



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

Lecture 2: Quality Attributes and Measurement

Gregory Gay
DIT635 - January 22, 2021

Today's Goals

- Discuss software quality in more detail.
 - **Quality attributes**
 - 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.

Software Quality

- We all want **high-quality** software.
 - We don't all agree on the definition of quality.
- Quality encompasses both **what** the system does and **how** it does it.
 - How *quickly* it runs.
 - How *secure* it is.
 - How *available* its services are.
 - How easily it *scales* to more users.
- Quality is 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

- **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

- **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

- **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.

Other 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”.

Our Focus

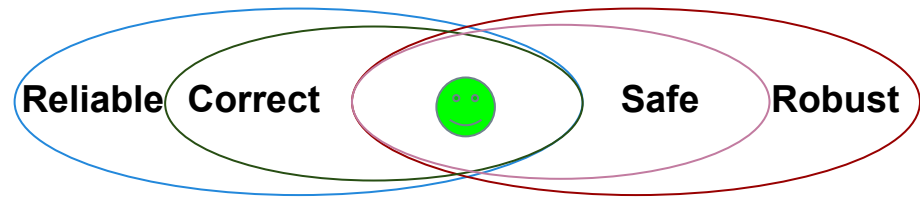
- Dependability
- Availability
- Performance
- Scalability
- Security
- **(Others important - but not enough time for all!)**

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 it 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 revolves around **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.

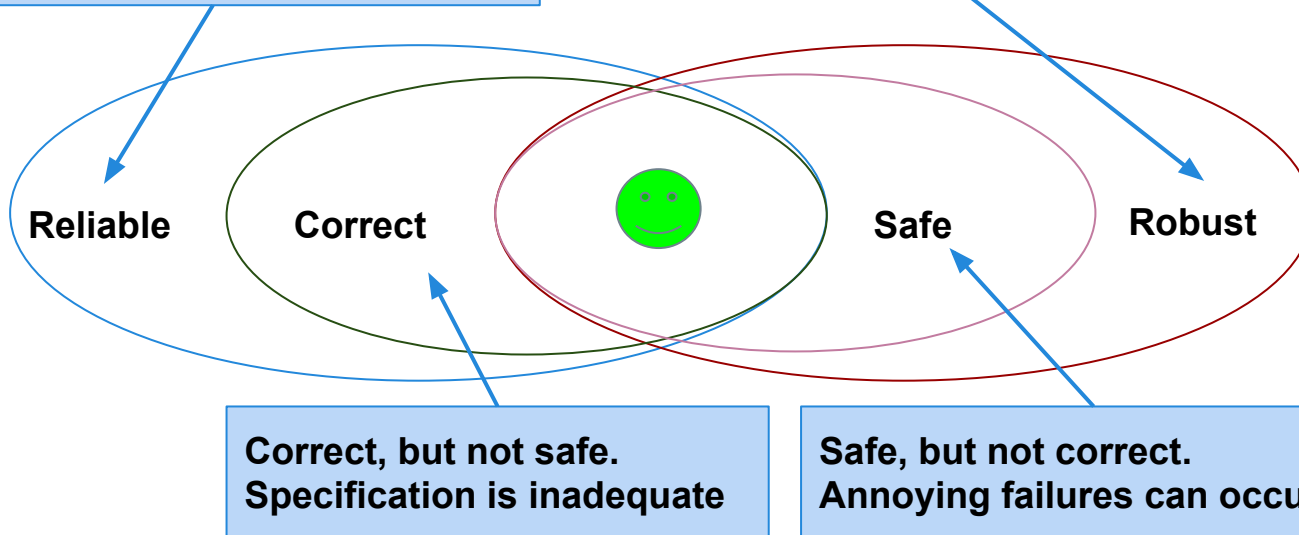
Robustness

- 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

**Reliable, but not correct.
Catastrophic failures can occur.**

**Robust, but not safe. Catastrophic
failures can occur.**



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 sufficient volume of testing.

Let's take a break!

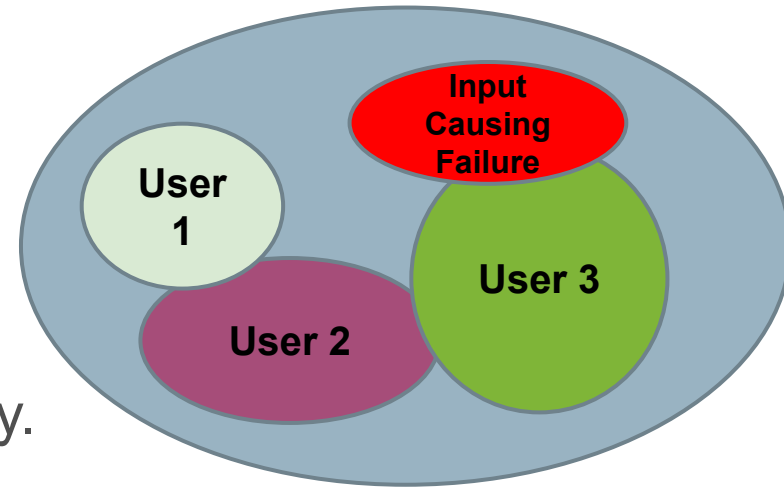
Measuring Reliability

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 \neq 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)**
 - $\text{POFOD} = 0.001$ 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?
- $40 / 10000 = 0.0004$
- 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 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?
- **Failures per year:**
 - approximately 8760 hours per year (24×365)
 - $0.001 \times 8760 = 8.76$ failures per year
- **Availability**
 - $8.76 \times 3 = 26.28$ 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!

Quality Attributes: Performance and Scalability



Performance

- Ability to meet timing requirements.
- Characterize pattern of input events and responses
 - Requests served per minute.
 - Variation in output time.
- Driving factor in software design.
 - Often at expense of other quality attributes.
 - **All** systems have performance requirements.

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.
- **Deadlines in processing:** 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.
 - Key consideration: user productivity.
 - How responsive is the user's device? The system?
 - Measured probabilistically (... 95% of the time)
 - Under a load of 350 update transactions per minute, 90% of "open account" requests should return a reply to the calling program within 10 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: Assuming a daily throughput of 850,000 requests, the process should take no longer than 4 hours, including writing results to a database.
 - Ex: It must be possible to resynchronize monitoring stations and reset database within 5 minutes.

Measurements - Response Jitter

- Response time is non-deterministic.
 - If non-determinism can be controlled, this is OK.
 - 10s +- 1s, great!
 - 10s +- 10 minutes, bad!
- Defines how much variation is allowed.
 - Places boundaries on when task can be completed.
 - If boundaries violated, quality is compromised.
 - Ex: “All writes to the database must be completed within 120 to 150 ms.”

Measurements - Throughput

- The workload a system can handle in a time period.
 - 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

- Possible to end up in situation where throughput goals conflict with response time goals.
 - With 10 users, each can perform 20 request per minute (throughput: 200/m).
 - With 100 users, each can perform 12 per minute (throughput: 1200/m - but at cost to response time).

Measurements - Deadlines

- Some tasks must take place as scheduled.
- If times are missed, the system will fail.
 - In a car, fuel must ignite when cylinder is in position.
 - Places a deadline on when the fuel must ignite.
- Deadlines can be used to place boundaries on when events must complete.

Measurements - Missed Events

- If the system is busy, input may be ignored.
 - Or, queued until too late to matter.
- Can track how many input events are ignored because the system is too slow to respond.
 - Set upper bound on how many events can be missed in a defined timeframe.

Scalability

- Ability to process increasing number of requests.
 - While meeting performance requirements.
- 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).

Assessing Scalability

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

Quality Attribute: Security

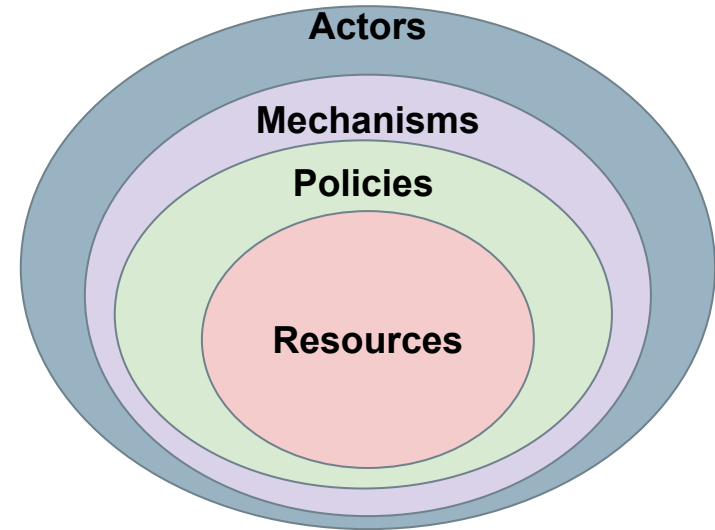


Security

- Ability to protect data and information from unauthorized access...
 - **... while still providing access to people and systems that are authorized.**
- Can we protect software from attacks?
 - Unauthorized access attempts.
 - Attempts to deny service to legitimate users.

Security

- Processes that allow owners of resources to control access.
 - Who: Actors (systems or users).
 - Resources are sensitive elements, operations, and data of the system.
 - Policies define legitimate access to resourced.
 - Enforced by security mechanisms used by actors to gain access to resources.



Security Characterization (CIA)

- Confidentiality
 - Data and services protected from unauthorized access.
 - A hacker cannot access your tax returns on an IRS server.
- Integrity
 - Data/services not subject to unauthorized manipulation.
 - Your grade has not changed since assigned.
- Availability
 - The system will be available for legitimate use.
 - A DDOS attack will not prevent your purchase.

Supporting CIA

- Authentication - Verifies identities of all parties.
- Nonrepudiation - Guarantees that sender cannot deny sending, and recipient cannot deny receiving.
- Authorization - Grants privilege of performing a task.



Security Approaches

- Achieving security relies on:
 - Detecting attacks.
 - Resisting attacks.
 - Reacting to attacks.
 - Recovering from attacks.
- Objects being protected are:
 - Data at rest.
 - Data in transit.
 - Computational processes.



Security is Risk Management

- **Not simply secure/not secure.**
 - All systems will be compromised.
 - Try to avoid attack, prevent damage, and quickly recover.
 - Balance risks against cost of guarding against them.
 - Set realistic expectations!



Assessing Security

- Measure of system's ability to protect data from unauthorized access while still providing service to authorized users.
- Assess how well system responds to attack.
 - Stimuli are attacks from external systems/users or demonstrations of policies (log-in, authorization).
 - Responses: auditing, logging, reporting, analyzing.

Assessing Security

- No universal metrics for measuring “security”.
- Present specific attack types and specify how system responds.
- Response assessed by appropriate metrics.
 - Time to identify attacker.
 - Amount of data protected.
 - Time to stop attack.

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 be available for use, especially after 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.

Key Points

- Security is the ability to protect data and information from unauthorized access...
 - ... while still providing access to people and systems that are authorized.
- Security is not “measured”, but requires defining attacks and actions to prevent or reduce impact of risk, then assessing those actions.

Next Time

- Quality Scenarios
 - Assessing whether quality thresholds are met.
- No exercise session today!
- **Form your teams!**
 - Deadline: January 26
 - E-mail me (ggay@chalmers.se) with list of team members.
 - Or e-mail if you want to be assigned to a team



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY