CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Lecture 2: Quality Attributes and Measurement

Gregory Gay

DIT636/DAT560 - January 21, 2026

When is Software Ready for Release?

When you can argue that it shows *sufficient quality*.

- Requires choosing **quality attributes**.
  - … specifying **measurements** and **thresholds**.
  - … 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

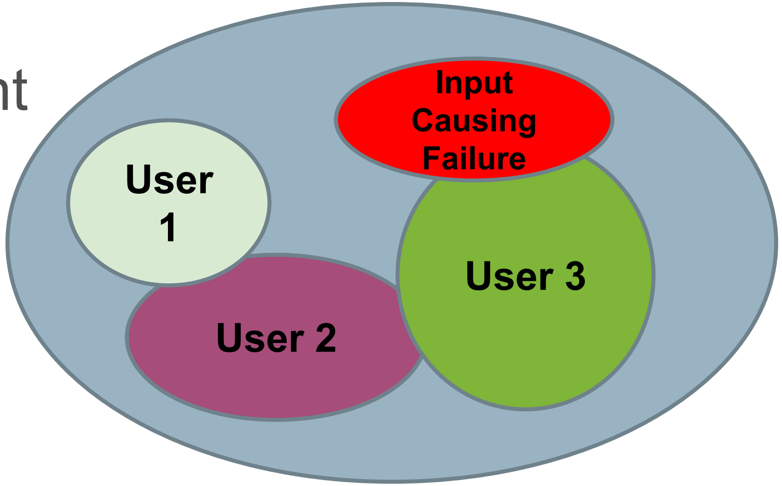
- **Availability**
  - Ability to carry out a task when needed, to minimize “downtime”, and to recover from failures.
- **Performance**
  - Ability to meet timing requirements. When events occur, the system must respond quickly.
- **Scalability**
  - Ability to maintain dependability and performance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

Quality Measurement

- Quality is always measured situationally.
  - **Not quality of the whole system, but of one "aspect"**
    - A class, sub-system, API endpoint, user-facing function, …
  - Relative to a **usage profile**.
    - Expected interaction patterns.

Improving Quality

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



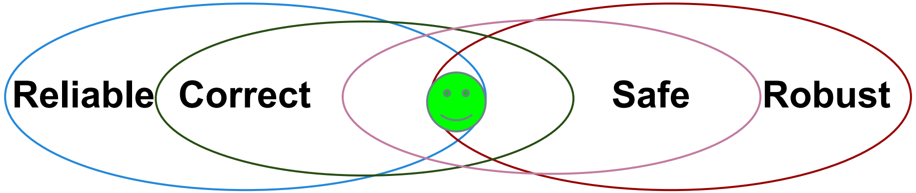
Quality Economics

- May be cheaper to accept a certain leave of quality and pay for failure costs.
- Depends on social/political factors and risks.
    - Reputation versus cost.
    - 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, number of executions
  - Even if we cannot prove correctness, we can show that the system ***almost always*** works.

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 **unspecified behaviors**.

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.
    - Subset of correctness, easier to prove.

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.

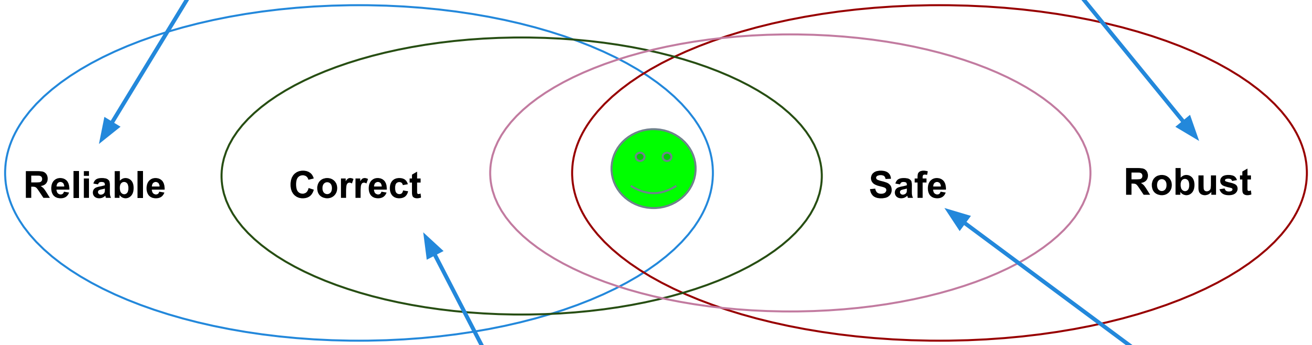
Examples - ATM

- We verify different situations where money is withdrawn ($ > balance, $ < balance, $ = balance, 0, …)
  - **Reliability**
- A thief may attach a “skimmer” to steal bank card details. We added a sensor and code to detect this.
  - **Safety**
- If the network connection is lost, we display an error screen and prevent any further actions from being taken.
  - **Robustness**

Dependability Property Relations

Reliable, but not correct. Catastrophic failures can occur.

Robust, but not safe. Catastrophic failures could occur, but measures have been put in place to potentially prevent issues.

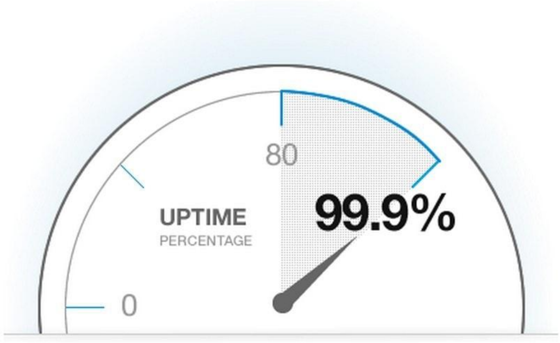


Correct, but not safe. Specification is inadequate

Safe, but not correct. Hazards avoided, but other 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

- Ability to **recover from - or work around - failures**.
  - After a failure occurs, ensure the 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

- System has "**recovered**" when 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: Time Available

- **(uptime) / (total time observed)**
  - Takes repair and restart time into account.
  - Does not consider incorrect computations.
    - Only crashes.
  - Keep an eye on digits of precision:
    - 0.9 = down for 144 minutes a day.
    - 0.99 =14.4 minutes
    - 0.999 = 84 seconds
    - 0.9999 = 8.4 seconds

Metric 2: Probability of Failure on Demand (POFOD)

- **(# failures) / (# requests)**
  - Likelihood that a request will fail
  - POFOD = 0.001 means that 1 out of 1000 requests fail.
- Used when every failure is serious.
  - Independent of frequency of requests.
  - 1/1000 sounds risky, but if only one failure in whole lifetime, may be good.

Metric 3: Rate of Occurrence of Fault (ROCOF)

- **(# failures) / (chosen period of time)**
  - Frequency of failures.
  - Often given as “X failures per Y seconds/minutes/hours”
    - You choose Y.
    - Often normalized to failures per minute, hour, day.
- Appropriate when requests are made on a regular basis (such as a shop).

Metric 4: Mean Time Between Failures (MTBF)

- **Average time between 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.

Measuring Availability

- How you **avoid**, **ignore**, or **recover** from failures.
  - *Avoid*:
    - Measure **reliability** when that failure *could* occur.
  - *Ignore*:
    - Induce failure, then measure subsequent **reliability**.
    - Compare to reliability when the failure did not occur.
  - *Recover:*
    - Measure **time** that it takes to return to normal operation.
    - Measure **effects of failure** on operation (e.g., like in “ignore”)

Let’s take a break!

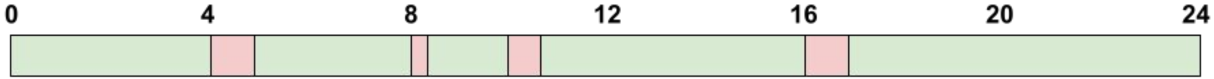
Reliability Metrics

- Time Available: **(uptime) / (total time observed)**
- POFOD: **(# failures) / (# requests)**
- ROCOF: **(# failures) / (period of time)**
- MTBF: **Average time between failures**

Activity

Recorded the following data:

- There were 1200 requests.
  - 60 of those requests resulted in failures.
  - 56 failures resulted in incorrect computations.
  - 4 failures resulted in crashes.
  - Uptime and downtime (caused by crashes):



Crash 1 (4:00): Down for 60 minutes
Crash 2 (8:00): Down for 15 minutes
Crash 3 (10:00): Down for 30 minutes
Crash 4 (16:00): Down for 60 minutes

Activity

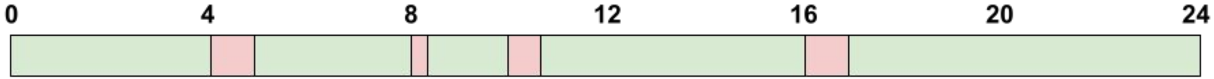
1. **What is the Time Available?**
   a. Total Time Observed =
      24 * 60 =
      1440 minutes
   b. Uptime =
      Total Time Obverved - Downtime
      1440 - (60 + 15 + 30 + 60) =
      1275 minutes
   c. Time Available = Uptime / Total Time Observed =
      1275 / 1440 =
      88.54%

Activity

2. **What is the POFOD?**
   a. POFOD = (# failures) / (# requests) =
      60 / 1200 =
      0.05
3. **What is the ROCOF in failures per hour?**
   a. ROCOF = (# failures) / (number of hours) =
      60 / 24 =
      2.5 failures per hour

Activity

4. **What is the MTBF (only crashes)?**
   a. We need the *uptime between each crash*.
      1 (0:00 - 4:00): 240 minutes
      2 (5:00 - 8:00): 180 minutes
      3 (8:15 - 10:00): 105 minutes
      4 (10:30 - 16:00): 330 minutes
   b. Take the average: (240 + 180 + 105 + 330) / 4 = 213.75 minutes



Crash 1 (4:00): Down for 60 minutes
Crash 2 (8:00): Down for 15 minutes
Crash 3 (10:00): Down for 30 minutes
Crash 4 (16:00): Down for 60 minutes

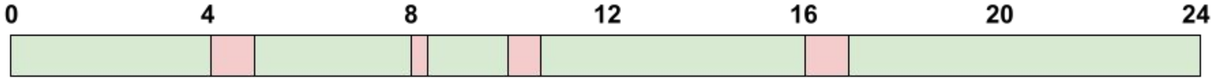
Activity

5. **If we were interested in assessing availability, what are some ways that we could do so?**
   - Pick failures we are interested in, show that they are *avoided*, *ignored*, *recovered* from.
     - *Avoid*: See if failure occurred.
     - *Ignored*: After failure occurs, is it likely to recur? Are other failures more likely to occur?
     - *Recovered*: How long was system down? Afterwards, is there an effect on reliability?

Activity

5. **If we were interested in assessing availability, what are some ways that we could do so?**

- Are crash 1 and 4 caused by the same failure?
- Did crash 2 lead to crash 3?



Crash 1 (4:00): Down for 60 minutes
Crash 2 (8:00): Down for 15 minutes
Crash 3 (10:00): Down for 30 minutes
Crash 4 (16:00): Down for 60 minutes

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 or reliability despite increasing number of requests.**
  - Horizontal scalability (“scaling out”)
    - Adding more resources to logical units.
      - Adding another server to a cluster.
  - 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 lower reliability.
- 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.
  - Affected by the **the system** and its **environment**.
    - The user’s hardware, the network, system’s hardware.
  - 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 - 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:
    - When there are 10 users, each user can perform 20 requests per minute (throughput: 200/m).
    - When there are 100 users, each can perform 12 per minute (throughput is 1200/m but at a cost for individual user).
      - i.e., performance is worse with more concurrent users.

Measurements - Event Deadlines

- Some tasks must take place as scheduled.
- If times are missed, the system will fail.
- **Can place deadlines on event completion.**

Which response measure should we use?

- We want every user’s transaction on the shop to complete quickly.
  - **Latency**
- Can our shop 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 in this situation.

Which response measure should we use?

- The user must sign with BankID and confirmation must be returned within 60 seconds.
  - **Deadline** - there is an absolute deadline for BankID processing to complete.
- Ensure that inventory database updates are properly synchronized.
  - **Jitter** - Imposes minimum and maximum timeframe on updates.

Assessing Scalability

- Scalability measures impact of adding or removing resources on **performance** or **reliability**.
- 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, Time Available, MTBF

Key Points

- Availability is the ability of the system to avoid, ignore, or 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 25
  - Assignment 0 on Canvas

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY