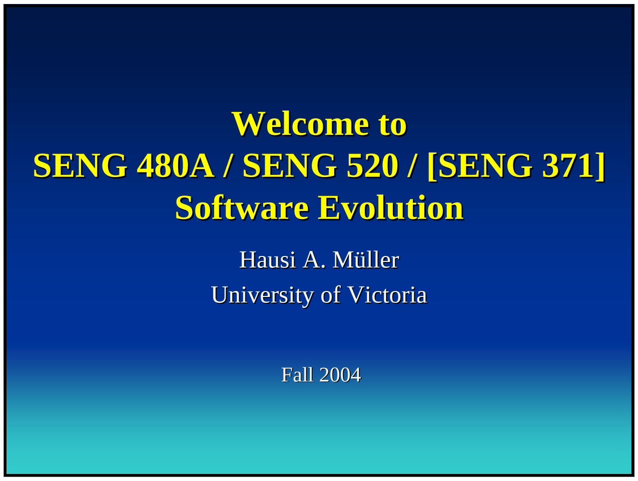
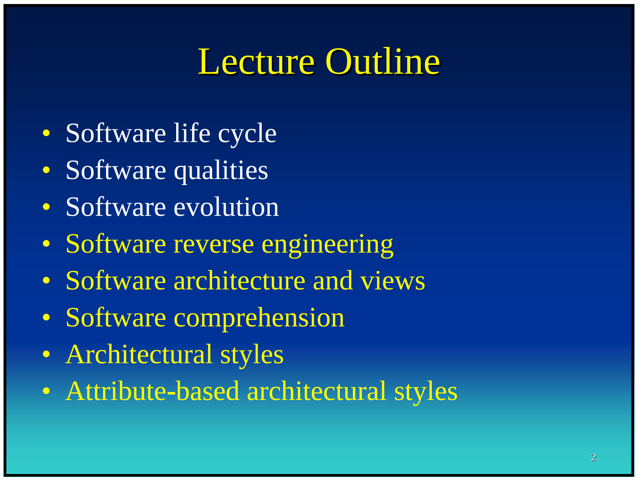


Welcome to SENG 480A / SENG 520 / [SENG 371] Software Evolution

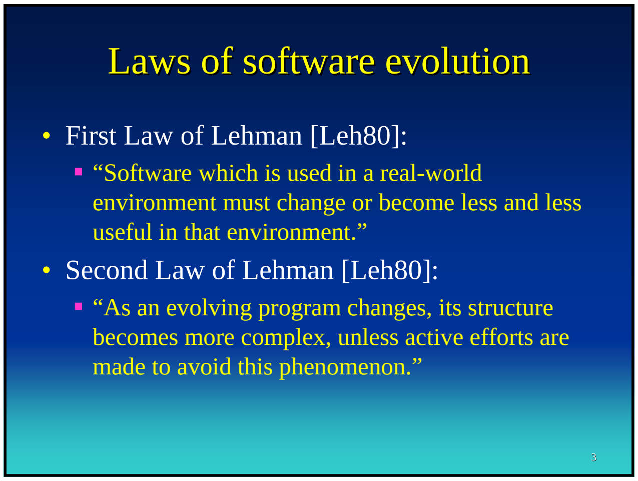
Hausi A. Müller
University of Victoria

Fall 2004



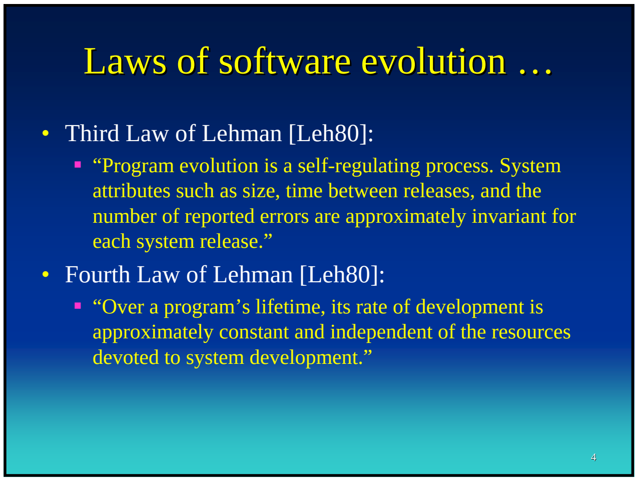
Lecture Outline

- Software life cycle
- Software qualities
- Software evolution
- Software reverse engineering
- Software architecture and views
- Software comprehension
- Architectural styles
- Attribute-based architectural styles



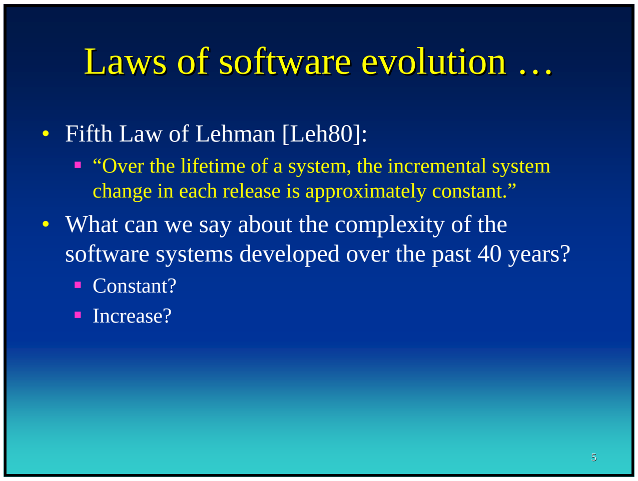
Laws of software evolution

- First Law of Lehman [Leh80]:
  - "Software which is used in a real-world environment must change or become less and less useful in that environment."
- Second Law of Lehman [Leh80]:
  - "As an evolving program changes, its structure becomes more complex, unless active efforts are made to avoid this phenomenon."



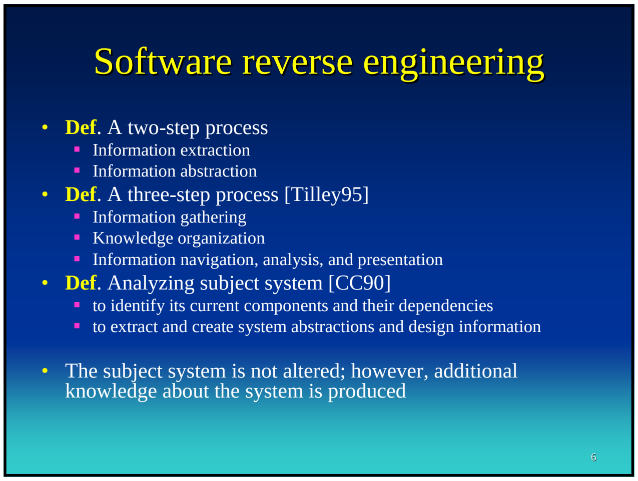
Laws of software evolution …

- Third Law of Lehman [Leh80]:
  - "Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors are approximately invariant for each system release."
- Fourth Law of Lehman [Leh80]:
  - "Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development."



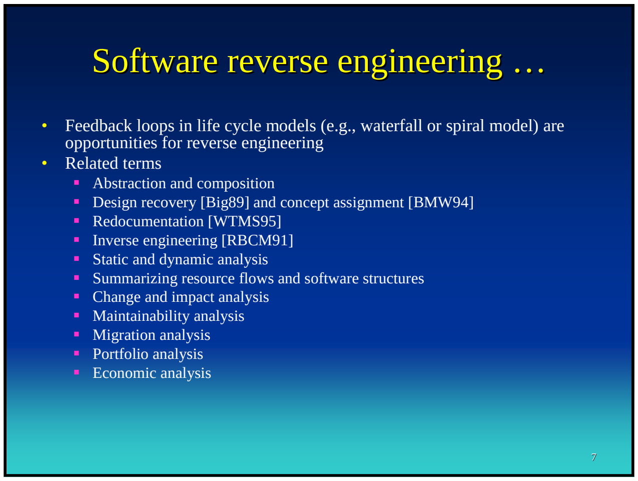
Laws of software evolution …

- Fifth Law of Lehman [Leh80]:
  - "Over the lifetime of a system, the incremental system change in each release is approximately constant."
- What can we say about the complexity of the software systems developed over the past 40 years?
  - Constant?
  - Increase?



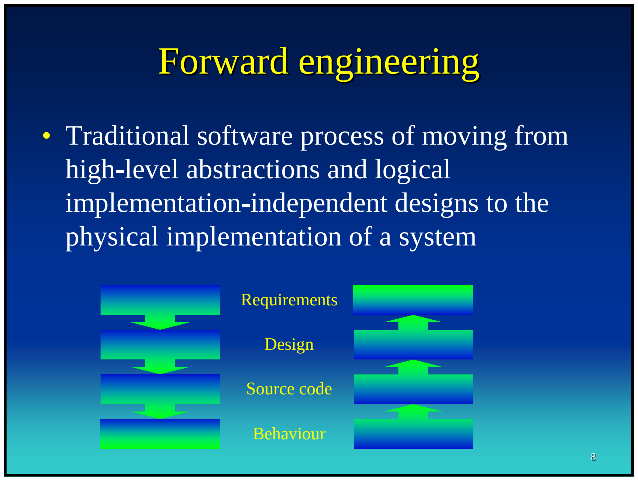
Software reverse engineering

- **Def**. A two-step process
  - Information extraction
  - Information abstraction
- **Def**. A three-step process [Tilley95]
  - Information gathering
  - Knowledge organization
  - Information navigation, analysis, and presentation
- **Def**. Analyzing subject system [CC90]
  - to identify its current components and their dependencies
  - to extract and create system abstractions and design information
- The subject system is not altered; however, additional knowledge about the system is produced



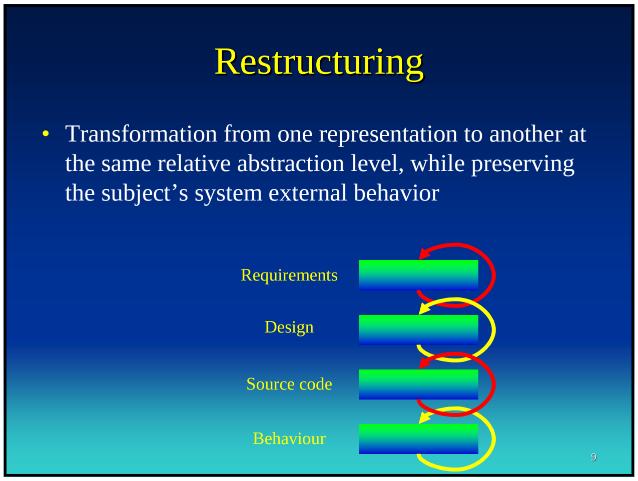
Software reverse engineering …

- Feedback loops in life cycle models (e.g., waterfall or spiral model) are opportunities for reverse engineering
- Related terms
  - Abstraction and composition
  - Design recovery [Big89] and concept assignment [BMW94]
  - Redocumentation [WTMS95]
  - Inverse engineering [RBCM91]
  - Static and dynamic analysis
  - Summarizing resource flows and software structures
  - Change and impact analysis
  - Maintainability analysis
  - Migration analysis
  - Portfolio analysis
  - Economic analysis



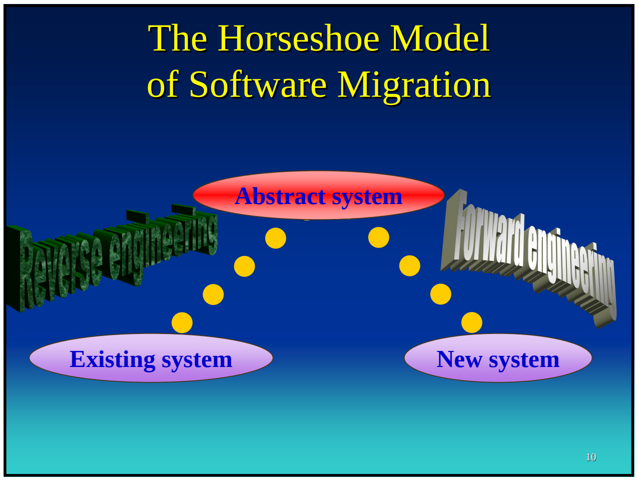
Forward engineering

- Traditional software process of moving from high-level abstractions and logical implementation-independent designs to the physical implementation of a system

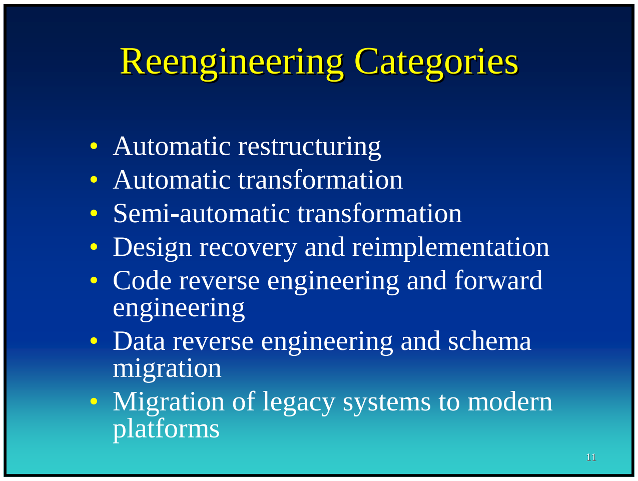


Restructuring

- Transformation from one representation to another at the same relative abstraction level, while preserving the subject's system external behavior

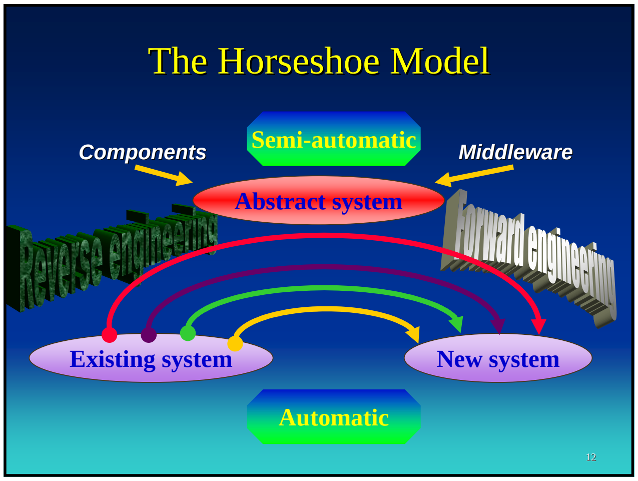


The Horseshoe Model of Software Migration

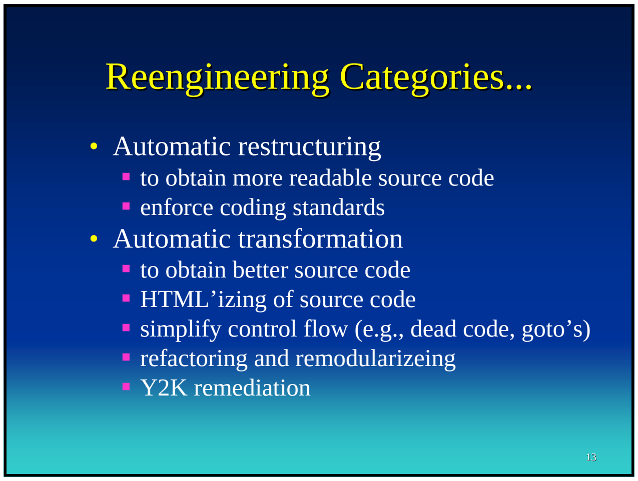


Reengineering Categories

- Automatic restructuring
- Automatic transformation
- Semi-automatic transformation
- Design recovery and reimplementation
- Code reverse engineering and forward engineering
- Data reverse engineering and schema migration
- Migration of legacy systems to modern platforms

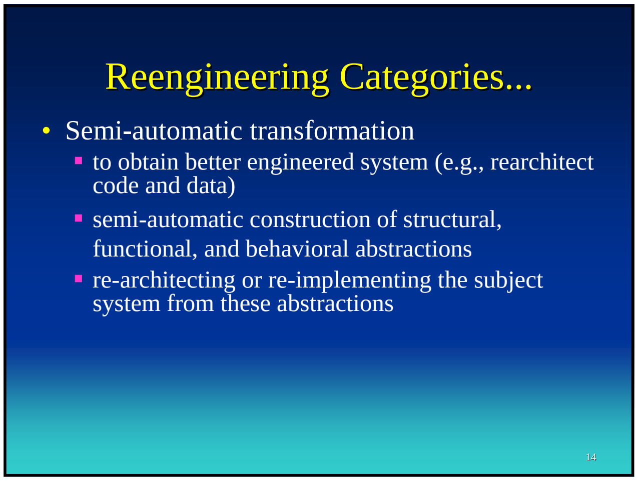


The Horseshoe Model



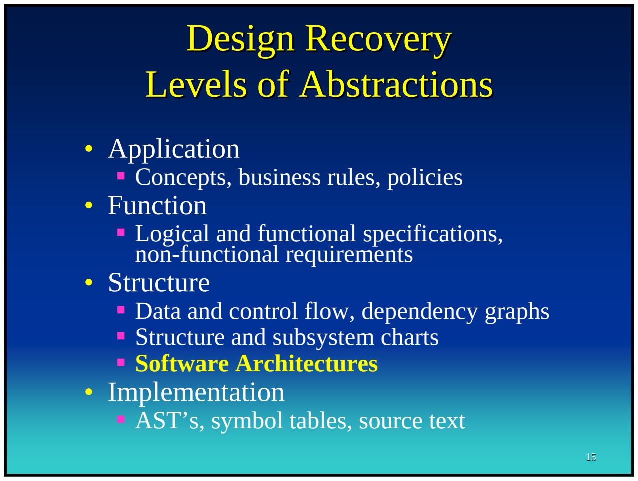
Reengineering Categories...

- Automatic restructuring
  - to obtain more readable source code
  - enforce coding standards
- Automatic transformation
  - to obtain better source code
  - HTML’izing of source code
  - simplify control flow (e.g., dead code, goto’s)
  - refactoring and remodularizeing
  - Y2K remediation



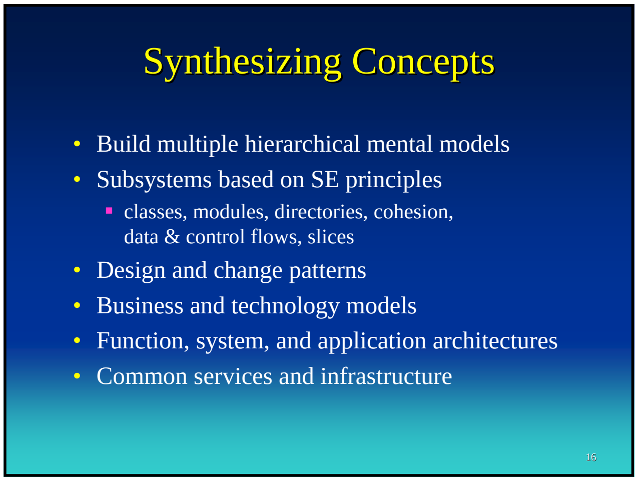
Reengineering Categories...

- Semi-automatic transformation
  - to obtain better engineered system (e.g., rearchitect code and data)
  - semi-automatic construction of structural, functional, and behavioral abstractions
  - re-architecting or re-implementing the subject system from these abstractions



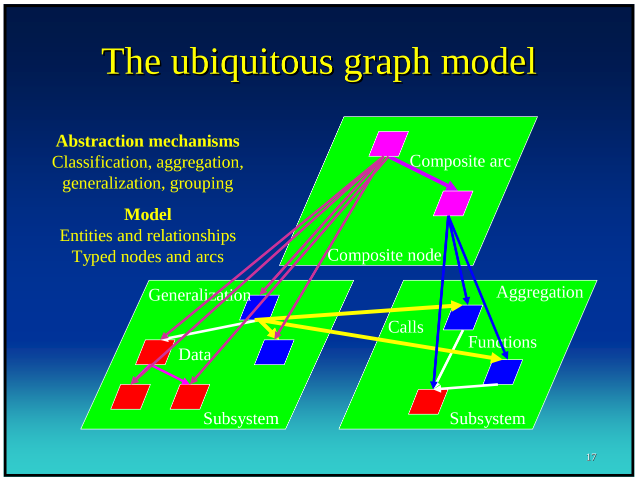
Design Recovery
Levels of Abstractions

- Application
  - Concepts, business rules, policies
- Function
  - Logical and functional specifications, non-functional requirements
- Structure
  - Data and control flow, dependency graphs
  - Structure and subsystem charts
  - **Software Architectures**
- Implementation
  - AST’s, symbol tables, source text



Synthesizing Concepts

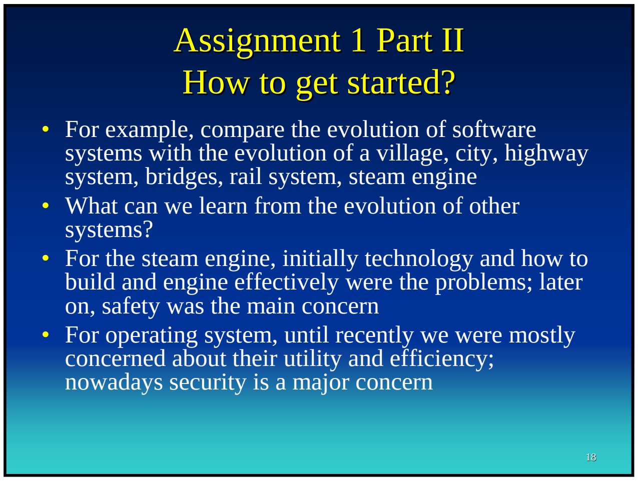
- Build multiple hierarchical mental models
- Subsystems based on SE principles
  - classes, modules, directories, cohesion, data & control flows, slices
- Design and change patterns
- Business and technology models
- Function, system, and application architectures
- Common services and infrastructure



The ubiquitous graph model

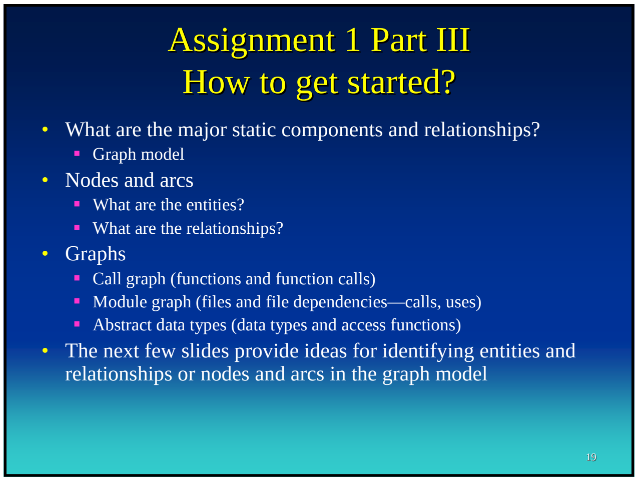
Abstraction mechanisms
Classification, aggregation,
generalization, grouping

Model
Entities and relationships
Typed nodes and arcs



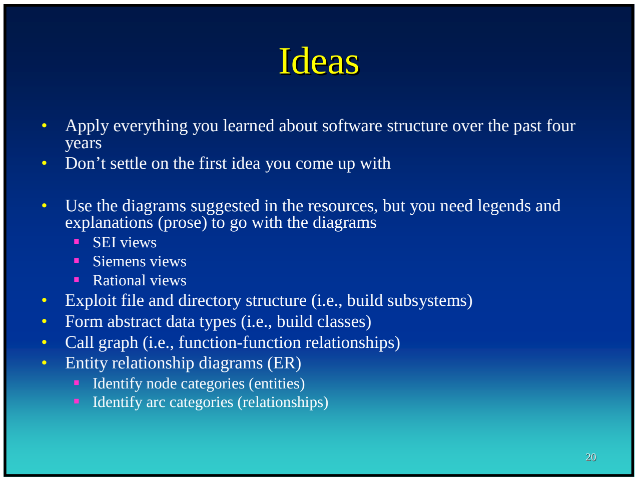
Assignment 1 Part II How to get started?

- For example, compare the evolution of software systems with the evolution of a village, city, highway system, bridges, rail system, steam engine
- What can we learn from the evolution of other systems?
- For the steam engine, initially technology and how to build and engine effectively were the problems; later on, safety was the main concern
- For operating system, until recently we were mostly concerned about their utility and efficiency; nowadays security is a major concern



Assignment 1 Part III How to get started?

- What are the major static components and relationships?
  - Graph model
- Nodes and arcs
  - What are the entities?
  - What are the relationships?
- Graphs
  - Call graph (functions and function calls)
  - Module graph (files and file dependencies—calls, uses)
  - Abstract data types (data types and access functions)
- The next few slides provide ideas for identifying entities and relationships or nodes and arcs in the graph model

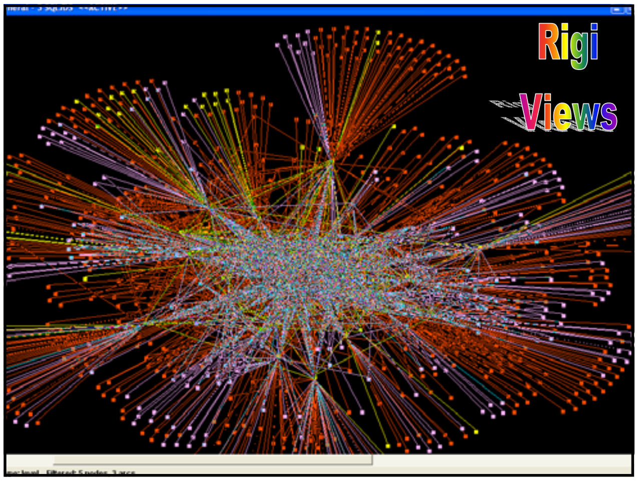


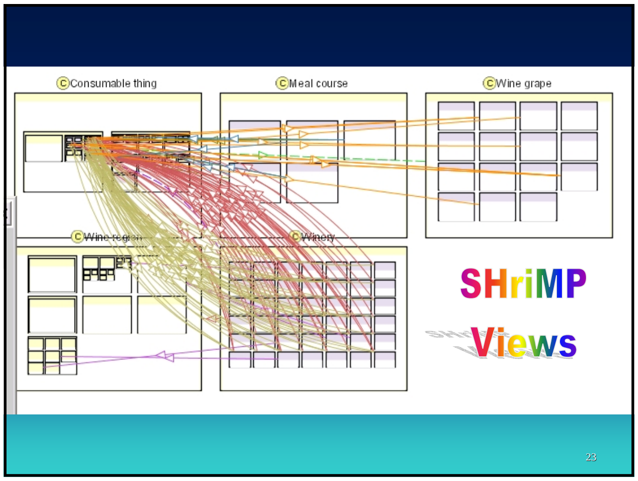
Ideas

- Apply everything you learned about software structure over the past four years
- Don’t settle on the first idea you come up with
- Use the diagrams suggested in the resources, but you need legends and explanations (prose) to go with the diagrams
  - SEI views
  - Siemens views
  - Rational views
- Exploit file and directory structure (i.e., build subsystems)
- Form abstract data types (i.e., build classes)
- Call graph (i.e., function-function relationships)
- Entity relationship diagrams (ER)
  - Identify node categories (entities)
  - Identify arc categories (relationships)

More ideas

- Form abstraction hierarchies
- Encapsulate control, data, control & data (objects)
- Summarize graphs to build hierarchies
- Compose views
  - emphasize important aspects
  - de-emphasize immaterial components
- Recognize and match design patterns
- Separate concerns
- Extract UML (class) diagrams
- Recognize and identify APIs and interfaces
- Mine configuration management system for
  - product lines
  - versions, releases
- Use visualization techniques
  - SHriMP
  - Rigi





Assignments

- Assignment 1
  - Gawk rsf posted
- Assignment 2 posted
  - Three parts
    - Definitions
    - Eclipse
    - Migration to SVG (or Avalon)