XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 81-85/8.Stock
A-1200 Wien
Tel: 0664-8242-600
E-mail: office@xion.at
Web: xion.at
Festnetz: +43/1/333 91 99-0
Fax: +43/1/333 91 99-199

x i o n . it systems ag

Software Wartung und Evolution

Dipl.-Ing. Dr. techn. Johannes Weidl-Rektenwald
Xion IT Systems AG

XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 81-85/8.Stock
A-1200 Wien
Tel: 0664-8242-600
E-mail: office@xion.at
Web: xion.at
Festnetz: +43/1/333 91 99-0
Fax: +43/1/333 91 99-199

x i o n . it systems ag

Lecture 2

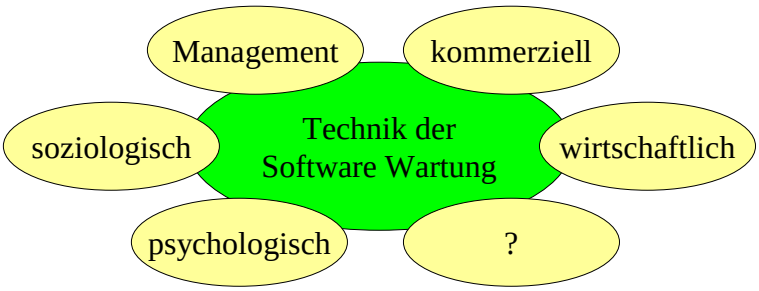
Lecture 2

- Inhalte
  - Aspekte der Software Wartung
  - Aktivitäten der Software Wartung
  - Software Wartungskrise
  - Legacy Systeme
  - Reverse Engineering
    - Redocumentation
    - Design Recovery

Aspekte der Software Wartung

... und das war´s?

Aspekte der Software Wartung



- Berücksichtigung aller Aspekte führt zur *holistischen* Betrachtung von Software Wartung

Aspekte der Software Wartung

- Technical
  - Understanding existing code
- Managerial
  - Reactive work context
- Economic
  - Justifying remedial work
- Commercial
  - How to estimate the effort of a change request?

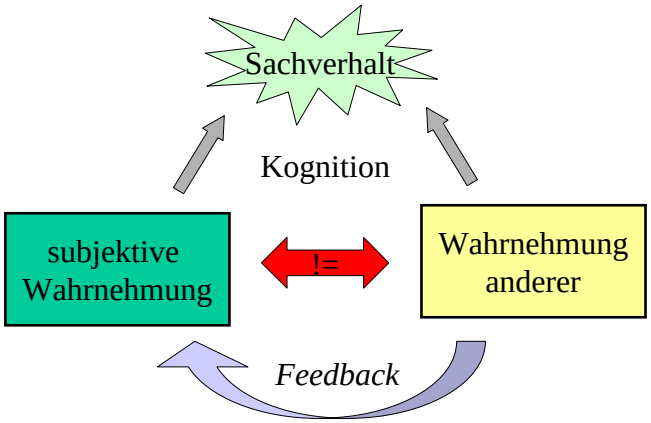
Aspekte der Software Wartung

- Sociological
  - Second hand / apprentice‘s work
  - Maintaining morale
- Psychological
  - Hesitate to touch a huge work of art
  - Fear to damage a working system constantly
  - „Kognitive Dissonanz“

Kognitive Dissonanz

- **Definition (nach** L. FESTINGER)
  - K. D. ist die Bezeichnung für einen unangenehm erlebten Zustand, der aus widersprüchlichen Erfahrungen oder Einstellungen in Bezug auf den gleichen Gegenstand hervorgeht und nach einer Veränderung verlangt.
  - Um den als unangenehm erlebten Zustand aufzulösen, müssen neue, spannungslösende Schritte eingeleitet werden. Dies lässt sich auf mehrere Arten erreichen, z.B. durch die Suche nach neuen **Informationen**, durch eine Einstellungsänderung, durch ein Handlungsänderung usw.
  - Die Theorie der k. D. beschäftigt sich mit der Verarbeitung von wichtigen Informationen, nachdem eine Entscheidung gefällt wurde.
    - Die Theorie besagt, dass nach einer getroffenen Entscheidung vorzugsweise die Informationen ausgewählt werden, die die Entscheidung als richtig erscheinen lassen, und dass gegenteilige Informationen nicht mehr beachtet werden.

Kognitive Dissonanz



Kognitive Dissonanz: Beispiel

- Sachverhalt: **Die Software funktioniert nicht so, wie sie sollte**
- Wahrnehmung Programmierer: „Ich bin mit dem Programm fertig. Es funktioniert."
- Wahrnehmung Quality Assurance: „Dieser Testfall funktioniert nicht."
- Feedback QA: „In ihrem Programm ist ein Bug"
- -> kognitive Dissonanz: „Ich bin fertig und das (weniger das Ergebnis, sondern der Zustand) gefällt mir" **versus** „Du hast Mist gebaut!"
- Auflösung der KD:
  - „Der Bug liegt sicher nicht bei mir! Fragen sie mal Kollegen X."
  - „Es steht aber genau so in den Anforderungen! Lesen sie die mal!"
  - „Das ist kein Bug, die User haben es mir so erklärt."
  - „Sie wissen ja nicht einmal, was ein Bug ist!"
- Kognitive Dissonanz führt also zu einer ***Wirklichkeitskonstruktion***, die versucht, die Dissonanz aufzulösen

Egoless Programming

- Über-Identifikation mit den Artefakten der Arbeit ist ein unerwünschter Nebeneffekt
- -> „Egoless Programming“
- Modern: Extreme Programming mit dem Leitfaden „Embrace Change“ [Kent Beck]
  - Ist psychologisches Problem, nicht rein technisches!

Beispiel für den psycho-sozialen Aspekt

- aus Petr Kroha „Softwaretechnologie“ [Kroha97]
- Windows of Opportunity [Yourdon92] - beschreiben günstigsten Zeitpunkt zur Neuentwicklung eines Softwaremoduls:
  - „der zuständige Programmierer geht in Rente oder kündigt
  - wenn die Komponente extrem gravierende Fehler aufweist und der Programmierer die Suche aufgegeben hat
  - *wenn es endlich möglich ist, den Widerstand leistenden Programmierer zu entlassen.*“

Kommerzieller Aspekt

Budget, Time-to-market

Software Qualität

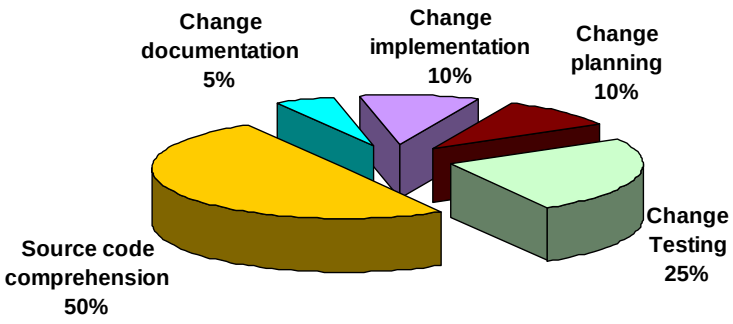
Aktivitäten der Software Wartung

Aktivitäten der Software Wartung

- Analyse bzw. Planung der Änderung
  - Verstehen des Systems, der „Architektur“
  - Source Code Verstehen, Bilden von Hypothesen
  - Verifizieren von Hypothesen
  - Change Impact Analysis
- Implementierung der Änderung
  - Restructuring
  - Change Propagation
- Verifikation und Validierung
- Re-Dokumentation

Management

Activities in Software Maintenance



Activities in Software Maintenance

Source: Principles of Software Engineering and Design, Zelkovits, Shaw, Gannon 1979

Software Wartungskrise

Software Wartungskrise

- Unter der „Software Wartungskrise“ versteht man die Unangemessenheit der Prozesse und Werkzeuge, um den Vorgang der Software Wartung auch für komplexe, umfangreiche Systeme qualitativ hochwertig (ökonomisch und technisch) durchführen zu können.

Legacy Systeme

Eigenschaften von Legacy Systemen

- Größe: > 1 Mio. LOC
- Alter: > 10 Jahre
  - Verwendung veralteter Programmiersprache wie COBOL, FORTRAN, PL/I, ...
  - Verwendung veralteter Datenspeicherung, z.B.
    - Flat Files
    - Hierarchische Datenbanken
  - Dokumentation veraltet
- Strategische Bedeutung
  - Bilden kritische Geschäftsprozesse ab – Unternehmen ohne System nicht vorstellbar
  - 7x24 Verfügbarkeit

[Kroha97]

Eigenschaften von Legacy Systemen

- Kosten
  - Wartungskosten bei Legacy Systemen typischerweise über 95% der Gesamtkosten
- Systemumgebung
  - Limitierte Hardwareressourcen
  - Begrenzte Anbindungsmöglichkeiten an Kommunikationsprotokolle (Middleware, z.B. CORBA)
- Komplexität
  - Neue Anforderungen können im System nicht mehr verwirklicht werden
  - Unzufriedenstellende Performanz (z.B. Transaktionsrate)

[Kroha97]

Probleme bei der Ablöse von Legacy Systemen

- Das neue System muss funktional äquivalent zum alten System sein
- Das neue System muss zusätzlich alle aktuellen Anforderungen implementieren
- Die Daten der Legacy Applikation müssen übernommen werden
- Die neue Applikation soll State-of-the-art Techniken verwenden (Datenbank, 3-Tier, Objektorientierung, Middleware, ...)
- Die Ausfallszeit durch die Ablöse soll minimal sein

[Kroha97]

Gründe für das Scheitern von Legacy Neuimplementierungen

- Um die Mittel bewilligt zu bekommen, müssen umfangreiche Erweiterungen direkt mit der Migration versprochen werden
- Neuentwicklung dauert lange – in der Zwischenzeit ändern sich die Anforderungen
- Es existieren versteckte Abhängigkeiten von vielen Programmen zur Legacy Applikation
- Politische Einflüsse verhindern eine Fertigstellung
- Management von Software Großprojekten ist im Allgemeinen schwierig

[Kroha97]

Migration von Legacy Systemen

- Meist inkrementell auf per-Modul Basis
  - Reverse Engineering
  - Restrukturierung und Reuse
  - Entwicklung neuer Komponenten
  - Integration
  - Datenmigration
  - Inbetriebnahme (Installation, Schulung, ...)

[Kroha97]

Beispiel: Technologien Erste Bank

- 1990 waren bei der Ersten Bank Systeme in folgenden Technologien im Einsatz
  - Assembler
  - PL/I
  - Cobol
  - Fortran-TRX
  - TRX-GEN 1/2
  - „Entscheidungstabellen, DELTA, TIMESHARING, DMS1100“
    - » Aus: Spezielle Aspekte der Informationsverarbeitung in der Wirtschaft, W. Konvicka 1995

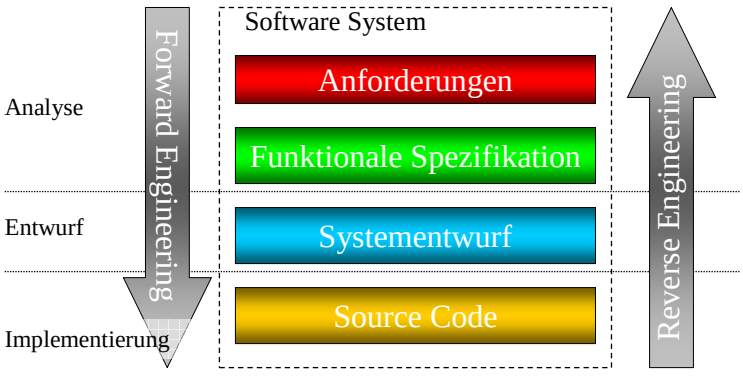
Reverse Engineering

Reverse Engineering: Definition

- Unter Reverse Engineering versteht man den Prozess der Analyse eines bestehenden Systems, mit dem Zweck
  - der Identifikation von Systemkomponenten und deren Beziehungen untereinander, sowie
  - der Erzeugung von Darstellungen des untersuchten Systems auf unterschiedlichen, höheren Abstraktionsstufen.

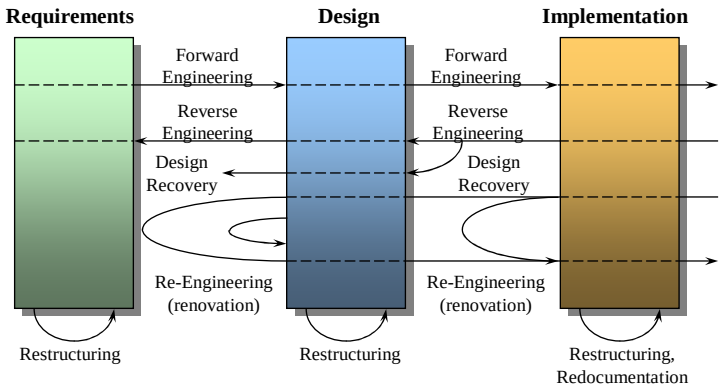
[Chikofsky/Cross]

Forward- und Reverse Engineering



[Klösch/Gall95]

Reverse Engineering Terminologie



[Chikofsky/Cross90]

Motivation

- Ca. 75% of software development time and expense goes toward maintenance
- Of this, ca. 50% goes to understanding the software and the bug/enhancement
- Consequently, we want to devise tools and techniques to support improved understanding of software

Difficulties

- Dimensions
  - application domain (what) and program (how)
- Information
  - Specific (machine-level) and abstract (design-level)
- Formality
  - Formal structure of programs and informal human understanding

Reverse Engineering verlangt Wissen

- Programmiersprache
  - Syntax
  - Semantik
- Programmierung (Algorithmen, Datenstrukturen, Patterns, ...)
- Application Domain („Domänenwissen“)

Rerverse Engineering: Subprozesse

- Redocumentation
- Design Recovery

Redocumentation

- Erzeugung oder Überarbeitung von semantisch äquivalenten Repräsentationen des Systems innerhalb desselben Abstraktionsniveaus
  - Datenfluss- und Kontrollflussdiagrammen, Cross Reference Listings, etc.
  - Automatisch generiert, meist ohne zusätzliche Informationen

Design Recovery

- Ist ein Prozess, in dem zusätzliche Information verwendet wird, um Abstraktionen des Systems zu generieren u. zw. auf höheren Abstraktionsstufen
  - Wissen über Anwendungsdomäne, informale Beziehungen
  - Wissen von Applikationsexperten über Architektur, Modulstruktur, etc.

Design Recovery: Repräsentationen

- Repräsentationen durch Design Recovery
  - Structure Charts
  - Nested Trees
  - Datenflussdiagramme
  - Kontrollflussdiagramme
  - Entity Relationship, ...

  Angereichert durch informale Information

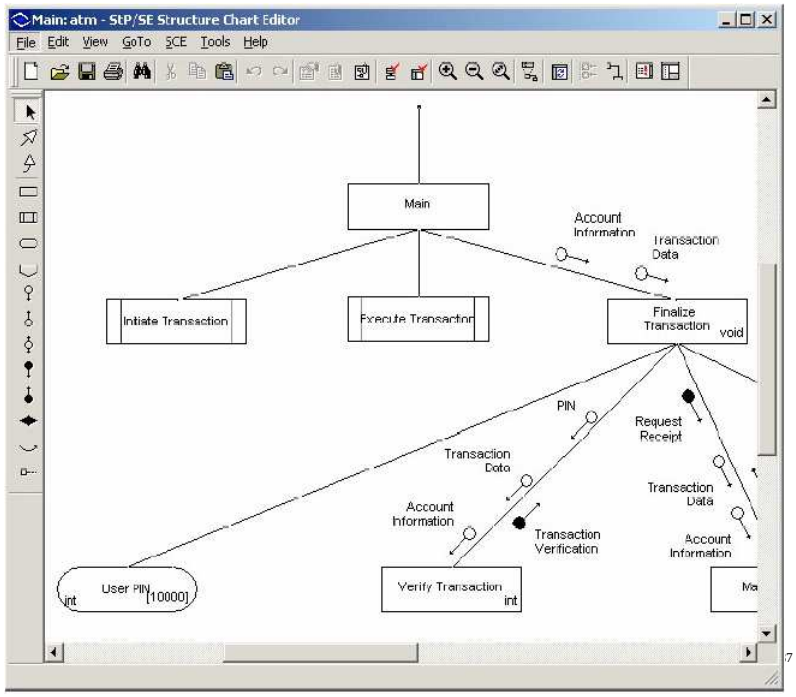
  - Informale Beschreibungen des Software Systems
    - Text
    - Diagramme

Arten von Design Recovery

- Modellbasiert
  - Desire von Biggerstaff [Biggerstaff89]
- Wissensbasiert
  - Basierend auf zentraler Wissensbasis (Repository)
  - Extrahierung von Programmclichés
    - z.B. Sortieralgorithmen, Listen, Bäume
- Formale Methoden
  - Transformation von v.a. COBOL in Z oder Z++
  - siehe ESPRIT Projekt „REDO“
- Objektorientiert
  - Objekt Identifizierung

Structure Charts

- Aus dem Strukturierten Design [Coad/Yourdon79]
- Stellen die Modulstruktur nach der funktionalen Dekomposition [Parnas72] in einer hierarchischen Form dar
- Beinhalten Information, welche Daten zwischen den Modulen ausgetauscht werden
- Black box Ansicht von Modulen, Verhalten wird durch Input/Output beschrieben
- In UML: Klassen + Event Trace Diagramme

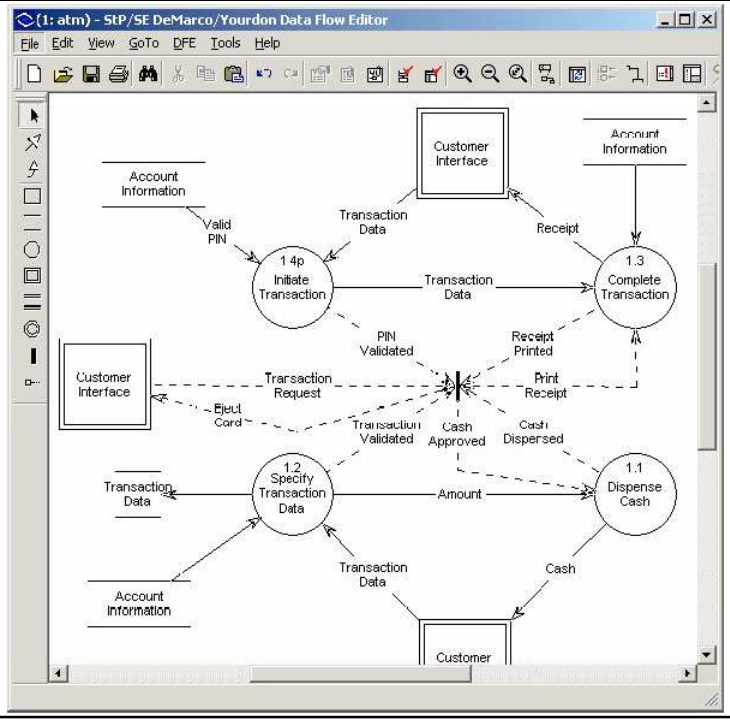


Nested Trees

- Stellen den impliziten Datenaustausch über globale Variablen dar
- Knoten
  - Repräsentieren Module
- Kanten
  - Repräsentieren die Definition eines Moduls in einem anderen
- Nested Tree ermittelt
  - Verschachtelung der Deklarationen von Prozeduren und Funktionen
  - Kann Gültigkeitsbereich für jede Variable im System darstellen

Datenflussdiagramme

- Stellen Datenfluss und Datentransformation dar
  - Datentransformation: Ersetzen der formalen Parameter von Prozeduren durch die aktuellen Parameter
- Verschiedene Ansätze zur „bottom-up“ (i.e. reverse) Generierung finden sich in der Literatur
  - z.B. Benedusi, Cimitile und De Carlini [BCD89]

- 

Kontrollflussdiagramme

- Visualisierung des Kontrollflusses
  - Zwischen Prozeduren
    - Call tree
  - Innerhalb einer Prozedur
    - „Logische Wege“ durch eine Prozedur werden visualisiert
    - Visualisieren die zyklomatische Komplexität (McCabe Metrik)

Design Decisions

- During design and implementation decisions are made according to specific design rationales
  - Formal representation: Design Decision Tree (DDT)
- Tools for making design decisions persistent during the development process are only in experimental stage
  - Therefore, most of the design decisions almost always have to be extracted when examining existing code
- Reverse engineering design decisions deals with automated decision extraction and injection, knowledge repositories, knowledge management

Ziele von Reverse Engineering

- Beherrschung der System Komplexität
- Erzeugen von fehlender oder alternativer Dokumentation
- Wiedergewinnung verlorener Information
- Erkennung von Seiteneffekten und Anomalien
- Migration auf eine andere Hardware/Software Plattform bzw. Integration in eine CASE Umgebung
- Erleichterung der Software-Wiederverwendung

[Klösch/Gall95]

Reverse Engineering Kandidaten

- Schlecht strukturierter Source Code
- Umfassende korrektive Wartung
- Veraltete Dokumentation
- Design Infos fehlen oder sind unvollständig
- Module sind unüberschaubar komplex
- Migration auf eine neue Plattform
- System soll durch ein neues abgelöst werden

[Klösch/Gall95]

Reverse Engineering: Vorteile

- Kosteneinsparung in der Software Wartung
- Ermöglichen weiterer Software Evolution
- Qualitätsverbesserung
- Wiederverwendung von Software Komponenten
- Vorteile im Wettbewerb
- Investitionssicherung

Reverse Engineering: Probleme

- Umfangreiches Source Code Volumen
- Mangelndes Wissen über das Programm
- Inkonsistente Entwicklungsstandards
- Nicht aktuelle oder fehlende Dokumentation
- Hohe Redundanz

Reverse Engineering bzw. Code Comprehension Tools

- Imagix4D (Imagix Corp.)
  - www.imagix.com
- Software Refinery, Refine/C (Reasoning Systems Inc.)
- Sniff+ (Wind River)
  - http://www.windriver.com/products/development_tools/ide/sniff_plus/
- Source Navigator
  - http://sourcenav.sourceforge.net
- Rational Rose
  - http://www.ibm.com/software/rational
- Software through Pictures SE (Aonix: StP/SE)
  - http://www.aonix.com/stp_se.html
- Sotograph
  - www.software-tomography.com

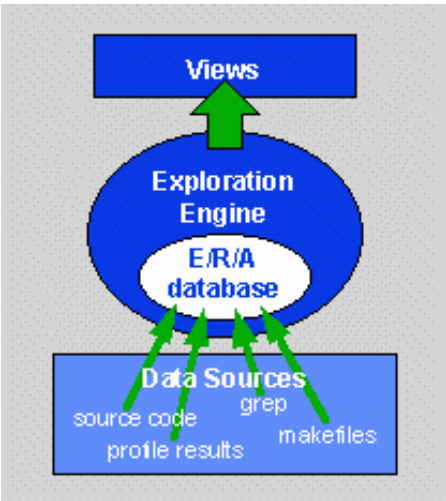
Die Tool Falle

- *"A fool with a tool is still a fool"*
  - English proverb
- Give software tools to good engineers. You want bad engineers produce less, not more, poor-quality software.
  - [Davis95]: Principles of Software Development

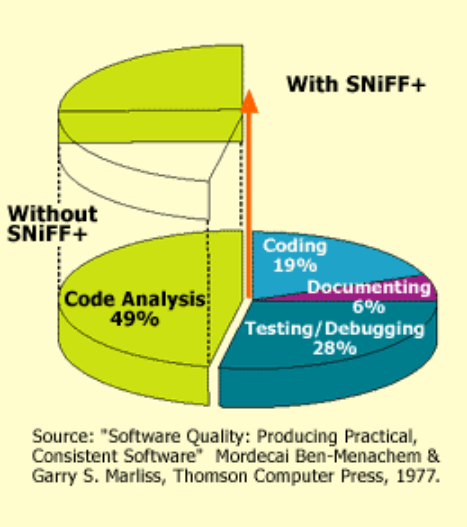
Imagix 4D: Funktionen

- Only for C/C++
- Provides graphical 3D output of analysed source
- Control Flow Analysis
  - Generation of augmented call graphs
- Annotated Flow Charts
  - Shows the logic flow for complex constructs
- Use Browser
  - Shows where and how a symbol is used
- Automated generation of HTML documentation

Imagix 4D: Toolkit



Sniff+



Source: "Software Quality: Producing Practical, Consistent Software" Mordecai Ben-Menachem & Garry S. Marliss, Thomson Computer Press, 1977.

101

Sniff+: Funktionen

- Source code editor
- Class browser
- Inheritance hierarchy browser
- Xref – Cross reference browser (uses/used)
- Include browser (Java: imports)
- Retriever – search for symbol names
- Grep
- Interface to debugger, build environment, revision control

102

Software Refinery

- Programming-language specific parsers (Fortran, C, Cobol, Ada)
- Standard analyzers (xref, def/use, call tree)
- OO Repository for holding results of analyses
- Wide-spectrum language for writing custom analyses
- Extensions: dialect (parser), intervista (GUI)