Week 1. Introduction, Reverse Engineering Taxonomy.

What is Reverse Engineering?

- Origins
  - Hardware — deciphering designs from finished products.
  - Hardware why is it used
    - Improve your own products
    - Analyze a competitors products
- What it is.
  - **M.G. Rekoff** defines reverse engineering (in a hardware sense) as: “the process of developing a set of a specifications for a complex hardware system by an orderly examination of specimens of that system”
    - Someone other than the developer: “without the benefit of any of the original drawings … for the purpose of making a clone of the original hardware system…”
  - This is essentially software RE in hardware terms.
- *Where*/*How* is it used?
  - Software Development
  - Software Security

RE in Software Development

Reverse Engineering and Design Recovery: A Taxonomy

Eliot Chikofsky, James Cross II

- “While the hardware objective traditionally is to duplicate the system, the software objective is to gain a sufficient design-level understanding to **aid maintenance**, **strengthen enhancement**, or **support replacement**.”

- Understanding code you didn’t write
  - Understanding *compiled* code you didn’t write

Reverse Engineering

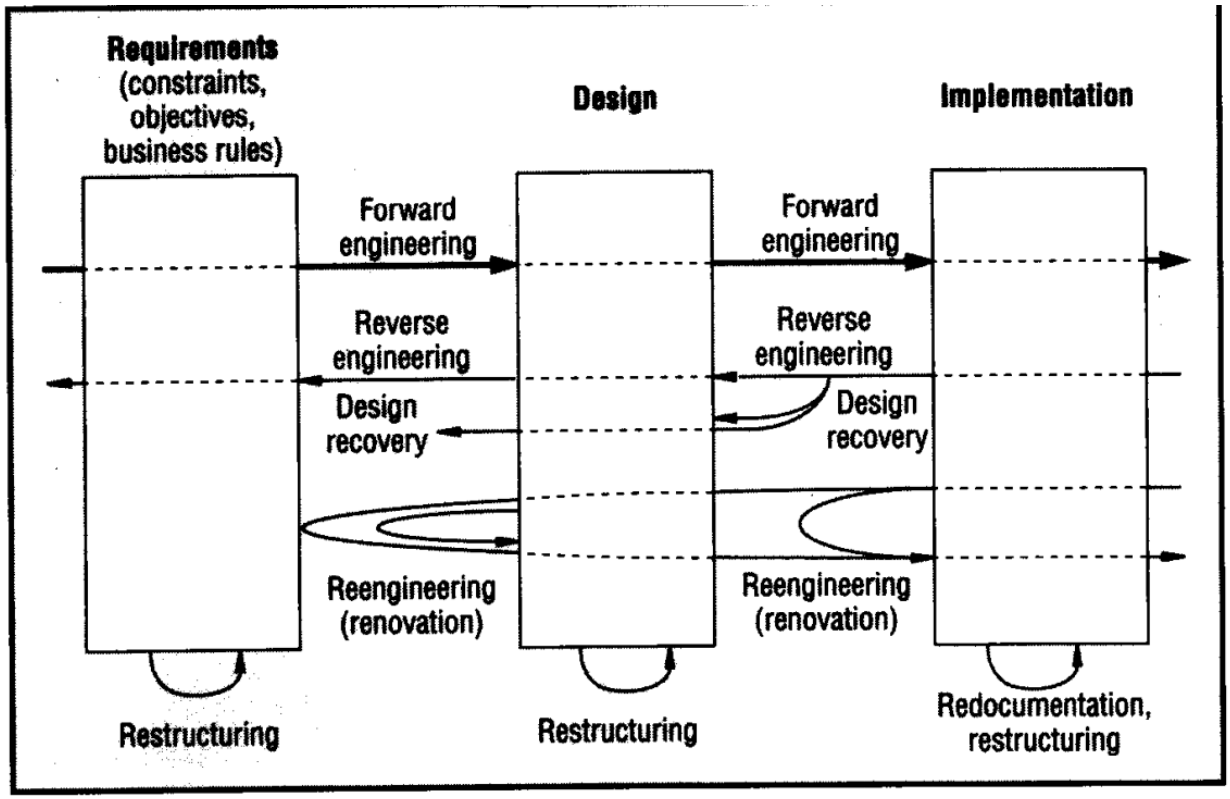


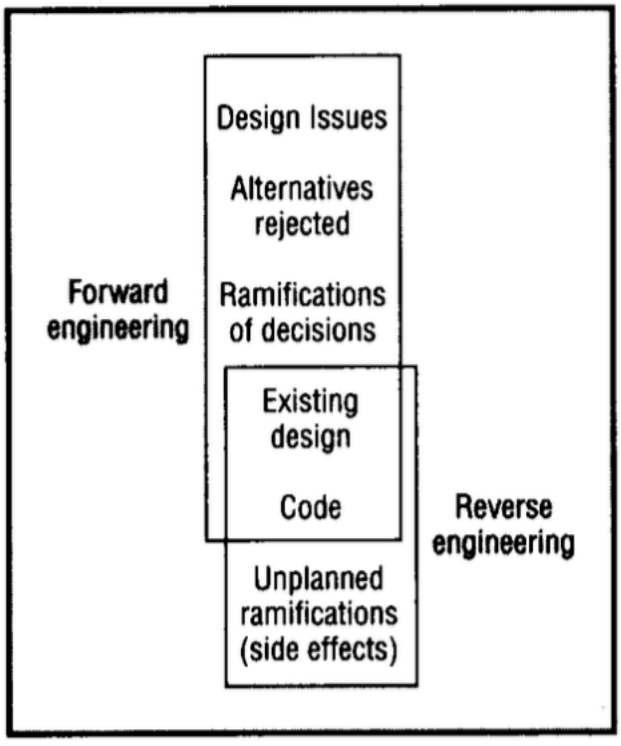
Figure 1. Relationship between terms. Reverse engineering and related processes are transformations between or within abstraction levels, represented here in terms of life-cycle phases.

- Requirements (specification of the problem being solved, including objectives, constraints, and business rules)
  - Design (specification of the solution)
  - Implementation (coding testing and delivery of system)
- **Forward Engineering**
  - Moving from high-level abstractions and logical implementation-independent designs -to-> physical implementation
  - The inverse of Reverse Engineering
- **Reverse Engineering**
  - Analyzing a system to
    - Identify the systems components and their interrelationships and

    - Create representations of the system in another form or at a higher level of abstraction
  - Synthesizing abstractions that are less implementation-dependent
  - While typically performed on an existing functional system, not a requirement. Can be done from any level of abstraction and at any stage in life cycle.
- Forms of Reverse Engineering
  - **Redocumentation**
    - Simplest and oldest form, considered to be unintrusive and weak form of **restructuring**
    - Creation or revision of a semantically equivalent representation with the same relative abstraction level
      - Creating documentation (class diagrams, functional diagrams, diagrams diagrams diagrams)
  - **Restructuring**
    - Transformation from one representation form to another at the same *relative* abstraction level, while preserving external behavior (functionality, semantics)
    - Altering code to improve structure in the traditional sense
      - Spaghetti (full of random goto everywhere up down left right) -> structured code
    - Data Normalization is an example of restructuring
    - Restructuring *can* happen with knowledge of form but not meaning
      - ie. Can convert If statements into Case statements or For to a While without knowing the programs purpose
    - May lead to better observations of the subject system that could lead to the suggestion of changes that would improve aspects of the system.
      - Should not change overall functionality of the system itself
- **Reengineering**
  - Examination and alteration of a subject system to create a new system and implement it (to get a more abstract view)

- Requires: some form of the above forms of RE followed by forward engineering
- Example
  - Reengineering of information-management systems, an organization reassesses how the system implements high level business rules and makes modifications to conform to changes in the business for the future
- While reengineering involves both forward and reverse engineering, it is not a *supertype* of the two.
  - It is not the principal driver of their progress, both techs are rapidly evolving independent of their application within reengineering

- **Why is it used?**
  - Purpose
    - Increase overall comprehensibility of the system for both maintenance and new development.
  - 6 Objectives
    - **Cope** with Complexity
      - Automated support is a key to controlling complexity.
      - Extracting relevant information from a system so decision makers can control process and product in systems evolution / automation evolution



Differences in viewpoints. RE *can help* capture lost information, some types of info are not shared between Forward and Reverse engineering. RE *can* provide observations that are unobtainable in FE.

- **Generate** alternate views (NOT TO BE USED STANDALONE)
  - Graphical representations as comprehension aids
    - Reverse engineering tools remove the ‘drag’/‘bottleneck’ that is normally associated with generating these documents. By generating (not perfect) representations from other forms (source/compiled code).
      - Control-flow diagrams
      - Structure charts
      - ER diagrams
  - RE tools can also provide documentation (non-graphical)
  - **Recover** lost information
    - This should really be a subcategory of generate alternate views, so it is.
    - Long lived systems tend to have information about modifications and perhaps even original designs lost to the passage of time.
    - again, not a substitute for preserving design history in the first place, can be useful.
- **Detect** side effects
  - See figure above. By generating designs and observing the system from its implementation instead of its design, we are able to identify unintended side effects.
- **Synthesize** higher abstractions
  - Generating things such as class diagrams and what not using CASE programs.
  - Very much still in its infancy (even ~30 years after taxonomy paper was published) it is in debate how much of this process can be automated.
- **Facilitate** reuse

    - RE can be used to identify reusable components in existing systems that could be used in future reengineered versions. ex. abstract data types (https://ieeexplore.ieee.org/document/287777)

- **Is it important?**
    - **Economics**
        - The cost of understanding software
            - **Time required** to comprehend
            - **Time lost** to misunderstanding
        - RE minimizes both, covering virtually all aspects of the SE life-cycle
    - **Is it important if you do everything perfect?**
        - Probably not.
        - Does anyone do anything perfect?
            - Maintenance is considered a cost center, not profit.
            - Documentation is often overlooked or rushed
            - Data loss can not always be 100% mitigated
                - Bad actors, etc.
        - Can it still be useful?
            - Yes
        - **Y2K**
            - This was a "*problem"* BECAUSE there weren't enough programmers who could understand / deal with legacy code. Reverse Engineering research was funded heavily during this time. Did it matter in the long run? No. Does that mean RE is useless? Also no. Just because this was a non-issue doesnt mean the problems that were identified during the Y2K problem cant be fixed or resolved with a better understanding of RE/RER.

RE in Software Security

- The majority of this course is focused around this. Brief coverage of the:

Reverse Engineering

- **What, Why, and Important?**
- **What** is it?
  - **Defensive**
  - The implementation of tools and topics covered with RE in Software Development for security purposes.
  - Understanding a system
  - Identifying flaws or bugs
  - Securing against the:
  - **Offensive**
    - Obtaining a desired output or state in a program with little to no knowledge on how to obtain that through intended means.
      - ex. You have a compiled program, if you provide the correct input, it will return a passcode or allow you to access other parts of a/the program. If you provide incorrect input, nothing happens.
        - RE techniques can be applied (dissasembling the compiled program into assembly instructions) to identify the structure and intention of the system, and you will be privy to any bug or exploit present that could help you reach your goal.
    - You analyze the logic of the program in it's normal behavior
    - Analyze which computations are done on our input
    - You analyze the logic of the program when given unexpected input
    - Analyze the behavior and see if we can exploit the new behavior
      - **https://security.stackexchange.com/questions/183656/how-do-security-experts-benefit-from-reverse-engineering**
  - **Why**
    - To Defend and Offend