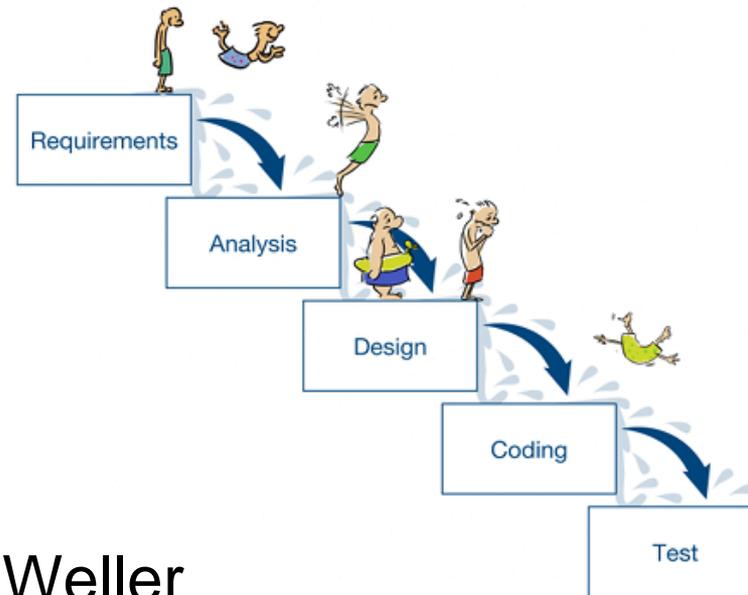




Media Engineering Prozessmodelle



R. Weller

University of Bremen, Germany

cgvr.cs.uni-bremen.de

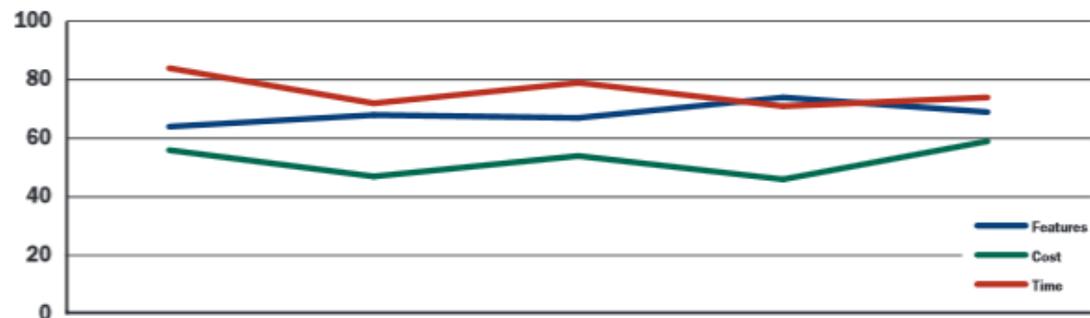
RESOLUTION

	2004	2006	2008	2010	2012
Successful	29%	35%	32%	37%	39%
Failed	18%	19%	24%	21%	18%
Challenged	53%	46%	44%	42%	43%

Project resolution results from CHAOS research for years 2004 to 2012.

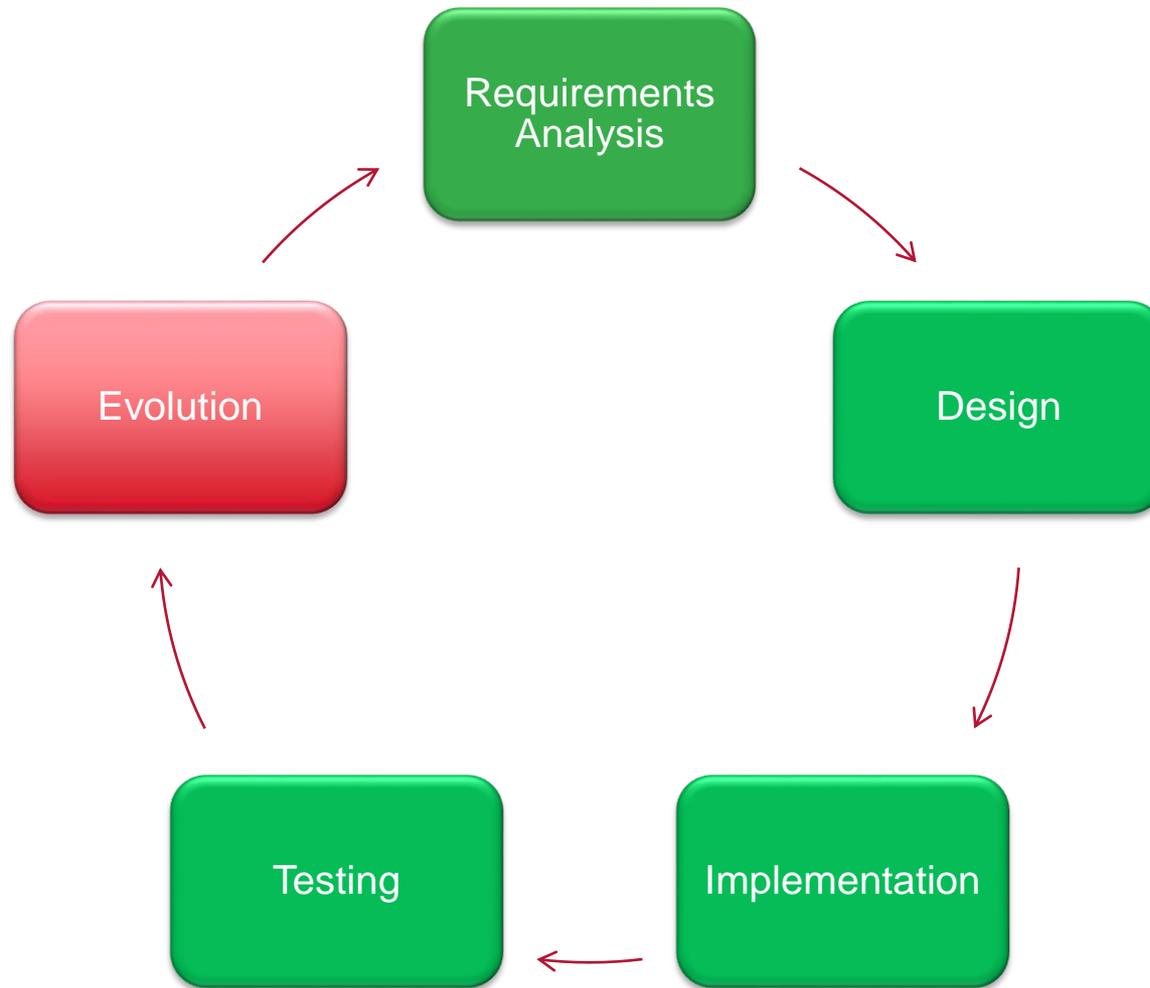
OVERRUNS AND FEATURES

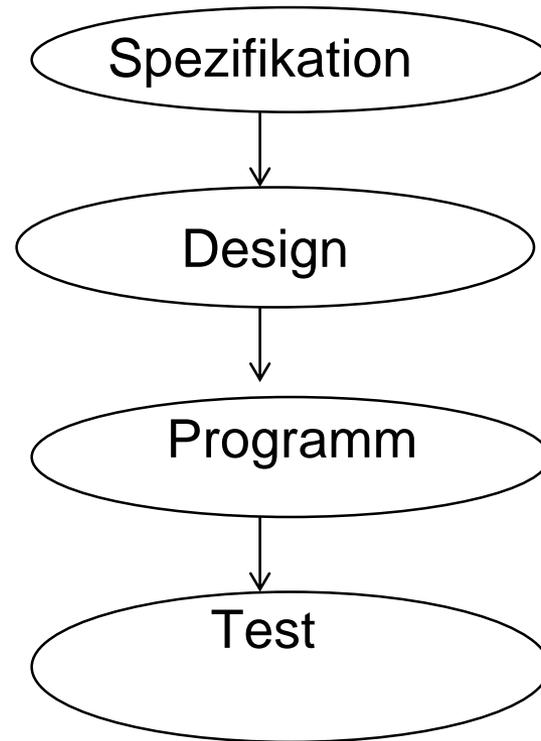
Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012.



	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%
FEATURES	64%	68%	67%	74%	69%

Der Software Development-Lifecycle





Was soll ich tun?

Wie mache ich es?

Was ich gemacht habe.

Habe ich es richtig gemacht?

↓ kausale Abhängigkeit
durch Informationsfluss

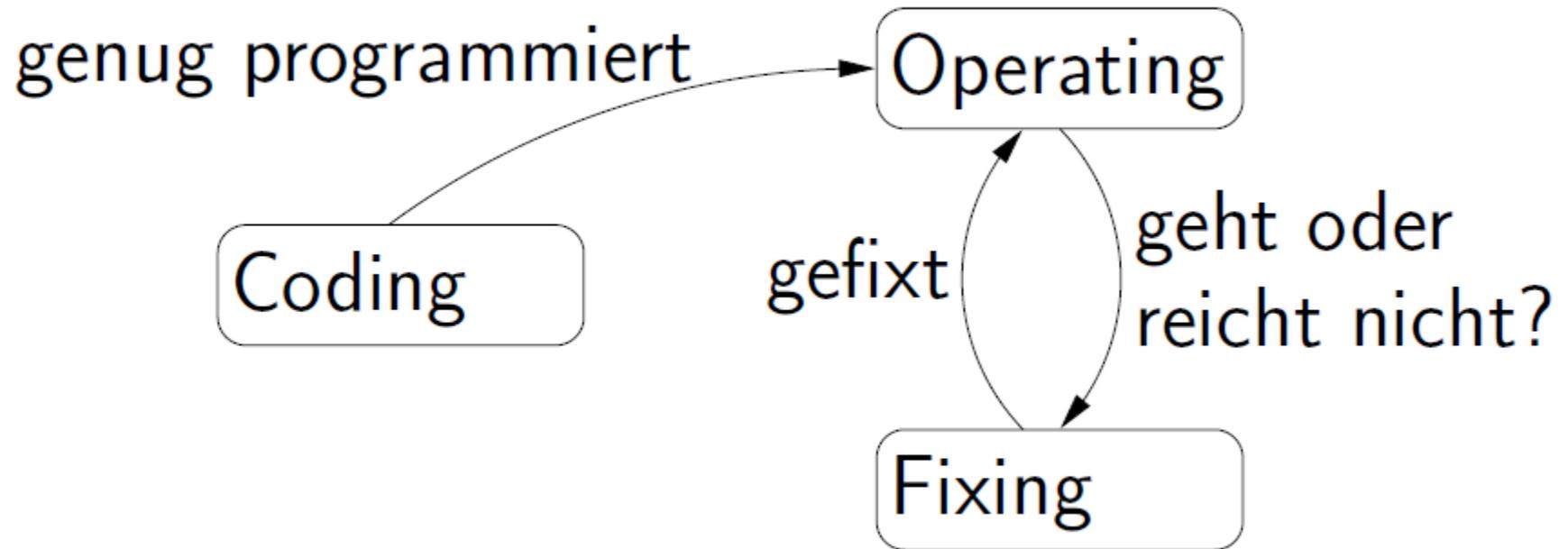
Definitionen

Softwareentwicklungsprozess: Der Prozess (d.h. die Sequenz der Schritte) durch den Benutzeranforderungen in ein Softwareprodukt umgewandelt werden (IEEE Std 610.12-1990 1990).

Prozessmodell: ein Prozess, der soweit abstrahiert ist, dass er für viele Organisationen Bedeutung hat (Ludewig 2003).



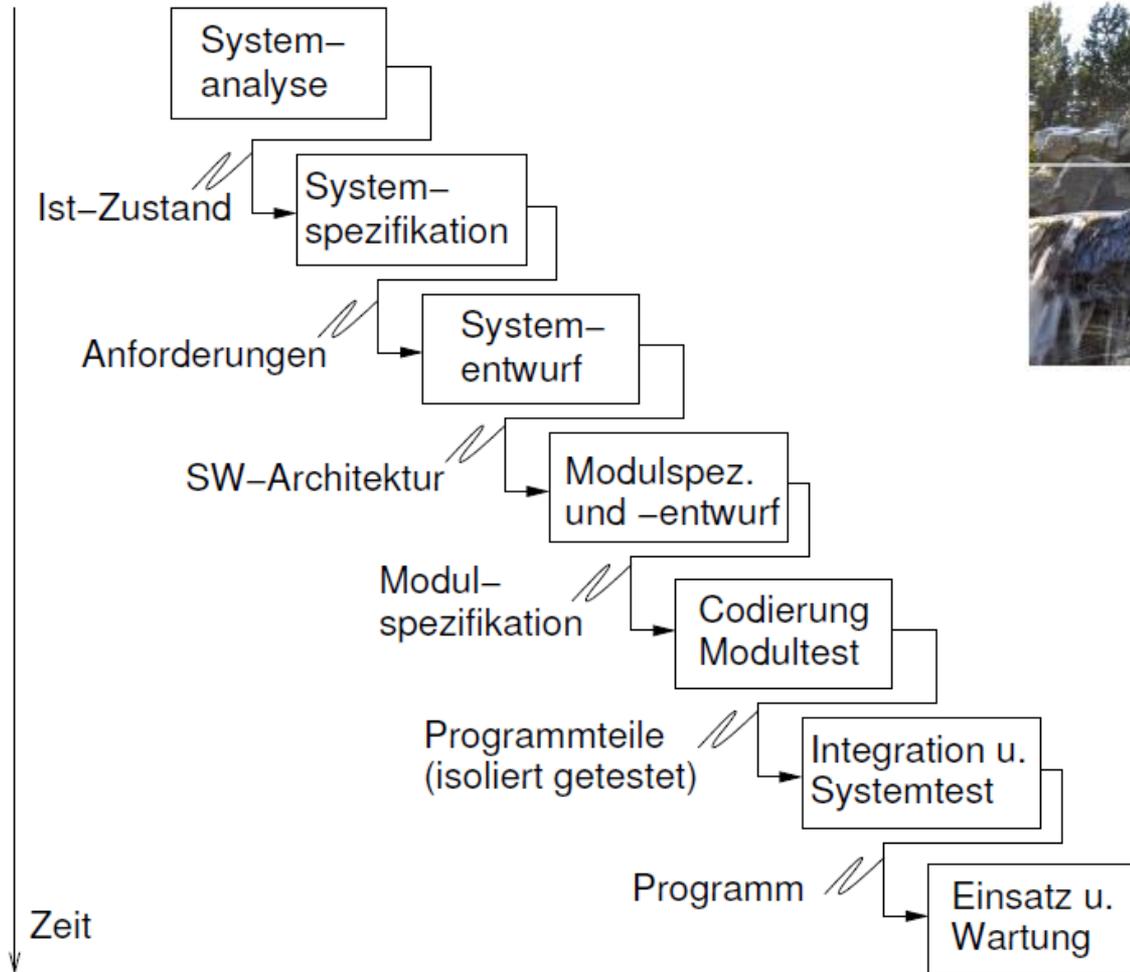




Grundlagen für guten Prozess:

- **Wohldefiniiertheit**
 - sonst Falsch-, Nicht-, Mehrarbeit
 - sonst Information nicht verfügbar oder unverständlich
- **Quantitatives Verstehen**
 - Objektive Grundlage für Verbesserungen
- **Kontinuierliche Verbesserung**
 - sonst Prozess schnell überholt
- **Angemessenheit**
 - Prozess muss dem zu lösenden Problem angemessen sein
 - d.h. effektiv und effizient sein

Striktes Wasserfallmodell Royce (1970)



Striktes Wasserfallmodell Royce (1970)

- Eigenschaften dieses Modells:
 - dokumenten-getrieben: jede Aktivität erzeugt Dokument
 - streng sequenzielle Aktivitäten
- Vorteile
 - + klar organisierter Ablauf
 - + relativ einfache Führung
 - + hohe Qualität der Dokumentation
- Nachteile
 - Entwicklung bildet langen Tunnel
 - 90%-fertig-Syndrom
 - Spezifikationsmängel lassen sich kaum frühzeitig erkennen
 - Entwicklung beim Kunden wird ignoriert



BRAND CAMP

by Tom Fishburne

THE NEW PRODUCT WATERFALL



HOW DO WE CHART OUR ENTIRE COURSE IF WE DON'T KNOW WHAT'S AHEAD?

PLAN



WHATEVER HAPPENS, JUST KEEP PADDLING!

BUILD

I WISH WE'D DESIGNED FOR THIS SCENARIO UPFRONT



TEST

PATCH IT AS BEST WE CAN. NO TIME TO CHANGE COURSE NOW

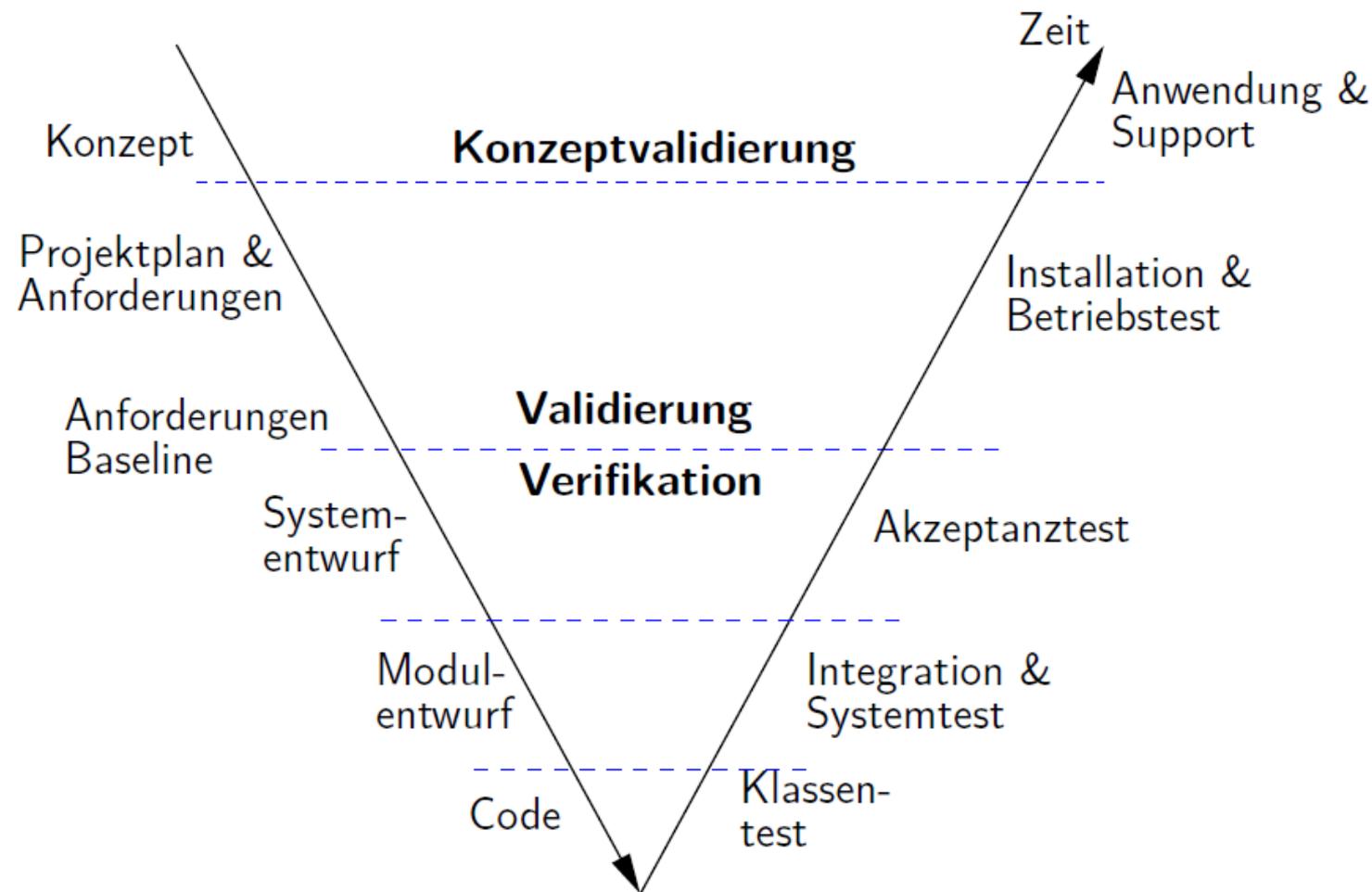


LAUNCH

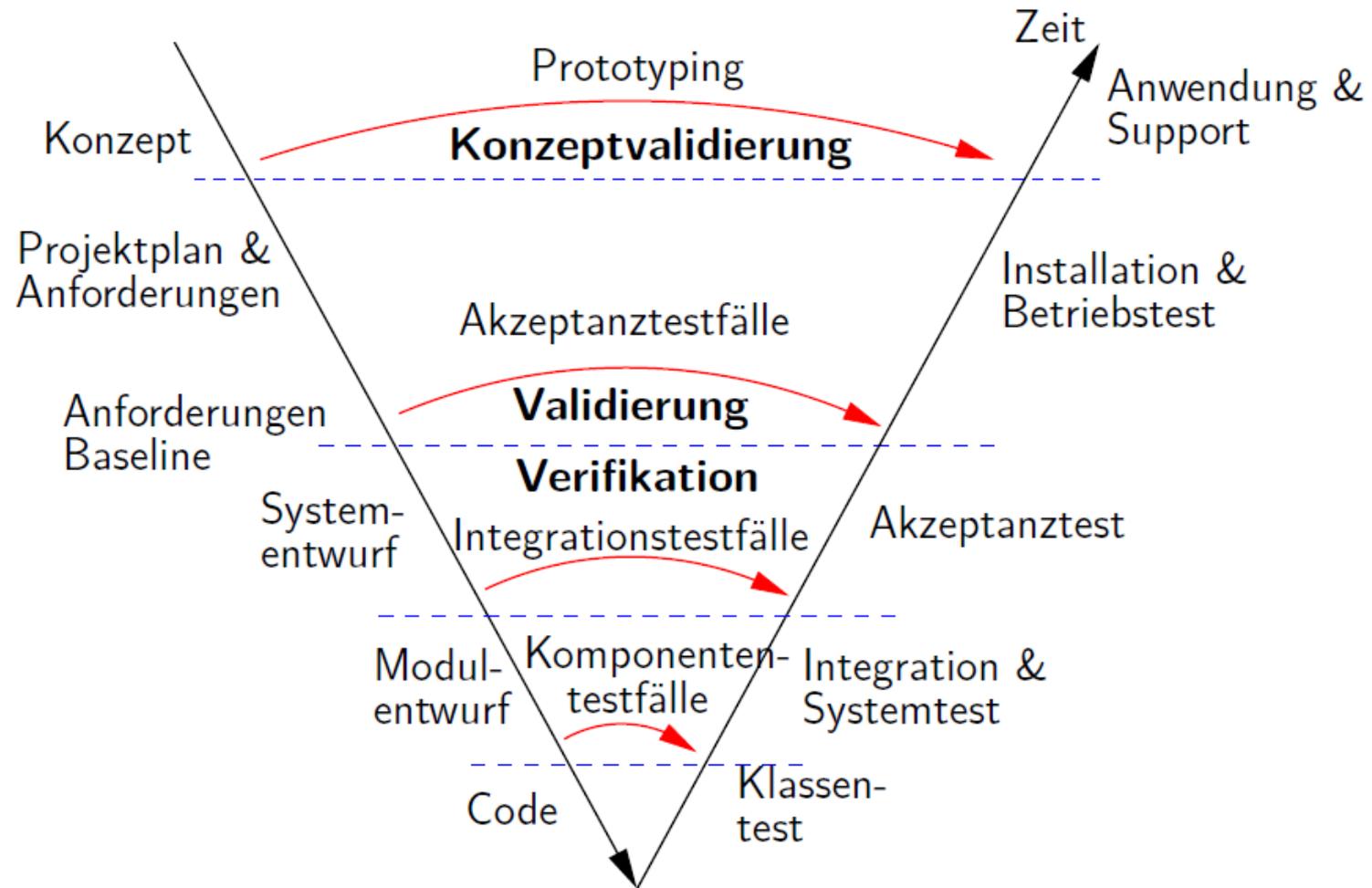
© 2010

TOMFISHBURNE.COM

V-Modell von Boehm (1979)



V-Modell von Boehm (1979)



V-Modell von Boehm (1979)

- Eigenschaften:
 - + betont Qualitätssicherung
 - + frühe Vorbereitung von Validierung und Verifikation
 - Fehler/Mängel werden früher entdeckt
 - + zusätzliche Parallelisierung
 - alle sonstigen Nachteile des Wasserfallmodells

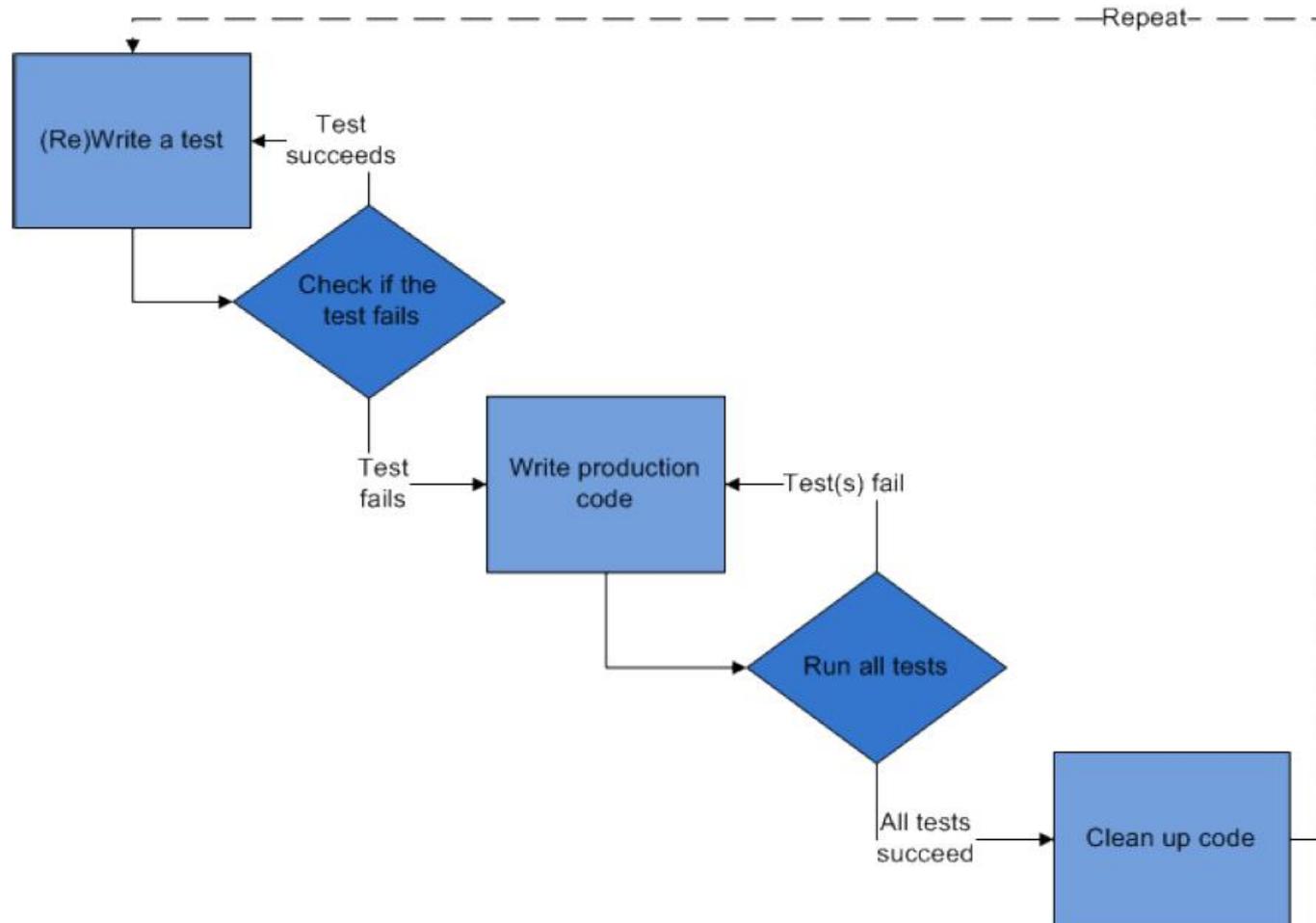
Testgetriebene Entwicklung (Sneed 2004)

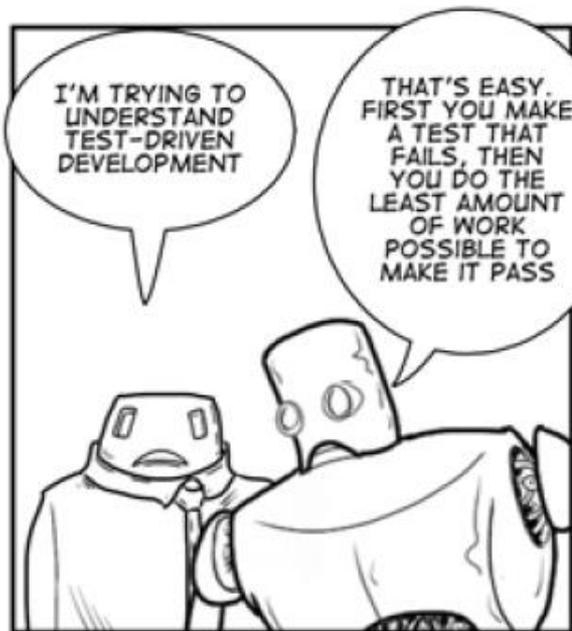
- Testgetriebene Entwicklung: Testfälle werden so früh wie möglich erstellt.
 - werden früh aus Anwendungsfällen abgeleitet
 - dienen als Baseline
 - treiben den Entwurf
 - treiben die Kodierung
- Test-Teams treiben die Entwicklung, statt von Entwicklern getrieben zu werden

Testgetriebene Entwicklung (Sneed 2004)

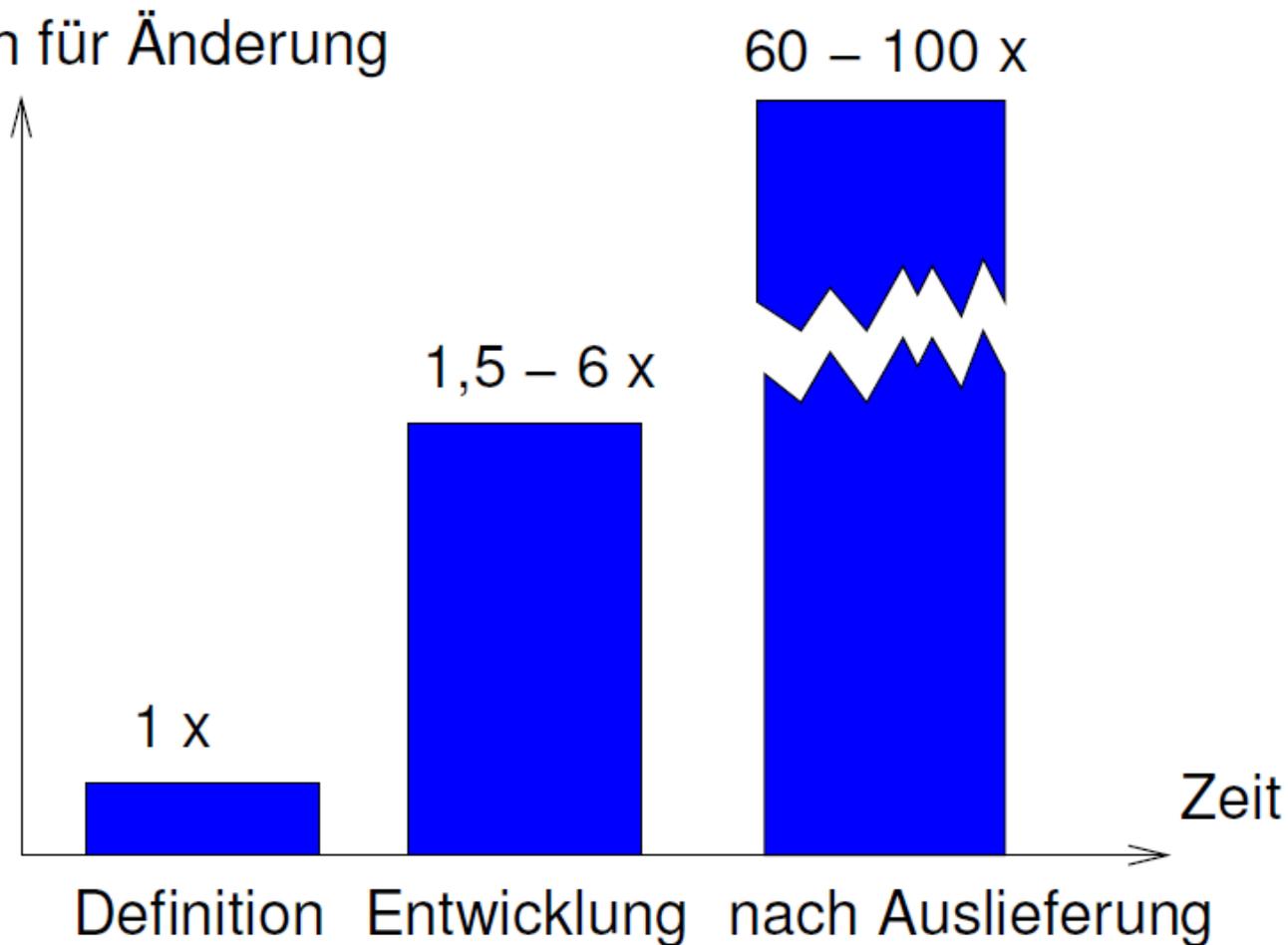
- Anwendungsfall: Kunde möchte Geld am Automaten abheben.
 1. Kunde gibt die Karte ein.
 2. System fragt nach PIN.
 3. Kunde gibt PIN ein.
 4. System fordert Benutzer auf, Karte zu entnehmen.
 5. Kunden entnimmt Karte.
 6. System fragt nach abzuhebendem Geldbetrag.
 7. Kunde gibt Betrag ein.
 8. System wirft das Geld aus.
 9. System beendet den Dialog.

Testfälle treiben die Kodierung:

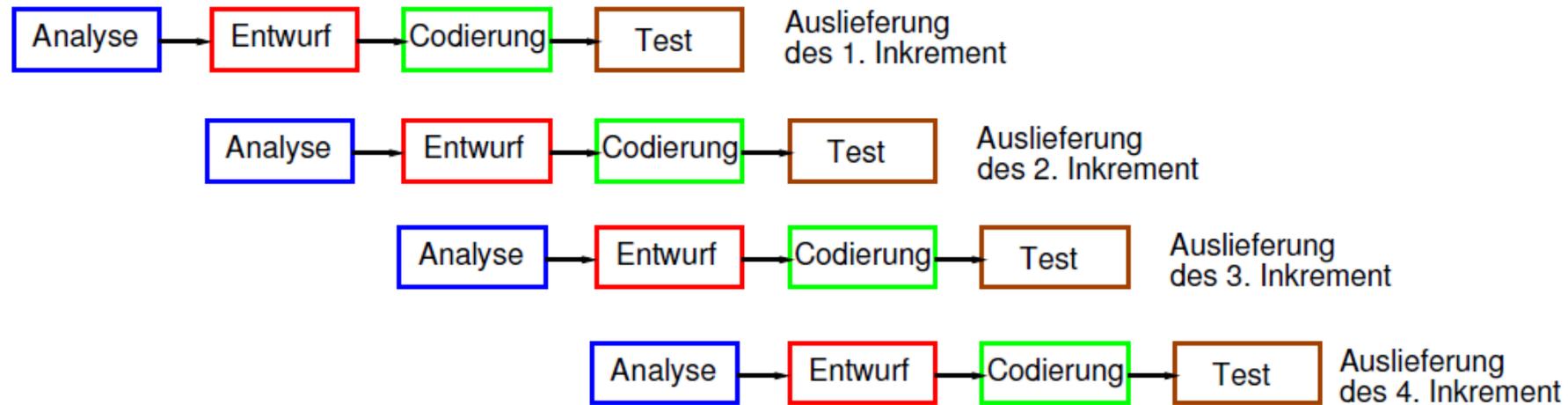




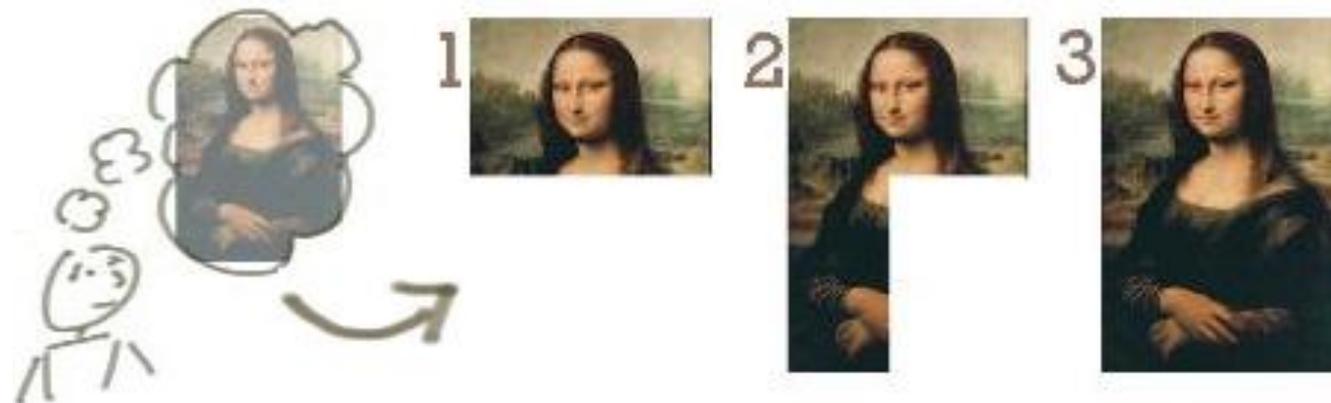
Kosten für Änderung



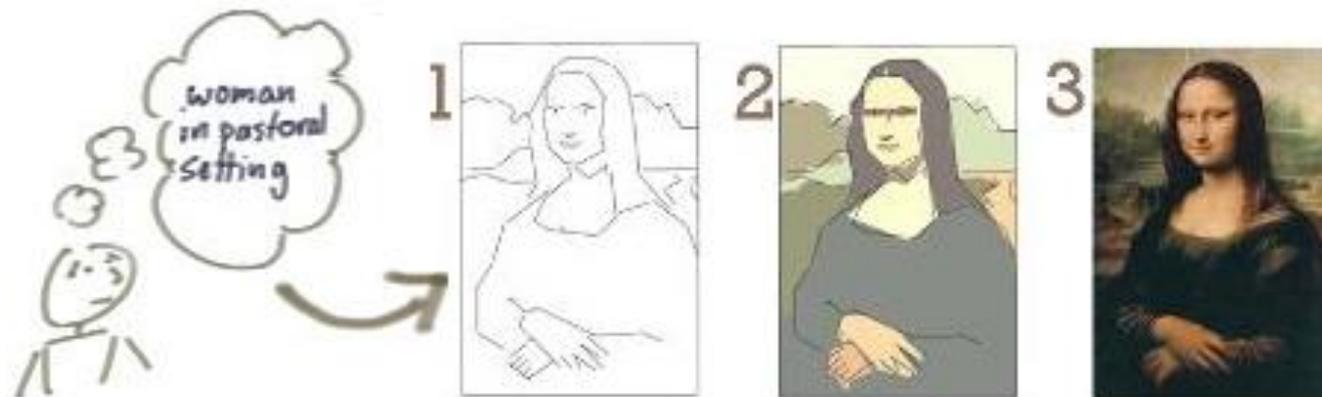
Inkrementelles Modell von Basili u. Turner (1975)



Incremental



Iterative

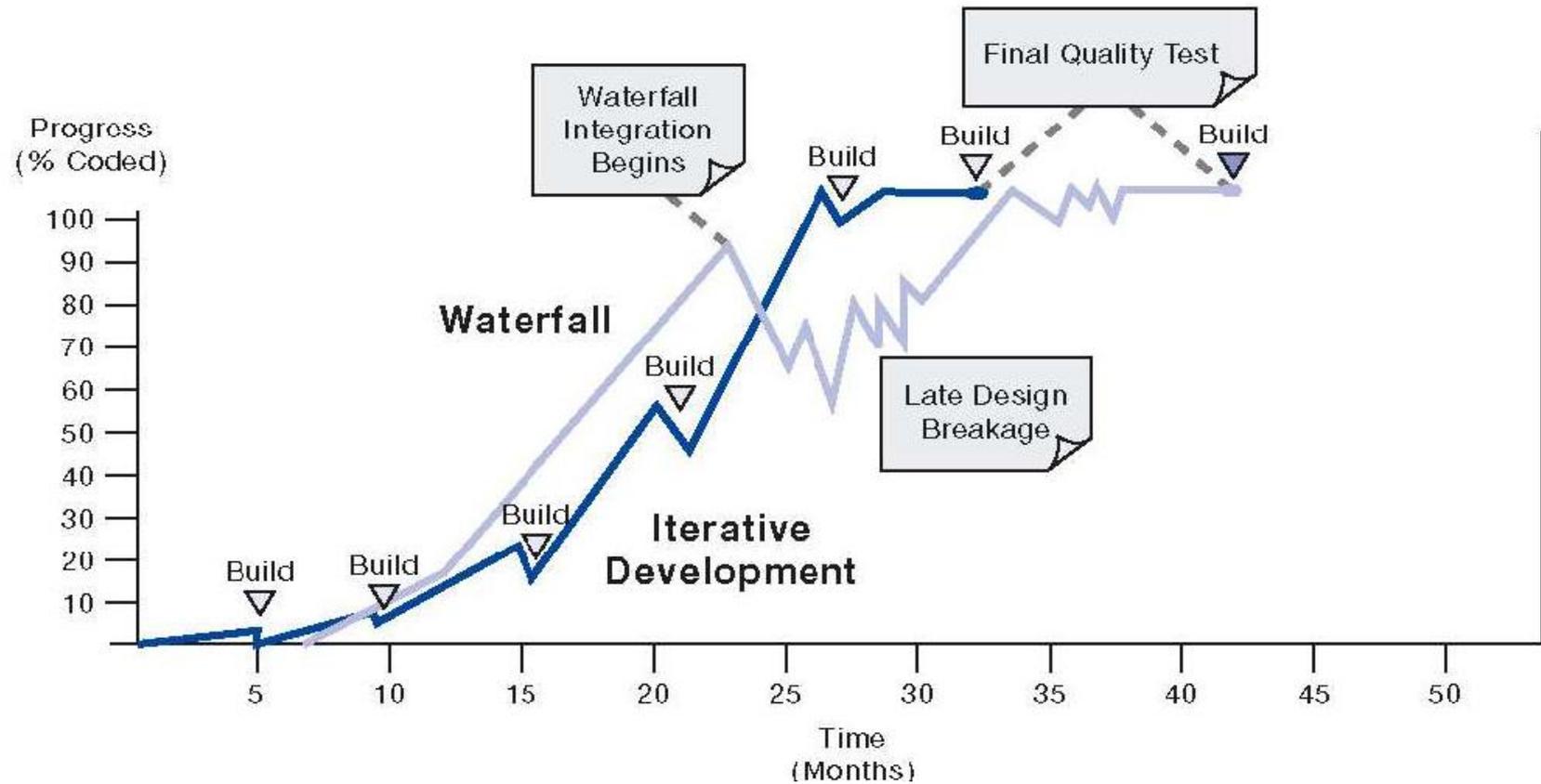


Iterationen:

1. grundlegende Funktionalität
 - Datei-Management, Editor, Textausgabe
2. erweiterte Funktionalität
 - Style-Files, Bearbeitung mathematischer Formeln, Einbinden von Graphiken
3. zusätzliche Funktionalität
 - Rechtschreibprüfung, Grammatiküberprüfung,
 - Überarbeitungsmodus
4. ergänzende Funktionalität
 - Tabellenkalkulation, Geschäftsgraphiken, E-Mail, Web-Browser, Scanner-Anbindung, Flipper

- Vorteile:
 - + Wartung wird als Erstellung einer neuen Version des bestehenden Produkts betrachtet.
 - + Entwicklung erfolgt stufenweise
 - brauchbare Teillösungen in kurzen Abständen
 - + Lernen durch Entwicklung und Verwendung des Systems.
 - + Gut geeignet, wenn Kunde Anforderungen noch nicht vollständig überblickt oder formulieren kann.
 - *"I know it when I see it"*
- Nachteile
 - Kernanforderungen und Architekturvision müssen vorhanden sein.
 - Entwicklung ist durch existierenden Code eingeschränkt.

Vergleich inkrementelles Modell und Wasserfallmodell





*If you do not
actively attack the
risks in your project,
they will actively
attack you.
– Gilb (1988)*

Spiralmodell von Boehm (1988)

Mehrere Iterationen der folgenden Schritte:

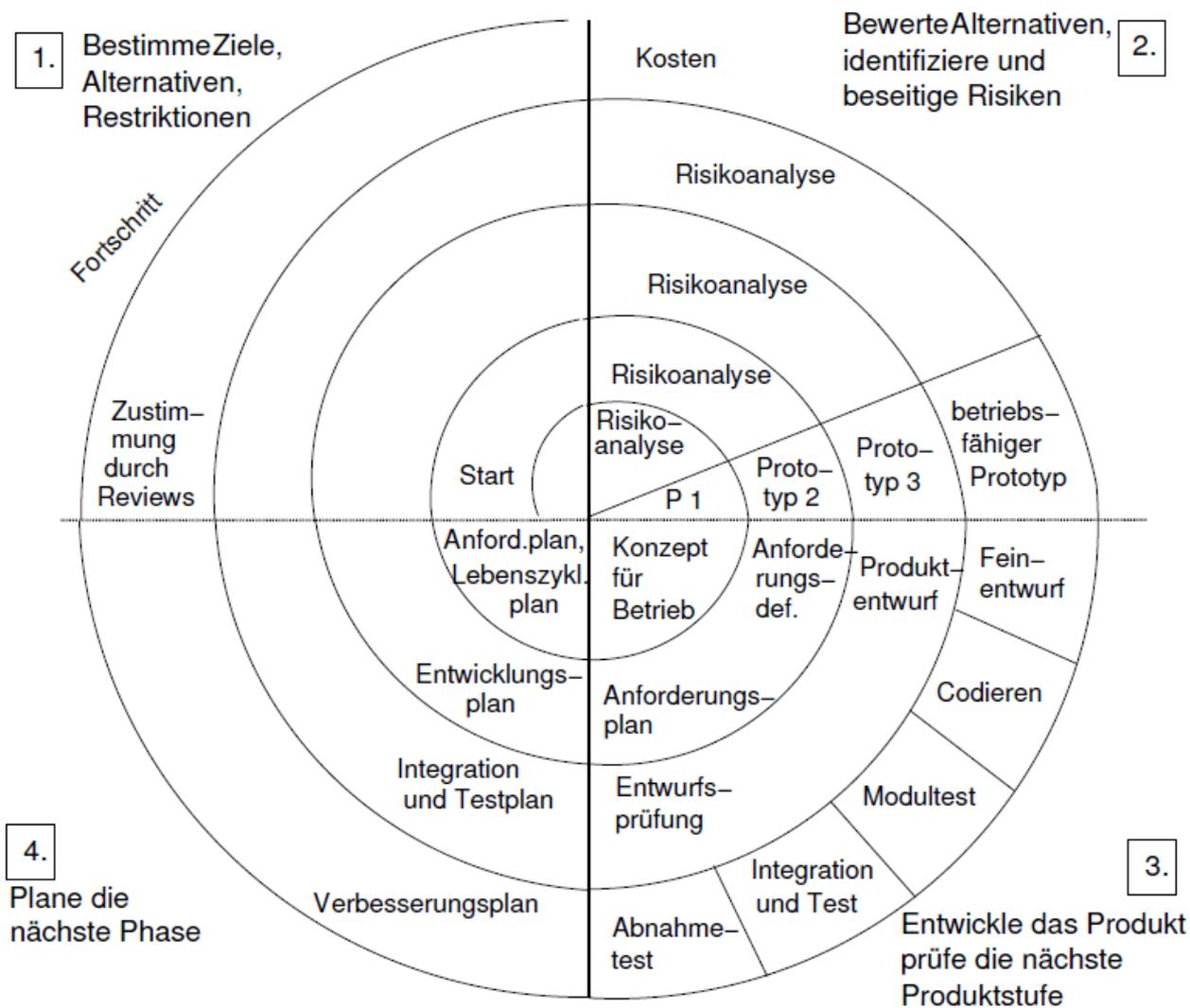
1. **Bestimmung der Ziele** und Produkte des Durchlaufs;
Berücksichtigung von Alternativen (z.B. Entwurfsvarianten) und Restriktionen (z.B. Zeitplan)
2. **Bewertung der Risiken** für alle Alternativen; Entwicklung von Lösungsstrategien zur Beseitigung der Ursachen
3. **Arbeitsschritte durchführen**, um Produkt zu erstellen
4. Review der Ergebnisse und **Planung** der nächsten Iteration

(Generische) Risiken:

- Ist das Konzept schlüssig? Kann es aufgehen?
- Was sind die genauen Anforderungen?
- Wie sieht ein geeigneter Entwurf aus?



Beispiel eines Spiralmodells mit vier Durchläufen



- Vorteile:

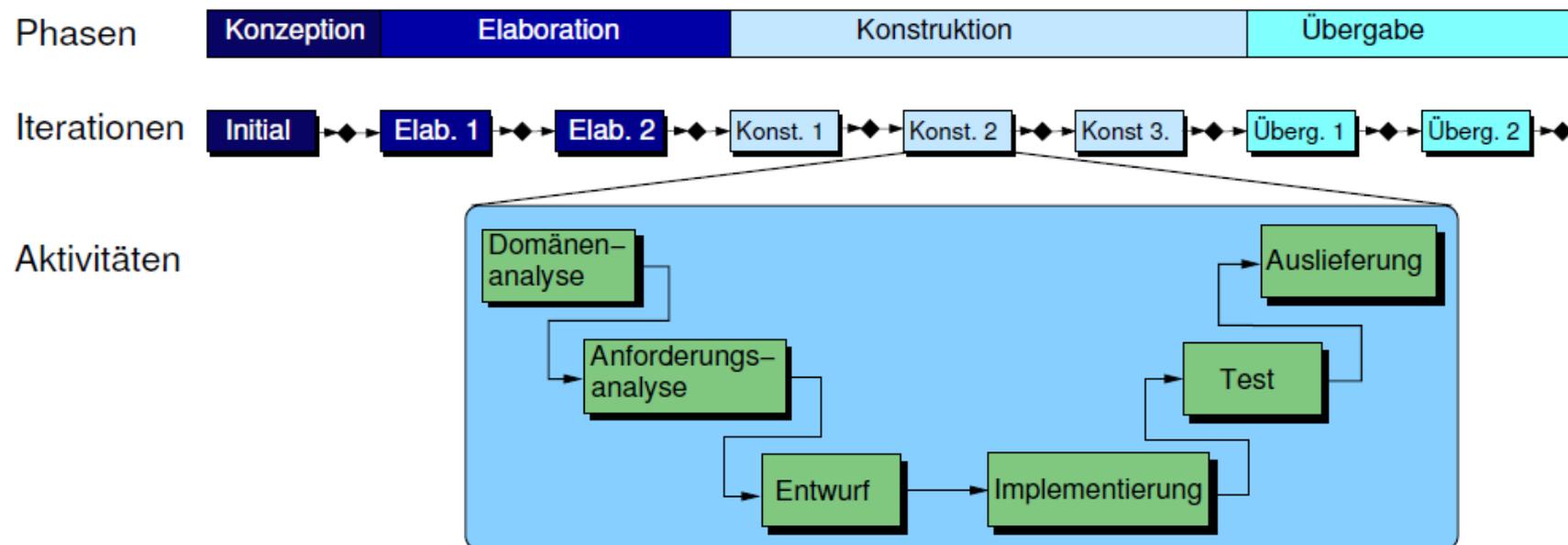
- + bei unübersichtlichen Ausgangslagen wird die Entwicklung in einzelne Schritte zerlegt, die jeweils unter den gegebenen Bedingungen das optimale Teilziel verfolgen

- Nachteile

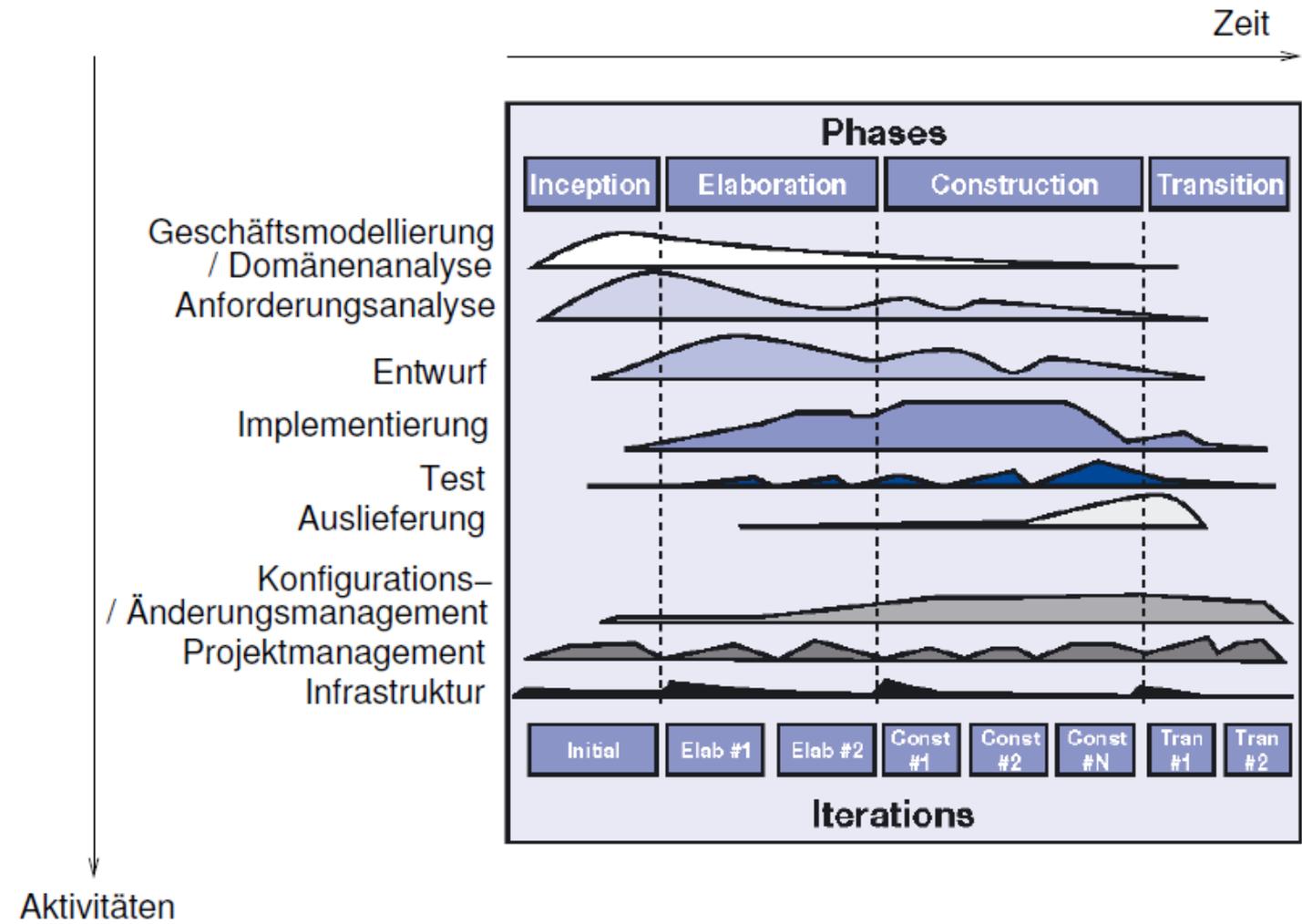
- schwierige Planung (was jedoch dem Problem inhärent ist)
 - setzt große Flexibilität in der Organisation und beim Kunden voraus

Meta-Modell: Iterationen können beliebigen Modellen folgen

Rational Unified Process RUP nach Gornik (2001)



Rational Unified Process RUP nach Gornik (2001)



RUP Konzeptionsphase (Inception)

Ziel: “Business-Case” erstellen und Projektgegenstand abgrenzen.

Resultate:

- Vision der Hauptanforderungen, Schlüsselfeatures und wesentliche Einschränkungen
- initiale Anwendungsfälle (10-20% vollständig) Glossar oder auch Domänenmodell
- initialer Business-Case: Geschäftskontext, Erfolgskriterien Schätzung des erzielten Gewinns, Marktanalyse etc.) und Finanzvorschau
- initiale Risikobetrachtung
Projektplan mit Phasen und Iterationen Business-Modell falls notwendig
- ein oder mehrere Prototypen



RUP Elaborationsphase (Elaboration)

Ziel: Verständnis der Anwendungsdomäne, tragfähige Software-Architektur, Projektplan, Eliminierung der Risiken

- Anwendungsfallmodell (mind. 80% fertig)
 - alle Anwendungsfälle und Akteure sind identifiziert,
 - die meisten Anwendungsfallbeschreibungen wurden entwickelt
- zusätzliche nichtfunktionale Anforderungen und Anforderungen, die nicht mit einem spezifischen Anwendungsfall assoziiert sind
- Beschreibung der Software-Architektur ausführbarer Architekturprototyp
- überarbeitete Liste der Risiken und überarbeiteter Business-Case
- Plan für das gesamte Projekt sowie grober Plan für die Iterationen und Meilensteine
- ein vorläufiges Benutzerhandbuch (optional) 



RUP: Konstruktionsphase (Construction)

Ziel: Fertigstellung, Integration und Test aller Komponenten; auslieferbares Produkt.

- Software-Produkt integriert in die entsprechende Plattform
- Benutzerhandbuch
- Dokumentation des gegenwärtigen Releases



RUP: Übergabephase (Transition)

Ziel: Produkt wird der Benutzergemeinde übergeben.

Hauptziele im Einzelnen:

- Benutzer sollen sich möglichst alleine zurecht finden
- Beteiligte sind überzeugt, dass die Entwicklungs-Baselines vollständig und konsistent mit den Evaluationskriterien für die Vision sind.
- Erreichen der letzten Produkt-Baseline so schnell und kostengünstig wie möglich



Typische Tätigkeiten:

- “Beta-Test”, um das neue System zu validieren
- Parallele Verwendung mit einem Legacy-System, das durch das Produkt ersetzt werden soll
- Konversion aller notwendigen Daten (Dateien und Datenbanken)
- Schulung aller Benutzer und Administratoren
- Übergabe an Marketing, Vertrieb und Verkäufer

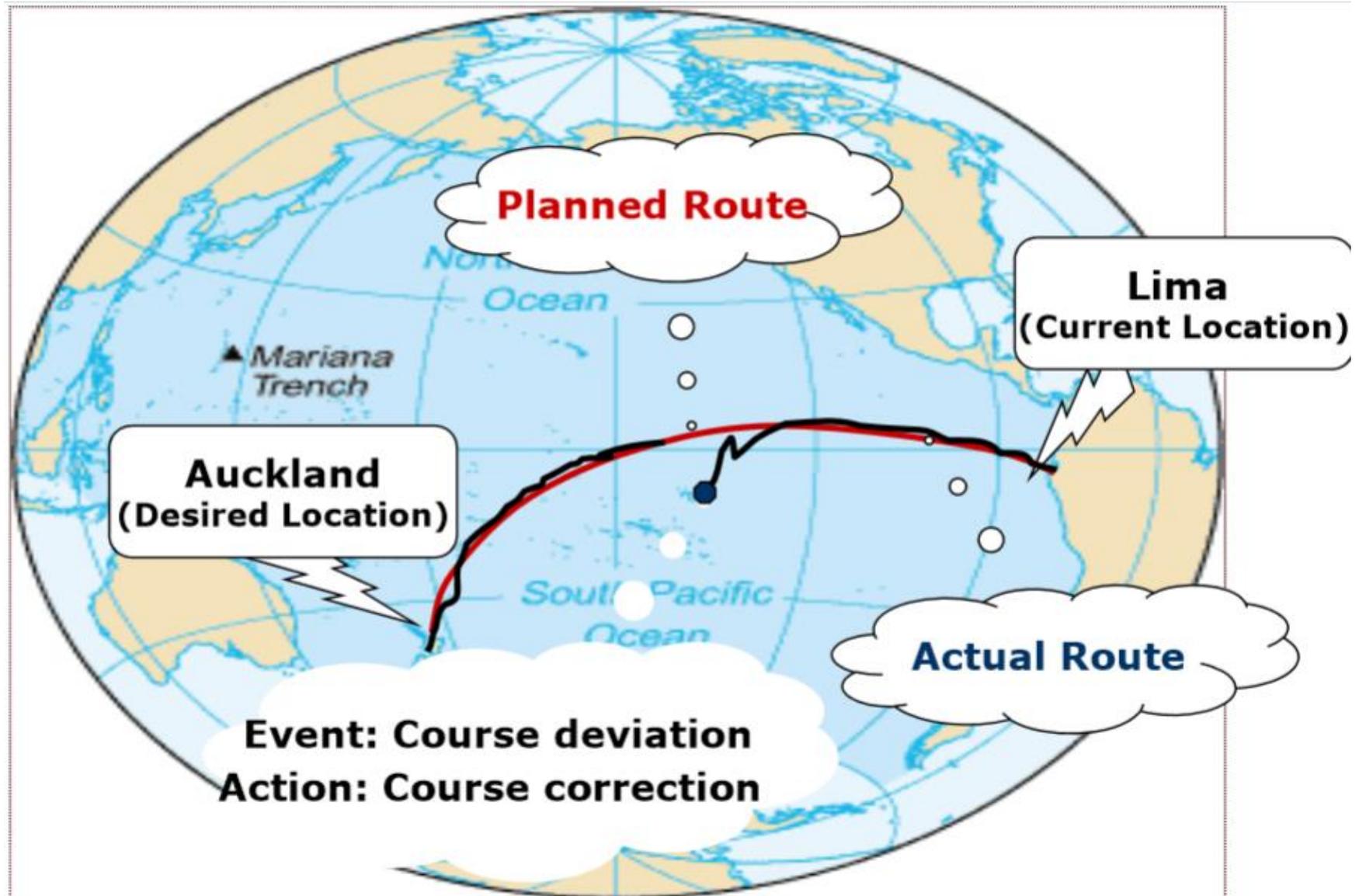


Empfohlene Anzahl von Iterationen nach Kruchten (1998)

Komplexität	niedrig	normal	hoch
Konzeption	0	1	1
Elaboration	1	2	3
Konstruktion	1	2	3
Übergabe	1	1	2
Summe	3	6	9

Bewertung der RUPs

- Übernimmt vom Spiralmodell die Steuerung durch Risiken
Konkretisiert die Aktivitäten (Spiralmodell ist ein Meta-Modell)
- Änderungen der Anforderungen sind leichter einzubeziehen als beim Wasserfallmodell
- Projekt-Team kann im Verlauf hinzulernen
- (Hauptsächlich) Konstruktionsphase kann inkrementell ausgestaltet werden





- Das sagt der Duden:

agil

Wortart: **Adjektiv**

Gebrauch: **bildungssprachlich**

Häufigkeit: ■ ■ □ □ □



BEDEUTUNGSÜBERSICHT

von großer Beweglichkeit zeugend; regsam und wendig

Beispiele

- ein agiler Geschäftsmann
- sie ist trotz ihres Alters körperlich und geistig noch sehr agil

Agiles Manifest (Februar 2001, Utah)

Wir entdecken bessere Wege zur Entwicklung von Software, indem wir Software entwickeln und anderen bei der Entwicklung helfen. Durch diese Tätigkeiten haben wir gelernt, dass uns

- | | | |
|---|---------------|---|
| <ul style="list-style-type: none"> ▪ <i>Individuen und Interaktion</i> ▪ <i>Funktionierende Software</i> ▪ <i>Kooperation mit Projektbetroffenen</i> ▪ <i>Reaktion auf Änderungen</i> | wichtiger als | <ul style="list-style-type: none"> ▪ Prozesse und Werkzeuge ▪ Umfangreiche Dokumentation ▪ Vertragsverhandlungen ▪ Verfolgung eines detaillierten Plans |
|---|---------------|---|

Natürlich sind auch die Dinge rechts wichtig, aber im Zweifelsfall schätzen wir die linken höher ein.



#113 - "AGILE DEVELOPMENT, EXPLAINED" - BY SALVATORE IOVENE, FEB. 21ST 2009

[HTTP://WWW.GEEKHEROCOMIC.COM](http://www.geekherocomic.com)

Prinzipien der agilen Entwicklung

- Kundenzufriedenheit durch **schnelle Auslieferung** nutzbarer Software
- funktionsfähige Software wird **häufig ausgeliefert** (eher Wochen als Monate)
- **funktionsfähige Software** ist das Maß für Fortschritt
- auch **späte Änderungen** der Anforderungen sind willkommen
- enge tägliche **Kooperation** von Geschäftsleuten und Entwicklern
- **Konversation** von Angesicht zu Angesicht ist die beste Kommunikationsform (gemeinsamer Ort)
- Projekte bestehen aus **Menschen**, denen man vertrauen sollte
- kontinuierliches Streben nach **technischer Exzellenz** und **gutem Design**
- **Einfachheit**
- **selbstorganisierte Teams**
- **kontinuierliche Anpassung** an sich ändernde Bedingungen

— <http://www.agilemanifesto.org/principles.html>

Varianten der agilen Entwicklung

- Agile Unified Process (AUP)
- Dynamic Systems Development Method (DSDM)
- Essential Unified Process (EssUP)
- **Extreme Programming (XP)**
- Feature Driven Development (FDD)
- Open Unified Process (OpenUP)
- **Scrum**

Extreme Programming (Beck 2000)

Extreme Programming (XP) ist eine agile Methode für

- kleinere bis größere Entwicklerteams (max. 10-15 Personen)
- Probleme mit vagen Anforderungen
- und Projekte, bei denen ein Kundenrepräsentant stets greifbar ist.

<http://www.extremeprogramming.org/>



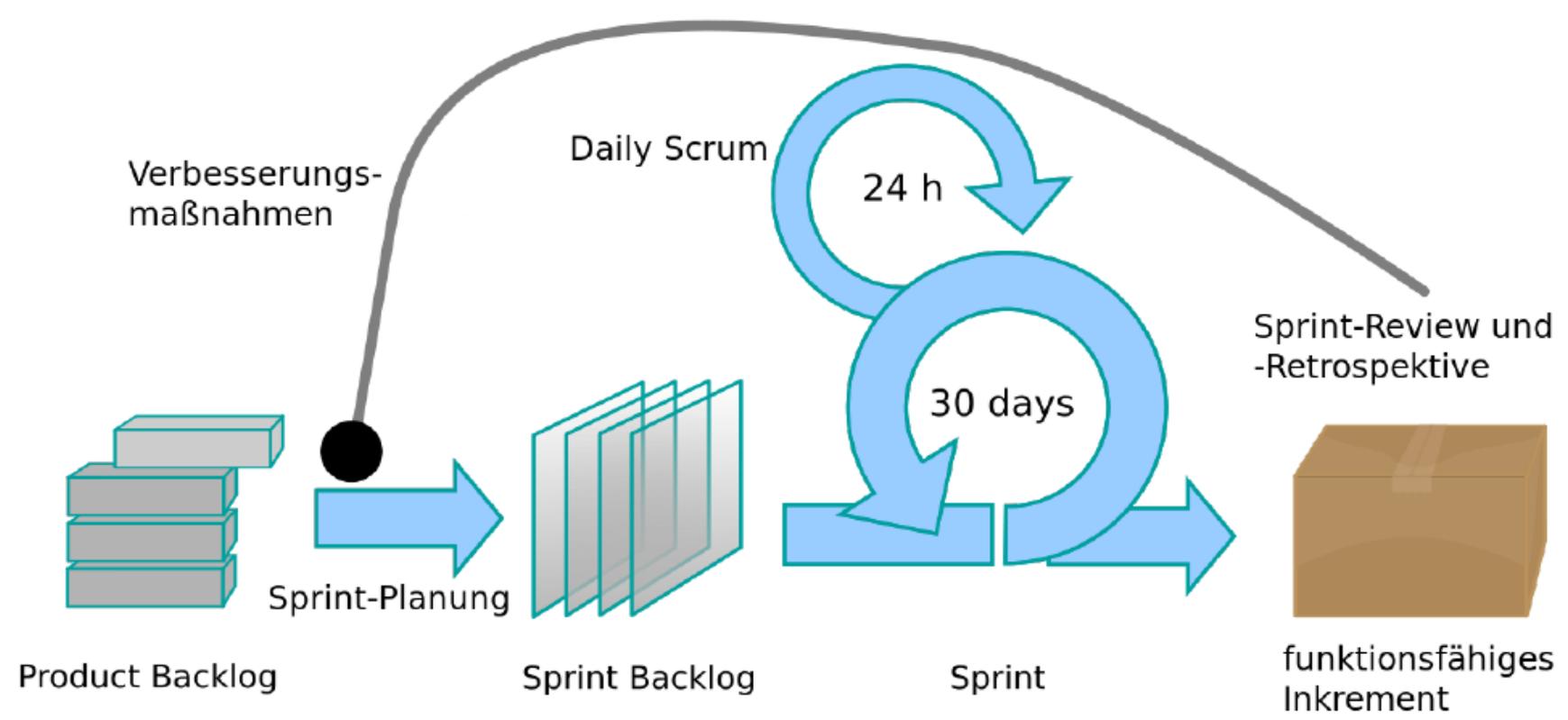
Extreme Programming (Beck 2000)

Anerkannte Prinzipien und Praktiken werden "extrem" umgesetzt:

- Code-Reviews → permanente Reviews durch Pair-Programming
- Testen → ständiges Testen: Unit-Tests sowie Akzeptanztests durch den Kunden/Benutzer
- klare Struktur → jeder verbessert sie kontinuierlich durch Refactoring
- Einfachheit → stets die einfachste Struktur wählen, die die aktuellen Anforderungen erfüllt
- Integration → permanente Integration auch mehrmals am Tag
- Validierung:
 - Kunde/Benutzer ist stets involviert bei der Planung neuer Iterationen und verantwortlich für Akzeptanztest
 - kurze Iterationen → Dauer in Minuten und Stunden, nicht Wochen, Tage, Jahre
- aber auch Auslassung anerkannter Prinzipien:
 - Dokumentation: mündliche Überlieferung, Tests und Quellcode
 - Planung: sehr begrenzter Horizont

- Kunde vor Ort
- eine Metapher statt einer Architekturbeschreibung
- 40-Stundenwoche
- Code ist kollektives Eigentum
- Kodierungsstandards





Product Owner:

- vertritt Endkundenbedürfnisse
- vereint Produkt- und Projektmanagementaufgaben
- fest integriert in Entwicklung

Aufgaben:

- Anforderungsbeschreibung und -management
- Releasemanagement und Return on Investment
- enge Zusammenarbeit mit dem Team
- Stakeholder-Management



Team:

- entscheidet selbst, wie viele Anforderungen in einem Inkrement umgesetzt werden
- legt Arbeitsschritte und Arbeitsorganisation selbst fest
- agiert autonom
- interdisziplinär besetzt
- Selbstorganisiert
- Klein
- Vollzeitmitglieder
- Arbeitsplätze in unmittelbarer Nähe



Scrum-Rollen (Pichler 2008)

Scrum-Master (vom Team bestimmt):

Aufgaben:

- etabliert Scrum
- unterstützt Team
- stellt Zusammenarbeit von Team und Product Owner sicher
- beseitigt Hindernisse
- verbessert Entwicklungspraktiken
- führt durch dienen

Fähigkeiten:

- Moderation
- Coaching
- Erfahrung in Softwareentwicklung



Scrum Product Backlog

Product Backlog enthält

- alle bekannten funktionalen und nichtfunktionalen Anforderungen
- weitere Arbeitsergebnisse (z.B. Aufsetzen der Test- und Entwicklungsumgebung)

... aber keine Aktivitäten!

Es wird vom **Product Owner** gepflegt

Prio	Thema	Beschreibung	Akzeptanz	Aufwand
1	Kalender	Administrator kann zentrale Kalenderdatenbank anlegen	Installationskript legt DB an	2
2	Kalender	Nutzer kann Termin eintragen	valider Termin wird eingetragen, invalider Termin wird zurückgewiesen	3
3	Kalender	Nutzer kann Termin löschen	gelöschter Termin ist entfernt; Löschung nur, wenn Rechte vorhanden	3
...



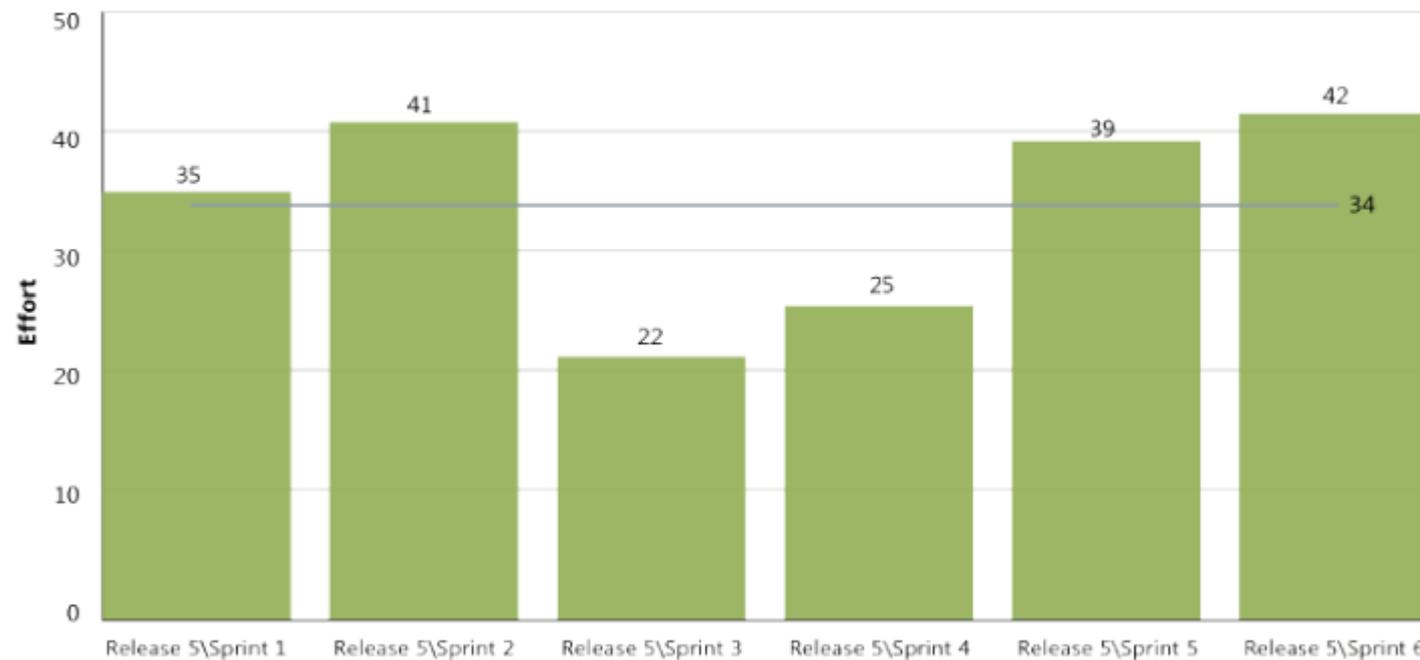
Punkteskala (Fibonacci-Reihe):

0	kein Aufwand
1	sehr kleiner Aufwand
2	kleiner Aufwand = $2 \times$ sehr kleiner Aufwand
3	mittlerer Aufwand = sehr kleiner + kleiner Aufwand
5	großer Aufwand = kleiner + mittlerer Aufwand
8	sehr großer Aufwand = mittlerer + großer Aufwand
13	riesiger Aufwand = großer + sehr großer Aufwand

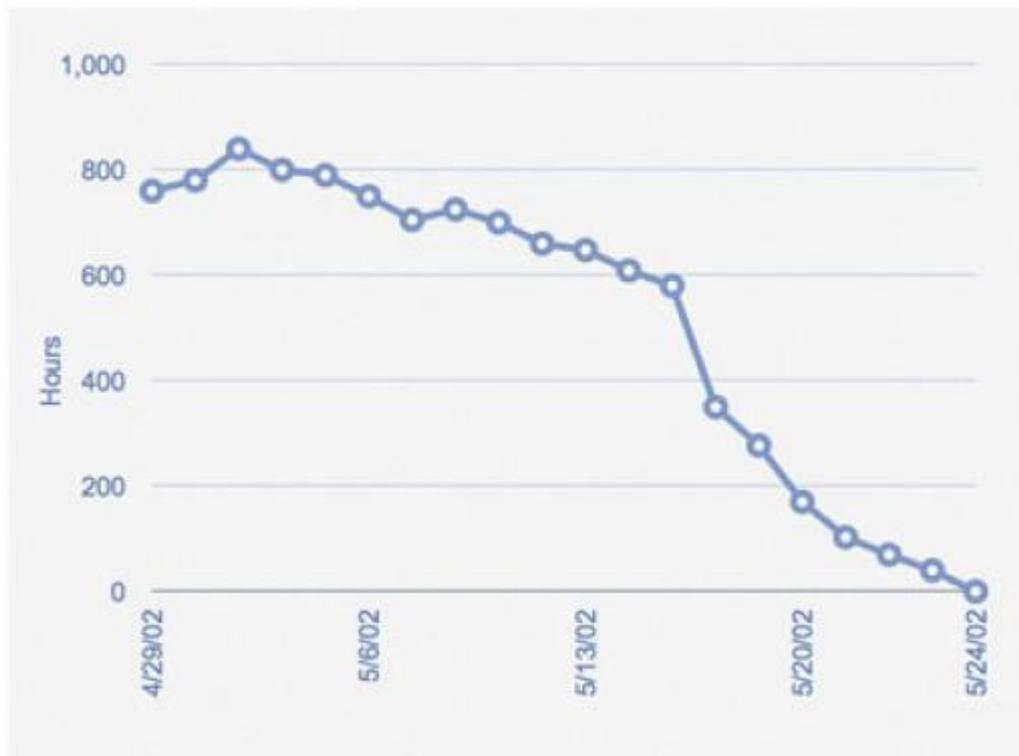
Scrum: Entwicklungsgeschwindigkeit

- Punkte werden im Projektverlauf auf Echtzeit abgebildet
- Entwicklungsgeschwindigkeit: $\text{Velocity} = \text{Punkte} / \text{Sprint}$

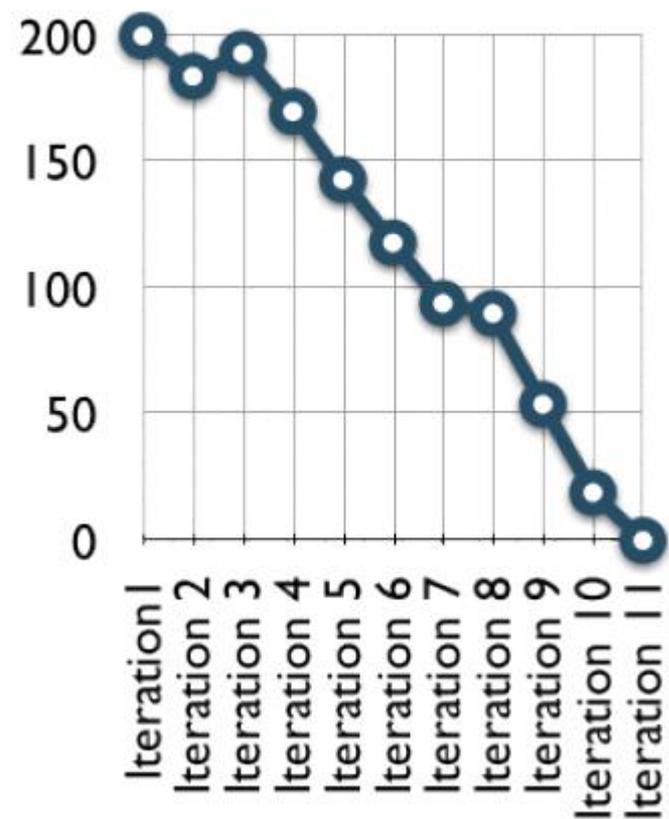
Velocity Chart



Sprint Burndown Chart



Release Burndown Chart



Agile versus weit voraus planende Prozessmodelle

Risiken agiler Methode:

- Skalierbarkeit, Kritikfähigkeit, Einfachheit des Entwurfs, Personalfuktuation, Personalfähigkeiten

Risiken weit voraus planender Prozessmodelle:

- Stabilität der Anforderungen, steter Wandel, Notwendigkeit schneller Resultate, Personalfähigkeiten

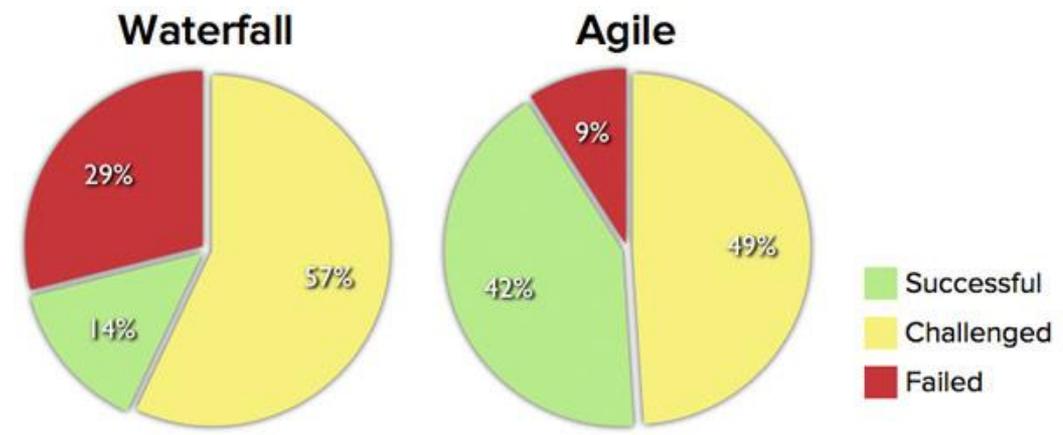
Generelle Risiken:

- Unsicherheiten bei der Technologie, unterschiedliche Interessengruppen, komplexe Systeme

— Boehm und Turner (2003)

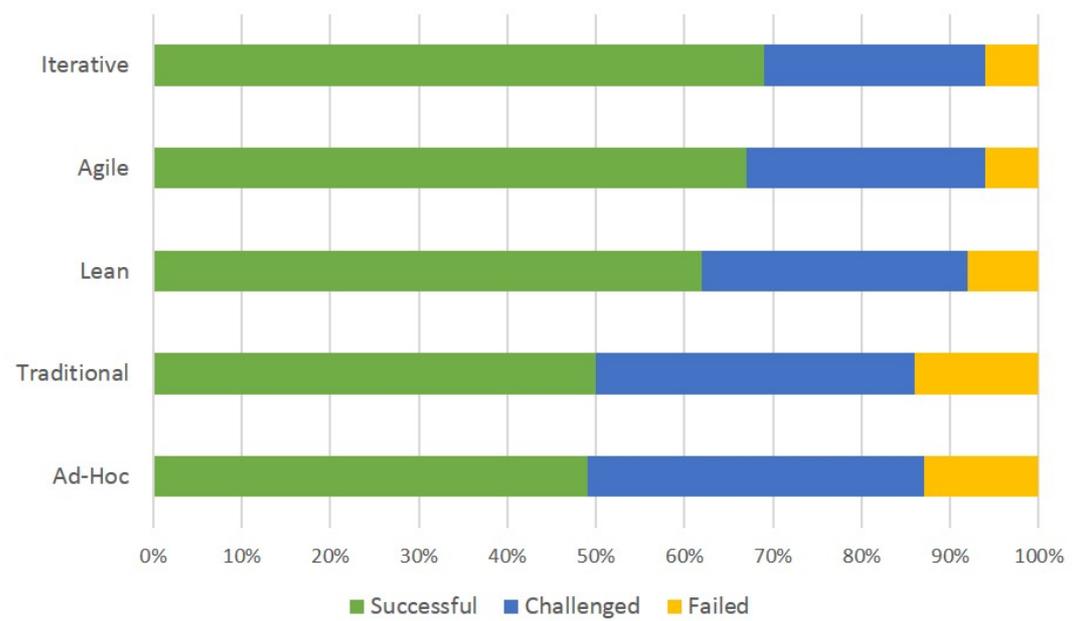


?



Source: The CHAOS Manifesto, The Standish Group, 2012.

AGILE PROJECTS ARE MORE SUCCESSFUL



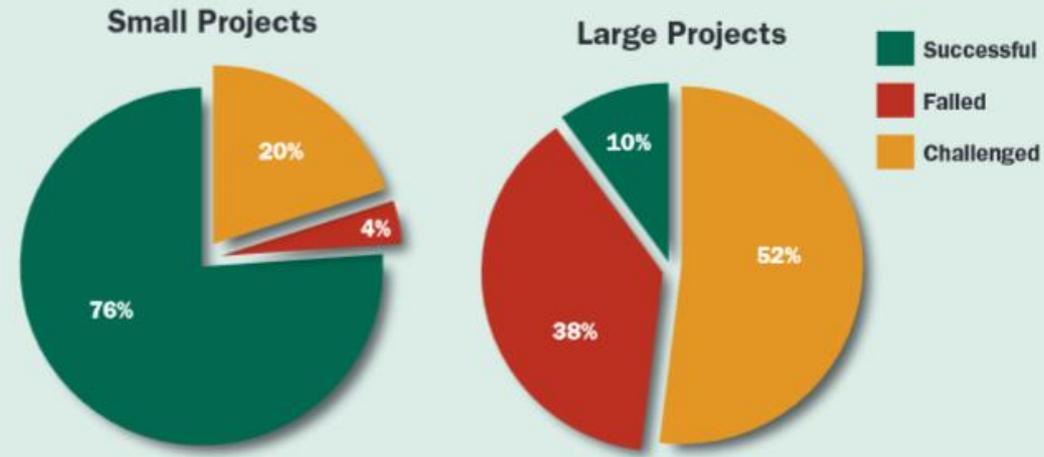
2011 IT Project Success Survey



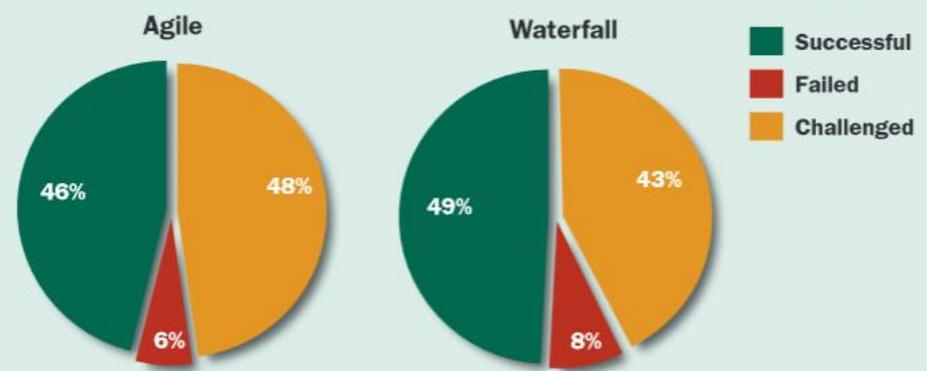
?

CHAOS RESOLUTION BY LARGE AND SMALL PROJECTS

Project resolution for the calendar year 2012 in the new CHAOS database. Small projects are defined as projects with less than \$1 million in labor content and large projects are considered projects with more than \$10 million in labor content.



AGILE V. WATERFALL SMALL PROJECTS



The charts show success rates for small software development projects using modern languages, methods, and tools, from 2003 to 2012.

- Cusumano, MacCormack, Kemerer, Crandall: "Software Development Worldwide: The State of the Practice", IEEE Software Nov. 2003

	India	Japan	US	Europe and other	Total
Number of projects	24	27	31	22	104
Architectural specifications (%)	83.3	70.4	54.8	72.7	69.2
Functional specifications (%)	95.8	92.6	74.2	81.8	85.6
Detailed designs (%)	100.0	85.2	32.3	68.2	69.2
Code generation (%)	62.5	40.7	51.6	54.5	51.9
Design reviews (%)	100.0	100.0	77.4	77.3	88.5
Code reviews (%)	95.8	74.1	71.0	81.8	79.8
Subcycles (%)	79.2	44.4	54.8	86.4	64.4
Beta testing (% > 1)	66.7	66.7	77.4	81.8	73.1
Pair testing (%)	54.2	44.4	35.5	31.8	41.3
Pair programming (%)	58.3	22.2	35.5	27.2	35.3
Daily builds (%)					
At the start	16.7	22.2	35.5	9.1	22.1
In the middle	12.5	25.9	29.0	27.3	24.0
At the end	29.2	37.0	35.5	40.9	35.6
Regression testing on each build (%)	91.7	96.3	71.0	77.3	83.7

Based on 104 projects surveyed

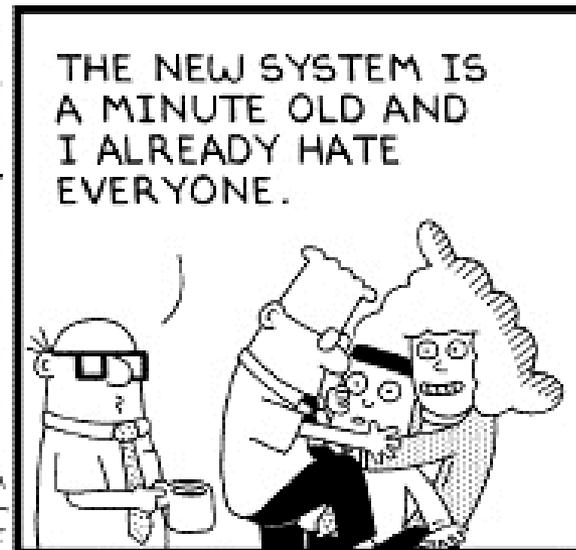
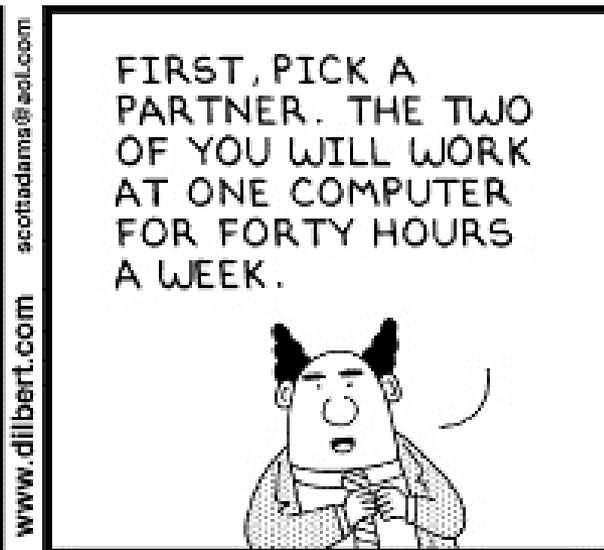
Wann welches Modell?

- Strikte, stark planende Modelle:
 - Wenn wohldefiniertes Resultat in definierter Zeit erreicht werden muss
 - Wenn sehr große Projektgruppen koordiniert werden müssen
 - Allgemein: Wenn auf Pläne und Dokumente zur Koordination nicht verzichtet werden kann

- Agile Modelle
 - Wenn hohe Unsicherheiten über die Anforderungen bestehen
 - Inhalt, Priorität
 - Wenn Änderungen von außen häufig sind
 - Anforderungen, Zeitplan, Budget, Qualitätsziele

- Eher stark planende Modelle:
 - Parallele Entwicklung von Hardware und Software (z.B. Auto)
 - Örtlich verteilte Entwicklung (z.B. in mehreren Firmen)

- Eher agile Modelle
 - Überhaupt kein Zeitplan (z.B. Hobbyprojekte)
 - Arbeit mit unausgereifter oder unbeherrschter Technologie
 -



www.dilbert.com scottadams@aol.com

1/9/03 © 2002 United Feature Syndicate, Inc.

Copyright © 2003 United Feature Syndicate, Inc.