Quality Attributes and Measurement

CSE 4495 - Lecture 2 - 21/06/2022

Today's Goals

- Discuss software quality in more detail.
    - Dependability, availability, performance, scalability, and security.
- How we build evidence that the system is good enough to release.
- How to **assess** whether each attribute is met.

Quality Attributes

- Describe **desired properties** of the system.
- Developers prioritize attributes and design system that meets chosen thresholds.
- Most relevant for this course: **dependability**
  - Ability to *consistently* offer correct functionality, even under *unforeseen* or *unsafe* conditions.

Quality Attributes

- **Availability**
  - Ability to carry out a task when needed, to minimize “downtime”, and to recover from failures.
- **Modifiability**
  - Ability to enhance software by fixing issues, adding features, and adapting to new environments.
- **Testability**
  - Ability to easily identify faults in a system.
  - Probability that a fault will result in a visible failure.

Quality Attributes

- **Performance**
  - Ability to meet timing requirements. When events occur, the system must respond quickly.
- **Security**
  - Ability to protect information from unauthorized access while providing service to authorized users.
- **Scalability**
  - Ability to “grow” the system to process more concurrent requests.

Quality Attributes

- **Interoperability**
  - Ability to exchange information with and provide functionality to other systems.
- **Usability**
  - Ability to enable users to perform tasks and provide support to users.
  - How easy it is to use the system, learn features, adapt to
    meet user needs, and increase confidence and satisfaction in usage.

Quality Attributes

- Resilience
- Supportability
- Portability
- Development Efficiency
- Time to Deliver
- Tool Support
- Geographic Distribution

Quality Attributes

- These qualities **often conflict**.
  - Fewer subsystems improves performance, but hurts modifiability.
  - Redundant data helps availability, but lessens security.
  - Localizing safety-critical features ensures safety, but degrades performance.
- Important to decide what is important, and set a threshold on when it is “good enough”.

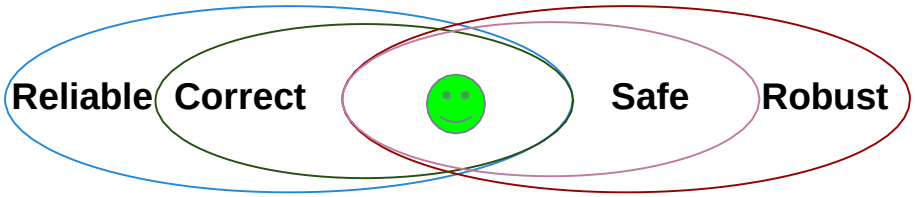
Quality Attributes

- Dependability
- Availability
- Performance
- Scalability
- Security
- **(Others important - but not enough time for all!)**

Dependability

When is Software Ready for Release?

- Provide evidence that the system is **dependable**.
- The goal of dependability is to establish four things about the system:
  - That it is **correct**.
  - That it is **reliable**.
  - That it is **safe**.
  - That is is **robust**.



Correctness

- A program is **correct** if it is always consistent with its specification.
- Depends on quality and detail of requirements.
  - Easy to show with respect to a weak specification.
  - Often impossible to prove with a detailed specification.
- Correctness is rarely provably achieved.

Reliability

- Statistical approximation of correctness.
- The likelihood of correct behavior from **some period of observed behavior**.
  - Time period, number of system executions
- Measured relative to a specification and usage profile (expected pattern of interaction).
  - Dependent on how the system is used by a type of user.

Dependence on Specifications

- Correctness and reliability:
  - Success relative to the strength of the specification.
    - **Hard to meaningfully prove anything for strong spec.**
  - Severity of a failure is not considered.
    - **Some failures are worse than others.**
- Safety revolves around **a restricted specification**.
- Robustness focuses on **everything not specified.**

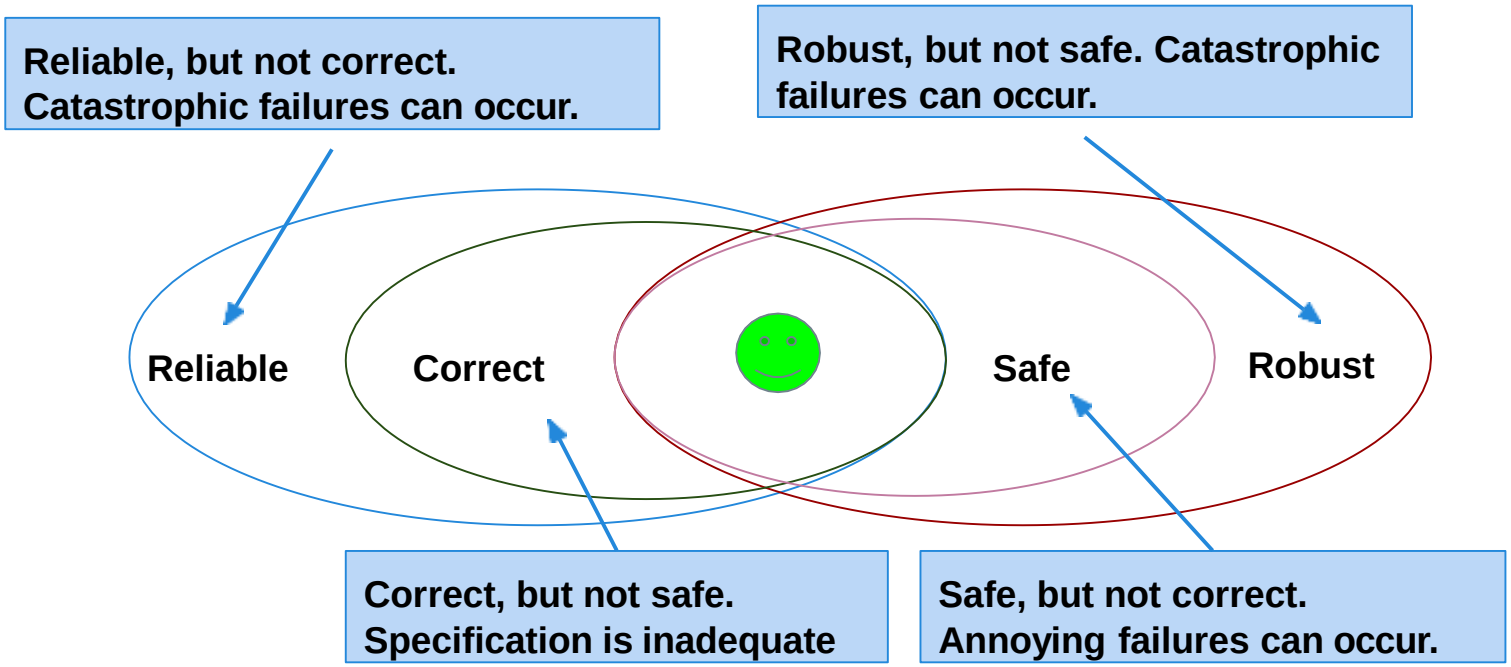
Safety

- Safety is the **ability to avoid hazards**.
  - Hazard = defined undesirable situation.
  - Generally serious problems.
- Relies on a specification of hazards.
  - Defines what the hazard is, how it will be avoided in the software.
  - We prove or show evidence that the hazard is avoided.
  - Only concerned with hazards, so proofs often possible.

Robustness

- Software that is “correct” may fail when the assumptions of its design are violated.
  - *How* it fails matters.
- **Software that “gracefully” fails is robust.**
  - Design the software to counteract unforeseen issues or perform graceful degradation of services.
    - Look at how a program could fail and handle those situations.
  - Cannot be proved, but is a goal to aspire to.

Dependability Property Relations



Measuring Dependability

- Must establish criteria for when the system is dependable enough to release.
  - Correctness hard to prove conclusively.
  - Robustness/Safety important, but do not demonstrate functional correctness.
- **Reliability is the basis for arguing dependability.**
  - Can be measured.
  - Can be demonstrated through testing.

Let’s take a break!

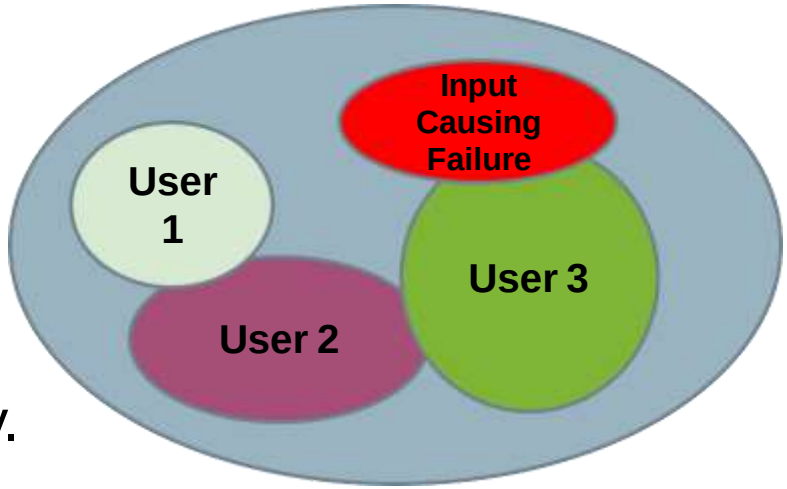
Measuring Reliability

What is Reliability?

- Probability of failure-free operation for a **specified time** in a **specified environment** for a **given purpose**.
  - Depends on system and type of user.
- How well users ***think*** the system provides services they require.

Improving Reliability

- **Improved when faults in the most frequently-used parts of the software are removed.**
  - Removing X% of faults != X% improvement in reliability.
    - In one study, removing 60% of faults led to 3% improvement.
  - Removing faults with serious consequences is the top priority.



Reliability is Measurable

- Reliability can be defined and measured.
- Reliability requirements can be specified:
  - Non-functional requirements define number of failures that are acceptable during normal use or time in which system is allowed to be unavailable.
  - Functional requirements define how the software avoids, detects, and tolerates failures.

How to Measure Reliability

- Hardware metrics often aren’t suitable for software.
  - Based on component failures and the need to repair or replace a component once it has failed.
  - In hardware, the design is assumed to be correct.
- Software failures are always design failures.
  - Often, the system is available even though a failure has occurred.
  - Metrics consider **failure rates**, **uptime**, and **time between failures**.

Metric 1: Availability

- Can the software carry out a task when needed?
  - Encompasses **reliability** and **repair**.
    - Does the system tend to show correct behavior?
    - Can the system recover from an error?
- The ability to mask or repair faults such that cumulative outages do not exceed a required value over a time interval.
  - **Both a reliability measurement AND an independent quality attribute.**

Metric 1: Availability

- Measured as **(uptime) / (total time observed)**
  - Takes repair and restart time into account.
  - Does not consider incorrect computations.
  - Only considers crashes/freezing.
  - 0.9 = down for 144 minutes a day.
    - 0.99 =14.4 minutes
    - 0.999 = 84 seconds
    - 0.9999 = 8.4 seconds

Availability

- Improvement requires understanding nature of failures that arise.
- Failures can be prevented, tolerated, removed, or forecasted.
  - How are failures detected?
  - How frequently do failures occur?
  - What happens when a failure occurs?
  - How long can the system be out of operation?
  - When can failures occur safely?
  - Can failures be prevented?
  - What notifications are required when failure occurs?

Availability Considerations

- Time to repair is the time until the failure is no longer observable.
  - Can be hard to define. Stuxnet caused problems for months. How does that impact availability?
- Software can remain partially available more easily than hardware.
- If code containing fault is executed, but system is able to recover, there was no failure.

Metric 2: Probability of Failure on Demand (POFOD)

- Likelihood that a request will result in a failure
- **(failures/requests over observed period)**
  - POFOD = 0.001 means that 1 out of 1000 requests fail.
- Used in situations where a failure is serious.
  - Independent of frequency of requests.
  - 1/1000 failure rate sounds risky, but if one failure per lifetime, may be good.

Metric 3: Rate of Occurrence of Fault (ROCOF)

- Frequency of occurrence of unexpected behavior.
- **(number of failures / total time observed)**
  - ROCOF of 0.02 means 2 failures per 100 time units.
  - Often given as “N failures per M seconds/minutes/hours”
- Most appropriate metric when requests are made on a regular basis (such as a shop).

Metric 4: Mean Time Between Failures (MTBF)

- Average length of time between observed failures.
  - Only considers time where system operating.
  - Requires the timestamp of each failure and the timestamp of when the system resumed service.
- Used for systems with long user sessions, where crashes can cause major issues.
  - E.g., saving requires resource (disc/CPU/memory) consumption.

Probabilistic Availability

- (alternate definition)
- Probability that system will provide a service within required bounds over a specified time interval.
  - **Availability = MTBF / (MTBF + MTTR)**
    - MTBF: Mean time between failures.
    - MTTR: Mean time to repair

Reliability Metrics

- Availability: **(uptime) / (total time observed)**
- POFOD: **(failures/ requests over period)**
- ROCOF: **(failures / total time observed)**
- MTBF: **Average time between observed failures.**
- MTTR: **Average time to recover from failure.**

Reliability Examples

- Provide software with 10000 requests.
  - Wrong result on 35 requests, crash on 5 requests.
  - What is the POFOD?
- Run the software for 144 hours
  - (6 million requests). Software failed on 6 requests.
  - What is the ROCOF? The POFOD?

Reliability Examples

- Provide software with 10000 requests.
  - Wrong result on 35 requests, crash on 5 requests.
  - What is the POFOD?
- 40 / 10000 = 0.0004
- Run the software for 144 hours
  - (6 million requests). Software failed on 6 requests.
  - What is the ROCOF? The POFOD?

Reliability Examples

- Provide software with 10000 requests.
  - Wrong result on 35 requests, crash on 5 requests.
  - What is the POFOD?
- 40 / 10000 = 0.0004
- Run the software for 144 hours
  - (6 million requests). Software failed on 6 requests.
  - What is the ROCOF? The POFOD?
- ROCOF = 6/144 = 1/24 = 0.04
- POFOD = 6/6000000 = (10^{-6})

Reliability Examples

- You advertise a piece of software with a ROCOF of
  1. 0.001 failures per hour.
     - However, it takes 3 hours (on average) to get the system up again after a failure.
     - What is availability per year?

Reliability Examples

- You advertise a piece of software with a ROCOF of 1. failures per hour.
  - However, it takes 3 hours (on average) to get the system up again after a failure.
  - What is availability per year?

- Failures per year:
  - approximately 8760 hours per year (24*365)
  - 0.001 * 8760 = 8.76 failures per year

Reliability Examples

- You advertise a piece of software with a ROCOF of 1. failures per hour.
  - However, it takes 3 hours (on average) to get the system up again after a failure.
  - What is availability per year?

- Failures per year:
  - approximately 8760 hours per year (24*365)
  - 0.001 * 8760 = 8.76 failures per year
- Availability
  - 8.76 * 3 = 26.28 hours of downtime per year.
  - Availability = 0.997 ((8760 - 26.28)/8760)

Additional Examples

- Want availability of at least 99%, POFOD of less than 0.1, and ROCOF of less than 2 failures per 8 hours.
  - After 7 full days, 972 requests were made.
  - Product failed 64 times (37 crashes, 27 bad output).
  - Average of 2 minutes to restart after each failure.
- What is the availability, POFOD, and ROCOF?
- Can we calculate MTBF?
- Is the product ready to ship? If not, why not?

Additional Examples

- Want availability of at least 99%, POFOD of less than 0.1, and ROCOF of less than 2 failures per 8 hours.
  - After 7 full days, 972 requests were made.
  - Product failed 64 times (37 crashes, 27 bad output).
  - Average of 2 minutes to restart after each failure.
- **ROCOF: 64/168 hours**
  - **= 0.38/hour**
  - **= 3.04/8 hour work day**

Additional Examples

- Want availability of at least 99%, POFOD of less than 0.1, and ROCOF of less than 2 failures per 8 hours.
  - After 7 full days, 972 requests were made.
  - Product failed 64 times (37 crashes, 27 bad output).
  - Average of 2 minutes to restart after each failure.
- **POFOD: 64/972 = 0.066**
- **Availability: Down for (37*2) = 74 minutes / 168 hrs**
  - **= 74/10089 minutes = 0.7% of the time = 99.3%**

Additional Examples

- Can we calculate MTBF?
  - No - need timestamps. We know how long they were down (on average), but not when each crash occurred.
- Is the product ready to ship?
  - No. Availability/POFOD are good, but ROCOF is too low.

Reliability Economics

- May be cheaper to accept unreliability and pay for failure costs.
- Depends on social/political factors and system.
  - Reputation for unreliability may hurt more than cost of improving reliability.
  - Cost of failure depends on risks of failure.
    - Health risks or equipment failure risk requires high reliability.
    - Minor annoyances can be tolerated.

Let’s take a break!

Self Reading: Quality Attributes- Performance and Scalability

Next Time

- More introduction:
  - Testing fundamentals.
  - Principles of analysis and testing.

- Reading:
  - Chapters 1-4 of testbook.