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

Software 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
- A 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**
- **Evolutionary Prototyping**
- **Incremental Development**
- **Automated Software Synthesis**

Software “maintenance”

**Maintenance philosophies**
- “Throw-it-over-the-wall” – someone else is responsible for maintenance
  - Investment in knowledge and experience is lost
  - Maintenance becomes a reverse engineering challenge
- “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
  - 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
  - Divides 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
    - Vertical families
      - E.g., 'basic,' 'deluxe' and 'pro' versions of a system
    - Horizontal families
      - Similar systems used in related domains

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:
    - It contains or implements all applicable stipulations in predecessor document
    - A given term, acronym, or abbreviation means the same thing in all documents
    - A given item or concept is referred to by the same name in the documents
    - All material in the successor document has its basis in the predecessor document
      - That is, no untraceable material has been introduced
    - The two documents do not contradict one another
Importance of Traceability

- **Verification and Validation**
  - assessing adequacy of test suite
  - assessing conformance to requirements
  - assessing completeness, consistency, impact analysis
  - assessing over- and under-design
  - investigating high level behavior impact on detailed specifications
  - detecting requirements conflicts
  - checking consistency of decision making across the lifecycle
- **Maintenance**
  - Assessing change requests
  - Tracing design rationale

Document access
- ability to find information quickly in large documents

Process visibility
- ability to see how the software was developed
- provides an audit trail

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

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

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
<table>
<thead>
<tr>
<th>Problematic Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Involvement</strong></td>
</tr>
<tr>
<td>¿ Who has been involved in the production of this requirement and how?</td>
</tr>
<tr>
<td><strong>Responsibility &amp; Remit</strong></td>
</tr>
<tr>
<td>¿ Who is responsible for this requirement?</td>
</tr>
<tr>
<td>¿ Who is currently responsible for it?</td>
</tr>
<tr>
<td>¿ At what points in its life has this responsibility changed hands?</td>
</tr>
<tr>
<td>¿ Within which group's remit are decisions about this requirement?</td>
</tr>
<tr>
<td><strong>Change</strong></td>
</tr>
<tr>
<td>¿ At what points in the life of this requirement has working arrangements of all involved been changed?</td>
</tr>
<tr>
<td><strong>Notification</strong></td>
</tr>
<tr>
<td>¿ Who needs to be involved in, or informed of, any changes proposed to this requirement?</td>
</tr>
<tr>
<td><strong>Loss of knowledge</strong></td>
</tr>
<tr>
<td>¿ What are the ramifications regarding the loss of project knowledge if a specific individual or group leaves?</td>
</tr>
</tbody>
</table>