Agiles Projektmanagement

"Keep It Simple, Stupid!"

Dr. Michael Heitland

Juli 2006

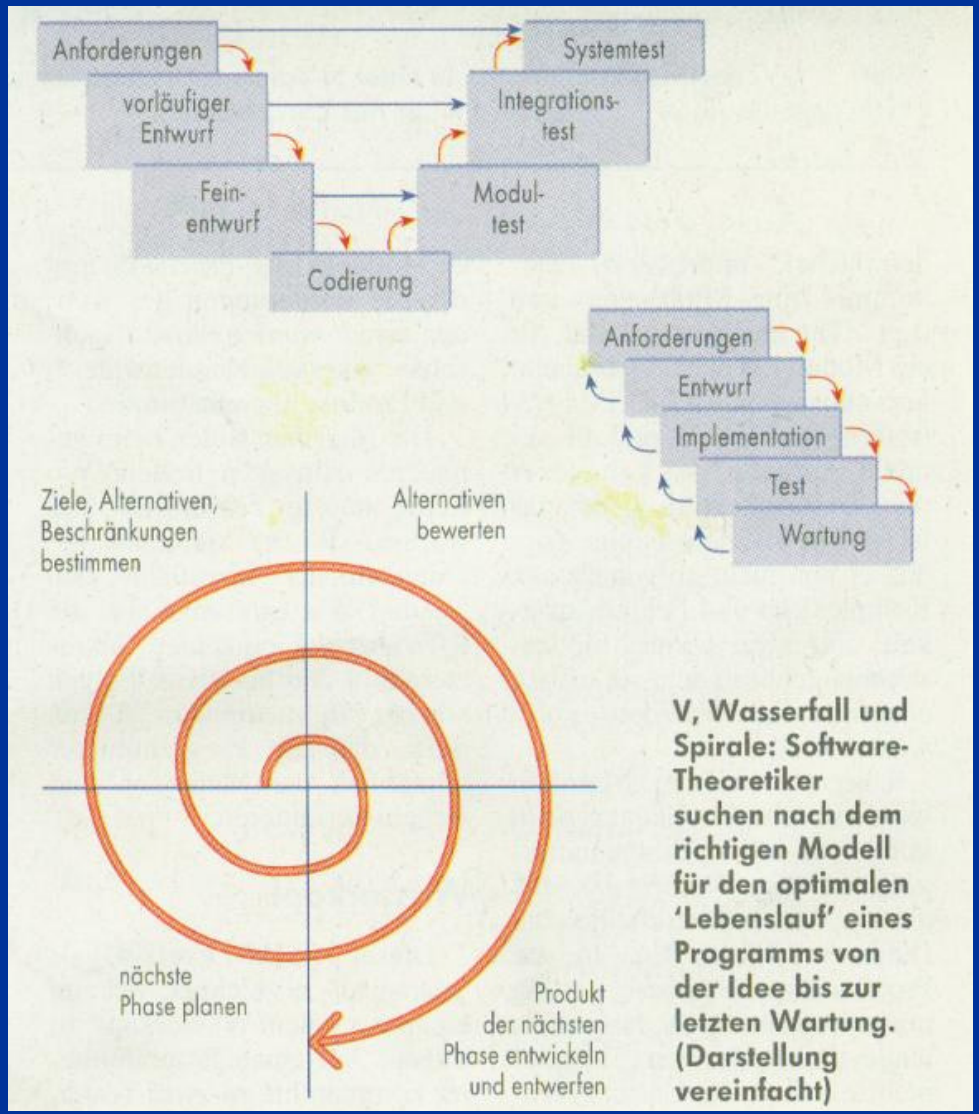
Gliederung

- Ausgangssituation
- Faktoren für erfolgreiche Projekte
- Qualitätsstandards
- Stellgrößen
- Agile Softwareentwicklung
- Extreme Programming
- Feature Based Programming
- Feature Driven Development
- Continuous Integration
- Scrum
- Joel Test - 12 Punkte zur Qualitätssicherung
- Kommunikation
- Fragen und Diskussion

Ausgangssituation

Seit den sechziger Jahren wird Software extrem komplex: Softwarekrise

1968 Nato-Tagung in Garmisch-Patenkirchen: Ausweg SW-Engineering



V, Wasserfall und Spirale: Software-Theoretiker suchen nach dem richtigen Modell für den optimalen 'Lebenslauf' eines Programms von der Idee bis zur letzten Wartung. (Darstellung vereinfacht)

c't 1998/19

Situation heute - alles gelöst?

- 18 % der IT-Projekte scheitern vollständig
- 53 % verlassen Rahmen für Kosten (+45 %), Zeit (+63 %), Fkt.umfang (- 33 %)
- 29 % werden pünktlich und erfolgreich beendet (1994: 16 %)

(http://standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf)

Faktoren für erfolgreiche Projekte

The CHAOS Ten

Executive Support	18
User Involvement	16
Experienced Project Manager	14
Clear Business Objectives	12
Minimized Scope	10
Standard Software Infrastructure	8
Firm Basic Requirements	6
Formal Methodology	6
Reliable Estimates	5
Other	5

Each factor has been weighted according to its influence on a project's success. The more points, the lower the project risk.

Qualitätsstandards

ISO-9000ff. (ISO 9000.3:2004, ISO 9001, Europa, Ziel: wiederholbare Prozesse):

- Kundenorientierung
- Führung
- Einbeziehung der Personen
- Prozessorientierter Ansatz
- Systemorientierter Managementansatz
- Kontinuierliche Verbesserung
- Sachbezogener Entscheidungsfindungsansatz
- Lieferantenbeziehung zu gegenseitigem Nutzen

CMM (Capability Maturity Model, Carnegie Mellon University, DoD USA, → CMMI):

- Initial (59 %): kein Prozess für SW-Entwicklung ist definiert
- Repeatable (24 %): Prozessregeln für alle eingeführt
- Defined (15 %): Prozessregeln werden stets überwacht
- Managed (2 %): quantitative Ziele vorgegeben und gemessen
- Optimized (0.6 %): Prozess wird fortwährend optimiert

Aufstieg um einen Level dauert mindestens 2 Jahre,
wird die Domäne gewechselt, beginnt man in der Regel wieder unten.

Stellgrößen

- Zeit
- Ressourcen
- Funktionsumfang
- Qualität

Oft Zeit, Ressourcen, Funktionsumfang fest vorgegeben, Qualität angepasst.

Im laufenden Projekt Ressourcen zu erhöhen hilft oft nicht:
neue Mitarbeiter müssen eingearbeitet werden,
Kommunikationsbedarf steigt (*Brook's Law*)

Besser: Zeit, (Ressourcen,) Qualität vorgeben, Funktionsumfang anpassen.

SW-Entwicklung ist sehr teuer:

- unter Berücksichtigung von Einarbeitung, Design, Fortbildung, Testen etc. schreibt ein Entwickler ca. 50 Zeilen / Tag,
  eine Zeile Quellcode kostet damit ca. 5 Euro (Sicherheitsbereich > 100,-)
- Kostensenkung nur mit effizientem Prozess

Agile Softwareentwicklung - 4 Werte

"Wir entdecken Wege, Software besser zu entwickeln, indem wir es selbst tun und anderen dabei helfen, es zu tun.

Durch diese Arbeit haben wir Folgendes zu schätzen gelernt:

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge
- Funktionierende Software ist wichtiger als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen
- Offenheit für unbekannte Änderungen ist wichtiger als eine Planverfolgung

Wir schätzen auf Grund unserer Erfahrungen die Punkte auf der rechten Seite, aber wir bewerten die Punkte auf der linken Seite höher."

Quelle: `http://www.agilemanifesto.org/`

Agile Softwareentwicklung - 12 Prinzipien

- Höchste Priorität:
  Kundenzufriedenheit durch frühzeitige und kontinuierliche Auslieferung exzellenter Software
- begrüße Änderungsanforderungen auch in späten Entwicklungsphasen - agile Prozesse nutzen Änderungsanforderungen als Wettbewerbsvorteil
- häufige Auslieferung ermöglicht frühe Kundenrückmeldung
- tägliche Zusammenarbeit zwischen Kunde und Entwickler
- vertraue motivierten Mitarbeitern, gib nötige Umgebung und Unterstützung
- direkte Kommunikation von Angesicht zu Angesicht
- funktionierende Software ist das Maß für den Fortschritt
- Entwicklung in gleichbleibendem Tempo, vermeide Überstunden
- verbessere ständig Design und technische Qualität
- Einfachheit
- selbstorganisierende Teams, so wenig zentrale Vorgabe wie möglich
- regelmäßige Selbstreflektion der Teams

Extreme Programming - Grundlagen

XP wurde 1996 von Kent Beck, Ward Cunningham und Ron Jeffries entwickelt:
"Extreme Programming" und "Extreme Programming Explained", Adison Wesley

4 Werte:

- Kommunikation
- Einfachheit
- Feedback
- Mut

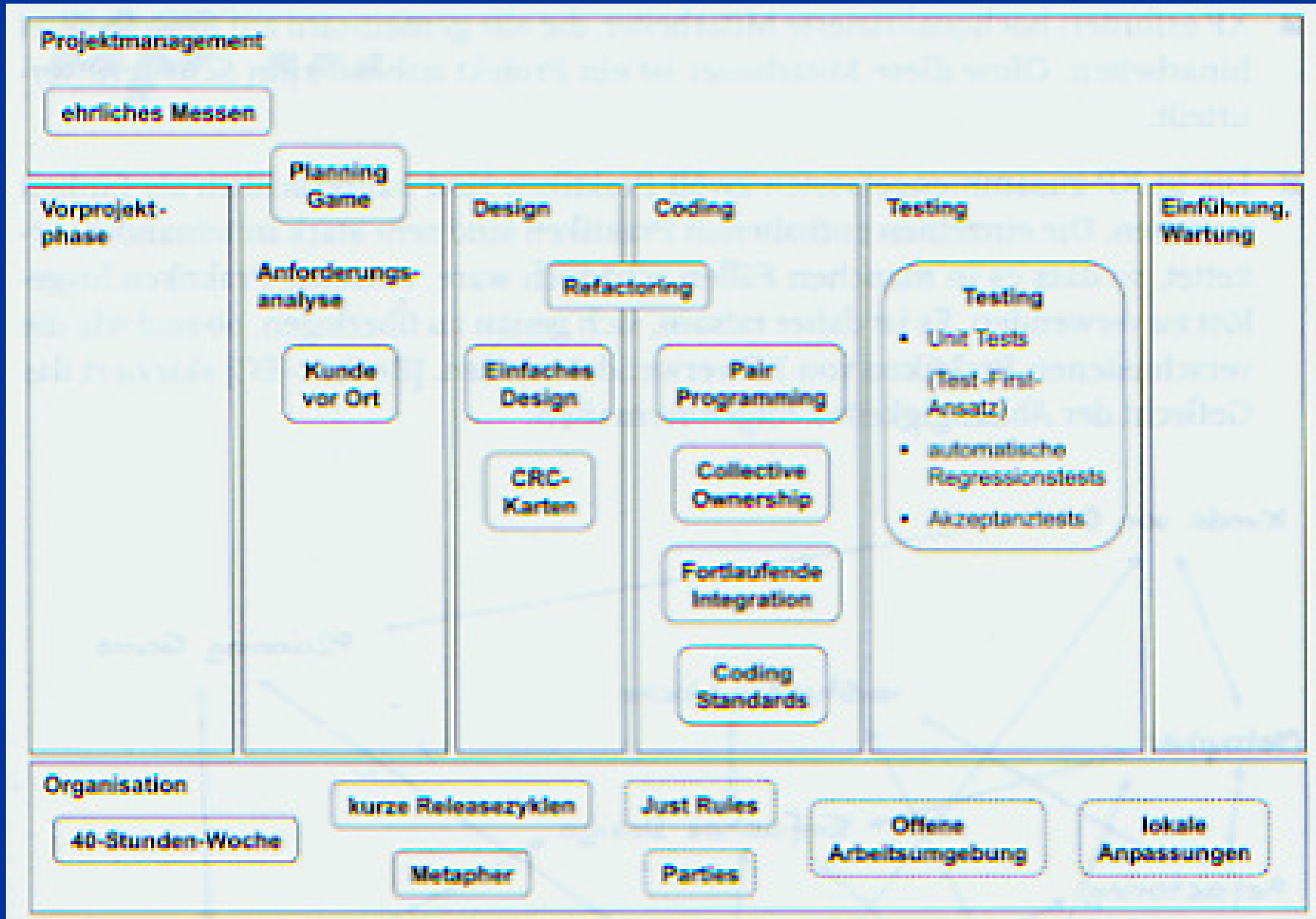
XP ist nur geeignet für folgende Projekte:

- kleine Teams bis max. 10 Personen
- es muss bereits einen Kunden geben, dieser muss vor Ort verfügbar sein
- hochqualifizierte Mitarbeiter
- alle 18 Praktiken (s. folgende Folie) müssen zusammen angewendet werden

Literatur:

Carsten Dogs, Timo Klimmer: Agile Softwareentwicklung kompakt

Extreme Programming - Praktiken



Feature Based Programming - Grundlagen

- schlankes SW-Prozessmodell, entwickelt von Stefan Richter
- Bottum-Up Prozess
- große Projekte in Teilprojekte zerlegen, Teilprojekte getrennt verwalten
- Projekt in Features zerlegen: handhabbar für Kunden, Entwickler, Manager
- Ziele: Termin- und Budgettreue
- Kommunikation: Projektmanager ↔ Kunde, Entwickler → Kunde
- Releasetermin stets einhalten (ggf. Feature ins nächste Release verschieben, durch Min-Max-Planung Gesamtzeitplan erreichbar)

Feature:

- kleine für den Kunden geschäftswertige Funktion
- vollständiger Durchstich (von GUI bis zum Persistencelayer inkl. Test)
- als Use Case beschreibbar
- max. 5 Mann-Tage (erhöht Motivation für Kunde + Entwickler)

Analyse + Konzeption:

- Projektmanager legt mit Kunde + Entwickler Feature fest
- Projektmanger muss selbst erfahrener Entwickler sein
- Min-Max-Aufwand wird durch ausführenden Entwickler geschätzt

Literatur:

- Stefan Richter: Feature Based Programming, ISBN 3827320771
- dotnetpro TV Folge 12

Feature Based Programming - Featureliste

Nr.	Kurzbeschreibung	Aufwand	Verantw.	Prio	Status	Release
13	Kunde anlegen	2-3 MT	Meier	1	impl.	1.2
14	Import aus CSV Datei	3-5 MT	Müller	3	offen	1.3

Featureliste (Erstellung durch Kunde + Entwickler, gemeinsame Sprache):

- Aufwand: maximal 5 Manntage (umfasst auch das Testen)
- Verantwortlicher: genau 1 Person, der Entwickler
- Featureliste ist Vertragsgegenstand
- Zeiterfassung pro Feature
- bei Bedarf direkte Kommunikation Entwickler / Kunde (keine Stille Post)
- zusätzlich entsteht bei Realisierung ausführliche Feature-Dokumentation

Maßnahmeplan:

- nicht Feature-bezogene Aufgaben (Server aufsetzen, Fortbildung etc.)

Releaseplan:

- Zuordnung von Features zu Releases bis Projektende
- Liefertermin + Budget fest, Featurezuordnung zu Release variabel
- Liefertermin wird stets gehalten, ggf. Feature verschieben
- alle 2 – 4 Wochen Auslieferung (garantiert schnelles Kunden-Feedback)

Feature Driven Development - Grundlagen

Grundlagen:

- Coad / De Luca 97: retten Projekt Bank of Singapure (50 Mitarb., 15 Mon.)
- lange Projektlaufzeiten (z.B. > 2 Jahre) sind gefährlich (Technikwandel!)
- Übersichtsmodell als Startpunkt
- Einsatz von OOP + UML (primär Klassen- und Sequenzdiagramme)
- detaillierte Spezifikation oft nicht praxisgerecht

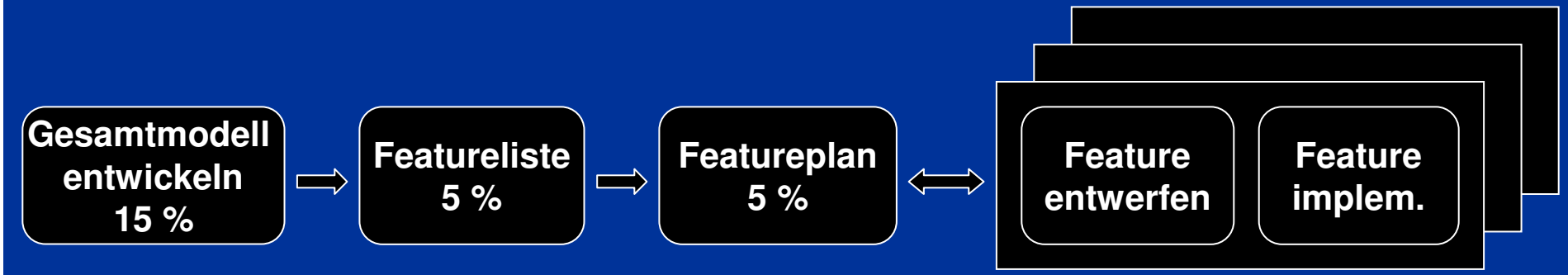
Features:

- Featureliste priorisieren: *must have, nice to have, add as we can, future*
- Releaseplan der priorisierten Features erstellen
- pro Feature maximal 2 Wochen, komplexere Features aufteilen
- Zerlegung: Gesamtmodell, Teilprojekte, Feature Sets, Features
- jeweils 1 Verantwortlicher pro Feature Set, Feature, Klasse
- am Feature arbeiten der Feature- und alle Klassen-Verantwortlichen

Umgebung:

- Unit-Tests durch Klassen-Verantwortlichen
- wöchentliche Statusmeetings
- CASE-Tools einsetzen zur Transformation UML ↔ Code
- FDD geeignet für bestehende Projekte mit vorhandenem Domänenwissen
- FDD ungeeignet für Projekte mit langer Planungs- / Klärungsphase

Feature Driven Development - Prozess



Überblick Fachgebiet	1 %
Entwurf	40 %
Review Entwurf	3 %
Implementieren + Testen	45 %
Review Impl.+Testen	10 %
Freigabe	1 %

Feature Driven Development - Diskussion

ETVX-Template:

- Entry: Vorbedingungen, damit Prozess starten kann
- Tasks: Aufgabenliste des Prozesses
- Verification: wie kann Aufgabenerfüllung getestet werden
- Exit: Nachbedingung + Output des Prozesses

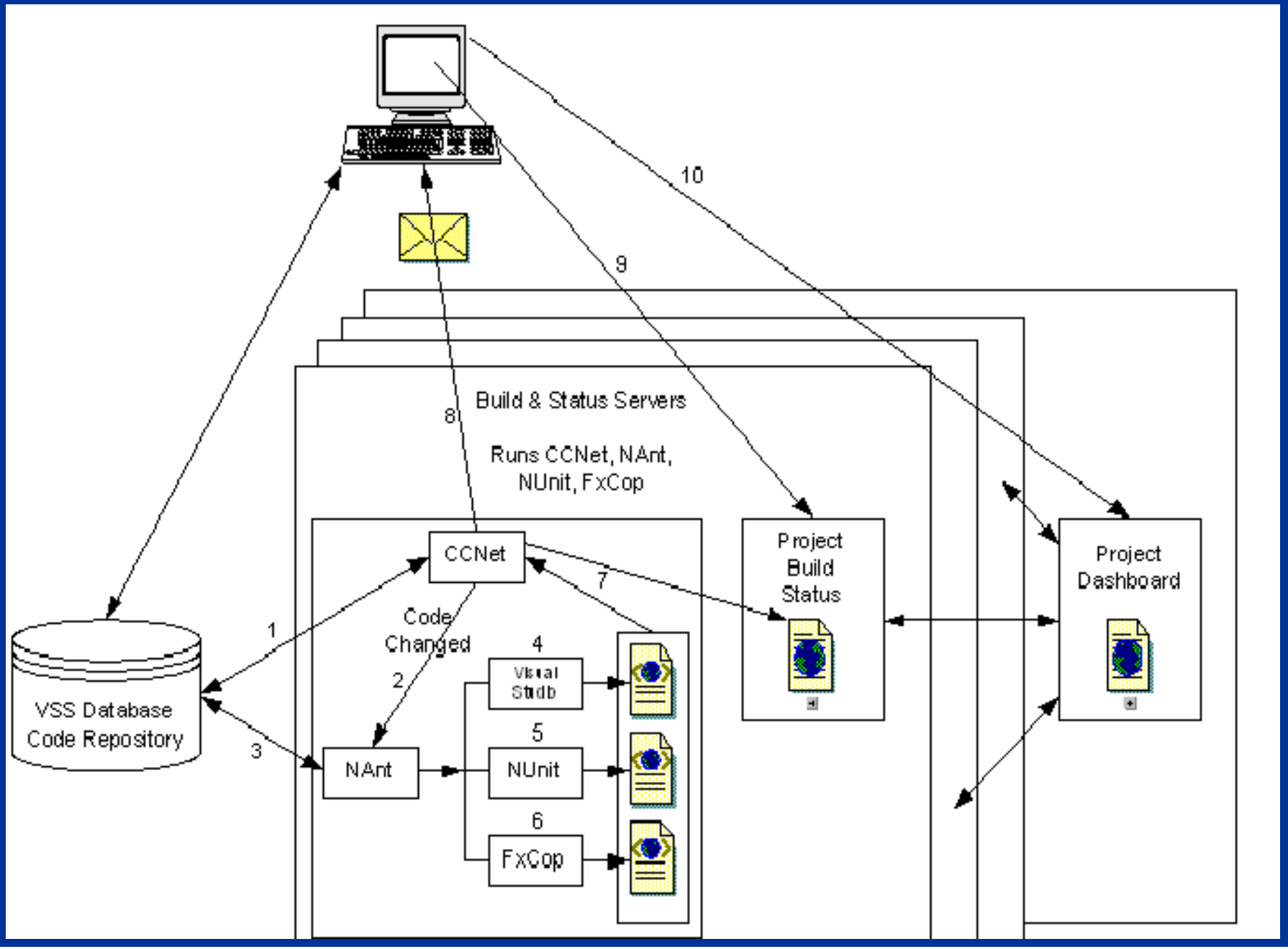
Agiler Prozess?

+ kurze Releasezyklen, regelmäßige Builds, Versionskontrolle
+ starke Kundeneinbindung
- Hierarchie: Programmmanager, Chefarchitekt, Chefentwickler
- Individual Code Ownership
- umfangreiche Änderungen im Modell nicht vorgesehen

Literatur:

- Coad / Lefebvre / De Luca:
  Java Modeling in Color with UML – Enterprise Components and Process
- Palmer / Felsing:
  A Practical Guide to the Feature-Driven Development
- Daniel Gyger: Feature-Driven Development,
  Universität Zürich, Seminararbeit WS 2003 / 2004

Continuous Integration – Cruise Control .NET Server



Scrum

- Schwaber / Sutherland 1995 (Scrum = Rugby-Spielzug), Einsatz bei MS, Nokia …
- Annahme: SW-Produktion zu komplex, um Arbeitsschritte genau zu planen
- wenig Vorgaben, Entwickler organisieren s. selbst und wählen Methoden
- fortlaufende Prozessverbesserung durch das Team (max. 8 Mitarbeiter)
- Quellen. www.scrumalliance.com, www.controlchaos.com

4 Rollen:

- *Management*: *grobe Zielvorgabe,* bleibt im Hintergrund, beseitigt Hindernisse
- *Product Owner*: Zielfestlegung, Budgetplanung, Prioritäten, Releasemanager, QS
- *Entwickler*: Aufwandsabschätzung, Commitment, Design, Impl., Testen, Selbstorg.
- *Scrum Master*: Prozessbeobachtung, Transparenz, Arbeitsbeding., Kunde, Manag.

Methoden:

- jedes Release entsteht innerhalb eines *Sprints* von maximal 30 Tagen
- Product Backlog: Liste der Kundenwünsche mit Kundenprios + technische Arbeiten
- Sprint Backlog: Auswahl des Product Backlogs für nächsten Sprint durch Kunde
- Sprint Backlog darf nicht mehr geändert werden, Änderungen → nächster Sprint
- nach jedem Sprint Review durch Team und Kunde
- jeden Mittag 15 minütiges Scrum Meeting im Stehen *(Standup Meeting):*
  + was habe ich geschafft?
  + was mache ich bis zum nächsten Scrum Meeting?
  + wo gibt es aktuell Probleme? (Klärung nach Meeting in kleiner Runde)

Joel Test - 12 Punkte zur Qualitätssicherung

Joel Test ist eine praxistaugliche Anpassung von SEMA (Software Engineering Measurement and Analysis):

- Source Control System (z.B. SourceSafe oder Subversion)
- automatisiertes Build durch ein einziges Skript
- nächtliches Build, nächtliche Unit-Tests mit autom. Benachrichtigung
- Bug Datenbank (z.B. Bugzilla)
- alle Bugs fixen bevor neuer Code geschrieben wird
- Terminplanung stets aktualisieren
- für jedes Projekt vorher Spezifikation erstellen
- ruhige Arbeitsräume (im Extremfall: telefonloser Raum)
- nur die besten Tools und ausreichend Hardware einsetzen (2 Monitore)
- ausreichend viele Tester (Verhältnis mindestens 1 Tester auf 2 Entwickler)
- bei Neueinstellungen: können Bewerber guten Code schreiben?
- Akzeptanztests durch mindestens 5 Personen

Kommunikation

NASA-Studie: Probleme liegen primär bei den Anforderungen, sekundär beim Design und erst tertiär bei der Codierung

Kommunikationswege auswählen:

- Kick Off Meeting
- tägliches Standup Meeting
- Review Meeting
- Email
- Intranet / Sharepoint
- Wiki
- Tracking System für Bugs / Features
- Zeiterfassung
- Elektronische Foren (*News Groups*)
- Schwarzes Brett (konventionell oder elektronisch)

Allerdings: zu viele Kommunikationswege überwachen stresst und kostet viel Zeit!

Regelmäßige Freiraumzeiten:

- Reflexion über eigene Arbeit
- Einarbeitung in neue Technologien

Schwieriges Ziel: richtige Mischung aus Effizienz und Kreativität erreichen

- Effizienz: kontinuierliche Prozessbeschleunigung, Kontrolle, Fortbildung
- Kreativität: Freiraum, Vertrauen, projektfremde Activities, keine Überstunden

Fragen und Diskussion

Dr. Michael Heitland

.NET Consulting & Training

www.dotnet-it.de