Lecture 10, Part 1: Non-Functional Requirements (NFRs)

- Definitions
  - Quality criteria; metrics
  - Example NFRs
- Product-oriented Software Qualities
  - Making quality criteria specific
  - Catalogues of NFRs
  - Example: Reliability
- Process-oriented Software Qualities
  - Softgoal analysis for design tradeoffs

© 2000-2003, Steve Easterbrook

What are Non-functional Requirements?

- Functional vs. Non-Functional
  - Functional requirements describe what the system should do
    - things that can be captured in use cases
    - things that can be analyzed by drawing sequence diagrams, statecharts, etc.
    - Functional requirements will probably trace to individual chunks of a program
  - Non-functional requirements are global constraints on a software system
    - e.g. development costs, operational costs, performance, reliability, maintainability, portability, robustness etc.
    - Often known as the "ilities"
    - Usually cannot be implemented in a single module of a program
- The challenge of NFRs
  - Hard to model
  - Usually stated informally, and so are:
    - often contradictory,
    - difficult to enforce during development
    - difficult to evaluate for the customer prior to delivery
  - Hard to make them measurable requirements
    - We'd like to state them in a way that we can measure how well they've been met

© 2000-2003, Steve Easterbrook

Example NFRs

- Interface requirements
  - how will the new system interface with its environment?
    - User interfaces and "user-friendliness"
    - Interfaces with other systems
- Performance requirements
  - time/space bounds
    - workloads, response time, throughput and available storage space
    - e.g. "the system must handle 1,000 transactions per second"
  - reliability
    - the availability of components
    - integrity of information maintained and supplied to the system
    - e.g. "system must have less than 1hr downtime per three months"
  - security
    - E.g. permissible information flows, or who can do what
  - survivability
    - E.g. system will need to survive fire, natural catastrophes, etc
- Operating requirements
  - physical constraints (size, weight),
  - personnel availability & skill level
  - accessibility for maintenance
  - environmental conditions
  - etc
- Lifecycle requirements
  - "Future-proofing"
    - Maintainability
    - Enhanceability
    - Portability
    - expected market or product lifespan
  - limits on development
    - E.g development time limitations,
    - resource availability
    - methodological standards
    - etc.
- Economic requirements
  - e.g. restrictions on immediate and/or long-term costs.

© 2000-2003, Steve Easterbrook

Approaches to NFRs

- Product vs. Process?
  - Product-oriented Approaches
    - Focus on system (or software) quality
    - Aim is to have a way of measuring the product once it's built
  - Process-oriented Approaches
    - Focus on how NFRs can be used in the design process
    - Aim is to have a way of making appropriate design decisions
- Quantitative vs. Qualitative?
  - Quantitative Approaches
    - Find measurable scales for the quality attributes
    - Calculate degree to which a design meets the quality targets
  - Qualitative Approaches
    - Study various relationships between quality goals
    - Reason about trade-offs etc.

© 2000-2003, Steve Easterbrook

Software Qualities

- Think of an everyday object
  - e.g. a chair
  - How would you measure it's "quality"?
    - construction quality? (e.g. strength of the joints,…)
    - aesthetic value? (e.g. elegance,…)
    - fit for purpose? (e.g. comfortable,…)
- All quality measures are relative
  - there is no absolute scale
  - we can sometimes say A is better than B…
    - … but it is usually hard to say how much better!
- For software:
  - construction quality?
    - software is not manufactured
  - aesthetic value?
    - but most of the software is invisible
    - aesthetic value matters for the user interface, but is only a marginal concern
  - fit for purpose?
    - Need to understand the purpose

© 2000-2003, Steve Easterbrook

Fitness

Source: *Budgen, 1994, pp58-9*

- Software quality is all about fitness to purpose
  - does it do what is needed?
  - does it do it in the way that its users need it to?
  - does it do it reliably enough? fast enough? safely enough? securely enough?
  - will it be affordable? will it be ready when its users need it?
  - can it be changed as the needs change?
- Quality is not a measure of software in isolation
  - it measures the relationship between software and its application domain
    - cannot measure this until you place the software into its environment…
    - …and the quality will be different in different environments!
  - during design, we need to *predict* how well the software will fit its purpose
    - we need good quality predictors (design analysis)
  - during requirements analysis, we need to *understand* how fitness-for-purpose will be measured
    - What is the intended purpose?
    - What quality factors will matter to the stakeholders?
    - How should those factors be operationalized?

© 2000-2003, Steve Easterbrook

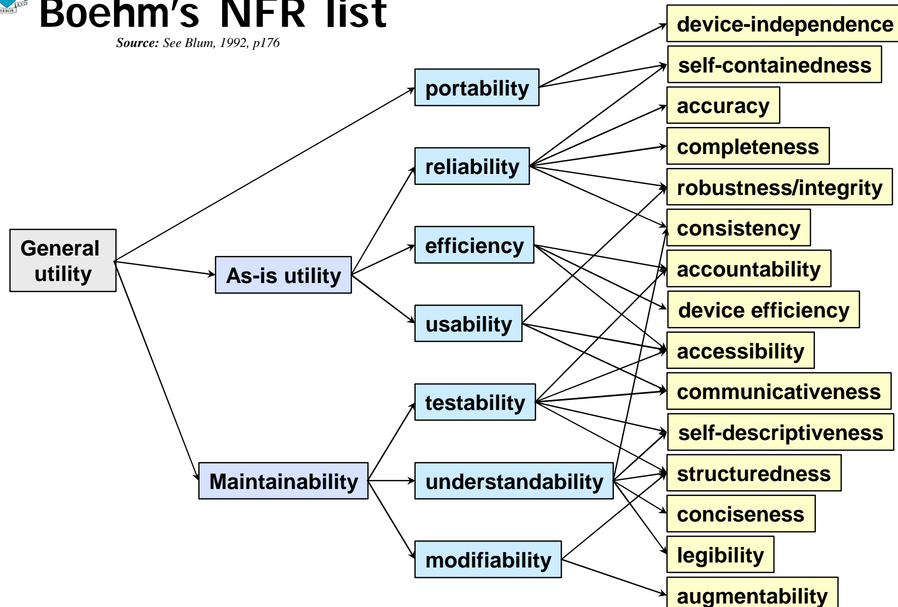
Factors vs. Criteria

- Quality Factors
  - These are customer-related concerns
    - Examples: efficiency, integrity, reliability, correctness, survivability, usability,...
- Design Criteria
  - These are technical (development-oriented) concerns such as anomaly management, completeness, consistency, traceability, visibility,...
- Quality Factors and Design Criteria are related:
  - Each factor depends on a number of associated criteria:
    - E.g. correctness depends on completeness, consistency, traceability,...
    - E.g. verifiability depends on modularity, self-descriptiveness and simplicity
  - There are some standard mappings to help you…
- During Analysis:
  - Identify the relative importance of each quality factor
    - From the customer's point of view!
  - Identify the design criteria on which these factors depend
  - Make the requirements measurable

© 2000-2003, Steve Easterbrook

Boehm's NFR list

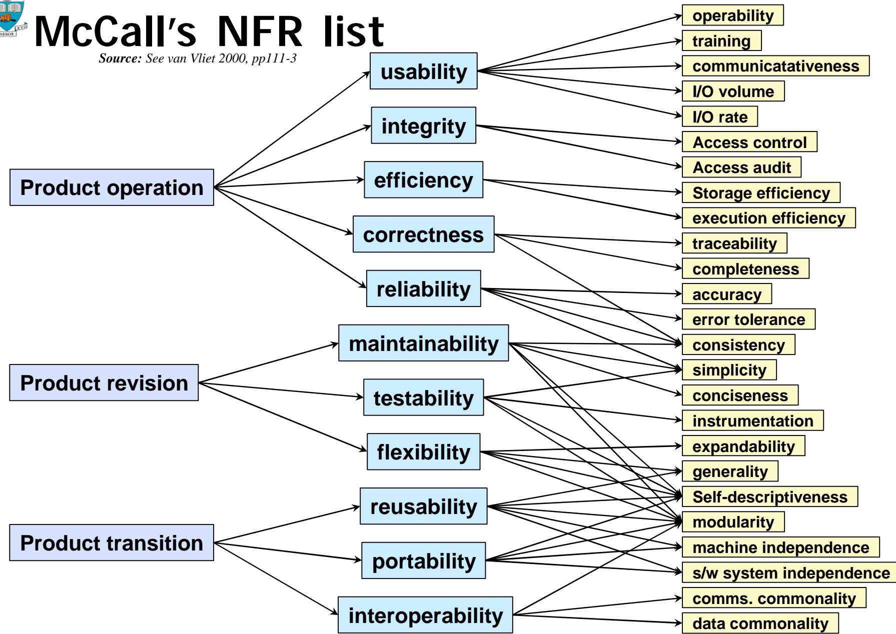
Source: See Blum, 1992, p176



© 2000-2003, Steve Easterbrook

McCall's NFR list

Source: See van Vliet 2000, pp111-3



© 2000-2003, Steve Easterbrook

Making Requirements Measurable

Source: Budgen, 1994, pp60-1

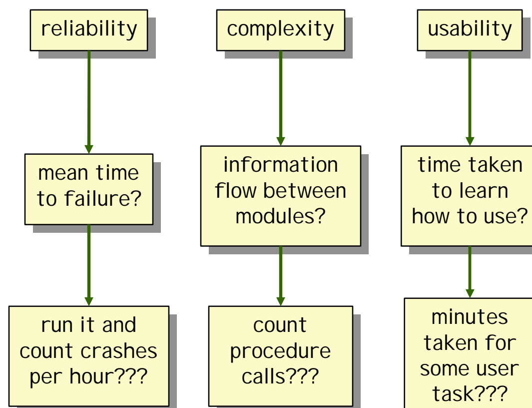
- We have to turn our vague ideas about quality into measurables

The Quality Concepts
(abstract notions of quality properties)

Measurable Quantities
(define some metrics)

Counts taken from Design Representations
(realization of the metrics)

examples...



© 2000-2003, Steve Easterbrook

Example Metrics

Quality	Metric
Speed	transactions/sec response time screen refresh time
Size	Kbytes number of RAM chips
Ease of Use	training time number of help frames
Reliability	mean-time-to-failure, probability of unavailability rate of failure, availability
Robustness	time to restart after failure percentage of events causing failure
Portability	percentage of target-dependent statements number of target systems

© 2000-2003, Steve Easterbrook

Example: Measuring Reliability

Definition
- the ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which it was intended.

Comments:
- Reliability can be defined in terms of a percentage (say, 99.999%)
- This may have different meaning for different applications:
  - Telephone network: the entire network can fail no more than, on average, 1hr per year, but failures of individual switches can occur much more frequently
  - Patient monitoring system: the system may fail for up to 1hr/year, but in those cases doctors/nurses should be alerted of the failure. More frequent failure of individual components is not acceptable.
- Best we can do may be something like:
  - "...No more than X bugs per 10KLOC may be detected during integration and testing; no more than Y bugs per 10KLOC may remain in the system after delivery, as calculated by the Monte Carlo seeding technique of appendix Z; the system must be 100% operational 99.9% of the calendar year during its first year of operation..."

© 2000-2003, Steve Easterbrook

Measuring Reliability…

- Example reliability requirement:
  - "The software shall have no more than X bugs per thousand lines of code"
  - ...But is it possible to measure bugs at delivery time?
- Use bebugging
  - Measures the effectiveness of the testing process
  - a number of seeded bugs are introduced to the software system
    - then testing is done and bugs are uncovered (seeded or otherwise)

$$\text{Number of bugs in system} = \frac{\text{\# of seeded bugs} \times \text{\# of detected bugs}}{\text{\# of detected seeded bugs}}$$

  - ...BUT, not all bugs are equally important!

© 2000-2003, Steve Easterbrook

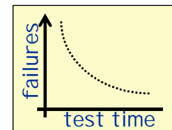
Example model: Reliability growth

Source: Adapted from Pfleeger 1998, p359

- Motorola's Zero-failure testing model
  - Predicts how much more testing is needed to establish a given reliability goal
  - basic model:

$$\text{failures} = ae^{-b(t)}$$

(a, b: empirical constants; t: testing time)



- Reliability estimation process
  - Inputs needed:
    - fd = target failure density (e.g. 0.03 failures per 1000 LOC)
    - tf = total test failures observed so far
    - th = total testing hours up to the last failure
  - Calculate number of further test hours needed using:

$$\frac{\ln(fd/(0.5 + fd)) \times th}{\ln((0.5 + fd)/(tf + fd))}$$

  - Result gives the number of further failure free hours of testing needed to establish the desired failure density
    - if a failure is detected in this time, you stop the clock and recalculate
  - Note: this model ignores operational profiles!

© 2000-2003, Steve Easterbrook

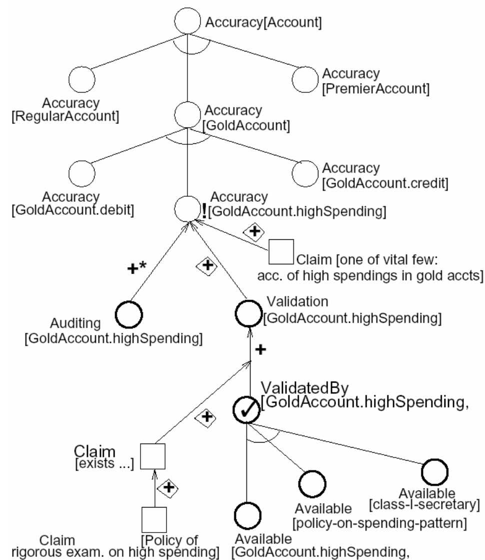
Making Requirements Measurable

- Define 'fit criteria' for each requirement
  - Give the 'fit criteria' alongside the requirement
  - E.g. for new ATM software
    - Requirement: "The software shall be intuitive and self-explanatory"
    - Fit Criteria: "95% of existing bank customers shall be able to withdraw money and deposit cheques within two minutes of encountering the product for the first time"
- Choosing good fit criteria
  - Stakeholders are rarely this specific
  - The right criteria might not be obvious:
    - Things that are easy to measure aren't necessarily what the stakeholders want
    - Standard metrics aren't necessary what stakeholders want
  - Stakeholders need to construct their own mappings from requirements to fit criteria

© 2000-2003, Steve Easterbrook

Using softgoal analysis

- Goal types:
  - Non-functional Requirement
  - Satisficing Technique
    - e.g. a design choice
  - Claim
    - supporting/explaining a choice
- Contribution Types:
  - AND links (decomposition)
  - OR links (alternatives)
  - Sup links (supports)
  - Sub links (necessary subgoal)
- Evaluation of goals
  - Satisficed
  - Denied
  - Conflicting
  - Undetermined



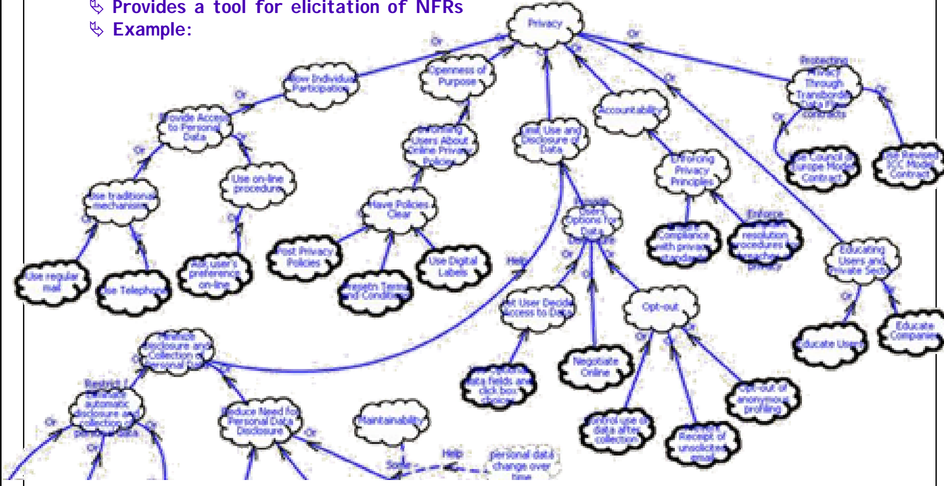
Source: *Chung, Nixon, Yu & Mylopoulos, 1999*

© 2000-2003, Steve Easterbrook

NFR Catalogues

Source: Cysneiros & Yu, 2004

- Predefined catalogues of NFR decomposition
  - Provides a knowledge base to check coverage of an NFR
  - Provides a tool for elicitation of NFRs
  - Example:



Lecture 10, Part 2: Verification and Validation

- Some Refreshers:
  - Summary of Modelling Techniques seen so far
  - Recap on definitions for V&V
- Validation Techniques
  - Inspection (see lecture 6)
  - Model Checking (see lecture 16)
  - Prototyping
- Verification Techniques
  - Consistency Checking
  - Making Specifications Traceable (see lecture 21)
- Independent V&V

The story so far

- We've looked at the following UML diagrams:
  - Activity diagrams
    - capture business processes involving concurrency and synchronization
    - good for analyzing dependencies between tasks
  - Class Diagrams
    - capture the structure of the information used by the system
    - good for analysing the relationships between data items used by the system
    - good for helping you identify a modular structure for the system
  - Statecharts
    - capture all possible responses of an object to all uses cases in which it is involved
    - good for modeling the dynamic behavior of a class of objects
    - good for analyzing event ordering, reachability, deadlock, etc.
  - Use Cases
    - capture the view of the system from the view of its users
    - good starting point for specification of functionality
    - good visual overview of the main functional requirements
  - Sequence Diagrams (collaboration diagrams are similar)
    - capture an individual scenario (one path through a use case)
    - good for modelling dialog structure for a user interface or a business process
    - good for identifying which objects (classes) participate in each use case
    - helps you check that you identified all the necessary classes and operations

© 2000-2003, Steve Easterbrook

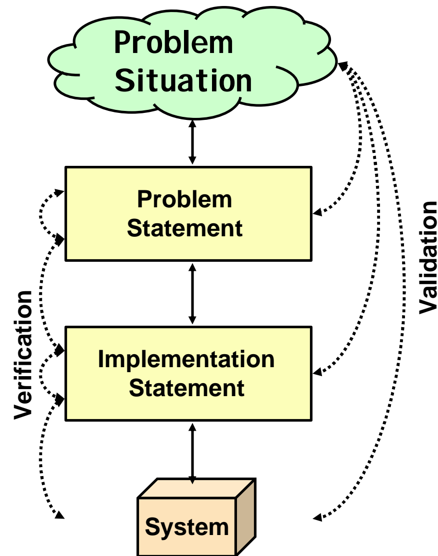
The story so far (part 2)

- We've looked at the following non-UML diagrams
  - Goal Models
    - Capture strategic goals of stakeholders
    - Good for exploring 'how' and 'why' questions with stakeholders
    - Good for analysing trade-offs, especially over design choices
  - Fault Tree Models (as an example risk analysis technique)
    - Capture potential failures of a system and their root causes
    - Good for analysing risk, especially in safety-critical applications
  - Strategic Dependency Models (i*)
    - Capture relationships between actors in an organisational setting
    - Helps to relate goal models to organisational setting
    - Good for understanding how the organisation will be changed
  - Entity-Relationship Models
    - Capture the relational structure of information to be stored
    - Good for understanding constraints and assumptions about the subject domain
    - Good basis for database design
  - Mode Class Tables, Event Tables and Condition Tables (SCR)
    - Capture the dynamic behaviour of a real-time reactive system
    - Good for representing functional mapping of inputs to outputs
    - Good for making behavioural models precise, for automated reasoning

© 2000-2003, Steve Easterbrook

Verification and Validation

- Validation:
  - "Are we building the right system?"
  - Does our problem statement accurately capture the real problem?
  - Did we account for the needs of all the stakeholders?
- Verification:
  - "Are we building the system right?"
  - Does our design meet the spec?
  - Does our implementation meet the spec?
  - Does the delivered system do what we said it would do?
  - Are our requirements models consistent with one another?



Refresher: V&V Criteria

Application Domain Machine Domain

- Some distinctions:
  - Domain Properties: things in the application domain that are true anyway
  - Requirements: things in the application domain that we wish to be made true
  - Specification: a description of the behaviours the program must have in order to meet the requirements
- Two verification criteria:
  - The Program running on a particular Computer satisfies the Specification
  - The Specification, given the Domain properties, satisfies the Requirements
- Two validation criteria:
  - Did we discover (and understand) all the important Requirements?
  - Did we discover (and understand) all the relevant Domain properties?

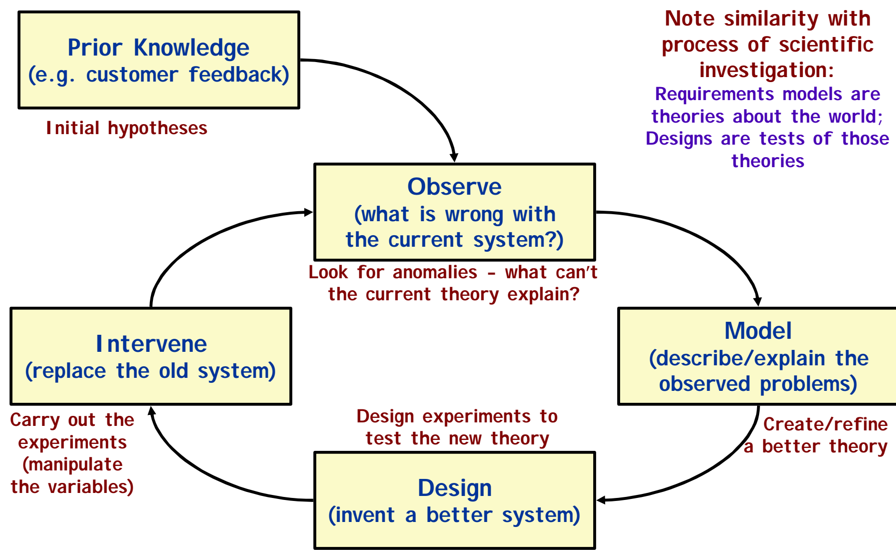
Source: Adapted from Jackson, 1995, p170-171

V&V Example

- **Example:**
  - Requirement R:
    - “Reverse thrust shall only be enabled when the aircraft is moving on the runway”
  - Domain Properties D:
    - Wheel pulses on if and only if wheels turning
    - Wheels turning if and only if moving on runway
  - Specification S:
    - Reverse thrust enabled if and only if wheel pulses on
- **Verification**
  - Does the flight software, P, running on the aircraft flight computer, C, correctly implement S?
  - Does S, in the context of assumptions D, satisfy R?
- **Validation**
  - Are our assumptions, D, about the domain correct? Did we miss any?
  - Are the requirements, R, what is really needed? Did we miss any?

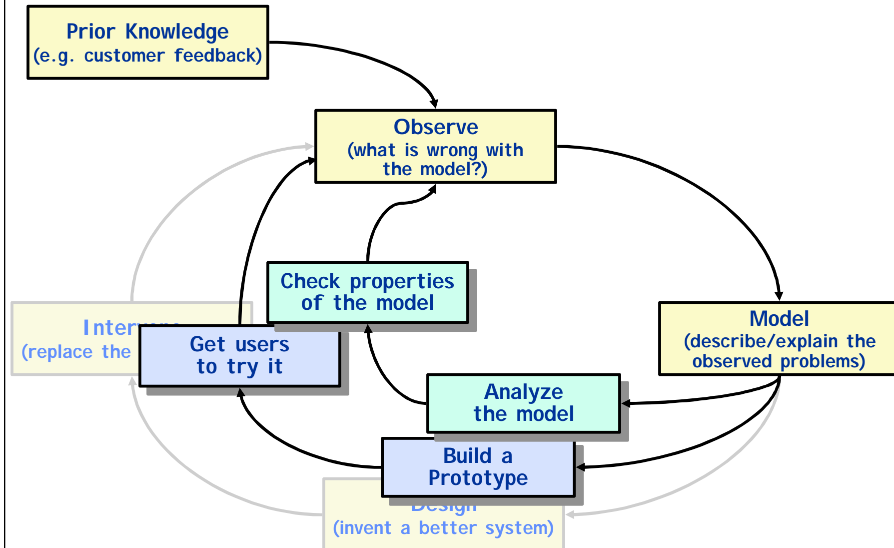
© 2000-2003, Steve Easterbrook

Inquiry Cycle



© 2000-2003, Steve Easterbrook

Shortcuts in the inquiry cycle



© 2000-2003, Steve Easterbrook

Prototyping

"A software prototype is a partial implementation constructed primarily to enable customers, users, or developers to learn more about a problem or its solution." [Davis 1990]

"Prototyping is the process of building a working model of the system" [Agresti 1986]

➲ Approaches to prototyping

- Presentation Prototypes
  - explain, demonstrate and inform – then throw away
  - e.g. used for proof of concept; explaining design features; etc.
- Exploratory Prototypes
  - used to determine problems, elicit needs, clarify goals, compare design options
  - informal, unstructured and thrown away.
- Breadboards or Experimental Prototypes
  - explore technical feasibility; test suitability of a technology
  - Typically no user/customer involvement
- Evolutionary (e.g. "operational prototypes", "pilot systems"):
  - development seen as continuous process of adapting the system
  - "prototype" is an early deliverable, to be continually improved.

© 2000-2003, Steve Easterbrook

Throwaway or Evolve?

- Throwaway Prototyping
  - Purpose:
    - to learn more about the problem or its solution…
    - discard after desired knowledge is gained.
  - Use:
    - early or late
  - Approach:
    - horizontal - build only one layer (e.g. UI)
    - "quick and dirty"
  - Advantages:
    - Learning medium for better convergence
    - Early delivery ® early testing ® less cost
    - Successful even if it fails!
  - Disadvantages:
    - Wasted effort if reqts change rapidly
    - Often replaces proper documentation of the requirements
    - May set customers' expectations too high
    - Can get developed into final product
- Evolutionary Prototyping
  - Purpose
    - to learn more about the problem or its solution…
    - …and reduce risk by building parts early
  - Use:
    - incremental; evolutionary
  - Approach:
    - vertical - partial impl. of all layers;
    - designed to be extended/adapted
  - Advantages:
    - Requirements not frozen
    - Return to last increment if error is found
    - Flexible(?)
  - Disadvantages:
    - Can end up with complex, unstructured system which is hard to maintain
    - early architectural choice may be poor
    - Optimal solutions not guaranteed
    - Lacks control and direction

Brooks: "Plan to throw one away - you will anyway!"

© 2000-2003, Steve Easterbrook

Model Analysis

- Verification
  - "Is the model well-formed?"
  - Are the parts of the model consistent with one another?
- Validation:
  - Animation of the model on small examples
  - Formal challenges:
    - "if the model is correct then the following property should hold..."
  - 'What if' questions:
    - reasoning about the consequences of particular requirements;
    - reasoning about the effect of possible changes
    - "will the system ever do the following..."
  - State exploration
    - E.g. use a model checking to find traces that satisfy some property

© 2000-2003, Steve Easterbrook

Basic Cross-Checks for UML

Use Case Diagrams
- Does each use case have a user?
  - Does each user have at least one use case?
- Is each use case documented?
  - Using sequence diagrams or equivalent

Class Diagrams
- Does the class diagram capture all the classes mentioned in other diagrams?
- Does every class have methods to get/set its attributes?

Sequence Diagrams
- Is each class in the class diagram?
- Can each message be sent?
  - Is there an association connecting sender and receiver classes on the class diagram?
  - Is there a method call in the sending class for each sent message?
  - Is there a method call in the receiving class for each received message?

StateChart Diagrams
- Does each statechart diagram capture (the states of) a single class?
  - Is that class in the class diagram?
- Does each transition have a trigger event?
  - Is it clear which object initiates each event?
  - Is each event listed as an operation for that object's class in the class diagram?
- Does each state represent a distinct combination of attribute values?
  - Is it clear which combination of attribute values?
  - Are all those attributes shown on the class diagram?
- Are there method calls in the class diagram for each transition?
  - ...a method call that will update attribute values for the new state?
  - ...method calls that will test any conditions on the transition?
  - ...method calls that will carry out any actions on the transition?

Independent V&V

- **V&V performed by a separate contractor**
  - Independent V&V fulfills the need for an independent technical opinion.
  - Cost between 5% and 15% of development costs
  - Studies show up to fivefold return on investment:
    - Errors found earlier, cheaper to fix, cheaper to re-test
    - Clearer specifications
    - Developer more likely to use best practices
- **Three types of independence:**
  - Managerial Independence:
    - separate responsibility from that of developing the software
    - can decide when and where to focus the V&V effort
  - Financial Independence:
    - Costed and funded separately
    - No risk of diverting resources when the going gets tough
  - Technical Independence:
    - Different personnel, to avoid analyst bias
    - Use of different tools and techniques

Some philosophical views of validation

➲ logical positivist view:

- "there is an objective world that can be modeled by building a consistent body of knowledge grounded in empirical observation"
- ✎ In RE, assumes there is an objective problem that exists in the world
  - Build a consistent model; make sufficient empirical observations to check validity
  - Use tools that test consistency and completeness of the model
  - Use reviews, prototyping, etc to demonstrate the model is "valid"

➲ Popper's modification to logical positivism:

- "theories can't be proven correct, they can only be refuted by finding exceptions"
- ✎ In RE, design your requirements models to be refutable
  - Look for evidence that the model is wrong
  - E.g. collect scenarios and check the model supports them

➲ post-modernist view:

- "there is no privileged viewpoint; all observation is value-laden; scientific investigation is culturally embedded"
- E.g. Kuhn: science moves through paradigms
- E.g. Toulmin: scientific theories are judged with respect to a *weltanschauung*
- ✎ In RE, validation is always subjective and contextualised
  - Use stakeholder involvement so that they 'own' the requirements models
  - Use ethnographic techniques to understand the weltanschauungen

© 2000-2003, Steve Easterbrook