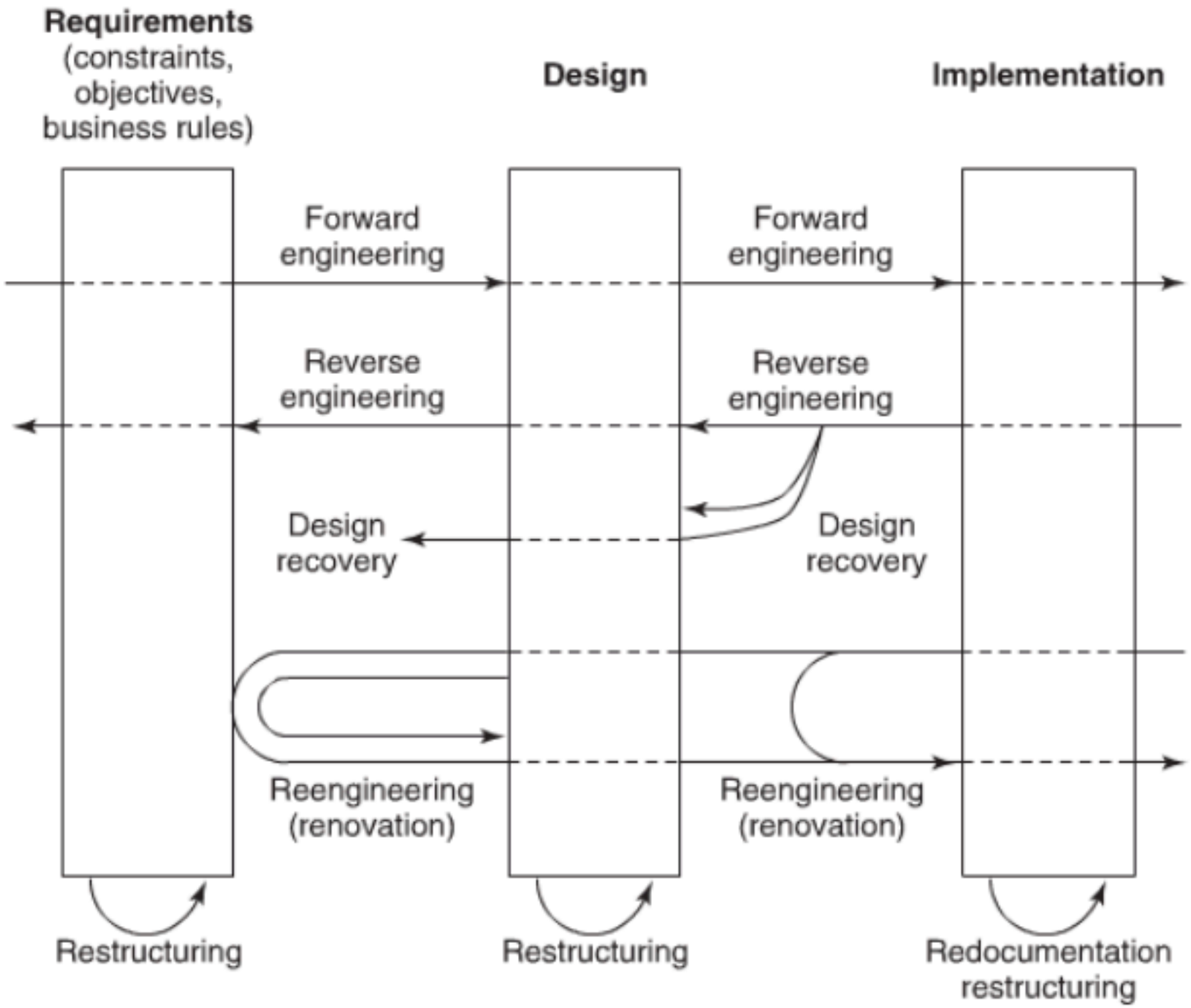
Architecture Recovery

Instructor: Yongjie Zheng

CS 5553: Software Architecture and Design

Definitions

- **Forward Engineering**: the process of moving from high-level abstractions and implementation-independent designs to the implementation of a system.
- **Reverse Engineering**: the process of creating high-level abstractions from source code or other software artifacts.
- **Architecture Recovery**: the process of extracting a system's architectural design by examining source code and other related artifacts.
- **Reengineering**: the process of reconstructing a system in a new form. It generally includes some form of reverse engineering followed by some form of forward engineering.
- **Restructuring**: the transformation from one representation form to another at the same abstraction level, while preserving the subject system's external behavior.



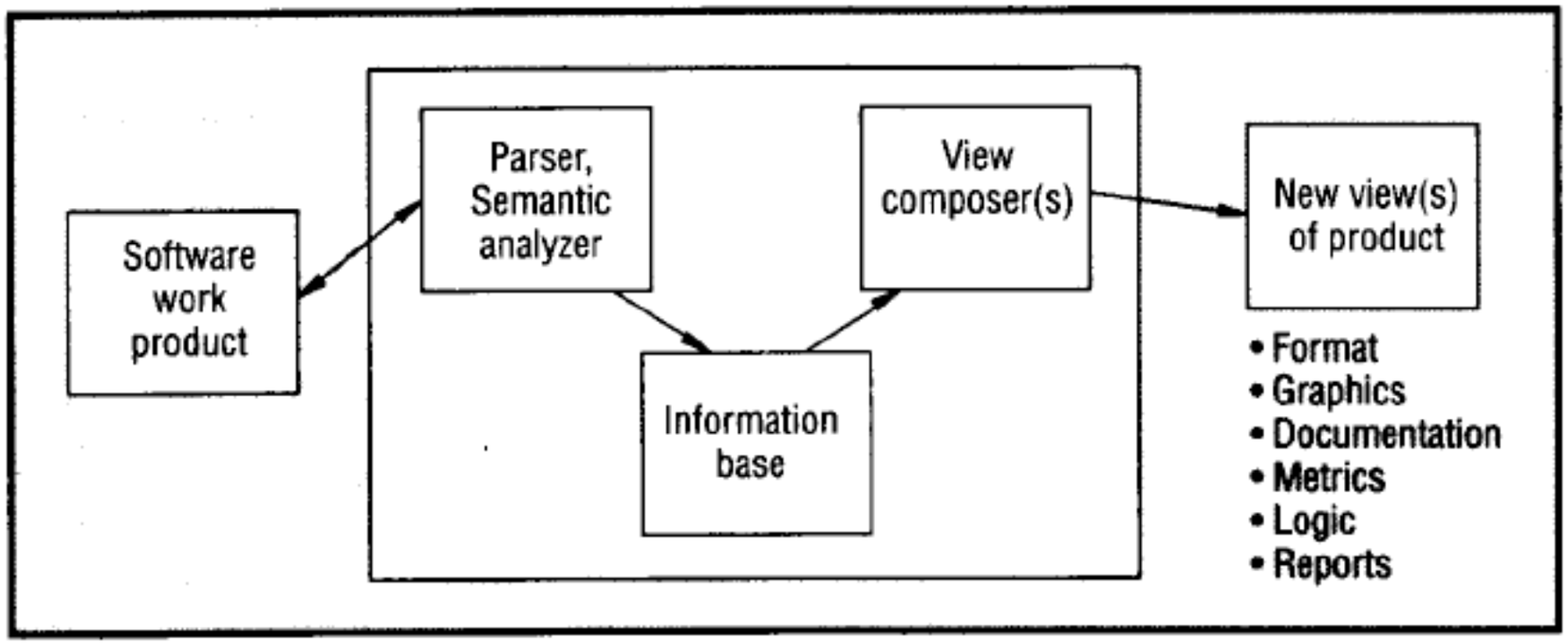
Excerpted from [CC90]

Applications of Architecture Recovery

- After-the-fact documentation.
  - What happens if a system is already implemented but has no recorded architecture?
- Program understanding in software maintenance.
- Consistency checking for architecture-implementation conformance.
  - Software architecture can easily become out of date with architecture and code under frequent changes during development and evolution.
- System optimization (reengineering).

Architecture recovery includes the activities of

- **Extraction**: collecting the structure and behavioral information about the system.
- **Abstraction**: clustering source code entities (e.g. classes, procedures) into architecture elements (e.g. components).
- (Optional) Clustering components into subsystems.
- Determining what relations exist between architecture elements based on the relations between the source code entities that are contained in the architecture elements.
- Visualization/modeling of the recovered architecture.



Excerpted from [CC90]

Source Code Analysis (Extraction)

- Structural information is usually extracted by statically analyzing the system's source code (i.e. static analysis).
- Behavioral information is collected during the system's execution (i.e. dynamic analysis).
- This step is well supported by some automated tools.
- What specific tools are you going to use in your project?

Available Tools for Code Analysis

- ObjectAid UML: http://www.objectaid.com/
- AgileJ: http://www.agilej.com/
- Architexa: http://www.architexa.com/
- eUML2: http://www.soyatec.com/euml2/
- Class Visualizer: http://www.class-visualizer.net/

Static Analysis

- Static analysis (usually done with a source code extractor) extracts the information about control-flow and data-flow dependencies, such as
  - Procedure calls
  - Definitions and usages of variables
  - Or even more: software organization, association and inheritance relationships among classes
- The output of static analysis could be:
  - A call graph
  - A set of relations saved in a file or database
  - Other source models

Dynamic Analysis

- Dynamic analysis collects a system's runtime information, such as
  - Interaction frequency (e.g. high cohesion and low coupling).
  - Events if an event-based execution model is used.
- Dynamic analysis often requires the system be instrumented and monitored during execution.
- Dynamic analysis can accommodate systems whose architecture changes dynamically.

Clustering (Abstraction)

- A set of criteria (viewpoints) are usually applied to cluster source code elements into architecture elements.
  - E.g. naming conventions, graph characteristics (strongly connected components).
- Many clustering techniques exist. Depending on what kinds of source code information they examine, these techniques can be roughly classified as
  - Syntactic clustering
  - Semantic clustering
- This step usually requires human interpretation.

Syntactic Clustering

- Focuses exclusively on the static relationships among code-level entities.
- Can be performed without executing the system.
- Examples of syntactic clustering criteria:
  - Entities belonging to the same package or file
  - Entities with similar names
  - Entities working on the same set of data
- May ignore or misinterpret subtle relationships, because dynamic information is missing.

Semantic Clustering

- Includes all aspects of a system's domain knowledge and information about the behavioral similarity of its entities.
- Requires interpreting the system entities' meaning, and possibly executing the system on a representative set of inputs.
- Examples of semantic clustering criteria
  - High cohesion and loose coupling
  - Entities related to the same fault
  - Entities with similar functionality

Representative Systems

- Systems that involve static analysis and syntactic clustering.
  - Regi, Dali
- Systems that involve dynamic analysis and semantic clustering.
  - DiscoTect, Pattern-Lint, Rapide
- Success stories of architecture recovery
  - The Apache architecture recovery project at the Hasso-Plattner Institute in Potsdam, Germany.
  - Architecture recovery for Linux ([BHB99]).

Variations of Architecture Recovery

- Also called top-down, or hybrid, architecture recovery. In contrast, the approach we introduced earlier is called bottom-up.
- Its primary characteristic is that a high-level model of the system (i.e. conceptual architecture) usually exists.
- The high-level model serves as a reference in the process of architecture recovery, and may be refined at the end.
- Example: software reflexion model ([MNS01]).

Challenges of Architecture Recovery

- Architecture recovery usually requires the system to be relatively complete, or even executable.
  - May not work in software development.
- Architecture recovery is hard to fully automate.
  - Machines are not good at abstraction.
  - Most methods are semi-automated.
- Design intent and rationale are hard to recover.