SW Quality Overview
ELEC372

References

- "The Elusive Target", Kitchenham, Pfleeger, IEEE Software Jan 96
- Software Architecture in Practice, Bass, Clements, Kazman
- Lectures from CFICSE, Terry Shepard

What does quality mean?

- Operational meanings:
  - Software does what is wanted and expected
  - On budget and on schedule
  - Users are happy
  - Level of bugginess is tolerable
  - Changes are not too expensive
  - Software is cost-effective in use and supports organizational mandate

Definitions of quality

- OED: the standard of something when compared to other things like it (a wine of excellent quality); of high quality, general excellence
- Crosby: zero defects
- ISO: The totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs.
- Juran: Fitness for Purpose
- DoD: The degree to which the attributes of the software enable it to perform its intended end use.

Quality is relative

- Rolls Royce: quality when cost is no object
- Ford Taurus: quality tradeoffs when cost is a factor
- Mini: minimum acceptable quality when cost is the dominant factor.
  - Costs include initial cost, cost of operation, and cost of repairs.

Quality - general

- is multidimensional (all aspects of a car)
- is subject to constraints (e.g. cars must fit on roads)
- is about acceptable compromises (e.g. poorer ride for lower cost)
- criteria are not independent (e.g. fuel use vs. vehicle weight)
- is not absolute (all levels of cars have quality)

Special qualities of software

- not physical
- needs are usually uncertain
- needs change
- platforms change (CPUs, OS, network)
- high expectations, especially of adaptability

Changing approaches to software quality

- increasing use of metrics
- increasing emphasis on process, components and intermediate products, as opposed to the end product
- software crisis persists, but progress is being made

Dimensions of Quality [partial list]

- Timeliness
- Functionality
- Cost
- Correctness
- Reliability
- Maintainability

Other Dimensions of Quality

Usability, Flexibility, Adaptability, Testability, Readability, Portability, Device independence, Self containedness, Reusability, Interoperability, Efficiency, Device efficiency, Integrity, Security, Accuracy, Robustness, Completeness, Consistency, Accountability, Accessibility, Usability, Self descriptive, Fault tolerance, Structuredness, Conciseness, Legibility, Augmentability, Modifiability, Understandability, Clarity, Resilience, Validity, Generality, Minimality, Modularity, Expandability,...

Quality - software

- is multidimensional (portability, usability, functionality)
- is subject to constraints (regulations, rules, resources)
- is about acceptable compromises (e.g. less functionality / lower cost)
- criteria are not independent (e.g. efficiency vs maintainability)
- is not absolute (different levels of quality are acceptable for different software types)

How can the Dimensions of Quality be assured?

- by establishing and enforcing a process
- process must include measurement
- by making sure everyone understands the qualities that must be embedded in the product
- by keeping focused on the relevant qualities throughout the development lifecycle
- by determining “up front” how to achieve each quality
- verification and validation

Cost of Quality

- prevention
  - quality planning
  - formal technical reviews
  - test equipment
  - training
- appraisal
  - in-process & interprocess inspection
  - equipment calibration & maintenance
  - testing
- failure - internal
  - rework
  - repair
  - failure mode analysis
- failure - external
  - complaint resolution
  - product return & replacement
  - help line support
  - warranty work

Views of Quality (not software specific)

- Transcendental: "eureka!", "the ideal"
- User: fitness for purpose
- Manufacturing: conformance to spec
- Product: characteristics of the product
- Value-based: willingness to pay

User View

- can be highly personalized
- typical user views:
  - reliability
  - performance
  - usability
- ISO 9126

Manufacturing View

- avoid rework and associated costs
- often leads to process focus (CMM, ISO 9001-3)
- process does not guarantee good products; only guarantees uniformity of output (does insist on improving process)

Product View

- measure internal characteristics?
  - Measurements chosen depend on type of product (e.g. code, design, …)
  - may not ensure quality of external characteristics
- measure external characteristics?
  - Corresponds to user/manufacturer views
  - measurements chosen depend on product (e.g. OS, spreadsheet, …)

Value-based view

- helps to decide trade-offs during development
  - cost vs. reliability
  - cost vs. functionality
- cost to consumer is affected by scale
  - high volume products have a higher perceived value
- value is defined by
  - potential savings/productivity enhancements
  - profit opportunities

Problems with Qualities

- qualities are at odds with one another
  - always a compromise
    - portability versus performance
    - security versus fault tolerance
    - modularity versus efficiency
- difficulty in trying to measure qualities e.g., accuracy, usability
- what does quality really mean?

Proposed solution 1: Focus on Product

- most dimensions of quality are measured after the fact
- need to identify actions that can embed needed qualities in the products, up front
- implementation quality model: identify what the programmer can do to ensure quality
  - link these actions to high level qualities

Proposed Solution 1 (cont'd)

- requirements quality model: all requirement statements must be measurable
- design quality model: define qualities for design, identify characteristics of design that would embed those qualities

Proposed solution 2: Focus on Process

- integrate quality decisions into the design process, e.g.:
  - by using quality checklists during inspection, design
  - by determining testing criteria up front
  - by following a documented design process

Quality Approach with Architectures

- (Bass, Clements, Kazman)
- product approach
  - design (architecture and low-level)
  - code
- quality attributes discernable at runtime
- quality attributes not discernable at runtime
- business qualities

Qualities Discernable at Runtime (1)

- performance
  - responsiveness of system
  - addressed at both architecture level and code level
- security
  - ability to resist unauthorized attempts at usage at denial of service
  - addressed through architectural solutions

Qualities Discernable at Runtime (2)

- availablity
  - proportion of time system is up and running
  - (mean time to failure)/(mean time to failure + mean time to repair)
  - fault-tolerant architectures, separation of concerns, modifiability, testability
- functionality
  - ability to do intended work
  - largely non-architectural in nature

Qualities Discernable at Runtime (3)

- usability
  - learnability, efficiency, memorability, error avoidance, error handling, satisfaction
  - mostly use interface concerns
  - flow of information and performance are architectural concerns

Qualities Not Discernable at Runtime (1)

- modifiability
  - ability to make changes quickly and cost effectively
  - quality attribute most closely aligned with architecture
- portability
  - ability to run under different computing environments
  - typically addressed by layered architecture

Qualities Not Discernable at Runtime (2)

- reusability
  - ability for structures and components to be reused in future applications
  - architectural components are units of reuse
- integrability
  - ability to make separately developed components work correctly together
  - depends on the external complexity of components and interactions with other components

Qualities Not Discernable at Runtime (3)

- testability
  - ease with which software can be made to demonstrate its faults through execution based testing
  - tied to observability and controlability
  - related to architectural documentation, separation of concern, information hiding, incremental development

Business Qualities

- time to market
- cost
- projected lifetime of system
- targeted market
- rollout schedule
- integration with existing legacy systems

ELEC 372

Software Inspection

References

- Software Inspection, Tom Gilb & Dorothy Graham, Addison & Wesley, 1993
- Managing the Software Process, Watts Humphrey, Addison-Wesley, 1990
- "Experience with Inspection in Ultralarge-Scale Developments", Glen W. Russell, IEEE Software, January 1991, pp. 25-31
- "Active Design Reviews: Principles and Practice", Parnas, David L., David M. Weiss, Proceedings 8th International Conference on Software Engineering, Aug. 1985
- "An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development", Porter et al, IEEE Transaction on SE, June 1997

What is Software Inspection?

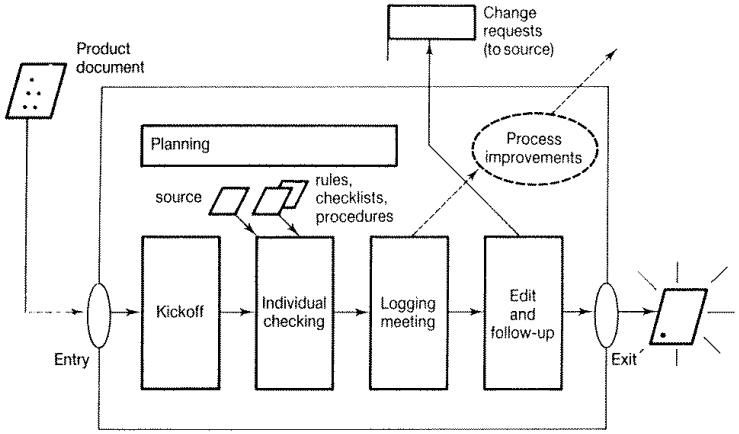
- Formal process to statically analyze a software product and record metrics related to that product.
  - AT&T classifies inspection as a form of testing
  - inspection does not replace dynamic testing
    - inspection finds defects
    - testing finds faults
      - need to do debugging to find defect
    - difficult to inspect for qualities such as performance, timing, load
  - inspection can be applied earlier than testing
  - inspection can be applied to any software product
  - static analysis tools can find some types of defects
    - undeclared variables, unused variables, control flow errors
    - should inspect for things tools can't find

Inspection Metrics

- to be classified as "inspection", metrics must be kept
- otherwise it's a technical review or walkthrough

- record
  - defects found in product under inspection
  - where defect found in product
  - time taken to perform inspection
  - size of product inspected
- may also record
  - category of defect (e.g.,major/minor)
  - what prompted detecting the defect
  - suggested improvements
  - failure counts collected during testing and in the field

Inspection process from Gilb and Graham



Product Inspection steps (simplified).

Steps in Fagan-Style Inspection (1)

- Entry
  - entry criteria set up for products to be inspected
  - a lot of controversy about when to inspect code
    - before or after testing
    - before or after compile
  - if entry criteria not met, product not inspected
    - e.g., too many typing errors
    - e.g., source code does not compile cleanly
  - inspections are expensive
    - don't waste inspectors time

Steps in Fagan-Style Inspection (2)

- Kickoff
  - familiarize everyone involved with task at hand
  - define everyone's roles
  - hand out materials needed
  - deal with general questions
  - set goals
  - discuss overall plan, timetable

Steps in Fagan-Style Inspection (3)

- Individual Checking
  - inspector works alone using documents handed out in kick-off meeting
  - aims to find maximum number of unique major potential defects
    - mainly by looking for discrepancies between source documents and products being inspected
    - most common approach is guidance by checklist
  - must complete by specified time
  - must use optimum checking rate
    - 1 hour per product document page: intensely studied
    - 20 LOC per hour to 200 LOC per hour
      - depends on complexity of code and familiarity of inspector
  - record issues found
    - description, location, possibly impact
  - track time taken

Steps in Fagan-Style Inspection (4) Logging Meeting

- purpose
  - log issues found by each inspector
  - no evaluation of issues allowed
  - to discover more issues (synergy)
  - identify and log ways to improve inspection process
- controlled by moderator
  - must be very skilled in handling people and issues
  - maintain time discipline and issue discipline
- also attended by product author and 2 or 3 inspectors
- must not last more than 2 hours
- not for explaining, defending, suggesting fixes, or training
- identify issues, then log them
- issues can be questions to product author and suggestions for improvements
  - not discussed in meeting
- NOT an attack or evaluation of product author
- management personnel should NOT be in attendance

Steps in Fagan-Style Inspection (5)

- Edit
  - log of issues given to product author to resolve
  - author issues change request to follow through on changes to be done
  - author can
    - classify or reclassify issues
    - voluntarily improve product as per issues raised
    - request rules or checklist be changed
- Follow-up
  - inspection leader makes sure satisfactory closure has taken place on all issues
  - change requests issues, changes made, etc.
  - if major changes, then product may be inspected again

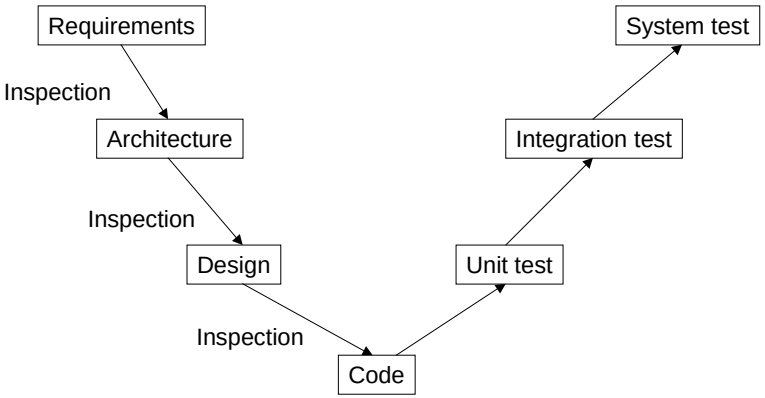
Steps in Fagan-Style Inspection (6)

- Exit
  - need exit criteria
  - need closure on all issues in writing
  - record metrics for inspection process (for process improvement)
  - product now considered "safe" to pass on to next phase in software development

Inspection Participants

- should receive training in doing inspections
- leader
  - must be skilled at the job
  - not necessarily the logging meeting moderator
  - organizes whole activity
    - looks at product (entry criteria) to see if it is ready for inspection
    - plans inspection
      - who to participate
      - divide product to be inspected into manageable chunks
      - schedules, booking rooms
      - distributing copies of documents
    - set up master plan for inspection
- inspectors
  - must be skilled in some aspect of the product
  - must be respected for that skill
  - must be objective in assessments

Inspections in a Software Process



Cost of running inspections is about 10-15% of cost of development budget.

Software Inspection Research (1)

- Research into inspection
  - inspection process
  - techniques for individual inspectors
- research at Lucent Technologies (University of Maryland)
  - experiment where they varied
    - number of inspectors
    - number of teams inspecting the same code
    - teams working in parallel and teams working in tandem (where bugs were fixed before 2nd inspection took place)
  - one of the conclusions
    - most significant improvements in inspection comes from improving the way the inspectors do their individual work

Software Inspection Research (2)

- Parnas & Weiss; Active Design Reviews
  - inspectors have to actively work with the product
  - used checklists that required more than a yes/no answer
  - inspectors asked to look for specific types of issues based on individual expertise
- University of Maryland, National Research Council, etc.
  - new techniques to help guide inspectors in their work
    - scenario-based reading
      - eg. put yourself in the role of a tester
      - have a series of questions and scenarios to work with as you inspect

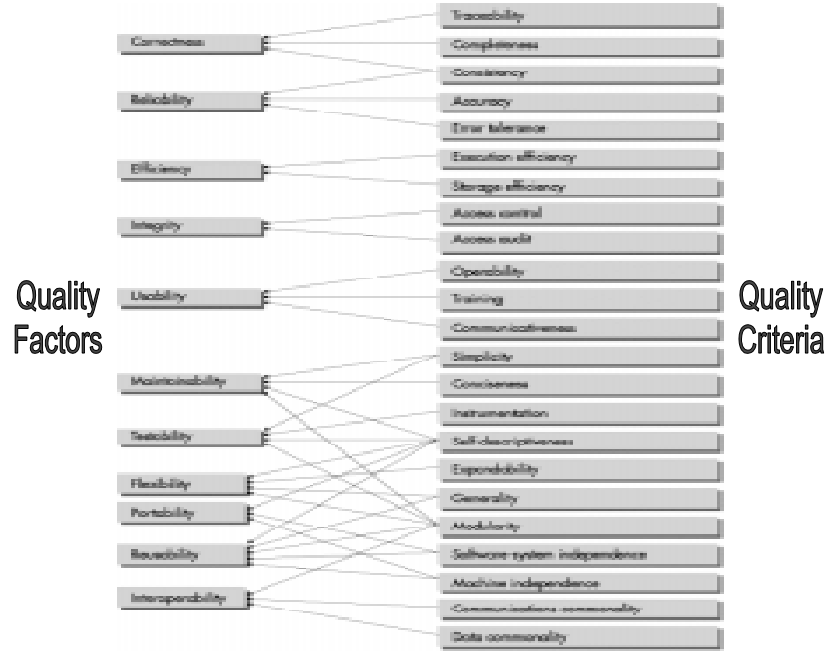
Checklists

- probably most common guidance used for inspections
- good checklists are hard to make up
  - rely on past experience
  - can get too long and unwieldy
  - need to be updated as new situations arise
  - inspectors have a tendency to follow the checklist and do no more
  - can be misinterpreted
  - may not cover everything necessary to the right level of detail

Approach for Checklists

- (general approach for inspections as well)
- determine what qualities are important for the software system
- determine what software design/code characteristics contribute to those qualities
- determine what you want the inspectors to look for
  - also determines practices you want the software developers to follow

McCall's Quality Model



Checklist for ELEC 372 - Quality Factors

- Choose the qualities of
  - correctness
  - reliability
  - efficiency
  - maintainability
  - testability

Checklist for ELEC 372 - Quality Criteria

- Divide the qualities into finer criteria
  - correctness
    - completeness, consistency
  - reliability
    - consistency, anomaly management
  - efficiency
    - effective processing, effective storage
  - maintainability
    - simplicity, self-descriptiveness, modularity, reusability
  - testability
    - simplicity, self-descriptiveness, modularity, instrumentation

Checklist for ELEC 372 - Examples

- self-descriptiveness
  - constants used (no magic numbers)
  - Descriptive variable/method/class names
- instrumentation
  - provide methods to dump the contents of complex data structures (ADTs)
- consistency
  - consistent use of white space

ELEC 372

The Software Development Process (1)

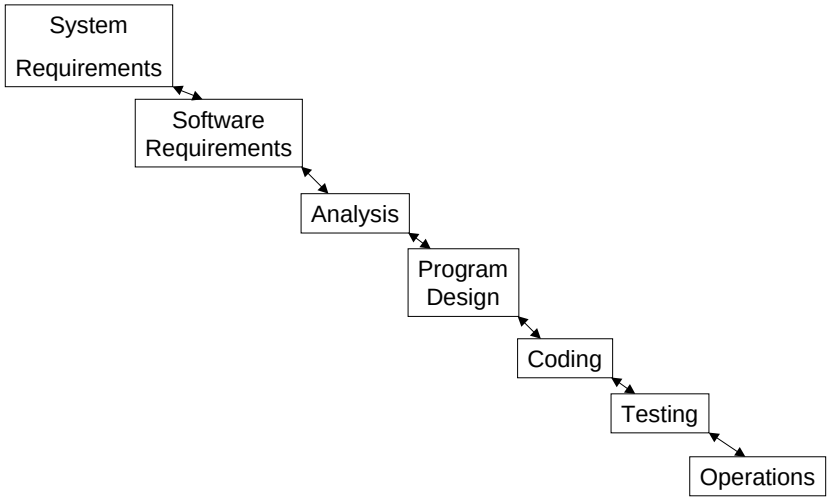
References

- Structured Rapid Prototyping, Connell & Shafer, 1989
- Managing the Software Process, Humphrey
- Software Engineering, A Practitioner's Approach, Pressman, 1997
- "Managing the Development of Large Software Systems: Concepts and Techniques", Dr. Winston Royce, 1970

Ad Hoc Process

- analysis ---> code --> test --> code --> test …
- works for small codes, one programmer that uses the code themselves
- for larger systems, doomed to failure
- "Waterfall Model"

Waterfall Model (Royce)



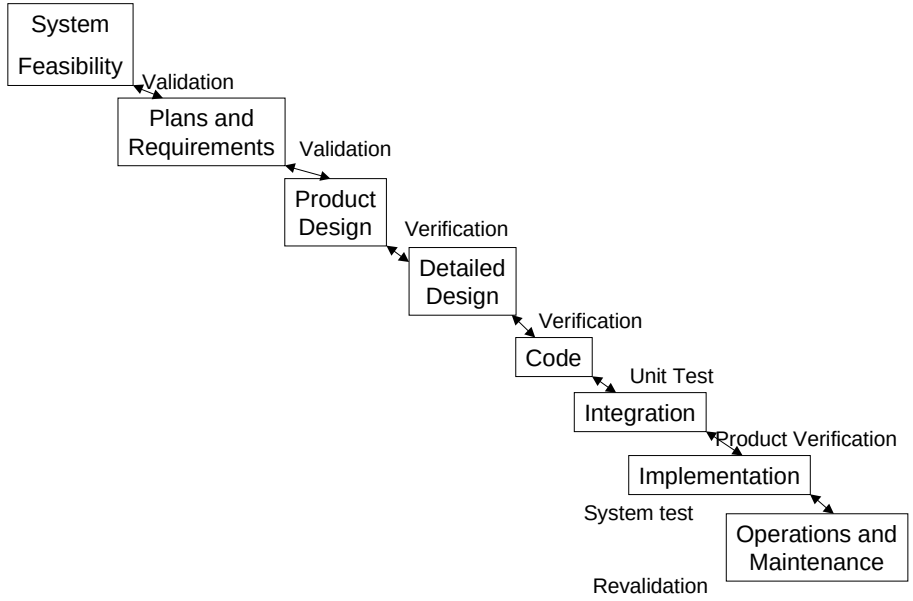
Premise of Waterfall Model (1)

- each step completed successfully provides strong foundation for next step
- forces developers to cope with data and control flows, errors, problems earlier in development
- each step should be performed by those skilled in those steps
- document everything: software requirements, preliminary design, interface design, final design, test plan/test results, operating instructions
  - "To procure 5 million dollars of software, I would estimate a 1500 page specification is about right." (Royce)

Premise of Waterfall Model (2)

- Plan, control, and monitor testing
  - extensive testing
  - use test specialists
  - visually inspect the code before testing
  - plan the test process
  - test every logic path
  - use testing standards, procedures, tools
- Involve the customer
  - include customers in reviews, preliminary design, final acceptance
  - involvement should be formal, in-depth, continuing
- Royce suggests "Do it twice."
  - also suggestion of Fred Brooks (Mythical Man-Month)

Boehm's Refined Waterfall Model



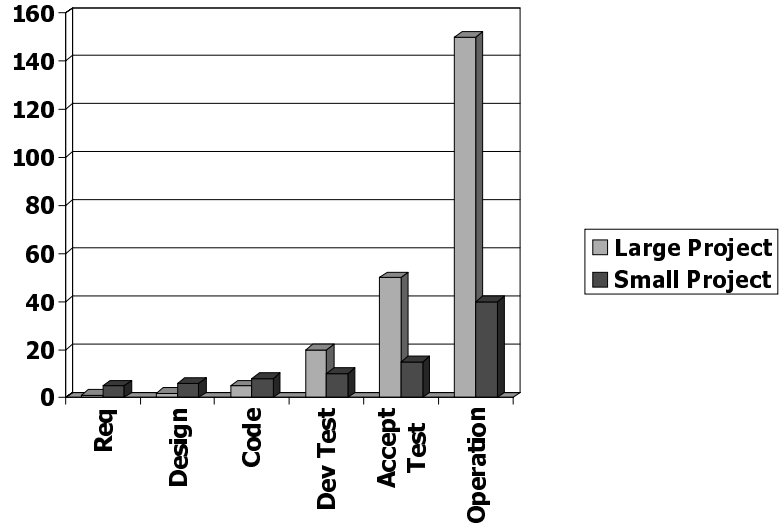
Reasons for New Model

- test phase done very late in development
  - each phase now culminated by a V&V activity
- economic rationale
  - for a successful software product all subgoals must be achieved at some stage anyways
  - any different order of subgoals produces a less successful software product
- usual reasons for lack of success
  - infeasibility and mismatches with user requirements
    - must have all steps present in process

Verification and Validation

- Verification
  - "Have we done the thing right?"
  - ISO 9000-3
    - "The process of evaluating the products of the given phase to ensure correctness and consistency with respect to the products and standards provided as input to that phase."
- Validation
  - "Have we done the right thing?"
  - ISO 9000-3
    - "The process of evaluating software to ensure compliance with specified requirements."

Relative Cost to Fix or Change Software Throughout Lifecycle



Strengths and Weaknesses of Waterfall Model (1)

- Primary function
  - determine order of stages in software development
  - set transition requirements from one stage to another
- Waterfall Model
  - document (or task) driven
  - good for acquisition of software
  - better than ad-hoc process
  - most widely used process model in standards and industry

Strengths and Weaknesses of Waterfall Model (2)

- weaknesses
  - emphasis on fully elaborated documents before going onto next stage
  - only works for well defined, mature, predictable types of software
  - doesn't work for highly interactive software, new technology, research
  - software supported by 4GLs does not need elaborate requirements and design documentation
  - real projects rarely follow sequential flow of model
  - difficult for customer to state all requirements explicitly at start of project
  - working version of program not available until very late in project life-span
  - major blunder may not be discovered until very late
  - sequential style leads to bottlenecks and doesn't lend itself to parallel activities

Software Prototyping

- traditional understanding of "prototype"
  - mock-up of a newly engineered product
    - prototype aircraft to test flying, check systems
    - make refinements before going into production
    - have to build nearly whole thing (airframe, engines, cockpit controls, hydraulics, landing gear, electronics, etc.) before test
- for software, prototype used to
  - try out on customers
    - firm up requirements
    - no functions, just skeleton, no special cases
    - at extreme, may appear to work
  - try out possible trouble areas
    - performance; time and resources
    - achievable; new technology
    - investigating different options

Key considerations for using Prototypes

- objectives for each prototype activity should be clearly established
- define the prototype process ahead (prototyping does NOT mean hacking)
- results from prototyping effort must be documented and analyzed before decision on how to proceed
- complete one prototype and analyze results before starting next iteration
- prototype results can be
  - throw-away
  - foundation for enhancement
  - repackaged as product

Competing Philosophies on Prototyping (1)

- throw away prototype
  - help customer identify requirements
  - tools used for prototype not suitable for production
- quick & dirty prototype
  - used to quickly bring up system, modify until customer grants approval
  - resulting software difficult to maintain
    - heavily patched and no documentation
  - working version available quickly
  - reincarnation of code & fix
  - misused by some software contractors?

Competing Philosophies on Prototyping (2)

- detailed design-driven prototype
  - from engineering: pre-production model
  - model complete and as perfect as possible
    - test-driven by user; could uncover major changes
- non-functioning mock-ups
  - can be "wizard-of-oz" set up
  - usually provides visual examples of inputs and outputs from system processes
  - used to follow flow of tasks
  - sometimes on paper only or on computer screens only
  - examples manually prepared by developers
    - no data actually input to computer
    - not results oriented

Weaknesses of Prototyping

- what do we do with prototype when it has served its purpose?
- customers see what appears to be a working system
  - usually held together with "chewing gum and baling wire"
  - customers expects prototype to be delivered system
  - management may collapse to customers demands
- developer has made implementation compromises
  - inappropriate O/S or programming language
  - may become an integral part of the system

The Incremental Model

- combination of waterfall model and prototyping
- each sequence of the process
  - produces a deliverable "increment" of the software
  - can incorporate the prototyping paradigm
- first increment is often the core product
  - basic requirements met
  - supplementary features undelivered
- evaluation with user
- plan for next increment
- repeat until full product delivered
- focus on delivery of operational product with each increment
- useful if staffing unavailable for a complete implementation

Evolutionary Rapid Prototyping

- "an easily built, readily modifiable, ultimately extensible, partially specified, working model of primary aspects of a proposed system"
- primary objective: "provide a cost-effective means of discovering the true and complete set of system functional requirements that will optimally satisfy the legitimate business needs of the user"
- not throw-away
  - evolves into final system
- not quick-and-dirty
  - detailed analysis and design take place
- not mock-up
  - uses real user data

Steps in Evolutionary Rapid Prototyping (1)

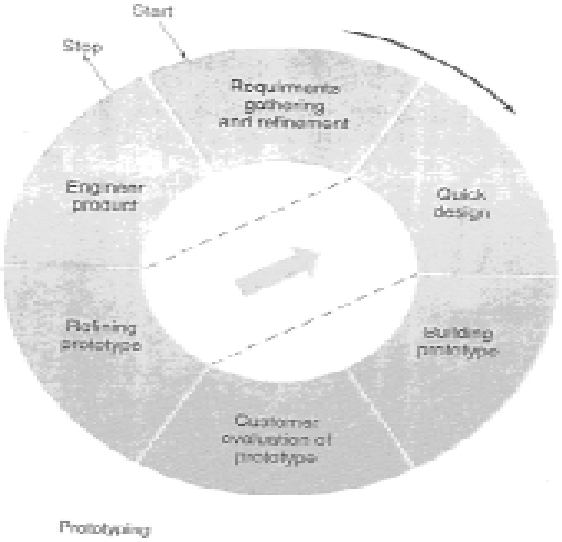
- ***Prototype most important aspects of the system first***
- ***Have a prototyping process/procedure defined ahead***

- divide application into well defined pieces
  - pieces that can be developed independently
- decide order of prototyping different pieces
  - ordering may change as development proceeds
- plan prototyping activities
  - decide objectives
  - don't get into endless loop of prototyping

Steps in Evolutionary Rapid Prototyping (2)

- get prototype working
- evaluate your results
  - show it to someone!
  - keep objectives in mind the whole time
- make refinements
- update requirements documentation
- freeze portion of prototype that's done
- move on to next piece to prototype
- build final product from prototype pieces
  - final product may need some fine tuning to satisfy product requirements
  - final product must be validated against project requirements before turning over to user

Evolutionary Rapid Prototyping



Weakness of Evolutionary Rapid Prototyping

- can easily turn into "quick and dirty"
- generally dependent on 4GLs
- non-functional issues such as performance left until last
  - no overall design done to address these
- some applications do not lend themselves to rapid prototyping
  - operating systems, telecomm software, real time systems, dbm systems, scientific research software
  - rapid prototyping focus is on user satisfaction