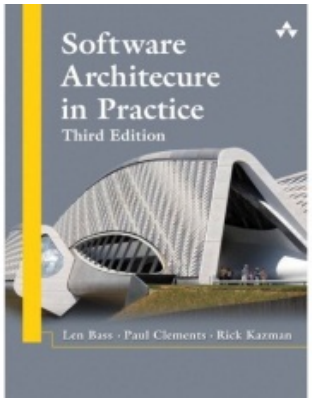
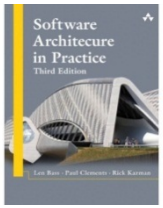
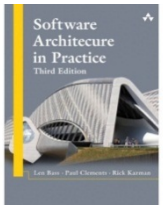


Chapter 20: Architecture Reconstruction and Conformance



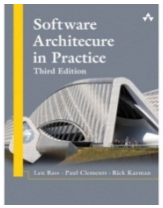
Chapter Outline

- Architecture Reconstruction Process
- Raw View Extraction
- Database Construction
- View Fusion
- Architecture Analysis: Finding Violations
- Guidelines
- Summary



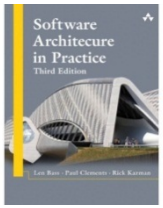
Why Reconstruction?

- 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.
- How do you maintain such a system?
- How do you manage its evolution to maintain the quality attributes that its architecture has provided?



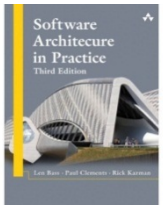
Purposes of Reconstruction

- 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.



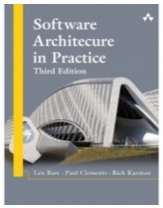
Reconstructing Mappings

- 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



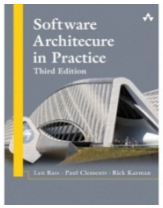
Tools for Reconstruction

- 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.



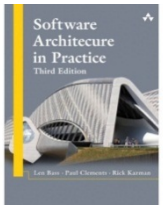
…but not Just Tools

- 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).



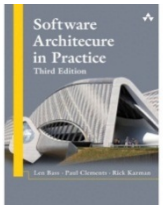
Reconstruction Workbenches

- 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.



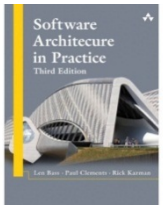
Reconstruction Phases

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.

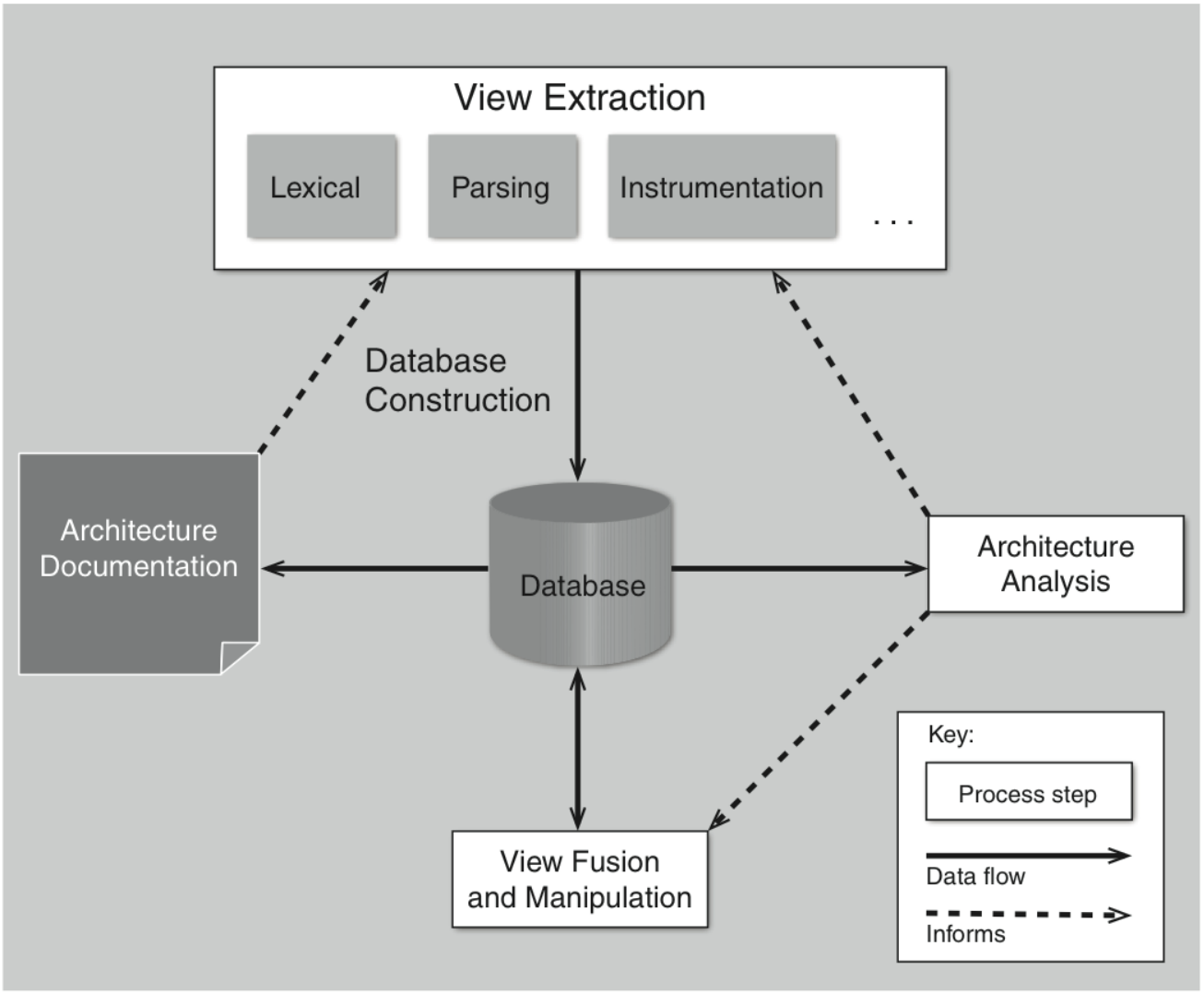


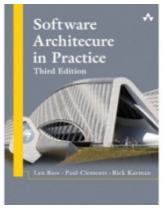
Reconstruction Phases

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.
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.



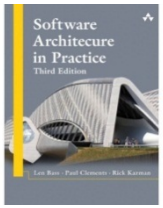
Reconstruction Phases





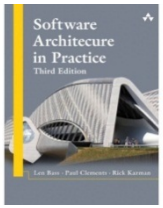
1. Raw View Extraction

- 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."
- Blend of the ideal (what information do you want to discover about the architecture that will most help you meet the goals of your reconstruction effort?) and the practical (what information can your available tools actually extract and present?)
- 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.



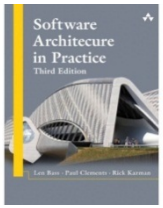
Typical List of Extracted Elements and their Relationships

Source Element	Relation	Target Element	Description
File	`includes`	File	C preprocessor `#include` of one file by another
File	`contains`	Function	Definition of a function in a file
File	`defines_var`	Variable	Definition of a variable in a file
Directory	`contains`	Directory	Directory contains a subdirectory
Directory	`contains`	File	Directory contains a file
Function	`calls`	Function	Static function call
Function	`access_read`	Variable	Read access on a variable
Function	`access_write`	Variable	Write access on a variable



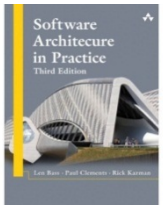
Static vs. Dynamic Information

- 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 instrumen-tation 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.

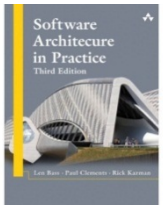


2. Database Construction

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

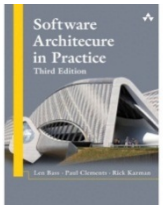
"White noise" view showing all relations of a particular sort:

Accurate, but not helpful



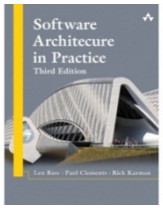
2. Database Construction

- 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 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.



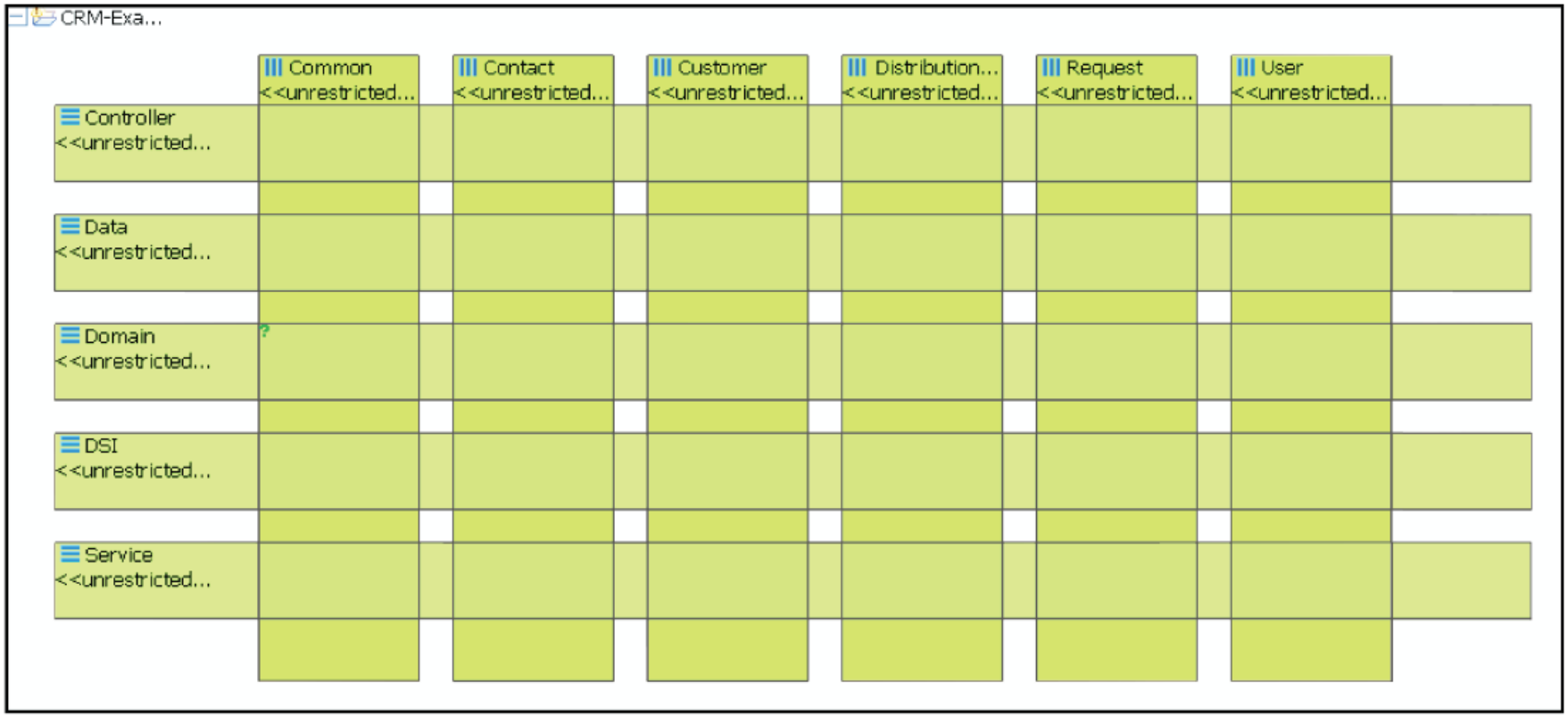
3. View Fusion

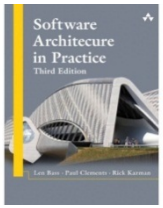
- Extracted views are manipulated to create *fused* views.
  - Fused views combine information from one or more extracted views, each of which may contain specialized information.
  - For example, a static call view might be fused with a dynamic call view to provide more holistic information about part 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 (which may be source artifacts or previously identified abstractions).
  - 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.





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.

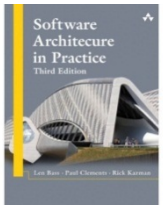
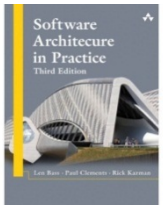

Software
Architecure
in Practice
Third Edition

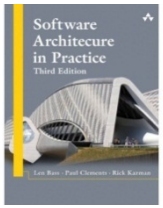
Figure 20.4


CRM-Exa...
Common
<<unrestricted...
<<public>>
Contact
Customer
Distribution...
Request
User
Controller
Data
Domain
DSI
Service
External
Junit
<<hidden>>
Reflection

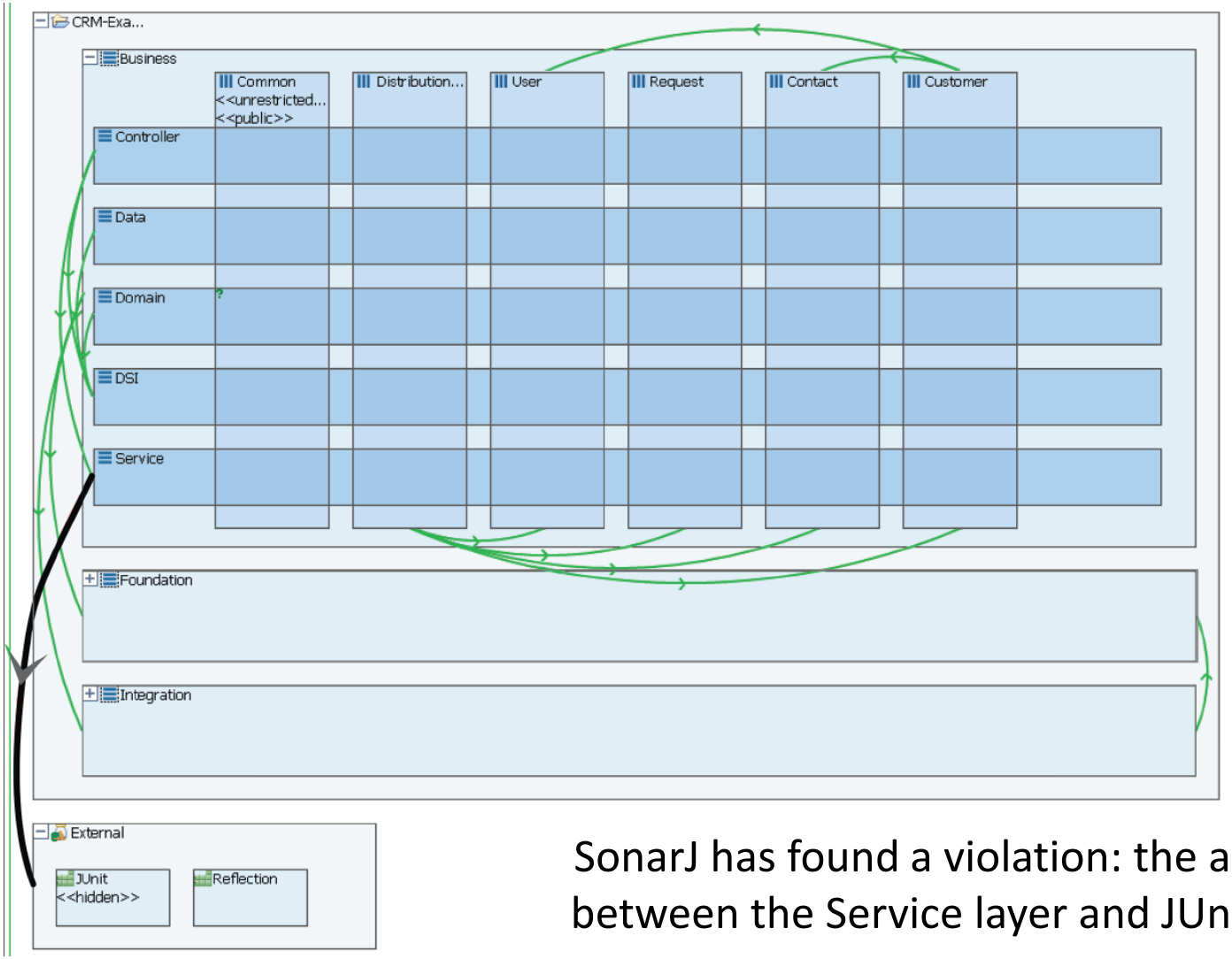


What Does the Figure Tell Us?

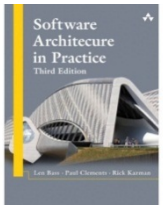
- Data layer (row 2) can access, and hence depends on, the DSI layer.
- Data layer may not access, and has no dependencies on, Domain, Service, or Controller (rows 1, 3, and 5).
- The JUnit component in the “External” component is defined to be inaccessible.
  - No portion of the application should depend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



Finding a Violation

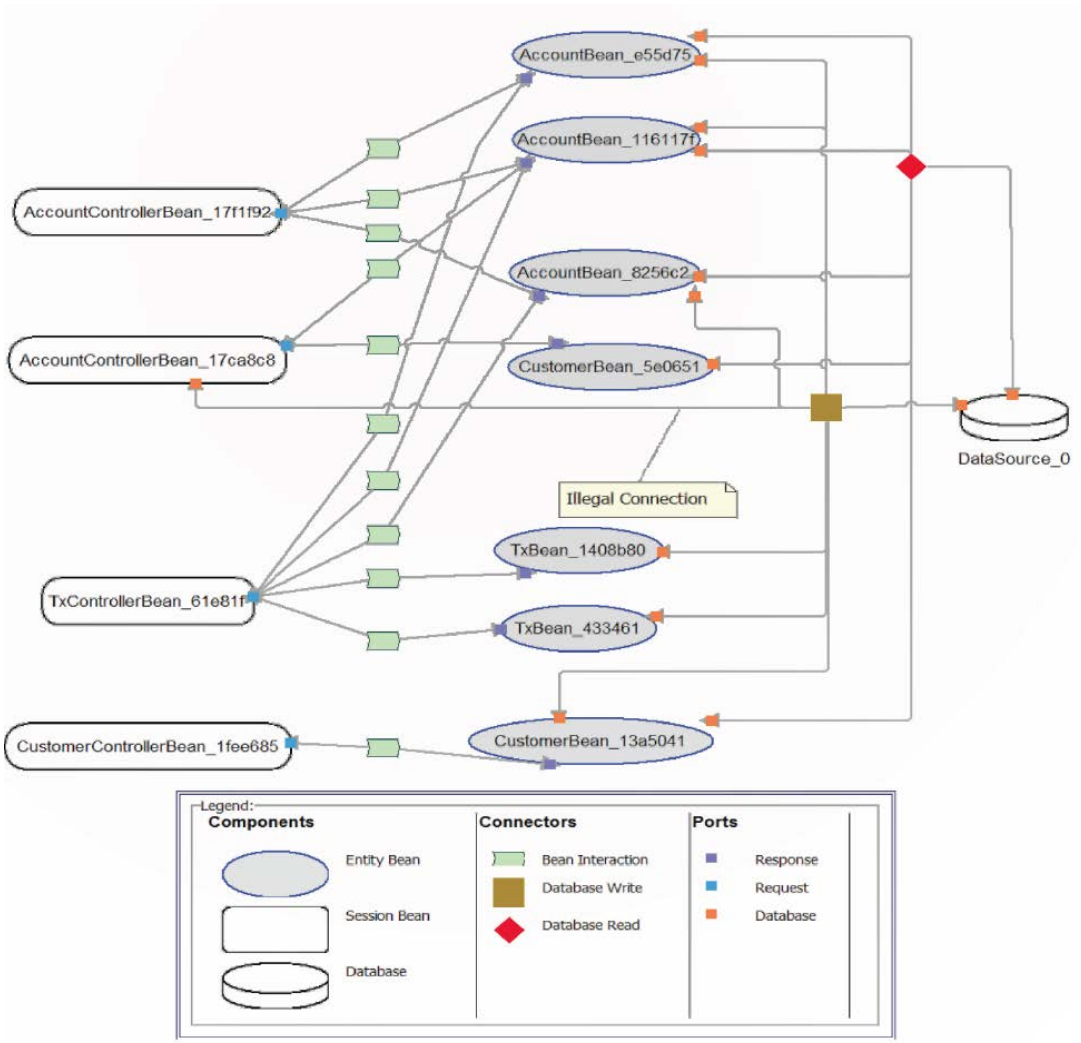


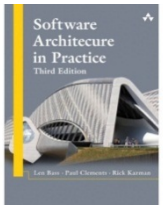
SonarJ has found a violation: the arc between the Service layer and JUnit.



Finding Dynamic Violations

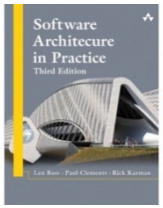
- What if we need to test a C&C hypothesis?
- To analyze the runtime architecture of the Duke's Bank sample EJB application, the code was "instrumented" using AspectJ.
- The figure shows that a "database write" connector was discovered in the dynamic analysis of the architecture.
- The documented architecture 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.





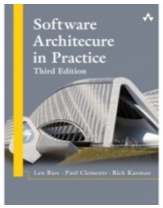
Guidelines

- 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.



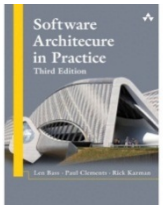
Guidelines

- 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.



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*