



Lecture 11, Part 1: Requirements Prioritization

- ⇒ **Why Prioritization is needed**
 - ↳ Basic Trade-offs
- ⇒ **Cost-Value Approach**
 - ↳ Sorting Requirements by cost/value
 - ↳ Estimating Relative Costs/Values using AHP
- ⇒ **What if stakeholders disagree?**
 - ↳ Visualizing differences in priority
 - ↳ Resolving Disagreements



Basics of Prioritization

- ⇒ **Need to select what to implement**
 - ↳ Customers (usually) ask for way too much
 - ↳ Balance time-to-market with amount of functionality
 - ↳ Decide which features go into the next release
- ⇒ **For each requirement/feature, ask:**
 - ↳ How important is this to the customer?
 - ↳ How much will it cost to implement?
 - ↳ How risky will it be to attempt to build it?
- ⇒ **Perform Triage:**
 - ↳ Some requirements **must** be included
 - ↳ Some requirements should definitely be excluded
 - ↳ That leaves a pool of "nice-to-haves", which we must select from.

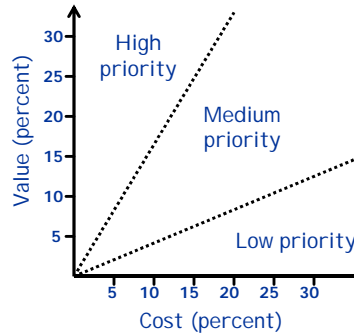


A Cost-Value Approach

Source: Adapted from Karlsson & Ryan 1997

⇒ Calculate return on investment

- ↳ Assess each requirement's importance to the project as a whole
- ↳ Assess the relative cost of each requirement
- ↳ Compute the cost-value trade-off:



Estimating Cost & Value

⇒ Two approaches:

- ↳ Absolute scale (e.g. dollar values)
 - > Requires much domain experience
- ↳ Relative values (e.g. less/more; a little, somewhat, very)
 - > Much easier to elicit
 - > Prioritization becomes a sorting problem

⇒ Comparison Process - options

- ↳ Basic sorting - for every pair of requirements (i,j) , ask if $i > j$?
 - > E.g. bubblesort - start in random order, and swap each pair if out of order
 - > requires $n^2(n-1)/2$ comparisons
- ↳ Construct a Binary Sort Tree
 - > Requires $O(n \log n)$ comparisons
- ↳ Construct a Minimal Spanning Tree
 - > for each pair (R_i, R_{i+1}) get the distance between them
 - > Requires $n-1$ comparisons



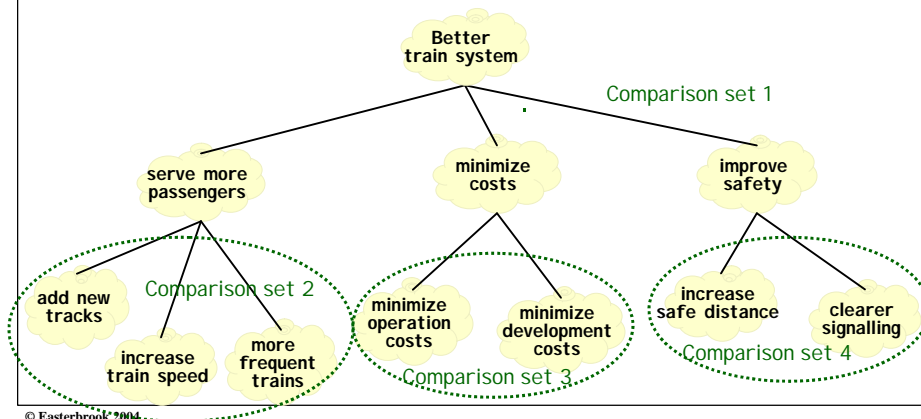
Some complications

- ⇒ Hard to *quantify* differences
 - ↳ easier to say "x is more important than y"...
 - ↳ ...than to estimate by how much.
- ⇒ Not all requirements comparable
 - ↳ E.g. different level of abstraction
 - ↳ E.g. core functionality vs. customer enhancements
- ⇒ Requirements may not be independent
 - ↳ No point selecting between X and Y if they are mutually dependent
- ⇒ Stakeholders may not be consistent
 - ↳ E.g. If $X > Y$, and $Y > Z$, then presumably $X > Z$?
- ⇒ Stakeholders might not agree
 - ↳ Different cost/value assessments for different types of stakeholder



Hierarchical Prioritization

- ⇒ Group Requirements into a hierarchy
 - ↳ E.g. A goal tree
 - ↳ E.g. A NFR tree
- ⇒ Only make comparisons between branches of a single node:





Analytic Hierarchy Process (AHP)

Source: Adapted from Karlsson & Ryan 1997

⇒ Create n x n matrix (for n requirements)

- ↳ For element (x,y) in the matrix enter:
 - > 1 - if x and y are of equal value
 - > 3 - if x is slightly more preferred than y
 - > 5 - if x is strongly more preferred than y
 - > 7 - if x is very strongly more preferred than y
 - > 9 - if x is extremely more preferred than y
 - > (use the intermediate values, 2,4,6,8 if compromise needed)
- ↳ ...and for (y,x) enter the reciprocal.

⇒ Estimate the eigenvalues:

- ↳ E.g. "averaging over normalized columns"
 - > Calculate the sum of each column
 - > Divide each element in the matrix by the sum of it's column
 - > Calculate the sum of each row
 - > Divide each row sum by the number of rows

⇒ This gives a value for each reqt:

- ↳ ...giving the estimated percentage of total value of the project



AHP example - estimating costs

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

Normalise columns

Req1 - 26% of the cost
 Req2 - 50% of the cost
 Req3 - 9% of the cost
 Req4 - 16% of the cost

Result

	Req1	Req2	Req3	Req4
Req1	0.21	0.18	0.18	0.48
Req2	0.63	0.54	0.45	0.36
Req3	0.11	0.11	0.09	0.04
Req4	0.05	0.18	0.27	0.12

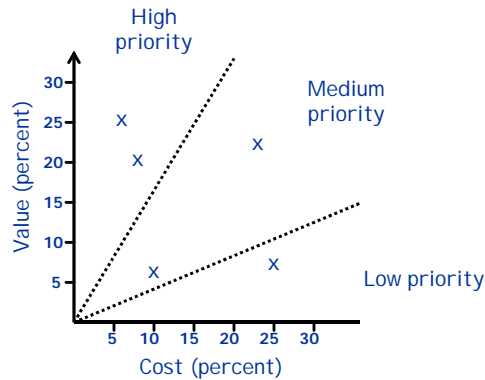
Sum the rows

sum	sum/4
1.05	0.26
1.98	0.50
0.34	0.09
0.62	0.16



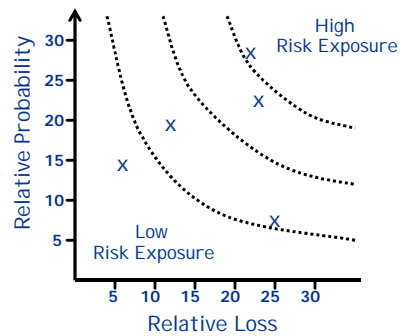
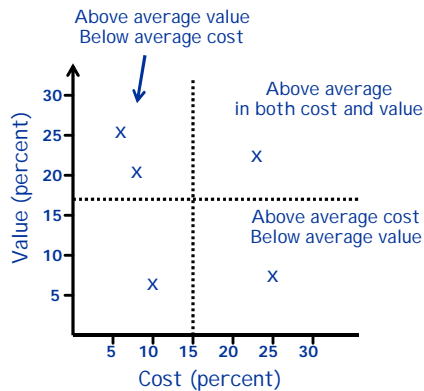
Plot ROI graph

- ⇒ Repeat AHP process twice:
 - ↳ Once to estimate relative value
 - ↳ Once to estimate relative cost
- ⇒ Use results to calculate ROI ratio:



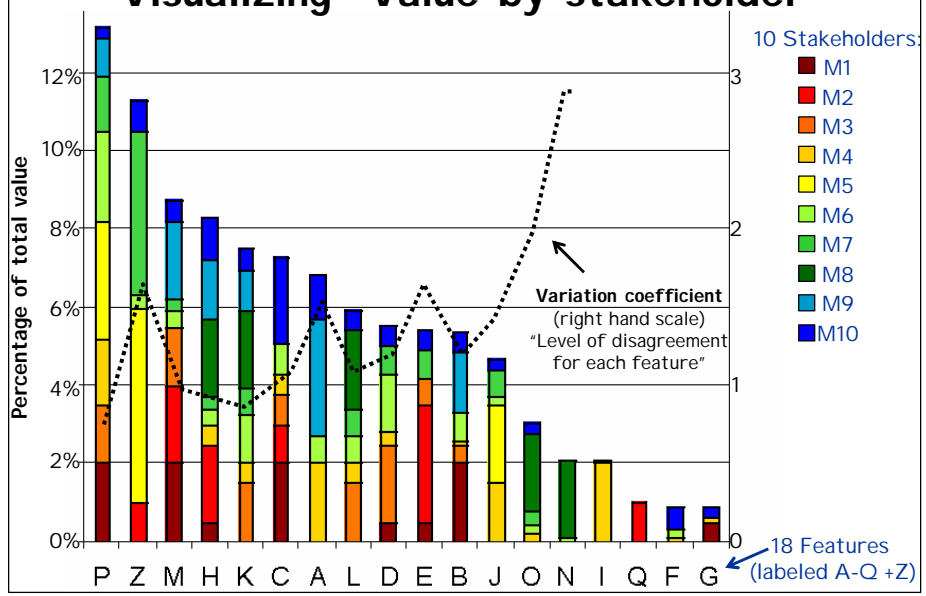
Other selection criteria

- ⇒ ROI ratio is not the only way to group requirements





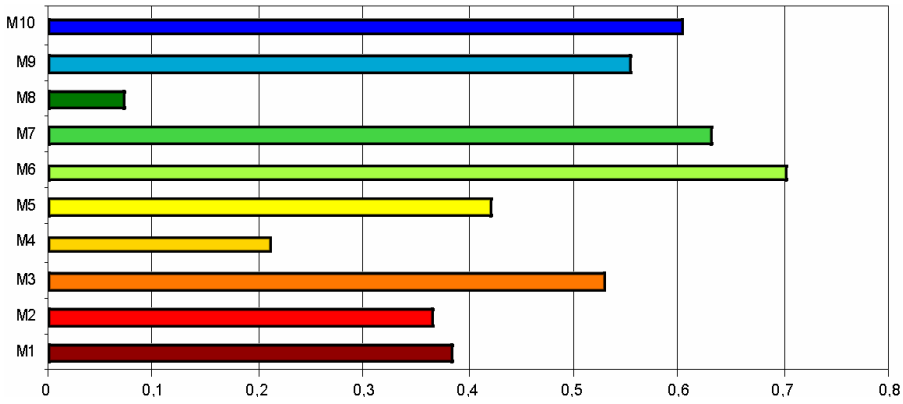
Visualizing "Value by stakeholder"



Visualizing stakeholder satisfaction

Graph showing correlation between stakeholder's priorities and the group's priorities

Can also be thought of as "influence of each stakeholder on the group"



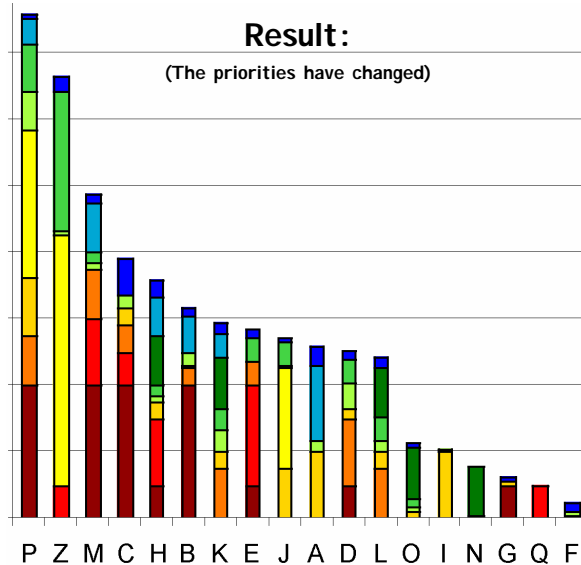
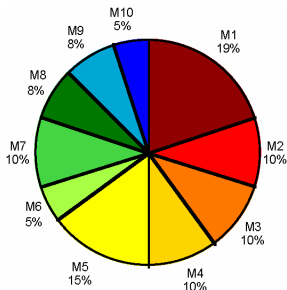


Can also weight each stakeholder

Weight each stakeholder

- E.g. to reflect credibility?
- E.g. to reflect size of constituency represented?

Example:



© Easterbrook 2004

Source: Adapted from Regnell et al, 2000



Resolving Stakeholder Conflict

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

© Easterbrook 2004



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



Lecture 11, Part 2: Software Evolution

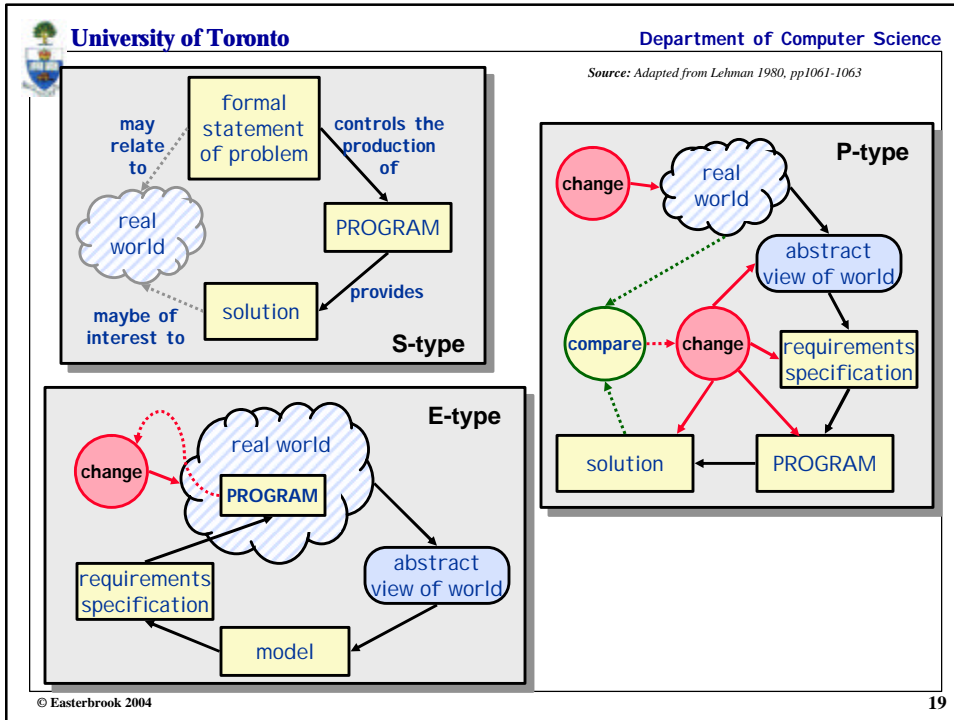
- ⇒ **Basics of Software Evolution**
 - ↳ Laws of software evolution
 - ↳ Requirements Growth
 - ↳ Software Aging
- ⇒ **Basics of Change Management**
 - ↳ Baselines, Change Requests and Configuration Management
- ⇒ **Software Families - The product line approach**
- ⇒ **Requirements Traceability**
 - ↳ Importance of traceability
 - ↳ Traceability tools



Program Types

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



- University of Toronto Department of Computer Science
- ## 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
- © Easterbrook 2004 20

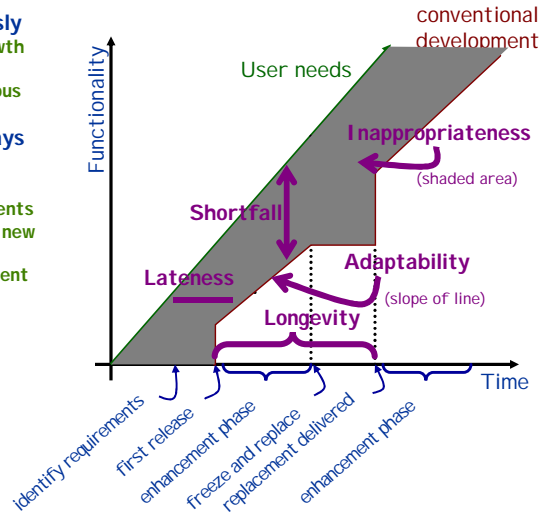


Requirements Growth

Source: Adapted from Davis 1988, pp1453-1455

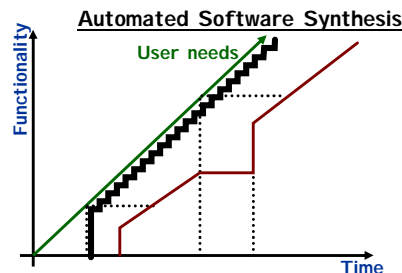
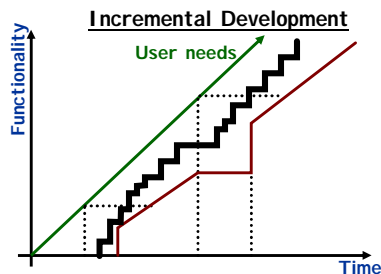
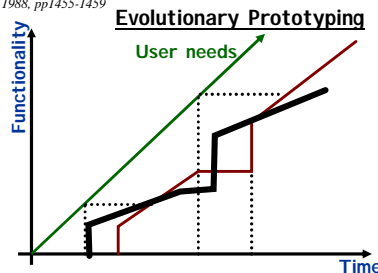
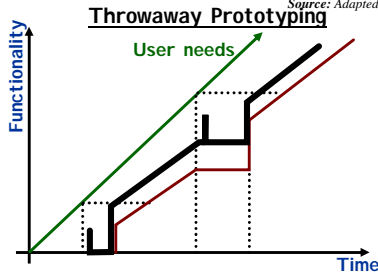
⇒ Davis's model:

- ↳ **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



Software Aging

Source: Adapted from Parnas, 1994

⇒ 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

- ↳ 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:
 - "(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 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



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?