Lecture 2
Software Reengineering


What is Reengineering? Reverse Engineering $\rightarrow$ Forward Engineering

Starting Point = LEGACY SYSTEM
- System expectations based strongly on usage experience
- Formal documentation not reliable
- Full software configuration not reliable

Target = Reimplementation of Legacy System
- Preserve behavior, change implementation or design
- Preserve correct behavior, repair erroneous behavior
- Preserve, report, and extend behavior
- Full software configuration of target system upon completion
  - Artifacts for each phase of life cycle
  - Traceability linking artifacts of one phase to artifacts of previous/next phase
  - Establishing traceability is HARD!

Reverse Engineering = Reconstruct Software Configuration
- Reverse engineering may encompass entire development life cycle
- Recover documentation from code or observed behavior
  - Code is only reliable artifact from development/maintenance
  - Code measurement/metrics – describes how much, complexity
  - Design documentation – provides abstraction about code components and their relationships
  - Requirements specification – abstraction about the intentions of the system, which connects design and the observed behavior
  - Traceability – documentation must show the linkage between one abstraction and those on either side: given design, where is the associated code? where is the requirements satisfied by the design.

- Dilemma of incomplete, inconsistent documentation
  - Deciding what to keep, what to discard
  - Investigation to verify accuracy of documentation
  - Temptation: throw it all away and start from scratch
Joint Application Development (JAD) techniques involving experts in areas of uncertainty are helpful.

Dilemma of no documentation
- Sometimes easier than partial/inconsistent documentation
- Move backwards through life cycle
- Decide which artifacts to reconstruct
  - What information is conveyed by the artifact?
  - How will artifact be used in forward engineering?
  - How will the artifact be used in project estimation/management?
  - How will the artifact be used to move to the next higher abstraction level?
- Automated tools required
  - To construct artifacts
  - To identify/document relationships
  - To help reason about the next higher abstraction level
  - To help reason about the next lower abstraction level.

Establish BENCHMARK for Legacy System = capture behavior/deficiencies
- Formulate requirements-based user test scenarios
  - Transform each requirement into series of test cases
    - Nominal – normal case behavior
    - Error handling – error response behavior
    - Boundary value – near extremes of inputs or capability
  - Link tests into series of user sessions
  - Convert test into one or more test scripts
  - Establish test environment (e.g., existing files)
- Run tests
  - Refine test scripts to ensure repeatability
  - Capture results
  - Evaluate results
  - Document discrepancies

Establish BASELINE for Legacy System = capture Software Configuration
- Archive the software configuration
  - Source code
  - Reverse-engineered artifacts
- Benchmarking tests
- Benchmarking results

- Archive the software configuration
  - Use repository
  - Use clear naming conventions
  - Automate traceability relations – *given* \( x \) *in* Design phase, *fetch its code* \( y \)

- Audit the software configuration
  - Confirm existence of required artifacts
  - Verify completeness, consistency, correctness of artifacts
  - Verify completion of changes to correct deficiencies

**Formalize Future Changes = set goals for target system**

- Document changes
  - Formal change process must be followed to change baseline artifact
  - Approval required to make a change
  - Verification required to ensure the only change was the approved change.

- Plan of Action
  - What to change
  - When to change
  - How to change
  - How to verify

**Fundamental Challenges/Approaches**

- Understanding others’ programs –