Softwareengineering

Zi. B 241
irene.rothe@h-brs.de
Instagram: irenerothedesign

Was ist Software?

→ Menge von Bits über eine Mauer geworfen?

ODER

→ Etwas, was einem das Leben leichter macht?

Verbesserungsansprüche

- Personen und Interaktionen sind wichtiger als Prozesse und Werkzeuge
- Zusammenarbeit mit Kunden ist wichtiger als Vertragsverhandlungen
- Reagieren auf Veränderungen ist wichtiger als Festhalten an einem starren Plan

Software ist das Ergebnis von Ingenieurtätigkeit

=

Softwareengineering

Es gibt keinen richtigen Weg, aber zahlreiche falsche.

Was gehört zum Softwareengineering?

- Phasen der Programmentwicklung
- Kriterien an einen Software-Entwurf
- Zyklus der Programmierung
- Tools zur Unterstützung
- Aufgaben eines Projektleiters
- Art der Zusammenarbeit (Agiles Programmieren)

Zitat aus einem Praxissemesterabschlussbericht 2019:

Ein mangelhaftes Grundkonzept kann später bei der Programmierung zu großen Fehlern führen, da man oft den Überblick über das Projekt verliert und daher das Grundkonzept bearbeiten muss oder im schlimmsten Fall sogar neu schreiben muss.

Hilfsmittel und Tipps

- Versionsverwaltungswerkzeuge
- Automatischer Buildprozess
- Werkzeug, das Tests unterstützt, Builds, Fehler anzeigt und verschickt
- Bugtracker-Werkzeug
- Aufgabenverteilung im Team: Dokumentation, Installation, Eingaben/Ausgaben, …
- Flussdiagramme
- 10-Fingertippen
- Scriptgesteuertes Erstellen einer Entwicklungsumgebung

Ich erzähle Ihnen nichts Originelles! Aber angeblich benutzen 50-70% all die bekannten und erprobten Konzepte überhaupt nicht!

Phasen der Programmentwicklung

1. Problemanalyse
2. Entwurf (Algorithmus)
3. Implementierung (Programm)
4. Test
5. Praxistest
6. Wartung

zu Phasen: Problemanalyse

- Was soll gemacht werden, was nicht? Abgrenzung von schon vorhanden Lösungen
- Muss/Soll/Nice-to-have
- Technische Bedingungen ändern sich besonders schnell (z. B. Schnittstellen zur Hardware)
- Verhandlungen mit Auftraggeber: Softwareentwickler und Auftraggeber sprechen nicht die „gleichen Sprachen“ (Lastenheft: Wünsche an das Produkt, Pflichtenheft: Was soll programmiert werden?)
- Festgehalten in **Pflichtenheft/White Paper**

zu Phasen: Fachlicher Entwurf

- Spezifikation aus fachlicher Sicht
- Aufbau einer Softwarearchitektur
- Eventuelle Implementierung eines Prototypes (Vorprodukt)
- Kosten/Design müssen im gesunden Verhältnis zueinander stehen
- Anwendung von Entwurfsmustern (Datenhaltung, Veränderung der Daten, Präsentation der Daten)
- **Specifikation 1 Paper** (Hilfsmittel: **Flussdiagramme, UML,…**)

zu Phasen: Technischer Entwurf

- Aufteilung in Teilsysteme (Komponenten)
- Aufteilung der Teilsysteme in Module (getrennt übersetzbar)
- Spezifikation gemeinsamer Schnittstellen (Methodenaufruf, Zugriff auf gemeinsame Daten)
- Definitionen der Datenstrukturen
- Formulierung grundlegender Funktionen
- **Technical Requirement Paper** (**UML** für Klassenaufbau, Use Cases für Prozessabläufe)

Implementierung

- Codierung der Module (20-40% des gesamten Entwicklungsprozesses)
- Testung der Module (Definition von Testfällen und Testdaten)
- Arbeit eines Programmierers: Editieren → Compilieren → Testen/Debuggen → und immer so weiter

Achtung: Benutzt man gezauberten Quelltext von „Assistenten“ MUSS dieser 100%ig verstanden werden.

zu Phasen: Testung

- Unittest
- Integrationstest
- Performance-Test
- Usability-Test

Bemerkungen:

- Testdaten (erst künstliche, dann reale)
- Test testen

Man weiß niemals, ob man sorgfältig genug getestet hat (Werkzeuge für Abdeckungsanalyse).

Testung

Man weiß niemals, ob man sorgfältig genug getestet hat (Werkzeuge für Abdeckungsanalyse).

Tests sind gut,

- um zu zeigen, wie ein Modul benutzt wird.
- als Hilfsmittel für alle zukünftigen Tests.

Testen ist keine technische Aufgabe, sondern eine kulturelle. Besser die Software selbst testen, als sie vom Kunden testen zu lassen.

Testung

Ablauf:

- Testung der Module, Teilsysteme, komplette Lösung
- Fehlersuche
- Fehlerbehebung

Aufgaben:

- Konformität des Programms mit den Spezifikationen
- Durchspielen der Software mit künstlichen und realen Daten (Frage: Wo kommen die her?)
- Genauigkeit der Ergebnisse (Abweichung innerhalb bestimmter Grenzen)
- Zahl der tolerierbaren Ausfälle oder Fehlfunktionen
- Qualität der mitgelieferten Dokumentation
- Benutzung von Programmverifikations-Programmen (sehr teuer)
- Zeitplan
- Gute Dokumentation der Fehler

System- und Integrationstest

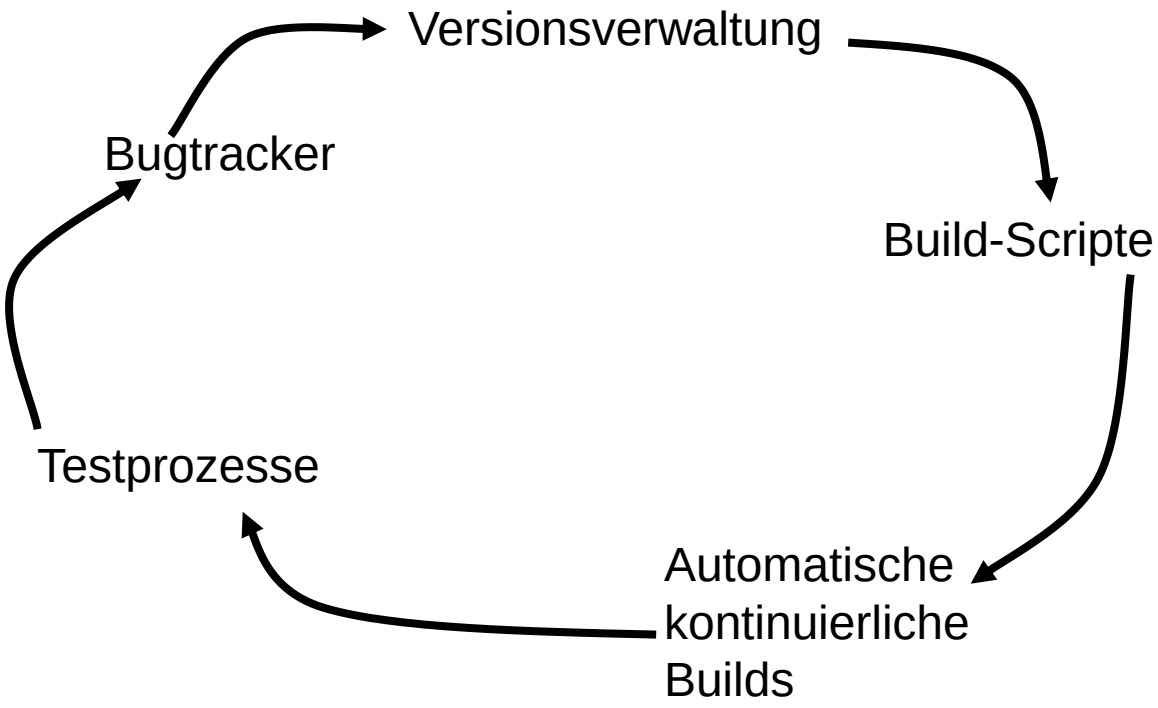
- Module werden zu Untersystem zusammengefasst und getestet
- Debuggen bei Fehlersuche, Fehlerbehebung, neuer Test
- Systemtest: α-Test, β-Test (bei ausgewähltem Kunden)
- Integrationstest = Abnahmetest

Achtung: Murphy‘s law gilt auch hier!

Durchführen von Programmtests

- Qualitätsplan (Was ist zu testen?)
- Zeitplan
- Planung und Programmierung automatischer Tests vor oder gleichzeitig mit der Produkt/Feature-Programmierung
- Automatische Testung
- Automatische Testreporterzeugung
- Werkzeug für Fehlerreports nutzen (z.B. Bugzilla)
- Gute Dokumentation des Tests, der Fehler, …

Zyklus und Einsatz von Softwarewerkzeugen (Infrastruktur)



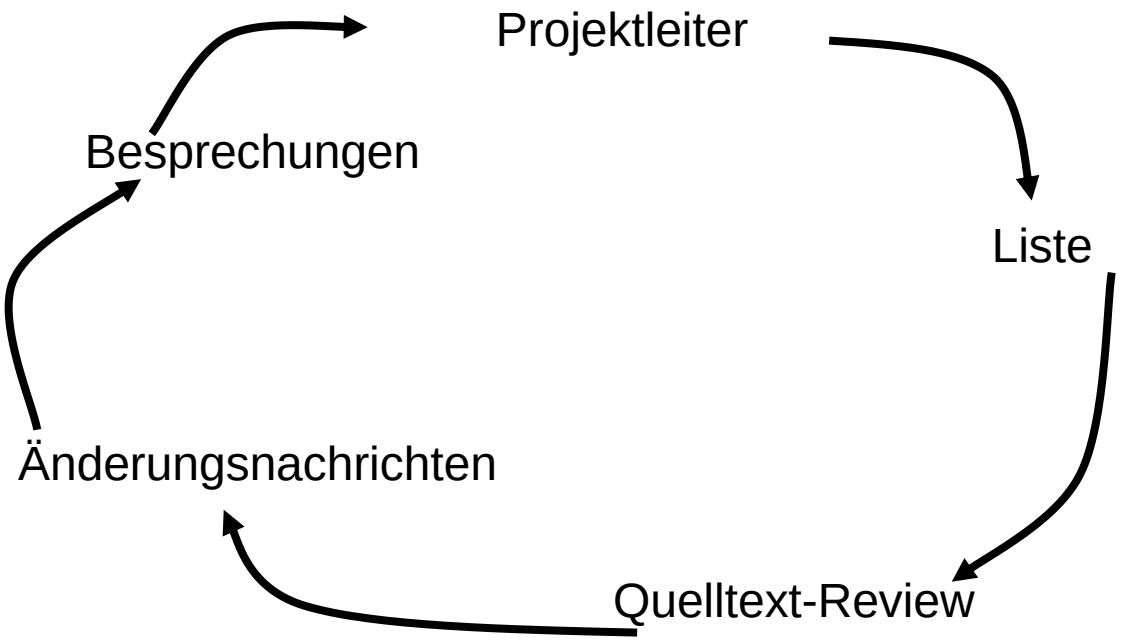
Einsatz von Softwarewerkzeugen

- Unterstützung von Entwicklung, Test, Dokumentation (Editoren, IDEs, Testsoftware, Dokumentationsprogramme)
- Debugger (Programme zur Fehlersuche): technischer Ablauf Schritt für Schritt
- Automatischer Buildprozess (z.B. make-Dateien)
- Versionierung (z.B. CVS: Aus- und Einchecken, geregelter Zugriff, Build-Nummernvergabe)
- Werkzeuge für Fehlerreports
- Werkzeuge zur Hilfe bei der Erstellung von Installationsprogrammen
- Werkzeuge zur Überwachung des Buildprozesses und der Tests

Arbeit mit Softwarewerkzeugen

- **Kurzanleitung** erstellen für Mitarbeiter (auch in Versionsverwaltungssystem einchecken)
- Bekanntmachung
- Tägliche Benutzung
- Check, ob alle Mitarbeiter gut damit klar kommen

Projektleitung



Projektleitung

- Stetige Motivation
- Koordinierung
- Kontrolle
- Projektplan (Grobpläne und Teilpläne)
- Zuteilung der Ressourcen
- Festlegung von Meilensteinen
- Festlegung von Qualitätszielen
- Aufwandsabschätzung
- Regelmäßige Projektsitzungen
- Vertreten des Projekts nach außen!!!!
- Qualitätssicherungsplan/LISTE (rot/gelb/grün)
- Projektabschluss und Bericht

Tools für Management

- Wikis
- Die LISTE
- Intranet

Tool: Die Liste

- Öffentlich
- Gruppenliste
- Persönlich Liste
- Prioritäten von Features (Listenelemente)
- Messbar
- Zielbezogen
- Z.B. GoogleDrive

Projektleiter

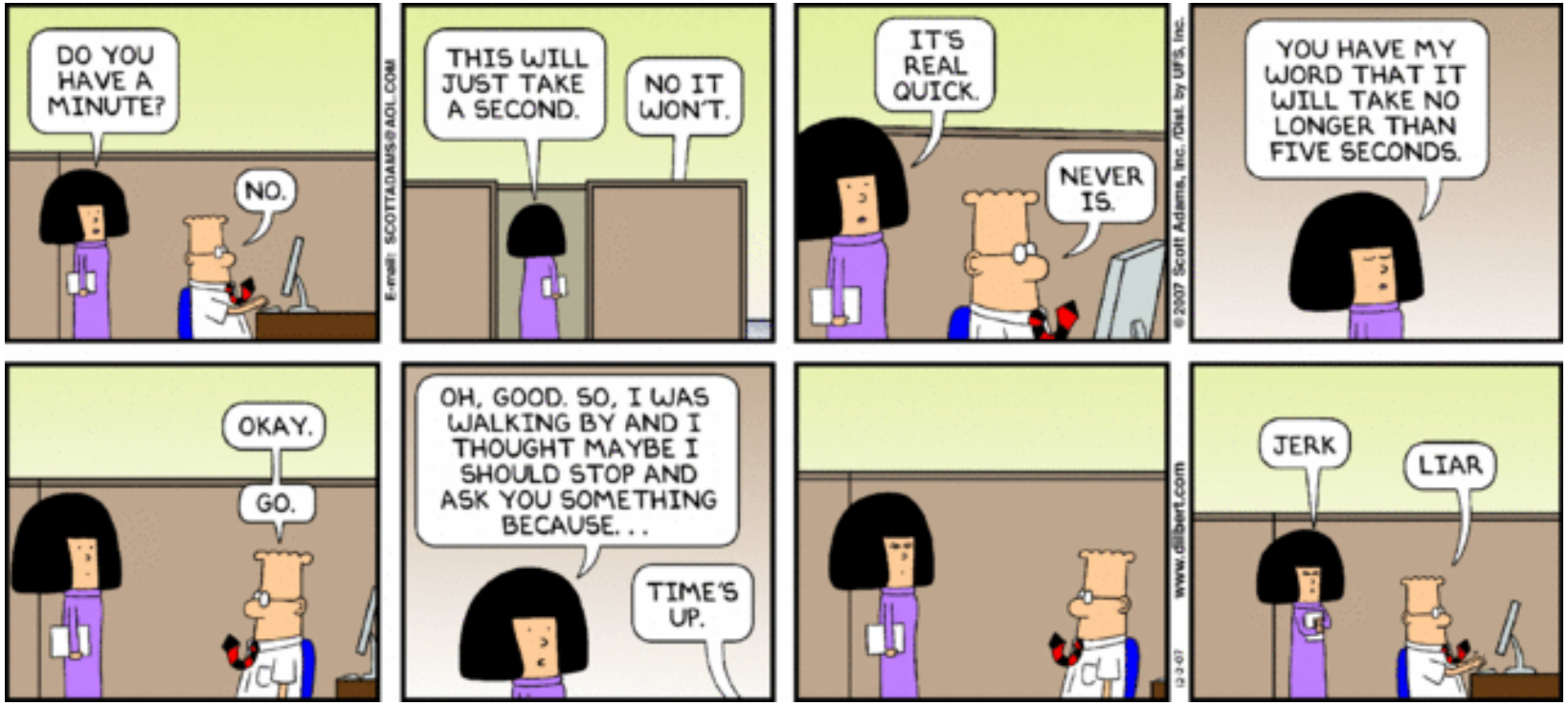
- Führen der Gruppenliste
- Priorisieren der Features
- Priorisieren der Fehler
- Abschirmen des Teams von Ablenkungen

Ein Projektleiter muss:

- wissen woran jedes Mitglied arbeitet
- fähig sein, in 5 Minuten Statusbericht zu geben
- die nächsten 5 Features kennen

Tägliche Besprechungen ersetzen das Großraumbüro.

Kommunikation



SCRUM (restarting a rugby game)

- Scrum master:
  - makes sure that team applies Scrum-ideology
  - Removes barriers
  - Shields team
- Scrum meeting:15 minutes (all team members stand up and talk):
  - What did you do yesterday
  - What will you do today
  - What for problems you see
  - Move post it on table
- Table: listed items/items in progress/done items
  - Items are post-its
- Vorteile:
  - Team bestimmt, wie lange für die einzelnen Aufgaben Zeit benötigt wird
  - Team ist selbst organisiert
  - Es gibt keinen Chef, sondern Selbstkontrolle
- Abschätzung von Zeit: jedes Teammitglied vergibt Punkte, Erklärung für höchste Punktzahl, Erklärung für niedrigste Punktzahl

Teamzusammensetzung

BART

- **Technical skills: Great**
- **Social skills: Okay**
- **Motto: I will NOT write docs**

HOMER

- **Technical skills: Below average**
- **Social skills: Okay**
- **Motto: When do we have lunch?**

Teamzusammensetzung

NERD

- **Technical skills: Unbeliavable**
- **Social skills: None**
- **Motto: C++**

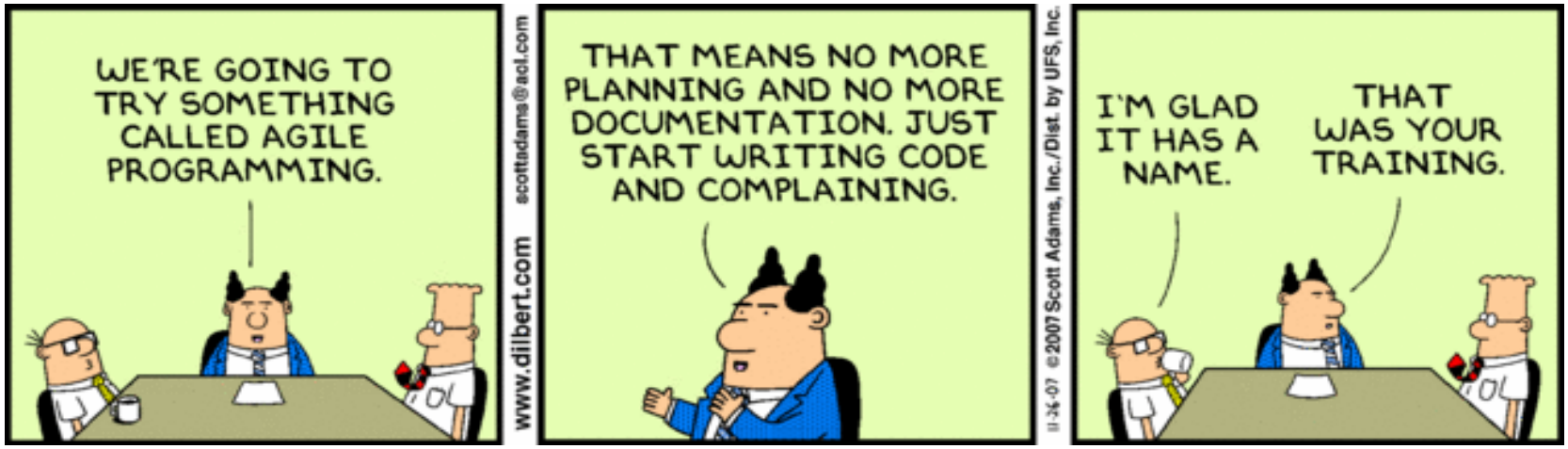
??

- **Technical skills: great**
- **Social skills: great**
- **Motto: For a change, lets walk backwards**

Agile Softwareentwicklung: Methoden

Beispiele:

- Paarprogrammierung
- Testgetriebene Entwicklung
- Code-Reviews
- Ständiges Refaktorisieren



Agile Softwareentwicklung: Code Review

- Aller 2-3 Tage
- Check Kommentare und Variablennamen
- Wissenstransfer
- Entdeckung von Dopplungen
- Darf nicht schmerzhaft sein, sondern nur hilfreich
- Funktioniert nur, wenn der Reviewer den Code versteht, sonst → Technik des Code-Erklärens
- Guter Start für Refaktorisierung
- Angabe der Person, die reviewed hat

Agile Softwareentwicklung: Paar Programming

- Zwei Programmierer arbeiten an einem Rechner
- Einer schreibt den Code, der andere kontrolliert
- Lösung von Problemen im Gespräch zu zweit
- Programmierer sollten sich abwechseln
- Zusammensetzung der Paare immer mal ändern
- Ziele: höhere Qualität, Wissenstransfer, Disziplin, Mentoring, Teambildung, weniger Unterbrechungen (Leute im Gespräch unterbricht man seltener als jemanden alleine am Schreibtisch)

Agile Softwareentwicklung: Refactoring

- Restrukturierung/Umgestaltung von Programmen, ohne Änderung des Verhaltens (Ähnlich zu Vereinfachung von Formeln in der Mathematik)
- Ziele sind die Erhöhung der:
  - Lesbarkeit
  - Verständlichkeit
  - Testbarkeit
  - Vermeidung von Redundanzen
- Beispiele:
  - Variablennamen-Änderung
  - Verschiebung einer Methode in eine andere Klasse
  - Aufteilung eines Moduls
  - Verschönerungen: Einrückungen, Klammersetzung
- Wird nur bei funktionierenden Programmen (Unit-Test erfolgreich durchlaufen) gemacht
- Eventuell Anlage eines Refactoring-Katalogs

Dokumentation im Quelltext

- **Kommentare** behandeln das *Wieso* (das Wie steht ja im Quelltext)
- **Variablennamen**: ausschreiben, nicht irreführend wählen
- **Autorname** setzen, bedeutet ***Stolz*** auf die Arbeit (gemacht vom ***Fachmann)***

Vorgaben bei der Programmierung

- Keine `goto` Anweisungen
- Verwenden vieler sinnvoller Methoden/Funktionen
- Keine ähnlichen Funktionen
- Wenige bis keine Redundanzen
- Unabhängige Module/Objekte (Veränderungen an einem Modul beeinflusst kein anderes Modul)
- So gut wie keine globalen Variablen
- Flexible Architektur: DB sollte austauschbar sein
- Nicht zu viele ineinander geschachtelte Schleifen

Programmierung

- Flussdiagramm: PapDesigner
- UML
- Diskussion über Klassen in der Gruppe
- pro Vierteljahr ein Buch über Programmierung lesen oder 2. Sprache lernen (Probleme werden in verschiedenen Sprachen unterschiedlich gelöst)

Neu: DevOps 1

= Zusammenführung von Development und IT-Operations

… ist ein Mindset

Es gibt 4 Arten von Arbeiten/Aufgaben:

- Businessaufgaben
- Interne Aufgaben
- Änderungsaufgaben
- Ungeplante Arbeit (Notfälle)

Neu: DevOps 2

Dafür braucht es **3 Wege**, die man gehen muss:

1. Arbeitsfluss von Entwicklung über IT-Operationen zum Kunden, Engpässe suchen (kleine Gruppen mit Arbeitsintervallen, keine fehlerhaften Produkte weitereichen, auf globale Ziele hin optimieren, kontinuierliche Builds inklusive der Umgebungen)
2. Schnelles und konstantes Feedback („Stoppen des Fließbands“, Messpunkte, gemeinsame Ziele, automatische Tests und Builds)
3. Kultur (Vertrauen, Verletzlichkeit zeigen), Experimentieren, Risiken eingehen, aus Fehlschlägen lernen, Wiederholung und Übung

Neu: DevOps 3

… ist ein Mindset

Das bedeutet fürs Team:

- man geht davon aus, dass alle Teammitglieder gut arbeiten wollen, gerne arbeiten, ein gutes Produkt liefern wollen
- effizientere Zusammenarbeit durch gegenseitiges Verständnis für die Arbeit der anderen
- Förderung von Lern- und Experimentierfreude
- hohe Zufriedenheit der Mitarbeitenden
- man sollte nicht der Idiot sein, der NICHT um Hilfe

Bemerkung: kann man überall im Leben anwenden

Neu: DevOps 4

- Skripte und Tools für die agile Entwicklung
- Automatisierung steht weit oben, z.B. Build- und Release-Prozesse
- Automatisiertes Testen
- Verwaltung über z.B. Github
- Infrastructure wird wie Code behandelst (automatisiert, versioniert…)
- Dokumentation, Schulungsmaterial…Euer Fingerabdruck!

→ IT+Bussiness müssen eins sein

→ IT hilft Business erfolgreich zu sein

Buch: „Projekt Phoenix: Der Roman über IT und DevOps - Neue Erfolgsstrategien für Ihre Firma“, Gene Kim , Kevin Behr, et al., O‘Reilly, 2015

Buch: „DevOps Cookbook“ DeBois, Wills, Orzen

Das Marie Kondo Prinzip

If it doesn‘t spark joy, get rid of it.

→ Wenn etwas keinen Spaß macht, ändere es!
→ If a class/method/line doesn‘t spark joy, get rid of it.
→ Weniger ist immer mehr
→ Joy: jede Zeile Code macht das, was sie soll (ohne 5 Bedingungen abzufragen oder 4 Fälle gleichzeitig abhandelt)

Frage nach Verstoß gegen single responsibilty principle, ansonsten refactoring, open close principle? → siehe SOLID (single resp., open-close, Liskov substitution (Ersetzbarkeit in Oberklassen), Interface segregation, Dependency inversion (Abstraktion, keine Konkretisierung), nicht mehr als 2 Ebenen tief, zu viele ifs in zu vielen Schleifen → no feeling of joy

Quelle: https://www.karim-geiger.de/blog/das-marie-kondo-software-design-principle

Literatur

- Richardson, Gwaltney: „Ship it!“, Hanser 2006
- Andrew Hunt, David Thomas: „Der pragmatische Programmierer“