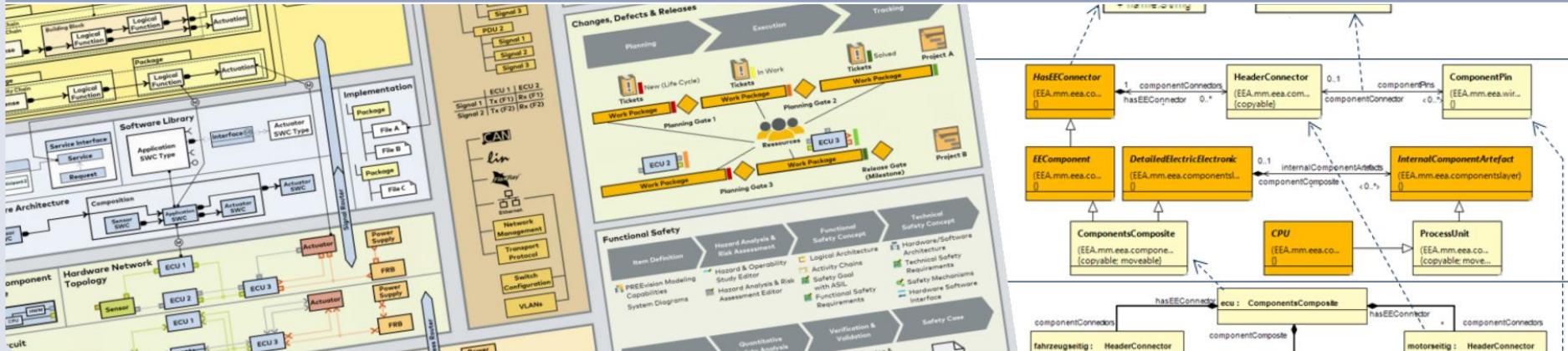


Vorlesung Software Engineering (SE)

Wintersemester 2017/2018

Kapitel 7 – Implementierung



7. Implementierung



@ScottAdamsSays
Dilbert.com



7-19-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel



Inhalt – Implementierung

7.1 Entwicklungswerkzeuge

7.1.1 Übersicht

7.1.2 Compiler und Debugger

7.1.3 Quellcode Versionsverwaltung

7.2 Continuous Development

7.2.1 Continuous Integration (CI)

7.2.2 Continuous Delivery (CD)

7.3 Quellcode Dokumentation

7.3.1 JavaDoc



Werkzeugarten

Editor

Texteditor **zum Erstellen** des Programm-Quelltextes (Quellcode)
(Notepad, vi, Emacs, NoteTab, joe, ...)

Compiler

zur Übersetzung des Quelltextes.
Liefert als Ergebnis eine Datei mit Binärcode (oder Fehlermeldungen).

Interpreter

zur Ausführung von vorhandenem Quelltext (oder 'Zwischencode' wie z.B. Java-Byte-Code) auf einem Rechnersystem.

Debugger

Diagnosewerkzeug **zur Fehlersuche**.

Linker

zum Verbinden von Binärprogrammen (Bibliotheken) zu einem ausführbaren Programm (Build).

CMS (Content Management System)

zur Verwaltung und Dokumentation.

IDE (Integrated Development Environment)
Programmsysteme **zur professionellen** (und komfortablen) **Programm-Entwicklung**.

Compiler, ggf. Interpreter, Debugger, Linker sind Programme, die...

... zum **Programmiersystem** einer Programmiersprache gehören.

... je nach Programmiersprache und Entwicklungssystem (IDE)...

- **kommerziell** erhältlich,
- **frei verfügbar** sind oder
- zum **Betriebssystem** gehören.



- Übersetzer
- Prüft Syntax
- Erzeugt Maschinencode/Bytecode



Maschinensprache (Binärcode)	Assembler-Sprache (Prozessorsteuerung)	Höhere Programmiersprachen
00110010010010101	<pre> MOVEM.L D3-D5, -(SP) MOVE.L D1, D3 ADD D1, D2 SWAP </pre>	<pre> float x,y; x = 7.9; y = x * 4.001; </pre>

Fehlersuche – grundlegendes Vorgehen

Fehler- Detektion

Beobachten eines unerwünschten oder unspezifizierten Zustands des Systems (**error**) oder Versagen des Systems (**failure**).

**Testen
(Monitoring)**

Fehler- Lokalisierung

Auffinden der Stelle, an welcher das Versagen des Systems sichtbar wird.

Suchen der Stelle, an der das System einen **unerwünschten Zustand** zuerst einnimmt.

**Debugging
(Simulation)**

Fehler- Analyse

Finden der **Fehlerursache** (**defect**)

Beheben des Fehlers

Patching

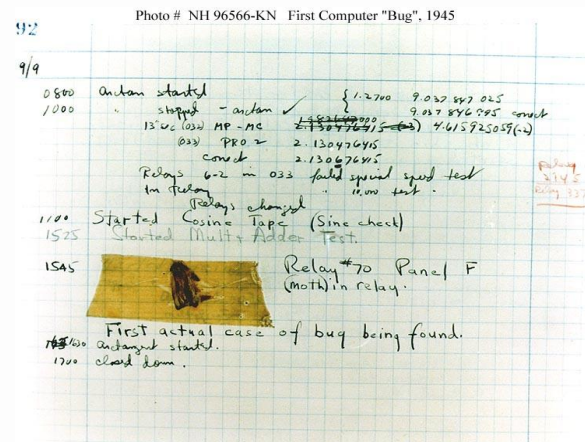
Edison in einem Brief von 1878:

»It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise – this thing gives out and [it is] then that "**Bugs**" – as such little faults and difficulties are called – show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.«

Erster tatsächlicher Computer-Bug:

1945 verfängt sich **eine Motte** (*Bug*) in einem Schaltrelais eines Computers der US Navy

- Insekt wird gefunden und in Logbuch dokumentiert





Geeignet für **imperative Sprachen**

Erstmals 1961: „DEC Debugging Tool“ (DDT) von Digital Equipment Corporation (DEC) für PDP-1

Ablaufkontrolle (*run-control*)

- Setzen von **Haltepunkten** (*breakpoints*)
- **Einzelschritt**
- Starten/Stoppen

Inspektion

- Quelltextansicht
- **Programmzustand** (Register, Variablen)
- **Aufrufliste** (*call stack*)
- Speicher
- Threads / **Prozesse**

Geeignet für **imperative Sprachen**

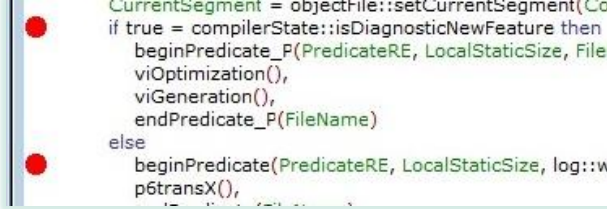
Erstmals 1961: „DEC Debugging Tool“ (DDT) von Digital Equipment Corporation (DEC) für PDP-1

Vorgehen

1. Aufstellen einer **These** über mögliche Position des Fehlers.
2. Setzen eines **Haltepunkts** vor der vermuteten Position.
3. **Annäherung** mit Hilfe von Breakpoints / Einzelschritt, dabei Überprüfung des Programmzustands.
4. These falsch? Zurück zu Schritt 1.
Ansonsten: **Korrigieren** des Fehlers.

Möglichkeiten im Debug

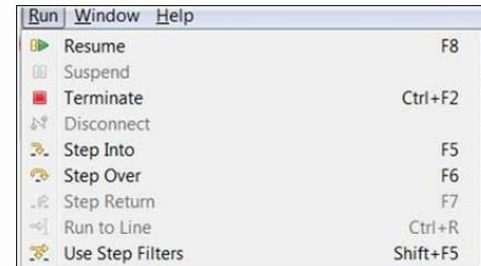
- Anhalten an **Breakpoints**
- Anschauen des **aktuellen Wertes** von Variablen



```
CurrentSegment = objectfile::setCurrentSegment(Co
if true = compilerState::isDiagnosticNewFeature then
beginPredicate_P(PredicateRE, LocalStaticSize, File
viOptimization(),
viGeneration(),
endPredicate_P(FileName)
else
beginPredicate(PredicateRE, LocalStaticSize, log::w
p6transX(),
```

Durchführung des Programms

- Step by Step
- Step Into
- Step Over



7.1.2 Compiler und Debugger

Beispiel: Debugger

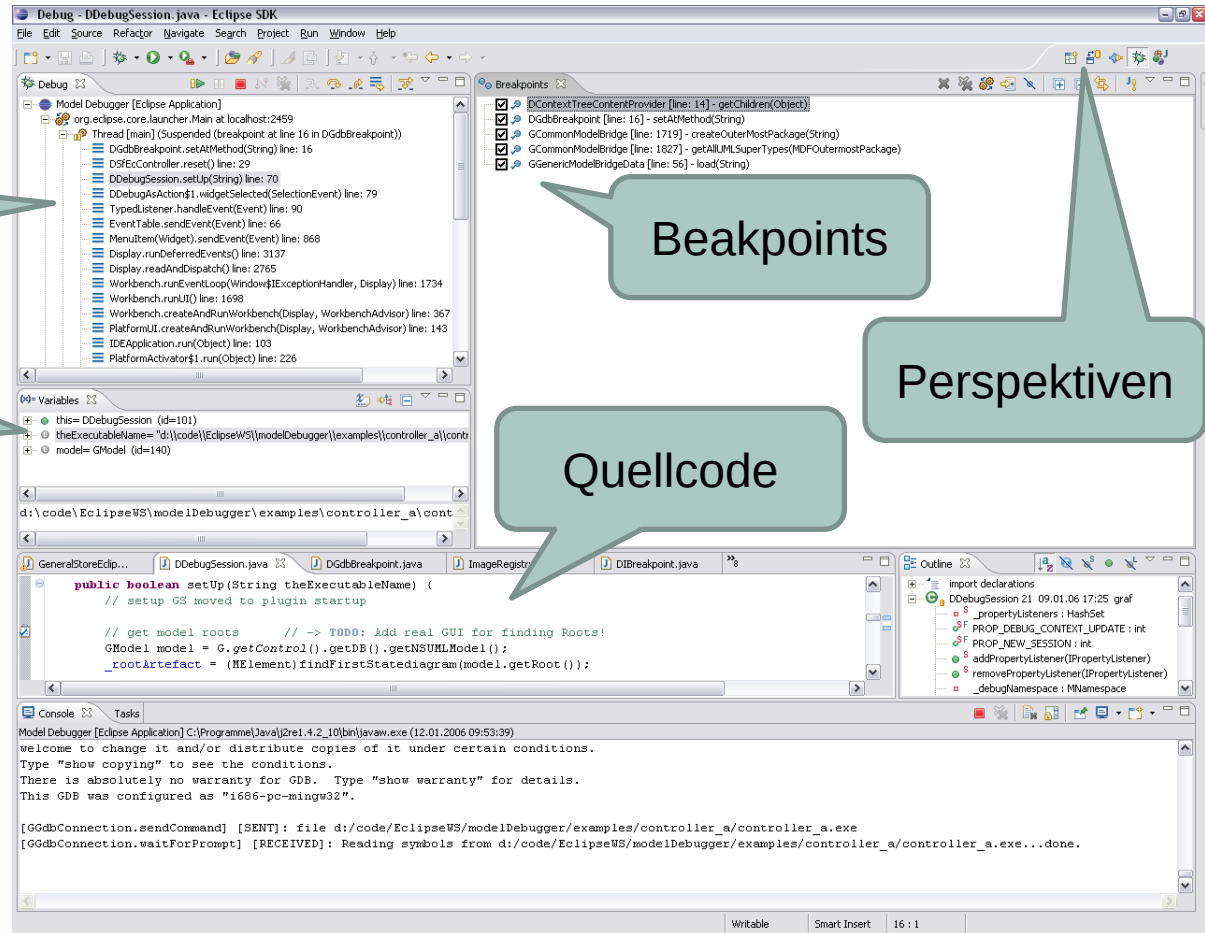
Aufruferliste

Variablen

Beakpoints

Perspektiven

Quellcode

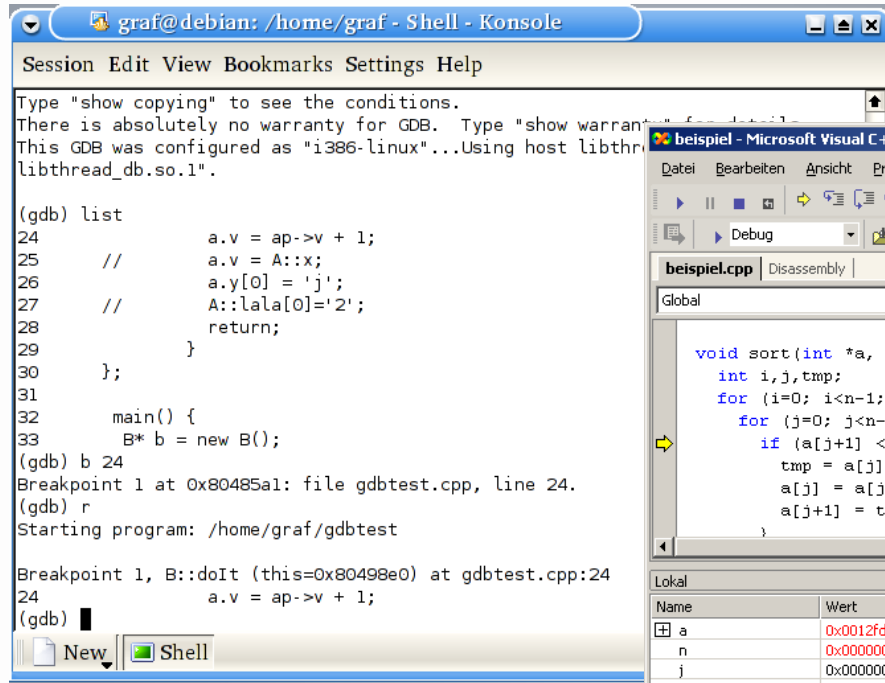


Integrierte Entwicklungs- umgebung (IDE)

- Beispiele
 - Eclipse (Open Source)
 - IntelliJ (JetBrain)
 - Visual Studio (Microsoft)

7.1.2 Compiler und Debugger

Klassisches Debugging - Werkzeuge

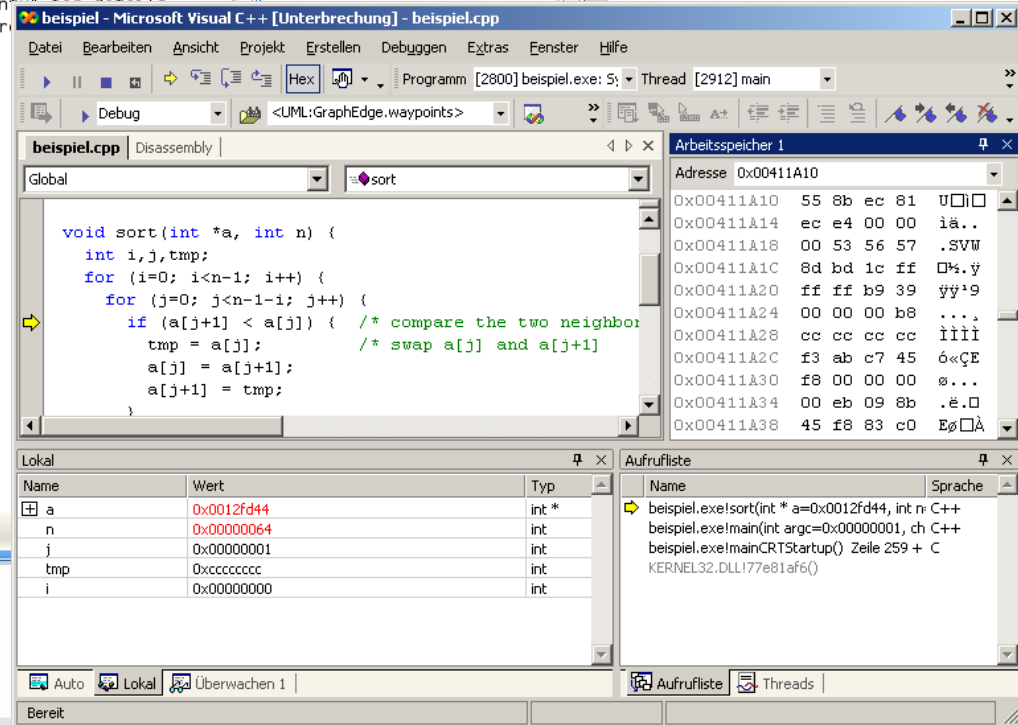


```
graf@debian: /home/graf - Shell - Konsole
Session Edit View Bookmarks Settings Help

Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warrant
This GDB was configured as "i386-linux"...Using host libthr
libthread_db.so.1".

(gdb) list
24         a.v = ap->v + 1;
25         //      a.v = A::x;
26         a.y[0] = 'j';
27         //      A::lala[0]='2';
28         return;
29     }
30 };
31
32     main() {
33         B* b = new B();
(gdb) b 24
Breakpoint 1 at 0x80485a1: file gdbtest.cpp, line 24.
(gdb) r
Starting program: /home/graf/gdbtest

Breakpoint 1, B::doIt (this=0x80498e0) at gdbtest.cpp:24
24         a.v = ap->v + 1;
(gdb) █
```



beispiel - Microsoft Visual C++ [Unterbrechung] - beispiel.cpp

Arbeitspeicher 1

Adresse	0x00411A10	0x00411A14	0x00411A18	0x00411A1C	0x00411A20	0x00411A24	0x00411A28	0x00411A2C	0x00411A30	0x00411A34	0x00411A38
	55 8b ec 81	ec e4 00 00	00 53 56 57	8d bd 1c ff	ff ff b9 39	00 00 00 b8	cc cc cc cc	f3 ab c7 45	f8 00 00 00	00 eb 09 8b	45 f8 83 c0
	U□□□	ia..	.SVW	□□.y	yy'9	...	iiii	6□ÇE	8...	.e.□	E□□A

Lokal

Name	Wert	Typ
a	0x0012fd44	int *
n	0x00000064	int
j	0x00000001	int
tmp	0xffffffff	int
i	0x00000000	int

Aufrufliste

Name	Sprache
beispiel.exe!sort(int * a=0x0012fd44, int n C++	
beispiel.exe!main(int argc=0x00000001, ch C++	
beispiel.exe!mainCRTStartup() Zeile 259 + C	
KERNEL32.DLL!77e81af6()	

Klassisch

Stoppen des Programms und
Inspektion des Zustands

Probleme

- Wann und von wem wurde **Zustand** (Variable, Register...) verändert?
- Zu weites Durchlaufen des Programms bedingt **Neustart**
- **Eingebettete Systeme**: Peripherie und Umgebung läuft weiter

Trace

Aufzeichnen des Programmablaufs

- **Rekonstruktion** des Systemzustands in Vergangenheit möglich

Vorteile

- **Keine Unterbrechung** des Programmablaufs
- „**Rückwärts**“-Debugging
- Untersuchung des Trace **auch Offline** möglich („Post-Mortem“)

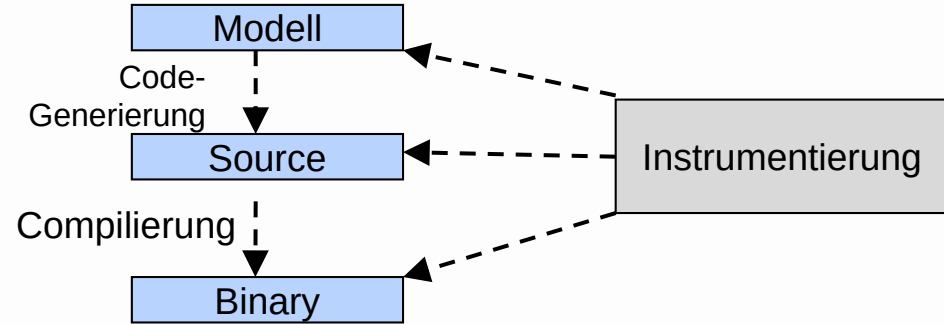
Dedizierte Hardware



iSystem iC4000
ActiveEmulator

- Überwacht Pins
- Nutzt On-Chip Schnittstellen (NEXUS 5001, BDM, JTAG)
- + Echtzeitfähig
- + Keine Beeinflussung des Programms
- Oft sehr teuer
- Bandbreite endlich

Instrumentierung



- Modell, Source oder Binary
- Statisch oder dynamisch
- + Beliebige Aquisition von Kontrollfluss und Daten
- Overhead Speicher und Performanz
- Veränderung des Programms („Heisenberg“-Effekt)

Konzept: Rückwärts in der Zeit: Eigener *class-loader* instrumentiert JAVA-Bytecode

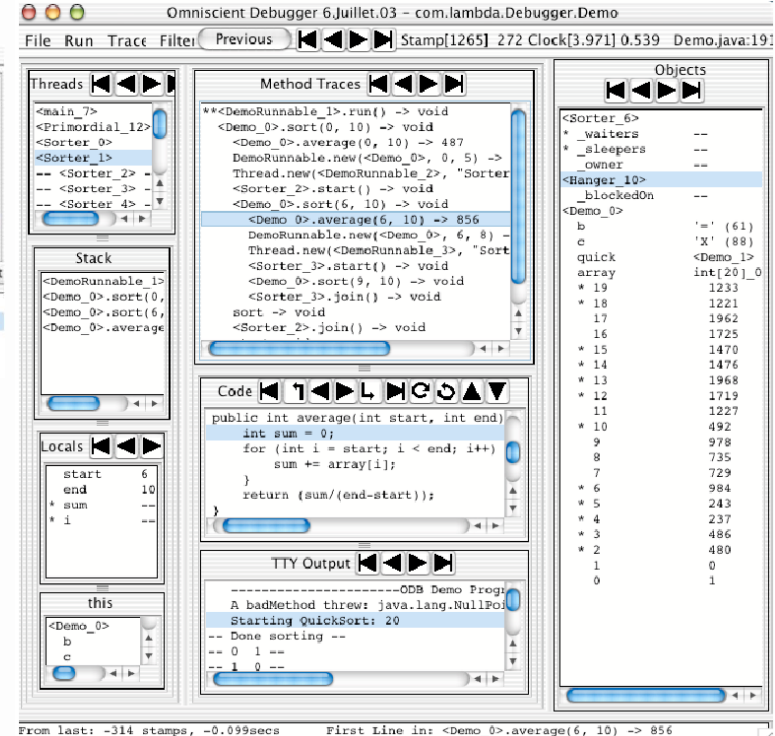
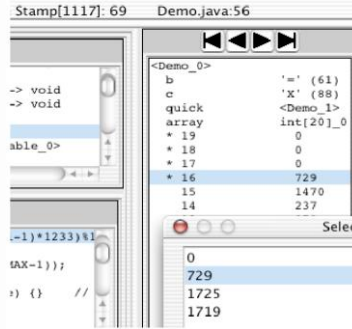
Protokolliert

- Kontrollfluss
- Variablenänderungen
- Umschalten zwischen Threads

Präsentation verknüpft

- Abfolge der Methodenaufrufe
- Quelltextansicht
- Überwachungsfenster
- Ausgabe

Abfolge der Werte einer Variable und Springen zum Zeitpunkt der Änderung



Begriff: Versionsverwaltung (Version Control)



Versionsverwaltung

Versionsverwaltung umfasst Aufgaben zur **Verwaltung verschiedener Versionen** von Dateien oder Programmen. Ziel ist **Rückverfolgbarkeit** der Änderungen (**Revisionen**); ferner deren Vergleich, Wiederherstellung, und Zusammenführung.

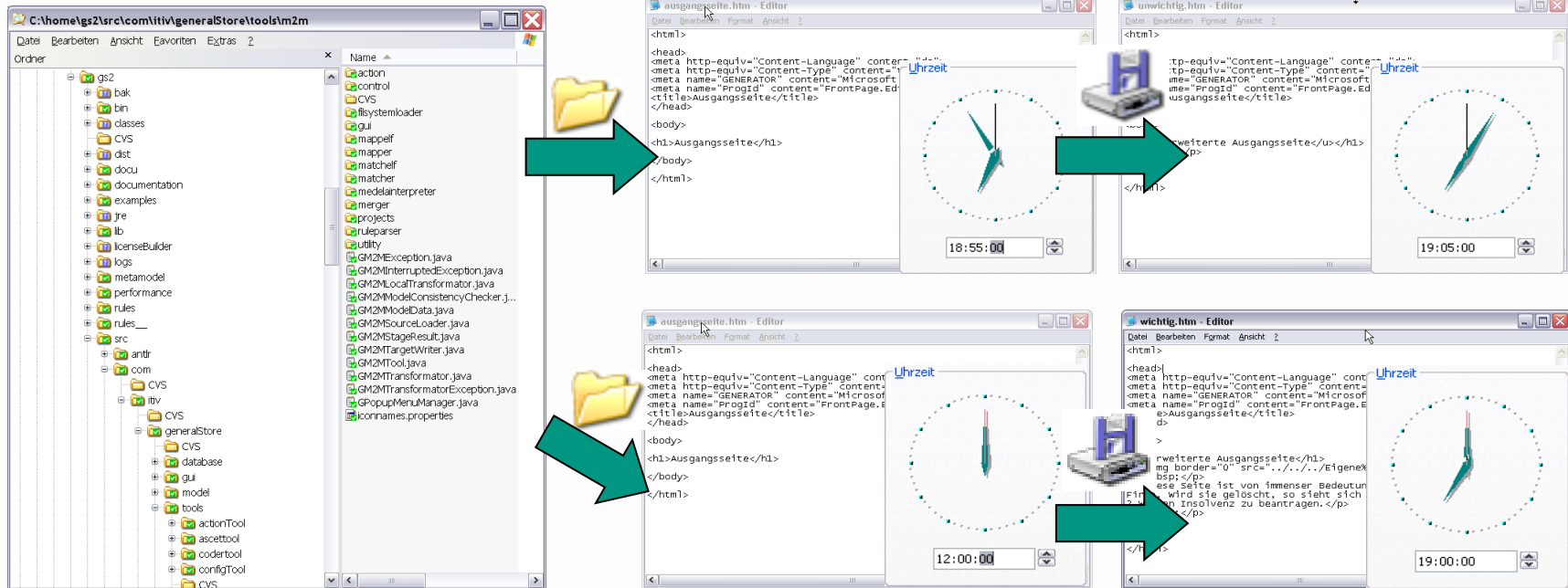


Version Control System (VCS)

Tool zur Versionsverwaltung, mit dem Revisionen systematisch in einem **Repository** abgespeichert werden können. Es bietet Funktionen, mit denen mehrere Entwickler kontrolliert auf verschiedenen Zweigen (Branches) parallel arbeiten können.

Wozu Quelltext Versionsverwaltung?

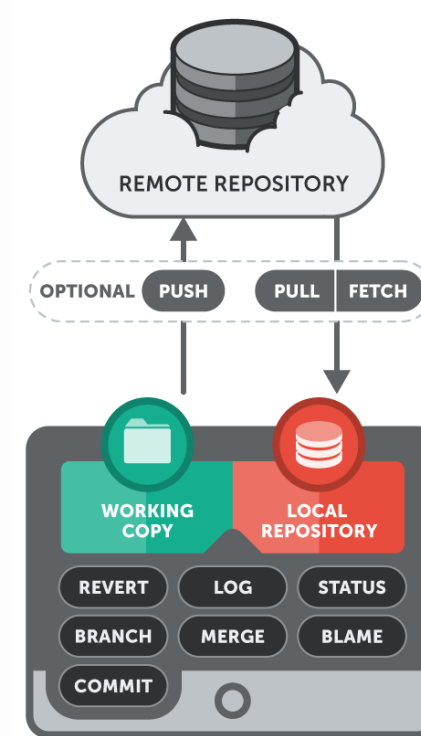
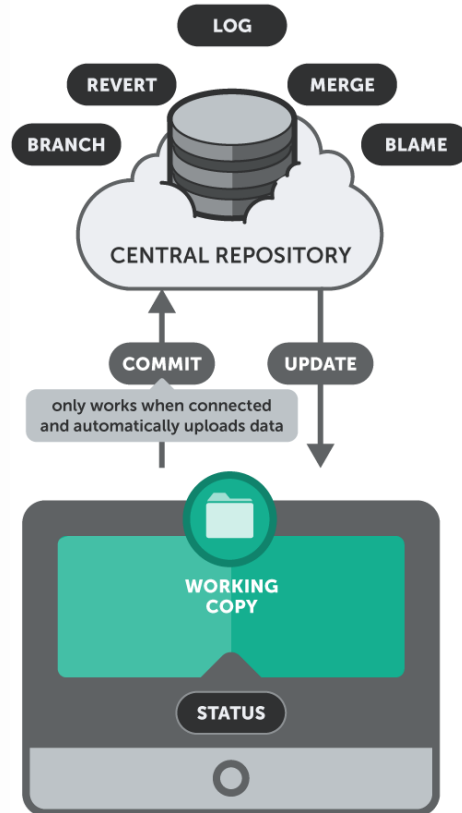
- Koordination im Team
- Parallele Entwicklung mehrerer Versionen
- Nachvollziehbarkeit
- Prozesssicherheit



Typen: Zentral vs dezentral

	Central VCS	Distributed VCS (DVCS)
Beispiel	SVN 	GIT 
Repository	nur <i>ein</i> zentrales (Central Repository)	- ein zentrales (Remote Repository) - zudem lokale Repos <i>für jeden Entwickler</i>
Branch	<i>Kopie</i> des Projekts in neuem Verzeichnis	<i>Verweis</i> auf eine bestimmte Revision ➤ auch lokal verfügbar
Checkout	erzeugt Snapshot <i>aktueller</i> Version (Working Copy : latest Version)	erzeugt lokale Kopie <i>des Repos</i> (Clone : mit Historie)
Commit	Sofort auf Central Repository ➤ nur mit Verbindung möglich	zunächst auf lokales Repository ➤ unabhängig von Remote Repository
	Änderungen <i>alle</i> auf einmal	Änderungen auswählen auf Staging Area ➤ Commits von Teil-Änderungen möglich
Versionierung	Commit erhöht Revisions-Nummer	Commit erzeugt Commit Hash (→ ID)

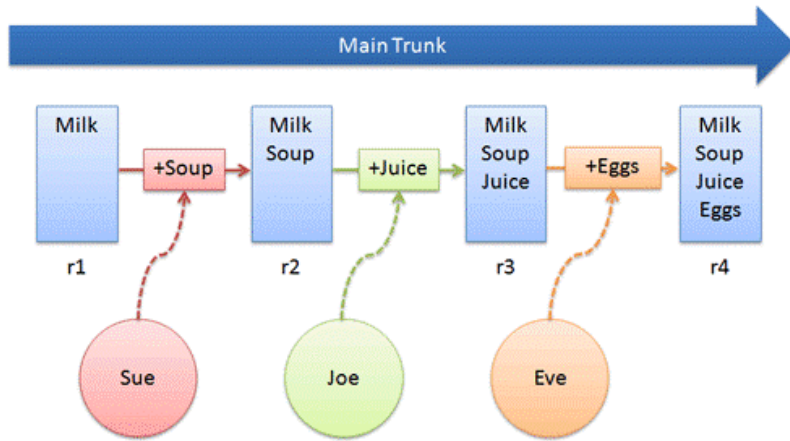
Typen: Zentral vs dezentral (II)



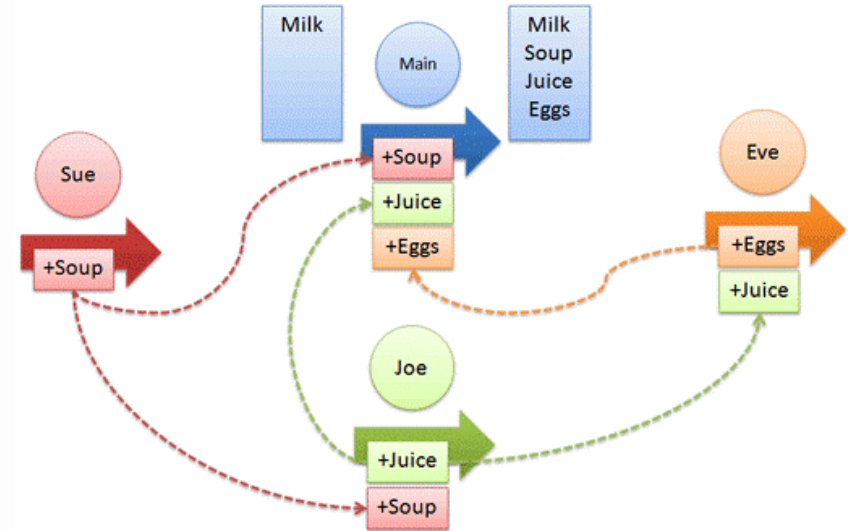
Quelle:
<https://www.git-tower.com/learn/git/ebook/en/command-line/appendix/from-subversion-to-git>

Beispiel: Zentral vs dezentral

zentral



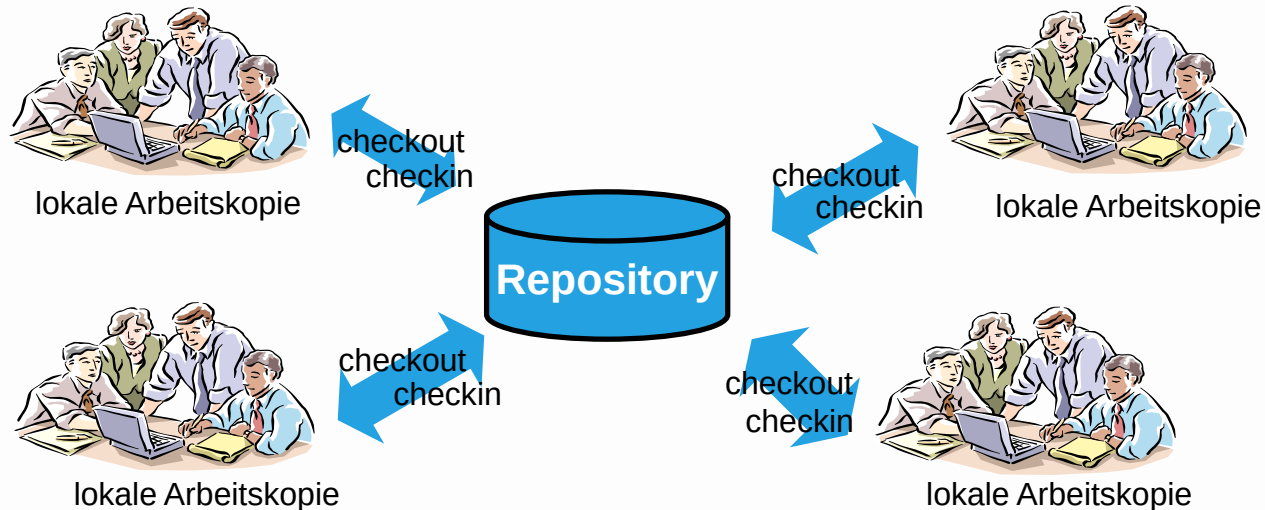
dezentral



Quelle: <https://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/>

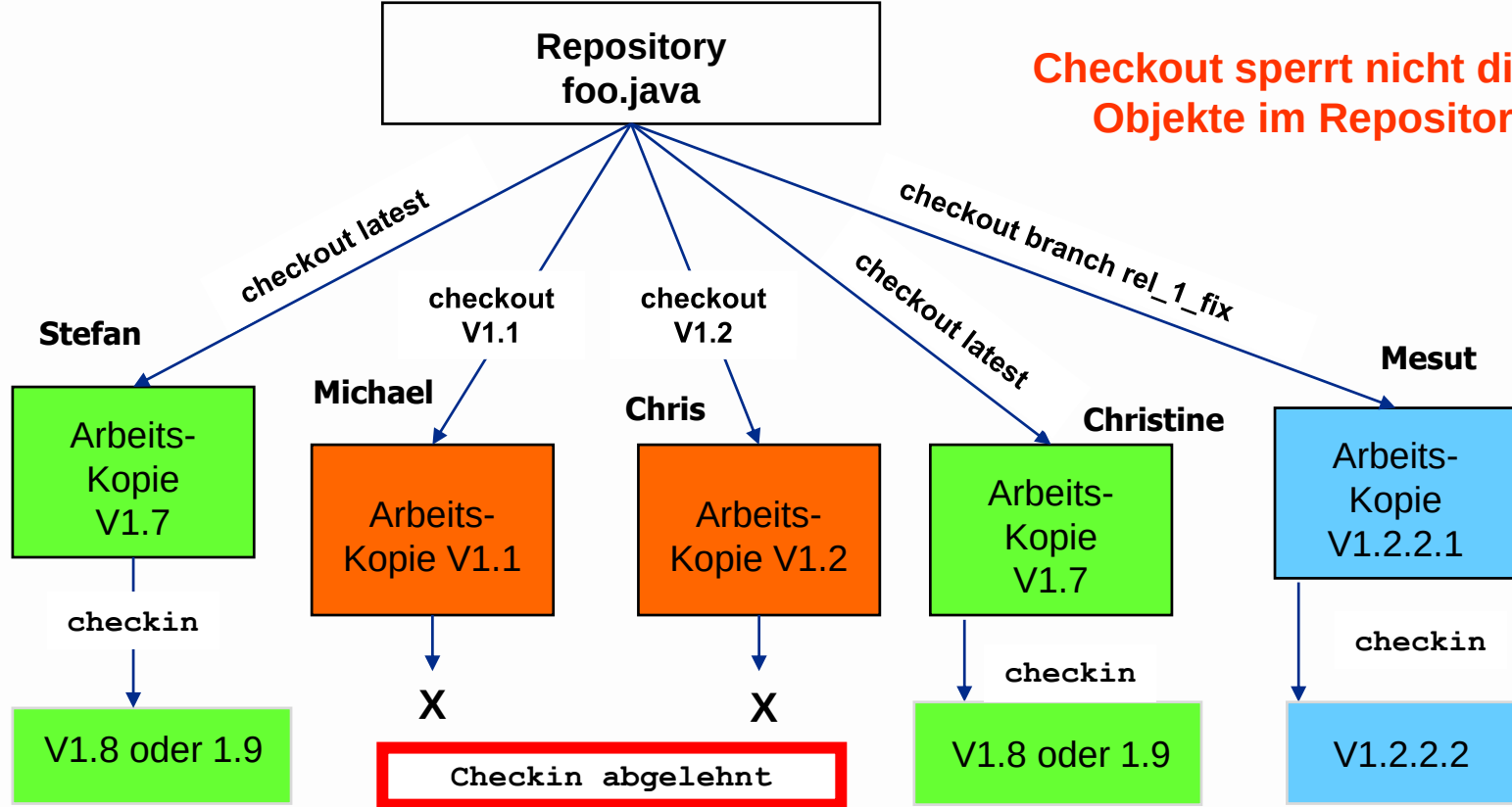
Der Ansatz von SVN (Concurrent Versions System)

- Zentrales Repository
- Beliebige Versionen abrufbar
- Jeder Entwickler hat eine lokale Arbeitskopie (**checkout**)
- Parallele Änderungen möglich (**concurrent**), Konfliktbehandlung beim **checkin**

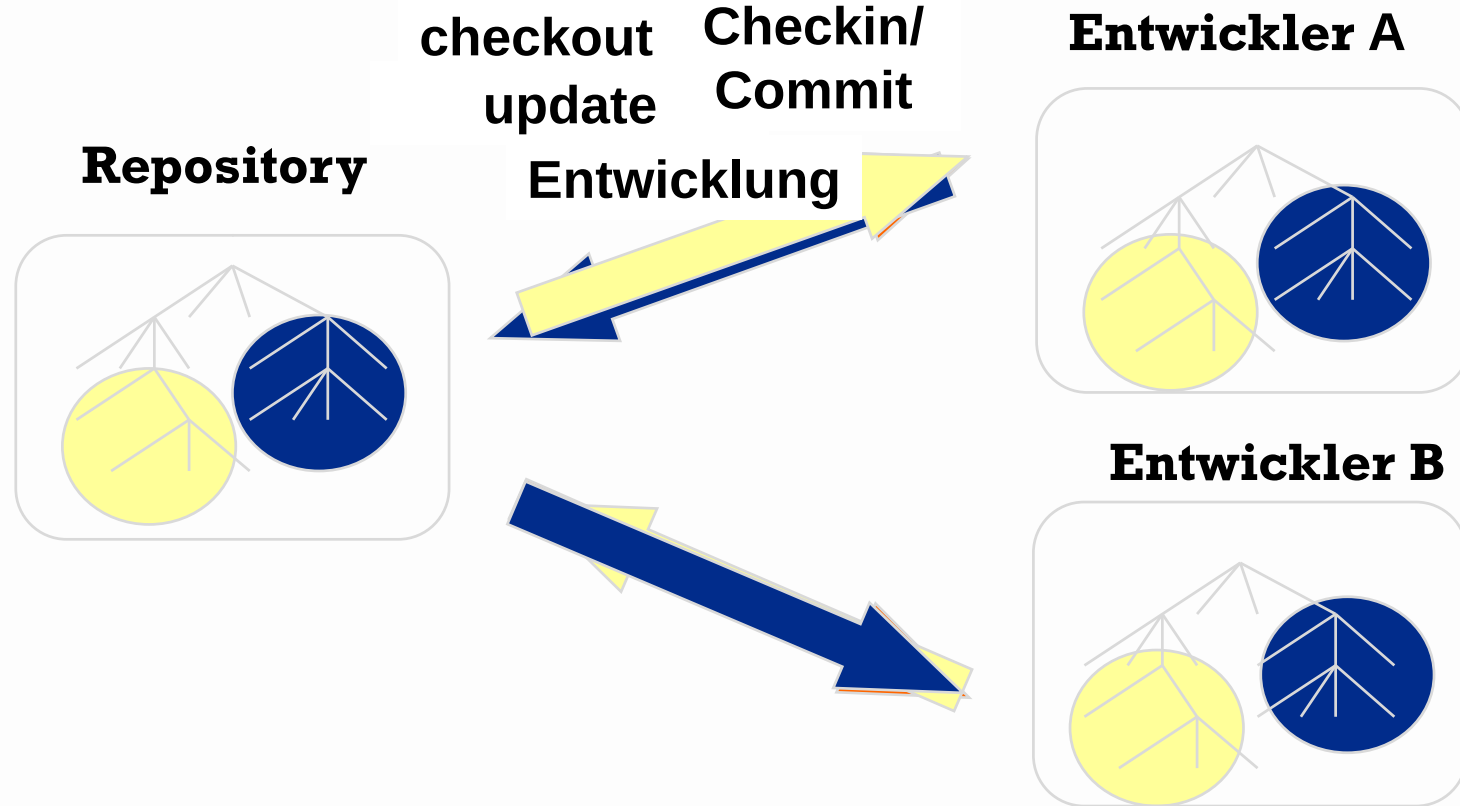


Gleichzeitiger Checkout (zentrales Repository)

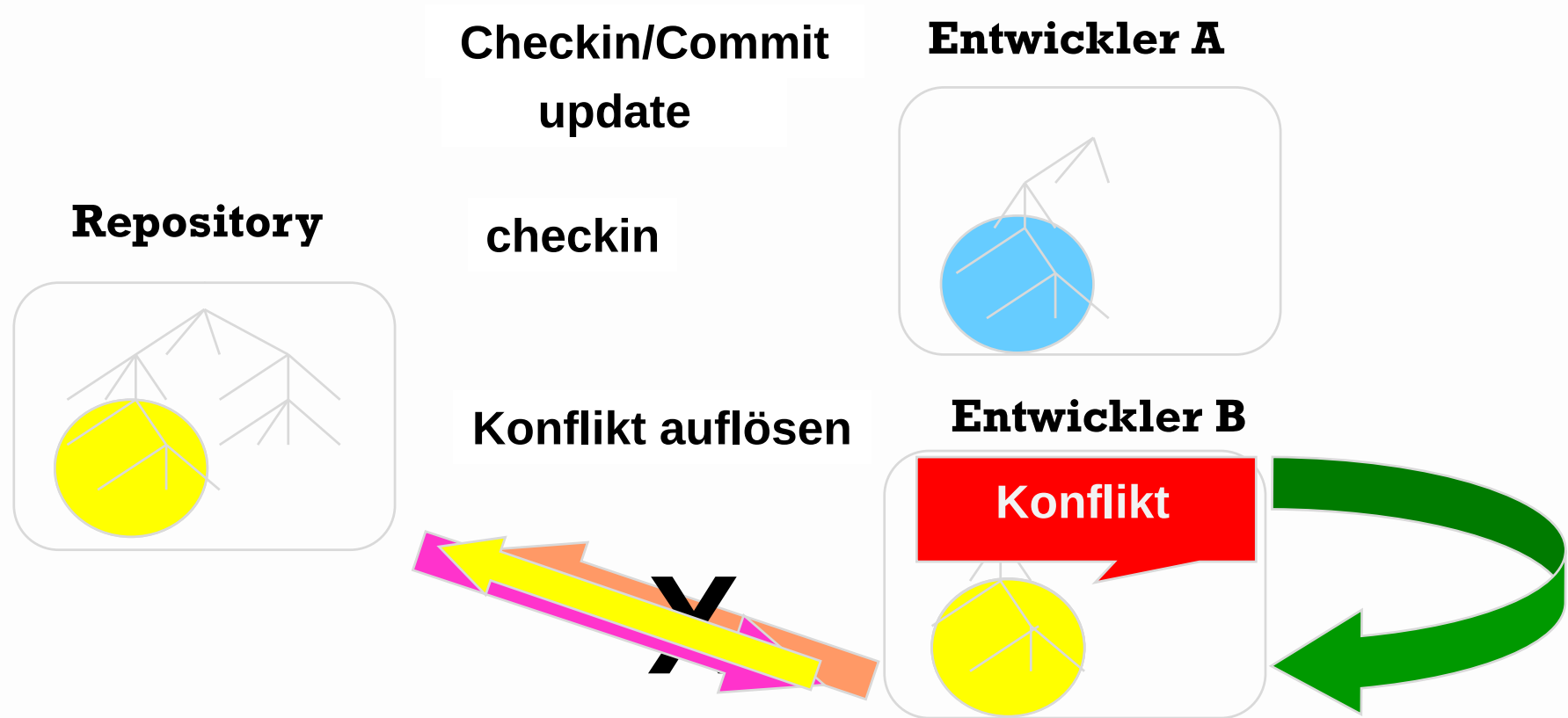
Checkout sperrt nicht die
Objekte im Repository



Ideale Entwicklung mit zentralem VCS



Reale Entwicklung mit zentralen VCS



Checkout

- Erstellt eine private Kopie der Dokumente innerhalb des lokalen Arbeitsverzeichnisses.
- Die Position ist beliebig
- Mehrere lokale Kopien mit unterschiedlichen Versionen sind möglich

Commit

- Übermittelt am Ende der Arbeit die Änderungen an den Dokumenten an den Server
- Die lokalen Kopien müssen die aktuellen Versionen sein

Update

- Vergleicht die lokalen Kopien mit dem Repository und aktualisiert sie bei Bedarf

1. Dateien im lokalen Arbeitsverzeichnis “auschecken”
2. Dateien editieren
3. Testen (lokaler Build, Syntaktische Analyse, ...)
4. Aktualisierung der lokalen Kopien mit **update**
bei werden die Änderungen der anderen Entwickler eingefügt (wenn nötig)
5. Nochmal testen, wenn in Schritt 4 patch/merge notwendig war
6. Änderungen übermitteln mit **commit**
7. Ab Schritt 2 wiederholen, bis zur nächsten Release
8. Release markieren (**tag/branch**, auch nachträglich möglich)
9. Modulnamen und Release-Tag/Branch zum System-Build übermitteln

- »Checke nur **kompilierbaren und getesteten Code** ein – die anderen werden es dir danken!«
- »Mache **vor einem Commit noch ein Update**, und achte auf zu lösende Konflikte.«
- »Führe **regelmäßig** Commit aus, und gib **sinnvolle Log-Einträge** an.«
- (...)
- »Denke immer daran, dass andere von deiner Arbeit abhängig sind.«



Systeme zur Versionsverwaltung



OpenCM



SVK



CMSynergy



Fragen?

Debugging

SVN



zentral/
dezentral

Version
Control
Systems (VCS)

GIT

Problem

Komplexes Programm besitzt etliche Komponenten, die **ständigen Änderungen** unterliegen, aber wieder zusammenfließen müssen.

Je **mehr Zeit** und Änderungen dabei zwischen zwei Releases liegt, desto höher und unvorhersehbarer die Art und Anzahl der **Fehler**.

Lösung

Qualitätssicherung: Neue Features und Anpassungen an der Software **schnell**, **sicher** und **nachhaltig** einspielen können.

- Regelmäßig neuer, funktionaler Build
- **Streamlining des Deployment-Prozesses**

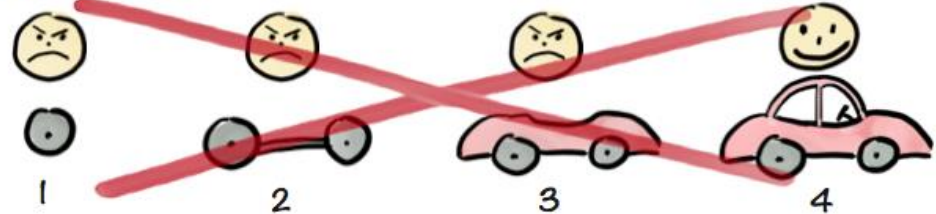


„Minimum Viable Product“

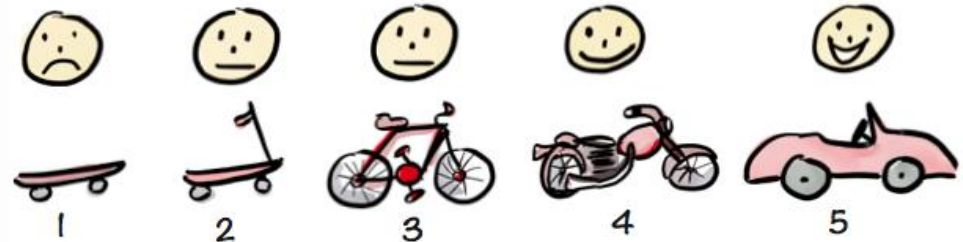
Ergebnisse **möglichst früh**
und **regelmäßig** liefern können
→ **schnelles Feedback**

- ✓ Kundenzufriedenheit
- ✓ Fortschrittskontrolle
- ✓ Geringere Risiken
- ✓ „Kinderkrankheiten“ erkennen

Not like this....



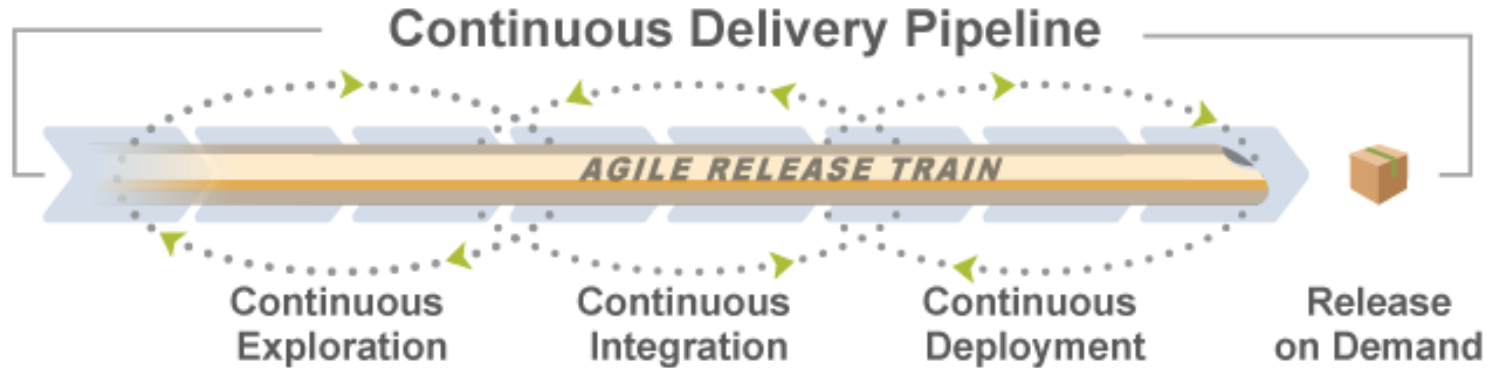
Like this!



Quelle: <http://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp>

Begriff: Continuous Delivery (CD)

! *Continuous Delivery (CD)* ist eine Disziplin der SE, bei der SW so gebaut wird, dass sie zu einem beliebigen Zeitpunkt veröffentlicht und in Produktion gegeben werden kann.



© Scaled Agile, Inc.

Vorteile

- ✓ Kleine und schnelle Iterationen
 - ✓ Schnelle und einfachere Fehlerentdeckung
 - ✓ Schnelleres Feedback
 - ✓ Bug-Fixes können schneller angewandt werden
 - ✓ Inkrementelles Entwicklungsmodell
 - ✓ Zufriedenere Entwicklungsteams
-
- Höhere Qualität
 - Geringere Kosten
 - **Insgesamt besseres Produkt**



Server-Teilung: Testumgebung → Produktivumgebung (I)

Staging-System

Testumgebung für die Entwickler unter annähernd realen Bedingungen

- Überprüfen der neuen Features und Anpassungen

Build-Automatisierung

- Automatisch erzeugt aus Quellcode auf CI-Server (CI=Continuous Integration)
- Ausführbare Version aus gewünschten Komponenten
- Sofortige (Mindest-)Prüfung mit automatischen Tests

Änderungen

Mehrere täglich durch die Entwickler

Server-Teilung: Testumgebung → Produktivumgebung (II)

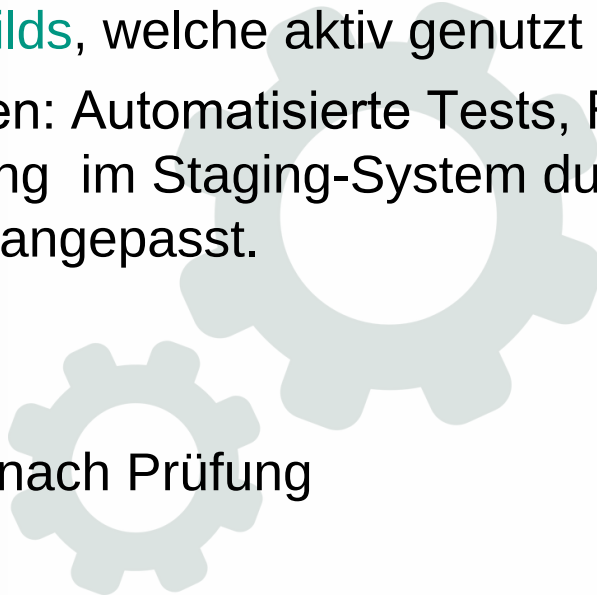
Produktiv-System

Umgebung für **stabile Builds**, welche aktiv genutzt werden

- ‚Feuerprobe‘ bestanden: Automatisierte Tests, Regressions-Tests, und allgemeine Verwendung im Staging-System durch Entwickler lief fehlerfrei, bzw. wurde angepasst.

Änderungen

Nur durch Server-Admin nach Prüfung



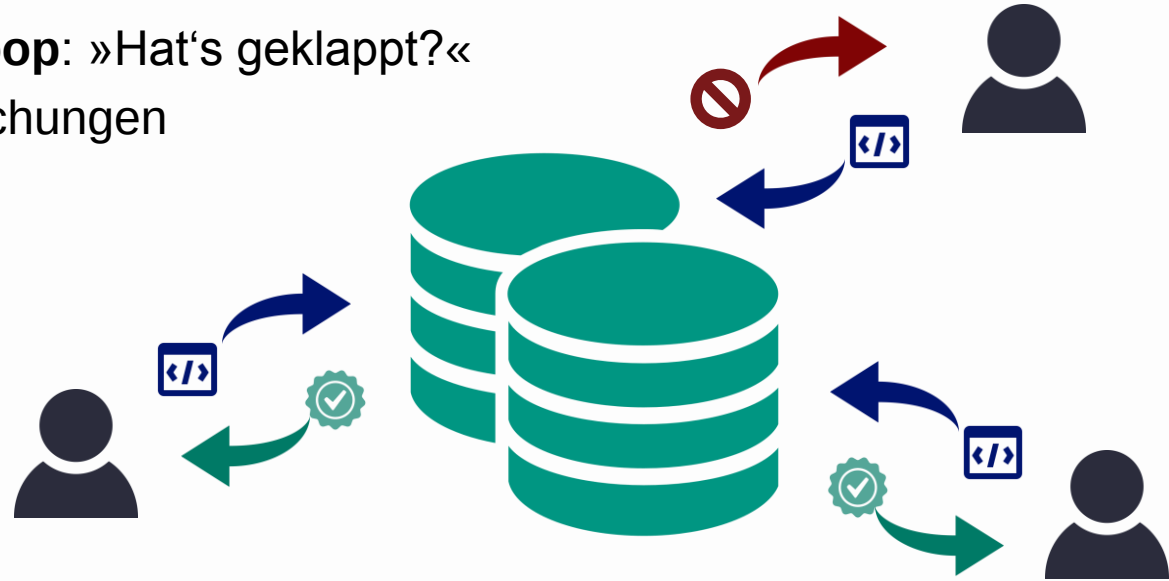
Server-Teilung (Build Übersicht)

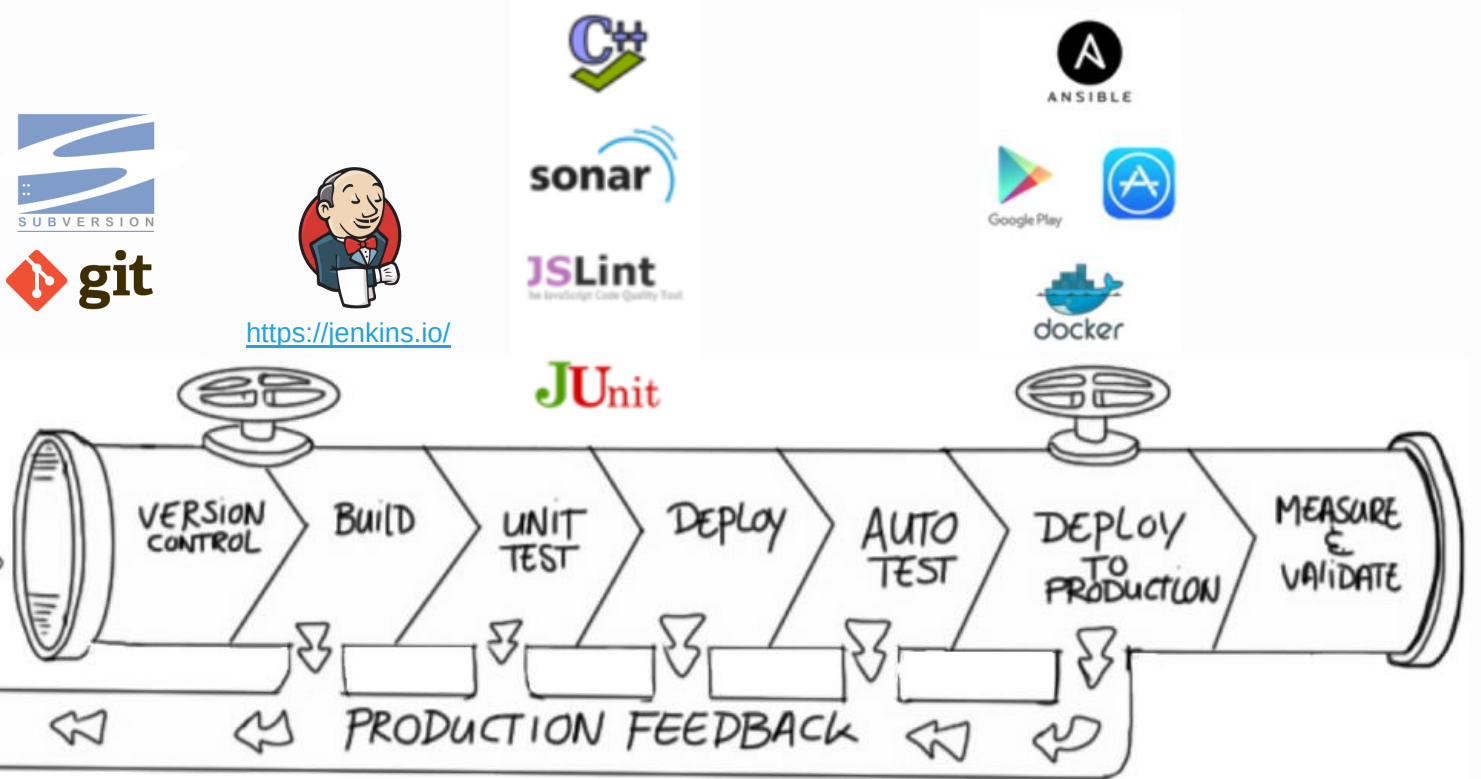
	CI Build	Release Candidate	Release
Automatisierte Tests	✓	✓	✓
Regression Tests	X	(✓)	✓
Live Nutzung geprüft	X	(✓)	✓
Stabilität	oft instabil	keine <i>Show-stopper</i>	möglichst hoch
Deployment	mehrere täglich	~wöchentlich	~quartalweise
Vorlage	<i>gewählter Quellcode</i>	aus stabilem CI Build	aus finalem RC
Nutzung	Lokal	Testserver	Liveserver
Zweck	Neueste Änderungen prüfen	System unter realen Bedingungen testen	Offizielles Kundenprodukt

CI=Continuous Integration; RC=Release Candidate

Shared Repository

- Automatische Verifizierung mit jeder Integration
- Automatische Tests und automatische Builds
- Schneller **Feedback-Loop**: »Hat's geklappt?«
- Entfernen der Unterbrechungen





Quelle: <http://xebia.github.io>

GitLab

- Webbasiert wie GitHub, mit noch mehr Features:
- Flow-based, mit merge-Requests
- Hat Code-Reviews
- Hat eingebauten CI-Server („Integrated CI/CD“)
- Multiple Runners, Parallele Builds



<https://about.gitlab.com/>

Jenkins

- Webbasierter, erweiterbarer automatisierungs-Server
- Viele Plugins, breite Unterstützung an Build-Tools
 - SVN, Subversion, GIT
 - Junit für automatisierte Tests



<https://jenkins.io/>

Beispiel: Jenkins

Jenkins Suchen Anmelden

Jenkins > v75x > [AUTO-AKTUALISIERUNG EINSCHALTEN](#)

Benutzer
Build-Verlauf
Projektbeziehungen
Fingerabdruck überprüfen
Build History
Claim Report
Disk Usage
Job Import Plugin
Plugin Usage
Abhängigkeitsgraph

Build Warteschlange —
Keine Builds geplant

Build-Prozessor-Status —

master
1 Ruhend
2 Ruhend
3 Ruhend
4 Ruhend
Unbekannter Task
Unbekannter Task
Unbekannter Task
Unbekannter Task

1 Morphheus PREvision-trunk #50558 ■
1 TRINITY_Sync_AR-Test-Cluster V8.0.x #329 ■

[Hier geht's zum Jenkins Stuttgart...](#)
[Hier geht's zum Jenkins Backup-Restore...](#)
[Hier geht's zum Jenkins Testsystem...](#)

Admin Alle CI Intertool JUnit JaCoCo M2R Morpheus Morpheus_Vx Platforms RCPTT TRINITY v65x v70x **v75x** v80x v85x z_Executor z_Executor_to_clone zz_EXP

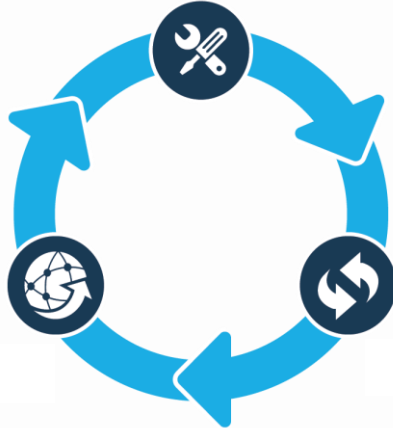
S	W	Name	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer	Nächster Start
●	☀	AUTOSAR PREvision -> CANoe v75x	6 Monate 27 Tage - #360	10 Monate - #356	2 Minuten 54 Sekunden	
●	☀	AUTOSAR PREvision -> DaVinci CFG.GM.v75x	6 Monate 27 Tage - #379	6 Monate 28 Tage - #375	21 Minuten	
●	☀	AUTOSAR PREvision -> DaVinci DEV.v75x	6 Monate 27 Tage - #373	10 Monate - #367	3 Minuten 1 Sekunde	
●	☀	Core.v75x	9 Monate 12 Tage - #372	1 Jahr 0 Monate - #261	3 Stunden 43 Minuten	
●	☀	Middletier Build with Test Plugins v75x	20 Stunden - #421	Unbekannt	8 Stunden 55 Minuten	17/12/2017 19:25
●	☀	Middletier CI Build v75x	26 Tage - #1482	Unbekannt	17 Minuten	
●	☀	Middletier CI Build_for_Test v75x	26 Tage - #1787	Unbekannt	21 Minuten	
●	☀	MMedit CI Build v75x	26 Tage - #1616	Unbekannt	36 Minuten	
●	☀	PREvision 3tier TestFW v75x	21 Stunden - #426	Unbekannt	5 Stunden 41 Minuten	17/12/2017 18:45
●	☀	PREvision AUTOSAR Im-/Export v75x	21 Stunden - #397	Unbekannt	5 Stunden 12 Minuten	17/12/2017 18:45
●	☀	PREvision Basic Test v75x	22 Stunden - #414	Unbekannt	5 Stunden 12 Minuten	17/12/2017 18:15
●	☀	PREvision CI Build 64 Bit v75x	26 Tage - #1748	Unbekannt	53 Minuten	
●	☀	PREvision DB Tests v75x	21 Stunden - #412	Unbekannt	13 Stunden	17/12/2017 18:45
●	☀	PREvision Migration v75x	19 Stunden - #409	Unbekannt	6 Stunden 38 Minuten	17/12/2017 19:52
●	☀	PREvision Misc Test v75x	22 Stunden - #448	2 Monate 2 Tage - #431	9 Stunden 39 Minuten	17/12/2017 17:30
●	☀	PREvision Modelimport/-export v75x	6 Tage 23 Stunden - #413	23 Stunden - #414	3 Stunden 31 Minuten	17/12/2017 17:00
●	☀	PREvision Care_Test v75x	18 Stunden - #410	Unbekannt	3 Stunden 3 Minuten	17/12/2017 22:00

Continuous Integration (CI)

! *Continuous Integration (CI)* ist eine Teildisziplin der CD. Sie beabsichtigt das regelmäßige **Integrieren von Code** zu einem Programm (auf einen Trunk), sowie dessen **Validierung**.

Prinzip

1. Code it
2. Build it
3. Test it

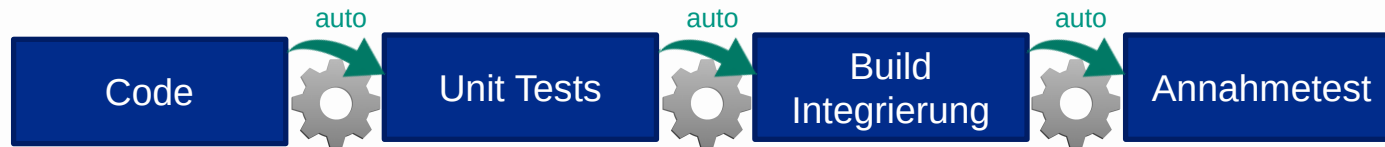


Tests einpflegen

- alle automatischen Tests (Unit Tests)
- zudem wählbar: Regression Tests

Anforderungen des CI

- **Shared Repository**: Zentraler Code, auf den alle Zugriff haben
- **Regelmäßige Commits**: Entwickler tätigen mindestens täglich ein Check-In ihres Fortschritts (Code wird bei jedem Check-In gebaut)
- Hoher **Automatisierungsgrad** (Deployment von Builds und Tests)
- Hohe **Geschwindigkeit** (*schnelle* Builds und Tests)



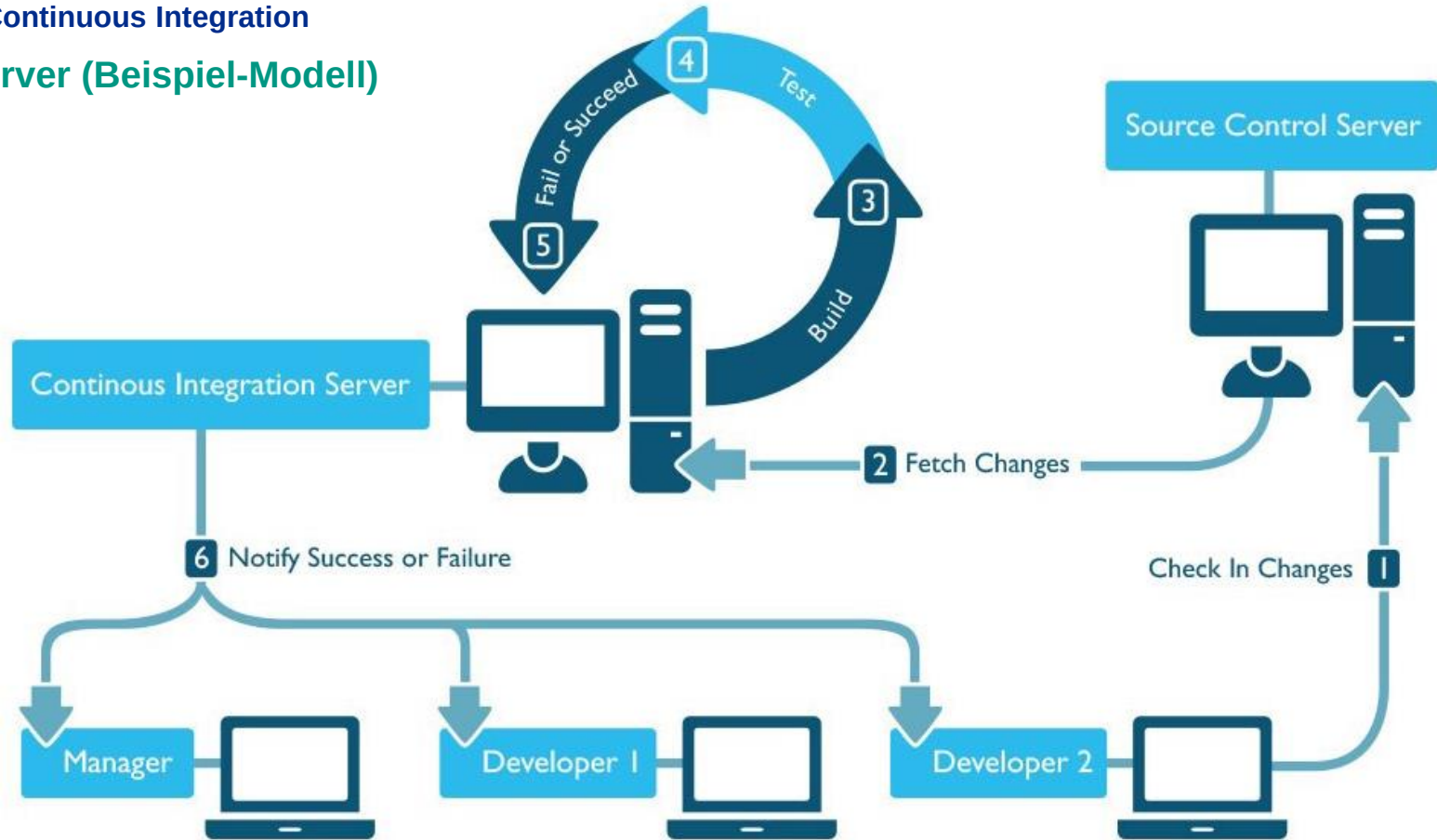
Anforderungen des CI (cont.)

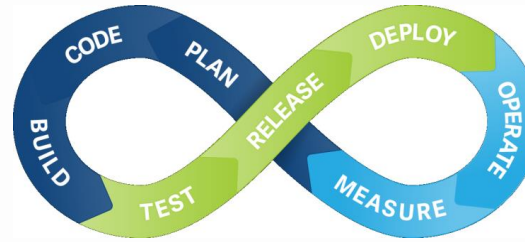
- (Herunter-)Skalierbarkeit der Umgebungen: Zum Teilbereiche testen
- Reproduzierbarkeit: Jeder hat Zugriff auf alle Builds und Reports
 - Alter Stand muss wieder abrufbar sein (sog. „Legacy Modelle“)
 - Produktumgebung muss beim Testen genau übereinstimmen
- Beispiel: Kunde nutzt alten Software-Stand und berichtet einen Bug. Tester benötigen denselben Build der Software für die Fehler-‘Repro‘.



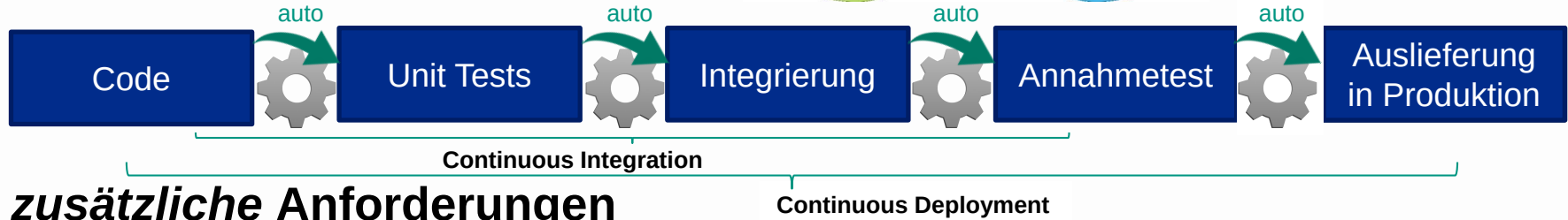
7.2.1 Continuous Integration

CI-Server (Beispiel-Modell)





➤ Verfeinerung von CI



zusätzliche Anforderungen

- Auslieferbarkeit der Software während gesamtem Lebenszyklus
 - stetige Feature-Umsetzung (durch agiles Vorgehen)
- Auslieferbarkeit jeder Version per Knopfdruck
 - Umfassende Automatisierung mit Deployment Routine
- **Dev-Ops-Kultur:** Kollaboration zwischen Auslieferern und Entwicklern

Continuous
Integration (CI)

Continuous
Deployment
(CD)



Shared
Repository

Programmier-Richtlinien

```

9      }
10      if (_pinNames2PortNumber.containsKey(thePinName)) {
11          portNumber = (String) _pinNames2PortNumber.get(thePinName);
12      } else if (thePinName.equals("get/get")) {
13          _pinNames2PortNumber.put(thePinName, "1");
14          portNumber = (String) _pinNames2PortNumber.get(thePinName);
15      } else if (thePinName.equals("set/aValue")) {
16          /*_pinNames2PortNumber.put(thePinName, "1");
17          portNumber = (String) _pinNames2PortNumber.get(thePinName); */
18      } else if (thePinName.equals("getAt/getAt")) {
19          _pinNames2PortNumber.put(thePinName, "1");
20          portNumber = (String) _pinNames2PortNumber.get(thePinName);
21      } else if (functionalElement.getAscetModelElement().toString().indexOf("ComplexModelElement") > -1) {
22          {
23      com::itiv::generalStore::tools::ascetool::ascet12::blockdiagram::MBlockDiagramElementPin pin = { com::itiv
24          if (pin.isInput()) {
25              if (_block2ListOfInputPins == null) {
26                  _block2ListOfInputPins = new HashMap();
27              }
28          /*
29              if ("C".equals(functionalElement.getAscetModelElement().getCodeComponent().getComponentType().toStrin
30              com::itiv::generalStore::tools::ascetool::ascet12::database::MCodeComponent cc1 = (com::itiv::genera
31              if ("C".equals(cc1.getComponentType().toString())) {
32                  if (_block2ListOfInputPins.containsKey(theMSDElement)) {
33                      java::util::List pinNameList = (java::util::List) _block2ListOfInputPins.get(theMSDElement);
34                      if (pinNameList.indexOf(thePinName) != -1) {
35                          pinPos = pinNameList.indexOf(thePinName);

```

Warum Programmier-Richtlinien?

- Verbessert die **Lesbarkeit** und deshalb die Wartbarkeit von Quellcode
- **Beschleunigt Einarbeitung** bei Personalwechsel und Wiedereinarbeitung
- Erleichtert das **gemeinsame Arbeiten** von verschiedenen Programmierern an einem Projekt
- Wenn der **Quellcode als Produkt** ausgeliefert wird, sollte dieser genauso verpackt und aufgeräumt sein **wie jedes andere** Produkt
- **Zeitersparnis bei Fehlerfindung**
- Erweiterung und Pflege des Programms

→ **Aber:** Nicht übertreiben! Alle Regeln mit Augenmaß anwenden.

Was wird festgelegt?

→ wird von IDE unterstützt

Formatierung (Layout)

Beispiel

```
    for (i=0;i<grenze;i++) foo(i) ;  
vs    for ( i=0; i<grenze; i++ ) foo( i );
```

Namens-Konventionen

- Z.B. Methoden, die ein Attribut lesen oder schreiben, beginnen mit *hole* oder *get* bzw. *setze* oder *set*
Beispiel: *getName*, *holeName*
- Sprechende Namen

Implementierungs-Dokumentation

- Vorspann
- Klassen Kurzbeschreibung
- Parameter Beschreibung
- Vorbedingungen

Beispiel: Einrückung in Schleifen

K&R (Kernighan & Ritchie) Stil,
oder „kernel style“ (Unix Kernel)

```
if (<Bedingung>) {  
    <Rumpf>  
}
```

„Whitesmiths Stil“

```
if (<Bedingung>)  
{  
    <Rumpf>  
}
```

„Allman Stil“ nach Eric Allman, auch
„BSD Stil“

```
if (<Bedingung>)  
{  
    <Rumpf>  
}
```

„GNU Stil“, nach GNU Emacs

```
if (<Bedingung>)  
{  
    <Rumpf>  
}
```

Werkzeuge

- Statische Code Analyse und Code Style Überwachung
 - **cppcheck** für C/C++
 - **checkstyle** für Java
 - **JSLint** für JavaScript
 - **SonarQube** Qualitätsbeurteilung
 - **Findbugs** zur Fehlermuster Suche

Examples of Coding Guidelines

Java:

www.geosoft.no/development/javastyle.html

www.horstmann.com/bigj/style.html

www.developer.netscape.com/docs/technote/java/codestyle.html

C++:

www.geosoft.no/development/cppstyle.html

www.oss.software.ibm.com/icu/userguide/conventions.html

C++/Java: agsrhichome.bnl.gov/Controls/doc/codingGuidelines/codingGuidelines.html

.NET: msdn.microsoft.com/library/default.asp?url=/library/enus/cpgenref/html/cpconnetframeworkdesignguidelines.asp

PERL and shell-level: www.dev.panopticsearch.com/portable_coding.html

PERL: www.lug.lk/~anu/misc/perl_coding_guidelines.txt

Wozu Code
Style?



einzeiliger Kommentar: *//* *...*

```
// der Kommentar geht bis zum Ende dieser Zeile  
String foo;
```

mehrzeiliger Kommentar: */** *...* **/*

```
/* Ab hier wird alles auskommentiert...  
String bar;  
...bis der Kommentar geschlossen wird                    */  
String baz;
```


JavaDoc ist ein Programm, das aus Java-Quelltexten **Dokumentationen** im HTML-Format erstellt.

Es befindet sich im sdk-Verzeichnis im Ordner `/bin`.



Class [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV](#) [NEXT](#)

Class Hierarchy

- class java.lang.Object
 - class [Sing](#)
 - class [vogelgesang](#)

Class [vogelgesang](#)

java.lang.Object
 |
 +--[vogelgesang](#)

public class [vogelgesang](#)
 extends java.lang.Object

Programm zum Bestimmen spezifischer Vogelgesänge
 zum Erstellen verschiedener Objekte.

Version:
 1.0 25 Apr 2002

Author:
 Tobias Fritsch

Constructor Detail

[vogelgesang](#)

public [vogelgesang](#)()

Method Detail

[main](#)

public static void [main](#)(java.lang.String[] args)

Hauptprozedur, die die eigentlichen Funktionsaufrufe macht.

Parameters:
 args - eingegebener Aufrufsstring

Class [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV](#) [NEXT](#)

[PREV CLASS](#) [NEXT CLASS](#)
 SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)
 DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Dokumentations-Kommentare

`/**`

`* beginnen immer mit /**`

`* und enden mit */`

- werden vor Klassen-, Variablen- und Methoden-Definitionen von dem Tool „javadoc“ interpretiert und **zum Erstellen einer Dokumentation** in Form von HTML-Dateien benutzt.
- `*`, Leerzeichen und Tabulatorzeichen am Zeilenanfang werden ignoriert. HTML-Text kann beliebig eingesetzt werden. Spezielle **Tags beginnen mit „@“** und werden besonders interpretiert.
- Der erste Satz enthält die **Kurzbeschreibung**. Der Rest die ausführliche Beschreibung.
- Können **HTML-Tags** außer `<H1>..<H6>` und `<HR>` enthalten.

Beispiel

```
/**
 * Methodenbeschreibung - Methodenbeschreibung - Methodenbeschreibung -
 * Methodenbeschreibung - Methodenbeschreibung - Methodenbeschreibung -
 * Methodenbeschreibung - Methodenbeschreibung - Methodenbeschreibung
 *
 * @return boolean
 * @param text1 - String der ausgegeben wird.
 * @param text2 - String dessen Länge überprüft wird.
 * @throws Exception
 * @see TestKlasse#printPicture
 * @deprecated Bitte verwenden sie nur noch Sys.out.println(String text).
 */
public boolean printText(String text1, String text2) throws Exception {
    System.out.println(text1);
    if (text2.length() >= 5) {
        throw new Exception();
    }

    return true;
}
```

JavaDoc Annotation „@return“

@return description

- vor Methoden-Definitionen
- Beschreibt den Rückgabewert einer Methode. Dafür sollten dessen Typ und Wertebereich angegeben werden.

Beispiel

```
* ...
* @return float - Nullstelle der Funktion f.
* ...
```



Returns:

float - Nullstelle der Funktion f.

```
/**
 * Methodenbeschreibung - Methode
 * Methodenbeschreibung - Methode
 * Methodenbeschreibung - Methode
 *
 * @return boolean
 * @param text1 - String der ausg
 * @param text2 - String dessen L
 * @throws Exception
 * @see TestKlasse#printPicture
 * @deprecated Bitte verwenden si
 */
public boolean printText(String t
    System.out.println(text
```

Method Summary

```
static int optionLength(java.lang.String option)
    Check for doclet added options here.
```



[java.applet](#)
 Interfaces
[AppletContext](#)
[AppletStub](#)
[AudioClip](#)
 Classes
[Applet](#)

[Overview](#)
[Package](#)
[Class](#)
[Use](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV](#)
[NEXT](#)

[FRAMES](#)
[NO FRAMES](#)

Java™ 2 Platform
 Std. Ed. v1.4.2

Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification

This document is the API specification for the Java 2 Platform, Standard Edition, version 1.4.2.

See:

[Description](#)

Java 2 Platform Packages

java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.

All Classes

[ClassDoc](#)
[ConstructorDoc](#)
[Doc](#)
[DocErrorReporter](#)
[Doclet](#)
[ExecutableMemberDoc](#)
[FieldDoc](#)
[MemberDoc](#)
[MethodDoc](#)
[PackageDoc](#)
[Parameter](#)
[ParamTag](#)
[ProgramElementDoc](#)
[RootDoc](#)
[SeeTag](#)
[SerialFieldTag](#)
[Tag](#)
[ThrowsTag](#)
[Type](#)

com.sun.javadoc

Class Doclet

```

java.lang.Object
|
+--com.sun.javadoc.Doclet

```

public abstract class **Doclet**
extends java.lang.Object

This class documents the entry-point methods in a Doclet. It may be used as the superclass of a doclet but this is not required.

Constructor Summary

Doclet()

Method Summary

static int	optionLength (java.lang.String option) Check for doclet added options here.
static boolean	start (RootDoc root) Generate documentation here.
static boolean	validOptions (java.lang.String[][] options, DocErrorReporter reporter) Check that options have the correct arguments here.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Was sollte man
kommentieren?



JavaDoc?