CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Lecture 2: Quality Attributes and Measurement

Gregory Gay

DIT636/DAT560 - January 17, 2024

Today's Goals

- Discuss software quality in more detail.
  - Focus on dependability, availability, performance, scalability, security.
  - 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 both **what** and **how**.
  - 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 important: **dependability**
  - Ability to *consistently* offer correct functionality, even under *unforeseen* or *unsafe* conditions.

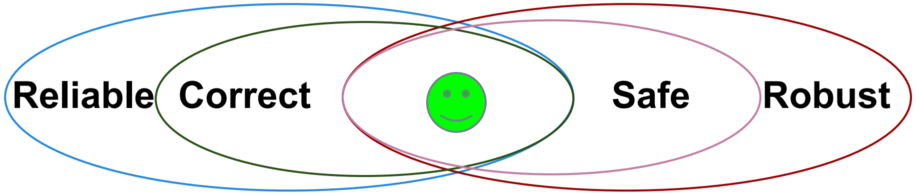
Quality Attributes

- **Availability**
  - Ability to avoid or recover from failures.
- **Performance**
  - Ability to meet timing or throughput requirements.
- **Scalability**
  - Ability to scale performance to available resources.
- **Security**
  - Ability to protect information from unauthorized access while providing service to authorized users.

Quality Attribute: 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 complexity of requirements and system.
  - 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 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 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**.
  - Hazard = a defined undesirable situation.
  - 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.

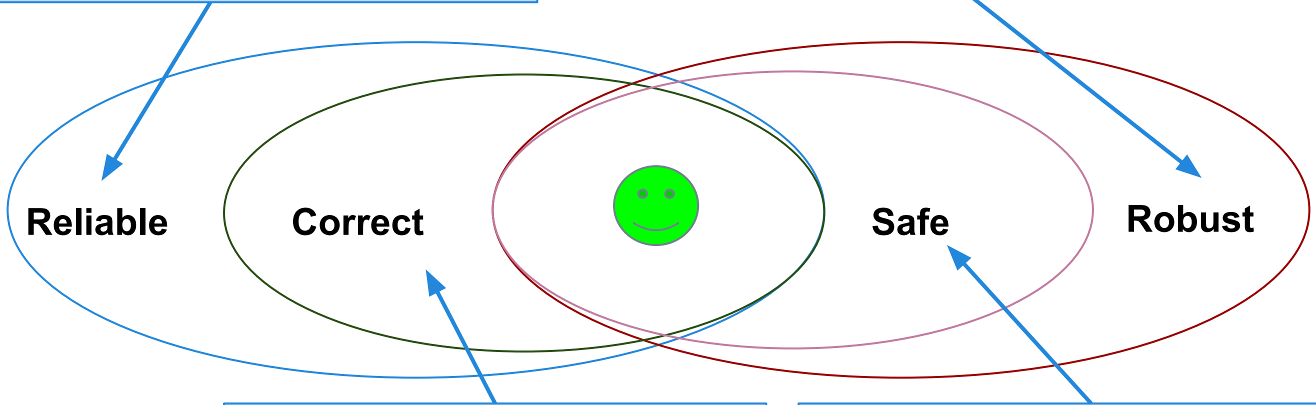
Robustness

- 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.

Measuring 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.

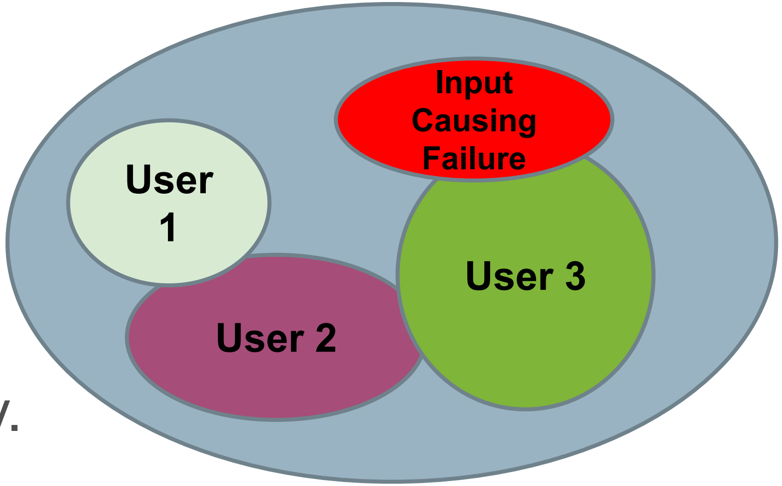
Reliability and Availability

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.

Improving Reliability

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



Reliability is Measurable

- Reliability can be defined and measured.
- Reliability requirements can be specified:
  - NFRs define numeric thresholds.
    - Number of failures that are acceptable.
    - How long system can 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.
  - 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**.

Availability

- Can the software carry out a task when needed?
  - As part of **reliability**:
    - A measurement of uptime over a period of observation.
    - Shows that system *generally* runs under normal circumstances.
  - As a **standalone quality attribute**:
    - The ability to mask or repair faults such that cumulative outages do not exceed a required value over a time interval.
    - Shows that, when a failure occurs, system can recover quickly.

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

Measurement 1: Availability

- **As a standalone quality attribute**: Ability to avoid or recover from failures.
  - Failures can be prevented, tolerated, or recovered from.
    - 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.

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**

menti.com,
code 1885 2744

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: Availability >= 99%, POFOD < 0.1, ROCOF < 0.3 per hour.
  - 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.
- What is the availability, POFOD, and ROCOF?
- Is the product ready to ship? If not, why not?

Additional Examples

menti.com, code 1885 2744

- 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

menti.com,
code 1885 2744

- 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.

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.

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.
- **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.
  - Key consideration: user productivity.
  - 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."

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 should take < 4 hours, including writing 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 controlled, this is OK.
    - 10s +- 1s, great!
    - 10s +- 10 minutes, bad!
- Defines how much variation is allowed.
  - 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

- Throughput can conflict with response time.
    - With 10 users, each 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.

Scalability

- Ability to process 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).

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.

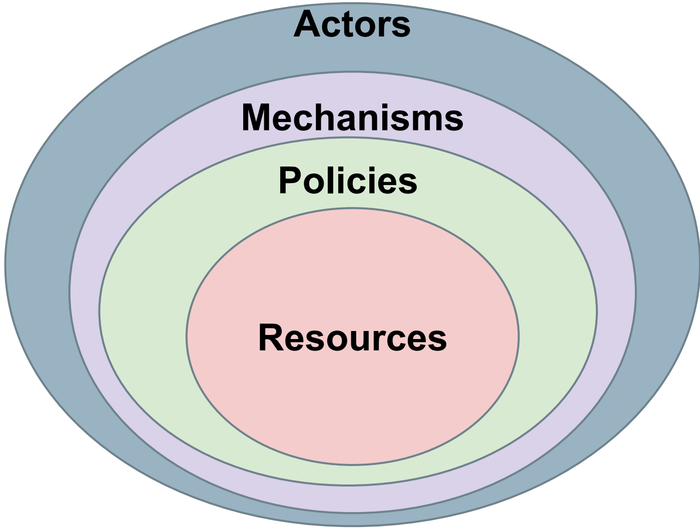
Quality Attribute: Security

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 allow owners of resources to control access.
    - Actors are systems or users.
    - Resources are sensitive elements, operations, and data of the system.
    - Policies define legitimate access to resources.
        - Enforced by security mechanisms used by actors to access 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
- No exercise session this week.

- **Form your teams!**
  - Deadline: January 21
  - Assignment 0 on Canvas

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY