Lecture 21: Software Evolution

→ Basics of Software Evolution
  % Laws of software evolution
  % Requirements Growth
  % Software Aging

→ Basics of Change Management
  % Baselines, Change Requests and Configuration Management

→ Requirements Families - The product line approach

→ Requirements Traceability
  % Importance of traceability
  % Traceability tools

Program Types

→ S-type Programs ("Specifiable")
  % problem can be stated formally and completely
  % acceptance: Is the program correct according to its specification?
  % This software does not evolve.
  % change to the specification defines a new problem, hence a new program

→ P-type Programs ("Problem-solving")
  % imprecise statement of a real-world problem
  % acceptance: Is the program an acceptable solution to the problem?
  % This software is likely to evolve continuously
  % because the solution is never perfect, and can be improved
  % because the real-world changes and hence the problem changes

→ E-type Programs ("Embedded")
  % A system that becomes part of the world that it models
  % acceptance: depends entirely on opinion and judgement
  % This software is inherently evolutionary
  % changes in the software and the world affect each other

Laws of Program Evolution

→ Continuing Change
  % Any software that reflects some external reality undergoes continual change
  % or becomes progressively less useful
  % change continues until it is judged more cost effective to replace the system

→ Increasing Complexity
  % As software evolves, its complexity increases...
  % ...unless steps are taken to control it.

→ Fundamental Law of Program Evolution
  % Software evolution is self-regulating
  % ...with statistically determinable trends and invariants

→ Conservation of Organizational Stability
  % During the active life of a software system, the work output of a development project is roughly constant (regardless of resources!)

→ Conservation of Familiarity
  % The amount of change in successive releases is roughly constant
### Requirements Growth

**→ Davis’s model:**
- User needs evolve continuously
- Imagine a graph showing growth of needs over time
- May not be linear or continuous (hence no scale shown)
- Traditional development always lags behind needs growth
  - First release implements only part of the original requirements
  - Functional enhancement adds new functionality
  - Eventually, further enhancement becomes too costly, and a replacement is planned
  - The replacement also only implements part of its requirements
- And so on...

### Alternative lifecycle models

- **Throwaway Prototyping**
  - User needs
  - Functionality
  - Inappropriateness (shaded area)
  - Lateness
  - Longevity (slope of line)

- **Evolutionary Prototyping**
  - User needs
  - Functionality
  - Incremental Development

- **Automated Software Synthesis**
  - User needs
  - Functionality

### Software “maintenance”

**→ Maintenance philosophies**
- “Throw-it-over-the-wall” - someone else is responsible for maintenance
- Investment in knowledge and experience is lost
- Mission orientation - development team make a long term commitment to maintaining/enhancing the software

**→ Basili’s maintenance process models:**
- Quick-fix model
  - Changes made at the code level, as easily as possible
  - Rapidly degrades the structure of the software
- Iterative enhancement model
  - Changes made based on an analysis of the existing system
  - Attempts to control complexity and maintain good design
- Full-reuse model
  - Starts with requirements for the new system, reusing as much as possible
  - Needs a mature reuse culture to be successful

### Software Aging

- **Causes of Software Aging**
  - Failure to update the software to meet changing needs
  - Customers switch to a new product if benefits outweigh switching costs
  - Changes to software tend to reduce its coherence

- **Costs of Software Aging**
  - Owners of aging software find it hard to keep up with the marketplace
  - Deterioration in space/time performance due to deteriorating structure
  - Aging software gets more buggy
  - Each “bug fix” introduces more errors than it fixes

- **Ways of Increasing Longevity**
  - Design for change
  - Document the software carefully
  - Requirements and designs should be reviewed by those responsible for its maintenance
  - Software Rejuvenation...
Managing Requirements Change

→ Managers need to respond to requirements change
  % Add new requirements during development
  % But not succumbing to feature creep
  % Modify requirements during development
  % Because development is a learning process
  % Remove requirements during development
    → requirements "scrub" for handling cost/schedule slippage

→ Key techniques
  % Change Management Process
  % Release Planning
  % Requirements Prioritization (previous lecture)
  % Requirements Traceability
  % Architectural Stability (next week's lecture)

Change Management

→ Configuration Management
  % Each distinct product is a Configuration Item (CI)
  % Each configuration item is placed under version control
  % Control which version of each CI belongs in which build of the system

→ Baselines
  % A baseline is a stable version of a document or system
    → Safe to share among the team
  % Formal approval process for changes to be incorporated into the next baseline

→ Change Management Process
  % All proposed changes are submitted formally as change requests
  % A review board reviews these periodically and decides which to accept
    → Review board also considers interaction between change requests

Towards Software Families

→ Libraries of Reusable Components
  % Domain specific libraries (e.g., Math libraries)
  % Program development libraries (e.g., Java AWT, C libraries)

→ Domain Engineering
  % Divide software development into two parts:
    → Domain analysis - identifies generic reusable components for a problem domain
    → Application development - uses the domain components for specific applications.

→ Software Families
  % Many companies offer a range of related software systems
    → Choose a stable architecture for the software family
  % Identify variations for different members of the family
  % Represents a strategic business decision about what software to develop

requirements Traceability

→ From IEEE-STD-830:
  % Backward traceability
    → i.e., to previous stages of development
    → The origin of each requirement should be clear
  % Forward traceability
    → i.e., to all documents spawned by the SRS
    → Facilitation of referencing of each requirement in future documentation
    → Depends upon each requirement having a unique name or reference number.

→ From DOD-STD-2167A:
  % A requirements specification is traceable if:
    → "(1) it contains or implements all applicable stipulations in predecessor document
    → (2) a given term, acronym, or abbreviation means the same thing in all documents
    → (3) a given item or concept is referred to by the same name in the documents
    → (4) all material in the successor document has its basis in the predecessor document, that is, no untraceable material has been introduced
    → (5) the two documents do not contradict one another"
Importance of Traceability

- **Verification and Validation**
  - assessing adequacy of test suite
  - assessing conformity to requirements
  - assessing completeness, consistency, impact analysis
  - investigating high level behavior impact on detailed specifications
  - detecting requirements conflicts
  - checking consistency of decision making across the lifecycle

- **Process visibility**
  - ability to find information quickly in large documents

- **Management**
  - change management
  - risk management
  - control of the development process

- **Maintenance**
  - Assessing change requests
  - Tracing design rationale

- **Document access**
  - very little automated support
  - full traceability is very expensive and time-consuming

- **Delayed gratification**
  - the people defining traceability links are not the people who benefit from it
    - development vs. V&V
    - much of the benefit comes late in the lifecycle
    - testing, integration, maintenance

- **Size and diversity**
  - Huge range of different document types, tools, decisions, responsibilities, ...
  - No common schema exists for classifying and cataloging these
  - In practice, traceability concentrates only on baselined requirements

Current Practice

- **Coverage:**
  - links from requirements forward to designs, code, test cases,
  - links back from designs, code, test cases to requirements
  - links between requirements at different levels

- **Traceability process**
  - Assign each sentence or paragraph a unique id number
  - Manually identify linkages
  - Use manual tables to record linkages in a document
  - Use a traceability tool (database) for project wide traceability
  - Tool then offers ability to
    - follow links
    - find missing links
    - measure overall traceability

Limitations of Current Tools

- **Informational Problems**
  - Tools fail to track useful traceability information
    - e.g. cannot answer queries such as “who is responsible for this piece of information?”
  - Inadequate pre-requirements traceability
    - “where did this requirement come from?”

- **Lack of agreement**...
  - ...over the quantity and type of information to trace

- **Informal Communication**
  - People attach great importance to personal contact and informal communication
    - These always supplement what is recorded in a traceability database
  - But then the traceability database only tells part of the story!
    - Even so, finding the appropriate people is a significant problem

Traceability Difficulties

- **Cost**
  - very little automated support
  - Full traceability is very expensive and time-consuming

- **Delayed gratification**
  - the people defining traceability links are not the people who benefit from it
    - development vs. V&V
  - Much of the benefit comes late in the lifecycle
    - testing, integration, maintenance

- **Size and diversity**
  - Huge range of different document types, tools, decisions, responsibilities, ...
  - No common schema exists for classifying and cataloging these
  - In practice, traceability concentrates only on baselined requirements

Problematic Questions

→ Involvement
  % Who has been involved in the production of this requirement and how?

→ Responsibility & Remit
  % Who is responsible for this requirement?
    > who is currently responsible for it?
    > at what points in its life has this responsibility changed hands?
  % Within which group's remit are decisions about this requirement?

→ Change
  % At what points in the life of this requirement has working arrangements of all involved been changed?

→ Notification
  % Who needs to be involved in, or informed of, any changes proposed to this requirement?

→ Loss of knowledge
  % What are the ramifications regarding the loss of project knowledge if a specific individual or group leaves?