die Größe des Sprint Backlogs oder andere Projektparameter variieren. Nach jedem Sprint wird die erreichte Funktionalität dem Kunden (oder allgemeiner *Product Owner*) vorgestellt. Während jedes Sprints kann ein *Burndown Chart* die Zahl noch umszusetzender Story

Points der Zeit im Sprint gegenüberstellen, sodass man sieht, ob man die gesteckten Ziele erreicht oder nicht, wenn man genau so weiter macht. Die Zahl noch zu erreichender Story Points nimmt idealerweise linear ab.

Analog dazu gibt es auch ein *Release Burndown Chart*, das den Gesamtfortschritt überwacht, sodass der Release Plan gegebenenfalls angepasst werden kann.

Probleme werden in das *Impediment Backlog* eingetragen.

Der *Product Owner* kann jederzeit neue Funktionalität fordern, diese wird aber zunächst ins Product Backlog eingefügt und daher erst im nächsten Sprint bearbeitet.

Ein Feature gilt als fertig, wenn es die durch den Kunden gegebene *Definition of Done* erfüllt.

b)

| Manifest-Element | Unterstützung durch Scrum |
|---|---|
| Individuals and interactions over processes and tools | Tägliche Meetings, keine strenge Vorschift, wie innerhalb eines Sprints vorzugehen ist |
| Working software over comprehensive documentation | Nach jedem Sprint kann eine Zwischenversion präsentiert werden, es gibt nur eine Handvoll Dokumente (s.o.) |
| Customer collaboration over contract negotiation | Reviews finden am Ende jedes Sprints statt, der Kunde kann jederzeit neue Anforderungen formulieren, es muss nicht alles ab Start fest stehen |
| Responding to change over following a plan | Zwar werden Pläne erstellt, sind jedoch änderbar. Änderungen können zu Anfang jedes Sprints Rechnugn getragen werden. |

c)

**Entwickle iterativ:** Sprints

**Überwache Erfüllung der Anforderungen:** Sprint reviews

**Verfolge Änderungen:** Product Backlog Refinement

**Schließe jeden Schritt mit Qualitätsprüfung ab:** Sprint reviews / DoD

**Lerne von Fehlern:** Sprint Retrospective Meeting

**Arbeite nach Plan:** Sprint / Product backlog

d)

**Product backlog:** Liste von Anforderungen die Umzusetzen sind. Anforderungen werden hier priorisiert und pro Sprint wird eine ausgewählte Anforderung komplett umgesetzt.

**Burndown Chart:** Visualisierung von Fortschritt in Bezug auf Gesamtprojekt / Sprint

**Definition of Done:** Kriterien und Anforderungen, wann eine Arbeit als fertig bezeichnet werden kann (z.B. Kommentare, Unit-Tests)

### Aufgabe 10.4: Kanban

Kanban ist ursprünglich ein Konzept zur schlanken Produktion von Autos, die Übertragung in die Software Entwicklung ist dem lean development entlehnt. Hierbei wird just-in-time und somit überschussfrei produziert und somit ein besserer Durchlauf der Materialien erreicht und Engpässe vermieden. Kaizen referenziert hier auf eine kontinuierliche Verbesserung ausgehend von der aktuellen Situation, und nicht etwa eine fundamentale Änderung in der Arbeitsweise oder einen Soll-Zustand als Ziel.

**Visualisierung des Arbeitsflusses und der Arbeit:** Alle nötigen Arbeitsphasen und -schritte bis zur Fertigstellung des Produktes oder der Software werden auf einem Whiteboard durch Notizzettel visualisiert. Ausserdem wird vermerkt, wann eine Phase als abgeschlossen gilt (Definition of Done). Somit durchwandern die Notizzettel der einzelnen Arbeitsschritte / Module die Phasen des Whiteboards von links nach rechts.

**Limitierung des WIP (Work In Progress):** Durch Ziehen neuer Aufgaben (statt "zugeschoben bekommen") und Begrenzung der Aufgabenzahl pro Station wird Überlastung einzelner Phasen vermieden und alte Aufgaben müssen abgeschlossen werden, bevor zu viele neuen begonnen werden können.

**Steuerung und Messung des Arbeitsflusses:** Durch Messung des Durchsatzes oder der Fehlerrate jeder Phase können Engpässe schnell behoben oder auf Probleme reagiert werden. Dieser Prozess findet kontinuierlich statt, der Gesamtprozess wird also (Kaizen folgend) evolutiv optimiert.

**Prozess-Regeln explizit machen:** Alle beteiligten Mitarbeiter müssen Regeln wie die Definition of Done und die Bedeutung der Phasen kennen, um den Gesamtprozess nachvollziehen zu können.

**Verbesserung durch bewährte Modelle:** Durch Anwendung von Modellen aus anderen Bereichen (z.B. Engpasstheorie) kann der Gesamtprozess weiter verbessert werden.