



Agile Software-Entwicklung

Informationswirtschaft II

Wolfgang H. Janko, Stefan Koch und Michael Hahsler

Inhalt

- Einleitung - Agile Manifesto
- Agile Vorgehensmodelle
 - Extreme Programming
 - Open Source Software-Entwicklung (?)

Agile Manifesto

Agile Manifesto der Agile Alliance (Februar 2001) durch
Vertreter von:

- Extreme Programming
- SCRUM
- DSDM Consortium
- Adaptive Software Development
- Crystal
- Feature-Driven Development
- Pragmatic Programming

unter anderem Kent Beck, Alistair Cockburn, Ward
Cunningham, Martin Fowler, Jim Highsmith,...

Agile Manifesto: Gundayee

- Die Umwelt wird immer turbulenter, dadurch kommt es immer öfter zu Änderungen an der Anforderungsspezifikation einer Software.
- Die Frage ist daher nicht, wie man dies verändern oder vorhersehen kann, sondern wie man dies bestmöglich behandelt.

Agile Manifesto: Values

Manifesto: 'We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions over processes and tools**
- **Working software over comprehensive documentation**
- **Customer collaboration over contract negotiation**
- **Responding to change over following a plan**

That is, while there is value in the items on the right, we value the items on the left more.'

Agile Manifesto: Grundprinzipien

- Our highest priority is to satisfy the customer through **early and continuous delivery** of valuable software.
- Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
- **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Agile Manifesto: Grundprinzipien

- **Business people and developers must work together** daily throughout the project.
- Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and active method of conveying information to and within a development team is **face-to-face conversation**.
- **Working software is the primary measure of progress**.

Agile Manifesto: Grundprinzipien

- Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to **technical excellence** and good design enhances agility.
- **Simplicity** - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from **self-organizing teams**.
- At regular intervals, the **team reflects on how to become more active**, then tunes and adjusts its behavior accordingly.

Extreme Programming

Kent Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, Reading, Massachusetts, 1999.

- nach dem Beispiel eines Entwicklungsteams bei Daimler-Chrysler
- für die Entwicklung in kleinen bis mittelgroßen Teams
- 'extreme' wegen übertriebenen Einsatzes anerkannter Techniken

Extreme Programming: 12 Grundprinzipien

1. Planungsprozess (schnelle Planung, Umfang der nächsten Iteration festlegen, Prioritäten vergeben, Vertreter von Entwicklungs- und Geschäftsseite)
2. kurze Releasezyklen (1-2 Monate, immer eine vollständige, sinnvolle Iteration)
3. Metapher (gesamte Entwicklung von einer gemeinsamen, einfachen Metapher geleitet, z.B. Desktop)

Extreme Programming: 12 Grundprinzipien

4. einfaches Design (besteht alle Tests, ohne Redundanzen, legt Absichten der Programmierer offen, kleinstmögliche Anzahl Klassen und Methoden)
- 5. Automatisiertes Testen** (Erstellung automatisierter Testfälle gleichzeitig zum Programmieren, dauerndes Testen)
6. Refactoring (vorhandenes Programm verbessern, um neue Funktionen leicht hinzufügen zu können)
- 7. Pair Programming** (2 Personen programmieren gemeinsam, wechseln sich ab, einer behält Überblick, beide sammeln Erfahrung)

Extreme Programming: 12 Grundprinzipien

8. gemeinsame Verantwortlichkeit (jeder hat Verantwortung für Gesamtsystem, darf alles ändern)
9. fortlaufende Integration (jeden Tag Integration, Durchführung aller Tests)
- 10. 40-Stundenwoche** (keine 2 Wochen hintereinander Überstunden notwendig, sonst wurde zuviel Funktionalität vereinbart)
11. Kunde vor Ort (für Fragen etc.)
12. Programmierstandards

Extreme Programming: Arbeitsumgebung

Betonung der Arbeitsumgebung

- keine Trennwände
- Gemeinschaftsraum für Entspannung, schnelle Besprechungen
- Essen vorrätig,...

Extreme Programming: Lebenszyklus

- **Erforschung** (Möglichkeiten erforschen, Architektur überlegen, Leistungsmerkmale bestimmen)
- **Planung** (realistischen Termin für kleinste, wertvollste Menge von Leistungsmerkmalen festlegen)
- **Iterationen bis zur ersten Version** (1-4 Wochen je Iteration, mit Tests)
- **Produktion** (1-wöchige Iterationen am Ende einer Version, vor allem zur Optimierung)
- **Wartung** (Erstellung neuer Versionen, zuerst wiederum Erforschung, Datenübernahme)
- **Tod** (wenn keine neuen Funktionalitäten mehr auftauchen)

Extreme Programming: Einschränkungen und Kritik

- nur mit bestimmten Technologien möglich: abgestimmt auf objekt-orientierte Entwicklung, Umgebung für schnelles Compilieren und Testen,...
- Kritik:
 - maximal 10 Personen
 - gute soziale Fähigkeiten Voraussetzung
 - ein Vorgehensmodell ?

Open Source Software und Entwicklung

Warum?

- Linux
- Apache Web Server
- Perl
- GNU Utilities
- ...

Begriffsdefinition: Open Source Software ist Software, deren **Lizenz** dem Benutzer bestimmte, weitgehende Rechte einräumt

Open Source Software: Anforderungen an Lizenzen

- **Free Redistribution** (freie Weitergabe ohne Gebühr)
- **Source Code** (Weitergabe in kompilierter Form und Source Code, dieser darf nicht absichtlich unverständlich gemacht sein)
- **Derived Works** (Änderungen sind erlaubt, müssen unter Bedingungen des Originals weitergegeben werden können)
- **Integrity of Author's Source Code** (z.B. durch Versionsnummernvergabe)
- **No Discrimination** against Persons, Groups of Fields of Endeavour (z.B. Genforschung)
- **Distribution of Licence** (muss einfach für jeden gültig sein)
- **Licence must not be Specific to a Product** (Unabhängigkeit von Distribution)
- **Licence must not Contaminate other Software** (z.B. keine Forderung, dass alles auf einer CD Open Source sein muss)

Open Source Software: Beispiel-Lizenzen

- GNU General Public Licence (**GPL**): bekanntestes Beispiel, Verschärfung: jede Modifikation muss unter GPL stehen, darf nicht in kommerzielle Produkte integriert werden ('Copyleft')
- GNU Library General Licence (**LGPL**): vor allem für Bibliotheken, darf in kommerzielle Produkte integriert werden
- Andere: X, BSD, Apache, Artistic Licence,...

Open Source Software: Historische Entwicklung

- eng verbunden mit Unix
- zu Beginn der Computerära fast alles Open Source (Auslieferung mit Source Code)
- nach AT&T Kartellprozess (durfte keine Software mehr verkaufen) freie Entwicklung von Unix (Uni Berkeley)
- später kommerzielle Versionen
- GNU Projekt für Entwicklung eines freien Betriebssystems
- zugleich Weiterentwicklung in Berkeley - BSD (Berkeley Software Distribution)
- Hauptaugenmerk: **Linux** (begründet von Linus Torvalds, Vorbild Minix für x386-Prozessoren)
- Gründung von GNOME und KDE (Oberfläche und Office Anwendungen)
- Netscape gibt Source Code von Browser frei (Mozilla-Projekt)
- ...

Open Source Software: Entwicklungsprozess

am besten anhand Vergleichs (Eric S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly and Associates, Sebastopol, California, 1999.)

Cathedral

- bisheriger Ansatz
- strikte, zentralisierte Kontrolle über Design und Code
- klare Managementstruktur
- feste Release-Zeiten

Bazaar

- große Anzahl Beteiligte (GNOME, Apache: kleiner innerster Kreis, hunderte Programmierer, tausende Mitwirkende)
- freiwillige Mitarbeit
- kleine Beiträge
- Programmieren und Testen stark parallelisieren ('Given enough eyeballs, all bugs are shallow.')
- Benutzer werden zu Mit-Entwicklern - dazu nötig: Source Code offen, schnelle Release-Zyklen zur Duplizierungsvermeidung, Aufmerksamkeitserhaltung ('Release early, release often.')
- eigentlich Spiral-Modell ('microspirals')

Open Source Software: Motivation

- direkter Bedarf
- Wunsch nach Reputation (Geschenkkultur)
- künstlerische Freude (Handwerkermodell)
- Investition in eigenes Humankapital (Übung, Reputation,...)
- eigene informelle Regeln für Reputation (wichtige Distributionen, innovative Projekte, mühsame Arbeit,... bringen mehr)
- ökonomisch: 'cooking-pot market' = Arbeitsteilung: ,I provide the chicken, you the goat, she the berries, together we share the spiced stew' vgl. Rishab Aiyer Ghosh in firstmonday 3(3)

Open Source Software : Projektgründung

- jemand mit Angebot unzufrieden
- beginnt Eigenentwicklung
- notwendig für Prozess: erste lauffähige Version, Vision,
- modulares Design (um Kommunikationsbedürfnis zu reduzieren), Bsp. Linux

Open Source Software: Projektentwicklung

- technische Vorkehrungen (Source-Code-Verwaltung, Bug-Verwaltung, Mailing-Listen, Web-Site,...)
- organisatorische Vorkehrungen (Owner/Maintainer, wohlwollender
- Diktator mit 'trusted lieutenants', Komitee - Apache)

Open Source Software: Projektende

- kaum feststellbar
- wegen Verlassen durch Maintainer
- sollte an einen Nachfolger übergeben
- Inbesitznahme eines verlassenen Projektes möglich (Web-Site UFO)

Open Source Software: Vergleich mit klassischer Software-Entwicklung

- Marketing-Anforderungen (durch Benutzer selbst, entweder implementieren oder einfordern, kaum formalisiert)
- Systemdesign (kaum vorhanden, vielleicht in einem Vorbild, Informationen für Aufwandsschätzung fehlen)
- Detaildesign (ebenfalls kaum vorhanden, im Code versteckt)
- Implementierung (Hauptaugenmerk der Mitglieder, Ausprobieren neuer Ansätze)
- Integration (wenig formelle Systemtests)
- Feldtest (vielfältig, motivierte Beteiligte)
- Support (eher durch kommerzieller Unternehmen angeboten)

ein Vorgehensmodell?

ein Software Process?

eine agile Form der Software-Entwicklung?

Open Source Software: Kritik

- Fehlen von Design: dadurch Aufwandserhöhung ↔ Gegenargument Modularisierung
- Hoher versteckter Aufwand durch Aufteilung versteckt: Programmierer und Debugger (viele Benutzer suchen, wenige finden Fehler)

Open Source Software: Geschäftsmodelle (Auswahl)

- **Loss-Leader/Market-Positioner:** soll Marktanteile für kommerzielle Software schaffen (Bsp. Netscape)
- **Give away the Recipe, open a restaurant:** Verkauf von Dienstleistungen (Customizing, Distributionen - RedHat, S.u.S.E.,...)
- **Accessorizing:** Verkauf von Zubehör (z.B. O'Reilly-Verlag)
- **Free the Software, Sell the Content:** Client frei, Inhalt kostet