

XP - eXtreme Programming

Autor: Dipl.-Math.
E. Engelhardt

Stand: 16. September 2001

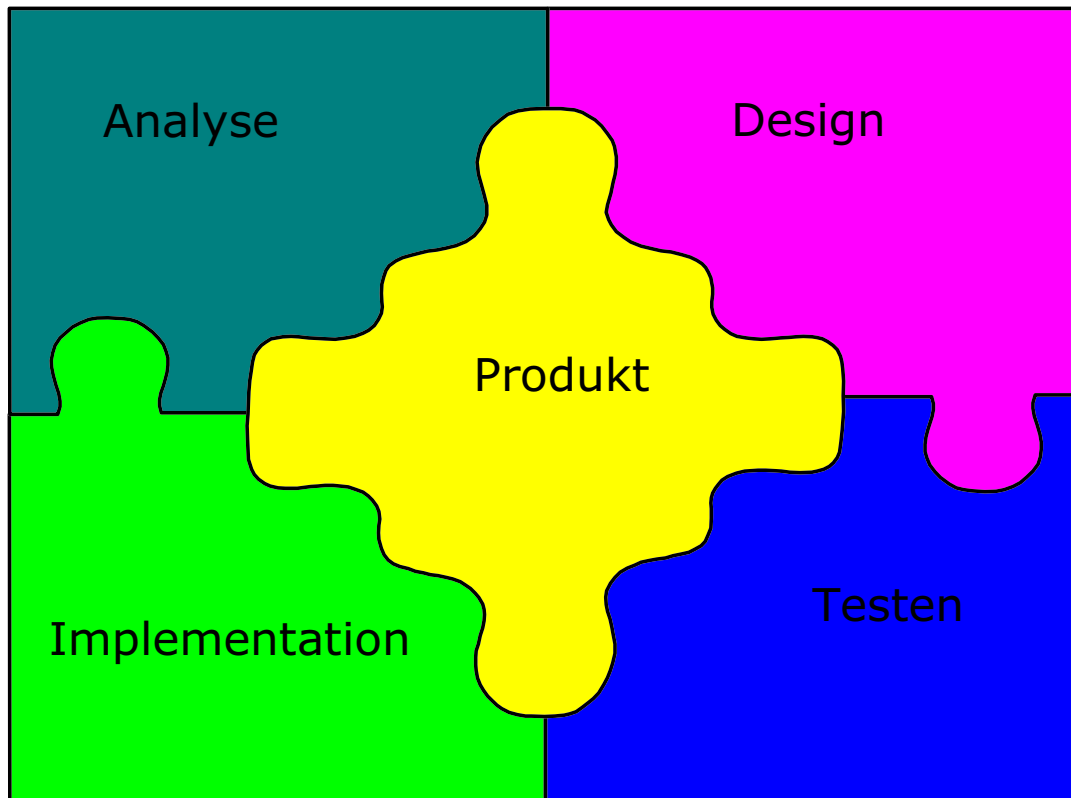
Softwarekrise



- ◆ Anforderungen des Marktes
 - Funktionstreue
 - Qualitätstreue
 - Termintreue
 - Kostentreue

- ◆ Erschwernisse während der Software-Erstellung
 - sich ändernde Produkthanforderungen
 - andere Hardware
 - veränderte Systemkomponenten
 - sich ändernde Werkzeuge (CASE-Tools)
 - hohe Portabilitätsforderungen (Software-Produkte haben längere Lebensdauer als Hardware)

==> Software-Krise

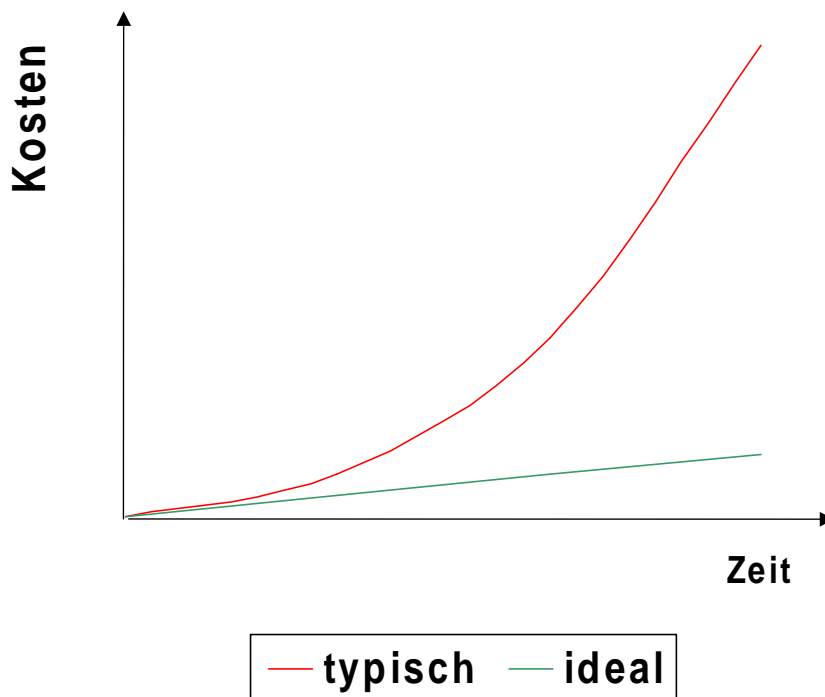


Risiken

- ◆ Geschäftliche Risiken
 - Anforderungen an geschäftliche Änderungen
 - Missverständnisse
 - Projekt-Abbruch
 - Schedules/Costs

- ◆ Technische Risiken
 - Qualität und Fehlerrate
 - Erweiterbarkeit
 - steigende Komplexität

- ◆ Team Risiken
 - „Master-Programmer“ gehen von Bord



Lineare Kostenänderungen

Keine großen Analysen
Kein großes Design
Keine Dokumentation



Implementation startet sofort, kein Overhead.
Sehr schnell eine erste Version (Time to market)



Feedback

eXtreme Programming

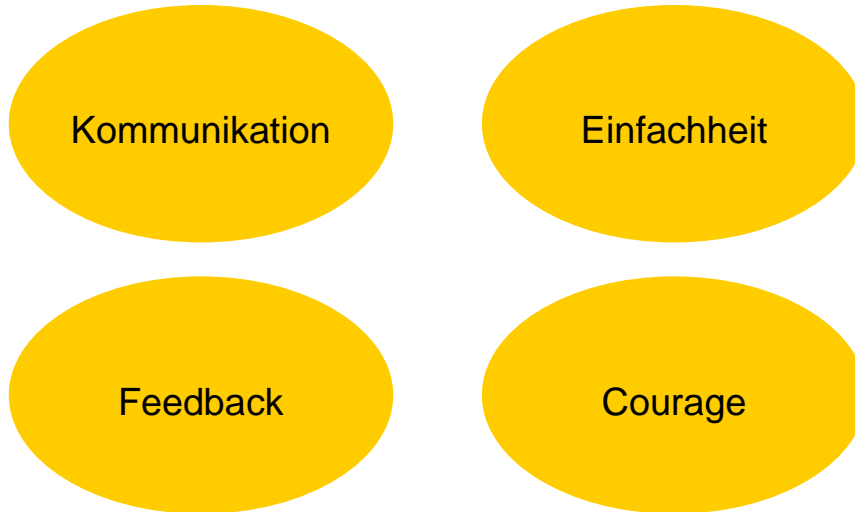
- ◆ Prozessmodell mit einer Reihe von Techniken (Praktiken)
 - entwickelt von Kent Beck, Ward Cunningham und Ron Jeffries
 - Praktiken sind nur in Gesamtheit sinnvoll
 - Praktiken stützen sich gegenseitig
 - konsequente (extreme) Bündelung der Praktiken und des Einsatzes

- ◆ Geeignet für kleine Softwareprojekte in kleinen Teams
 - maximal 10 -15 Mitarbeiter
 - Entwickler arbeiten räumlich zusammen
 - gleiche Arbeitszeiten (kein Schichtbetrieb)

- ◆ Nicht zu verwechseln mit chaotischer Vorgehensweise

- ◆ Kleine Releases
- ◆ Planungsspiel
- ◆ Automatisierte Tests
- ◆ Systemmetapher
- ◆ Einfacher Entwurf
- ◆ Refaktorisierung
- ◆ Programmieren in Paaren
- ◆ Gemeinsames Code-Eigentum
- ◆ Continuirliche Code-Integration
- ◆ 40-Stunden-Woche
- ◆ Kundenvertreter im Team
- ◆ Programmier-Richtlinien

Vier fundamentale konstante Werte, die den menschlichen und finanziellen Notwendigkeiten dienen.



Kleine Releases

- ◆ Hochgradig iterativer Entwicklungsprozess durch sich ständig ändernde Anforderungen
- ◆ Release besteht aus mehreren Iterationen
- ◆ Das erste Release ist ein bereits funktionierendes Kernsystem
- ◆ Kunde kann nach jeder Iteration Abweichungen von Wünschen feststellen und korrigieren

- ◆ Planung der Iterationen aufgrund von vorgegebenen Spielregeln
- ◆ Verfügbare Entwicklungs-Ressourcen und Kundenwünsche in Einklang bringen
- ◆ Anforderungen in Form von sogenannten Storys
 - Storys auf Karteikarten
 - Kunde bestimmt Priorität
- ◆ Aufwand für Iteration wird geschätzt
 - Schätzung durch Entwickler
 - Korrekturen können vorgenommen werden

- ◆ Test spielen zentrale Rolle bei XP, ohne Tests ist XP nicht realisierbar!
- ◆ Wissen über gewünschte Funktion wird in Testfällen niedergelegt
- ◆ Entwickler schreiben Tests für ihre Klassen
 - Unit-Tests
 - erst Test schreiben, dann implementieren
- ◆ Kunde entwickelt Testfälle
 - werden von Entwicklern umgesetzt
- ◆ Alle Tests automatisch ausführbar machen

- ◆ Suche nach einer einfachen Lösung, die momentane Anforderungen abdeckt
- ◆ zukünftige Erweiterungen werden zunächst nicht berücksichtigt (oft nie realisiert bzw. anders realisiert)
- ◆ Anforderungen können sich ändern
- ◆ Einbau von Funktionalität, die nie gefordert wurde und nicht gebraucht wird
- ◆ Sollten Änderungen von grundlegenden Strukturen nötig sein, wird refaktoriert

- ◆ Vereinfachung des Entwurfs unter Beibehaltung der Semantik
 - Verständlichkeit und Änderbarkeit des Codes soll verbessert werden
 - Semantik ist in Tests festgelegt und muss geprüft werden
- ◆ Dokumentation veraltet sehr schnell und wird oft nicht benutzt, deshalb wird in XP die Dokumentation auf Code beschränkt
 - Code muss selbsterklärend sein
 - Das wird erreicht durch Struktur und Namensgebung

- ◆ Paarbindung ist nicht fest
 - Wissen über das System verbreitet sich über alle Entwickler
 - ein Entwickler kann ausfallen ohne katastrophale Folgen
- ◆ Entwicklungsaufwand steigt nur geringfügig, aber Code wird besser verständlich und enthält weniger Fehler
- ◆ Einer programmiert und einer kontrolliert
 - Rollen können getauscht werden

40-Stunden-Woche

Kontinuierliche Code-Integration

- ◆ Entwickler oder geänderter Code wird sofort in Code-Basis integriert
 - ein Entwicklerpaar spielt Änderungen ein und testet sie
 - Treten Fehler auf, so wird Änderung sofort zurück gegeben
- ◆ Immer ein lauffähiges System verfügbar!

Gemeinsames Code-Eigentum

- ◆ Code gehört allen Entwicklern imProjekt
- ◆ Jedes Entwicklerpaar kann jederzeit überall Änderungen vornehmen

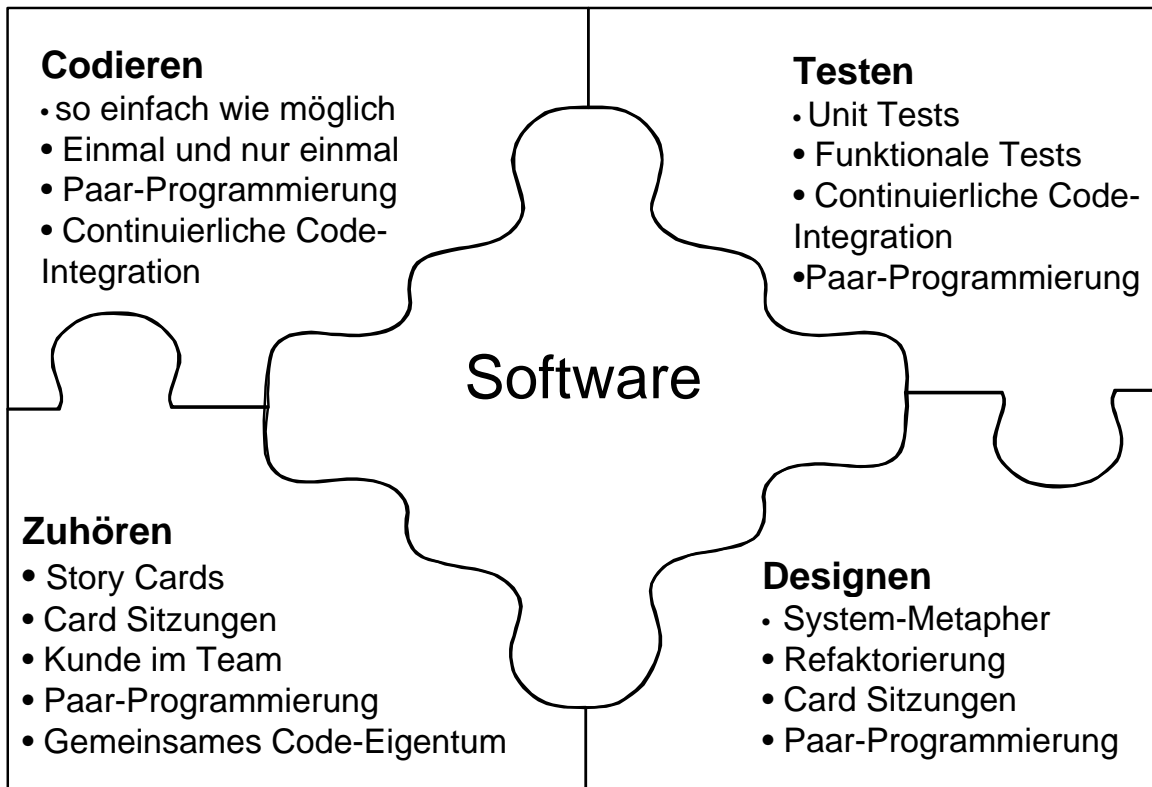
- ◆ Keine Spezifikation vorhanden ==> viele Rückfragen an Kunden
- ◆ Vertreter des Kunden muss permanent verfügbar sein
- ◆ Ein künftiger Anwender des Systems
- ◆ Kundenvertreter entwickelt Testfälle für die funktionalen Tests
 - werden durch Entwicklerpaar umgesetzt

Programmier-Richtlinien

- ◆ Richtlinien werden vor Beginn der Arbeiten vereinbart
 - Code-Conventions
 - selbsterklärende Namensgebungen (deutsch, englisch, ...)
 - Struktur (Einrückungen, Klammern, ...)
 - Kommentare
- ◆ Alle Entwickler halten sich an vereinbarte Richtlinien
 - leicht Wechsel möglich (Programme sehen gleich aus)

Systemmetapher

- ◆ Welchem Bild entspricht das Programm (Architektur)?
 - Schreibtisch
 - Ameisenhaufen
 - Produktionsstraße



Literaturangaben

- [1] Kent Beck: „*eXtreme Programming explained*“, ADDISON-WESLEY Verlag, ISBN 3-8273-1709-6
- [2] Ron Jeffries, Ann Anderson, Chet Hendrickson: *Extreme Programming installed*, ADDISON-WESLEY Verlag, ISBN 3-8273-1818-1
- [3] *Extremes Programmieren*, Informatik Spektrum, Springer-Verlag Band 23, Heft 2, April 2000
- [4] *Internet HTML-Seiten*, www.Xprogramming.com
www.extremeprogramming.org
c2.com/cgi/wiki?ExtremeProgrammingRoadMap