CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Lecture 2: Quality Attributes and Measurement

Gregory Gay

DIT636/DAT560 - January 22, 2025

When is Software Ready for Release?

Software is ready for release when you can argue that it shows sufficient quality.

- Requires choosing **quality attributes**.
  - Requires specifying **measurements** and **thresholds**.
  - May require different measurements and thresholds for **different functionality and execution scenarios**.
- Assessed through **Verification and Validation**.

Today's Goals

- Discuss quality attributes
  - Dependability, availability, performance, scalability.
- Discuss measurement of these attributes
  - How we build evidence that the system is "good enough".
  - How to assess whether each attribute is met.

Software Quality

- We all want **high-quality** software.
  - We don't all agree on the definition of quality.
- Quality encompasses **what** and **how**.
  - How *dependable* it is.
  - But also…
    - How *quickly* it runs.
    - How *available* its services are.
    - How easily it *scales* to more users.
- Hard to measure and assess objectively.

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

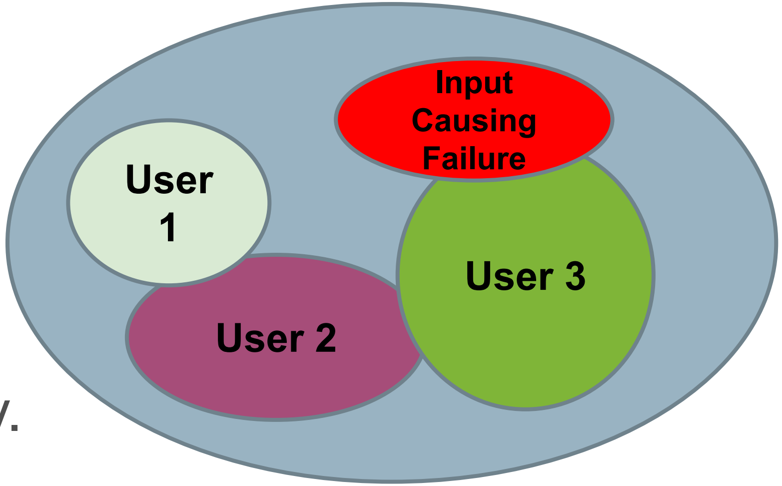
- **Availability**
  - Ability to carry out a task when needed, to minimize "downtime", and to recover from failures.
- **Performance**
  - Ability to meet timing requirements. When events occur, the system must respond quickly.
- **Scalability**
  - Ability to maintain dependability and performance as the number of concurrent requests grows.

Quality Measurement

- Quality is always measured situationally.
  - **Never quality of the whole system**, but of a component of the system.
    - Quality of a method, class, sub-system, API endpoint, user-facing function, …
  - Measured relative to **usage profile**.
    - Expected interaction pattern.

Improving Quality

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



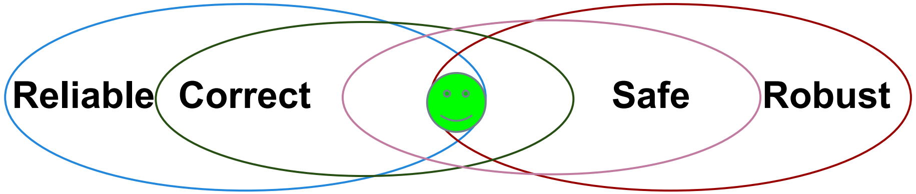
Quality Economics

- May be cheaper to accept a certain leave of quality and pay for failure costs.
- Depends on social/political factors and system.
  - Reputation versus cost of improvement.
  - Cost depends on risks of failure.
    - Health risks or equipment failure risk requires high quality.
    - Minor annoyances can be tolerated.

Quality Attribute: Dependability

Dimensions of Dependability

- 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 completeness of requirements.
    - Easy to show with a weak specification.
    - Often impossible with a detailed specification.
- Rarely *provably* achieved.

Reliability

- *Statistical approximation* of correctness.
  - The likelihood of correct behavior from **some period of observed behavior**.
    - Time period or number of system executions
  - Even if we cannot prove correctness, we can show that the system almost always works.
    - Testing can demonstrate reliability, but not correctness.

Dependence on Specifications

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

Safety

- Safety is the **ability to correctly handle hazards**.
  - Known undesirable situations.
  - Generally serious problems.
- Relies on a specification of hazards.
  - Defines each hazard, how it will be avoided or handled.
  - Prove that the hazard is avoided.
    - Only concerned with hazards, so proofs often possible.

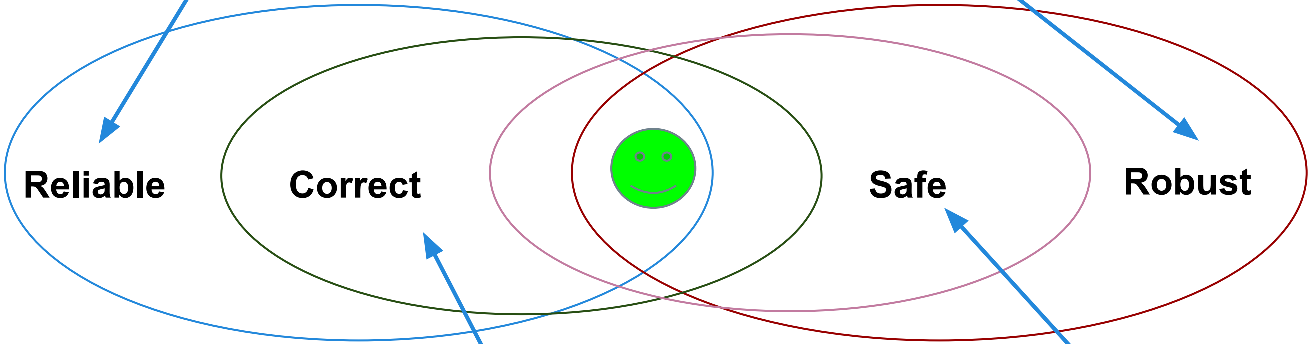
Robustness

- 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

Reliable, but not correct. Catastrophic failures can occur.

Robust, but not safe. Catastrophic failures can occur.

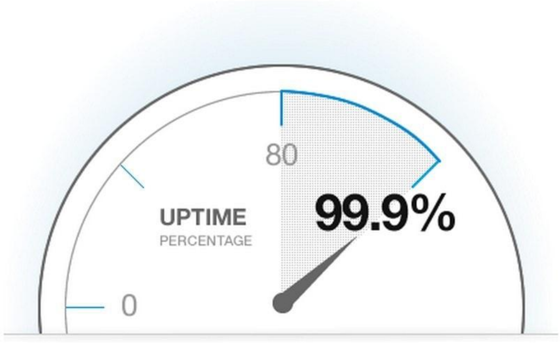


Correct, but not safe. Specification is inadequate

Safe, but not correct. Annoying failures can occur.

Assessing Dependability

- When is the system dependable enough?
  - Correctness hard to prove.
  - Robustness/Safety important, but do not demonstrate *normal* dependability.
- **Reliability is the basis for arguing dependability.**
  - Can be measured.
  - Can be demonstrated through testing.
  - Can reflect normal and abnormal usage.



Quality Attribute: Availability

Availability

- The ability to **carry out a task when needed**, and to **recover** or work around faults **when it fails**.
  - When a failure occurs, ensures system can recover.
  - System is seen as more reliable if failures can be corrected or masked before they affect the user.

Availability

- Failures can be prevented, tolerated, or repaired.
  - 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.
  - 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.

Measuring Reliability and Availability

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.
  - Design is assumed to be correct.
- Software failures are generally **design failures.**
  - System often available despite failure.
  - Metrics consider **failure rates**, **uptime**, and **time between failures**.

Measurement 1: Availability

- **(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 as a Measurement

- As part of **reliability**:
  - Measurement shows that system *generally* runs under normal circumstances.
- As a **standalone quality attribute**:
  - Measurement shows that, when a failure occurs, system can recover quickly.

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

Let’s take a break!

Reliability Metrics

- Availability: **(uptime) / (total time observed)**
- POFOD: **(failures/ requests over period)**
- ROCOF: **(failures / time elapsed in target unit)**
- MTBF: **Average time between observed failures**

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 24 hours
  - (6 million requests). Software failed on 6 requests.
  - What is the ROCOF in failure/hour? The POFOD?
- ROCOF = 6/24 = 0.25 failures per hour
- POFOD = 6/6000000 = (10^{-6})

Additional Examples

- Target: ROCOF < 0.3 per hour, POFOD < 0.1.
  - After 7 days, 972 requests were made.
  - Product failed 64 times (37 crashes, 27 incorrect output).
  - Average of 2 minutes to restart after each failure.

- **ROCOF: 64/168 hours**
  - **= 0.38/hour**
- **POFOD: 64/972 = 0.066**

Additional Examples

- Target: Availability >= 99%.
  - After 7 days, 972 requests were made.
  - Product failed 64 times (37 crashes, 27 incorrect output).
  - Average of 2 minutes to restart after each failure.
- **Availability: Down for (37*2) = 74 minutes / 10089 minutes = 0.7% of the time = 99.3%**
- Is the product ready to ship?
  - No. Availability/POFOD are good, but ROCOF is too high.

Quality Attributes: Performance and Scalability

Performance

- Ability to meet timing requirements.
  - When events occur, how fast does the system respond?
  - Captures performance-per-user and across-users.
  - Captures variance in performance.
- Driving factor in software design.
  - Often at expense of other quality attributes.
  - **All** systems have performance requirements.

Scalability

- Ability to maintain performance despite increasing number of requests.
  - Horizontal scalability (“scaling out”)
    - Adding more resources to logical units.
      - Adding another server to a cluster.
      - “elasticity” (add or remove VMs from a pool)
  - Vertical scalability (“scaling up”)
    - Adding more resources to a physical unit.
      - Adding memory to a single computer.

Scalability

- How can we effectively utilize additional resources?
- Requires that additional resources:
  - Result in performance improvement.
  - Did not require undue effort to add.
  - Did not disrupt operations.
- The system must be designed to scale
  - (i.e., designed for concurrency).

Measuring Performance and Scalability

Performance Measurements

- **Latency:** The time between the arrival of the stimulus and the system's response to it.
- **Response Jitter:** The allowable variation in latency.
- **Throughput:** Usually number of transactions the system can process in a unit of time.
- **Processing Deadlines:** Points where processing must have reached a particular stage.
- **Number of events not processed** because the system was too busy to respond.

Measurements - Latency

- Time it takes to complete an interaction.
- **Responsiveness** - how quickly system responds to routine tasks.
  - How responsive is the user's device? The system?
  - Measured probabilistically ("... 95% of the time")
  - "Under load of 350 updates per minute, 90% of 'open account' requests should complete within 10 seconds. 99% should complete within 12 seconds"

Measurements - Latency

- **Turnaround time** = time to complete larger tasks.
    - Can task be completed in available time?
    - Impact on system while running?
    - Can partial results be produced?
  - Ex: “With daily throughput of 850,000 requests, process must take a maximum of 4 hours, including writing to a database.”
  - Ex: “In 99% of cases, it must be possible to resynchronize monitoring stations and reset database within 5 minutes.”

Measurements - Response Jitter

- Response time is non-deterministic.
  - If controlled, this is OK.
    - 10s +- 1s, great!
    - 10s +- 10 minutes, bad!
- Jitter defines how much variation is allowed.
  - Ex: “All writes to the database must be completed within an interval of 120 to 150 ms.”

Measurements - Throughput

- The workload a system can handle in a time period.
  - Measures performance **across** all users.
  - Shorter the processing time, higher the throughput.
  - As load increases (and throughput rises), response time for individual transactions tends to increase.
    - With 10 concurrent users, request takes 2s.
    - With 100 users, request takes 4s.

Measurements - Throughput

- Throughput goals can conflict with latency goals.
  - For example:
    - With 10 users, each user can perform 20 requests per minute (throughput: 200/m).
    - With 100 users, each can perform 12 per minute (throughput is 1200/m but at a cost for individual user).

Measurements - Event Deadlines

- Some tasks must take place as scheduled.
- If times are missed, the system will fail.
- Deadlines place boundaries on event completion.
- Can also track how many input events are ignored because the system is too slow to respond.
    - Set limit on how many events can be missed over time.

Which response measure should we use?

- The pacemaker must shock the heart no more than 8ms after the last heartbeat.
  - Event deadline - there is an absolute limit in performance
- We want to make sure our web shop can handle Black Friday traffic.
  - Throughput - make sure all requests are handled in a short period of time.
  - May prioritize completing the batch over individual users.

Which response measure should we use?

- We want every user's transaction on the web shop to complete in a satisfying timeframe.
  - Latency
    - May choose to prioritize low latency over high throughput.
- We want to ensure that database updates are properly synchronized.
  - Response jitter.
    - Imposes minimum and maximum timeframe on updates.

Assessing Scalability

- Ability to address more requests is often part of **performance** or **reliability** assessment.
- Assessing scalability directly measures impact of adding or removing resources.
- Response measures reflect:
  - Changes to performance.
  - Changes to reliability or availability.
  - Load assigned to existing and new resources.

Key Points

- Dependability is one of the most important software characteristics.
  - Aim for correctness, reliability, safety, robustness.
  - Often assessed using reliability.
- Reliability depends on the pattern of usage of the software. Different users will interact differently.
- Reliability measured using ROCOF, POFOD, Availability, MTBF

Key Points

- Availability is the ability of the system to recover from a failure.
- Performance is about management of resources in the face of demand to achieve acceptable timing.
  - Usually measured in terms of throughput and latency.
- Scalability is the ability to “grow” the system to process an increasing number of requests.
  - While still meeting performance requirements.

Next Time

- Quality Scenarios
- No exercise session this week.

- **Form your teams!**
  - Deadline: January 26
  - Assignment 0 on Canvas

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY