Freie Universität Berlin

3D-Visualisierung von Software am Beispiel von CodeCity und ConeTrees

Ahmet-Serdar Karakaya

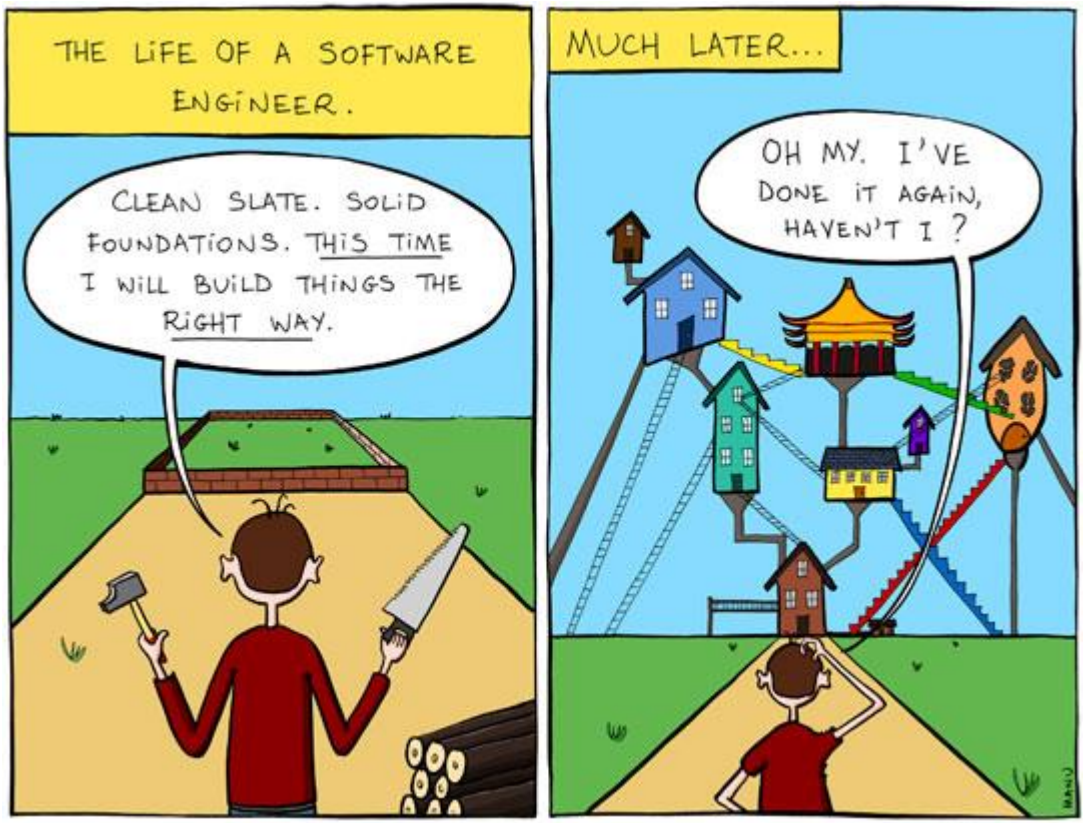
Freie Universität Berlin
Institut für Informatik

11.06.2015

Seminar "Beiträge zum Software Engineering"

Gliederung

- Einleitung
- Reverse Engineering
- Software-Visualisierung
- Beispiel 1 – Cone Trees
- Beispiel 2 – CodeCity
- Evaluation


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?

Einleitung

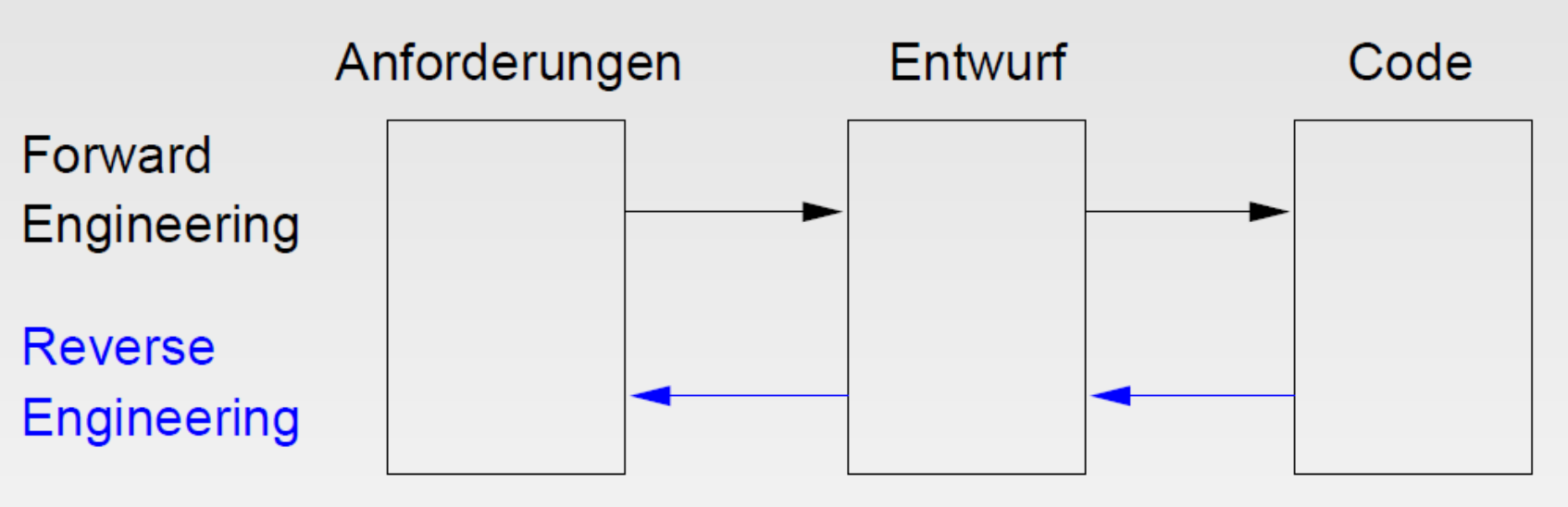
- Software Engineering: The systematic approach to the **development**, operation, **maintenance**, and retirement of software
  - IEEE Standard Glossary of Software Engineering Terminology (ANSI 83)

- Guide to the Software Engineering Body of Knowledge (von IEEE) zählt folgende Wissensgebiete auf:
  1. Software Requirements
  2. **Software Design**
  3. **Software Construction**
  4. **Software Testing**
  5. **Software Maintenance**
  6. Software Configuration Management
  7. Software Engineering Management
  8. Software Engineering Process
  9. Software Engineering Tools and Methods
  10. **Software Quality**

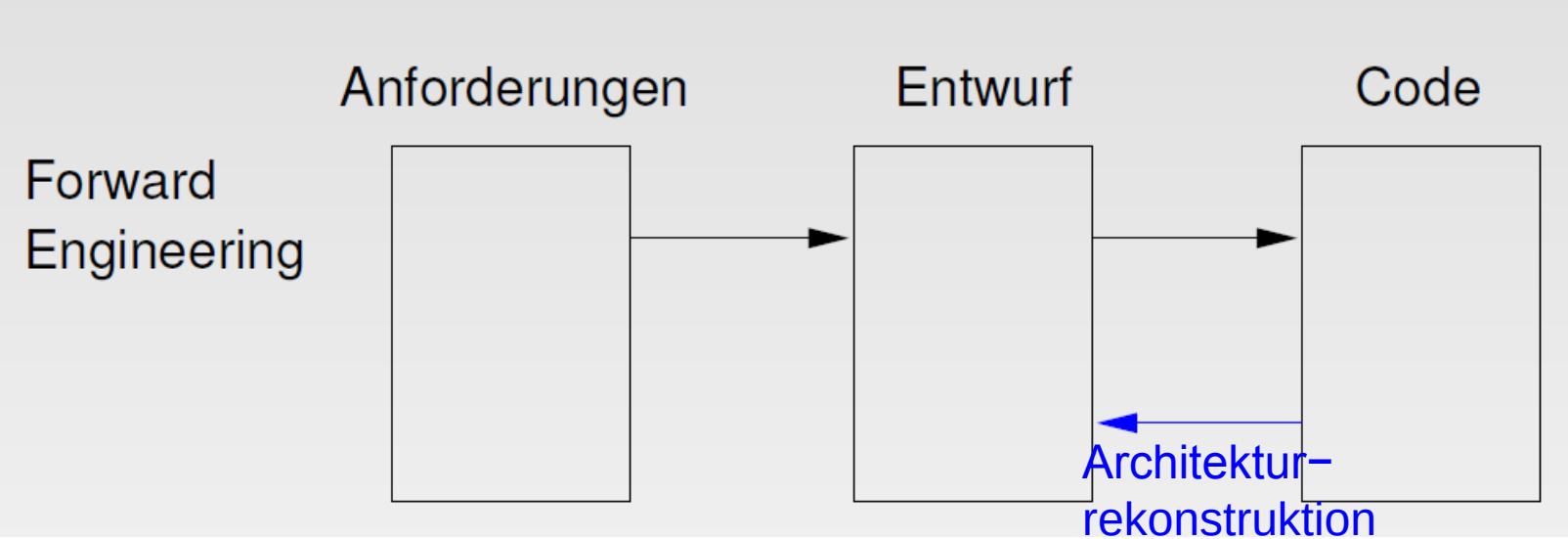
Gliederung

- ✓ Einleitung
- ➢ Reverse-Engineering
- Software-Visualisierung
- Beispiel 1 – Cone Trees
- Beispiel 2 – CodeCity
- Evaluation

Reverse Engineering



Reverse Engineering



Reverse Engineering

- *„Software-Systeme zu erstellen, die **nicht geändert** werden müssen, ist **unmöglich**. Wurde Software erst einmal in Betrieb genommen, entstehen **neue** Anforderungen und vorhandene Anforderungen **ändern** sich ... “*

  - I. Sommerville

- Software-Evolution
  - fortgesetzter Wandel
  - ansteigende Komplexität

  ⟹ ständige Anpassung erforderlich

  ⟹ Begrenzung der Komplexität erforderlich

Reverse Engineering

- *„Software-Systeme zu erstellen, die **nicht geändert** werden müssen, ist **unmöglich**. Wurde Software erst einmal in Betrieb genommen, entstehen **neue** Anforderungen und vorhandene Anforderungen **ändern** sich ... “*

  - I. Sommerville

- Software-Evolution
  - fortgesetzter Wandel
  - ansteigende Komplexität

- Wünsche
  - Kontrollierte Wartung der Software ⟹ Design bleibt erhalten
  - Konsistenz aller Dokumente bleibt erhalten

Reverse Engineering

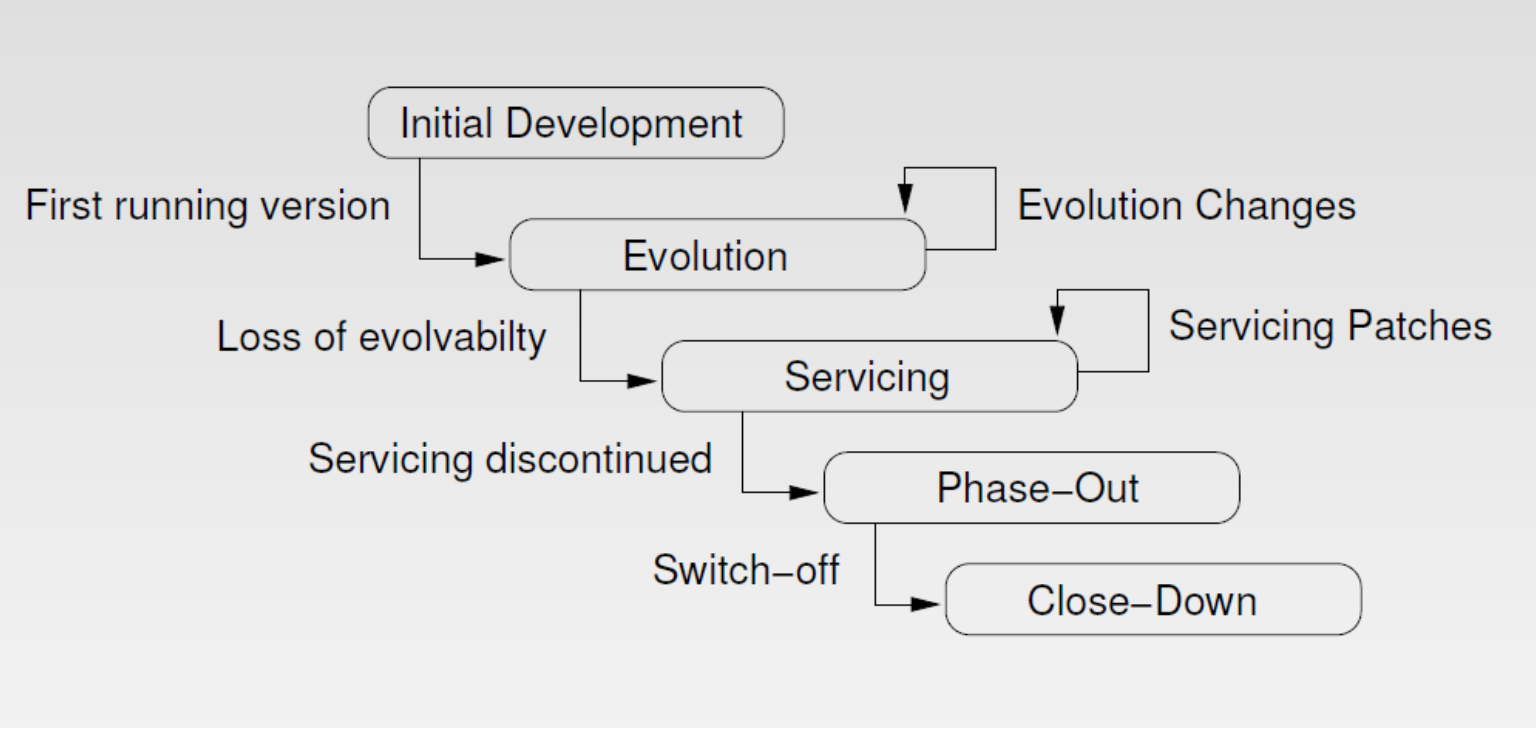
- *„Software-Systeme zu erstellen, die **nicht geändert** werden müssen, ist **unmöglich**. Wurde Software erst einmal in Betrieb genommen, entstehen **neue** Anforderungen und vorhandene Anforderungen **ändern** sich ... “*

  - I. Sommerville

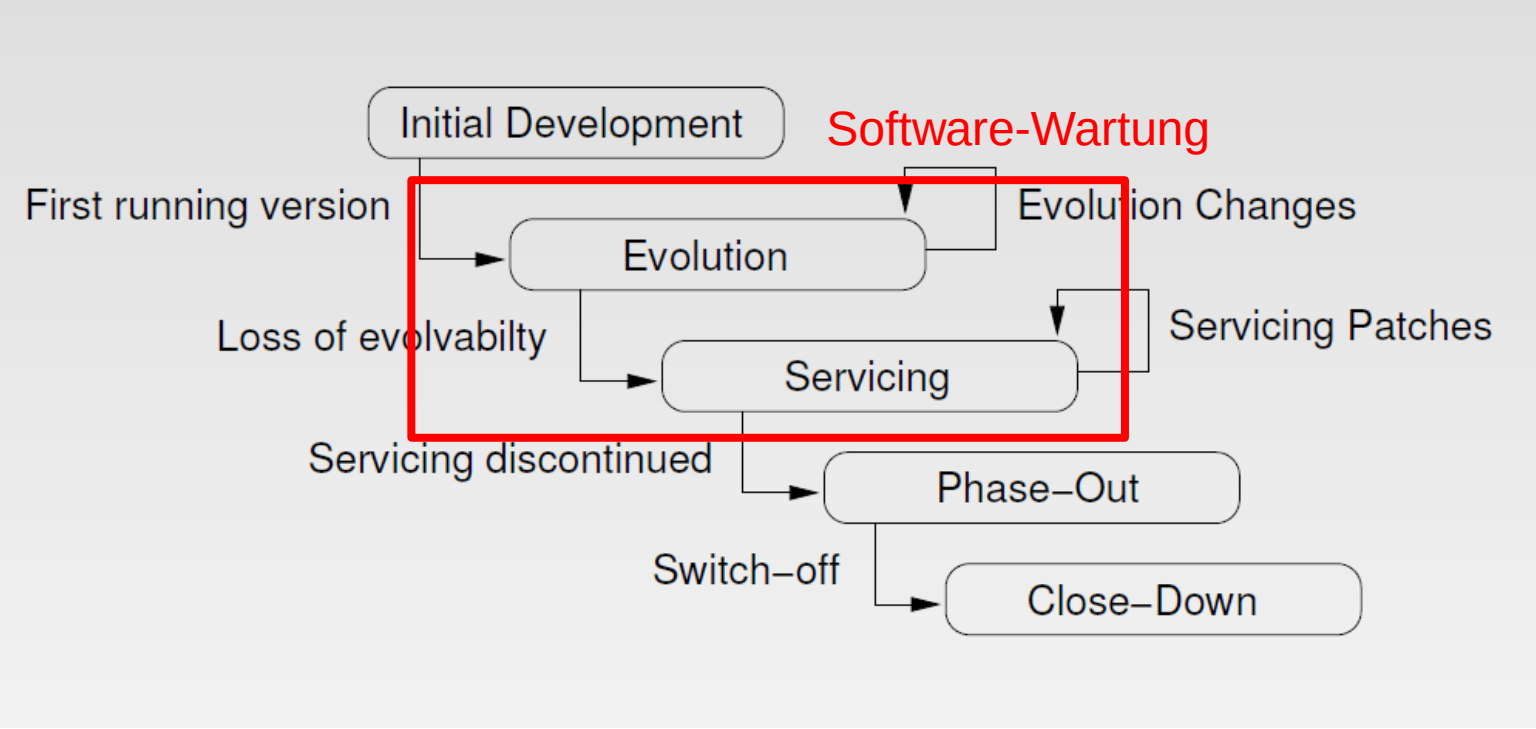
- Software-Evolution
  - fortgesetzter Wandel
  - ansteigende Komplexität

- Realität:
  - ursprüngliche Systemstruktur wird ignoriert
  - Dokumentation ist unvollständig/veraltet
  - Mitarbeiter verlassen Projekt

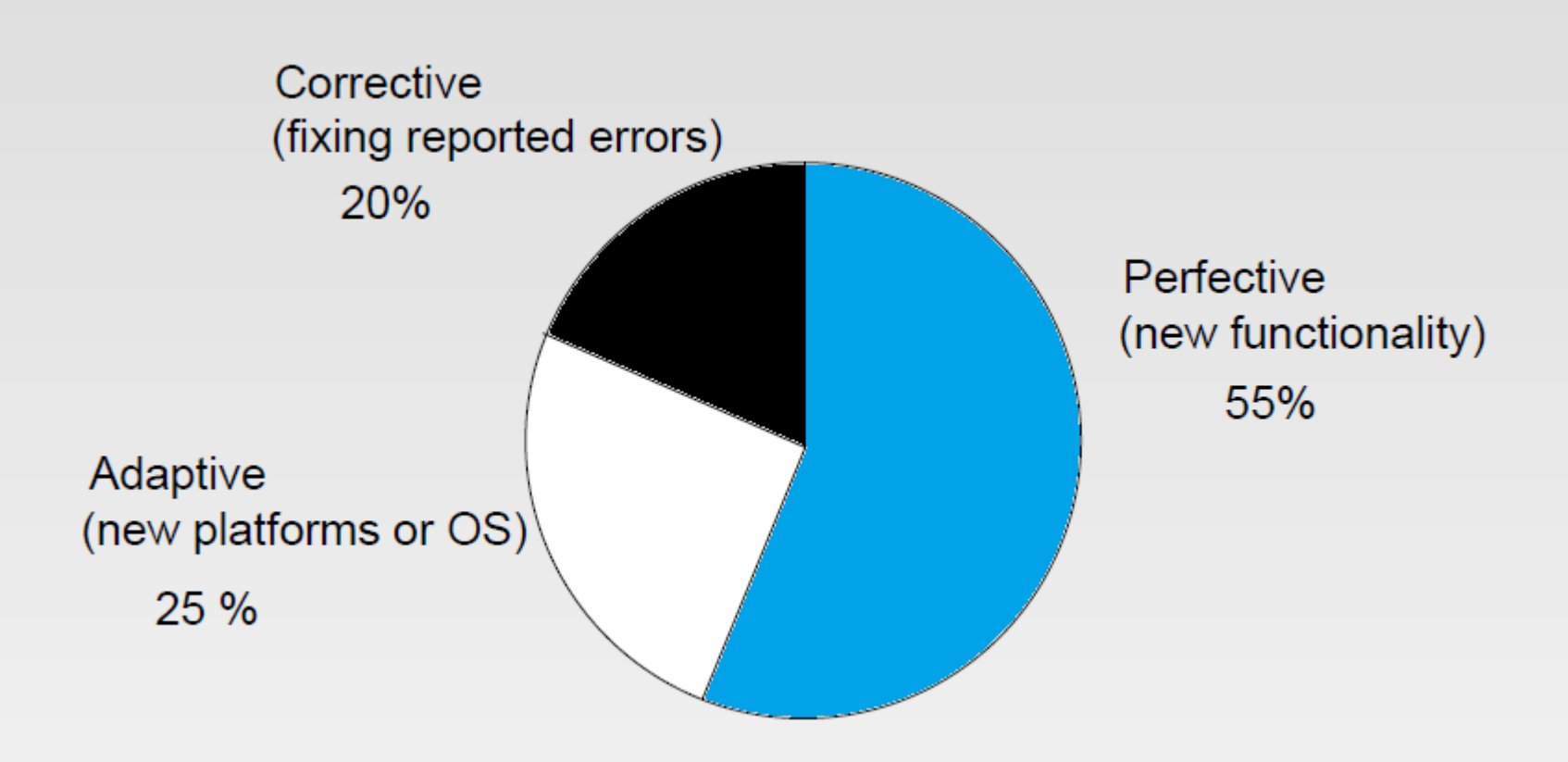
Reverse Engineering



Reverse Engineering

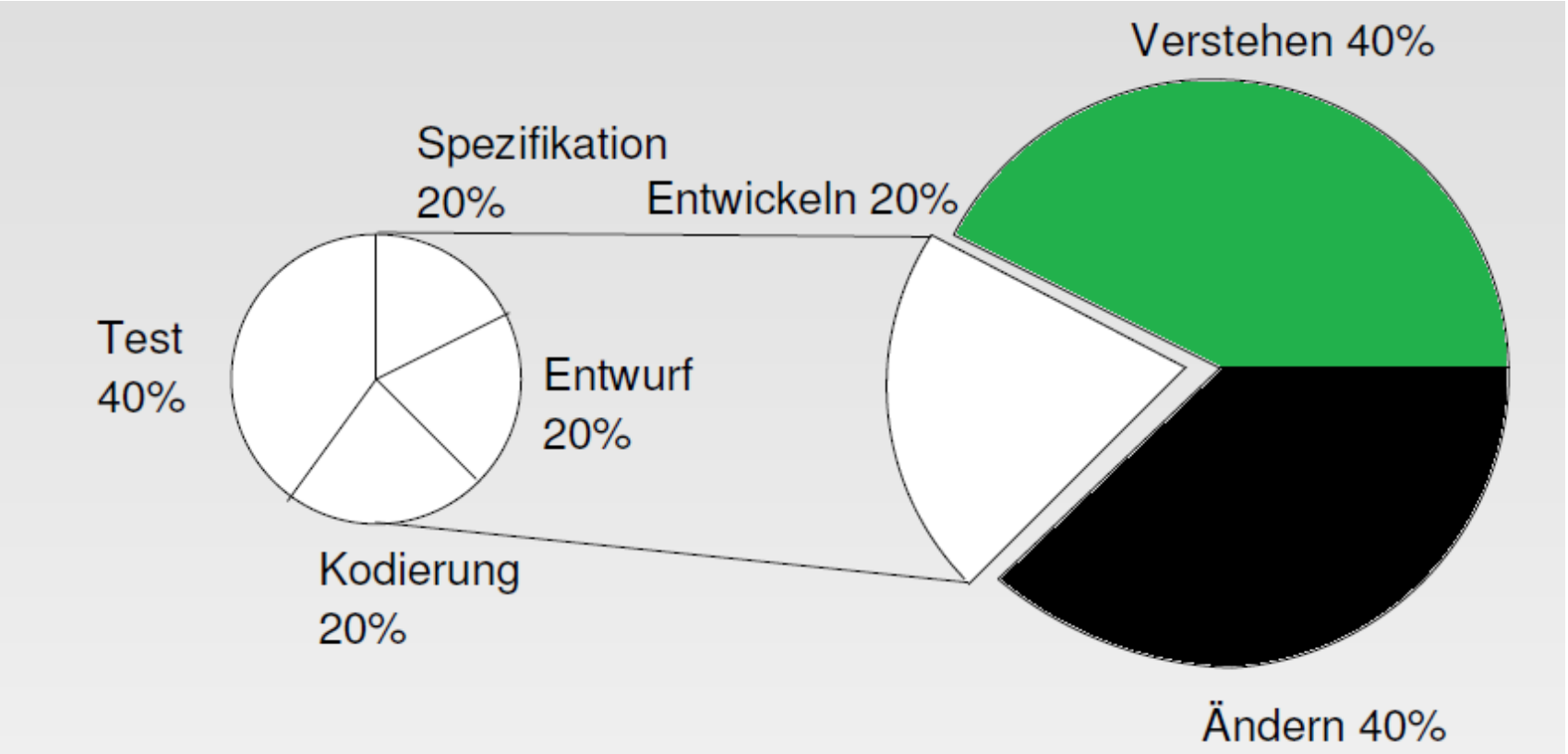


Reverse Engineering



- E.B. Swanson (1980)

Reverse Engineering



- Fjedstad und Hamlen (1979)

Reverse Engineering

- Ziele:
  - Komplexität kontrollieren
  - Alternative Sichtweisen
  - Wiedererhaltung verloren gegangener Information
  - Erkennung von Wirkungen/Nebeneffekten
  - Abstraktion
  - Wiederverwendung

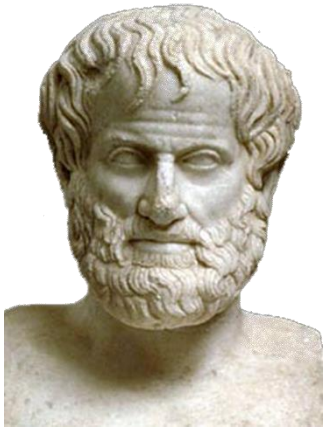
Gliederung

- ✓ Einleitung
- ✓ Reverse-Engineering
- ➢ Software-Visualisierung
- Beispiel 1 – Cone Trees
- Beispiel 2 – CodeCity
- Evaluation

Software-Visualisierung

„Die Seele kann nicht *ohne Bilder* denken“

-Aristoteles



„Imagination oder Visualisierung und besonders die Benutzung von Diagrammen haben einen entscheidenden Anteil an der wissenschaftlichen Forschung“

-René Descartes

Software-Visualisierung

- Software-Visualisierung:
  - visuelle Repräsentationen verschiedener Aspekte einer Software
  - Ziel: Softwaresysteme und Softwareentwicklungsprozesse zu verbessern

Software-Visualisierung

- Software-Visualisierung:
  - visuelle Repräsentationen verschiedener Aspekte einer Software
  - Ziel: Softwaresysteme und Softwareentwicklungsprozesse zu verbessern

- Ziele (von Reverse Engineering):
  - **Komplexität kontrollieren**
  - **Alternative Sichtweisen**
  - Wiedererhaltung verloren gegangener Information
  - Erkennung von Wirkungen/Nebeneffekten
  - **Abstraktion**
  - Wiederverwendung

Software-Visualisierung

- Software-Visualisierung:
  - visuelle Repräsentationen verschiedener Aspekte einer Software
  - Ziel: Softwaresysteme und Softwareentwicklungsprozesse zu verbessern

- Herausforderungen:
  - Viele Informationen
  - Verschiedene Aufgaben
  - Verschiedene Arten
  - Begrenzte Ressourcen

Software-Visualisierung

- Mögliche Fragen:
  - Wie groß ist das Softwaresystem und wie ist es strukturiert?
  - Welcher Architekturstil wird verwendet und was sind die Subsysteme?
  - Welche sind die wichtigsten Klassen und Klassenhierarchien?
  - Wie ist die Qualität der Software? Sind alle Subsysteme qualitativ gleich? (Implementierung und Wiederverwendbarkeit)
  - Welche Entwurfsmuster wurden wo eingesetzt?

Software-Visualisierung

- Software-Visualisierung sehr wichtig für Reverse-Engineering
- Keine SV kann „alles“ zeigen
- Verschiedene Arten der SV für verschiedene Aufgaben
- Noch viele Forschungsfragen
  - Noch keine Standards in der Notation
  - Bildschirm effektiver Nutzen
  - …

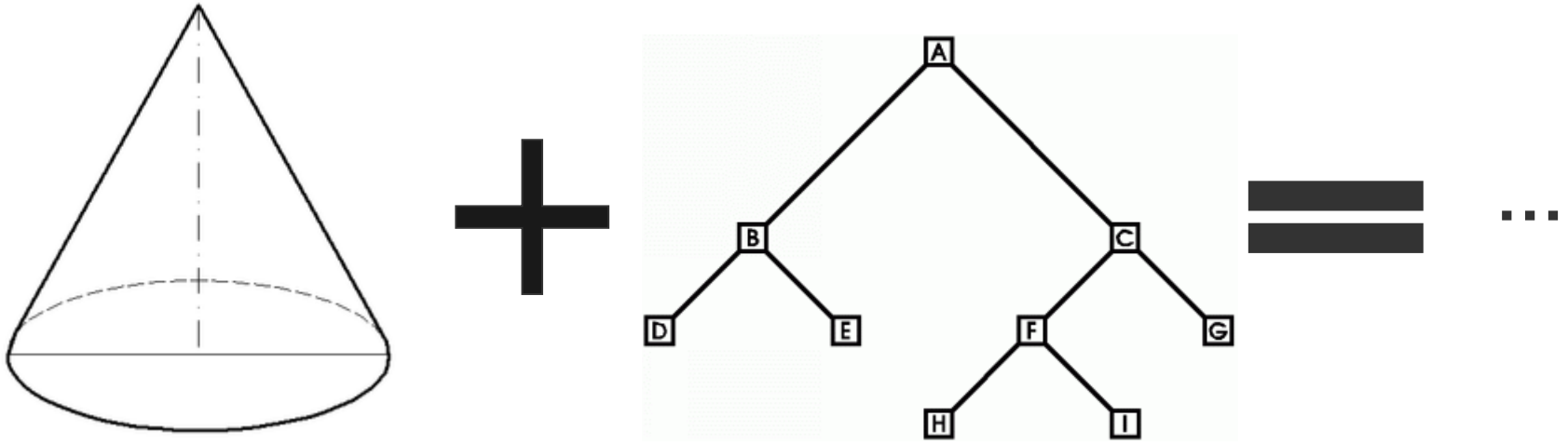
Gliederung

- ✓ Einleitung
- ✓ Reverse-Engineering
- ✓ Software-Visualisierung
- ➢ Beispiel 1 – Cone Trees
- Beispiel 2 – CodeCity
- Evaluation

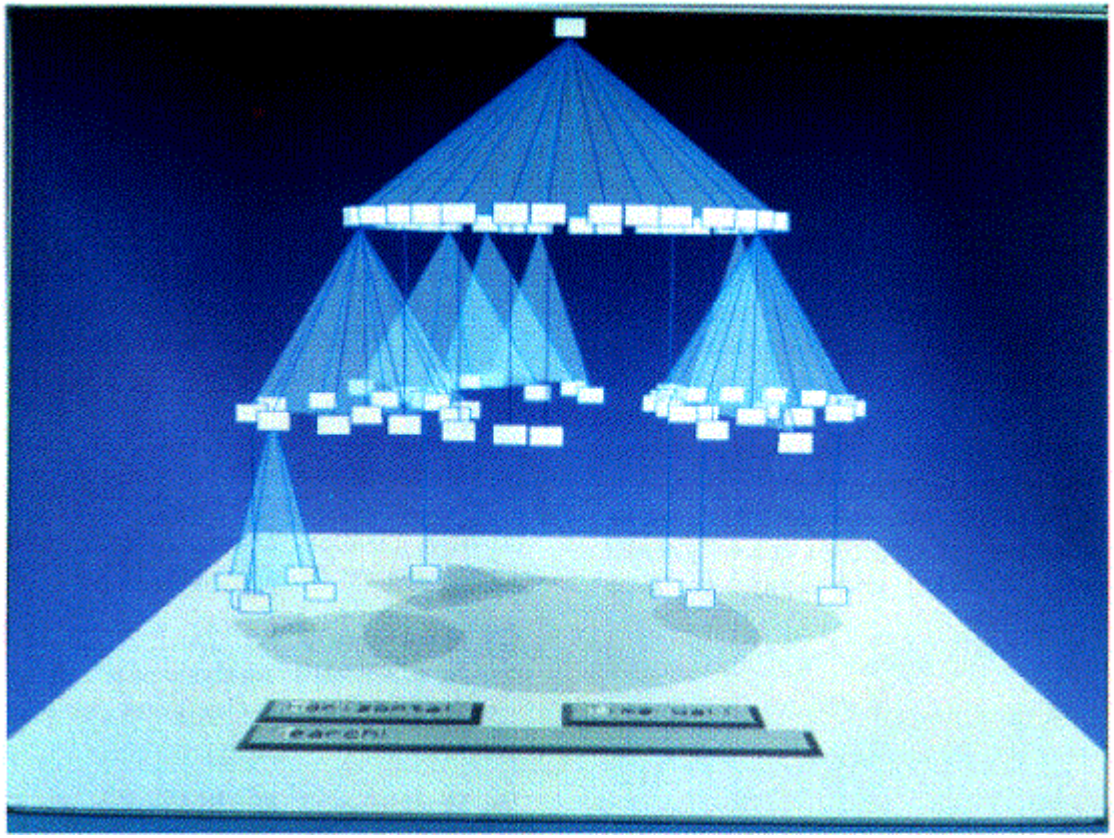
Cone Trees

- Entwickelt:
  - Von G.G. Robertson, J.D. Mackinlay und S.K. Card
  - 1991
  - Palo Alto, Kalifornien, USA („Silicon Valley“)

- Zweck: Hierarchie visualisieren

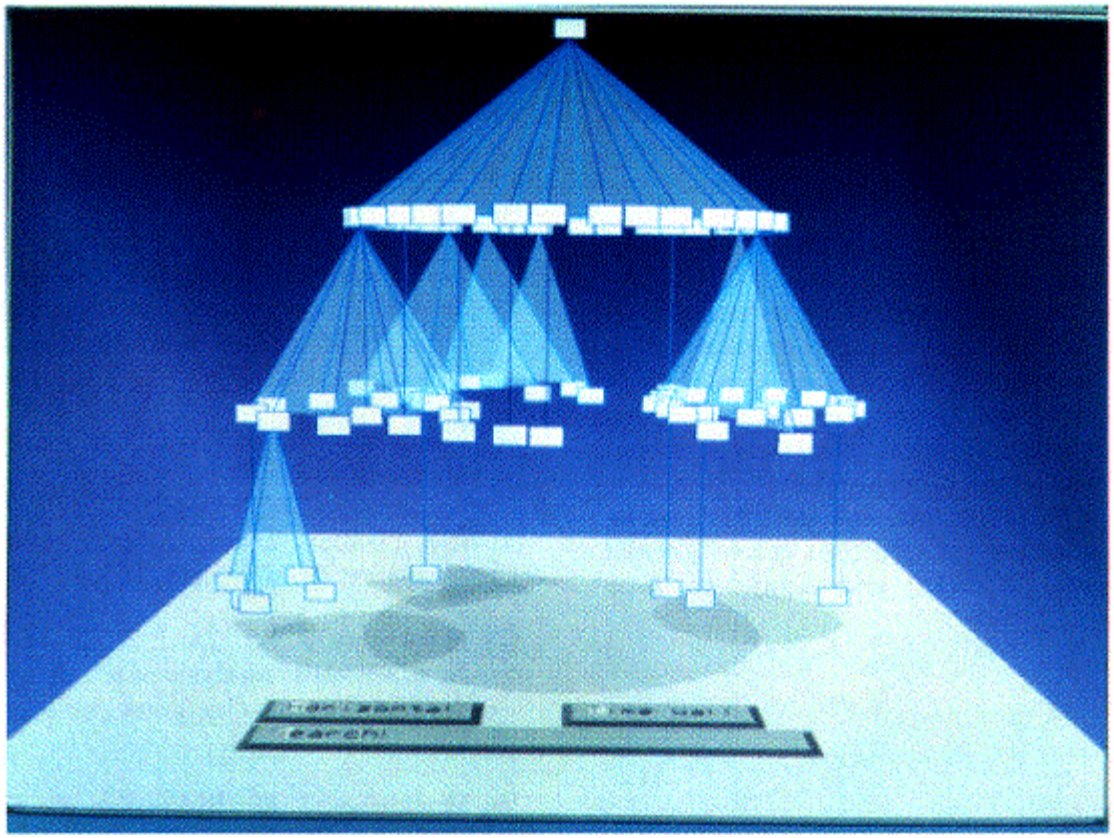


Cone Trees



Cone Trees

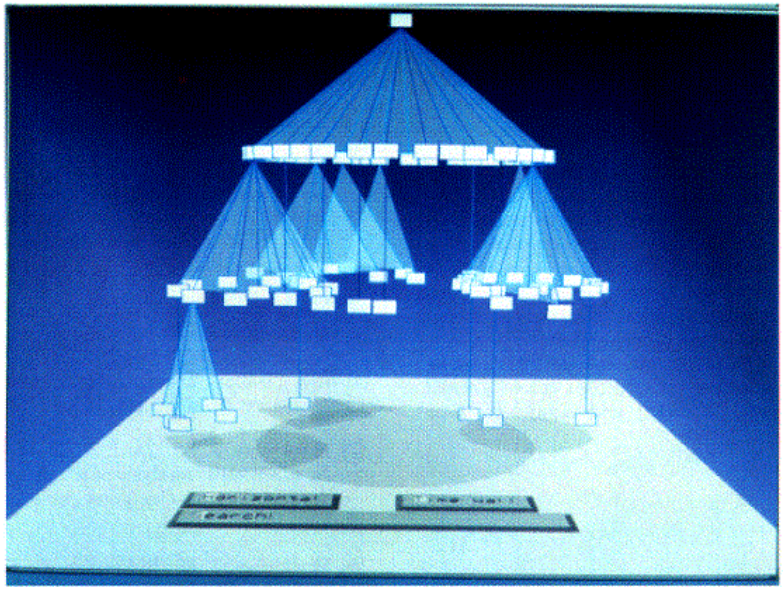
- Ausgangssituation:
  - Nur erste Ebene
  - Alles aufgedeckt
- Navigation:
  - Auf- und zuklappen von Teilbäumen
  - Suchfunktion
  - Rotation des Kegels
- Visualisierung:
  - 3D
  - 2D (Schatten)
  - Blinken bei Selektion
  - Lichteffekte

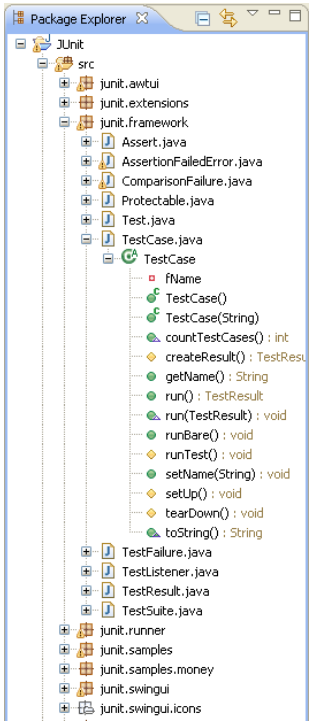


Cone Trees

- Vorteile:
  - Raumnutzung
  - Fokus ist wählbar
  - skaliert

- Nachteile:
  - Mangelnde Tiefe
  - Navigation
  - Rotation des Kegels
  - Skaliert mäßig
  - „nur“ Hierarchie





Gliederung

- ✓ Einleitung
- ✓ Reverse-Engineering
- ✓ Software-Visualisierung
- ✓ Beispiel 1 – Cone Trees
- ➢ Beispiel 2 – CodeCity
- Evaluation

CodeCity

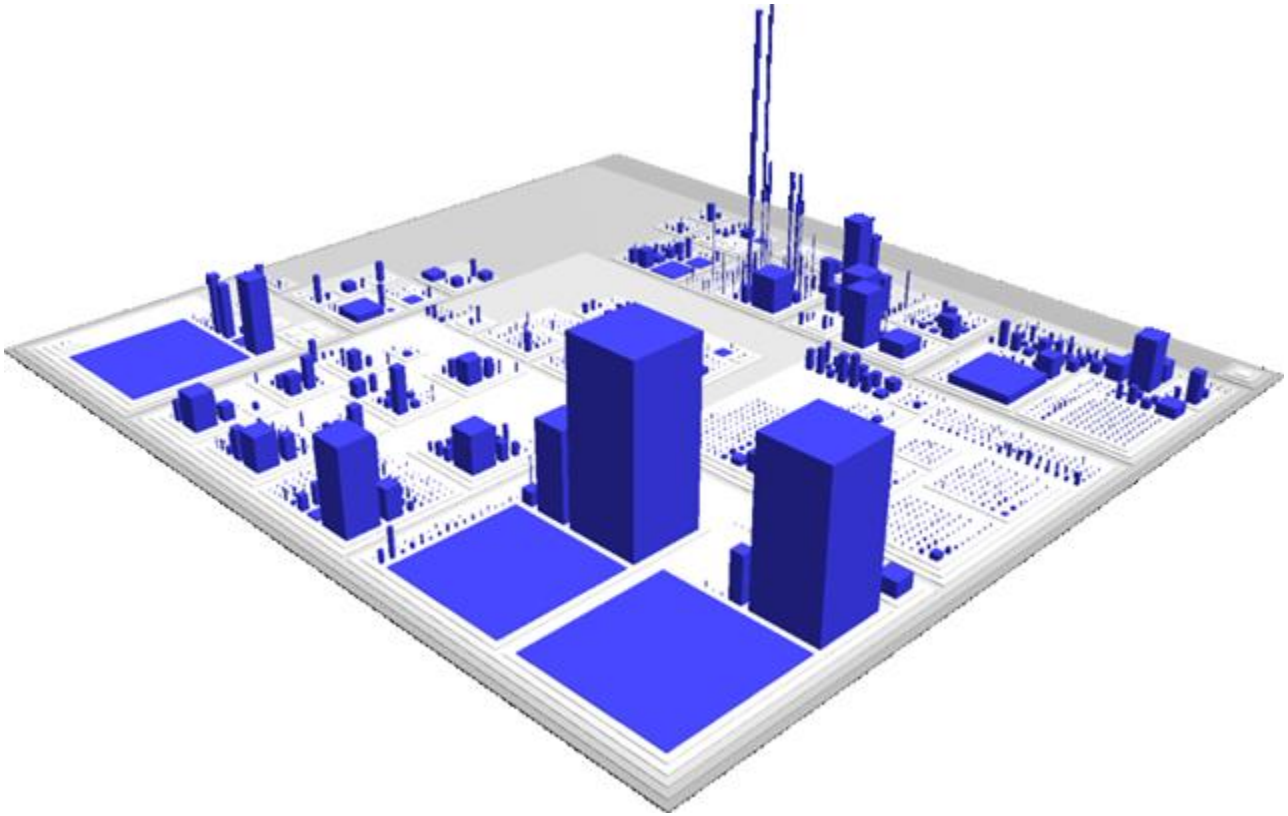
- Entwickelt:
  - Von Richard Wettel und Michele Lanza
  - 2008
  - *Universität der italienischen Schweiz*, Lugano, Schweiz

- Zweck: Große objektorientierte Softwaresysteme analysieren

- Stadt-Metapher:
  - Pakete bilden Bezirke
  - Klassen mit Methoden: Gebäude
  - Klassen ohne Methoden: Parks bzw. Grünanlagen

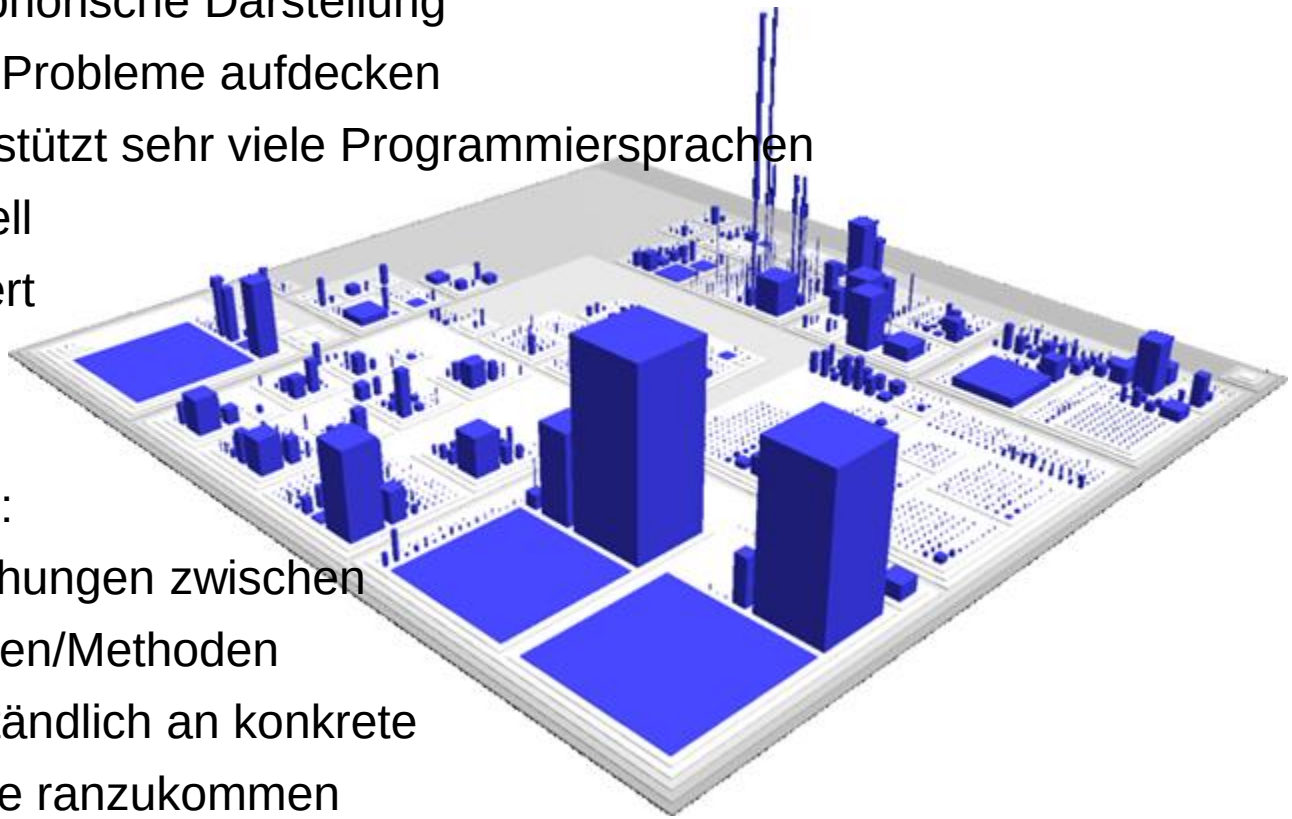
CodeCity

- Live-Demo



CodeCity

- Vorteile:
  - Metaphorische Darstellung
  - Kann Probleme aufdecken
  - Unterstützt sehr viele Programmiersprachen
  - Schnell
  - Skaliert



- Nachteile:
  - Beziehungen zwischen Klassen/Methoden
  - Umständlich an konkrete Werte ranzukommen

Gliederung

- ✓ Einleitung
- ✓ Reverse-Engineering
- ✓ Software-Visualisierung
- ✓ Beispiel 1 – Cone Trees
- ✓ Beispiel 2 – CodeCity
- ➢ Evaluation

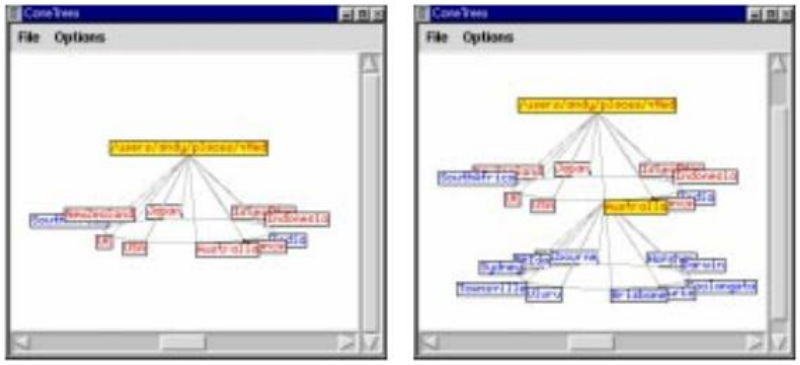
Evaluation

- Umfrage zur Software-Visualisierung im Bereich Reverse-Engineering
  - 2003 von R. Koschke durchgeführt
  - 82 Teilnehmer (aus der Forschung)

- Haupterkenntnisse:
  1. Forscher legen viel Wert auf Software-Visualisierung
  2. Visualisierung am meisten für „mid-class“-Artefakte verwendet
  3. Die meist verwendeten SVen sind nicht animiert, was die Befragten aber bevorzugen würden

Evaluation

- Studie zu Cone Trees
  - 1995 von A. Cockburn und B. McKenzie
  - 12 Teilnehmer (Informatikabsolventen)
  - Aufgaben abwechselnd mit ConeTrees und „normalem“ Baum gelöst
  - Insgesamt 1 Stunde (10 Minuten für die Einweisung in Cone Trees)



(a) One cone. (b) Expanding one directory.

Figure 1: Basic cone tree interface.

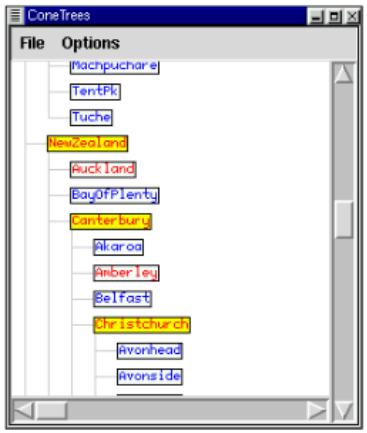


Figure 4: The ‘tree browser’ control interface.

Evaluation

- Ergebnisse
  - Alle Aufgaben mit „normaler“ Baumstruktur schneller
  - Vorsprung steigt mit der Dichtheit an Daten
  - Cone Trees wurden von Teilnehmern begrüßt

- Beurteilung der Studie:
  - 12 Informatikabsolventen etwas zu wenig
  - Nur 10 Minuten Einführung
  - Eigene Implementierung sehr „stumpf“
  - Textbasierte Aufgaben günstiger für „normale“ Baumstruktur
  - Reihenfolge bei der Benutzung der Tools verzerrt die Ergebnisse

Evaluation

- Kontrolliertes Experiment zu CodeCity
  - 2011 von R. Wettel, M. Lanza und R. Robbes durchgeführt
  - 41 Teilnehmer (21 aus Akademie, 20 aus Industrie)
  - 4 Gruppen: 2 verschiedene Tools und 2 verschiedene Systeme
    - Eclipse + Excel vs. CodeCity
    - Mittleres und großes System (Findbugs und Azureus)
  - 4 verschiedene Städte aus 3 Ländern
  - Verschiedene Aufgabentypen (Überblick, Details,…)
  - Gemessen wurden Korrektheit und Zeit für die Bearbeitung

Evaluation

- Ergebnisse:
  - 24% bessere Ergebnisse mit CodeCity
  - 12% weniger Zeitaufwand mit CodeCity
  - Detailaufgaben nur ein wenig besser
  - Überblicksaufgaben viel besser

- Beurteilung
  - Viel Wert auf interne und externe Gültigkeit gelegt
  - Autoren führen Studie über eigene Arbeit durch..

Freie Universität Berlin

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?

Literatur

- ”‘Object-Oriented Reverse Engineering - Coarse-grained, Fine-grained, and Evolutionary Software Visualization”’, Dissertation, Michele Lanza

- Fjedstad und Hamlen 1979 Fjedstad, R.K. ; Hamlen, W.T.: Application Program Maintenance Study: Report to our Respondents. In: Proceedings of the GUIDE 48. Philadelphia, PA, 1979

- E.B. Swanson 1980 E.B. Swanson, B.P. L. und: Software Maintenance Management. Reading, MA : Addison-Wesley, 1980

- Chikofsky und Cross II. 1990 Chikofsky, Elliot J. ; Cross II., James H.: Reverse Engineering and Design Recovery: A Taxonomy. In: IEEE Software 7 (1990), Januar, Nr. 1, S. 13–17

- Stephan Diel 2007 Software Visualization