Dependability Properties

Dr. Paul West

Department of Computer Science
College of Charleston

January 16, 2014

- When is the best time to detect a design error? During design? Initial development? Testing Phase? Alpha? Beta? Production?
- Quality added after development produces poor costly software.
  - Quality results from a set of inter-dependent activities
  - Analysis and testing are crucial but far from sufficient.
- Testing is not a phase, but a lifestyle
  - Testing and analysis activities occur from early in requirements engineering through delivery and subsequent evolution.
  - Quality depends on every part of the software process
- An essential feature of software processes is that software test and analysis is thoroughly integrated and not an afterthought

The Quality Process

- Quality process: set of activities and responsibilities
  - focused primarily on ensuring adequate dependability
  - concerned with project schedule or with product usability
- The quality process provides a framework for
  - selecting and arranging activities
  - considering interactions and trade-offs with other important goals.

Interactions and Tradeoffs

Example: high dependability vs. time to market

- Mass market products:
  - better to achieve a reasonably high degree of dependability on a tight schedule than to achieve ultra-high dependability on a much longer schedule
- Critical medical devices:
  - better to achieve ultra-high dependability on a much longer schedule than a reasonably high degree of dependability on a tight schedule

Properties of the Quality Process

- Completeness: Appropriate activities are planned to detect each important class of faults.
- Timeliness: Faults are detected at a point of high leverage (as early as possible)
- Cost-effectiveness: Activities are chosen depending on cost and effectiveness
  - cost must be considered over the whole development cycle and product life
  - the dominant factor is usually the cost of repeating an activity through many change cycles.

Planning and Monitoring

- The quality process
  - Balances several activities(testing, coding, designing) across the whole development process
  - Selects and arranges them to be as cost-effective as possible
  - Improves early visibility
- Quality goals can be achieved only through careful planning
- Planning is integral to the quality process

Process Visibility

- A process is visible to the extent that one can answer the question
  - How does our progress compare to our plan?
  - Example: Are we on schedule? How far ahead or behind?
- The quality process has not achieved adequate visibility if one cannot gain strong confidence in the quality of the software system before it reaches final testing
  - quality activities are usually placed as early as possible
    - design test cases at the earliest opportunity (not “just in time”)
    - uses analysis techniques on software artifacts produced before actual code.
  - motivates the use of ad hoc/proxy measures
    - Ex: the number of faults in design or code is not a true measure of reliability, but we may count faults discovered in design inspections as an early indicator of potential quality problems

A&T Strategy

- Identifies company- or project-wide standards that must be satisfied
- AKA Company Testing Strategy
  - procedures required, e.g., for obtaining quality certificates
  - techniques and tools that must be used
  - documents that must be produced

A&T Plan

- A comprehensive description of the quality process that includes:
  - objectives and scope of A&T activities
  - documents and other items that must be available
  - items to be tested
  - features to be tested and not to be tested
  - analysis and test activities
  - staff involved in A&T
  - constraints
  - pass and fail criteria
  - schedule
  - deliverables
  - hardware and software requirements
  - risks and contingencies

Quality Goals

- Process qualities (visibility)
  - Product qualities
  - internal qualities (maintainability, reusability, traceability)
    - Affects external properties like turn around time and prevent future errors.
  - external qualities
    - usefulness qualities:
    - usability, performance, security, portability, interoperability
    - dependability
    - correctness, reliability, safety, robustness

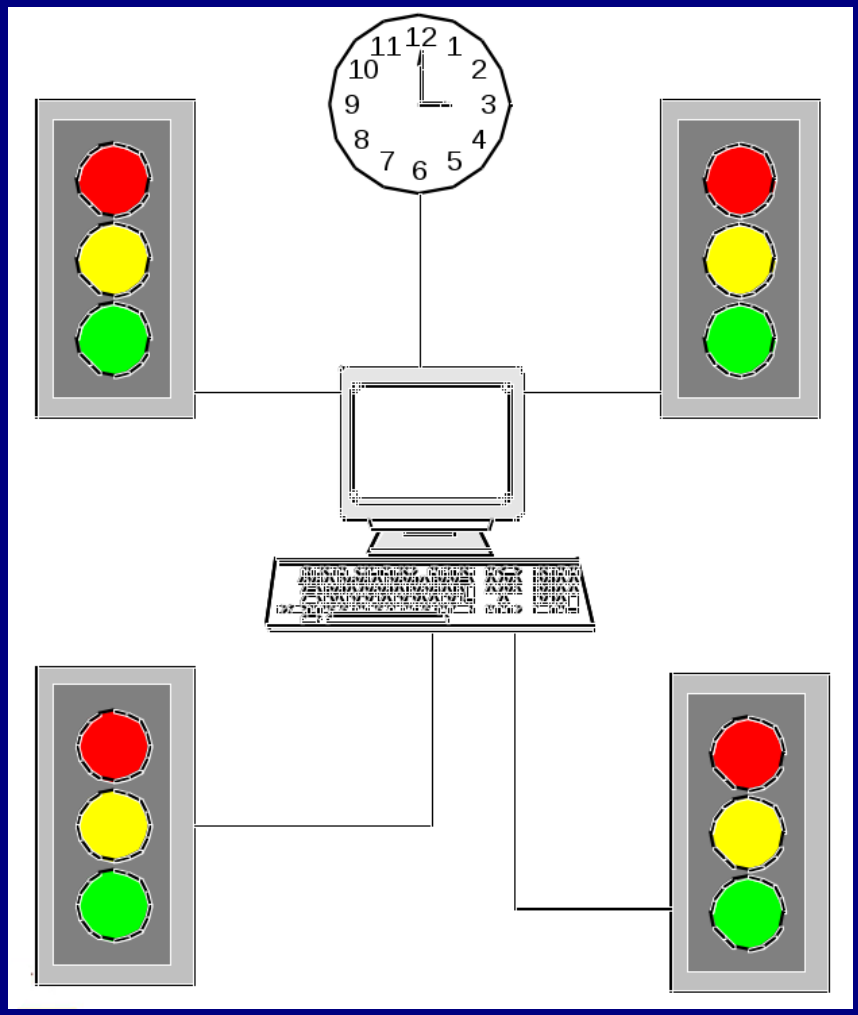
Dependability Qualities

- Correctness:
  - A program is correct if it is consistent with its specification
    - seldom practical for non-trivial systems, although always possible with the “proper” specification.
- Reliability:
  - likelihood of correct function for some “unit” of behavior
    - relative to a specification and usage profile
    - statistical approximation to correctness (100% reliable = correct)

Dependability Qualities

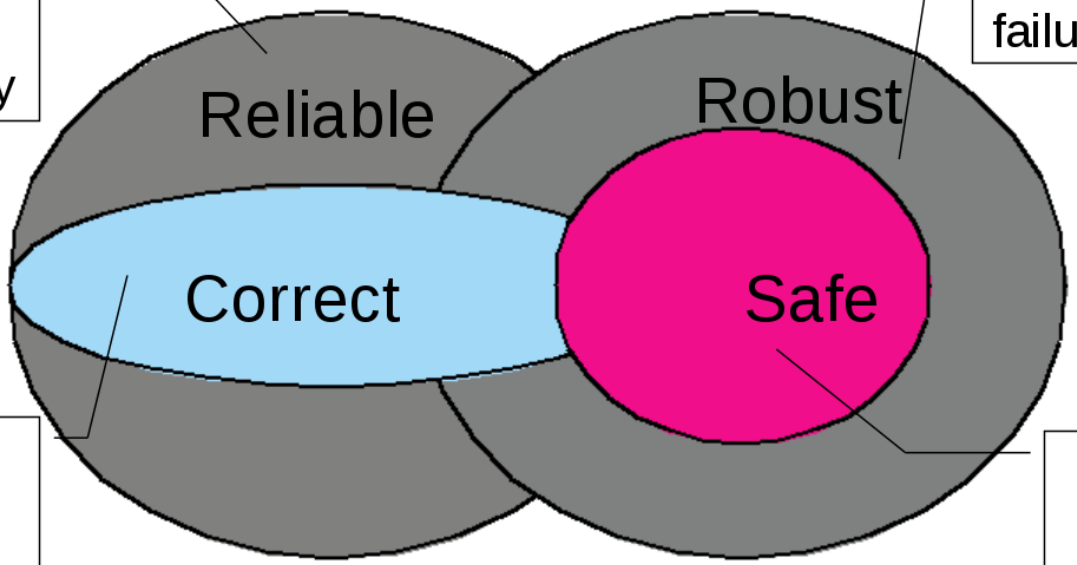
- Availability: Time the system is up.
  - e.g. router up 1 hour out of 24. Reliability is $\frac{23}{24} = 95.8\%$.
- MTBF: The mean time between failure
  - If the router is down once (in previous example), then MTBF = 23 hours. If twice then MTBF = 11.5 hours.
- Safety: Preventing/Minimizing undesirable effects/hazards.
  - Note that correctness, reliability, availability, and MTBF do not take into account the severity of the problem.
  - e.g. MS Word crashing vs losing your document.
  - Robustness
    - acceptable (degraded) behavior under extreme conditions
  - Note that safety attempts to prevent a hazard while Robustness is about failing gracefully.
  - e.g. An overloaded web server turning away users instead of denying all users/crashing.

Dependability quality Example



- Correctness, reliability: let traffic pass according to correct pattern and central scheduling
- Robustness, safety: Provide degraded function when possible; never signal conflicting greens.
  - Blinking red / blinking yellow is better than no lights; no lights is better than conflicting greens

Relation among Dependability Qualites



reliable but not correct: failures occur rarely

robust but not safe: catastrophic failures can occur

correct but not safe or robust: the specification is inadequate

safe but not correct: annoying failures can occur