Forschungszentrum Informatik
an der Universität Karlsruhe

Werkzeuggestützte Softwarebewertung

Grundlagen und Fallbeispiele

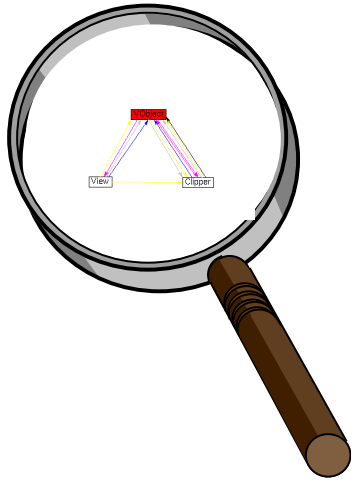
Markus Bauer

Programmstrukturen

Förderverein FZI, 31.3.2004

Übersicht

- Einführung
- Softwarequalität und Design
- Techniken zur Qualitätsuntersuchung
  - Architekturuntersuchung
  - Metrik- und musterbasierte Schwachstellensuche
  - Codeduplikation
- Erfahrungen



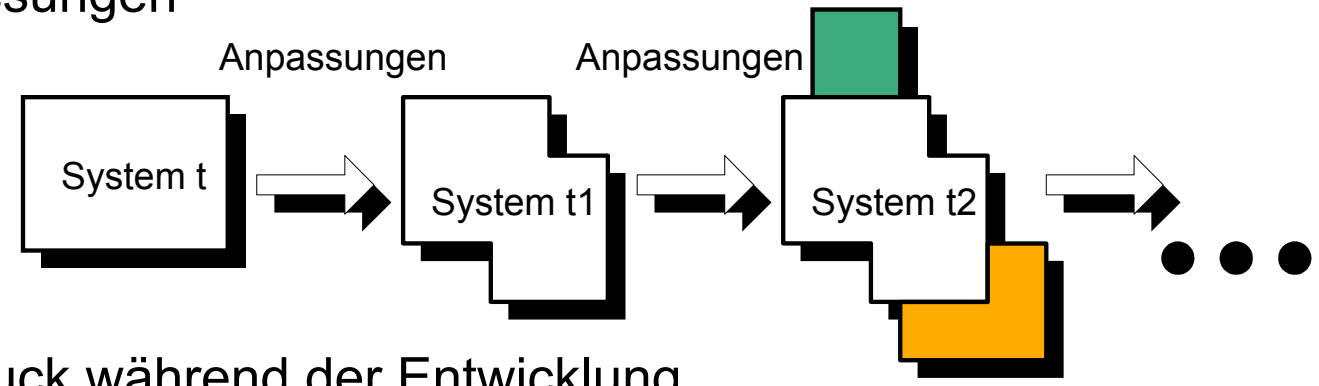
Softwarequalität

- *„Qualität ist, wenn der Kunde wieder kommt, und nicht die Ware.”* (Hans-Helge Stechl, Vorstand BASF)
- Externe Qualität = Kundenperspektive:
  - Einfache Verwendbarkeit, Performance, Robustheit, ...
- Interne (Software-)Qualität = Entwicklerperspektive:
  - Flexibilität, Verständlichkeit, Wartbarkeit, ...
- Hypothese: Gute interne Qualität ist eine Voraussetzung für gute externe Qualität!

Qualitätsprobleme – Warum?

- Anforderungen an Softwaresysteme sind schwer zu erfassen und ändern sich ständig
  - Kluft zwischen Fachexperten und Softwareexperten
  - Lebensdauer moderner Systeme
  - Neue Einsatzkontexte

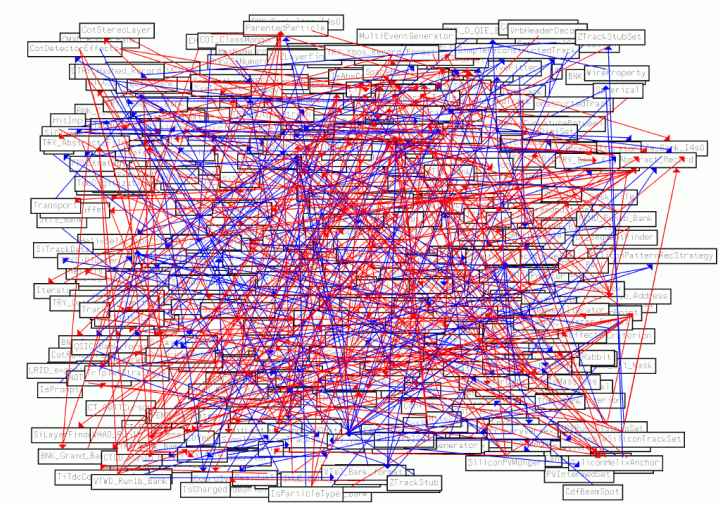
⇒ Softwarestrukturen degenerieren durch wiederholte Anpassungen

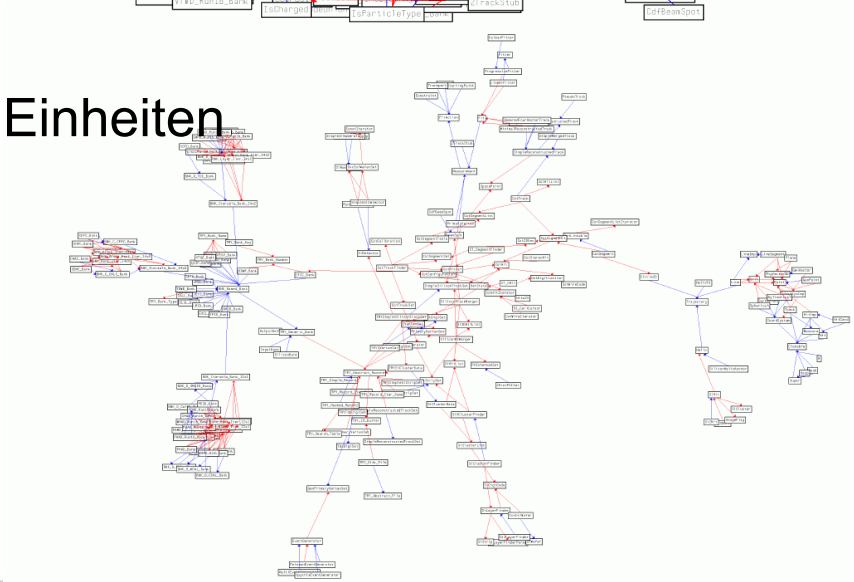


- Zeitdruck während der Entwicklung
- Know-How-Defizite der Softwareentwickler

Gutes Design: Grundsätze

- Abstraktion:
  Schaffen einer vereinfachten Sicht auf Konzepte der Anwendungsdomäne
  (→Klassenbildung)
- Kapselung:
  Trennen von Schnittstelle und Implementierung
- Modularisierung:
  Zerlegen der Komplexität in handhabbare Einheiten
  (→Subsystembildung)

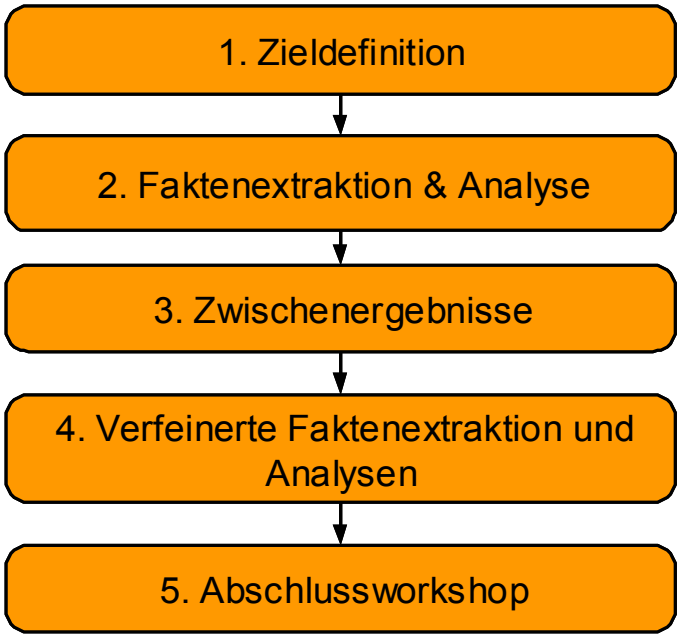




- Vernünftige Komplexität
- Geringe Kopplung, hohe Kohäsion

Dienstleistung Softwareassessments

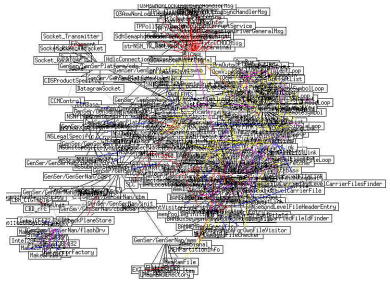
- Ziel: Bewertung der Qualität von Software, Identifikation von Schwachstellen
- Durchführung:
  - Zwei Experten des FZI untersuchen mit Werkzeugunterstützung Systeme in Java, C oder C++ in 5 Arbeitstagen (peer work!)
  - Vorort beim Kunden: Know-How-Austausch, Vertraulichkeit
  - Festpreis
- Ergebnisse:
  - Workshop zur Diskussion der Analyseergebnisse
  - Optional: Ausführlicher Qualitätsbericht + Mängelliste



Beispiele

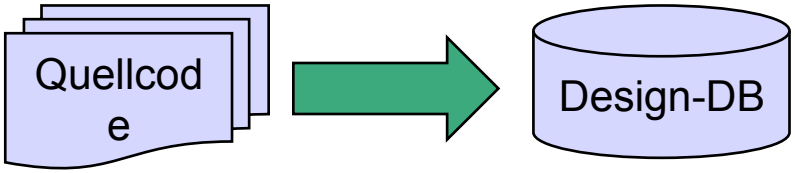
- Zahlreiche Softwareassessments:
  ABB, DaimlerChrysler, Debis, Deutsche Telekom, IBM, Nokia, SOLID Technologies, VTT und viele weitere
- Systeme aus Forschungseinrichtugnen
  ET++, 65 kLOC, C++, 770 classes
  VTT eXpert web application, 10 kLOC, 16 classes, Java, JSP
- Engineering Software
  150 kLOC, C++/DCOM
  2 MLOC C++/DCOM
- Software aus Telekommunikationssystemen
  500 kLOC, C++, C, Assembler
  2 MLOC, C,C++
  1 MLOC C++, 120 kLOC Java, CORBA
  20 MLOC, ~120 kLOC, Chill
  2 MLOC Java
- Vertragsmanagementsystem für Versicherungen
  1MLOC, Java/EJB, 6000 classes
- Datenbanksystem für eingebettete Systeme
  1MLOC, C, 28 subsystems, ~300 complex data types, 14.000 functions

Werkzeugunterstützung

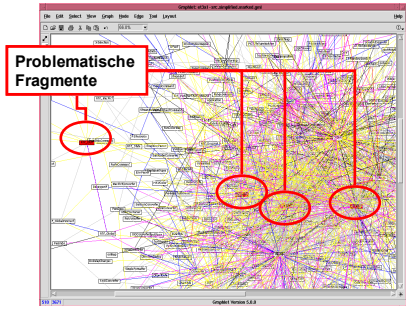


Visualisierung

Abstraktion

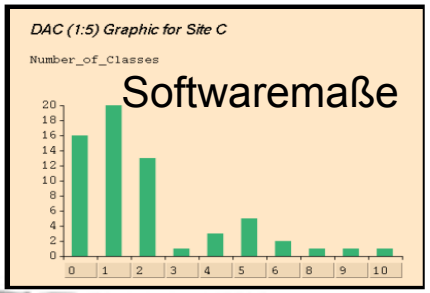


Faktenextraktion



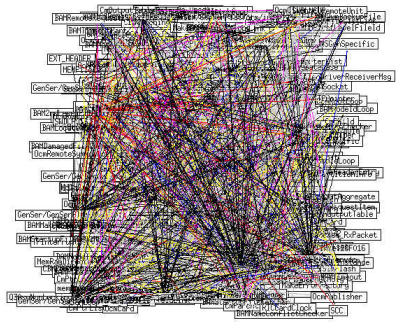
Problemidentifiktion;
Prüfung von Heuristiken

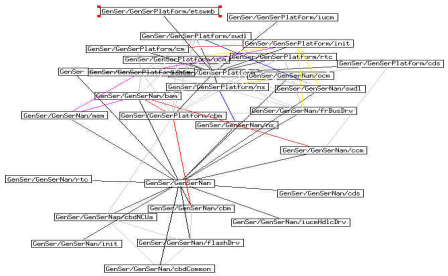
GOOSE + Echidna



Vom Detail auf ‘s Ganze...

- Idee:
  Aus Detailinformationen eine Übersicht über das Gesamtsystem gewinnen
  - Faktenextraktion
  - Selektion
  - Aggregation

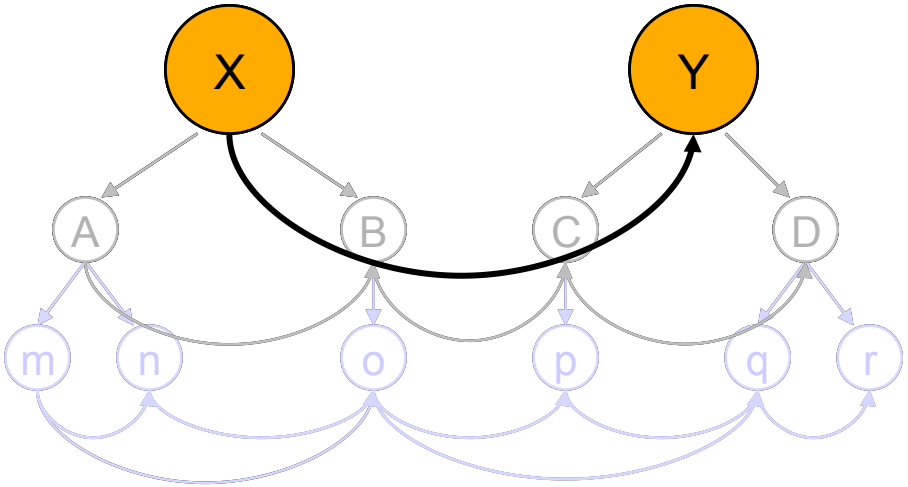




Subsysteme

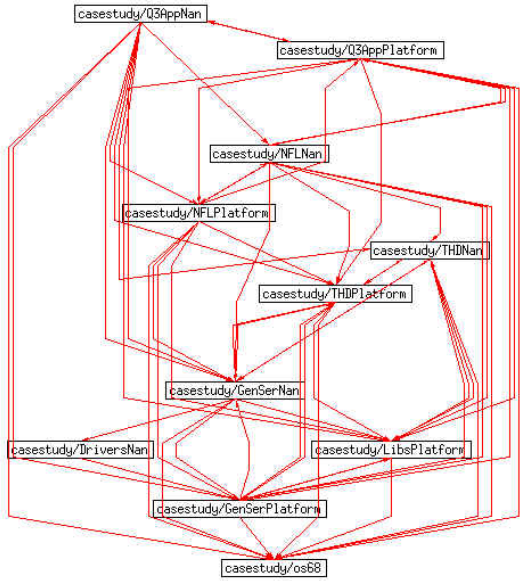
Klassen

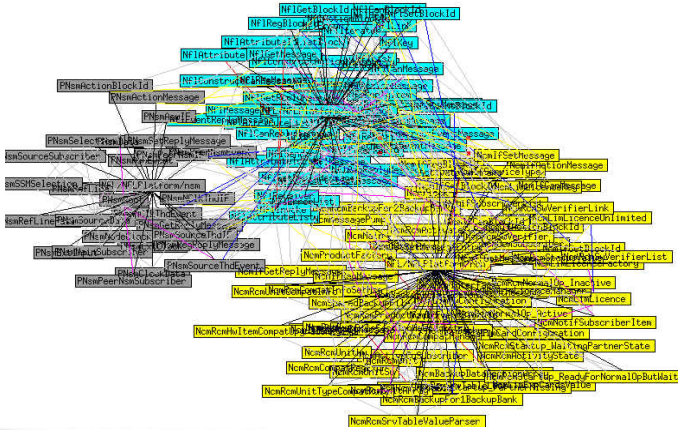
Funktionen/Methoden



Visualisierung

- Idee:
  Visuelle Fähigkeiten des Menschen nutzen, um komplexe Systeme zu verstehen
- Anwendungen:
  - Systemzusammenhänge verstehen
  - Prüfen von Abhängigkeiten (Schichtung, Framework vs. Anwendungsimplementierung)
  - Überprüfen der Subsystembildung
  - Identifikation von *god classes*



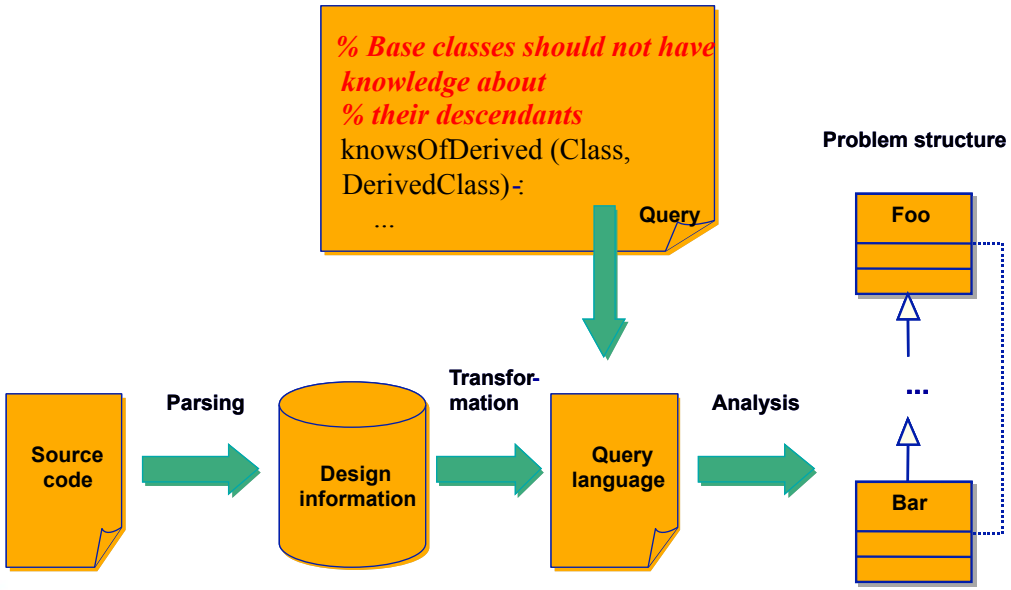


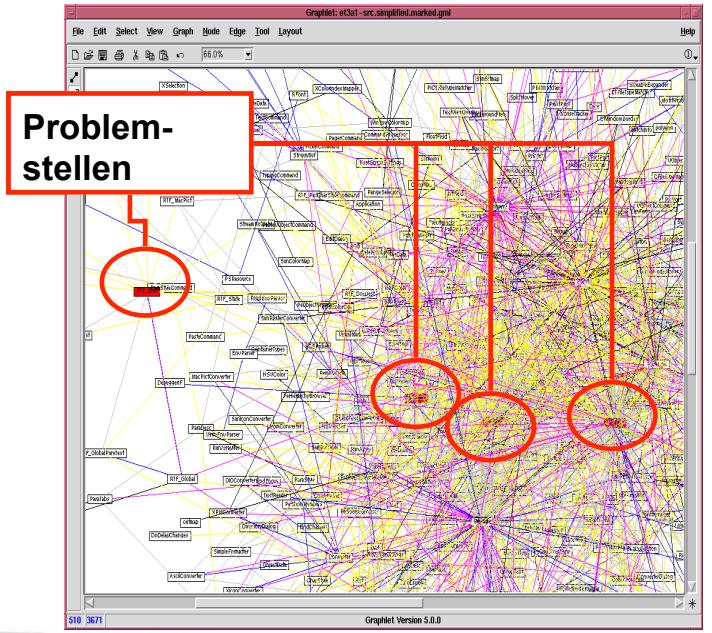
Schwachstellensuche

- Architekturuntersuchung
  - Prüfen von Architekturregeln
- Metriken
  - Kopplung, Kapselung und Komplexität von Subsystemen und Klassen
  - Komplexität und Aufrufabhängigkeiten zwischen Methoden
- Musterbasierte Schwachstellensuche
  - Auffinden von Bad Smells, z.B.: Oberklassen mit Kenntnis ihrer Unterklassen
  - Überwachung (struktureller) Programmierrichtlinien
- Analyse von Codeduplikation
- Kundenspezifische Analysen

Technik zur Schwachstellensuche

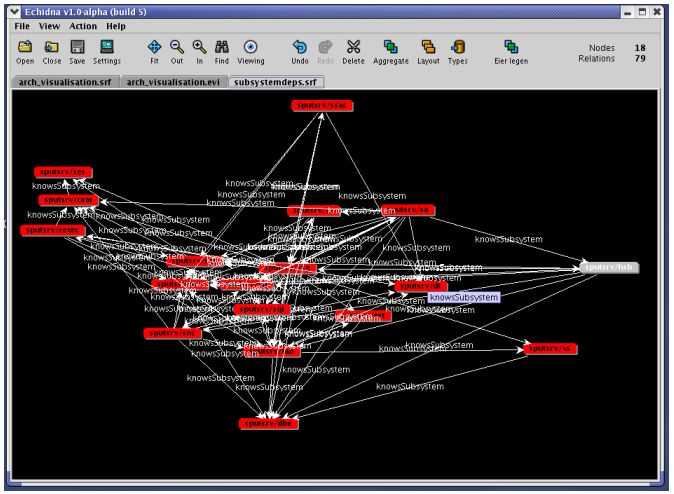
- Faktenextraktion: Struktur-“Datenbank“ des Systems
- Abfragen:
  - Musterbasiertes Suchen
  - Messen: Metriken



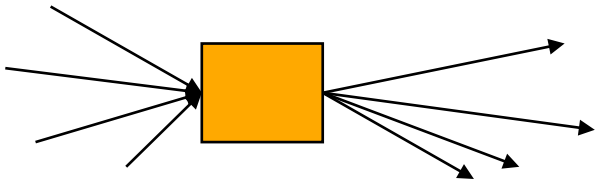


Architekturuntersuchung

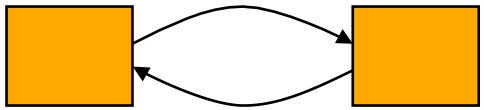
- Visuelle Überprüfung der Architektur



- Überprüfung von Architekturregeln
  - Definition und Überprüfung von Regeln bezüglich Abhängigkeiten zwischen Subsystemen
  - Beispiel: shouldNotDependOn('subsystemA',X) :-
    not(isUtilityLayer(X)).
- Suche nach Abhängigkeitszyklen und Flaschenhälsen zwischen Subsystemen und Datentypen



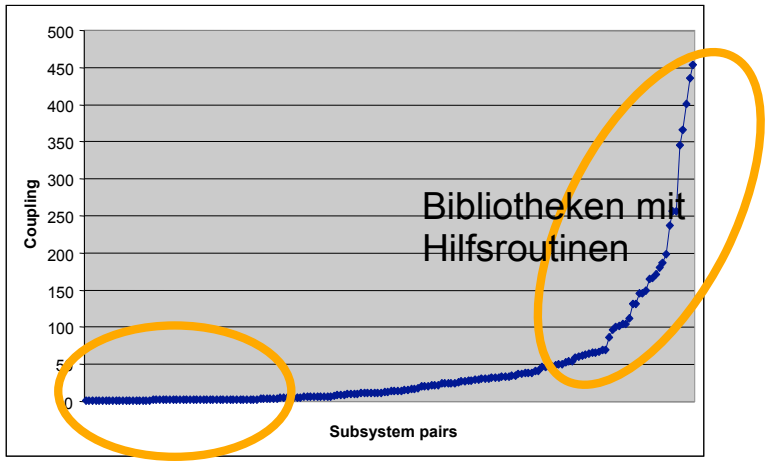
Flaschenhals



Zyklus

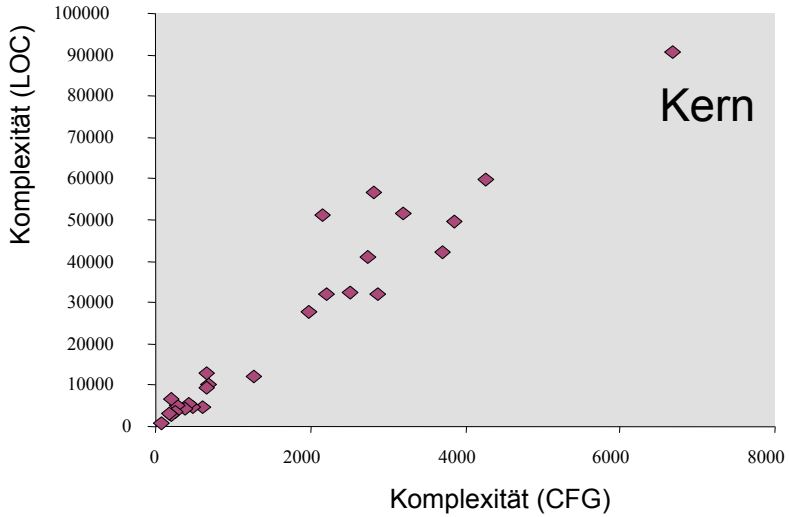
Subsysteme: Kopplung und Komplexität

Kopplung



Kopplungswerte
inklusive Kern

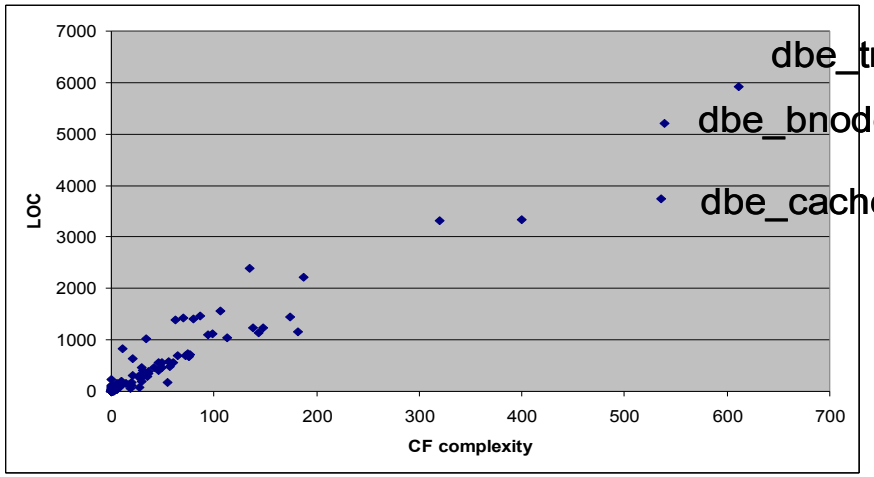
Komplexität

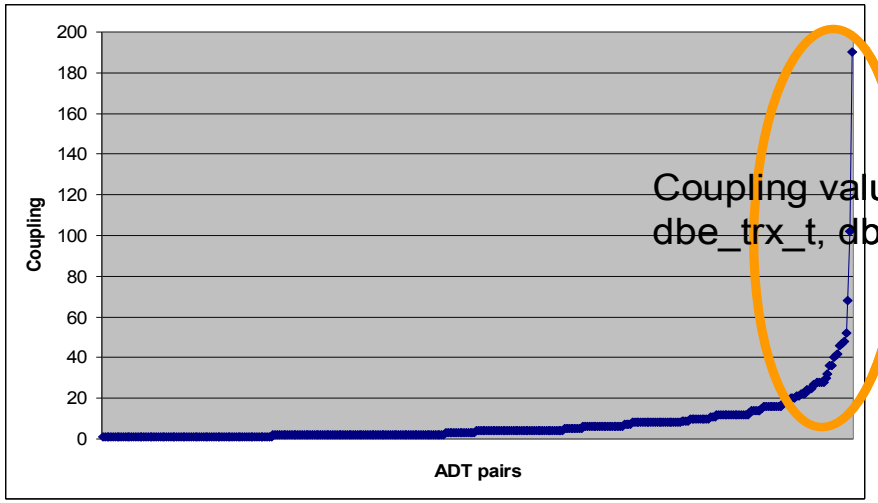


- Gut: Meist geringe Kopplung zwischen Subsystemen, meist vernünftige Komplexität
- Einige Subsysteme (Kern) sind komplex aber gut gekapselt

Mikrostrukturen: Komplexität und Kopplung

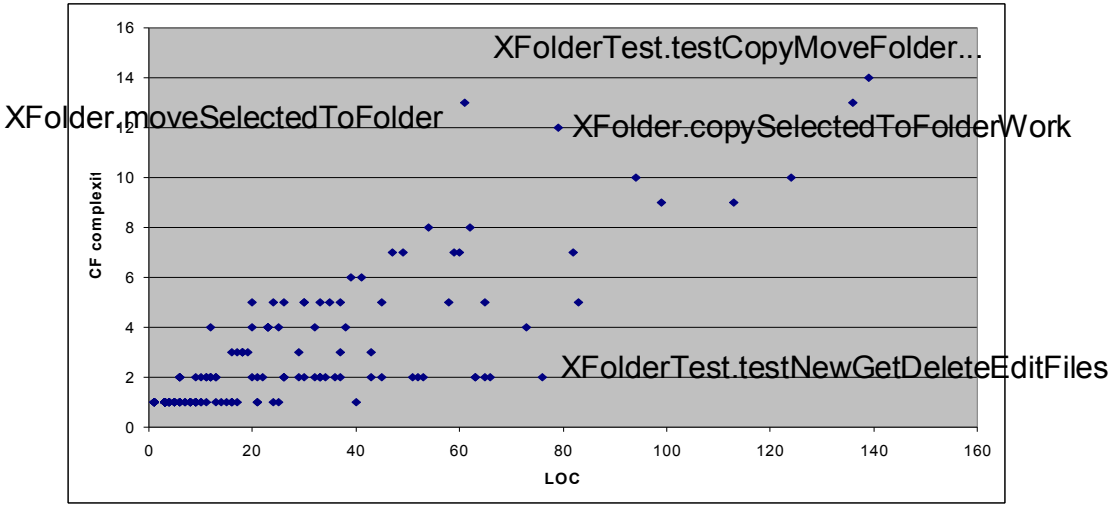
- Komplexitätsanalysen
  Gut: nur wenige Datentypen sind sehr komplex
- Kopplung
  Gut: nur wenige Paare von Datentypen mit hohen Kopplungswerten
- Probleme:
  - Einige Datentypen komplex und hoch mit anderen gekoppelt
  - Repräsentieren zentrale Konzepte; wahrscheinlich kann dies nicht verhindert werden





Untersuchung von Funktionen

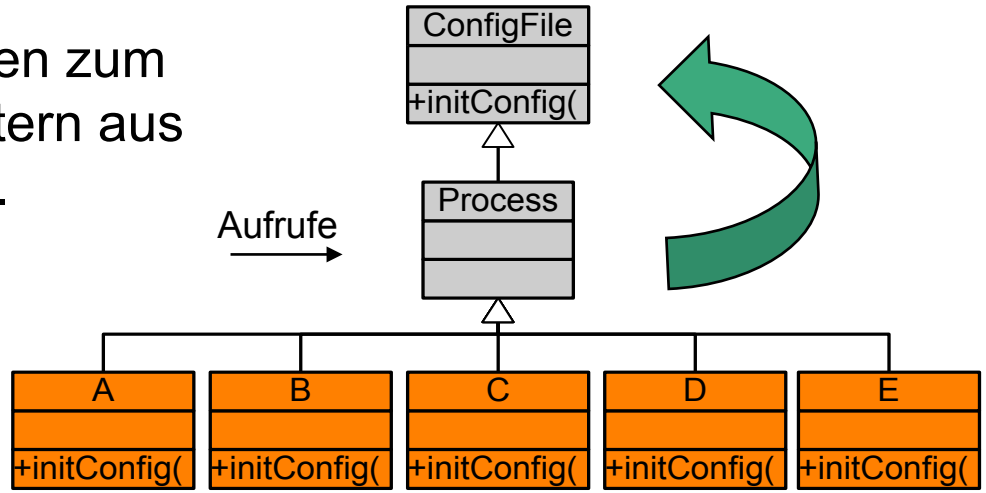
- Beispiel: Relative Komplexität von Methoden



- Rekursionsketten und Anzahl der Methodenparameter:
  - Rekursionskette: Schleife im statischen ermittelten Aufrufgraph
  - Lange Ketten: wahrscheinlich nicht absichtlich erzeugt; Fehlerfälle oder schwieriger Code (z.B. Mozilla: 117)
  - Anzahl der Parameter: Benutzung von Methoden mit vielen Parametern schwierig! (Rekord: 67)

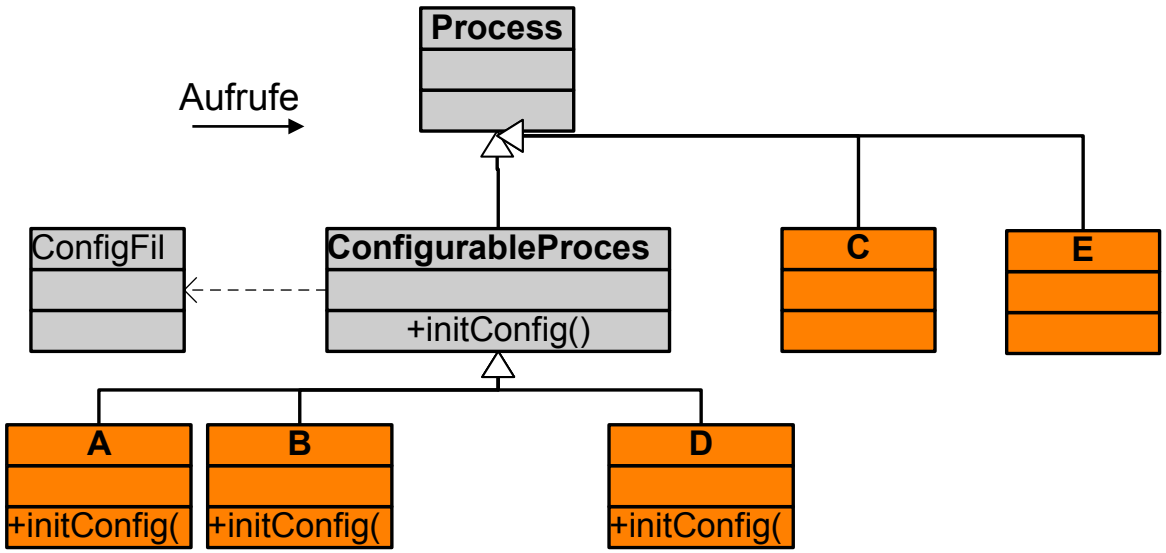
Mikrostrukturproblem

- *ConfigFile* definiert Operationen zum Einlesen von Prozessparametern aus einer Konfigurationsdatei, z.B. *initConfig()*



- Spezielle Prozesse *A* bis *E* überschreiben *initConfig():*
  - *C* und *E*‘s Implementierung von *initConfig()* ist leer
  - *A, B* und *E*‘s Implementierungen von *initConfig()* orientieren sich an der von *ConfigFile* (Codeduplikation!); zudem sind sie recht komplex.
- Viele Aufrufstellen verwenden die Schnittstelle von Process;
- *ConfigFile* wird nie verwendet!
- (*initConfig()* enthält case-Statements mit Typabfragen nach *A, B* und *E*)

Mikrostrukturproblem -- Lösung



- Es gibt jetzt explizit konfigurierbare und nicht konfigurierbare Prozesse
- Vererbung jetzt semantisch OK: Spezialisierung
- Gemeinsame Funktionalität von *initConfig()* in *A, B* und *D*:
  - Rumpfimplementierung in *ConfigurableProcess*
  - Hook-Methoden zur spezifischen Anpassung (Template Method Pattern) in *A, B, D*
- Einlesen der Konfigurationsdatei kann an *ConfigFile* delegiert werden

Hindernisse

- Skepsis der Entwickler
  - „Hilfe, meine Leistung wird begutachtet“
- Defizite auf Werkzeugebene
  - Faktenextraktion und industrieller Code (unvollständig, Makros, Sprachdialekte)
  - große Menge an Rohdaten
  - Fehlende Abstraktionsmechanismen
- Verfahren zur automatische Identifikation liefern nur potentielle Schwachstellen
  - Diskussion mit Entwicklern nötig
  - Fehlendes Domänenwissen
  - Zeitmangel

Nutzen

- Regelmäßige Assessments erleichtern das Erhalten der Produktqualität
  - Definition und Durchsetzung von allgemeingültigen, technologie- und unternehmensspezifischen Qualitätsstandards
- Stärkung der Intuition der Entwickler
  - Objektive Messwerte
- Anregung zur Diskussion
  - Welche Teile sind verbesserungsbedürftig?
  - Entstehen eines neuen Qualitätsbewusstseins
- Neue Perspektiven durch externe Analysten
  - In Frage stellen existierender Lösungen
- Standortbestimmung: Vergleich mit anderen Systemen
  - Vorsicht: Anwendungsdomäne, Programmierstil beeinflussen Messwerte

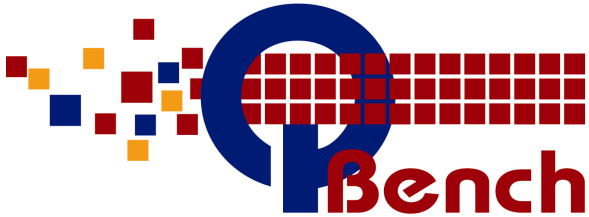
Unser persönlicher Rat...

- Eine gute Struktur ist der Schlüssel zum Erfolg eines Systems!
- KISS: Keep it simple, stupid! (A. Tanenbaum)
  Wenn etwas zu kompliziert erscheint ➔ Vereinfachen!
- Zerlege Probleme in Teilprobleme!
  Miller‘s Law: Eine gute Struktur sollte es erlauben, dass man nie mehr als 7 (+/-2) Dinge im Kopf haben muss.
- Benenne Konzepte vernünftig!
  Wenn man etwas nicht benennen kann, hat man es noch nicht verstanden. ➔ Überdenken!

Kontakt

Markus Bauer, <bauer@fzi.de>
Forschungsbereich PROST
FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
76131 Karlsruhe
http://www.fzi.de
Tel.: +49 721 9654 630
Fax: +49 721 9654 609

http://compobench.fzi.de



http://www.qbench.de