

Projektmanagement und Software Engineering

Dieter Kranzlmüller

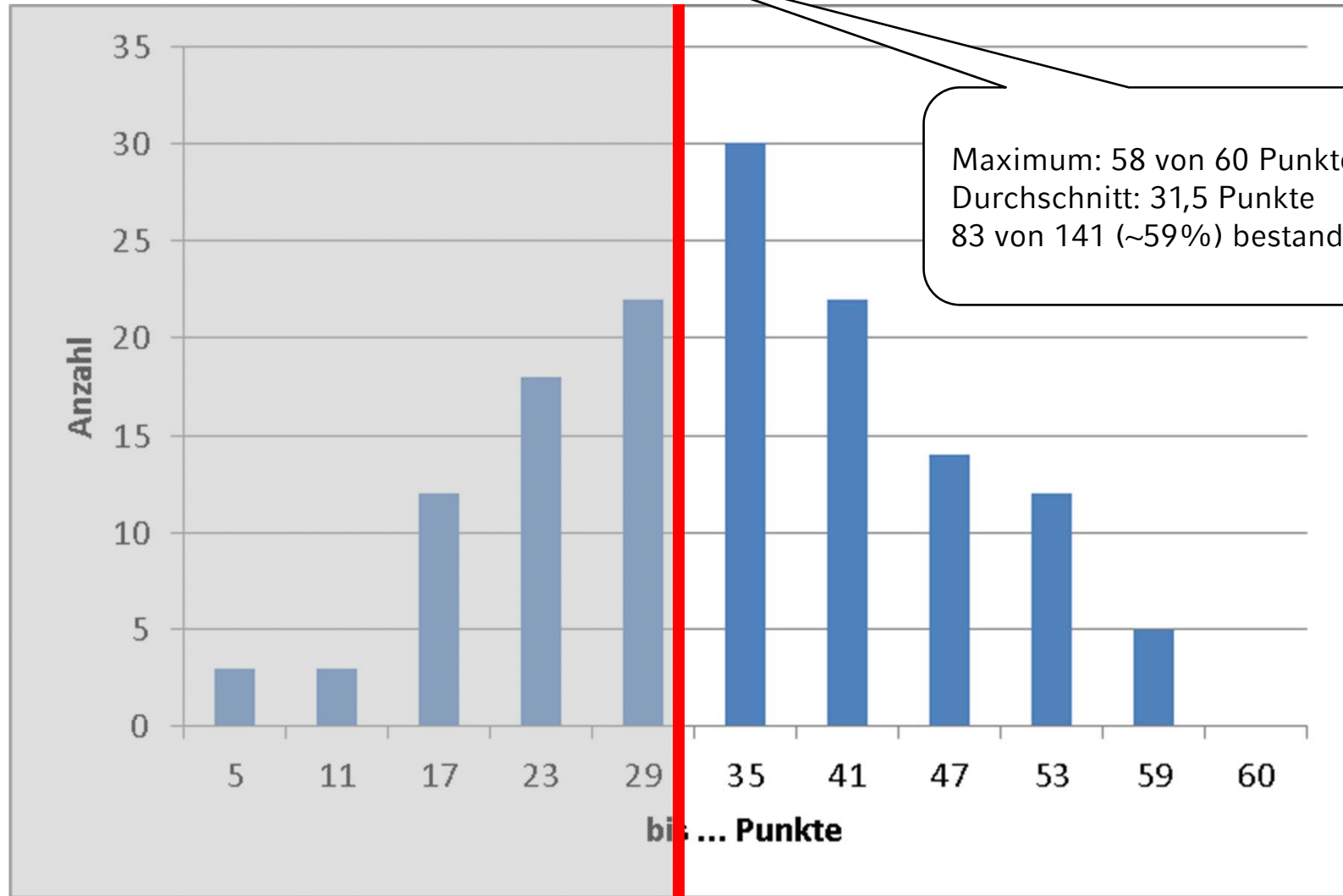
Nils gentschen Felde



Eignungsfeststellungsverfahren

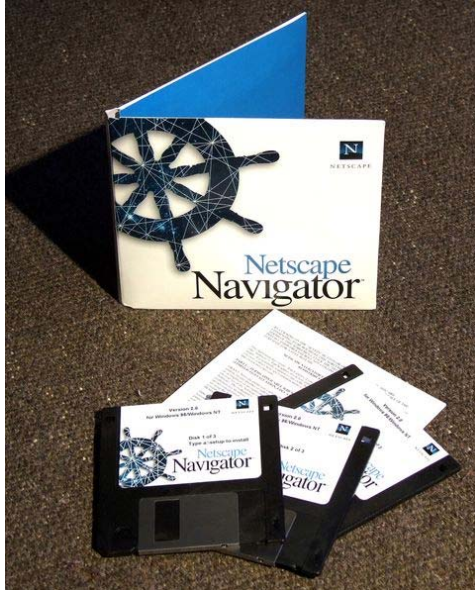


Punkteverteilung im EFV



- Bearbeitung eines Projekts bis zum Ende des Semesters
 - Bearbeitung in Gruppen aus 3-4 Personen
 - Wöchentliche Treffen mit dem Tutor Ihrer Gruppe
 - Notenfindung durch Prüfung zum Semesterende
- Heutige Veranstaltung
 - Projekt und Meilensteine vorstellen
 - Theoretische Grundlagen für Bearbeitung des Projekts erläutern
 - Überblick über Termine bis zum Ende des Semesters geben

- 4 Meilensteine
 - Vorgaben, die es einzuhalten gibt (Lernziele)
 - Allerdings viele Freiheiten bei der Ausgestaltung
- Erstellung eines eigenen Zeit- und Projektplans

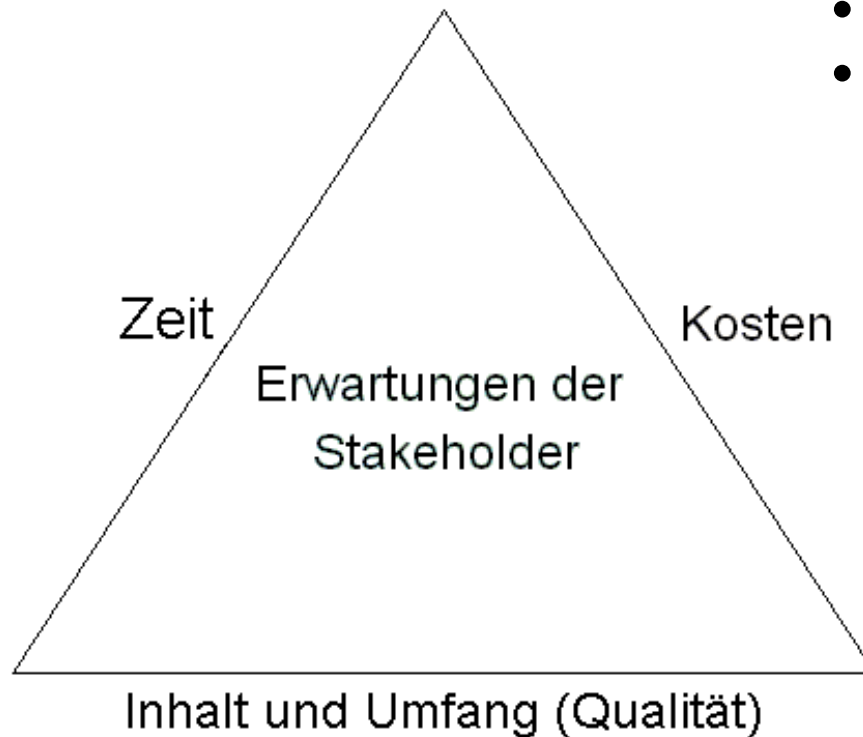


Server:

- Web-Oberfläche
- „Spielleiter“
- sysprak.priv.lab.nm.ifi.lmu.de

- Projektmanagement und Software Engineering
 - Motivation und Definition
 - Vorgehensmodelle
 - Synchronisation und Teamarbeit
 - Implementierung

Das „magische Dreieck“



Quelle: <http://www.pmq.s.de>

- Konflikte zwischen Dimensionen!
- Keine der drei Größen ist unbegrenzt vorhanden
→ Kompromiss notwendig

„Projektmanagement ist die Anwendung von Methoden, Hilfsmitteln, Techniken und Kompetenzen in einem Projekt. Es umfasst das [...] Zusammenwirken der verschiedenen Phasen des Projektlebenszyklus.“

ISO 21500:2013-06

- ✓ Projekt**grenzen**/Projekt**ziele** adäquat definieren
- ✓ Projekt**pläne** entwickeln & periodisches Controlling
- ✓ Projekte prozessorientiert **strukturieren**
- ✓ Projekt**organisation** projektspezifisch gestalten
- ✓ Spezifische Projektkultur entwickeln

Vorgehen

- Vorgehensmodelle
- Planungstools

Teamarbeit & Synchronisation

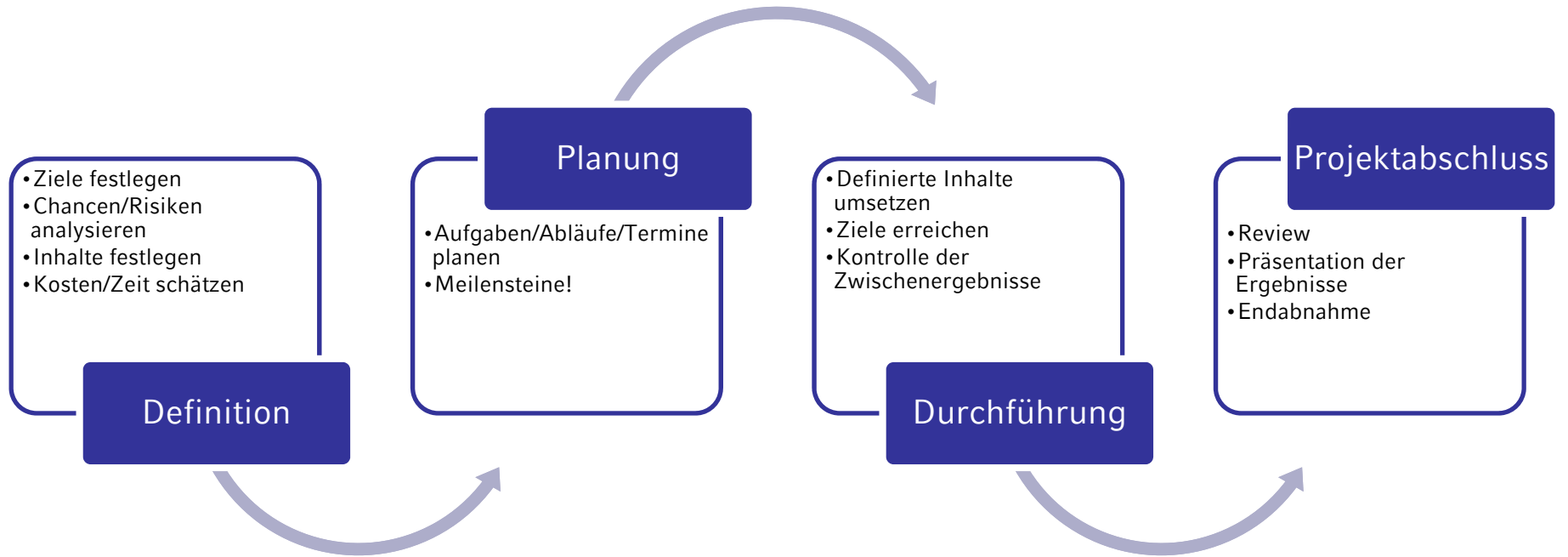
- Termine & Aufgaben
- Code & Bugtracking
- Vokabular

Implementierung

- Modularisierung
- Prototyping
- Dokumentation
- Testen



Systempraktikum



03.11.2014

08.12.2014

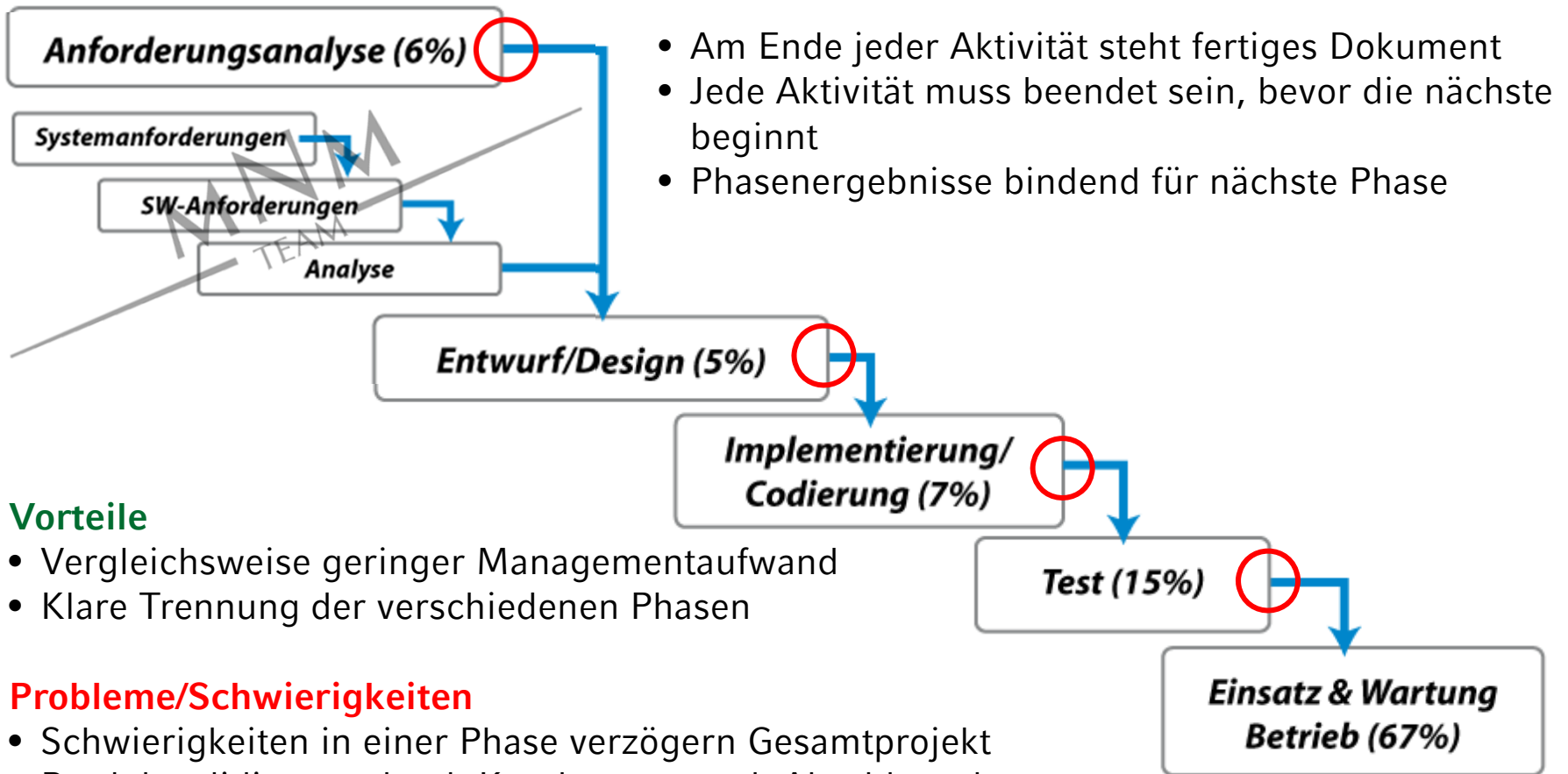
15.12.2014

02.2015

- Auch „Software-Prozess-Lehre“
 - Befasst sich mit dem Vorgang der Herstellung von Software
- Ziel: Software-Herstellung soll
 - produktiver (weniger ausgelieferte Fehler bei weniger Aufwand)
 - schneller (Durchlaufzeit, Time-to-market)
 - beweglicher,
 - vorhersagbarer
 - nachvollziehbarer werden

	Sequentiell	Iterativ	Adaptiv/Agil
Eigenschaften	<ul style="list-style-type: none"> Sequentielles Vorgehen Klar definierte Phasen und Ergebnisse 	<ul style="list-style-type: none"> Entwicklungsprozess wird mehrfach iteriert, d.h. besteht aus Folge von Zyklen (Iterationen) Am Ende jedes Zyklus steht neue (ausführbare) Version des Software-Produkts, die vorherige verbessert/erweitert 	<ul style="list-style-type: none"> Weiterentwicklung des iterativen Paradigmas: dynamische Planung der Iterationen Kontinuierliche Anpassung an Veränderungen Minimale(r) Prozess/Dokumentation - gerade so viel wie nötig
Besonders geeignet wenn	<ul style="list-style-type: none"> Klare, relativ fixe Funktionalität Qualität/Zuverlässigkeit wichtiges Kriterium, z.B. medizinische Systeme 	<ul style="list-style-type: none"> Anforderungen sind nicht von Beginn an klar/instabil Funktionalität/Termin wichtiger als Aufwand 	<ul style="list-style-type: none"> Ziele zu Beginn unklar, sich ändernde Anforderungen/Umgebung
Vorteile	<ul style="list-style-type: none"> Einfach durchzuführen Sehr effizient bei bekannten & konstanten Anforderungen 	<ul style="list-style-type: none"> Häufiger Kontakt zum Anwender jeweils bei d. Erstellung einer neuen Version ("evolutionäres Prototyping") Frühe Erkennung von Risiken Berücksichtigung volatiler Anf. 	<ul style="list-style-type: none"> Verspricht besseres Kosten/Nutzen-Verhältnis Kleine Schritte ermöglichen bessere Reaktion auf Komplexität Nicht alle Entscheidungen am Anfang notwendig
Nachteile	<ul style="list-style-type: none"> Risiken gesammelt am Schluss („Big Bang“) Starr während des Ablaufs 	<ul style="list-style-type: none"> Architektur bestehender und neuer Funktionalitäten passt kompatibel → ggf. erheblicher Mehraufwand komplexeres Projektmanagement 	<ul style="list-style-type: none"> Qualitätseigenschaften nicht garantiert Herkunft von Funktionalität oft schwierig nachzuvollziehen (für medizinische Systeme usw. daher ungeeignet) Teilweise wenig Akzeptanz
Beispiele	Wasserfallmodell, V-Modell	Spiralmodell, Unified Process	Extreme Programming, SCRUM

Auswahl erfordert Abwägung/Trade-off

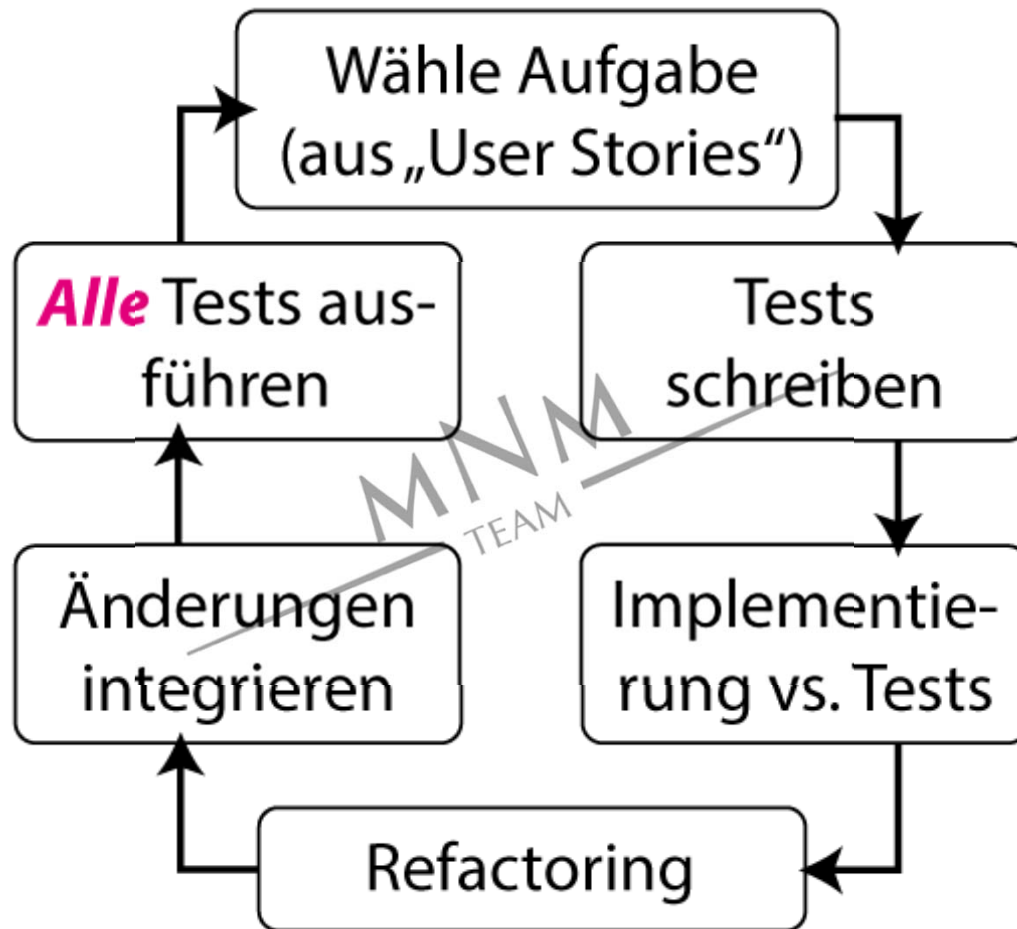


Vorteile

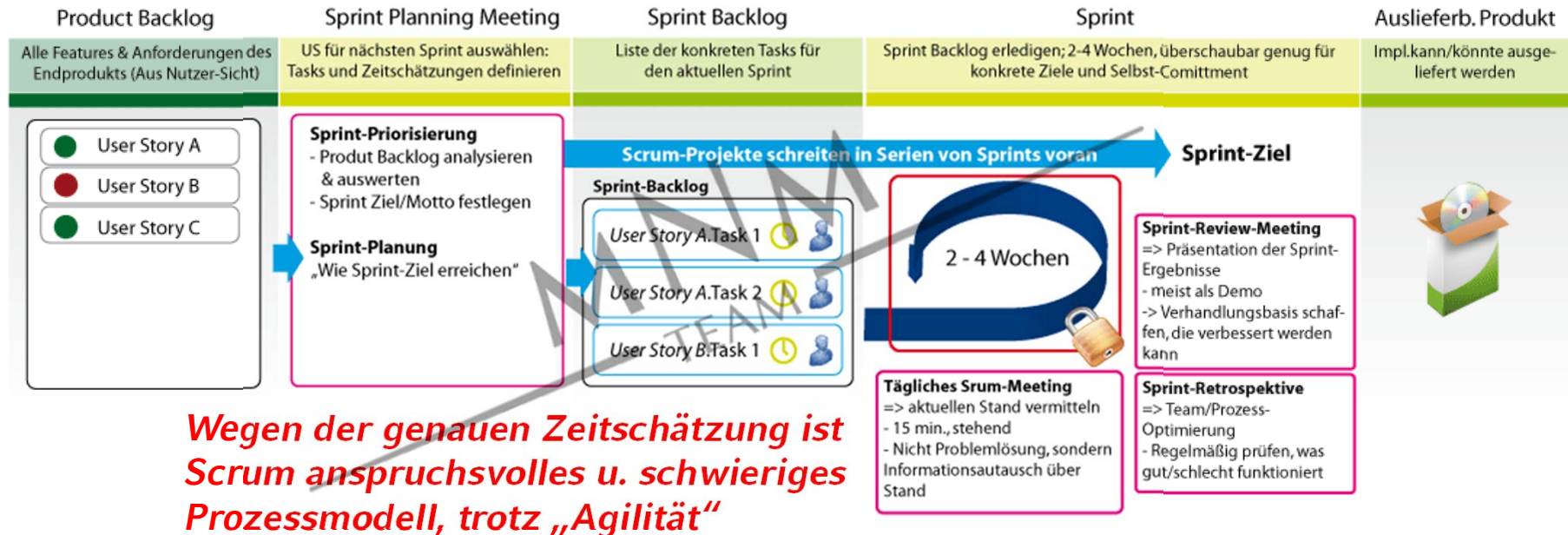
- Vergleichsweise geringer Managementaufwand
- Klare Trennung der verschiedenen Phasen

Probleme/Schwierigkeiten

- Schwierigkeiten in einer Phase verzögern Gesamtprojekt
- Produktvalidierung durch Kunden erst nach Abschluss d. gesamten Entwicklung



- Testgetrieben
- Tests sind „Anforderungen“
- viel Kommunikation (paarweises Programmieren, schnelle Rückmeldung durch Anwender)
- Inkrementelle Erweiterung (iterativ) des Gesamtsystems
- Basiert auf den Prinzipien „simplicity“, „communication“, „feedback & courage“.



Vorteile

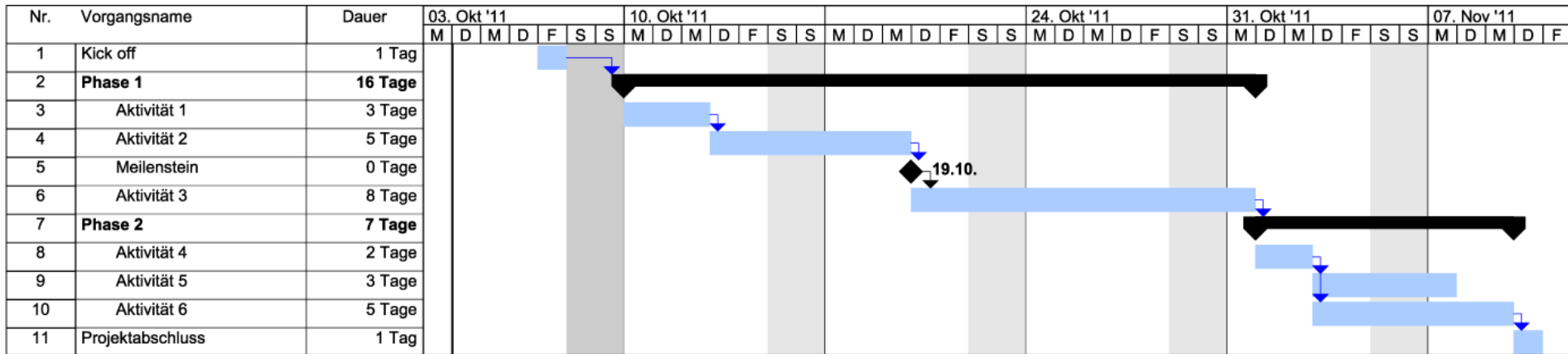
- Viel & offene Kommunikation
- Schnelle Reaktion auf Anforderungsänderungen möglich

Probleme

- Viele formale Vorgaben (Rollen, Meetings inkl. Vorbereitung, Artefakte, etc.)
- Häufig idealisiert, z.B. keine Änderungen während des Sprints
- Planung & Verwendung der Tools braucht Zeit
- Abhängigkeiten zwischen Tasks normal

- **Welches** Problem soll gelöst werden
→ *Problemabgrenzung*
- **Wo** ist das Problem
→ *Problemkontext*
- **Warum** muss es gelöst werden
→ *Ziele der Beteiligten*
- **Wie** könnte ein Softwaresystem helfen
→ *Szenarien*
- **Was** sind mögliche Hindernisse
→ *Machbarkeit und Risiko*

**Schwierigkeit:
Zeitschätzung**



Freie Software

- Dojo Toolkit – [dojox/gantt](#)
- dotProject
- GanttProject
- Open Workbench
- PHProjekt
- ProjectLibre
- Redmine
- TaskJuggler

Kostenpflichtige Lizenz

- A-Plan
- ACOS Plus 1
- Genius Project
- InLoox
- Merlin (Software)
- Microsoft Project
- Microsoft Visio
- JCV Gantt Pro (Add-on für Mindjet MindManager)
- PLANTA Project
- Plant Simulation
- Projektron BCS
- RPlan
- Wrike

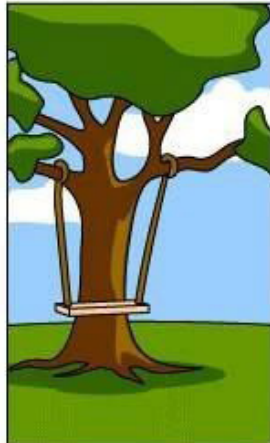
Datum	ID	Beschreibung	Verantwortlich
	MS1		
dd.mm.yyyy	Modellierung	Klassendiagramme erstellen o.ä.	Alle
	Repository aufsetzen	SVN-Repository im CIP-Pool für Referenzimplementierung einrichten	Alle
	Modularisierung	Header-Files im Quellcode vorbereiten und ins Referenz-SVN einchecken	Alle
	...		
	...		
	MS2		
	...		
	...		
	MS3		
	...		
	...		
	MS4		
	...		
	...		

Abgabe dieser Tabelle

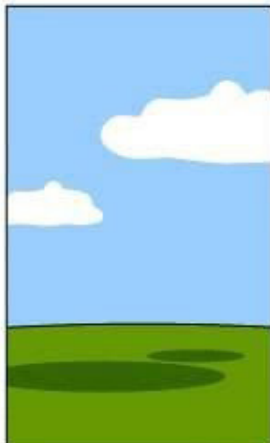
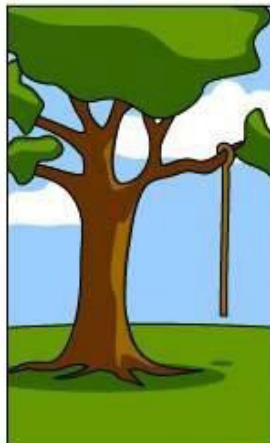
- bis **10.11.2014, 23:59 Uhr**
- Abgabe per Email an sysprak-admin@nm.ifi.lmu.de
im Betreff „<Gruppennummer> - Abgabe der Meilensteinplanung“
- Es gibt nur eine Abgabe pro Gruppe



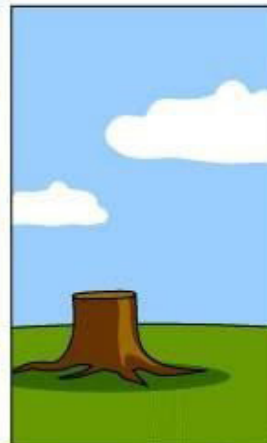
Was der Kunde erklärte

Was der Projektleiter
verstandWie es der Analytiker
entwarfWas der Programmierer
programmierte

Was der Berater definierte

Wie das Projekt
dokumentiert wurde

Was installiert wurde

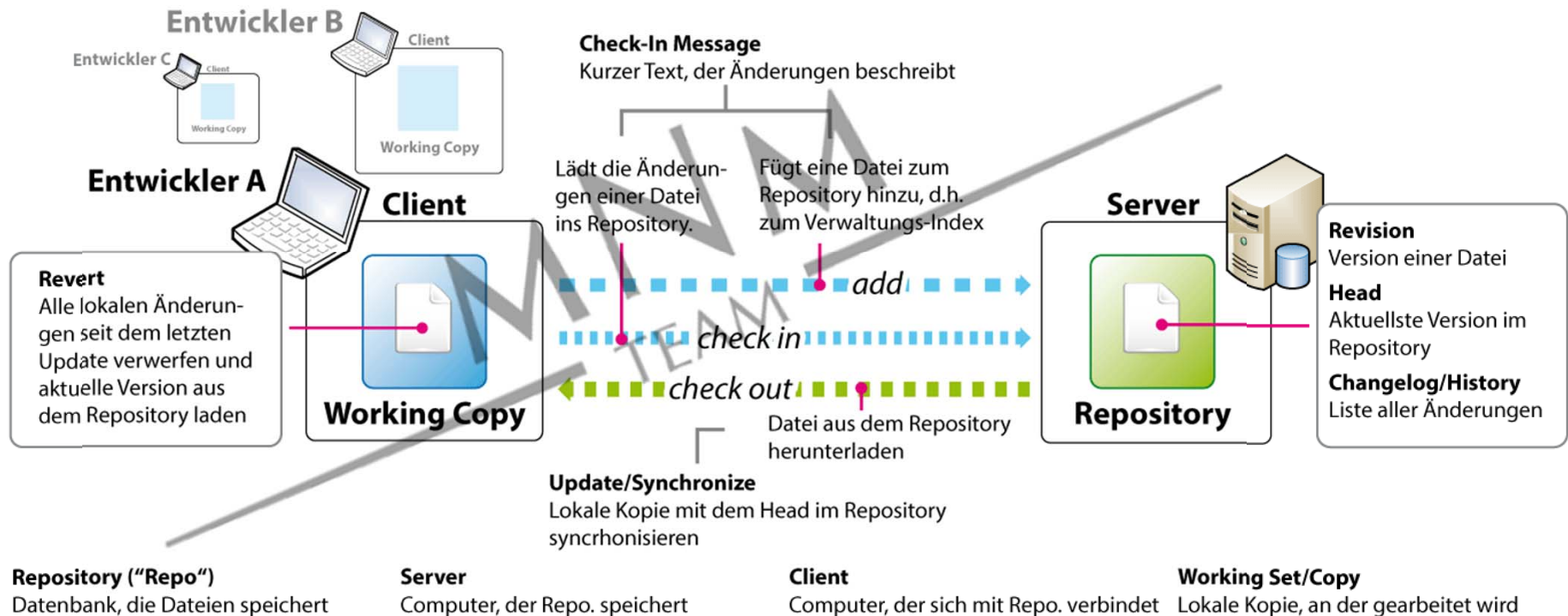
Was dem Kunden in
Rechnung gestellt wurde

Wie es gewartet wurde

Was der Kunde wirklich
gebraucht hätte

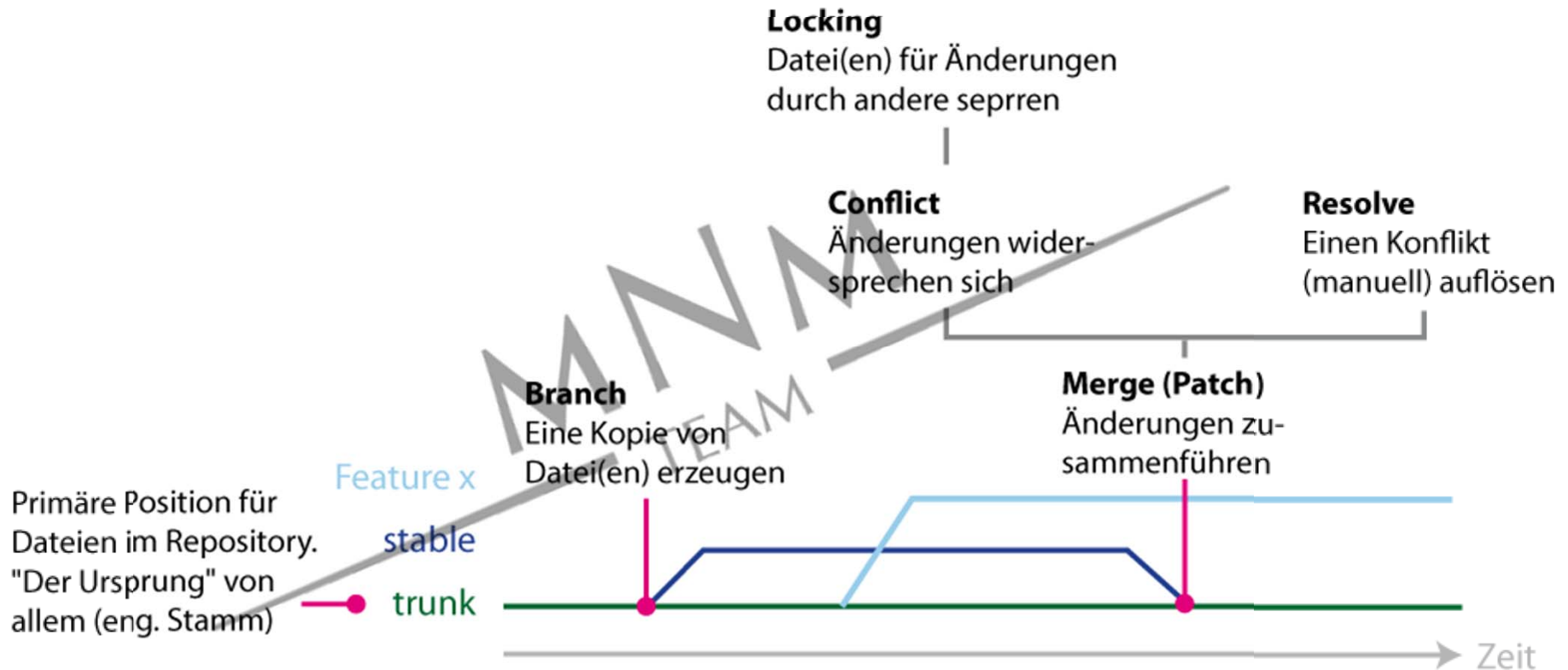
- RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels
 - MUST - This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification
 - SHALL - ...
- Gemeinsame Terminologie festlegen
- Patterns (siehe später)

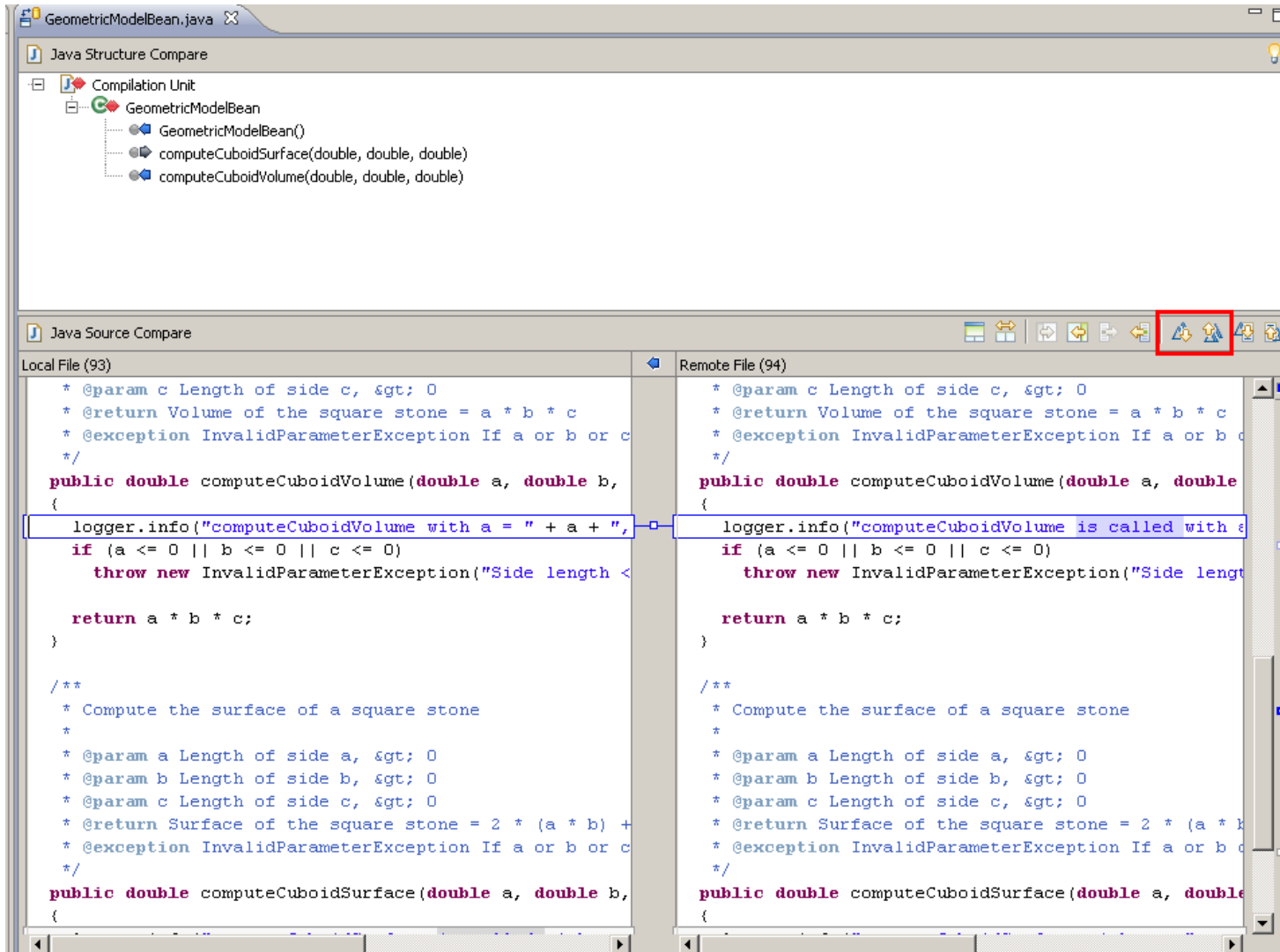
- **Backup & Restore** – Zu einer beliebigen früheren Version zurückkehren (Short- und Long-Term-Undo)
- **Synchronisation** – Kollaboratives Arbeiten an einem Projekt oder sogar einer Datei
- **Änderungsverfolgung** – Zeitstempel, etc.
- **Sandboxing** – In lokaler Kopie arbeiten und testen, bevor committet wird
- **Mehrere Zweige** – Es kann für ein Programm mehrere Zweige (Branches) geben, z.B. „Productive“, „Edge“, „Feature x“. Merge führt anschließend zusammen
- **Eigentümerfeststellung** – Von Dateien, Änderungen und Branches



betterexplained.com/articles/a-visual-guide-to-version-control/
www.ericssink.com/scm/source_control.html

Code-Verwaltung – Beispiel für Branches





GeometricModelBean.java

Java Structure Compare

- Compilation Unit
 - GeometricModelBean
 - GeometricModelBean()
 - computeCuboidSurface(double, double, double)
 - computeCuboidVolume(double, double, double)

Java Source Compare

Local File (93)

```
/* @param c Length of side c, >= 0
 * @return Volume of the square stone = a * b * c
 * @exception InvalidParameterException If a or b or c
 */
public double computeCuboidVolume(double a, double b,
{
    logger.info("computeCuboidVolume with a = " + a + ",
    if (a <= 0 || b <= 0 || c <= 0)
        throw new InvalidParameterException("Side length <

    return a * b * c;
}

/**
 * Compute the surface of a square stone
 *
 * @param a Length of side a, >= 0
 * @param b Length of side b, >= 0
 * @param c Length of side c, >= 0
 * @return Surface of the square stone = 2 * (a * b) +
 * @exception InvalidParameterException If a or b or c
 */
public double computeCuboidSurface(double a, double b,
{
```

Remote File (94)

```
/* @param c Length of side c, >= 0
 * @return Volume of the square stone = a * b * c
 * @exception InvalidParameterException If a or b
 */
public double computeCuboidVolume(double a, double
{
    logger.info("computeCuboidVolume is called with a
    if (a <= 0 || b <= 0 || c <= 0)
        throw new InvalidParameterException("Side lengt

    return a * b * c;
}

/**
 * Compute the surface of a square stone
 *
 * @param a Length of side a, >= 0
 * @param b Length of side b, >= 0
 * @param c Length of side c, >= 0
 * @return Surface of the square stone = 2 * (a * b) +
 * @exception InvalidParameterException If a or b
 */
public double computeCuboidSurface(double a, double
{
```



... und alles ist
ein Nagel

	Concurrent Versions System (CVS)	Apache Subversion (SVN)	Git
URL	www.nongnu.org/cvs	subversion.apache.org	git-scm.com
Eigenschaften	<ul style="list-style-type: none"> • Älteste Code-Verwaltung, daher recht ausgereift, aber weniger Funktionen 	<ul style="list-style-type: none"> • Bietet atomare Operationen: entweder werden <i>alle</i> Änderungen übernommen oder gar keine 	<ul style="list-style-type: none"> • Dezentrales Repository → kein Checkout, sondern Clon des gesamten Repositories • Jede Datei und jeder Commit erhält eine Checksumme
Voraussetzungen	<ul style="list-style-type: none"> • Server läuft i.d.R. auf UNIX-Systemen, Clients für verschiedene OS vorhanden 		<ul style="list-style-type: none"> • Primär für Unix-Systeme entwickelt
Vorteile	<ul style="list-style-type: none"> • Ausgereift 	<ul style="list-style-type: none"> • Unterstützt atomare Operationen • Günstigere Branch-Operation • Viele IDE-Plugins und Clients vorhanden 	<ul style="list-style-type: none"> • Lokale Branches möglich • Datenintegrität gewährleistet • Hohe Performance (war eines der Design-Ziele) • History-Tree vollständig offline verfügbar • Staging: Commit kann noch mal formatiert werden
Nachteile	<ul style="list-style-type: none"> • Dateien verschieben/umbenennen ist keine neue Revision • Keine atomaren Operationen 	<ul style="list-style-type: none"> • Teilweise nicht ausreichende Funktionalitäten zur Repository-Verwaltung • Vglws. langsame Diff-Operation 	<ul style="list-style-type: none"> • Steile Lernkurve

- CVS www.wasd.web.cern.ch/wwwasd/cvs/tutorial/cvs_tutorial_toc.html
- SVN svnbook.red-bean.com
- Git git-scm.com/book

Grundidee: dasselbe (bewährte) generische Lösungsmuster für sich **ähnende Probleme** wiederverwenden

Muster (Pattern)

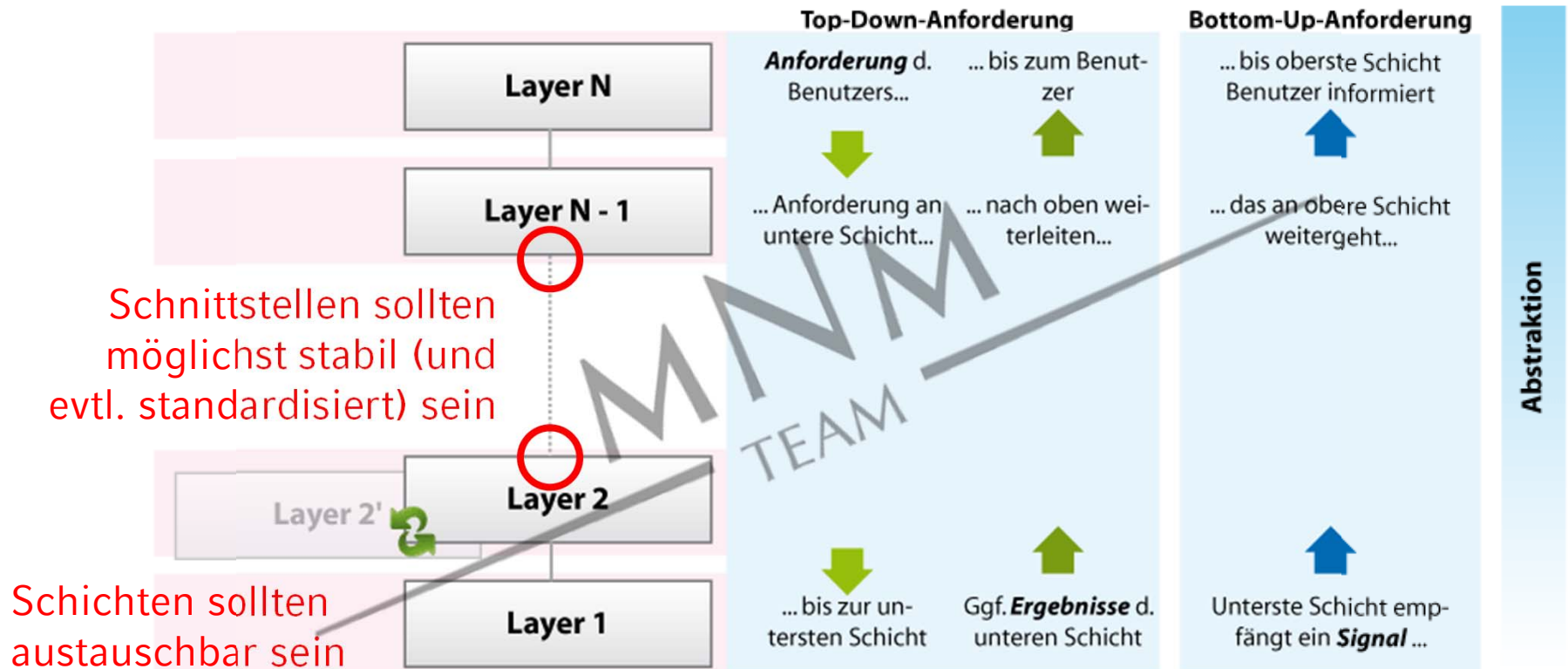
*Abstrakte & programmiersprachen-unabhängige
Schablone, die eine vorbildliche, bewährte Lösung für ein
ausgewähltes, immer wiederkehrendes Problem beschreibt*

Vorteile

- ✓ Wiederverwendung von Lösungsprinzipien
- ✓ Abstrakte Dokumentation von Entwürfen
- ✓ Gemeinsames Vokabular zur (schnellen)
Verständigung unter Entwicklern

System konstruieren, das Aktivitäten

- wie HW-Steuerung, Sensoren, Bitverarbeitung auf niederer Ebene vereinigt,
- wie Planung, Strategien, Anwenderfunktionalität auf hoher Ebene vereinigt, wobei
- auf hoher Ebene durch Aktivitäten auf niederer Ebene realisiert werden



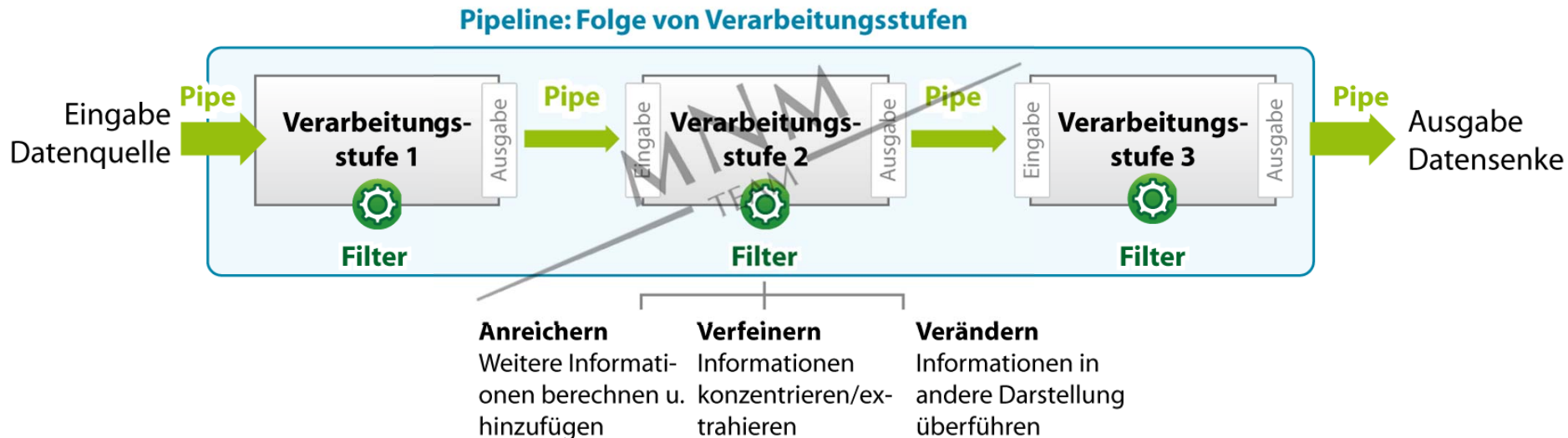
Trennung von Aktivitäten entspr. Abstraktionsgrad

Beachten

- Änderungen am Quellcode sollen möglichst wenige Ebenen betreffen
- Jede Ebene sollte separat realisierbar sein
- „Optimale“ Anzahl Schichten finden schwierig
 - zu wenig → komplexe monolithische Komponenten, schwer wartbar
 - zu viele → Reduktion der Komplexität einzelner Schichten, aber hohe Anzahl zu durchlaufender Schnittstellen kann zu Leistungseinbußen führen

System konstruieren, das Eingabe-Datenstrom verarbeiten/umwandeln soll, wobei

- System nicht als monolithischer Block gebaut ist
- Austausch oder Neukombination von Teilen möglich ist
- Nicht aufeinander folgende Verarbeitungsstufen entkoppelt sind



Vorteile

- Kein Speichern von Zwischenergebnissen (z.B. in Dateien) notwendig
- Flexibilität durch Filter-Austausch und -Rekombination
- Filter können als Prototypen erstellt werden

Nachteile

- Effizienzsteigerung durch Parallelisierung oft nicht möglich (z.B. da Filter aufeinander warten oder nur ein Prozessor arbeitet)
- Overhead durch Datentransformation
- Fehlerbehandlung schwer realisierbar

- Ziele
 - Wartbarkeit des Codes erhöhen
 - Kommunikation zwischen Entwicklern erleichtern
 - „Sustainability“
 - Über Code nachdenken
- Regeln für gute Kommentare
 - So viel wie nötig, so wenig wie möglich
 - Keine Trivialitäten kommentieren
 - Algorithmen, Designentscheidungen, Zusammenhänge erläutern
 - Was ist besonders verwirrend?
 - Wie könnten Schwierigkeiten einem „Dummy“ erklärt werden?

Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, [...].

Erstellt aus kommentiertem/annotiertem Code online (HTML) und offline (Latex) Dokumentationen.

```
/*! A test class */  
  
class Test  
{  
public:  
    /** An enum type.  
     * The documentation block cannot be put after  
     */  
    enum EnumType  
    {  
        int EVa1,      /**< enum value 1 */  
        int EVa2       /**< enum value 2 */  
    };  
    void member();     /**< a member function.  
  
protected:  
    int value;          /**< an integer value */  
};
```

Member Enumeration Documentation

enum Test::EnumType

An enum type.

The documentation block cannot be put after the enum!

Enumerator

EVa1	enum value 1
EVa2	enum value 2

www.stack.nl/~dimitri/doxygen/

Inline-Kommentare

- Funktionsaufrufe mit vielen Parametern oder Verwendung des Rückgabewertes
- Erläuterungen von Variablenverwendungen

```
int iterations = 12;      // set main home directory
if (connect(address)) {  // Successfully connected to address
    ...
}
```

Descriptive Block

- Funktionen und Libraries beschreiben

```
/**
 * a normal member taking two arguments and returning an integer value.
 * @param a an integer argument.
 * @param s a constant character pointer.
 * @see Test()
 * @return The test results
 */
int testMe(int a, const char *s);
```

improvingsoftware.com/2011/06/27/5-best-practices-for-commenting-your-code/

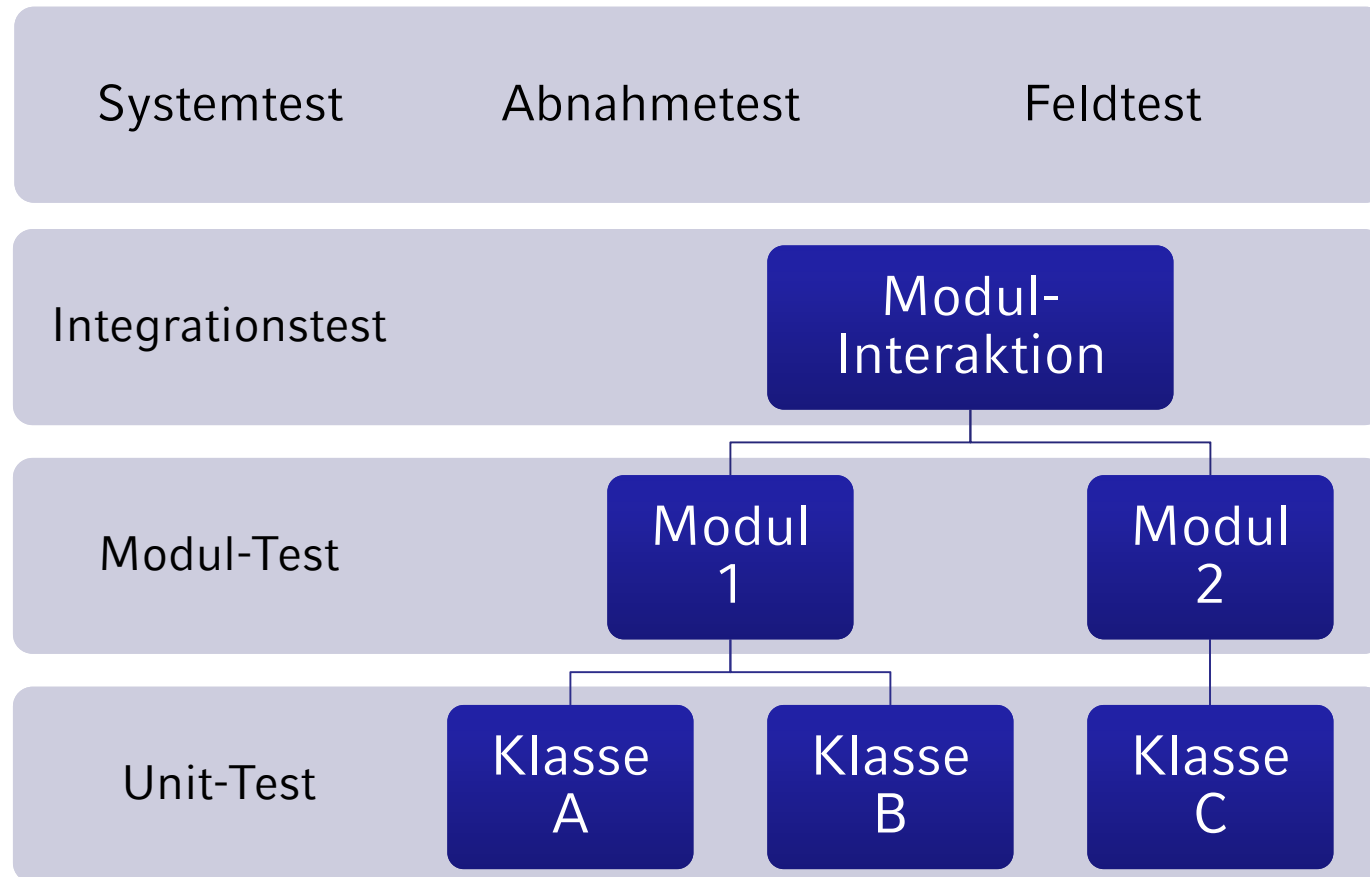
- Prozess, ein Programm auf systematische Art u. Weise auszuführen, um Fehler zu finden
 - Sehr wichtiger Punkt
 - Muss in Planungszeiträume mit einberechnet werden
- Fehler ist
 - Abweichung zw. Ist-Verhalten (im Testlauf festgestellt) u. Soll-Verhalten (in Spezifikation gefordert)
 - nicht erfülltes, vom Kunden vorausgesetztes Qualitätskriterium
- Testen ist destruktiv: zeigt nicht Korrektheit des Programms, sondern Inkorrektheit

- Zentrale Fragen

- Wann wurde genug/ausreichend getestet?
- Sind die Tests vollständig, wurde die gesamte Software getestet?

Wichtig

- Tests müssen reproduzierbar sein
 - Unabhängig von spezieller Datenbasis
- Nach Testen wieder „aufräumen“
(z.B. angelegte Dateien entfernen)



- Zusammenspiel mehrerer Module
- Wenn alle Module korrekt, wieso Integrationstest?
 - Undokumentierte Seiteneffekte
 - unterschiedliche Interpretation v. Operationen
 - Performance meist erst hier sichtbar (Performance additiv)
 - Nebenläufigkeit/Race Conditions

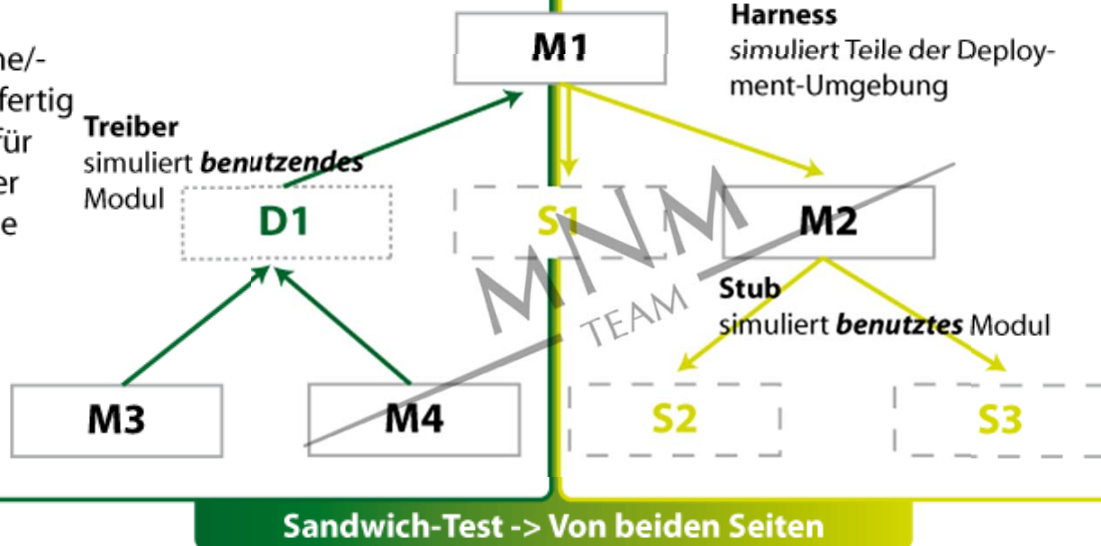
Nicht-inkrementelle Integrationstests

- wartet, bis alle Module vorhanden/fertig sind
- Vermeidet Kosten für Scaffolding (Gerüst bauen, um Tests zu unterstützen)

Bottom-Up

- Kleine Bereiche/-Module schon fertig
- Treiber dient für Anwendung der fertigen Module

Treiber
simuliert *benutzendes*
Modul

**Harness**

simuliert Teile der Deploy-
ment-Umgebung

Top-Down

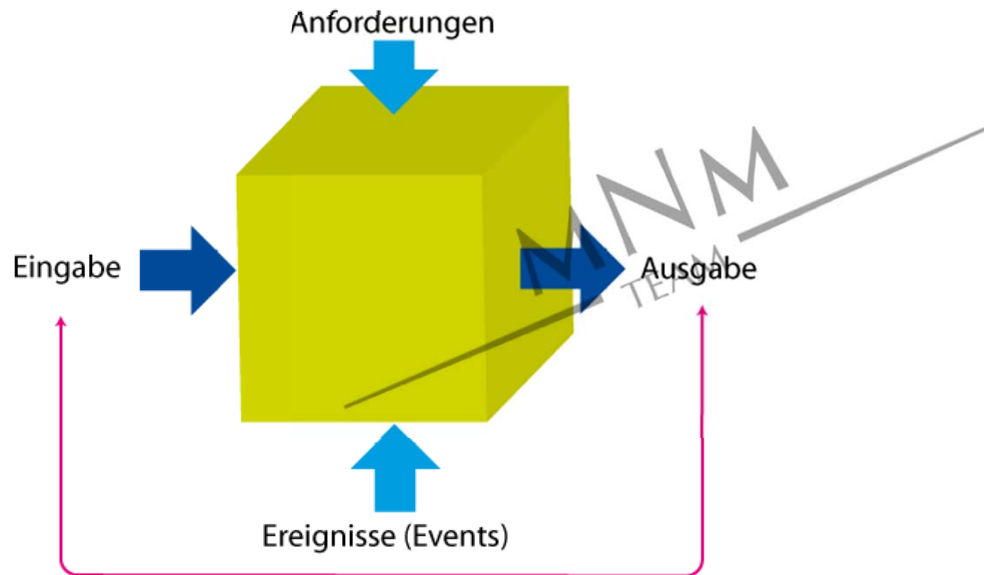
- Fertige Module von oben
- Nicht fertige „Blätter“ werden durch Stubs ersetzt
- Stubs werden nach u. nach durch fertige Module ersetzt
- Bei Integration neuer Module -> Tests erneut ausführen
- Bsp.: Erst GUI erstellen (schnelleres User-Feedback) Daten kommen später; solange durch Stubs simuliert

Regressionstest

- Nach Refactoring/Ersetzen von Stubs/Treibern prüfen, ob nichts kaputt gemacht wurde
- Allein durch Überlegen können nicht alle Auswirkungen u. ggf. Fehler gefunden werden
- Anwendung in Continuous Information System: bei Tests nach Commit wird Ergebnis an Entwickler gemailt (wie bei Capgemini)

Black-Box-Test

(Funktionales/Spezifikationsor. Testen)

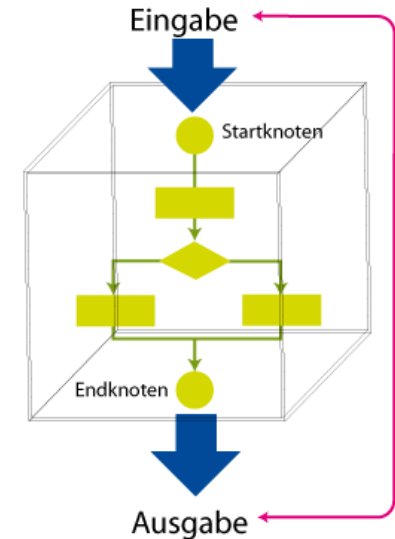


Testfall-Definition: Ausgehend von Spezifikation
Intern d. Testobjekts sind b. Testfall-Def. nicht bekannt

- Test gegen Spezifikationen
- Analyse anhand d. Ein-/Ausgabe
- Alleine sind nicht ausreichend, da Spezifikation höheres Abstraktionsniveau besitzt als Implementierung!

White-Box-Test

(Strukturorientiertes Testen)



Testfall-Definition: Ausgehend von Testobj.-Struktur
Testfälle werden von Entwickler beschrieben

- Möglichst alle Code-Zeilen/Anweisungen, Bedingungen Pfade min. 1x durchlaufen
- Software selbst wird betrachtet/analysiert
- Struktur d. Software Informationsquelle für Testfall-Auswahl

- Mit Prototypen Eignung der Idee prüfen
- Machbarkeits-Studien
- Rapid Prototyping: Quick & Dirty -
Implementierung in (anderer)
Programmiersprache, um fertige Funktionen zu
simulieren; Ausarbeitung in sauberem Code und
verwendeter Sprache
- Zeigt früh Verbesserungsmöglichkeiten
- Erspart Entwicklung unnötiger Features

- Projektmanagement - Trade-Off im „magischen Dreiecks“ finden
- Vier Phasen eines Projekts
- Kommunikation ist wichtige Grundlage
- Patterns für Wiederverwendbarkeit einsetzen
- Software testen

Abgabe der Meilensteintabelle

- bis **10.11.2014, 23:59 Uhr**
- Abgabe per Email, im Betreff „<Gruppennummer> - Abgabe der Meilensteinplanung“
- Es gibt nur eine Abgabe pro Gruppe

Präsentation der Zwischenstände am 08.12. und 15.12.

- Jede Gruppe wählt einen Sprecher
- Dieser stellt Zwischenstand 5 Minuten vor
- Jede Gruppe schickt genau eine Folie jeweils am Freitag vorher an uns