Lecture 11: Evolving Requirements

Last Week:
Specification & Validation
Document Standards
Inspections & Prototyping
Prioritization

This Week:
Evolving Requirements
Change management
Product Families
Traceability
Inconsistency management

Next Week:
How much Formality?
Appropriate use of
Formal methods in RE

Outline

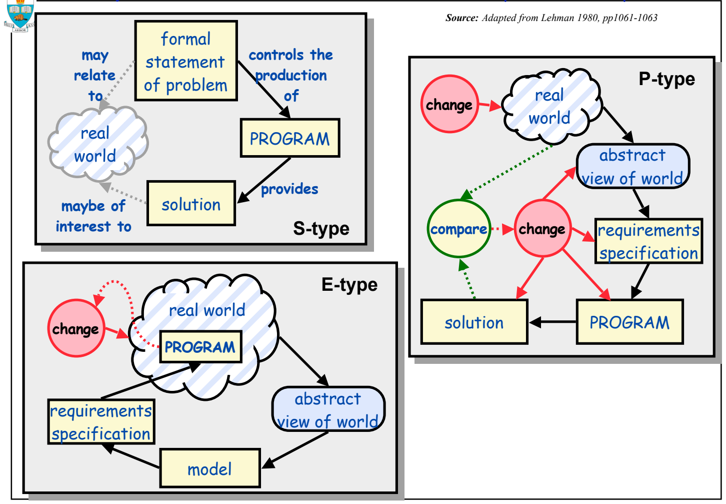
- **Basics of Software Evolution**
  - Laws of software evolution
  - Baselines, Change Requests and Configuration Management
  - Beyond specification singularity
  - Software Families - The product line approach
- **Requirements Traceability**
  - Importance of traceability
  - Traceability tools
  - Contribution structures
- **Inconsistency Management**
  - Basics of viewpoints
  - Expressing consistency rules
  - Reasoning in the presence of inconsistency

Program Types

Source: Adapted from Lehman 1980, pp1061-1063

- **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

Source: Adapted from Lehman 1980, pp1061-1063



Laws of Program Evolution

Source: *Adapted from Lehman 1980, pp1061-1063*

→ 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

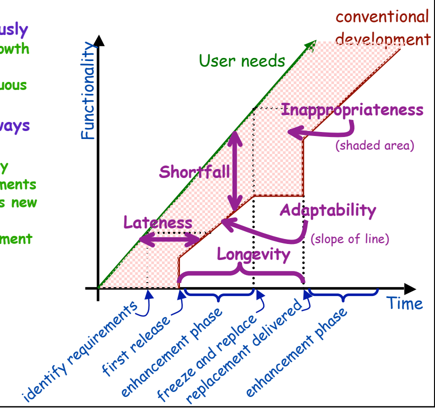
- The amount of change in successive releases is roughly constant

Requirements Growth

Source: *Adapted from Davis 1988, pp1453-1455*

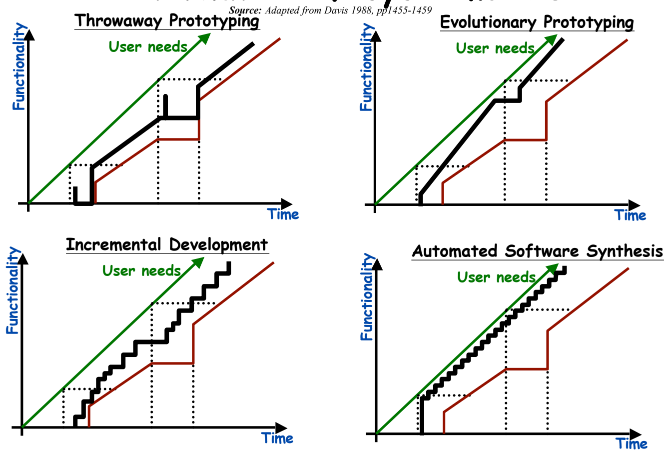
→ Davis's model:

- 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

Source: *Adapted from Davis 1988, pp1455-1459*



Software "maintenance"

Source: *Adapted from Blum, 1992, p492-495*

→ 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

Traditional Change Management

→ 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

→ Elements of Change Management

- Configuration Items
  - Each distinct product during development is a configuration item
  - version control of each item
  - control which version of each item belongs in which build of the system
- Baselines
  - A baseline is a stable version of a document that can be shared 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 change requests periodically and decides which to accept
  - Review board considers interaction between change requests

Beyond "Product Singularity"

→ Most RE techniques focus on individual models

- "Build a model, get it consistent and complete, then validate it"
- Assumes that RE is a process with a single definite output
  - The output is a complete, consistent, valid specification of the requirements.

→ This ignores reality!

- Requirements Engineering isn't just about obtaining a specification
  - Requirements are volatile; changes need to be managed continuously
  - The specification is never complete anyway!
- There is never just one model:
  - There are multiple versions of models over time
  - There are multiple variants of models that explore different issues
  - There are multiple components of models representing different decompositions
  - Families of models evolve over time (add, delete, merge, restructure the family)
- RE must address requirements evolution
  - How do we manage incremental change to requirements models?
  - How can multiple models (specifications) be compared?
  - How will changes to a model affect the properties established for it?
  - How do you capture the rationale for each change?
  - How do we reason about inconsistent and incomplete models?

Towards Software Families

→ Software reuse aims to cut costs

- Developing software is expensive, so aim to reuse for related systems
  - Successful approaches focus on reusing knowledge and experience rather than just software products
  - Economics of reuse are complex as it costs more to develop reusable software

→ 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

Requirements Traceability

→ Definition (DOD-STD-2167A):

"(1) The document in question contains or implements all applicable stipulations in the 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 or description 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"

→ In short:

- A demonstration of completeness, necessity and consistency
- a clear allocation/flowdown path (down through the document hierarchy)
- a clear derivation path (up through the document hierarchy)

Source: *Adapted from Palmer, 1996, p 367*

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

Traceability Tools

- Approaches:
  - hypertext linking
    - hotwords are identified manually, tool records them
  - unique identifiers
    - each requirement gets a unique id; database contains cross references
  - syntactic similarity coefficients
    - searches for occurrence of patterns of words
- Limitations
  - All require a great deal of manual effort to define the links
  - All rely on purely syntactic information, with no semantics or context
- Examples
  - single phase tools:
    - TeamWork (Cadre) for structured analysis
  - database tools, with queries and report generation
    - RTM (Marconi)
    - SLATE (TD Technologies)
    - DOORS (Zycad Corp)
  - hypertext-based tools
    - Document Director
    - Any web browser
  - general development tools that provide traceability
    - RDD-100 (Ascent Logic) - documents system conceptual models
    - Foresight - maintains data dictionary and document management

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

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 requirements 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?

Contribution Structures

- **'author' attribute too weak**
  - does not adequately capture ownership of information
  - refers to person that wrote the document rather than the person who originated the content
  - fail to capture situations where many people participate
  - fail to capture changing patterns of participation
- **Contribution structures**
  - link requirements artifacts (contributions) to agents (contributors) via contribution relations
- **Roles**
  - Principal
    - who motivated the artefact (responsible for consequences)
  - Author
    - who chose the structure and content (responsible for semantics)
  - Documentor
    - who recorded/transcribed the content (responsible for appearance)

Conflict Resolution - basics

- **Defining Conflict**
  - In Social psychology, focus is on interdependence and perception:
    - "the interaction of interdependent people who perceive opposition of goals, aims, and values, and who see the other party as potentially interfering with the realization of these goals" [Putnam & Poole, 1987]
  - In RE, focus typically is on logical inconsistency:
    - E.g. conflict is a divergence between goals - there is a feasible boundary condition that makes the goals inconsistent [van Lamsweerde et al. 1998]
  - Note:
    - conflict may occur between individuals, groups, organizations, or different roles played by one person
- **Resolution Method:**
  - The approach used to settle a conflict
    - Methods include negotiation, competition, arbitration, coercion, and education
    - Not all conflicts need a resolution method: not all conflicts need to be resolved.
  - Three broad types of resolution method can be distinguished:
    - Co-operative (or collaborative) methods, which include negotiation and education;
    - Competitive methods, which include combat, coercion and competition;
    - Third Party methods, which include arbitration and appeals to authority.

Basic approaches to conflict resolution

- **Negotiation**
  - …is collaborative exploration:
    - participants attempt to find a settlement that satisfies all parties as much as possible.
  - also known as:
    - integrative behaviour
    - constructive negotiation
  - distinct from:
    - distributive/competitive negotiation
- **Competition**
  - is maximizing your own gain:
    - no regard for the degree of satisfaction of other parties.
    - but not necessarily hostile!
  - Extreme form:
    - when all gains by one party are at the expense of others
    - I.e a zero-sum game.
- **Third Party Resolution**
  - participants appeal to outside source
    - the rule-book, a figure of authority, or the toss of a coin.
    - can occur with the breakdown of either negotiation or competition as resolution methods.
  - types of third party resolution
    - judicial: cases presented by each participant are taken into account
    - extra-judicial: a decision is determined by factors other than the cases presented (e.g. relative status of participants).
    - arbitrary: e.g. toss of a coin
- **Bidding and Bargaining**
  - Bidding:
    - participants state their desired terms
  - Bargaining:
    - participants search for a satisfactory integration of bids.

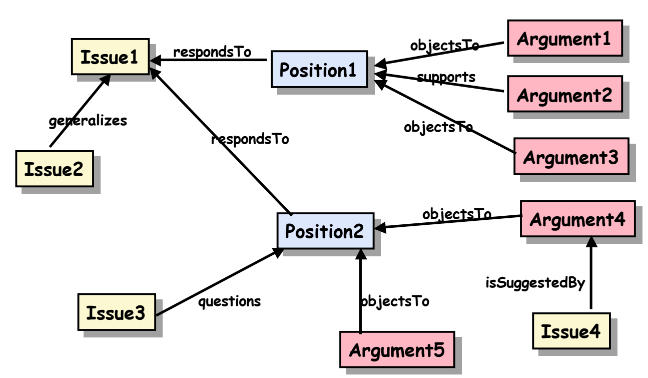
Conflict in Social Psychology

- **Causes of Conflict**
  - Deutsch (1973):
    - control over resources
    - preferences and nuisances (tastes or activities of one party impinge upon another)
    - values (a claim that a value or set of values should dominate)
    - beliefs (dispute over facts, information, reality, etc.)
    - the nature of the relationship between the parties.
  - Robbins (1989):
    - communicational (insufficient exchange of information, noise, selective perception)
    - structural (goal compatibility, jurisdictional clarity, leadership style)
    - personal factors, (individual value systems, personality characteristics.
- **Interesting Results**
  - deviant behaviour & conflict are normal in small group decision making
  - more aggression and less co-operation when communication is restricted
    - a decrease in communication tends to intensify a conflict (the contact hypothesis)
  - heterogeneous teams experience more conflict;
  - homogeneous groups are more likely to make high risk decisions (groupthink)
  - effect of personality is overshadowed by situational and perceptual factors

Using Argumentation Structuring…

- **gIBIS**
  - developed by Conklin [1989]
  - Represents argumentatation process as a hypertextual graph
  - Basic Process
    - Identify issues
    - Identify positions that one can adopt with respect to the positions
    - link arguments that support or refute positions
- **Synoptic**
  - Developed by Easterbrook [1991]
  - Tool support for collaborative task-focussed negotiation
  - Basic Process:
    - Get each participant to externalise their conceptual model(s)
    - Find correspondences between the models
    - Classify mismatches
    - Generate options for resolving each mismatch

gIBIS argumentation structure



Using Pre-existing Domain Models…

→ Oz

- developed by Robinson [1992]
- Uses pre-existing domain model to compare conflicting perspectives
- Basic process:
  - Identify perspectives (collections of beliefs)
  - Record perspectives by annotating a domain model of goals and objectives
  - Domain model links product attributes to goals
  - Choose combinations of product attributes to maximise participants' satisfaction

→ WinWin

- developed by Boehm & colleagues [mid 1990s]
- explicitly identifies win-conditions for each participant
- Incorporates domain knowledge-base of quality requirements and product attribute links
- Basic Process:
  - Enter win conditions for each participant
  - identify attribute strategies for win conditions
  - determine negative effects for each strategy on each win condition
  - resolve disagreements manually

Viewpoints - Motivations

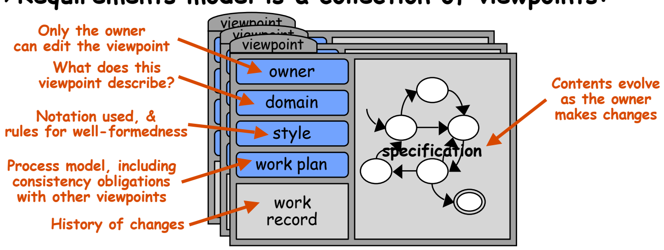
Multiple Perspectives	Distributed Modeling
Many different stakeholders	Collaborating analysts & stakeholders
Diverse kinds of Domain Knowledge	Multiple modeling methods
Conflicting views (& negotiation)	Continuous evolution of requirements
Many representation schemes	Imperfect communication links

→ Delaying Resolution of Inconsistency

- Inconsistency caused by:
  - Conflict between knowledge sources
  - Different interpretations
  - Communication problems between developers
  - Different development speeds
  - Divergence from prescribed method
  - Mistakes
- Single model with consistency enforcement is too restrictive
  - Single model becomes a bottleneck for distributed modeling process
  - Consistency enforcement prevents entry of divergent/tentative ideas
- Inconsistencies generally arise where there is the most uncertainty
  - Premature resolution may entail premature design decisions
  - Inconsistency implies more knowledge acquisition needed!
  - *More radically:* Some inconsistencies never get fixed…

The basic framework

→ Requirements model is a collection of viewpoints:



- Viewpoints are instantiated from viewpoint templates
  - Template only has style and work plan slots filled
  - Development of templates is a separate "method engineering" task
  - A method provides a set of templates designed to be used together
- Viewpoints contain consistency rules (no central control)
  - Internal consistency rules for checking a viewpoint's specification
  - External consistency rules for inter-viewpoint checks
  - Work plan provides guidance for when to apply each consistency rule

Advantages of the approach

→ Stakeholder buy-in and Traceability

- Viewpoint owners can be roles, people, teams,…
- Each stakeholder's contribution is modeled in an appropriate notation
  - Stakeholders can identify and validate their own contributions
  - Increases stakeholder 'ownership' of the requirements process
- Requirements can be traced back to a source/authority

→ Structuring the development process

- Each viewpoint is an independent 'workpiece'
  - viewpoints as a distributed, loosely-coupled, suite of development tools
- No global control, no global enforcement of consistency
  - supports synchronous and asynchronous working
  - consistency checking rules act as explicit re-synchronization points

→ Structuring the descriptions

- Different stakeholders' contributions are modeled separately
  - Separation of concerns
  - Richer models through the use of multiple problem structures
- Resolution of inconsistency can be delayed
  - Supports negotiation by allowing detailed comparison of viewpoints
  - Encourages early modeling and expression of divergent views

Inconsistency Management

- **Inconsistency arises from:**
  - Conflict between knowledge sources
  - Different interpretations
  - Communication problems between developers
  - Different development speeds
  - Divergence from prescribed method
  - Mistakes
- **Definition of inconsistency**
  - "two parts of a specification do not obey some relationship that should hold between them". (Easterbrook & Nuseibeh, 1995)
  - Relationships may link
    - syntactic elements of partial specifications;
    - semantics of elements in partial specifications;
    - sub-processes of the overall development process.
  - Relationships arise from:
    - definition of the method;
    - practical experience with the method;
    - local contingencies during development.

Example Consistency Rules

- **E.g 1: in structured analysis:**
  - In a data flow diagram, if a process is decomposed in a separate diagram, then the input flows into the parent process must be the same as the input flows into child data flow diagram.
- **E.g. 2: Use of domain concepts:**
  - For a particular Library System, the concept of operations document states that "User" and "Borrower" are synonyms. Hence, the list of user actions described in the help manuals must correspond to the list of borrower actions in the requirements specification.
- **E.g. 3: Process rules:**
  - Coding should not begin until the Systems Requirement Specification has been signed off by the Project Review Board (PRB). Hence, the program code repository should be empty until the SRS has the status 'approved by PRB'.

Lessons about inconsistency in practice

- **some inconsistencies never get fixed**
  - because the cost of changing the documentation outweighs the benefit
  - humans are good at inventing workarounds
- **living with inconsistency is a risky decision**
  - risk factors change, so the risk must be constantly re-evaluated
- **some consistency checks are not worth performing**
  - waste of money to establish consistency where change is anticipated
  - … also where documents are early drafts, or are full of known errors
- **inconsistency is deniable**
  - e.g. because of face saving and defensiveness - inconsistency seen as bad!
  - e.g. because you can always question the formalization!