

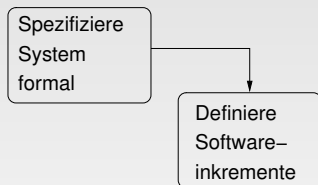
# Cleanroom Development



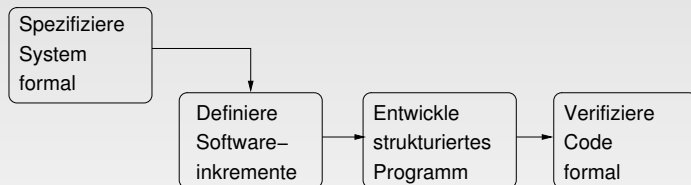
# Cleanroom Development (Mills u. a. 1987)

Spezifiziere  
System  
formal

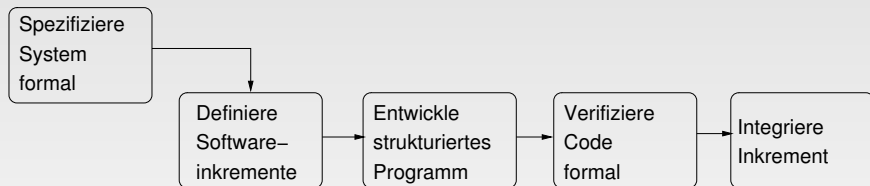
# Cleanroom Development (Mills u. a. 1987)



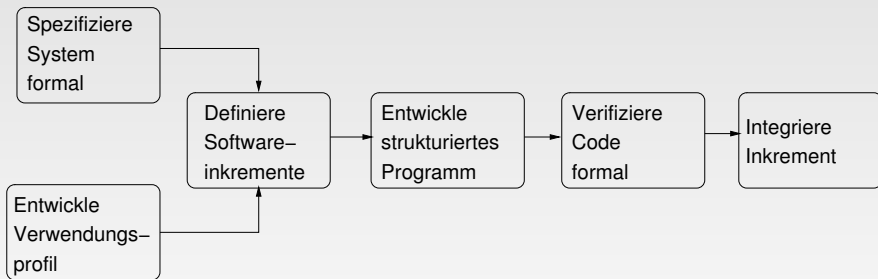
# Cleanroom Development (Mills u. a. 1987)



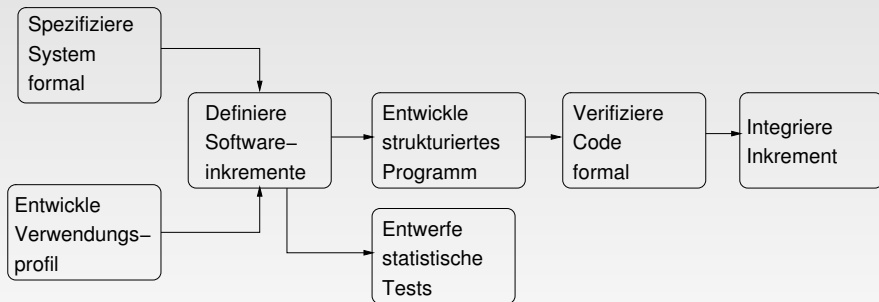
# Cleanroom Development (Mills u. a. 1987)



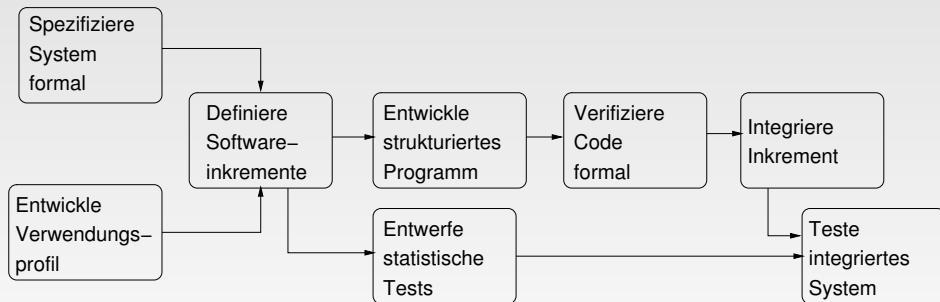
# Cleanroom Development (Mills u. a. 1987)



# Cleanroom Development (Mills u. a. 1987)

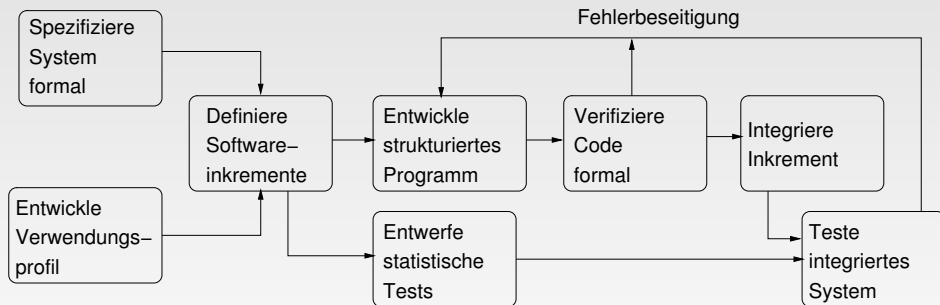


# Cleanroom Development (Mills u. a. 1987)





# Cleanroom Development (Mills u. a. 1987)



## Schlüsselstrategien

- Formale Spezifikation
- Inkrementelle Entwicklung
- Strukturierte Programmierung
- Statische Verifikation
- Statistisches Testen
  - basiert auf Verwendungsprofilen (die Verwendungsweise der Software in der Praxis)
  - die häufigsten (und kritischsten) Verwendungsarten werden verstärkt getestet

## Gruppen:

- Spezifikationsteam:
  - verantwortlich für Entwicklung und Wartung der Systemspezifikation
  - erstellt kundenorientierte und formale Spezifikation
- Entwicklungsteam:
  - verantwortlich für Entwicklung und Verifikation der Software
  - Software wird nicht ausgeführt hierzu!
  - verwendet Code-Inspektion ergänzt durch Korrektheitsüberlegungen (nicht streng formal)
- Zertifizierungsteam:
  - verantwortlich für statistische Tests

Erfahrungen Cobb und Mills (1990):

- weniger Fehler als bei traditioneller Entwicklung
- bei vergleichbaren Kosten

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

# 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

# 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



# 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

# 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

# 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

# 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

# Agile versus weit vorausplanende Prozessmodelle (Boehm und Turner 2003)

Risiken agiler Methode:

- Skalierbarkeit, Kritikalität, Einfachheit des Entwurfs, Personalfluktuatation, Personalfähigkeiten

# Agile versus weit vorausplanende Prozessmodelle (Boehm und Turner 2003)

Risiken agiler Methode:

- Skalierbarkeit, Kritikalität, Einfachheit des Entwurfs, Personalfluktuations, Personalfähigkeiten

Risiken weit vorausplanender Prozessmodelle:

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

# Agile versus weit vorausplanende Prozessmodelle (Boehm und Turner 2003)

## Risiken agiler Methode:

- Skalierbarkeit, Kritikalität, Einfachheit des Entwurfs, Personalfluktuations, Personalfähigkeiten

## Risiken weit vorausplanender Prozessmodelle:

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

## Generelle Risiken:

- Unsicherheiten bei der Technologie, unterschiedliche Interessengruppen, komplexe Systeme



# Capability Maturity Model

- Entwickelt vom SEI 1985-91 für DoD
- Beschreibt Stufen der Prozessreife
- Maßstab/Leitfaden für Verbesserungen
- Idee: besserer Prozess → besseres Produkt
- 5 Stufen (CMM Level 1-5)
- Definiert Schlüsselbereiche (Key Process Areas)
- Steigende Transparenz des Prozesses