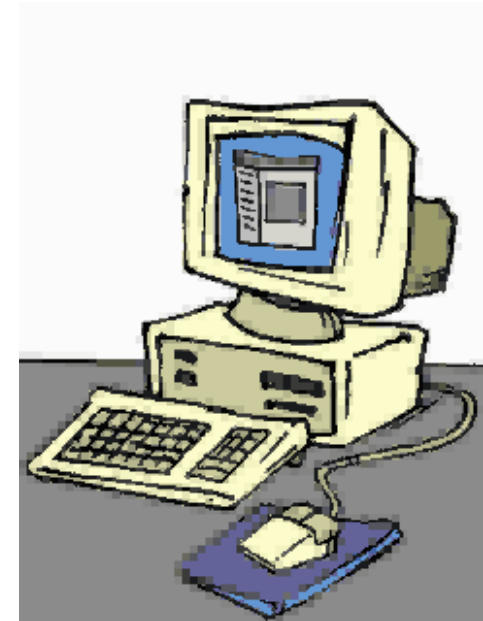


7. Implementierung

Dr.-Ing. Clemens Reichmann

Clemens.Reichmann@vector.com,
Tel. 0721-91430-200

Institut für Technik der Informationsverarbeitung
Fakultät für Elektrotechnik & Informationstechnik
Universität Karlsruhe (TH)



7.1 Entwicklungswerkzeuge

7.1.1 Compiler und Debugger

7.1.2 CVS

7.1.3 IDE (Beispiel eclipse)

7.2 Quellcode-Dokumentation

7.2.1 Programmierrichtlinien (Codestyle)

7.2.2 JAVADOC

7.3 Programmierkonzepte

7.3.1 Selbstkontrolliertes Programmieren

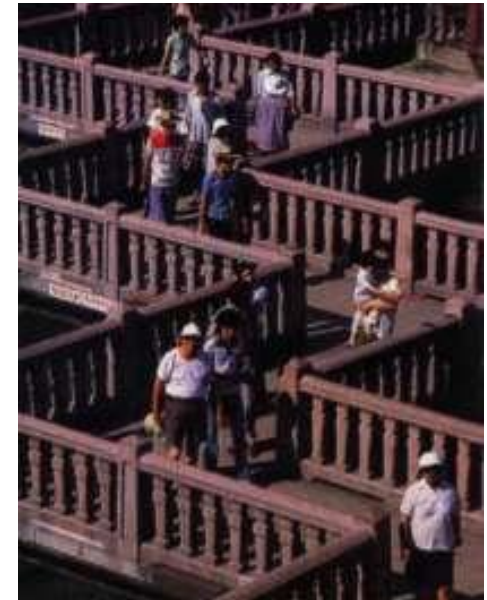
7.3.2 Ausnahmen (exceptions)

7.3.3 Thread



- Programmier-Werkzeuge:
 - **Editor** Text-Editor zum Erstellen des Programm-Quelltextes (Quellcode) (Notepad, vi, Emacs, NoteTab, joe, ...)
 - **Compiler** übersetzt den Quelltext und liefert als Ergebnis eine Datei mit Binärcode (oder Fehlermeldungen).
 - **Interpreter** führt vorliegenden Quelltext (oder "Zwischencode" wie z.B. Java-Byte-Code) auf einem Rechnersystem aus.
 - **Debugger** Programm zur Fehlersuche
 - **Linker** Verbinden von Binärprogrammen (Bibliotheken) zu einem ausführbaren Programm.
 - **IDE** (Integrated Development Environment) Programmsysteme zur professionellen/komfortablen Programmentwicklung

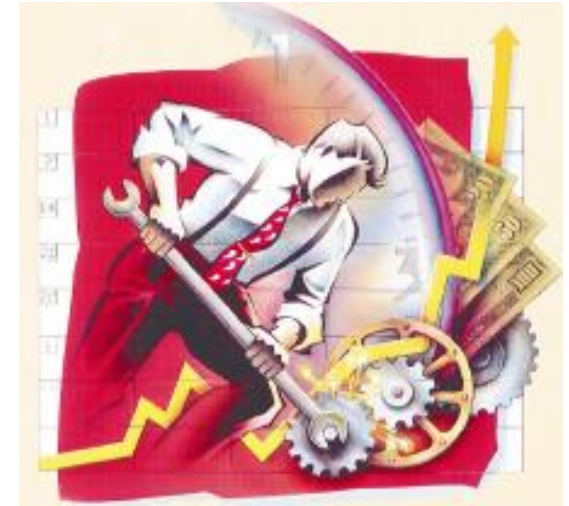
- Compiler, ggf. Interpreter, Debugger, Linker sind Programme, die
 - zum Programmiersystem einer Programmiersprache gehören
 - je nach Programmiersprache und Entwicklungssystem (IDE)
 - kommerziell erhältlich,
 - frei verfügbar sind oder
 - zum Betriebssystem gehören.



- **JDK/SDK (Java Developers Kit/Software Developers Kits)**
- **Standard-Werkzeuge für die Java-Programmierung (frei bei Sun Microsystems erhältlich „java.sun.com“).**

javac	Java-Compiler. Erzeugt aus dem Java-Quelltext Java-Byte-Code.
java	Java-Byte-Code- Interpreter zum Ausführen von Applikationen (startet die Java Virtuelle Maschine).
appletviewer	Ausführen und Anzeigen von Applets
javadoc	Generieren von Dokumentation aus Java-Quelltext
jdb	Debugger

- Übersetzer
- Prüft Syntax
- Erzeugt Maschinencode/Bytecode



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

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.

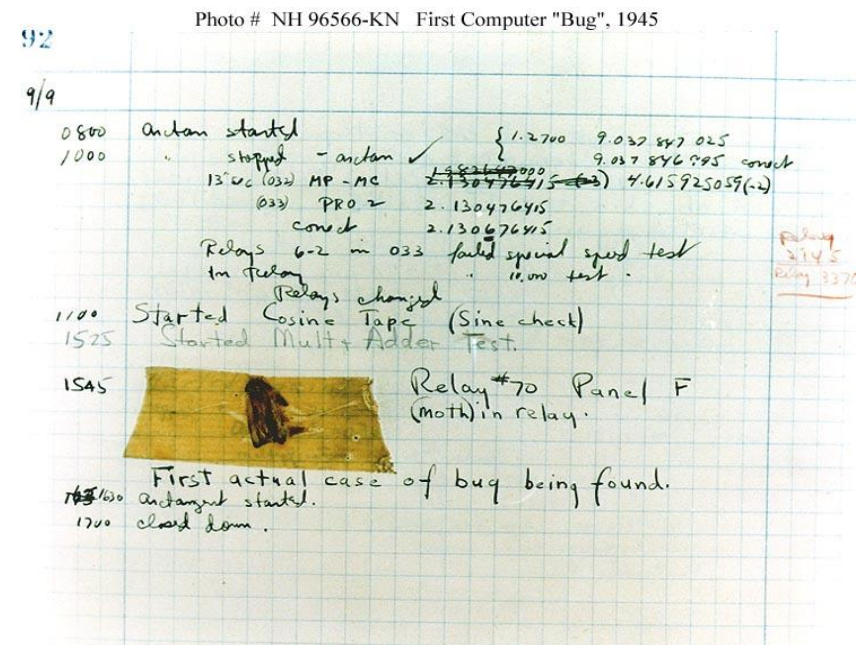
**Debugging /
Simulation**

Fehler- analyse

- Finden der Fehlerursache (**defect**)

- Beheben des Fehlers

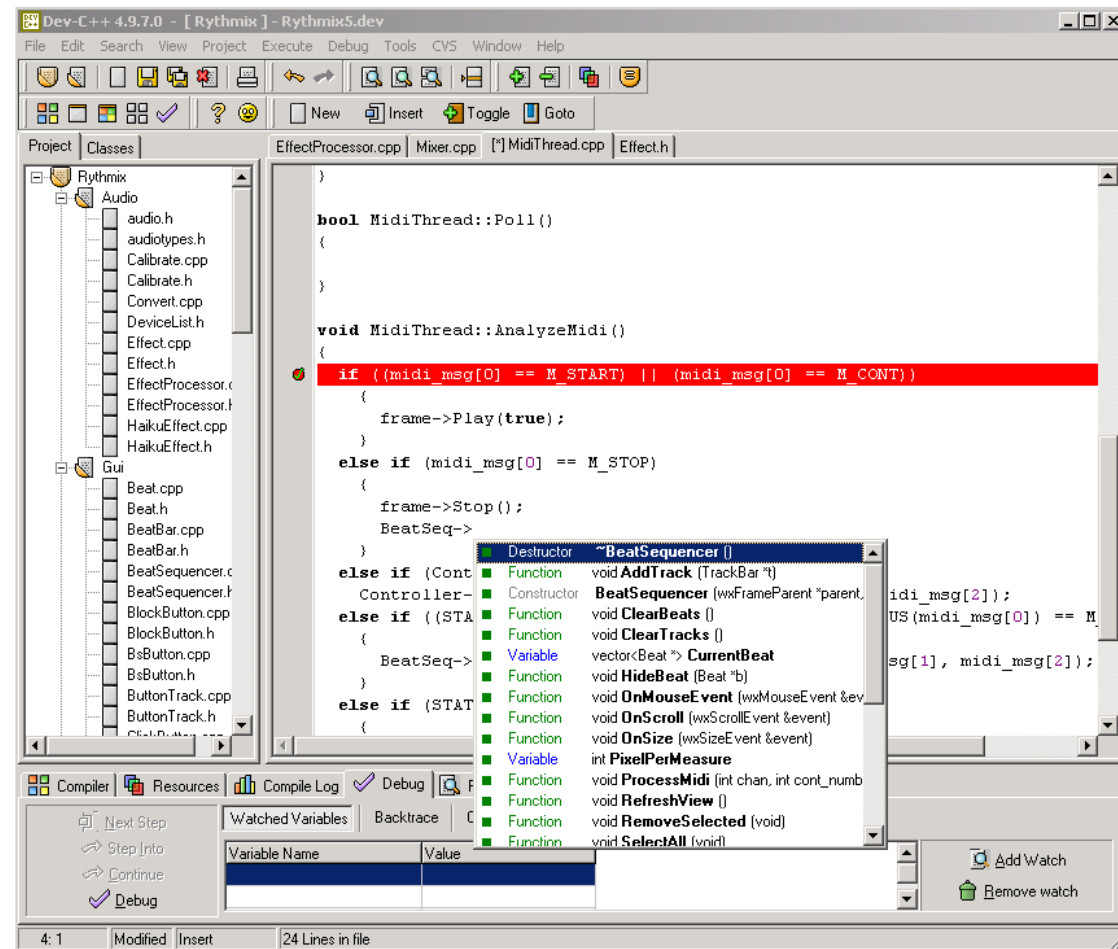
- Edison in einem Brief von 1878:
 - It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise -- this thing gives out and [it is] then that "**Bugs**" -- as such little faults and difficulties are called -- show themselves and months 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

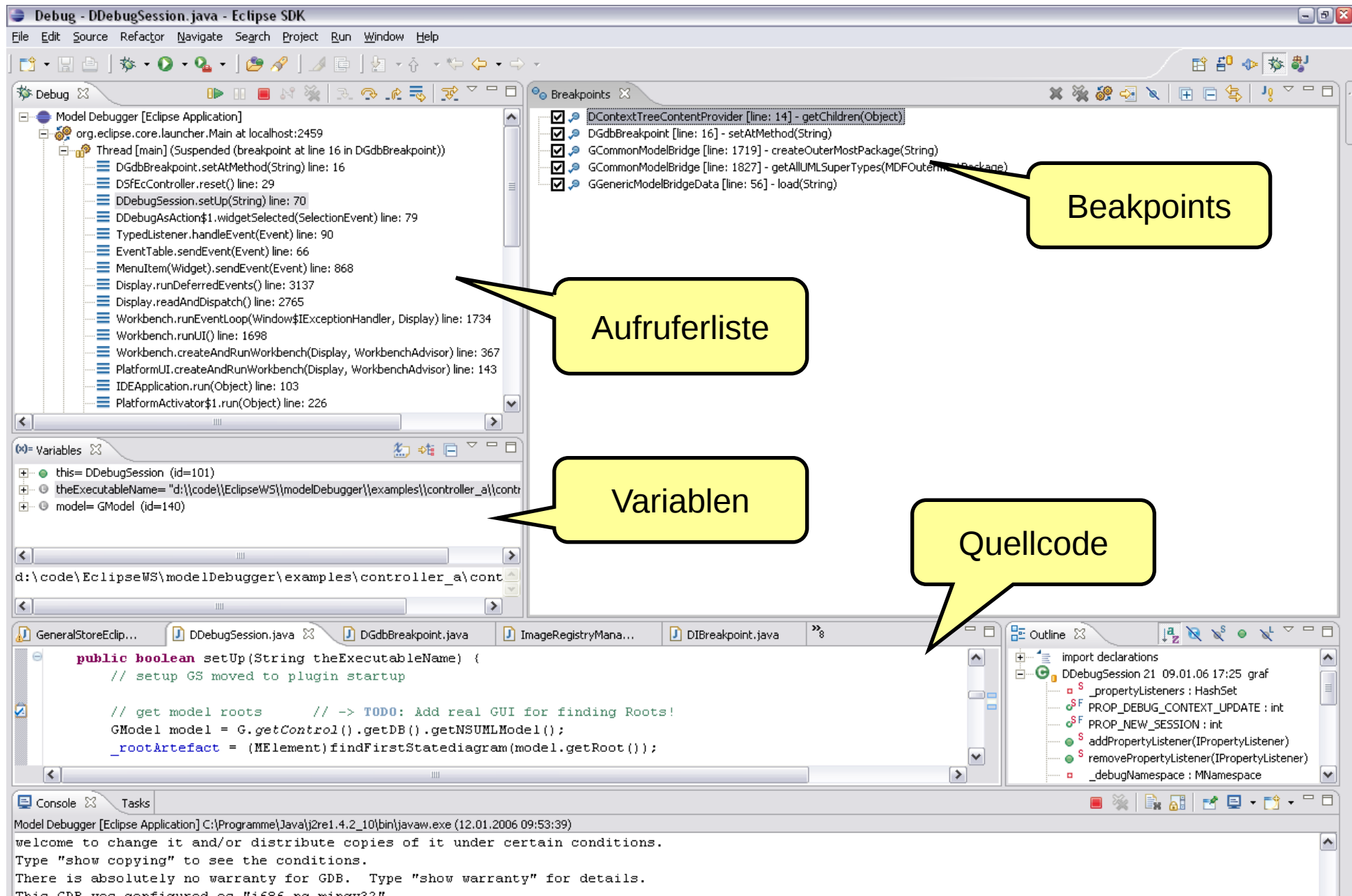


- Geeignet für imperative Sprachen
- Erstmals 1961: „DEC Debugging Tool “ (DDT) von Digital Equipment Corporation (DEC) für PDP-1
- **Ablaufkontrolle (*run-control*)**
 - Setzen von Haltepunkten (*breakpoints*)
 - Einzelschritt
 - Starten/Stoppen
- **Inspektion**
 - Quelltextansicht
 - Programmzustand (Register, Variablen)
 - Aufrufliste (*call stack*)
 - Speicher
 - Threads / Prozesse
- **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, weiter mit 1.
 5. Ansonsten: Korrigieren des Fehlers



- Möglichkeiten im Debug:
 - Anhalten an Breakpoints
 - Anschauen des aktuellen Wertes von Variablen
- Durchführung des Programms:
 - „Step by Step“
 - „Step into“
 - „Step over“





Debug - DDebugSession.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Debug [Eclipse Application]

- org.eclipse.core.launcher.Main at localhost:2459
 - Thread [main] (Suspended (breakpoint at line 16 in DGdbBreakpoint))
 - DGdbBreakpoint.setAtMethod(String) line: 16
 - DSfEcController.reset() line: 29
 - DDebugSession.setUp(String) line: 70
 - DDebugAsAction\$1.widgetSelected(SelectionEvent) line: 79
 - TypedListener.handleEvent(Event) line: 90
 - EventTable.sendEvent(Event) line: 66
 - MenuItem.sendEvent(Event) line: 868
 - Display.runDeferredEvents() line: 3137
 - Display.readAndDispatch() line: 2765
 - Workbench.runEventLoop(Window\$ExceptionHandler, Display) line: 1734
 - Workbench.runUI() line: 1698
 - Workbench.createAndRunWorkbench(Display, WorkbenchAdvisor) line: 367
 - PlatformUI.createAndRunWorkbench(Display, WorkbenchAdvisor) line: 143
 - IDEApplication.run(Object) line: 103
 - PlatformActivator\$1.run(Object) line: 226

Breakpoints

- ☒ DContextTreeContentProvider [line: 14] - getChildren(Object)
- ☒ DGdbBreakpoint [line: 16] - setAtMethod(String)
- ☒ GCommonModelBridge [line: 1719] - createOuterMostPackage(String)
- ☒ GCommonModelBridge [line: 1827] - getAllUMLSuperTypes(MDFOuterMostPackage)
- ☒ GGenericModelBridgeData [line: 56] - load(String)

Aufruferliste

Variables

- this= DDebugSession (id=101)
- theExecutableName= "d:\\code\\EclipseWS\\modelDebugger\\examples\\controller_a\\cont
- model= GModel (id=140)

d:\\code\\EclipseWS\\modelDebugger\\examples\\controller_a\\cont

GeneralStoreEclip... DDebugSession.java DGdbBreakpoint.java ImageRegistryMana... DIBreakpoint.java

```
public boolean setUp(String theExecutableName) {  
    // setup GS moved to plugin startup  
  
    // get model roots    // -> TODO: Add real GUI for finding Roots!  
    GModel model = G.getControl().getDB().getNSUMLModel();  
    _rootArtefact = (MElement) findFirstStatedDiagram(model.getRoot());  
}
```

Outline

- import declarations
- DDebugSession 21 09.01.06 17:25 graf
 - S _propertyListeners : HashSet
 - S PROP_DEBUG_CONTEXT_UPDATE : int
 - S PROP_NEW_SESSION : int
 - S addPropertyListener(IPropertyListener)
 - S removePropertyListener(IPropertyListener)
 - S _debugNamespace : MNamespace

Console

Model Debugger [Eclipse Application] C:\\Programme\\Java\\j2re1.4.2_10\\bin\\javaw.exe (12.01.2006 09:53:39)

welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i686-pc-mingw32".

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 warranty" for details.
This GDB was configured as "i386-linux"...Using host libthread_db library "/lib/
libthread_db.so.1".

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

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

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

Datei Bearbeiten Ansicht Projekt Erstellen Debuggen Extras Fenster Hilfe

Program [2800] beispiel.exe: 5; Thread [2912] main

Debug <UML:GraphEdge.waypoints>

beispiel.cpp Disassembly

```
Global
void sort(int *a, int n) {
    int i,j,tmp;
    for (i=0; i<n-1; i++) {
        for (j=0; j<n-1-i; j++) {
            if (a[j+1] < a[j]) { /* compare the two neighbors
                                /* swap a[j] and a[j+1]
                tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
            }
        }
    }
}
```

Arbeitspeicher 1

Adresse	0x00411A10
0x00411A10	55 8b ec 81 U[]□
0x00411A14	ec e4 00 00 iä..
0x00411A18	00 53 56 57 .SVW
0x00411A1C	8d bd 1c ff □%.ÿ
0x00411A20	ff ff b9 39 ÿÿ'9
0x00411A24	00 00 00 b8 ...
0x00411A28	cc cc cc cc iiii
0x00411A2C	f3 ab c7 45 ó«ÇE
0x00411A30	f8 00 00 00 s...
0x00411A34	00 eb 09 8b .ë.□
0x00411A38	45 f8 83 c0 Eø□Ä

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++...	C++
beispiel.exe!mainCRTStartup() Zeile 259 + C...	C
KERNEL32.DLL!77e81af6()	

Auto Lokal Überwachen 1

Bereit

GeneralStore.java - [D:\code\MDebug\GeneralStore] - [GeneralStore] - D:\code\MDebug\GeneralSt...

File Edit Search View Go To Code Analyze Refactor Build Run Tools CVS Window Help

GeneralStore

GeneralStore

```
if (theFilterName.equals("")) {
    theFilterName = GXMISynchronizerGUI.NO_FILTER;
}
GFileReader originalFileReader = null;
String xmiVersion = null;
String oldProperty = null;
File tempFile = null;
try {
    tempFile = new File(new File("vs temp in xml").getCanonicalPath());
}
```

Debug - GeneralStore

Console Threads Frame Watches

readModelProcessed():55... Thread "GSwingWorker" @3829 in group "...

this = {com.itiv.generalStore.tools.xmisynchronizer.GImporter@3853}

theFile: java.io.File = {sun.awt.shell.Win32ShellFolder2@4175}

monModelBridge = {com.itiv.generalStore.model.GGe...

GGenericModelBridgeData = {com.itiv.generalStore.m...

postPackage = {ru.novosoft.uml.impl.UMLUMLPackage...

Writer = {ru.novosoft.uml.impl.UMLRepositoryImpl@...

Reader = {ru.novosoft.uml.impl.UMLRepositoryImpl@...

Insert Import Popup: ON 98M of 126M

- 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“

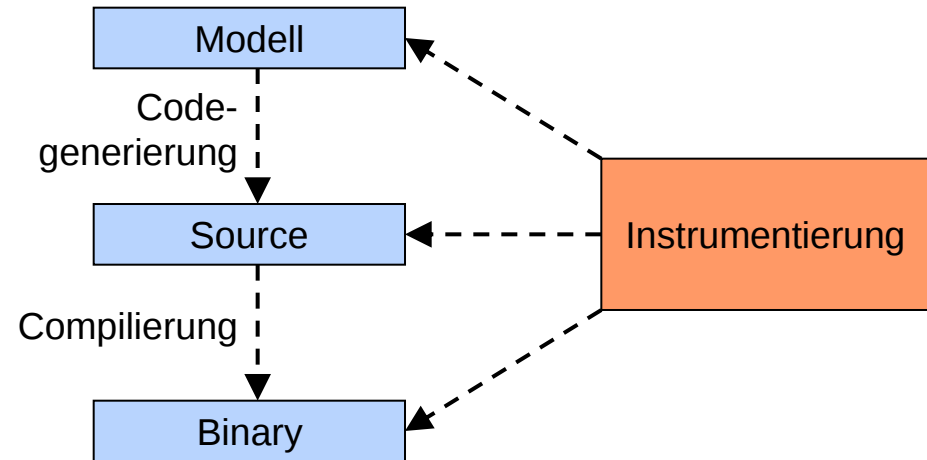
- Dedizierte Hardware



iSystem iC4000 ActiveEmulator

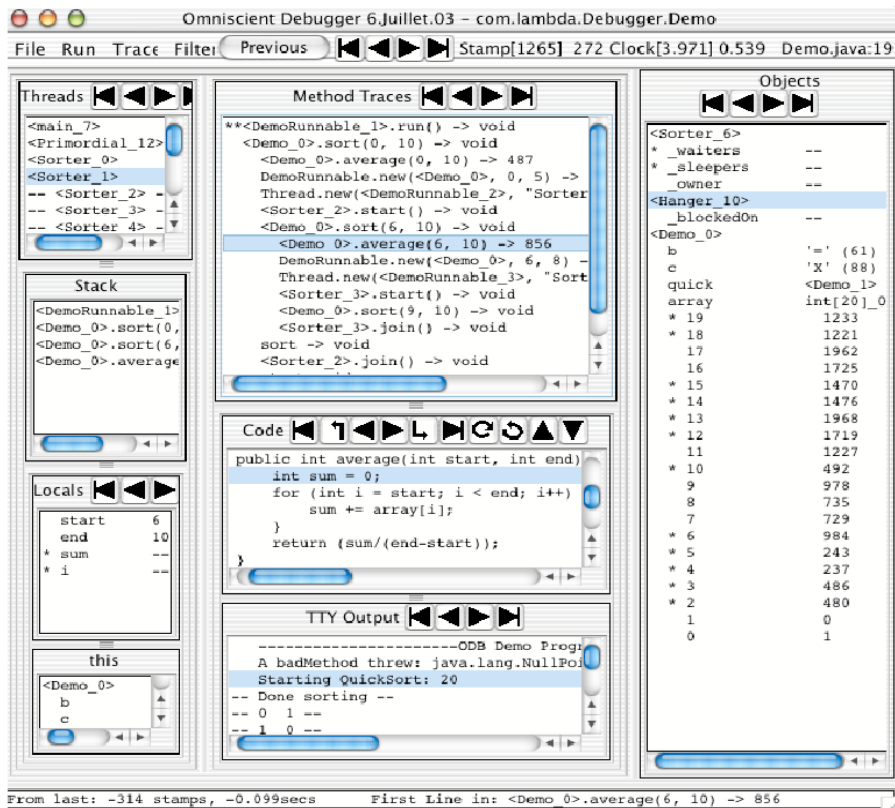
- Emuliert CPU
 - Überwacht Pins
 - Nutzt On-Chip Schnittstellen (NEXUS 5001, BDM, JTAG)
-
- + Echtzeitfähig
 - + Keine Beeinflussung des Programms
 - Oft sehr teuer
 - Bandbreite endlich

- Instrumentierung

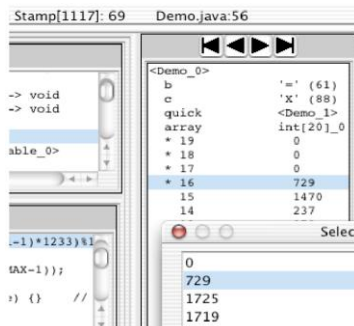


- Modell, Source oder Binary
- Statisch oder dynamisch

- + Beliebige Aquisition von Kontrollfluss und Daten
- Overhead Speicher und Performanz
- Veränderung des Programms („Heisenberg“-Effekt)

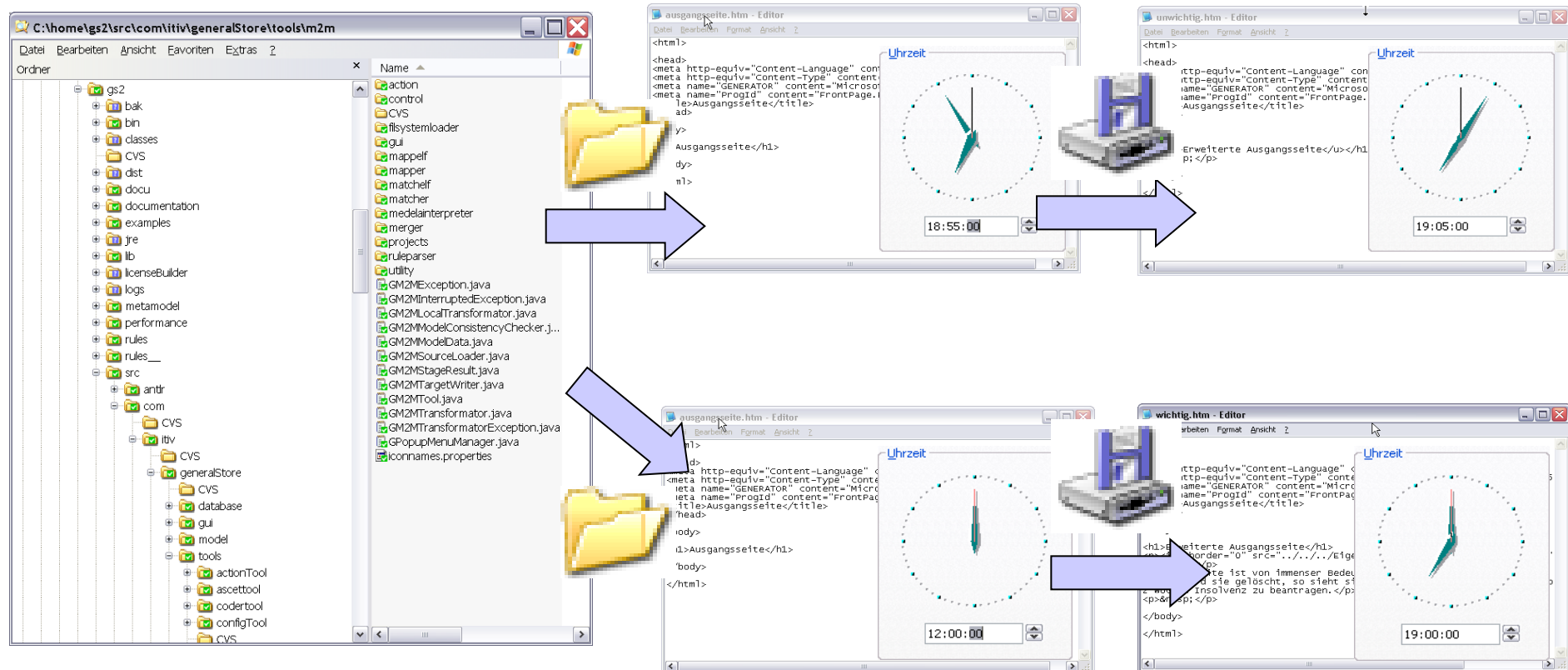


- Konzept: **Rückwärts in der Zeit**
- Eigener *class-loader* instrumentiert JAVA-Bytecode
- Protokolliert
 - Kontrollfluss
 - Variablenänderungen
 - Umschalten zwischen Threads
- Präsentation verknüpft
 - Abfolge der Methodenaufrufe
 - Quelltextansicht
 - Überwachungsfenster
 - Ausgabe
- Abfolge der Werte einer Variable und Springen zum Zeitpunkt der Änderung

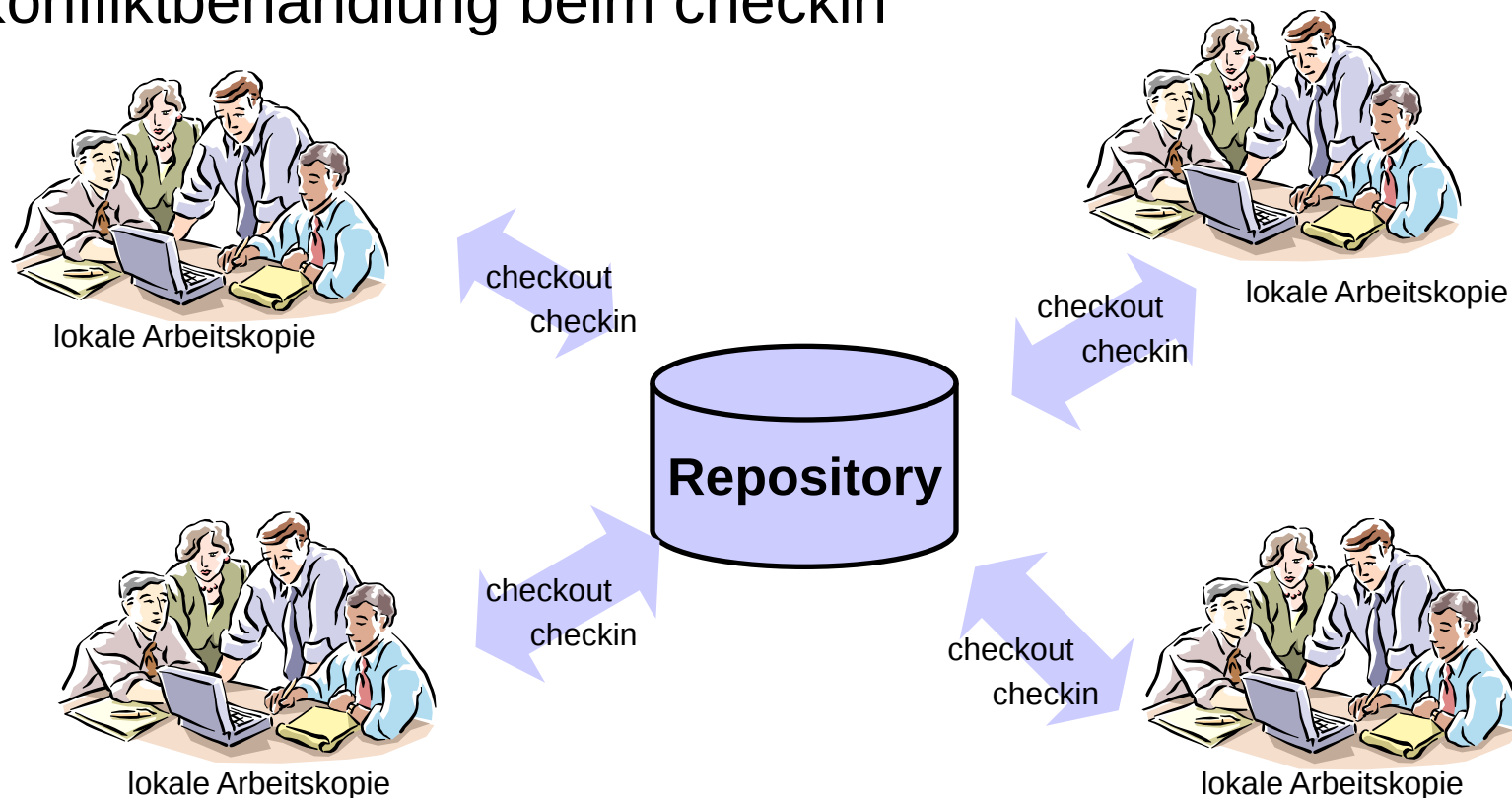


- Concurrent Versions System (CVS)
- Aegis
- Arch
- BitKeeper
- CMSynergy
- Co-Op
- Darcs
- GIT
- Monotone
- OpenCM
- Perforce
- PureCM
- Subversion
- Supersversion
- svk
- Vesta
- Visual SourceSafe
- usw.

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



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



- Der Befehl `cvcs update` bewirkt ein Versionsabgleich zwischen Repository und Workspace. Datei für Datei und Zeile für Zeile werden Änderungen ermittelt und „gemerged“:

	Repository	Workspace
Fall A		
Fall B		
Fall C		
Fall D		

	Repository	Workspace
Fall A		
Fall B		
Fall C		
Fall D		

Fall A:

- Nix passiert

Fall B:

- Update meldet, dass eine Änderung im Workspace vorgenommen wurde

Fall C:

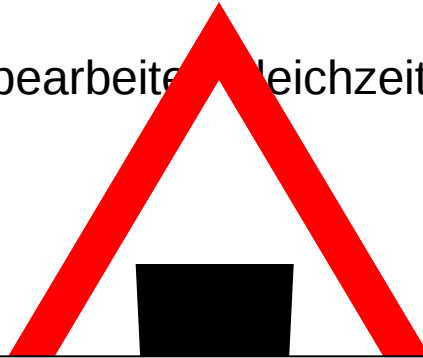
- Kopie im Workspace wird durch Version im Repository ersetzt

Fall D:

- Drei Dateiversionen werden zusammengeführt
 - Ursprüngliches Repository Checkout
 - Neue Version im lokalen Workspace
 - (geänderte) Version im Repository
- Sie werden durch „Three Way Merge“ zusammengeführt

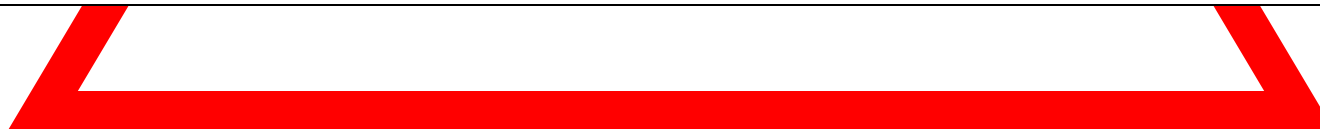
- Checkout
 - Erstellt eine private Kopie der Dokumente innerhalb des lokalen Arbeitsverzeichnisses.
 - Die Position ist beliebig
 - Mehrere lokale Kopien mit unterschiedlichen Versionen sind möglich
- Commit
 - Übermittelt am Ende der Arbeit die Änderungen an den Dokumenten an den Server
 - Die lokalen Kopien müssen die aktuellen Versionen sein
- Update
 - Vergleicht die lokalen Kopien mit dem Repository und aktualisiert sie bei Bedarf

- Geht meistens gut, außer:
 - Teammitglieder A und B bearbeiten gleichzeitig das gleiche Programm.

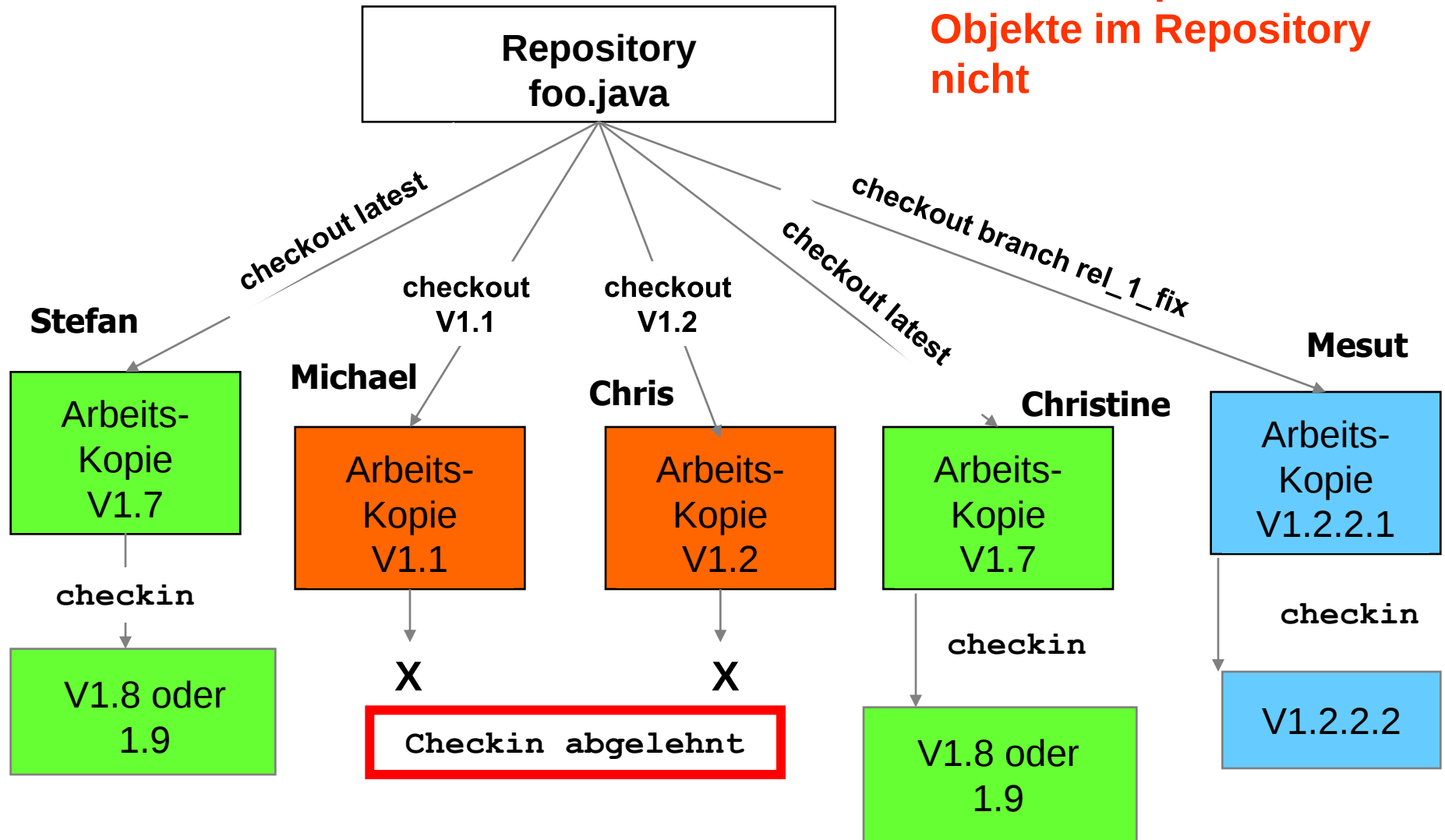


Problem: Wenn zwischen Update und Commit eine größere Zeitspanne liegt, besteht die Gefahr, dass ein anderer während dessen ebenfalls updatet und einen Commit durchführt, wodurch das Update des ersten Teilnehmers veraltet und das System in ein „Versionsdilemma“ gerät

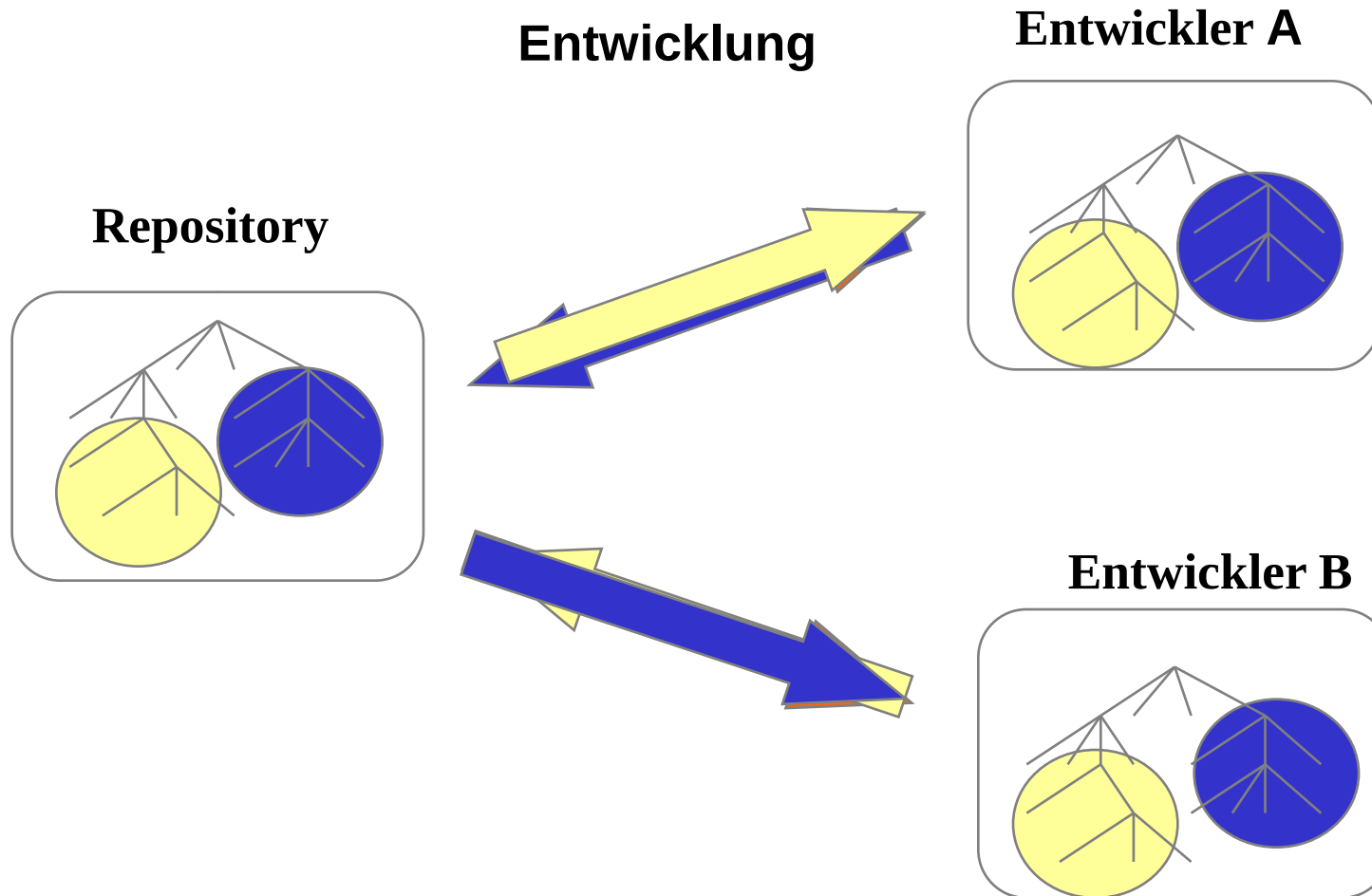
Deshalb: Vor Commit *IMMER* Update durchführen

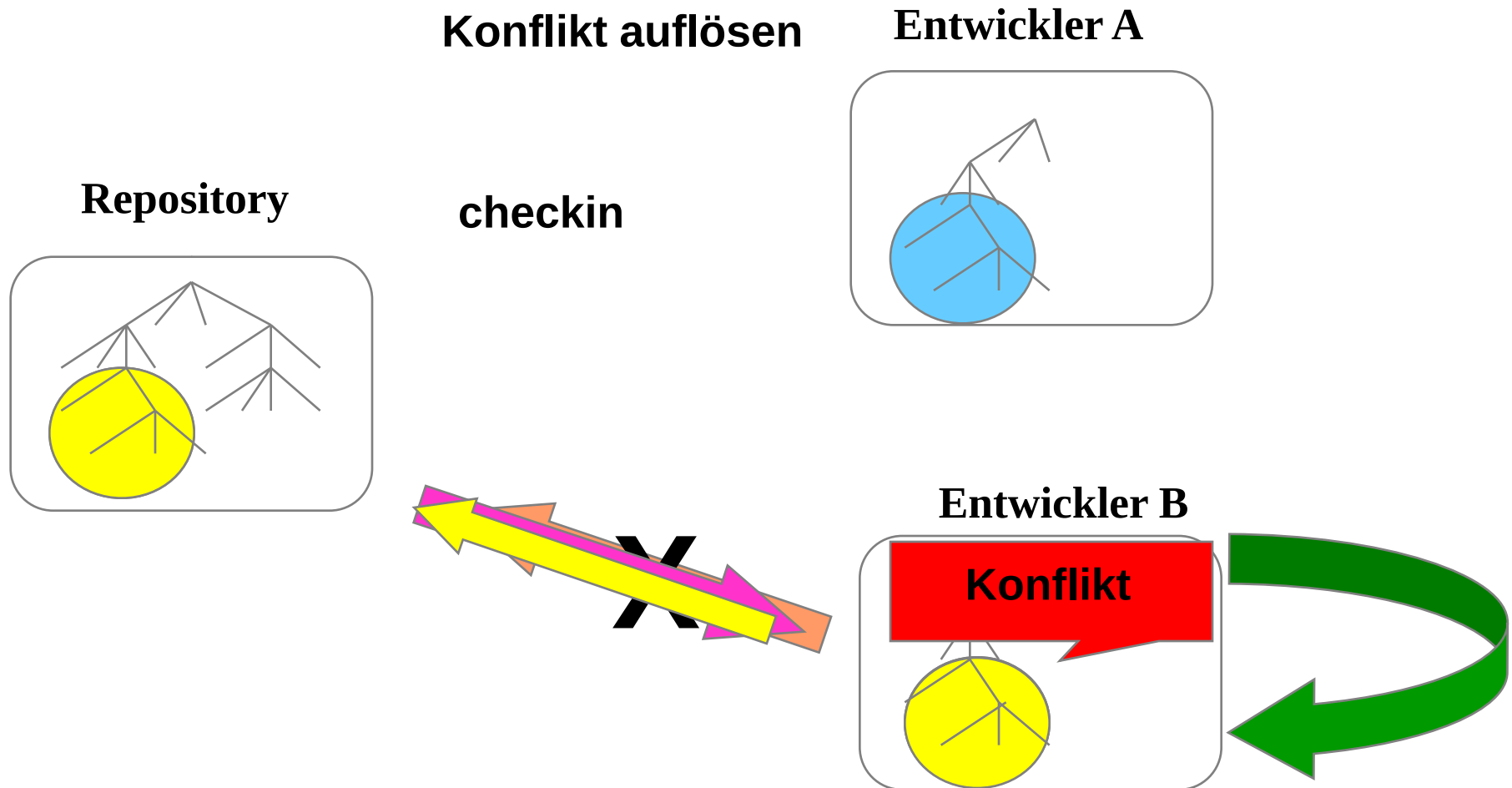


**Checkout sperrt die
Objekte im Repository
nicht**



1. Dateien im lokalen Arbeitsverzeichnis “auschecken”
2. Dateien editieren
3. Testen (lokaler Build, Syntaktische Analyse, ...)
4. Aktualisierung der lokalen Kopien mit **update**
dabei werden die Änderungen der anderen Entwickler
eingefügt (wenn nötig)
5. Nochmal testen, wenn in Schritt 4 gepatcht wurde
6. Änderungen übermitteln mit **commit**
7. Ab Schritt 2 wiederholen, bis zur nächsten Release
8. Release markieren (**tag**)
9. Modulnamen und Release-Tag zur System-Synthese
übermitteln

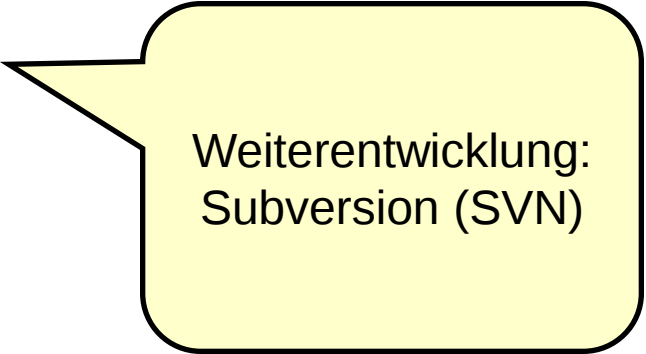




- Zum Markieren eines Status zur späteren Wiederherstellung
- Einchecken zusammenhängender Dateien mit einer einzelnen Operation
→ Dabei eine Log-Nachricht angeben
- Einchecken ist ein Backup
- In der Uni einchecken, um dann zu Hause schnell auschecken zu können
- Vorher immer ein Update

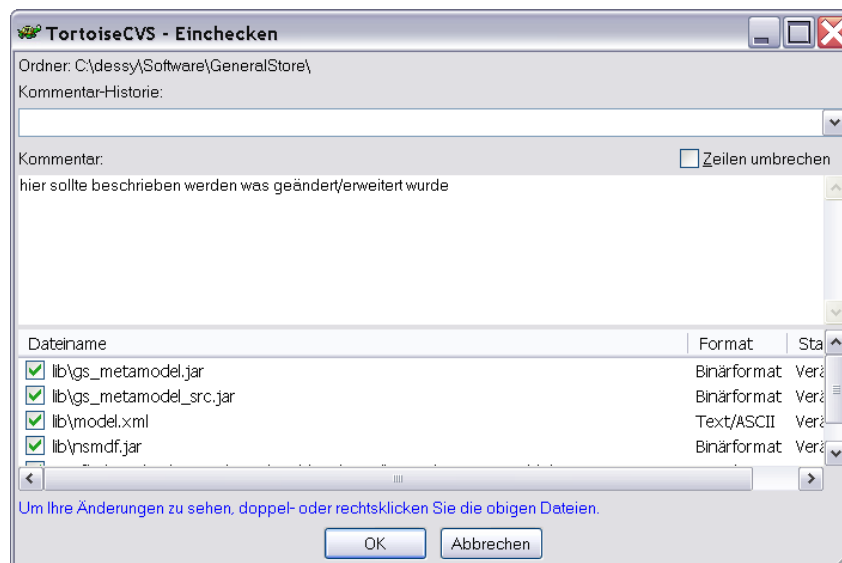
- „Checke nur **kompilierbaren und getesteten Code** ein — die anderen werden es dir danken
- Mache **vor einem commit noch ein update** und achte auf zu lösende Konflikte
- Führe **regelmäßig commit** aus und gib **sinnvolle Log-Einträge** an
(...)
- Denke immer daran, daß andere von deiner Arbeit abhängig sind“

- Schlechter Umgang mit Verzeichnissen
- Schlechter Umgang mit Binärdateien
- Umbenennen von Dateien ist schwierig
- Keine Sicherheitsmechanismen
- Gewisse Features erfordern Disziplin



Weiterentwicklung:
Subversion (SVN)

- Gibt es in allen Größen und Formen
- Potenzielle Vorteile:
 - Übersichtliche Darstellung
 - Grafische Revisionsbäume
 - Grafische Aufbereitung von Änderungen
 - **Integration in IDE**
- Nachteil: Funktionsweise und Features stark unterschiedlich





- integrierte Entwicklungsumgebung zur Entwicklung von Software
- Komponenten:
 - Texteditor
 - Compiler bzw. Interpreter
 - Linker
 - Debugger
 - Quelltextformatierungsfunktion
 - Versionsverwaltung
 - Projektmanagement
 - UML-Modellierung
 - Erstellung von grafischen Benutzeroberflächen.
- Beispiele
 - Borland C++ Builder, IntelliJ IDEA (JAVA, .NET), Microsoft Visual Studio, Eclipse, KDevelop

- Autovervollständigung

```
Point p = new Point();
```

```
p.
```

- x int - Point
- y int - Point
- clone() Object - Point2D
- distance(double arg0, double arg1) double - Point2D
- distance(Point2D arg0) double - Point2D
- distanceSq(double arg0, double arg1) double - Point2D
- distanceSq(Point2D arg0) double - Point2D
- equals(Object arg0) boolean - Point
- getClass() Class - Object
- getLocation() Point - Point

```
Point p = new Point();
```

```
p.setLocation();
```

- setLocation(double x, double y) void - Point
- setLocation(int x, int y) void - Point
- setLocation(Point p) void - Point
- setLocation(Point2D p) void - Point2D

Changes the point to have the specified location.

This method is included for completeness, to parallel the setLocation method of Component. Its behavior is identical with move(int, int).

Parameters:

x the x coordinate of the new location.

y the y coordinate of the new location.

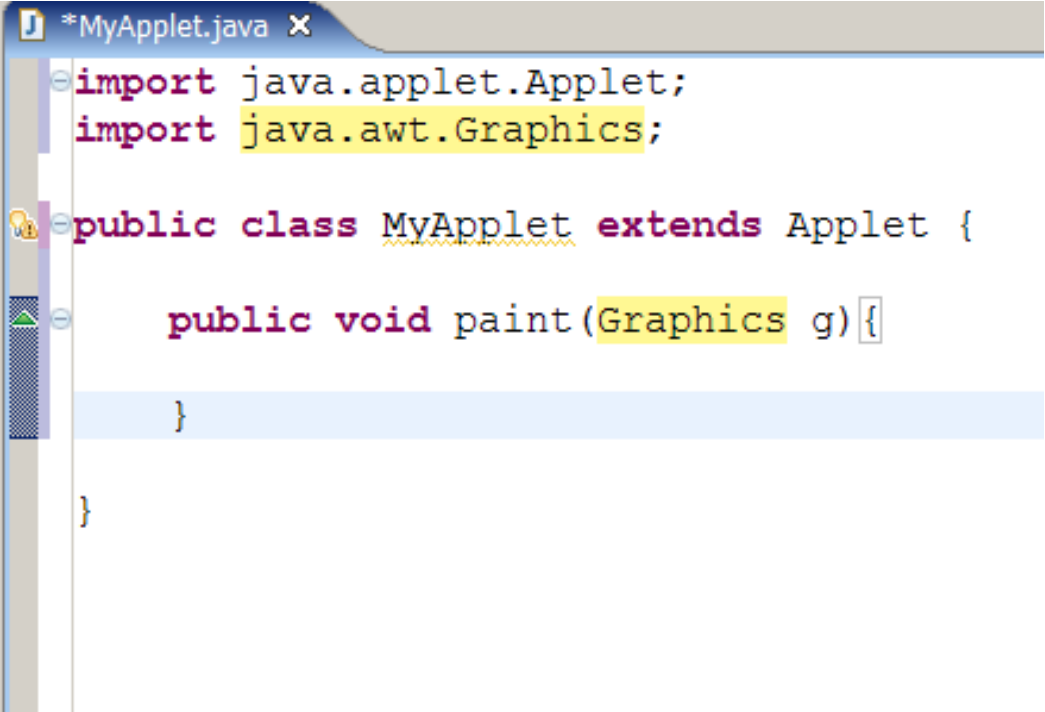
See Also:

java.awt.Component.setLocation(int, int)

java.awt.Point.getLocation

java.awt.Point.move(int, int)

- Syntax Highlighting
- Autocompilierung
- Refactorings
- Suche
- Navigation durch den Quellcode
- Codegeneratoren
- Hervorhebungen



```
*MyApplet.java x
import java.applet.Applet;
import java.awt.Graphics;

public class MyApplet extends Applet {

    public void paint(Graphics g){

    }

}
```

- Motivation
- Geschichte
- Architektur
 - Platform Runtime
 - Eclipse Platform
 - Java Development Tools (JDE)
 - Plugin Development Environment (PDE)
- Zusammenfassung



- Was ist eclipse?
 - open source Entwicklungsumgebung
 - Deckt viele OS ab (Windows, Linux, Solaris,...)
 - Sprachneutral (Java, C, Cobol, ...)
 - erweiterbare Plattform für die Werkzeugintegration
 - gesamter Softwarelebenszyklus abdeckbar
 - bringt Entwicklungsumgebung für Erweiterungen gleich mit
 - beinhaltet neue GUI für Java-Applikationen
 - Framework für Anwendungsentwicklung



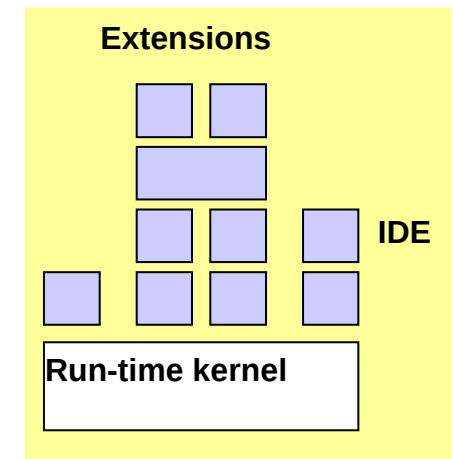
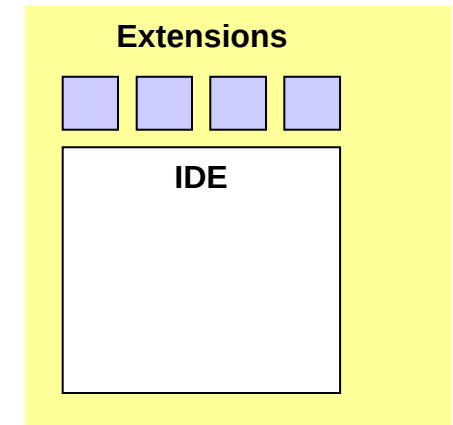
- Als kommerzielle Version von OTI + IBM geplant
- Entwicklung ab April 1999
- 2001 Version 1.06 an open source -Gemeinde übergeben
- aktuell: 3.1.1
- Kommunikationsplattform: <http://www.eclipse.org>
- große + sehr aktive Community
- Eclipse foundation: IBM, Borland, BEA, Intel, HP, SAP, ...



- Welche IDEs gibt es noch?
 - JBuilder, NetBeans, IntelliJ IDEA, WSAD, ...
- Wann bzw. warum setzt man IDEs anstatt einfacher Editoren ein?
 - große Projekte
 - heute Software benötigt, die gesamten Software-lebenszyklus möglichst nahtlos unterstützt
 - Installation, Einarbeitung, Datenaustausch, Teamwork, ... erleichtert
 - Anpassbarkeit
 - Refactoring, Debugging, Code-Schablonen, Syntaxcheck, Code Completion, Hovering

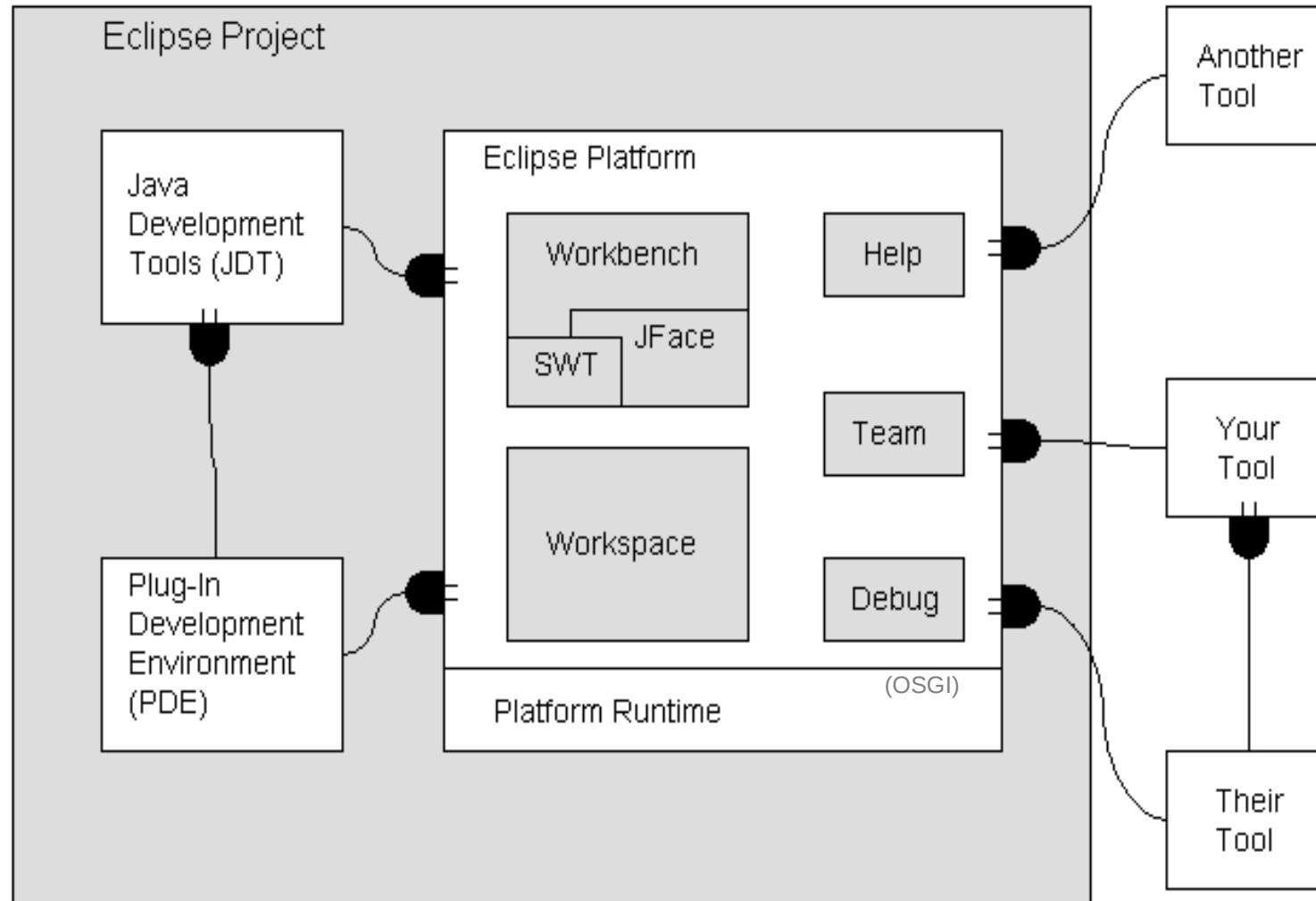


- Übliche IDE-Architektur
 - monolithisch => Erweiterungen nur wie vorgesehen
 - Erweiterungen wirken oft fremd
- Eclipse:
 - Bestandteile: Plugins + Platform Runtime
 - Plugins nutzen Plugins
 - Endanwender richten eigene Umgebung ein (Installieren + Deinstallieren Plugins)
 - Erweiterung Teil der Philosophie

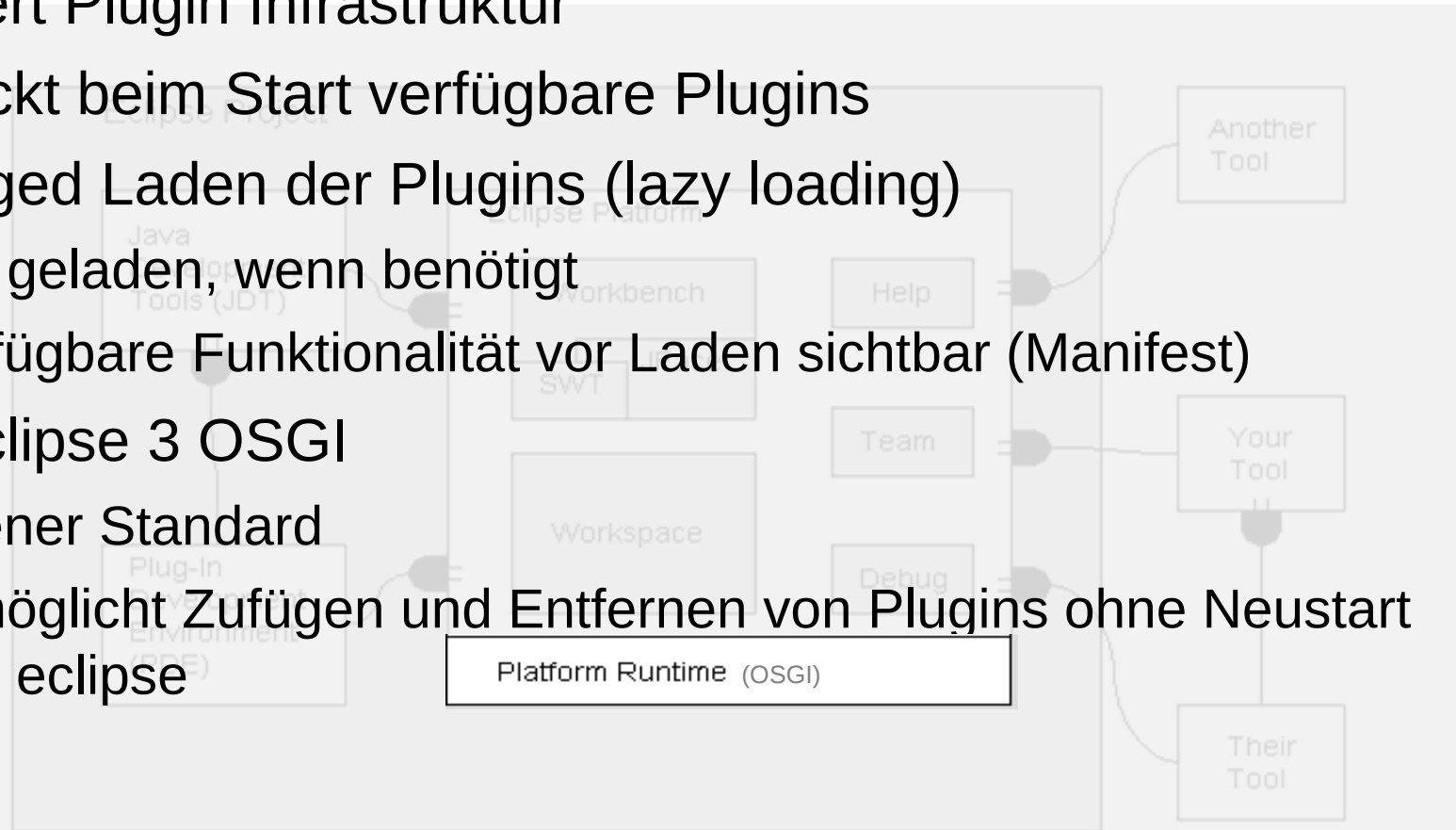


- Plugins:
 - zwingend: Plugin-Manifest (plugin.xml)
 - Deklaration
 - beschreibt Konfiguration des Plugins
 - beschreibt Integration des Plugins in die Plattform, d.h. welche Erweiterungspunkte genutzt, welche bereitgestellt
 - optional: Java-Archiv
 - Implementierung
 - optional: Ressourcen
 - Bilder
 - Hilfetexte
 - ...

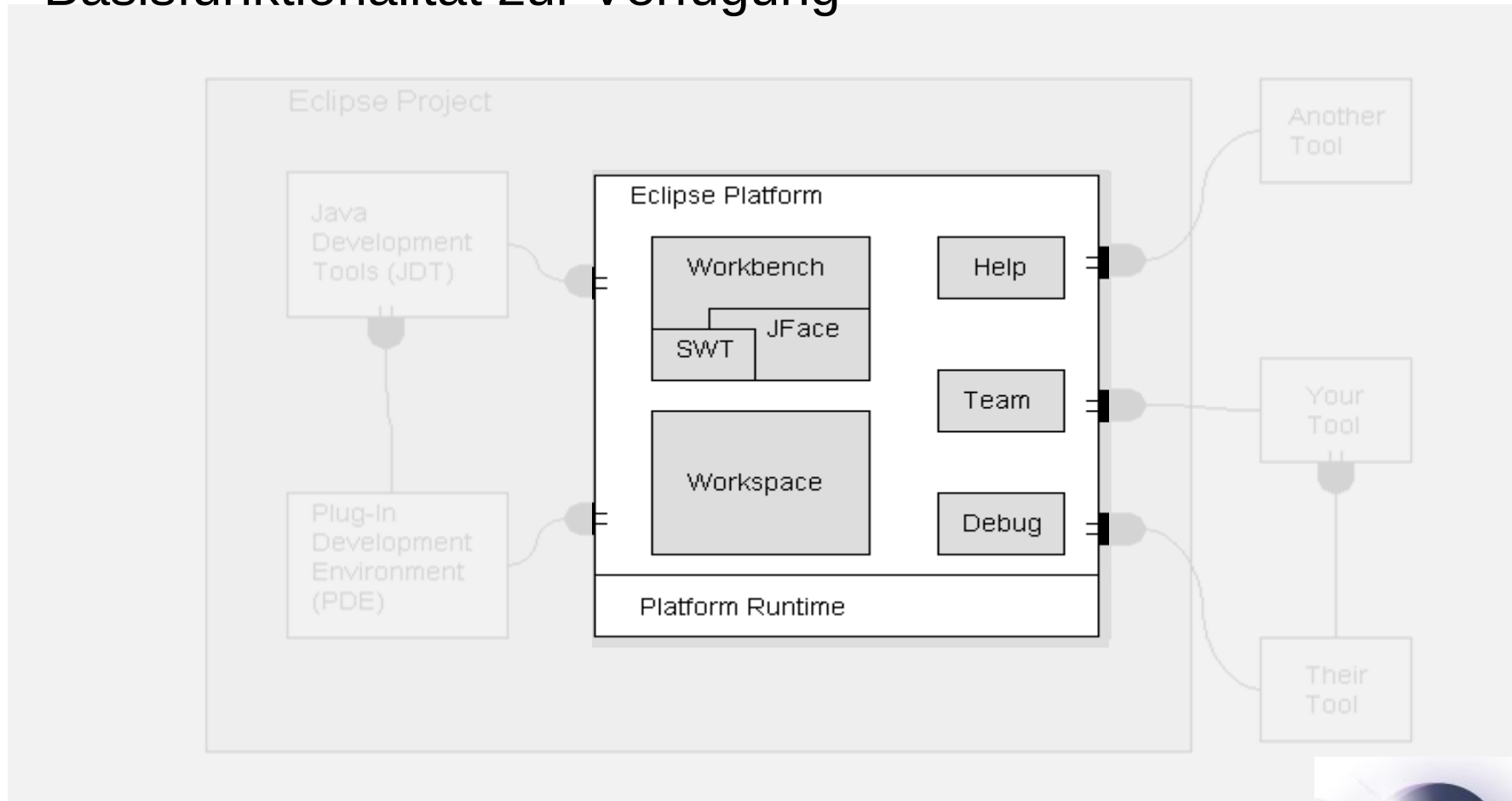




- Kern, kein Plugin (aber formal als Plugin angelegt)
- definiert Plugin Infrastruktur
- entdeckt beim Start verfügbare Plugins
- managed Laden der Plugins (lazy loading)
 - nur geladen, wenn benötigt
 - verfügbare Funktionalität vor Laden sichtbar (Manifest)
- seit eclipse 3 OSGI
 - offener Standard
 - ermöglicht Zufügen und Entfernen von Plugins ohne Neustart von eclipse

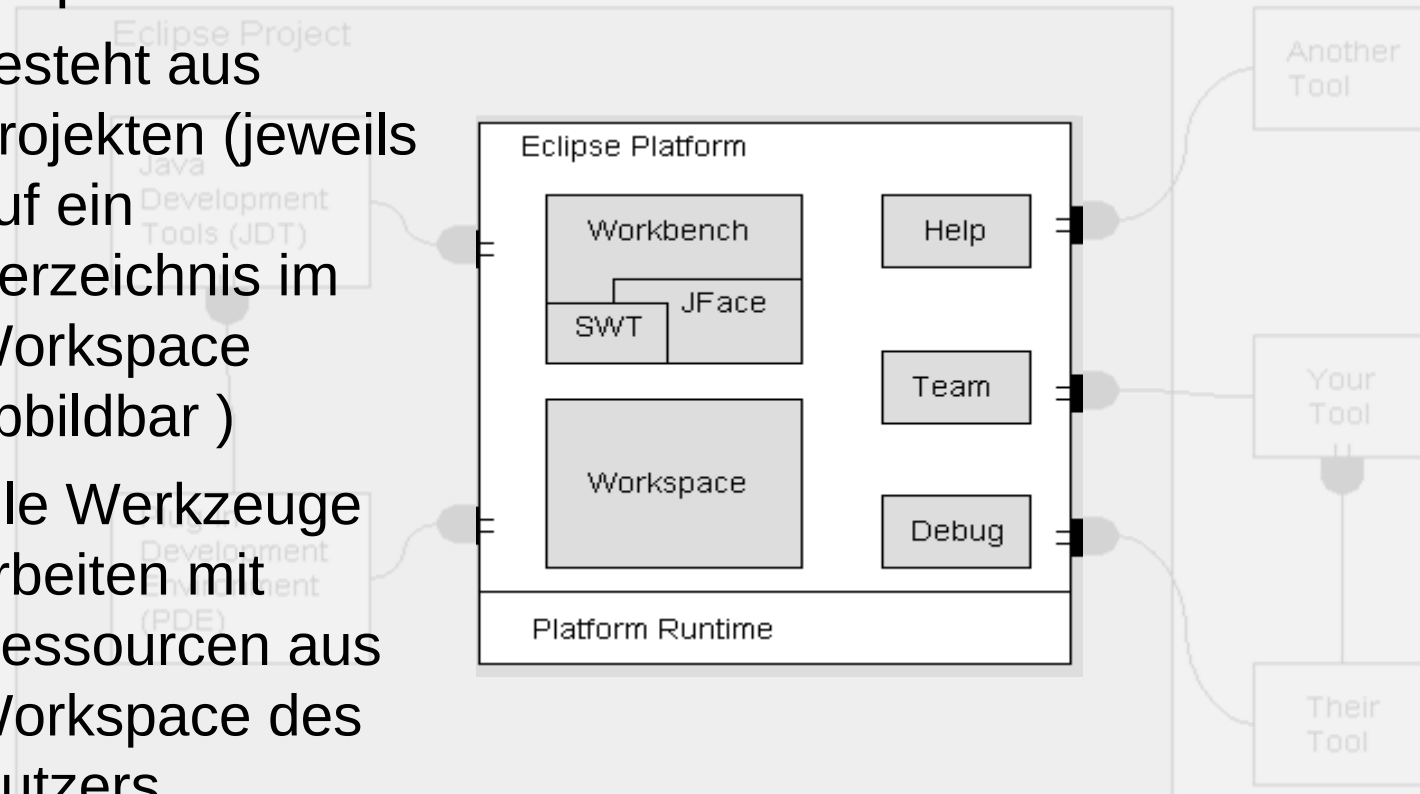


- Kernkomponenten, stellen domänen-spezifische Basisfunktionalität zur Verfügung



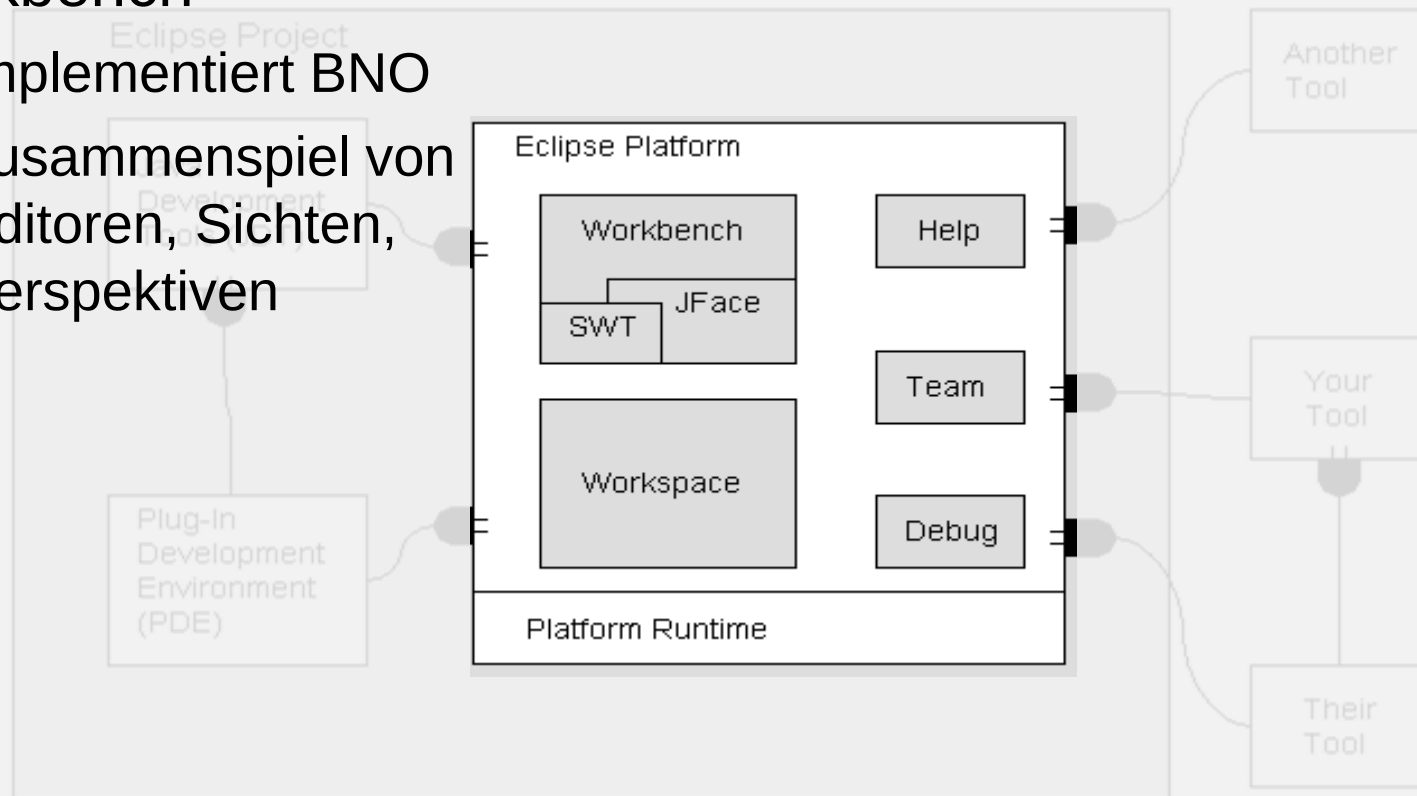
- Workspace

- besteht aus Projekten (jeweils auf ein Verzeichnis im Workspace abbildbar)
- alle Werkzeuge arbeiten mit Ressourcen aus Workspace des Nutzers



- Workbench

- Implementiert BNO
- Zusammenspiel von Editoren, Sichten, Perspektiven



Workbench (3.0.1)

The screenshot displays the Eclipse IDE in the Java Workbench perspective. The Package Explorer on the left shows a project structure with a package named 'etis' containing several Java files. The central Editor window shows the code for 'ABauteil.java', which defines an abstract class 'ABauteil' with methods for setting and adding 'Lieferant' (supplier) elements. The Problems view at the bottom lists two issues: a warning that the method 'addLieferant' must return a boolean, and an error that the import 'java.util.Iterator' is never used.

Perspektive

Editor

View

```
package etis;

import java.util.ArrayList;

public abstract class ABauteil {

    public ABauteil(String name) {

        /**
         * @uml.property name="lieferant"
         * @uml.associationEnd inverse="aBauteil:etis.Lieferant" multiplicity="(0 -1)"
         * @uml.association name="liefert"
         */
        private Collection lieferant;

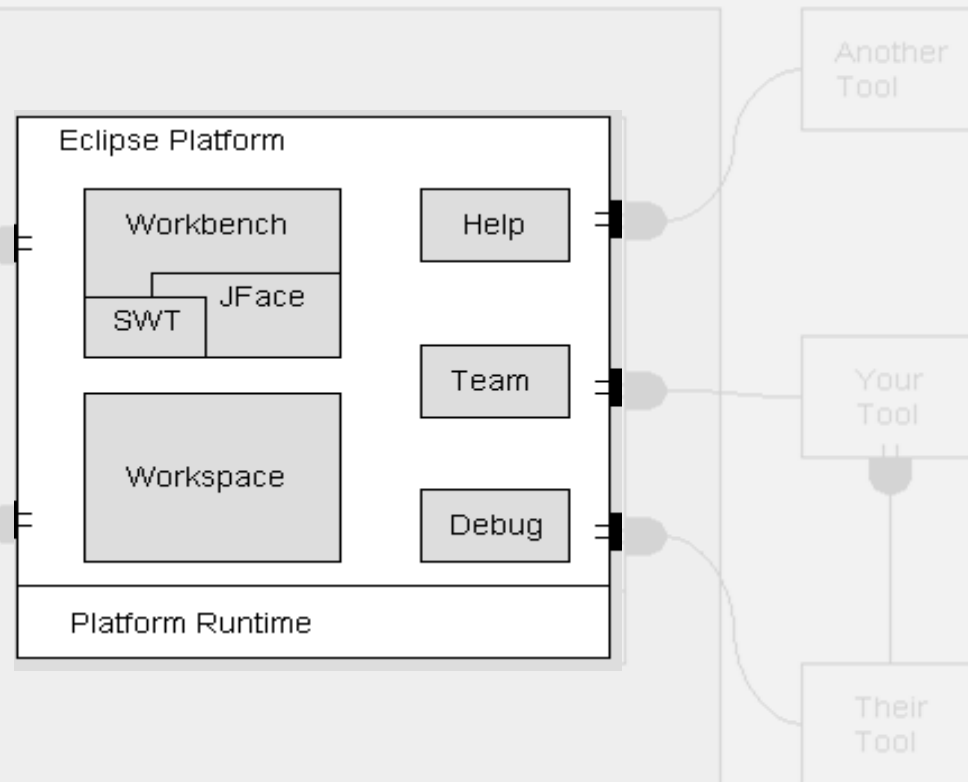
        /**
         * @uml.property name="lieferant"
         */
        public void setLieferant(java.util.Collection value) {
            lieferant = value;
        }

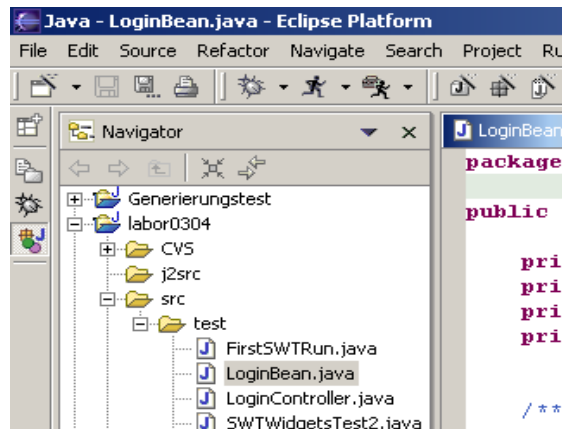
        /**
         * @uml.property name="lieferant"
         */
        public boolean addLieferant(etis.Lieferant element) {
            //return lieferant.add(element);
        }
    }
}
```

Description	Resource	In Folder	Location
This method must return a result of type boolean	ABauteil.java	ETISUML2/src/etis	line 34
The import java.util.Iterator is never used	BauteilGruppe.java	ETISUML2/src/etis	line 13

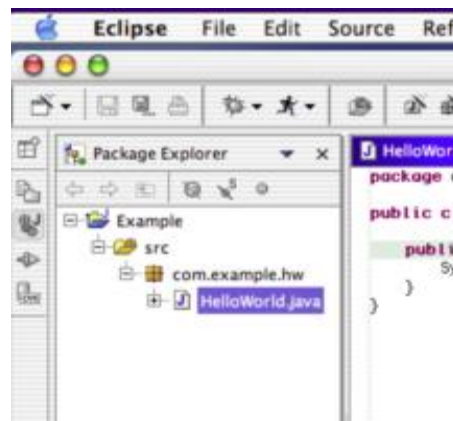


- SWT (Standard Widget Toolkit)
 - Bereitstellung GUI-Komponenten (Button, Trees, ...)
 - OS-unabhängige API
 - nutzt plattform-eigene Widgets oder emuliert diese

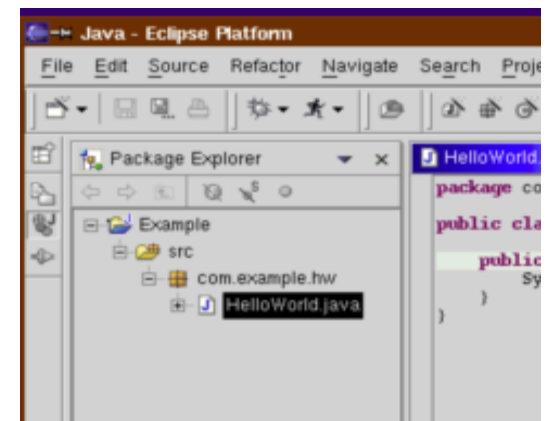




Eclipse auf Windows
XP



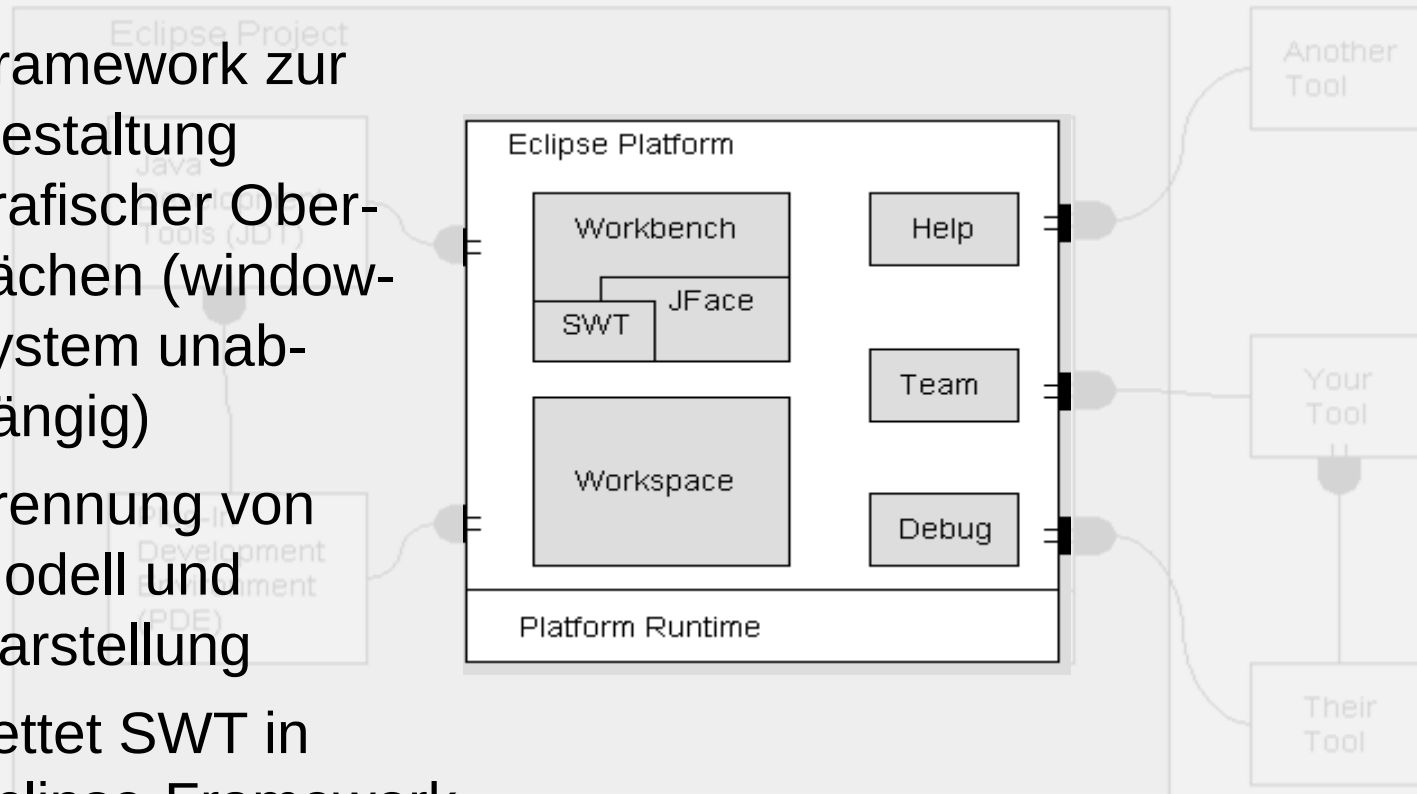
Eclipse auf
Mac OS X (Carbon)



Eclipse auf
Linux (Motif)

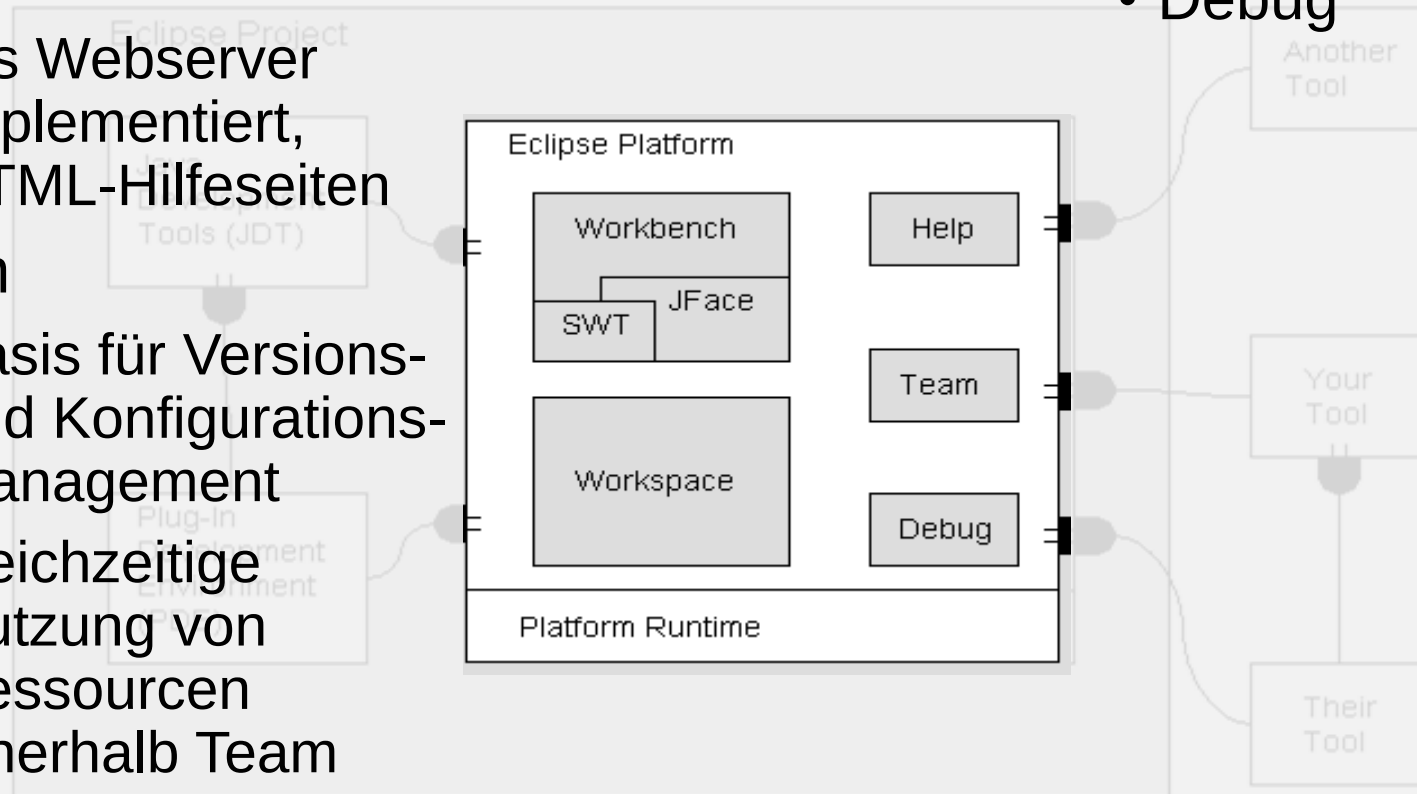


- JFace
 - Framework zur Gestaltung grafischer Oberflächen (window-system unabhängig)
 - Trennung von Modell und Darstellung
 - bettet SWT in Eclipse-Framework



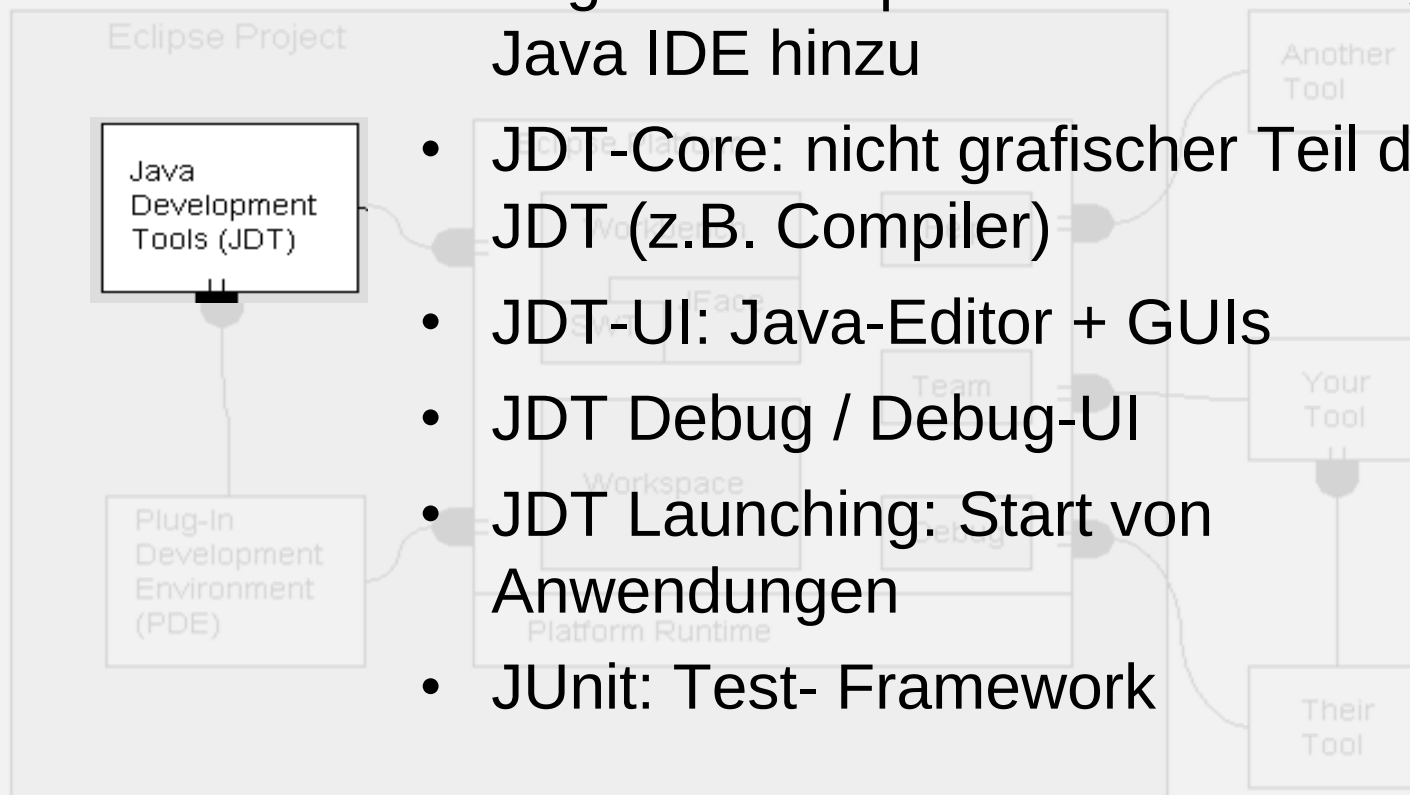
- Help
 - Als Webserver implementiert, HTML-Hilfeseiten
- Team
 - Basis für Versions- und Konfigurationsmanagement
 - gleichzeitige Nutzung von Ressourcen innerhalb Team

• Debug

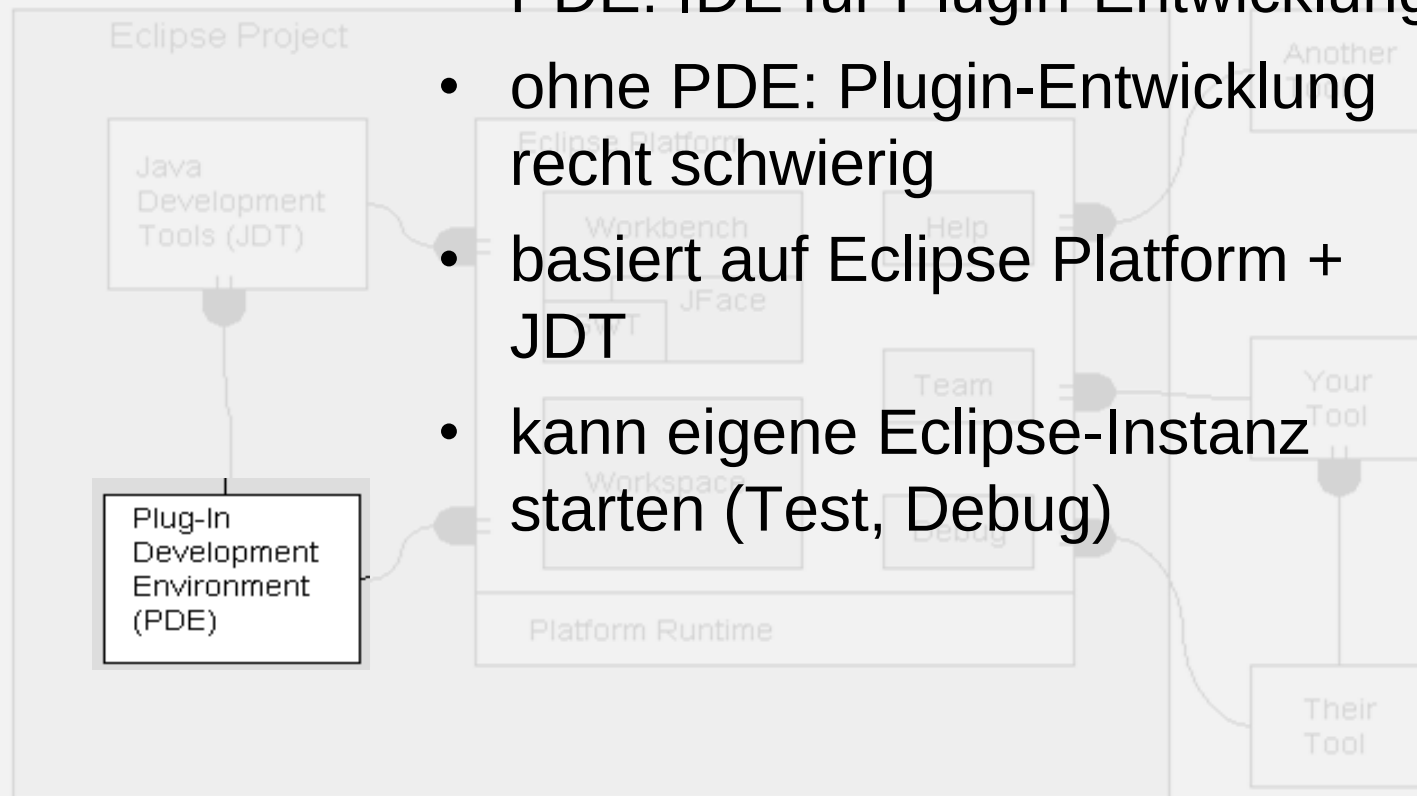


Java Development Tools (JDT)

- Fügen zu Eclipse eine vollständige Java IDE hinzu
- JDT-Core: nicht grafischer Teil des JDT (z.B. Compiler)
- JDT-UI: Java-Editor + GUIs
- JDT Debug / Debug-UI
- JDT Launching: Start von Anwendungen
- JUnit: Test- Framework



- PDE: IDE für Plugin-Entwicklung
- ohne PDE: Plugin-Entwicklung recht schwierig
- basiert auf Eclipse Platform + JDT
- kann eigene Eclipse-Instanz starten (Test, Debug)



- open source Entwicklungsumgebung
 - Nachteil: Standardeditoren müssen z.T. als Plugins nachgerüstet werden (JSP, XML)
- Plattform für Werkzeugintegration
 - Anpassbarkeit + Erweiterbarkeit durch wieder-verwendbare Komponenten
- Werkzeug für schnelle effiziente Werkzeugimplementierung
 - Große Community z.Z. > 800 Plugins
- beinhaltet neue GUI für Java-Applikationen
- Eclipse auch als Application Framework nutzbar



- Backschat, M., Edlich, J2EE-Entwicklung mit Open-Source-Tools, Spektrum Akademischer Verlag, München, 2004
- http://www.eclipse.org/eclipse/presentation/eclipse-slides_files/frame.htm
- Eclipse Homepage: www.eclipse.org
- Eclipse Plattform Technical Overview. Object Technology International, Inc., 02/2003
- Markus Weyerhäuser: Die Programmierumgebung Eclipse. JAVASpektrum, 02/2003
- Gamma, E., Beck, K., Contributing to eclipse, Addison-Wesley, Bosten, 2004
- Daum, B., Java-Entwicklung mit Eclipse 3, dpunkt.verlag, Heidelberg, 2005





```
7      if (_pinNames2PortNumber == null) {
8          _pinNames2PortNumber = new HashMap();
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")
22     {
23 com::itiv::generalStore::tools::ascettool::ascet12::blockdiagram::MBlockDiagramElementPin pin
24         if (pin.isInput()) {
25             if (_block2ListOfInputPins == null) {
26                 _block2ListOfInputPins = new HashMap();
27             }
28 /*         if ("C".equals(functionalElement.getAscetModelElement().getCodeComponent().getComponent
29 com::itiv::generalStore::tools::ascettool::ascet12::database::MCodeComponent cc1 = (com
30         if ("C".equals(cc1.getComponentType().toString())) {
31             if (_block2ListOfInputPins.containsKey(theMSDElement)) {
32                 java::util::List pinNameList = (java::util::List) _block2ListOfInputPins.get(theMSDE
33                 if (pinNameList.indexOf(thePinName) != -1) {
34                     pinPos = pinNameList.indexOf(thePinName);
35                     pinPos = pinPos + 2;
36                     portNumber = "" + pinPos;
37                 } else {
38                     pinNameList.add(thePinName);
```

- 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.

- Formatierung (Layout)
 - z.B.
for(i=0;i<grenze;i++)foo(i);
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*
- Implementierungs-Dokumentation
 - Vorspann
 - Klassen Kurzbeschreibung
 - Parameter Beschreibung
 - Vorbedingungen

→ wird von IDE unterstützt

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>  
}
```

- Java: geosoft.no/development/javastyle.html
- Java: www.horstmann.com/bigj/style.html
- Java: developer.netscape.com/docs/technote/java/codestyle.html
- C++: geosoft.no/development/cppstyle.html
- C++: oss.software.ibm.com/icu/userguide/conventions.html
- C++/Java: www.agsrhichome.bnl.gov/Controls/doc/codingGuidelines/codingGuidelines.html
- .NET: msdn.microsoft.com/library/default.asp?url=/library/enu/s/cpgenref/html/cpconnetframeworkdesignguidelines.asp
- PERL and shell-level:
dev.panopticsearch.com/portable_coding.html
- PERL: www.lug.lk/~anu/misc/perl_coding_guidelines.txt

- JavaDoc ist ein Programm, das aus Java-Quelltexten Dokumentationen im HTML-Format erstellt.
- Es befindet sich im Ordner /bin des sdk's



Class

PREV

Class Tree

PREV CLASS NEXT CLASS

Class vogel

java.lang.Object

+++vogelgesang

public class vogel

extends java.lang.Object

Programm zum Be

zum Erstellen vers

Version:

1.0 25 Apr

Author:

Tobias Frits

PREV

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 CLASS NEXT CLASS

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

FRAMES NO FRAMES

DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

Es gibt zwei Arten von Kommentaren:

- einzeilige Kommentare

`//` der Kommentar geht bis zum Ende der Zeile

- mehrzeilige Kommentare

`/*` Der Kommentar beginnt mit `/*`

und endet mit

`*/`

```
/**
 * 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;
}
```

`/**`

`* 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.

- @deprecated
 - **@deprecated** deprecated-text
 - vor allen Definitionen
 - Das mit dem deprecated-tag gekennzeichnete Objekt wird in einer späteren Version des Programmes möglicherweise nicht mehr enthalten sein. In dem Text kann angegeben werden, welches andere Objekt die Funktion übernimmt. Falls es keinen Ersatz gibt, sollte „No replacement“ angegeben werden.
 - Beispiel:
 - @deprecated wird ersetzt durch Equation.f
 - ==>
 - Deprecated. wird ersetzt durch Equation.fs

- @exception, @throws
 - **@exception** `class-name` `description`
 - vor Methoden-Definitionen
 - In der Methode wird die Ausnahme „`class-name`“ nach außen weitergereicht.
 - Beispiel:
 - @exception Keine Ausnahmen
 - ==>
 - Throws:
 - Keine - Ausnahmen

- @param
 - **@param** parameter-name description
 - vor Methoden-Definitionen
 - Beschreibt die Parameter einer Methode. Als Beschreibung kann beliebiger Text zur Erläuterung der Bedeutung und der Funktionsweise des Parameters angegeben werden.
 - Beispiel:

@param args Enthält die Parameter, die beim Start des Programmes übergeben werden.

==>

Parameters:

 - args - Enthält die Parameter, die beim Start des Programmes übergeben werden.

- @return
 - **@return** `description`
 - vor Methoden-Definitionen
 - Beschreibt den return-Wert einer Methode. Als Beschreibung sollten Typ und Wertebereich des return-Wertes angegeben werden.
 - Beispiel:
 @return float - Nullstelle der Funktion f.
 ==>
 Returns:
 - float - Nullstelle der Funktion f.

- @see
 - @**see** name label
 - vor allen Definitionen
 - Erzeugt eine Referenz auf ein anderes Objekt. Mögliche Formen sind:
 - @see package.class#member label
 - Referenz auf ein anderes Objekt
 - @see label
 - Referenz auf ein URL
 - @see "string"
 - allgemeine Referenz, z.B. auf ein Buch oder Artikel
 - Beispiel:
@see Hello#main main
==>
See Also:
[main](#)

- @link
 - {@link name label}
 - vor allen Definitionen
 - Kann genauso verwendet werden wie das see-tag, nur daß die Referenz im Text erzeugt wird und nicht als extra Absatz.
 - Beispiel:

Zur Berechnung der Nullstelle nutze man die Methode {@link equation#newton newton} der Klasse {@link equation equation}.

==>

Zur Berechnung der Nullstelle nutze man die Methode [newton](#) der Klasse [equation](#)

- @since
 - **@since** `since-text`
 - vor allen Definitionen
 - Erläutert seit wann das entsprechende Objekt verfügbar ist.
 - Beispiel:
 - @since Version 1.1.6
 - ==>
 - Since:
 - Version 1.1.6

- @version
 - **@version** `version-text`
 - vor Klassen-Definitionen
 - Gibt die Version der Klasse an.
 - version-tags werden übernommen, wenn
 - javadoc -version
 - benutzt wird.
 - Beispiel:
 @version 1.0
 ==>
 Version:
 – 1.0

```
@author name-text  
@deprecated deprecated-text  
@exception class-name description  
{@link name label}  
@param parameter-name description  
@return description  
@see reference  
@since since-text  
@serial field-description  
@serialField field-name field-type field-description  
@serialData data-description  
@throws class-name description  
@version version-text
```

tag	package	class interface	field	method constructor	javadoc
author		x			-author
deprecated	x	x	x	x	
exception				x	
link	x	x	x	x	
param				x	
return				x	
see	x	x	x	x	
since	x	x	x	x	
serial			x		
serialField			x		
serialData				x	
throws				x	
version		x			-version

1. Befehlszeilen-Kommando:

```
javadoc [options] [packagenames] [sourcefile] [classnames]
```

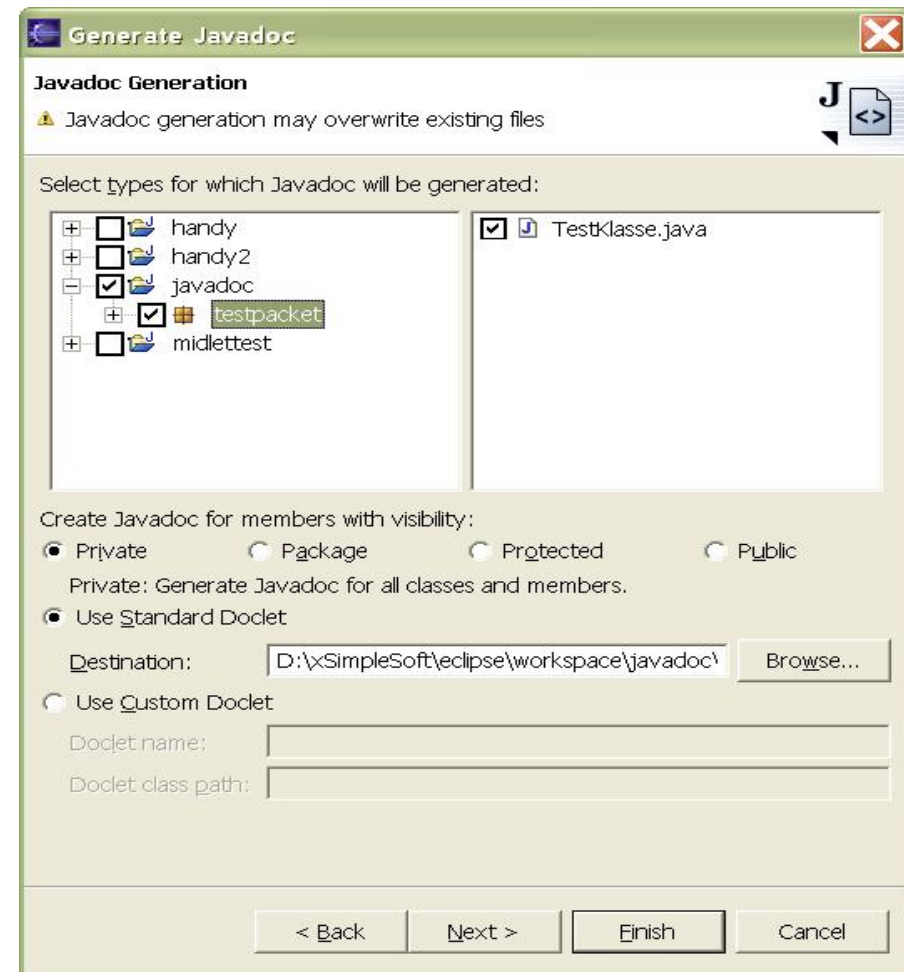
```
javadoc *.java
```

Erzeugt eine Html-Dokumentation aller java Dateien des aktuellen Verzeichnisses.

-public	Elemente des Typs public werden dokumentiert.
-protected	Elemente des Typs public und protected werden dokumentiert (das ist die Voreinstellung)
-package	Elemente des Typs package, public und protected werden dokumentiert.
-private	Alle Elemente werden dokumentiert.
-doclet	Alternatives Doclet.
-classpath	Gibt die Liste der Pfade zur Suche von Klassendateien an.

2. Eclipse:

Durch Auswahl des Menüpunktes
„**Datei -> Export -> JavaDoc**“
öffnet sich folgendes Fenster:





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

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

