



SE 4352

Software Architecture and Design

Fall 2018

Module 10



Announcements

- Assignment 6 (last homework) on eLearning, deadline 11/06/18

Recall

- Software Security
- Threat Modeling
- CIA
- Countermeasures

Remainder of Semester

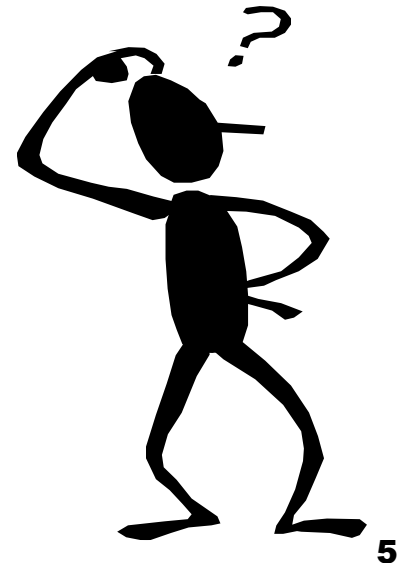
- Exam 2 at testing center
 - November 15
- Exam 2 review in class
 - November 13
- Fall break
 - November 19 - 23
- Team Project report due
 - November 27
- Team Project presentations in class
 - November 29, December 4,
December 6



When Does an Architect Architect?

- ?

- You have an unexpected case:
 - You finished one course project using Java
 - Your program runs OK
 - But, by accident, you delete the java file
 - How to hand in your project?



What is Reverse Engineering ?

- You have an unexpected case:
 - You finished one course project using Java
 - Your program runs OK
 - But, by accident, you delete the java file
 - How to hand in your project?
- Reverse Engineering



Reverse Engineering

- “Reverse engineering” term derived from hardware development
 - the process of discovering of how competitor’s system worked.
 - in software engineering,
 - it is the process of discovering how your own system works.
- Software systems become difficult to understand and maintain since over time their size and complexity has had a continuous evolution.

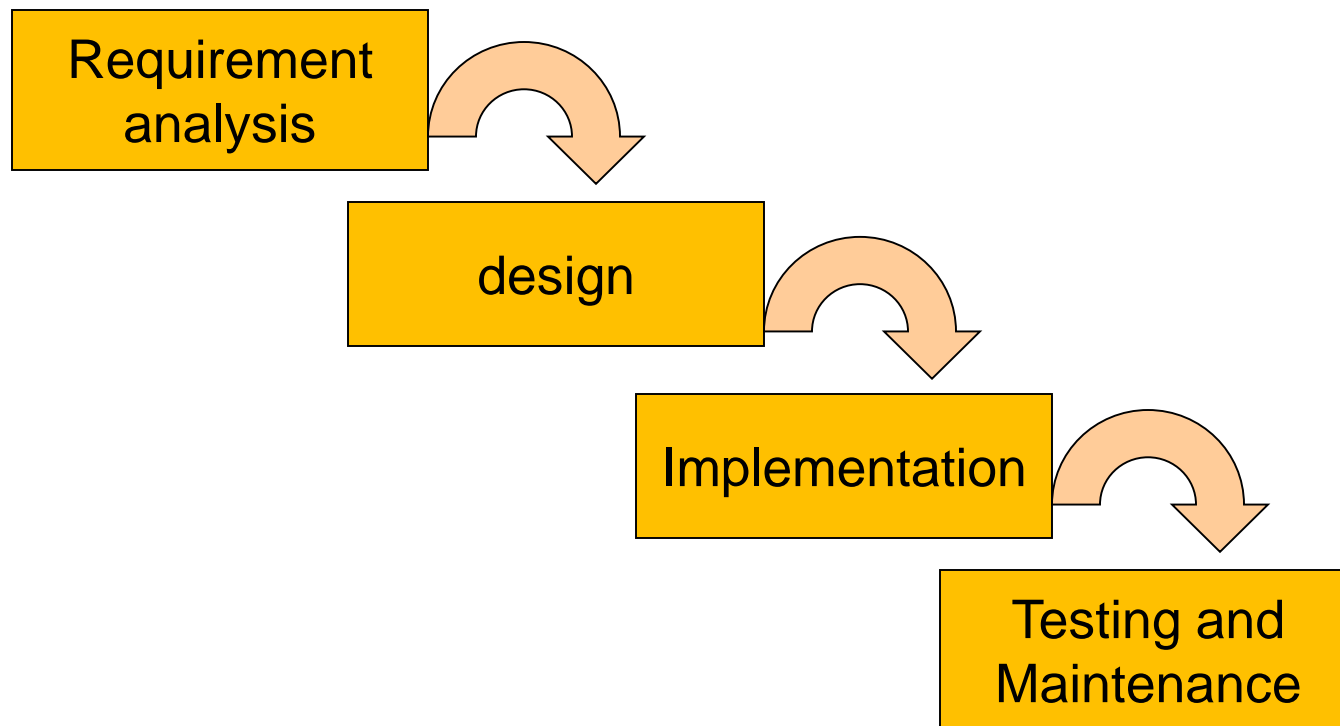


Reverse engineering

- Analysing software with a view to understanding its design and specification
- May be part of a re-engineering process but may also be used to re-specify a system for re-implementation
- Builds a program data base and generates information from this
- Program understanding tools (browsers, cross-reference generators, etc.) may be used in this process

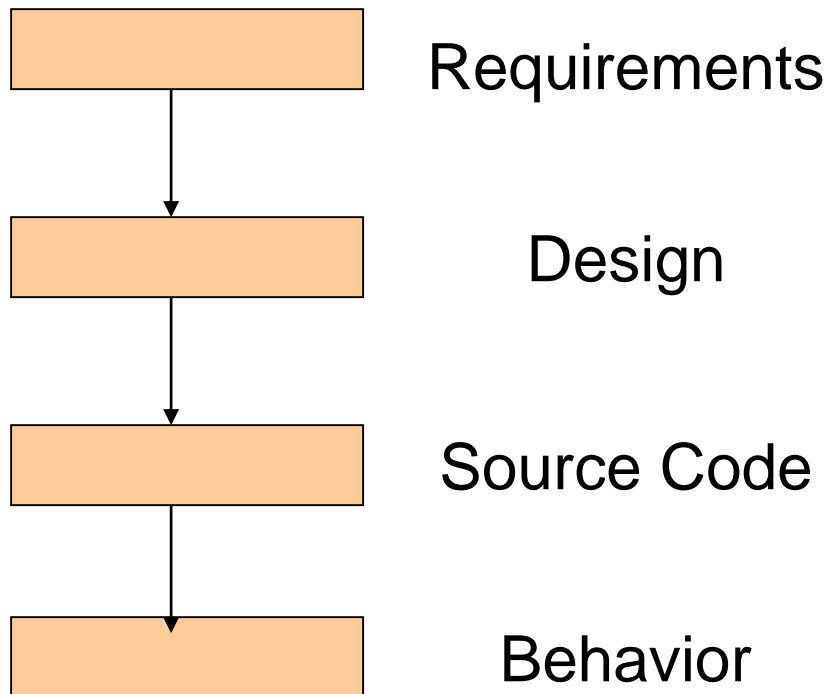
Reverse engineering

Waterfall Model of software development



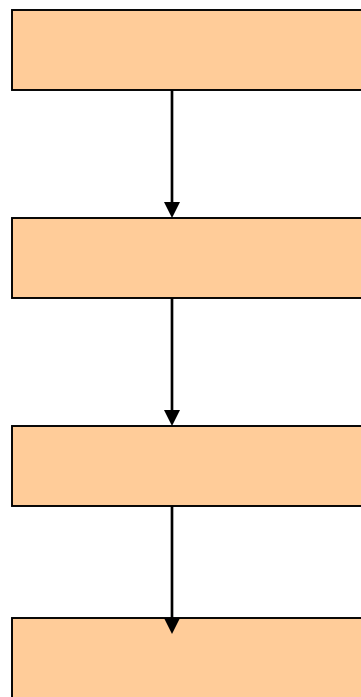
What is Forward Engineering ?

Forward Engineering



What is Reverse Engineering ?

Forward Engineering



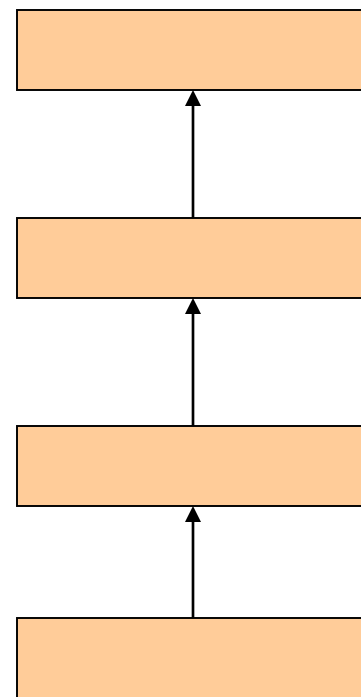
Requirements

Design

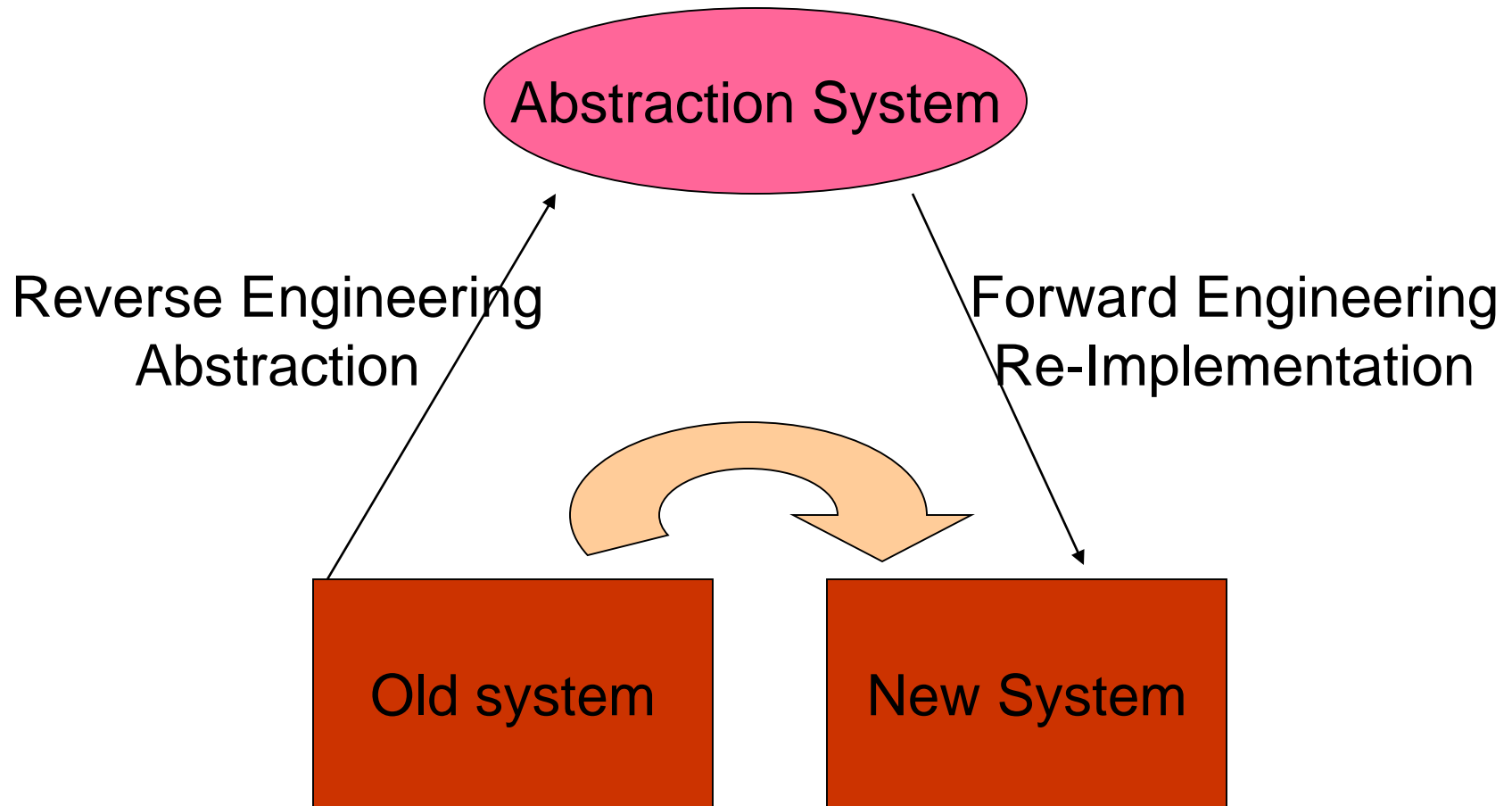
Source Code

Behavior

Reverse Engineering



Reverse Engineering





More about Reverse Engineering

- RE encompasses any activity that is done to determine how a product works, to learn the ideas and technology that were used in developing that product.
- RE can be done at many levels
- RE generally belongs to Software Maintenance

The Early Days of RE



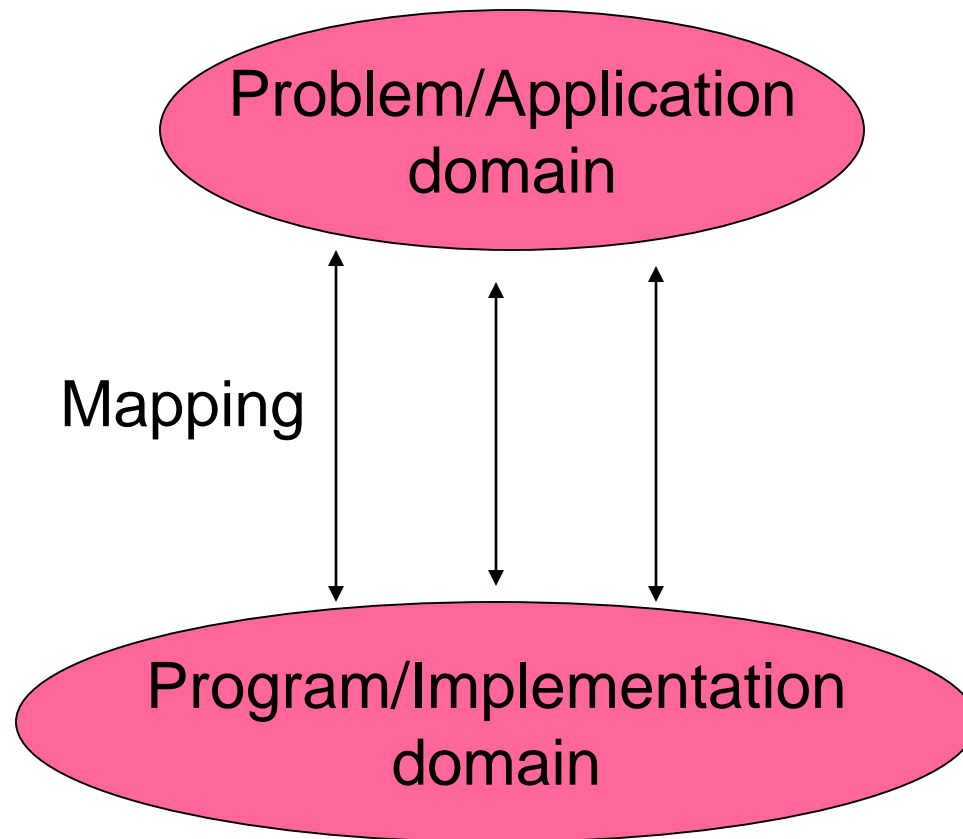
- Law of Software Revolution (Lehman, 1980)
- Fundamental strategies for program comprehension (Brooks, 1983)
- Taxonomy of Reverse Engineering (Chikofsky&Cross, 1990)
- WCRE (Working Conference on R.E., 1990)
- IWPC (Int. Workshop on Program Comprehension)

Why do we need RE ?

- Recovery of lost information
 - providing proper system documentation
- Assisting with maintenance
 - identification of side effects and anomalies
- Migration to another hw/sw platform
- Facilitating software reuse



Scope and Task of Reverse Engineering



Reverse Engineering Terminology

- Design recovery
 - is a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify meaningful higher level abstractions. Design recovery recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domains.
- Redocumentation
 - is the creation or revision of a semantically equivalent representation within the same relative abstraction level.. Redocumentation is the simplest and oldest form of reverse engineering, and can be considered to be an unintrusive, weak form of restructuring.



Reverse engineering

- Reverse engineering often precedes re-engineering but is sometimes worthwhile in its own right
 - The design and specification of a system may be reverse engineered so that they can be an input to the requirements specification process for the system's replacement
 - The design and specification may be reverse engineered to support program maintenance



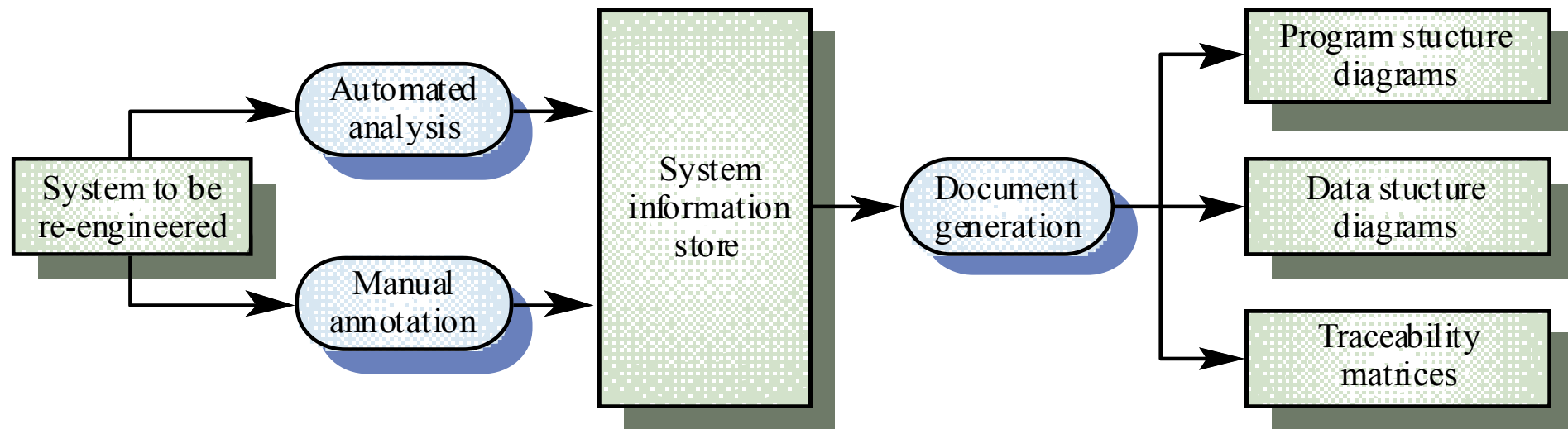
Reverse Engineering

- the software system can be reasoned about in many different views:
 - structural view:
 - functional view:
 - behavioral views:

Reverse Engineering

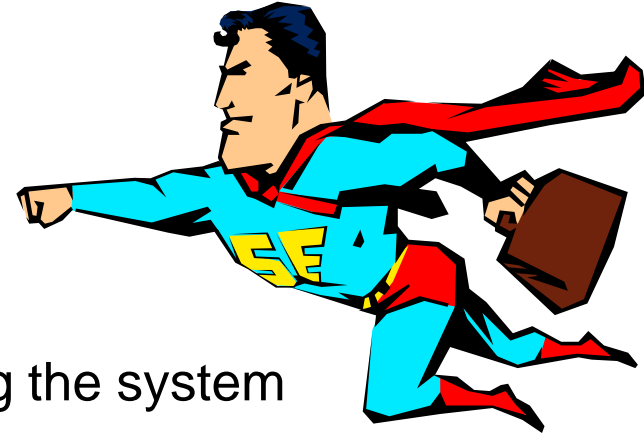
- the software system can be reasoned about in many different views:
 - structural view: the basis is from structure charts, call graphs (unit interaction), module and subsystem graphs, various metrics and organizational views [many can be constructed with CASE tools].
 - functional view: usually can be obtained from the design, specification and requirements documents.
 - behavioral views: conceptual, temporal, process, domain and user interactive views.

The reverse engineering process



What's involved?

- Reverse Engineering
 - Design Recovery
 - reproduce all the info for understanding the system
 - Re-documentation
 - provide different views of the system
- Re-Engineering
 - first phase --understanding the system
 - second phase--forward engineering





When Does an Architect Architect?

- ?



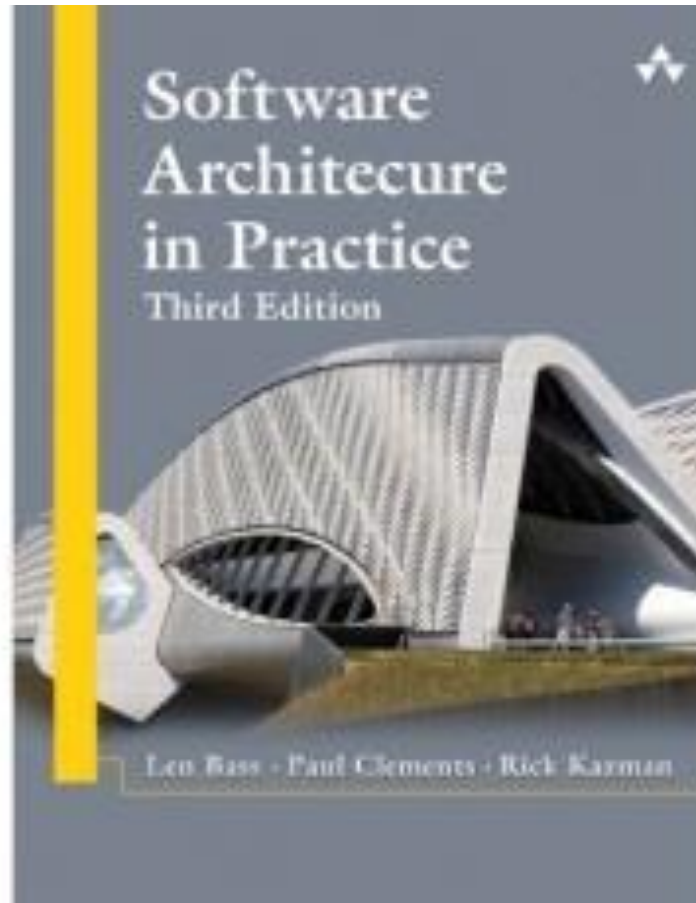
Introduction

“Software architecture reconstruction is a process of obtaining the documented architecture of an existing system.”

- Deursen, Hofmeister, Koschke, Moonen, Riva.

- **Ideal World:** Architectural information is documented during the Architectural design phase and is updated regularly to reflect the current system architecture.
- **Real World:** Architectural information is outdated and does not reflect the current architecture of the system.

Chapter 20



Why Reconstruction?

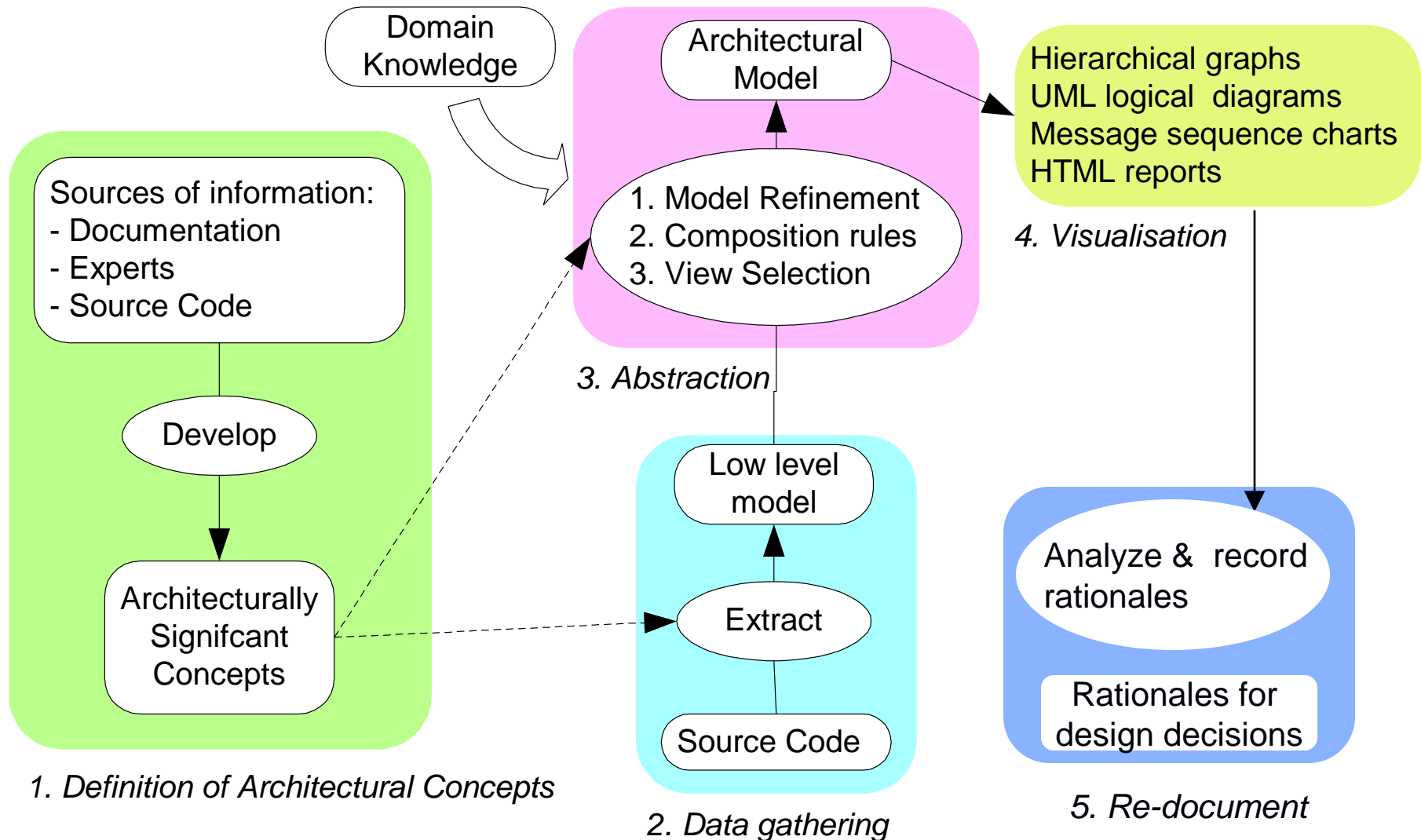
- Suppose you have been given responsibility for a system that already exists, but you do not know its architecture.
 - Perhaps the architecture was never recorded by the original developers, now long gone.
 - Perhaps it was recorded but the documentation has been lost.
 - Perhaps it was recorded but the documentation is no longer synchronized with the system after a series of changes.
 - Perhaps the designers didn't build as architected.
- How do you maintain such a system?
- How do you manage its evolution to maintain the quality attributes that its architecture has provided?



Purposes of Reconstruction

- To document an architecture where the documentation never existed or where it has become hopelessly out of date
- To ensure conformance between the *as-built* architecture and the *as-designed* architecture.
- In architecture reconstruction, the as-built architecture is reverse-engineered from existing system artifacts.
- A form of reverse software engineering

Architecture Reconstruction





Extracting Architecture

- Various techniques
 - Human experts
 - Recognizing known patterns
 - Static and dynamic analysis
 - Clustering and data mining
 - Comparing products within a family



Reconstructing Mappings

- When a system is initially developed, its architectural elements are mapped to specific implementation elements: functions, classes, files, objects, and so forth.
- When we reconstruct those architectural elements, we need to apply the inverses of the original mappings:
 - Use automated and semi-automated extraction tools
 - Probe the original design intent of the architect.
 - Typically we use a combination of both techniques

Tools for Reconstruction

- Architecture reconstruction is a **tool-intensive activity**.
 - Tools extract information about the system, typically by scouring the source code.
 - They may also analyze other artifacts as well, such as build scripts or traces from running systems.
 - We need tools that aid in building and aggregating the architectural abstractions that we need.
 - If our tools are usable and accurate, the end result is an architectural representation that aids the architect in reasoning about the system.
 - Of course, if the original architecture and its implementation are “spaghetti,” the reconstruction will faithfully expose this lack of organization.

...but not Just Tools

- Architecture reconstruction tools are not a **panacea**.
 - It may not be possible to generate a useful architectural representation.
 - Not all aspects of architecture are easy to automatically extract.
 - E.g., there is no programming language construct in any major programming language for “layer” or “connector” or other architectural elements; we can’t simply pick these out of a source code file.
- Architecture reconstruction is an interpretive, interactive, and iterative process involving many activities; it is **not automatic**.
- It requires the skills and attention of both the reverse-engineering expert and, in the best case, the architect (or someone who has substantial knowledge of the architecture).



Reconstruction Workbenches

- An architecture reconstruction *workbench* should be open (making it easy to integrate new tools as required) and provide an integration framework whereby new tools that are added to the tool set do not impact the existing tools or data unnecessarily.



Reconstruction Phases

1. *Raw view extraction.*

- Raw information about the architecture is obtained from various sources, primarily source code, execution traces, and build scripts.
- Each of these sets of raw information is called a view.

2. *Database construction.*

- Convert the raw extracted information into a standard form
- Use that to populate a reconstruction database.
- We will use the database to generate authoritative architecture documentation.

Reconstruction Phases

3. *View fusion and manipulation.*

- Combines the various views of the information stored in the database.
- Individual views may not contain complete or fully accurate information. View fusion can improve the overall accuracy.
- Example: a static view extracted from source code might miss dynamically bound information such as calling relationships. This could then be combined with a dynamic view from an execution trace, which will capture all dynamically bound calling information, but which may not provide complete coverage.

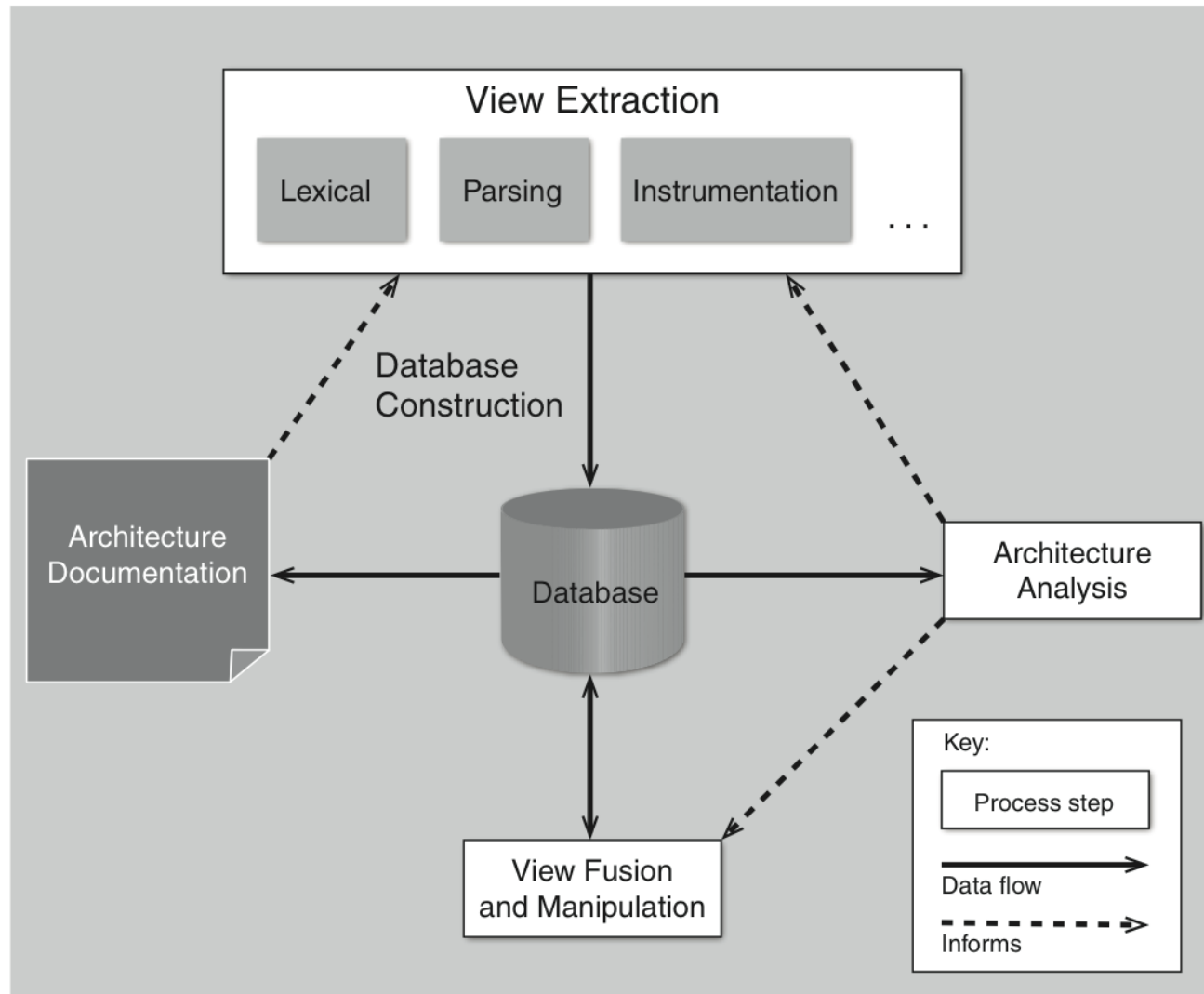


Reconstruction Phases

4. Architecture analysis.

- View fusion will result in a set of hypotheses about the architecture. These hypotheses take the form of architectural elements (such as layers) and the constraints and relationships among them. These hypotheses need to be tested to see if they are correct.
- If not, repeat earlier steps.

Reconstruction Phases



1. Raw View Extraction

- Raw view extraction involves analyzing a system's existing design and implementation artifacts to construct one or more models of it.
- These views will be refined later to support reconstruction goals
 - Supporting the overall architecture documentation effort.
 - Answering specific questions about the architecture.”
- From the source artifacts (code, header files, build files, and so on) and other artifacts (e.g., execution traces), you can identify and capture the elements of interest within the system (e.g., files, functions, variables) and their relationships to obtain several base system views.

Static vs. Dynamic Information

- Static information is obtained by observing only the system artifacts.
- Dynamic information is obtained by observing how the system runs.
 - Some architecturally relevant information may not exist in the source artifacts because of late binding:
 - Polymorphism
 - Function pointers
 - Runtime parameterization
 - Plug-ins
 - Service interactions mediated by brokers
 - The precise topology of a system may not be determined until runtime.
- The goal is to fuse both to create more accurate system views.
- May need to use tools that can generate dynamic information about the system (e.g., profiling tools, instrumentation that generates runtime traces, or aspects in an aspect-oriented programming language that can monitor dynamic activity).

Common Tools for View Extraction

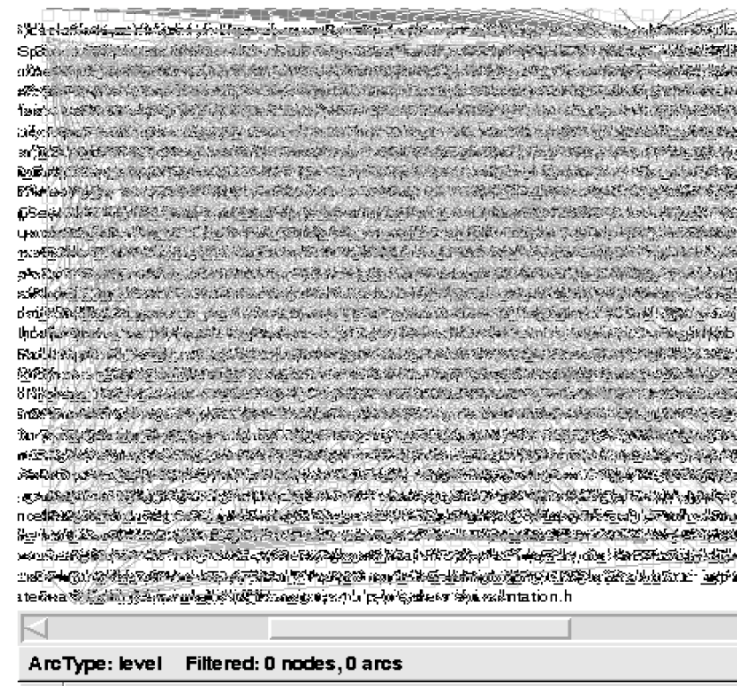
Tool	Static or dynamic	Description
Parsers	Static	Parsers analyze the code and generate internal representations from it (for the purpose of generating machine code). It is possible to save this internal representation to obtain a view.
Abstract Syntax Tree (AST) analyzers		AST analyzers do a similar job to parsers, but they build an explicit tree representation of the parsed information. We can build analysis tools that traverse the AST and output selected pieces of architecturally relevant information in an appropriate format.
Lexical analyzers		Lexical analyzers examine source artifacts purely as strings of lexical elements or tokens. The user of a lexical analyzer can specify a set of code patterns to be matched and output. Similarly, a collection of ad hoc tools such as grep and Perl can carry out pattern matching and searching within the code to output some required information. All of these tools—code-generating parsers, AST-based analyzers, lexical analyzers, and ad hoc pattern matchers—are used to output static information.
Profilers	Dynamic	Profiling and code coverage analysis tools can be used to output information about the code as it is being executed, and usually do not involve adding new code to the system.
Code instrumentation tools		Code instrumentation, which has wide applicability in the field of testing, involves adding code to the system to output specific information while the system is executing. Aspects, in an aspect-oriented programming language, can serve the same purpose and have the advantage of keeping the instrumentation code separate from the code being monitored.

2. Database Construction

- Raw views may be too specific to aid in architectural understanding.

“White noise” view
showing all relations of a
particular sort:

Accurate, but not helpful



2. Database Construction

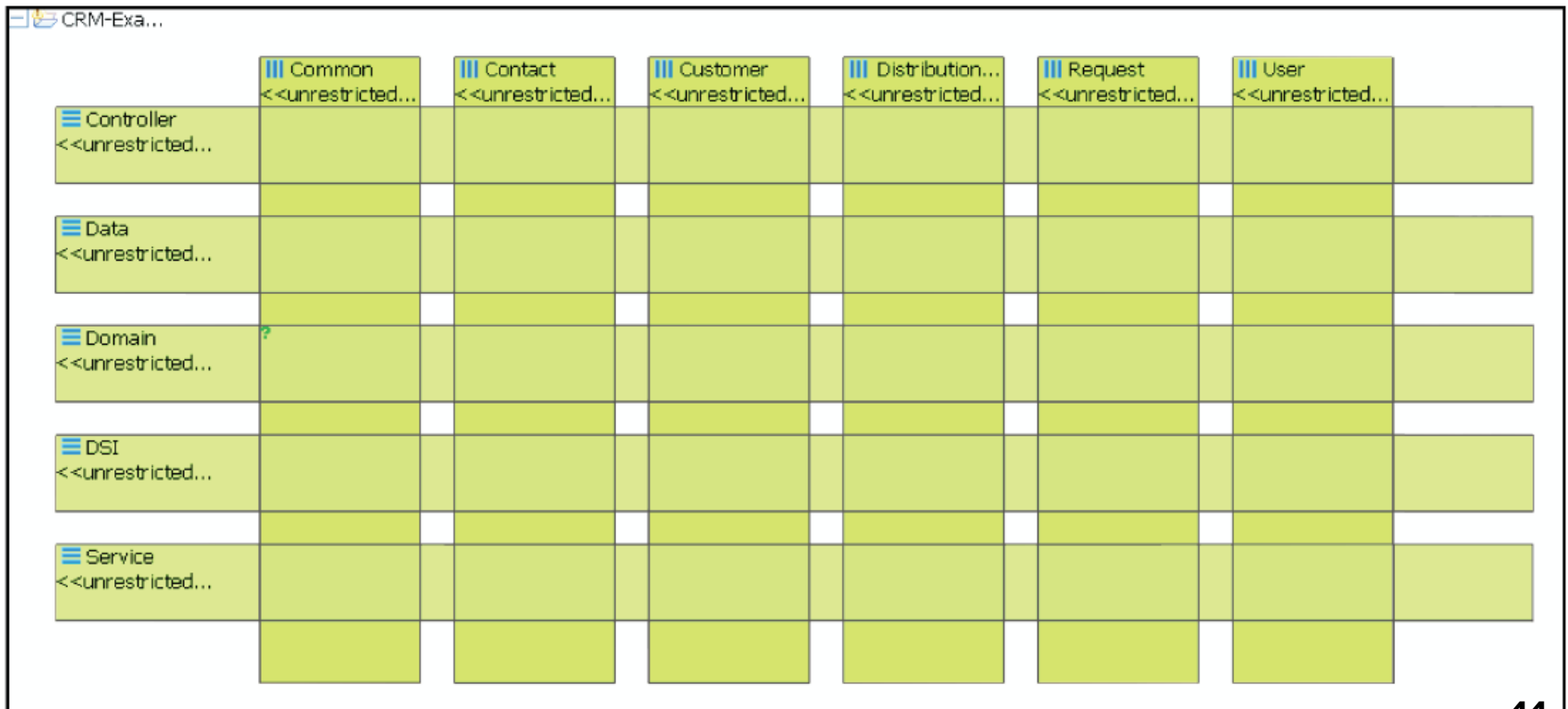
- We need to manipulate such views, to collapse information...
 - E.g., hiding methods inside class definitions
- ...and to show abstractions
 - E.g., showing all of the connections between business objects and user interface objects, or identifying distinct layers
- Use a database to store the extracted information
 - Amount of information being stored is large
 - The manipulations of the data are tedious and error-prone if done manually.
- Some reverse-engineering tools like SonarJ encapsulate a database, and so the user of the tool need not be concerned with its operation.
- Using a suite of tools and building a workbench usually requires choosing a database and internal representations of the views.

3. View Fusion

- Extracted views are manipulated to create **fused views**.
 - Fused views combine information from one or more extracted views, each of which may contain specialized information.
 - For example, a static call view might be fused with a dynamic call view to provide more holistic information about part of the system
- Creating a fused view is creating a hypothesis about the architecture and a visualization of it to aid in analysis.
 - These hypotheses result in new aggregations that show various abstractions or clusterings of the elements
 - By interpreting these fused views and analyzing them, it is possible to produce hypothesized architectural views of the system.
 - These views can be interpreted, further refined, or rejected.
 - There are no universal completion criteria for this process; it is complete when the architectural representation is sufficient to support the analysis needs of its stakeholders.

Example of a Fused View

Table shows early results from the tool SonarJ. SonarJ first extracts facts from a set of source code files and lets you define a set of layers and vertical slices through those layers in a system. SonarJ will instantiate the user-specified definitions of layers and slices and populate them with the extracted software elements.



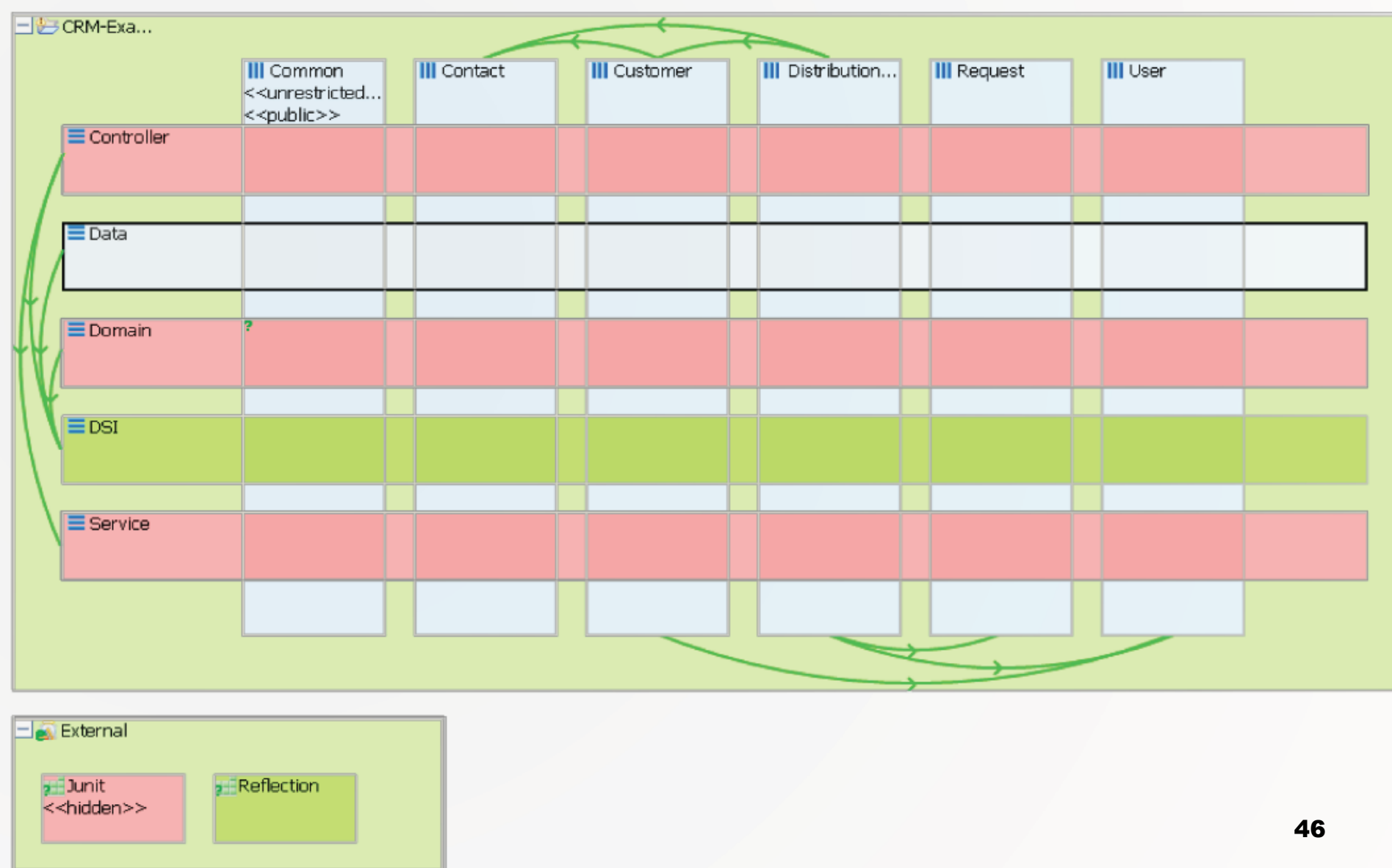
The screenshot shows the SonarJ CRM-Example interface. It features a table with 6 columns representing vertical slices and 5 rows representing layers. The columns are labeled: Common, Contact, Customer, Distribution..., Request, and User. The rows are labeled: Controller, Data, Domain, DSI, and Service. Each cell in the table is a light green rectangle. A small green question mark icon is visible in the first cell of the Domain row.

	Common <<unrestricted...	Contact <<unrestricted...	Customer <<unrestricted...	Distribution... <<unrestricted...	Request <<unrestricted...	User <<unrestricted...
Controller <<unrestricted...						
Data <<unrestricted...						
Domain <<unrestricted...	?					
DSI <<unrestricted...						
Service <<unrestricted...						

4. Architectural Analysis: Finding Violations

- View fusion gave us a set of hypotheses about the architecture.
 - These hypotheses take the form of architectural elements (sometimes aggregated, such as layers) and the constraints and relationships among them.
- These hypotheses need to be tested to see if they are correct—to see if they conform with the architect's intentions.
- Figure 20.4 shows the results of adding relationships and constraints to the architecture initially created in previous figure. These relationship and constraints are information added by the architect, to reflect the design intent.
- In this example, the architect has indicated the relationships between the layers.
- Using these relationships and constraints, a tool such as SonarJ is able to automatically detect and report violations of the layering in the software.

Figure 20.4

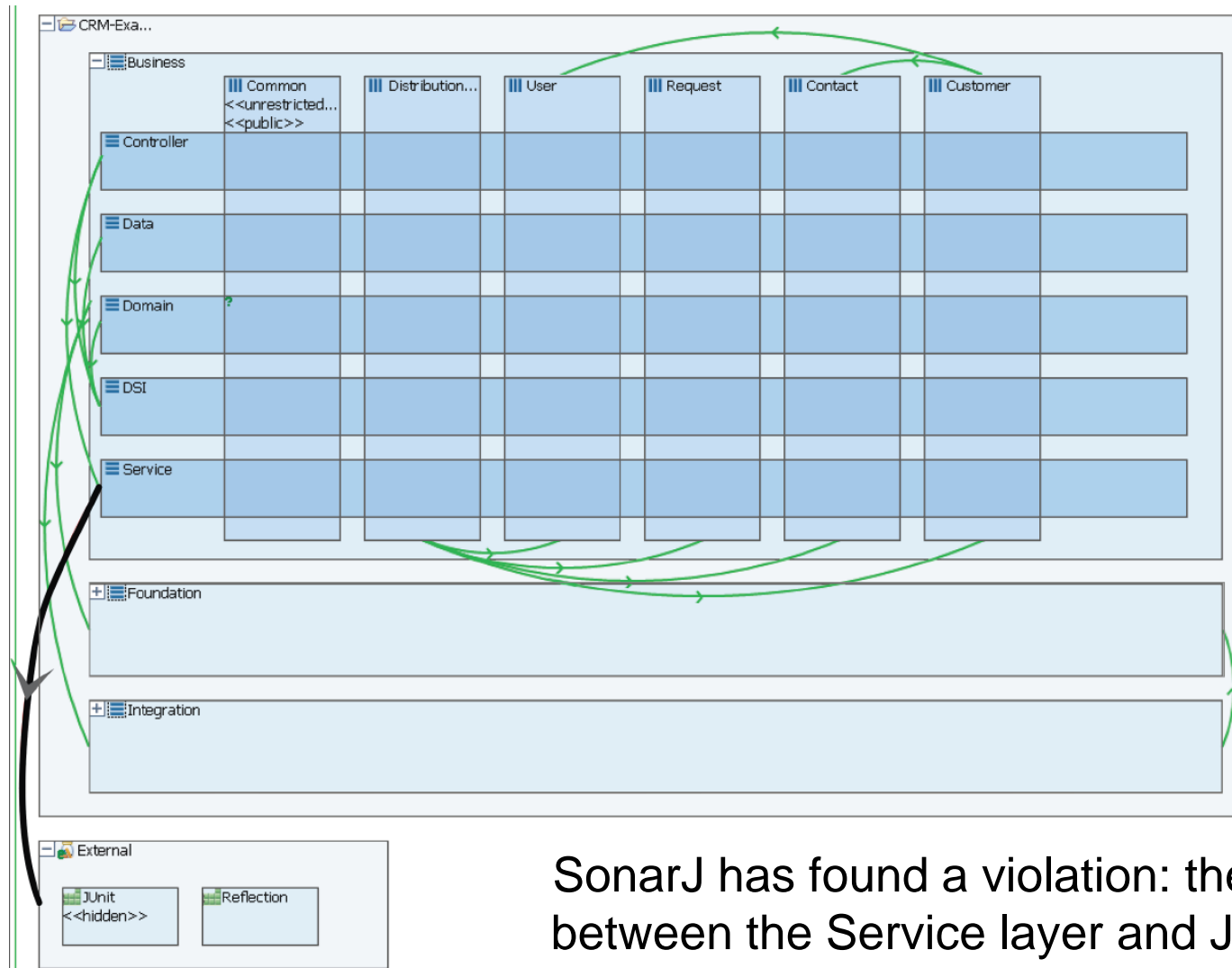




What Does the Figure Tell Us?

- Data layer (row 2) can access, and hence depends on, the DSI layer.
- Data layer may not access, and has no dependencies on, Domain, Service, or Controller (rows 1, 3, and 5).
- The JUnit component in the “External” component is defined to be inaccessible.
 - No portion of the application should depend upon JUnit, because this should only be used by test code.

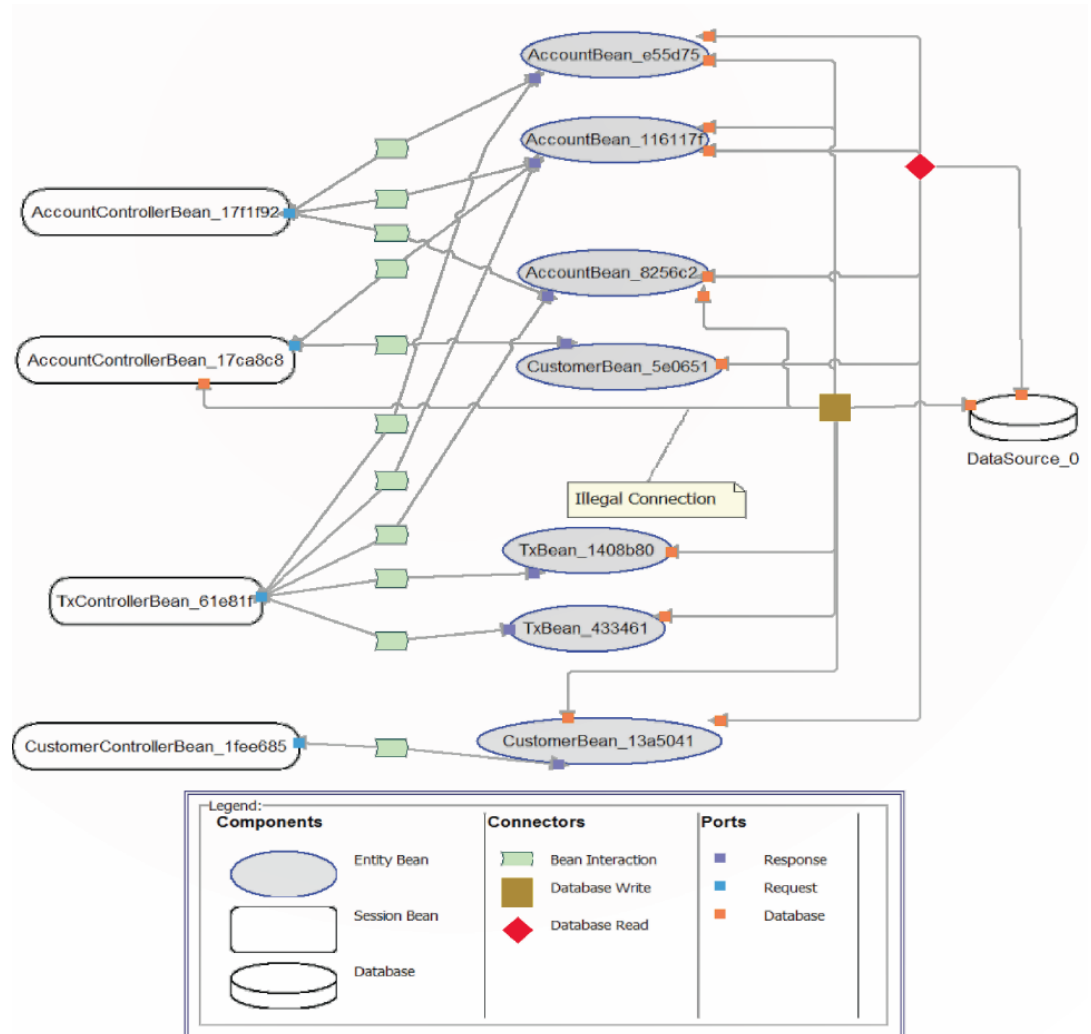
Finding a Violation



SonarJ has found a violation: the arc between the Service layer and JUnit.

Finding Dynamic Violations

- What if we need to test a C&C hypothesis?
- To analyze the runtime architecture of the Duke's Bank sample EJB application, the code was "instrumented" using AspectJ.
- The figure shows that a "database write" connector was discovered in the dynamic analysis of the architecture.
- The documented architecture of Duke's Bank forbids such connections.
- All database access is supposed to be managed by entity beans, and only by entity beans.
- This would be difficult to find just analyzing the source code.





Guidelines

- Have a goal and a set of objectives or questions in mind before undertaking an architecture reconstruction project.
- Obtain some representation, however coarse, of the system before beginning the detailed reconstruction process. This representation serves several purposes, including the following:
 - It identifies what information needs to be extracted from the system.
 - It guides the reconstructor in determining what to look for in the architecture and what views to generate.
- Identifying layers is a good place to start.



Guidelines

- The existing documentation for a system may not accurately reflect the system as it is implemented. Therefore it may be necessary to disregard the existing documentation.
- Tools can support the reconstruction effort and shorten the reconstruction process, but they cannot do an entire reconstruction effort automatically.
 - The work involved in the effort requires the involvement of people (architects, maintainers, and developers) who are familiar with the system.

- 
- How to recover bytecode from .class file under Unix/Windows with JDK?

Example--Java Decompiler

- How to recover bytecode from .class file under Unix/Win with JDK?

% **javap -c** <filename>

% **javap -help** (to see the options)

- Java Decompilers

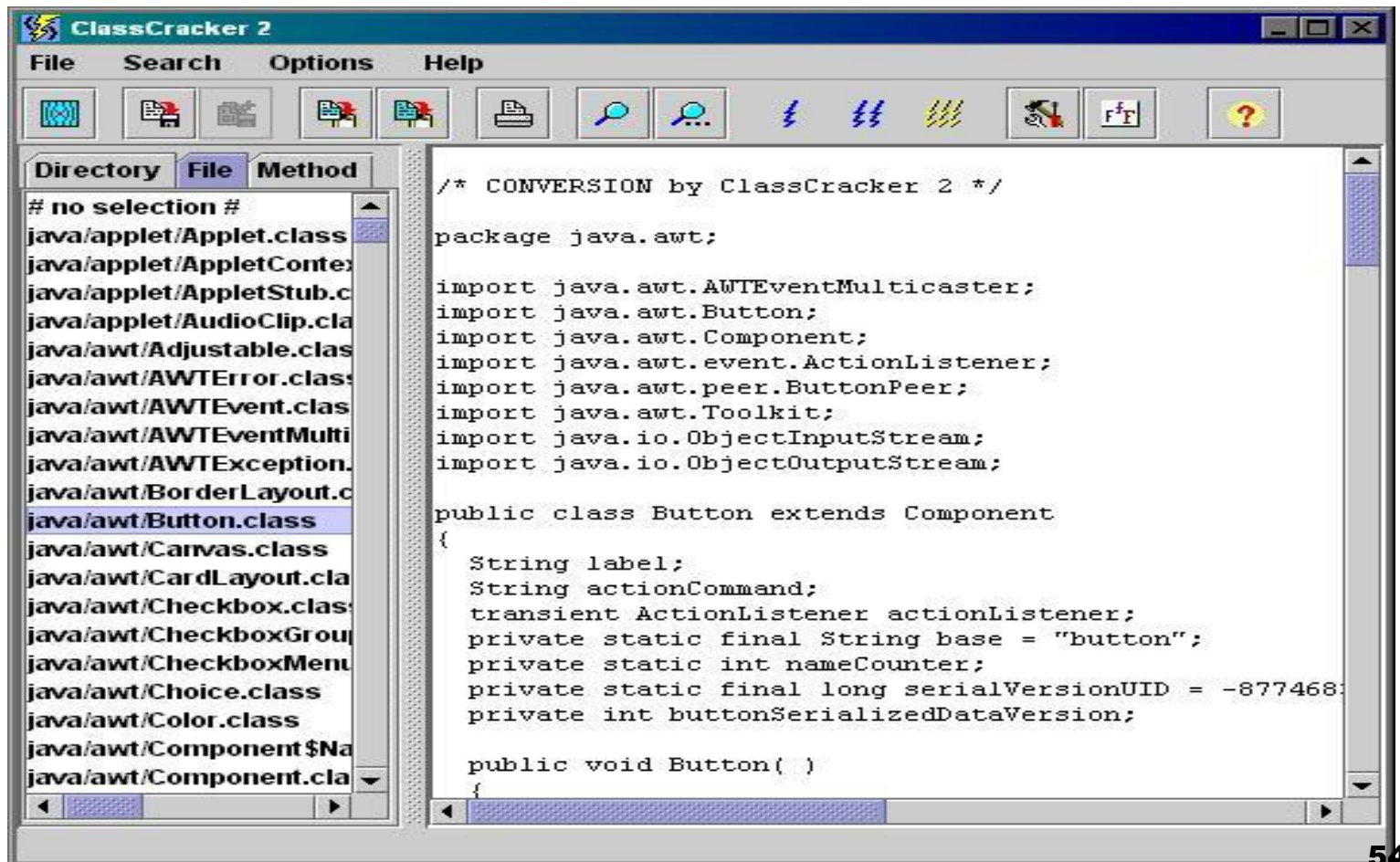
- ☐ "ClassCracker" <http://www.pcug.org.au/~mayon/>

- ☐ "DeCafe Pro" from DeCafe, France at <http://decafe.hypermart.net/index.htm>

- ☐ "SourceAgain" from Ahpah corp at <http://www.ahpah.com>

Example--Java Decompiler

- ClassCracker 2 Interface



Example--Java Decompiler

- Components of ClassCracker 2
 - Java decompiler
 - retrieves Java source code from Java class files
 - Java disassembler
 - produces Java Assembly Code
 - A Java class file viewer
 - displays Java class file structures.



Example--Java Decompiler

- Features of ClassCracker 2
 - User visual interface.
 - Can decompile class files within zip or jar files.
 - Conversion mode (JAVA, JASM or JDUMP) is selectable
 - A Batc Mode allows multiple class files to be decompiled simultaneously
 - more.....





Example--Java Decompiler

- ClassCracker 2 System Requirements
 - All platform (Window/Linus/Unix)
 - JDK /JRE
- Do not believe it?
 - From myClass_origin.class ==>myClass.java
 - % **javac** myClass.java (==>myClass.class)
 - % **diff** myClass.class myClass_origin.class



Example--Java Decompiler

- ClassCracker 2.0--want to try it?
 - Free download at
<http://www.pcug.org.au/~mayon/classcracker/ccgetdemo.html>
- Bridge 1.0---Free
 - <http://www.geocities.com/SiliconValley/Bridge/8617/jad.html>

- 
- 
- What is the difference in reverse engineering two different high level languages - Java & C++?

Summary

- Architecture reconstruction and architecture conformance are crucial tools in the architect's toolbox to ensure that a system is built the way it was designed, and that it evolves in a way that is consistent with its creators' intentions.
- The results of architectural reconstruction can be used in several ways:
 - If no documentation exists or if it is seriously out of date, the recovered architectural representation can be used as a basis for documenting the architecture.
 - It can be used to recover the as-built architecture, or to check conformance against an "as-designed" architecture.
 - The reconstruction can be used as the basis for analyzing the architecture or as a starting point for reengineering the system to a new desired architecture.
 - The representation can be used to identify elements for reuse or to establish an architecture-based software product line.



Summary

The software architecture reconstruction process comprises the following phases:

1. *Raw view extraction*
2. *Database construction*
3. *View fusion*
4. *Architecture analysis*

