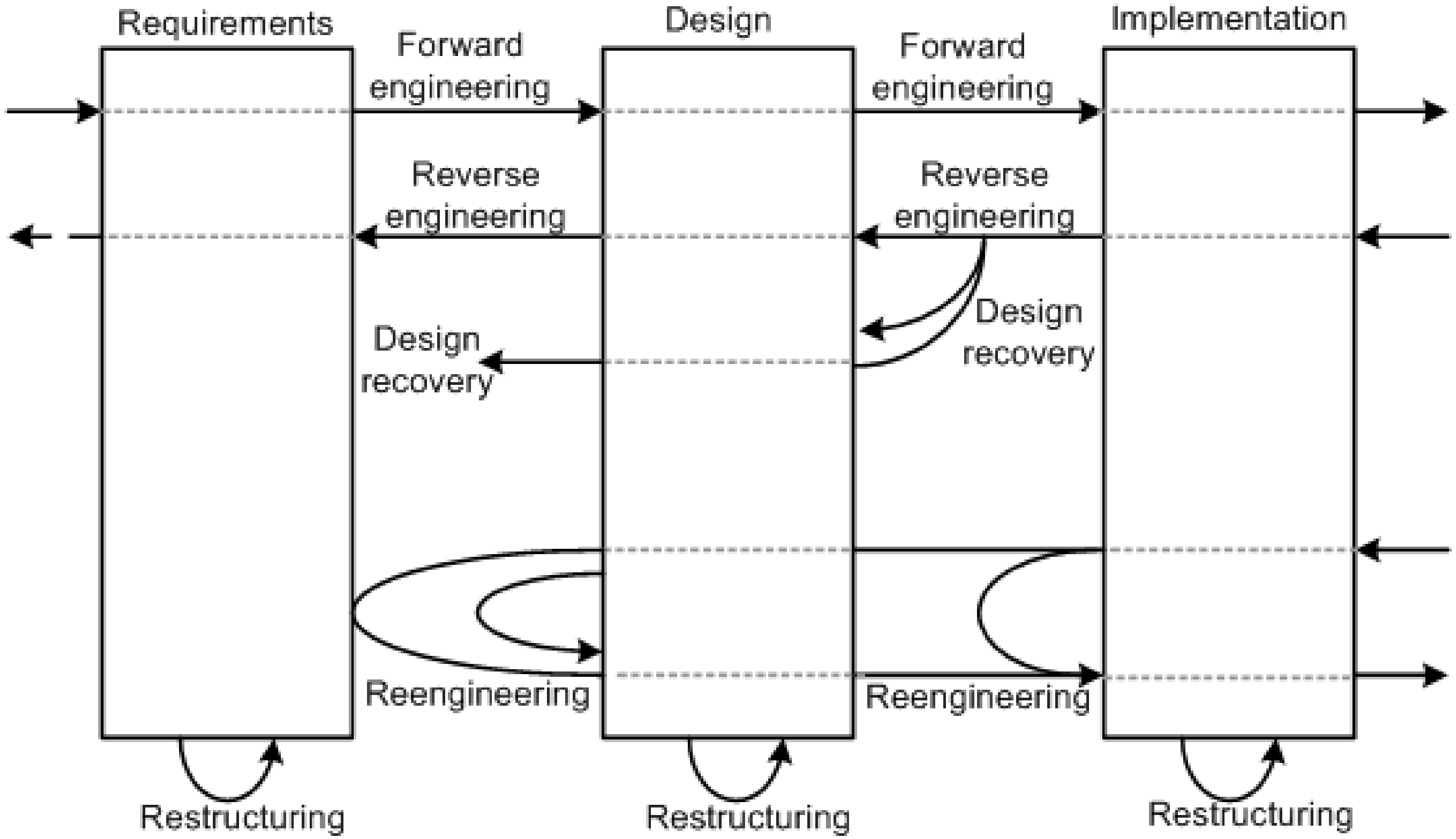
SOFTWARE RE-ENGINEERING

Unit 6

THE RELATIONSHIP BETWEEN FORWARD ENGINEERING, REENGINEERING, AND REVERSE ENGINEERING



CODE REVERSE ENGINEERING

- Six objectives of reverse engineering, as identified by Chikofsky and Cross II:
  - generating alternative views.
  - recovering lost information.
  - synthesizing higher levels of abstractions.
  - detecting side effects.
  - facilitating reuse.
  - coping with complexity.

SIX KEY STEPS IN REVERSE ENGINEERING, THE IEEE STANDARD FOR SOFTWARE MAINTENANCE

Step 1: Partition source code into units.

Step 2: Describe the meanings of those units and identify the functional units.

Step 3: Create the input and output schematics of the units identified before.

Step 4: Describe the connected units.

Step 5: Describe the system application.

Step 6: Create an internal structure of the system.

PROBLEM AREAS

- redocumenting programs
- identifying reusable assets
- discovering design architectures,
- recovering design patterns
- building traceability between code and documentation
- finding objects in procedural programs
- deriving conceptual data models
- detecting duplications and clones
- cleaning up code smells
- aspect-oriented software development
- computing change impact

- transforming binary code into source code
- redesigning user interfaces
- parallelizing largely sequential programs
- translating a program to another language
- migrating data
- extracting business rules
- wrapping legacy code
- auditing security and vulnerability
- extracting protocols of network applications

GMT PARADIGM

- A high level organizational paradigm is found to be useful while setting up a reverse engineering process, as advocated by Benedusi et al.

- The high level paradigm plays two roles:
  - (i) define a framework to use the available methods and tools, and
  - (ii) allow the process to be repetitive.

- The paradigm, namely, Goals/Models/Tools, which partitions a process for reverse engineering into three ordered stages: Goals, Models, and Tools.

GOALS

- In this phase, the reasons for setting up a process for reverse engineering are identified and analyzed.

- Analyses are performed to identify the information needs of the process and the abstractions to be created by the process.

- The team setting up the process first acquires a good understanding of the forward engineering activities and the environment where the products of the reverse engineering process will be used.

- Results of the aforementioned comprehension are used to accurately identify:
  - (i) the information to be generated.
  - (ii) the formalisms to be used to represent the information.

MODELS

- In this phase, the abstractions identified in the Goals stage are analyzed to create representation models.
- Representation models include information required for the generation of abstractions.
- Activities in this phase are:

  - identify the kinds of documents to be generated.

  - to produce those documents, identify the information and their relations to be derived from source code.

  - define the models to be used to represent the information and their relationships extracted from source code.

  - to produce the desired documents from those models, define the abstraction algorithm for reverse engineering.
- The important properties of a reverse engineering model are: expressive power, language independence, compactness, richness of information content, granularity, and support for information preserving transformation.

TOOLS

- In this phase, tools needed for reverse engineering are identified, acquired, and/or developed in-house.

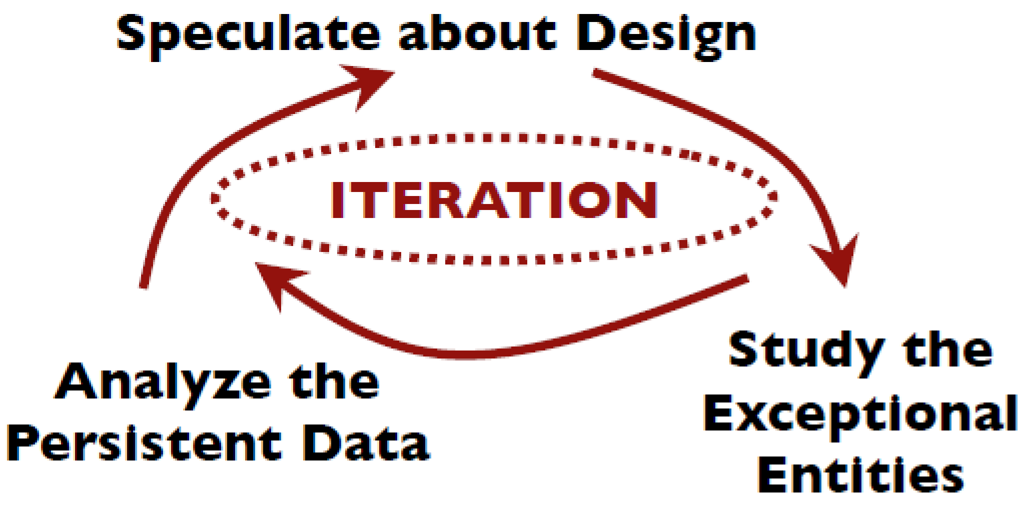
- Those tools are grouped into two categories:
  - (i) tools to extract information and generate program representations according to the identified models.
  - (ii) tools to extract information and produce the required documents. Extraction tools generally work on source code to reconstruct designdocuments.

- Therefore, those tools are ineffective in producing inputs for an abstraction process aiming to produce high-level design documents.

TECHNIQUES USED FOR REVERSE ENGINEERING

- The well-known analysis techniques that facilitate reverse engineering are:
  1. Lexical analysis.
  2. Control flow analysis.
  3. Data flow analysis.
  4. Program slicing.
  5. **Visualization.**
  6. **Program metrics**.

VISUALIZE DESIGN



Speculate about Design

ITERATION

Study the Exceptional Entities

Analyze the Persistent Data

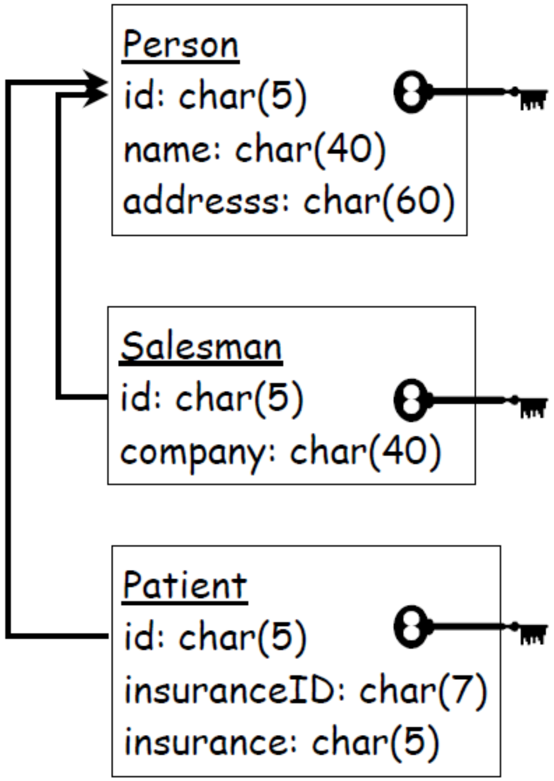
ANALYZE THE PERSISTENT DATA

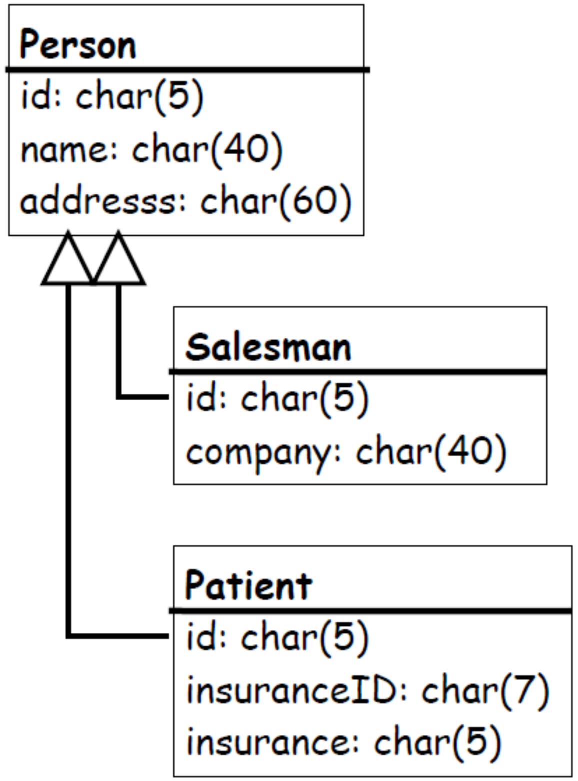
- Problem: Which objects represent valuable data?

Database	**UML**
Table	class
Columns	class attributes
Candidate keys	class associations
Naming conventions	Inheritance

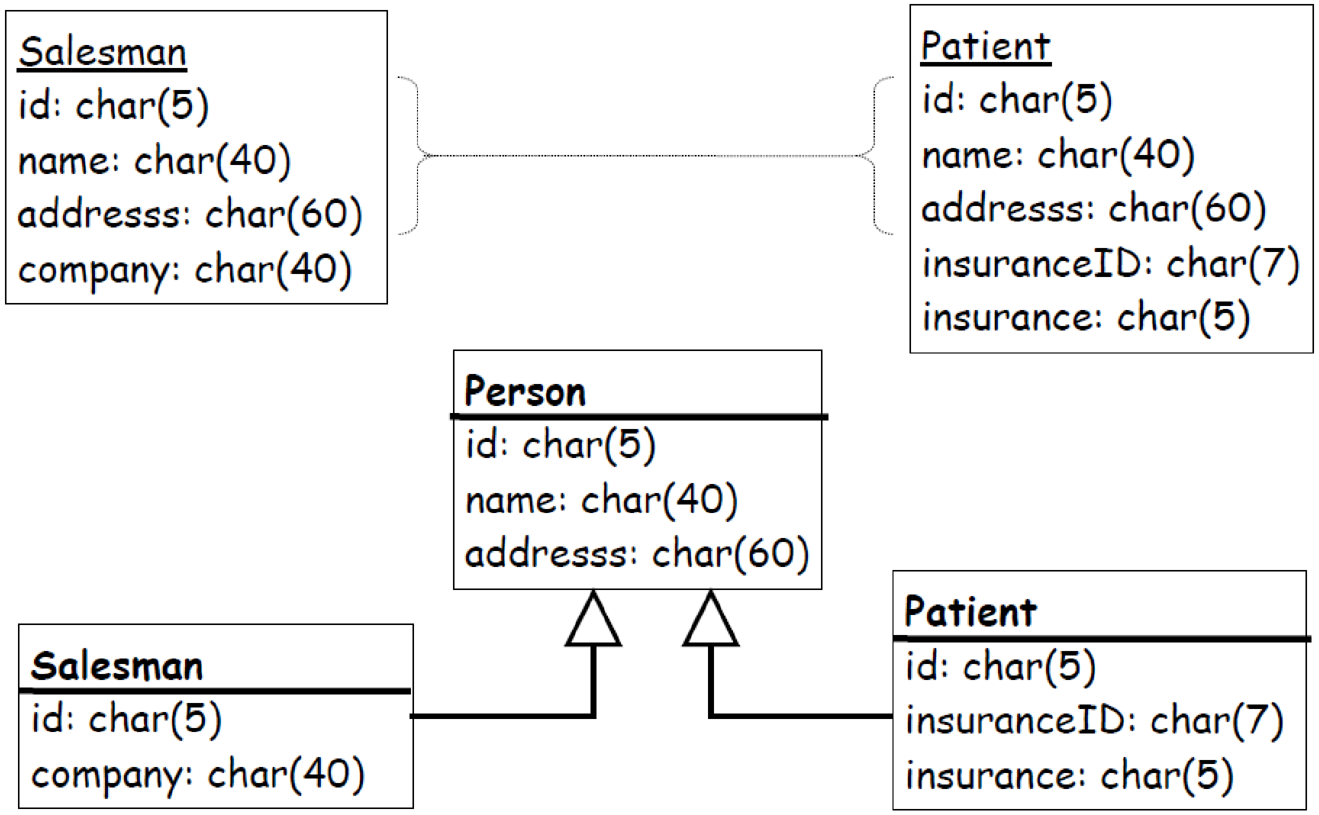
- Unique indices
- Foreign keys
- Use explicit foreign key declarations
- Infer from column types + naming conventions + view declarations + join clauses

ONE TO ONE EXAMPLE





ROLL DOWN EXAMPLE



Salesman
id: char(5)
name: char(40)
addresss: char(60)
company: char(40)

Patient
id: char(5)
name: char(40)
addresss: char(60)
insuranceID: char(7)
insurance: char(5)

Person
id: char(5)
name: char(40)
addresss: char(60)

Salesman
id: char(5)
company: char(40)

Patient
id: char(5)
insuranceID: char(7)
insurance: char(5)

SPECULATE ABOUT DESIGN

- **Problem:** How do you recover the design from source code?

- **Solution:** Develop hypotheses and check them
  - Develop a plausible class diagram and iteratively check and refine your
  - design against the actual code

- **Variants:**
  - Speculate about Business Objects
  - Speculate about Design Patterns
  - Speculate about Architecture

STUDY THE EXCEPTIONAL ENTITIES

- **Problem:** How can you quickly identify design problems?
- **Solution:** Measure software entities and study the anomalous ones
- Visualize metrics to get an overview
  - Use simple metrics
  - Lines of code
  - Number of methods
  - Length of methods

STEP THROUGH THE EXECUTION

- **Problem:** How do you uncover the run-time architecture?
  - Collaborations are spread throughout the code
  - Polymorphism may hide which classes are instantiated

- **Solution:** Execute scenarios of known use cases and step through the code with a debugger
  - Set breakpoints
  - Change internal state to test alternative paths

LOOK FOR THE CONTRACTS

- **Problem:** What does a class expect from its clients?
  - Interfaces are visible in the code but how to use them?

- **Solution:** Look for common programming idioms
  - Look for “key methods”
    - Method name, parameter types (important type -> important method)
  - Constructor calls
    - Shows which parameters to pass
  - Template/hook methods
    - Shows how to specialize a sub-class