Dependability properties

Barbara Russo

SwSE - Software and Systems Engineering Research Group

Quality process

- Dependability goals and properties are concerns of Quality Process (QP)
- QP is a means to fulfill required, desired, specific functional and non-functional properties that are called Quality Goals

Quality process

- A QP is needed
  - Even when specifications do no state specific quality properties of a system there is always a degree of quality in place to satisfy
    - E.g., modern development requires a certain coverage with unit testing

Quality goals

- Improving software product during and after development
- Assessing software product quality before delivery
- Improving quality process within and across projects

Quality goals

- Product goals are goals of software quality engineering
- Process goals are means to achieve product goals
- To achieve goals we need to
  - Make them explicit
  - E.g., discover them during a feasibility study

Quality goals

- Product
  - Internal qualities
  - External qualities
    - Usefulness qualities
    - **Dependability qualities**

Dependability qualities

- We have seen:
  - Correctness
  - Reliability
  - Robustness
  - Safety

- Specification Self-Consistency

Correctness

- System behaviour can be only successful or failing
  - Example: a program cannot be 30% correct
  - A program is correct if all its possible behaviours are successes
- A (hardware/software) system is correct if all its sub-parts (e.g., sub-systems, components, external libraries, devices) behave correctly

Correctness

- What is a successful behavior of a system?
- A system is **correct** if it is **consistent with all its specifications**
  - Specifications can be very badly defined!
  - Correctness is seldom practical
  - Failures might not yet be known: zero-days vulnerabilities

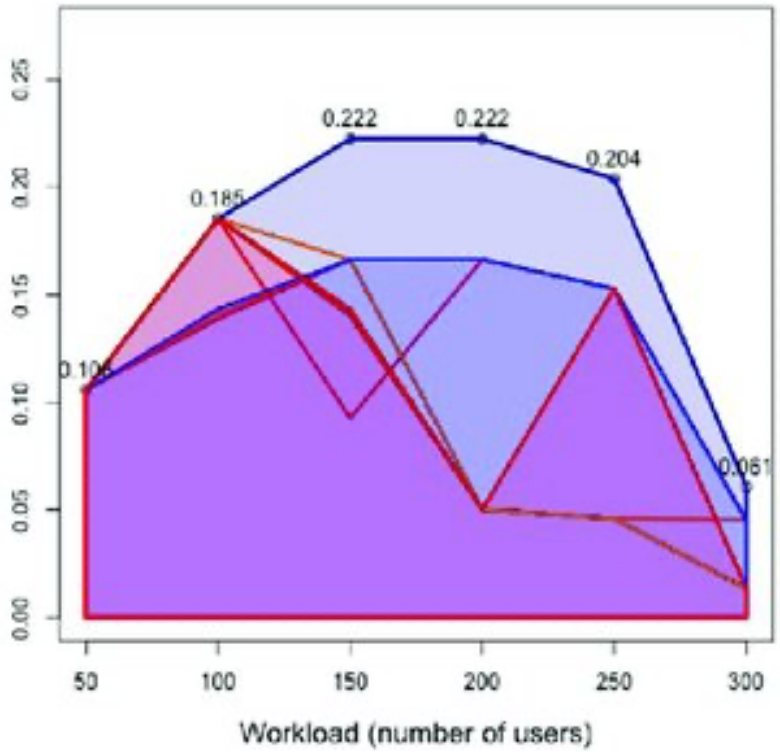
Reliability

- **Statistical approximation to correctness**: probability that a system deviates form the expected behaviour
  - Likelihood against given specifications
- Unlike correctness it is defined **against an operational profile of a software system**

Operational profile

- *The probability that a given number of users (workload intensity) would access a system/ functionality/service/operation concurrently*
- It is a quantitative characterization of how a system will be used
  - It shows how to increase productivity and reliability and speed development by allocating development resources to function on the basis of use

Operational profile - system level

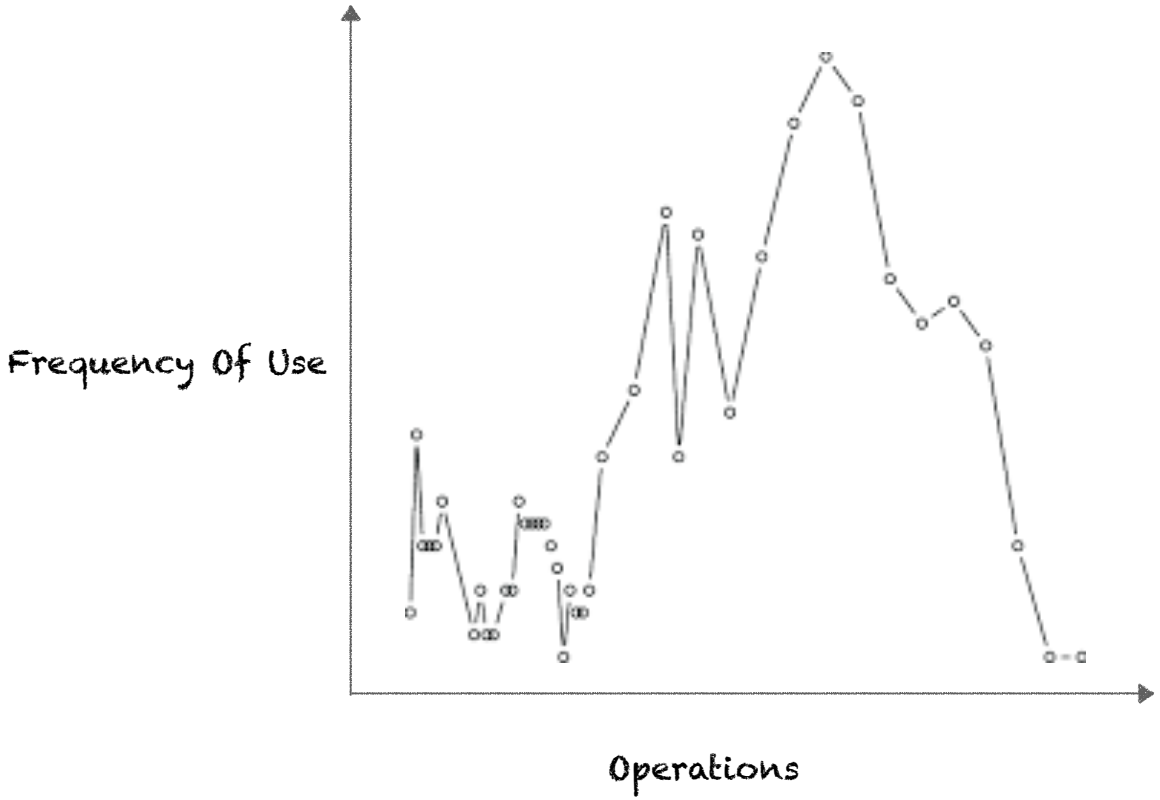


Configuration (α)		
Memory	CPU	# Replicas
8	**0.25**	**4**
16	0.25	2
8	0.25	2
16	0.25	4
8	0.25	1
16	0.5	4
8	0.5	2
16	0.5	4
8	0.5	1
16	0.5	1
8	0.5	2
16	0.25	1

Operational profile

- Example: to guide verification:
  - if the verification is terminated and system is deployed or the software is released because of schedule constraints, **the most-used operations will have received the most testing** and the reliability level will be the maximum that is practically achievable for the given test time

Operational profile - operation level



Major Measures of Reliability

- **Availability:** the portion of time in which the software operates with no down time
- **Time Between Failures:** the time elapsing between two consecutive failures
- **Cumulative number of failures:** the total number of failures occurred at time

Robustness

- A system **maintains operations** under exceptional circumstances
- It fails "softly" outside its normal operating parameters
- It is "fault tolerant"
  - Despite faults, it operates

Example

- **Unusual circumstance:** unforeseen (not in the specifications) load of users accessing a web site
- **Robust software:**
- A workaround: Maintain the same throughput speed while stopping last arrived users until the load is decreased
    - It does not decrease performance for registered users

Example

- **Action to be taken to increase robustness:** Augment software specifications with appropriate responses to given unusual circumstances (enrich the operational profile with unlikely situations)

Safety

- Robustness in case of hazardous behaviour (hazard)
- A **hazard** is **any agent that can cause harm or damage to humans, property, or the environment**
- Safety is very focused on functionalities that can determine hazards, not concerned with other issues on functionalities

Safety

- Often needed in critical systems, but not only:
  - Word crashes -> reliability or robustness
  - Word crashes and corrupts existing files -> safety

Hazard

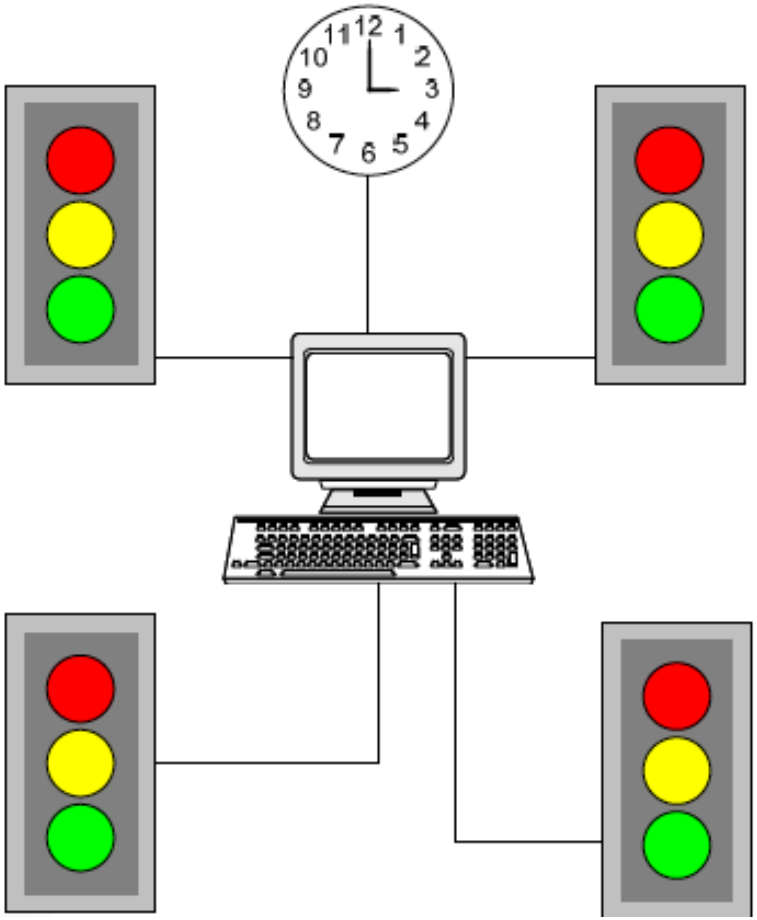
- Safety is meaningless without a specification of hazards
- It is important to identify and classify hazards for the given software

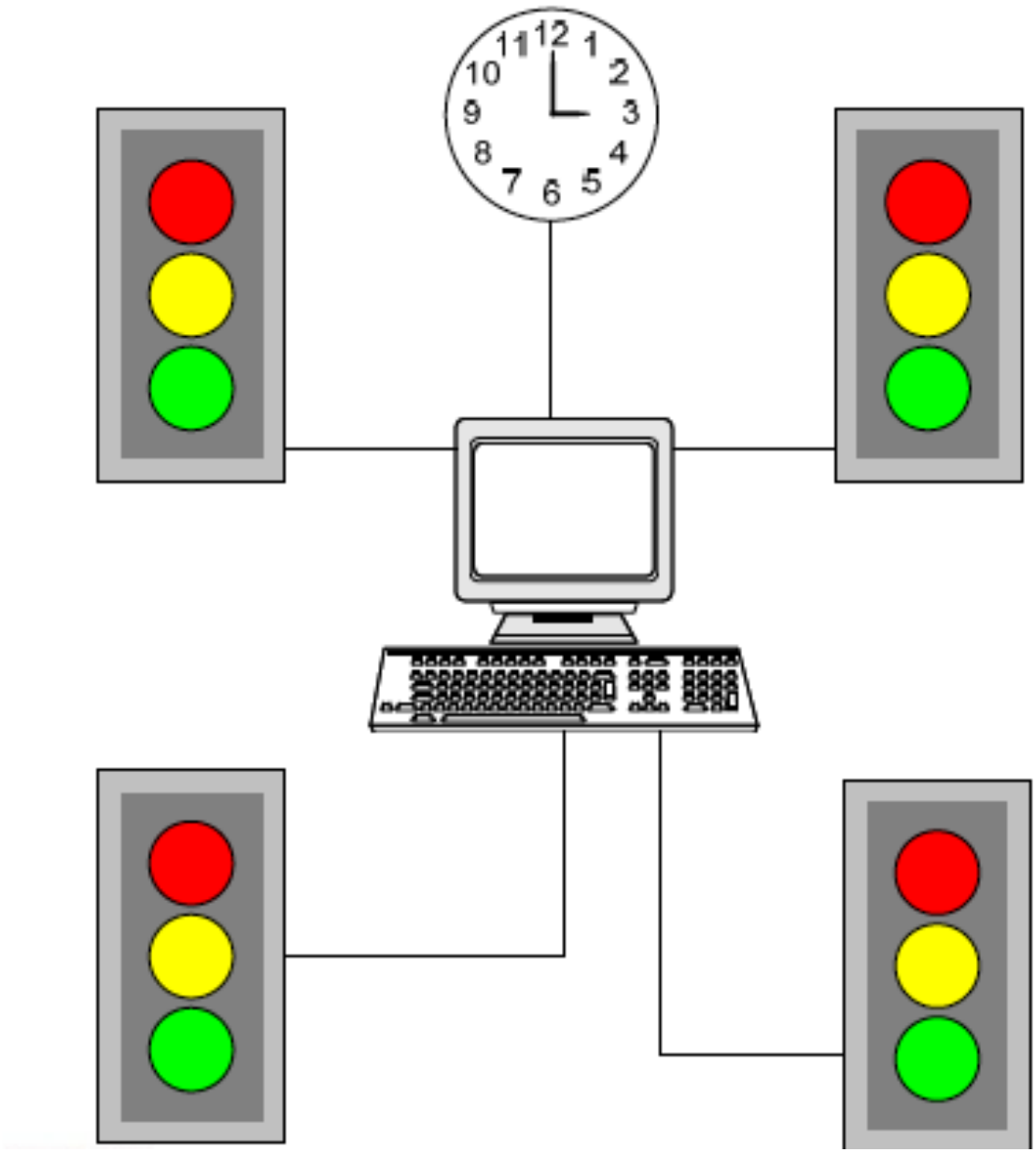
Hazard - how to detect them

- Do it in the embedded environment (often hazards are related to specific environmental circumstances)
- Separate the analysis of hazards from any other verification activity

Exercise

- Discuss the dependability properties of a traffic light system and classify its hazards

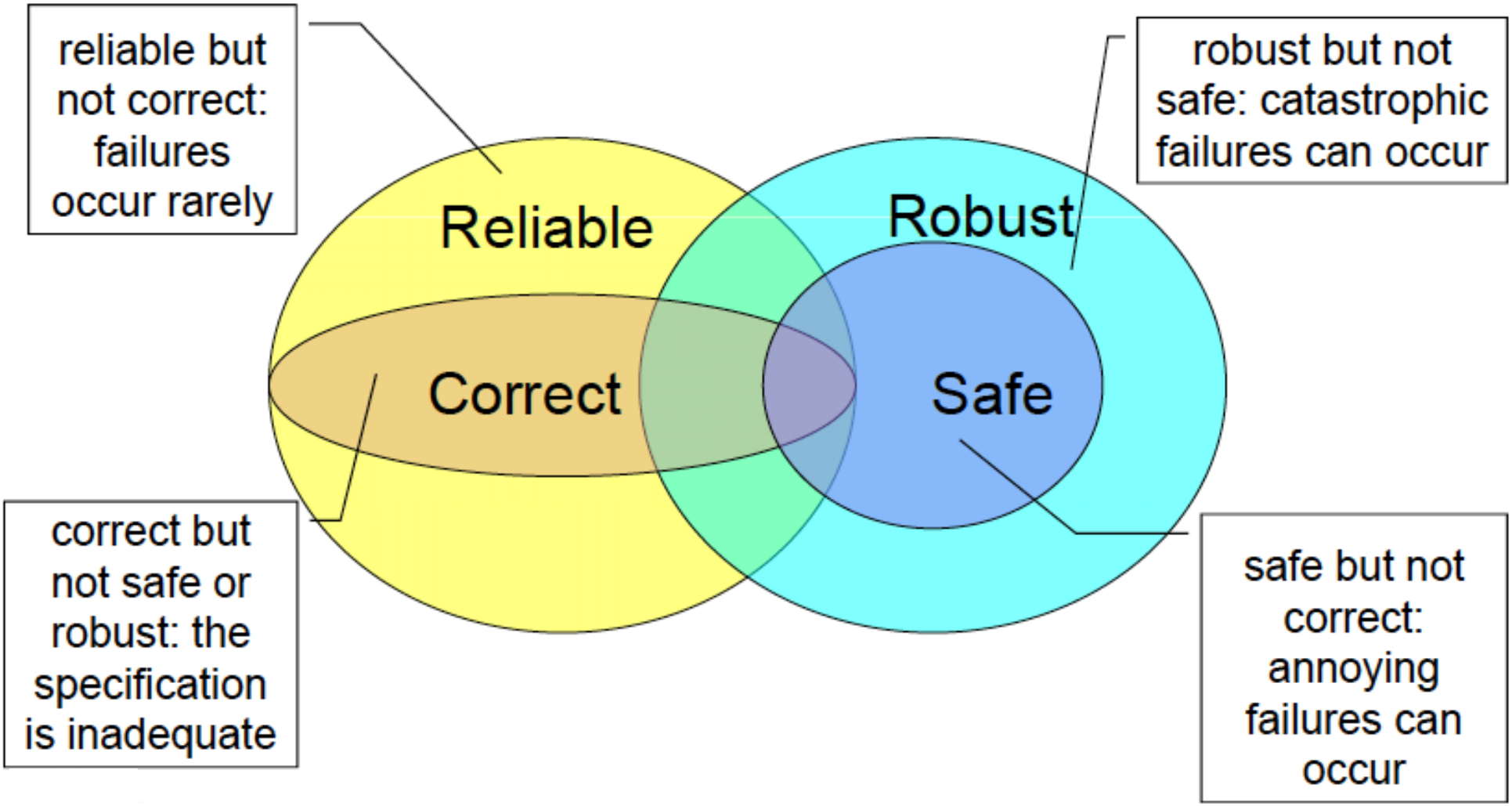




- **Reliability:** built according to central scheduling and practice
- **Robustness, safety:** degraded function when possible; never signal conflicting greens
  - Blinking red / blinking yellow is better than no lights;
  - No lights is better than conflicting greens

Source: Mauro Pezze' and Michal Young

Relations



Source: Mauro Pezze' and Michal Young

Exercise

- Find examples of
  - correct but not safe,
  - reliable but not correct,
  - robust but not safe,
  - safe but not correct

Which type of issue is this?

Problem Statement:
For many downloads (especially large files) the publisher often releases MD5 checksums in order to assure that the download arrives as intended. The problem is that this checksum often requires a manual check by the recipient of the download. The recipient is often unwilling or unable to do this verification and continues on assuming the download is good, without actually checking.

Relates to Spec section:
http://dev.w3.org/html5/spec/links.html#links-created-by-a-and-area-elements
http://dev.w3.org/html5/spec/text-level-semantics.html#the-a-element

Possible Solution:
A possible solution would be to include the checksum as a machine readable attribute or tag that specifies the checksum and algorithm for the user agent to verify the download after the download finishes, giving instant feedback to the user.

Exercise

- Analyse an issue report and classify it in terms of the dependability properties.
- Example: SpringFramework Jira
  - testBindInstantFromJavaUtilDate fails on systems in the Pacific/Auckland time zone [SPR-16534]
  - https://jira.spring.io/browse/SPR-16534?jql=project%20%3D%20SPR%20AND%20created%3E%3D-1w%20ORDER%20BY%20created%20DESC

Security

- Reflects a system's ability to protect itself from attacks
- Security is increasingly important when systems are networked to each other
- Security is an essential pre-requisite for reliability and safety

Effects of security

- If a system is networked and insecure then statements about its reliability and safety are unreliable:
  - Intrusion (attack) can change the system's operating environment or data and invalidate the assumptions (specifications) upon which the reliability and safety are made

R. Alguliyev, Y. Imamverdiyev, and L. Sukhostat, "Cyber-physical systems and their security issues,"Computers in Industry, vol. 100, pp. 212-223, sep 2018

Examples of insecurity damages

- Denial of Service
  - system forced into state where providing service is impossible or significantly degraded
- Corruption of Programs or Data
  - modifications made by unauthorised user
- Disclosure of Confidential Information
  - information managed by system is exposed to people who are not authorised users

Attack type	Description
Denial of Service (DoS)	Blocking traffic in order to make network and service unavailable (e.g. flooding with false requests)
Man-in-the-Middle (MITM)	Sending a modified message to a target on order to take, from the system creator point of view, undesired function
Eavesdropping	Intercepting any transmitted data by the system
Spoofing	Pretending to be a part of the system in order to get involved in system activities
Reply (Playback)	Re-transmitting received packet from the destination node in order to gain system's trust
Compromised Key	The key that secures communication is the target of the attack
Node capture	Taking over a node to leak information that could include encryption keys and threatening the security of the whole system using it
False Node	Adding an additional node to the network to attack data integrity by sending malicious data
Node Outage	Stopping node services to affect availability and integrity
Path-Based DOS	Over-flooding a routing path to reduce the nodes availability
Resistance	Forcing compromised sensors and controllers to start to operate at a different resonant frequency
Integrity	Injecting false sensor measurements to disrupt the system external control inputs
Routing	Introducing routing loops that cause transmitting delay or extended source paths
Wormhole	Announcing false paths through which all packages are routed in order to make information holes
Jamming	Introducing noise in a wireless channel between a sensor nodes and remote base station
Selective Forwarding	A compromised node selects certain packages and forwards them while drops and discards the others
Sinkhole	Announcing the best routing path that actually leads to another nodes
Buffer Overflow	Exploiting the opportunities of any vulnerability in the software that leads to buffer overflow
Malicious Code	Launching malicious code, such as a virus or a worm, which can cause network to slow down or create damage

Security Assurance

- Vulnerability Avoidance
  - System designed so vulnerabilities can not occur (e.g. no network connection)
- Attack Detection and Elimination
  - System designed so attacks on vulnerabilities do not occur (e.g. use of anti-virus software)
- Exposure Limitation
  - System designed so damage from attacks is minimal (e.g. a backup policy that allows restoration of damaged files)

Thesis topic

- Mirai - virus
- Predict Mirai attacks by exploring system's performance over time in microservice-based systems
  - SUT Sockshop - TicketTrain

Thesis topic

- Mining examples of non-secure code (CWE https://cwe.mitre.org) / Common effects of non-secure code sures (CVE https://www.cvedetails.com) / attacks to non-secure code www.exploit-db.com) repositories to recommend developers with vulnerable code
- bugs related to non-secure code e code
- Dev code comments related to non-secure code

Time to watch!

- Watch the following lecture from MIT open course on Verification and Validation and report back any unknown fact on “system” verification and validation

- https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-842-fundamentals-of-systems-engineering-fall-2015/class-videos/session-9-verification-and-validation/

Verification techniques

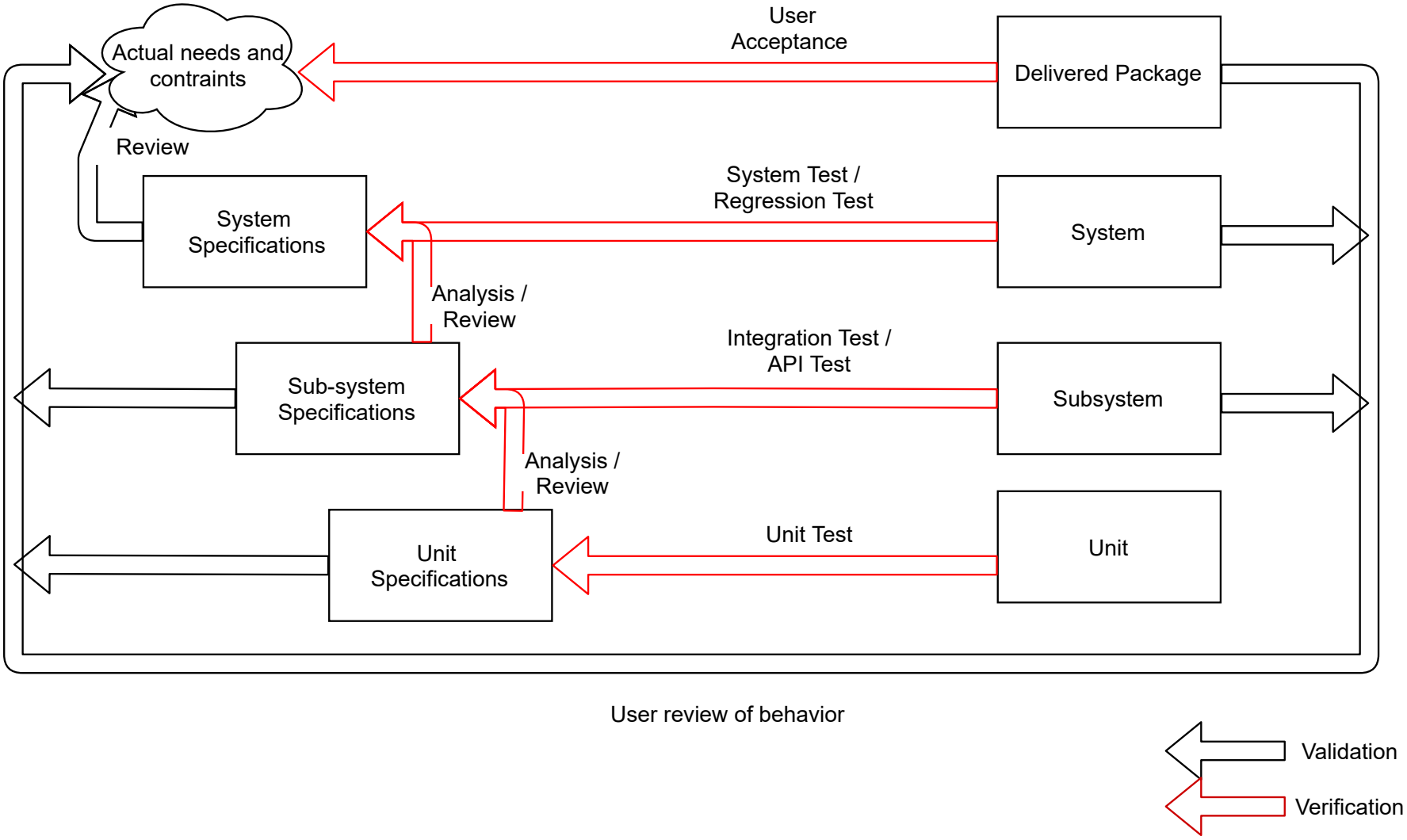
Barbara Russo

SwSE - Software and Systems Engineering Research Group

Verification techniques - types

- Program review
- Program analysis
- Testing

Verifications techniques



Analysis

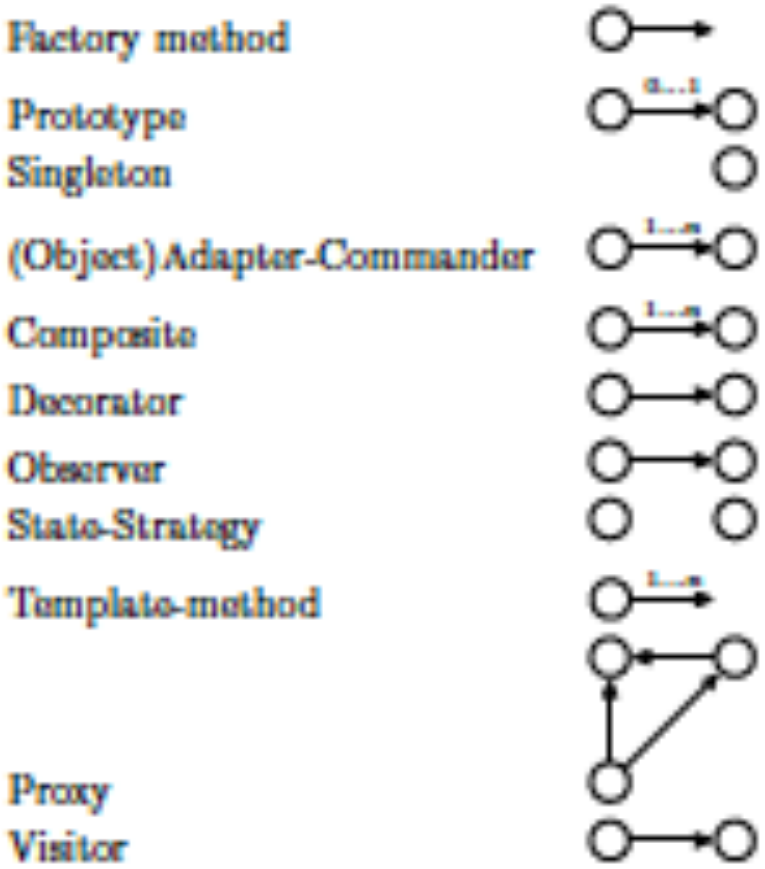
- Analysis of the system/software program and its artefacts
- Analysis techniques that do **not need to execute code** or **run the system** are the most used as they can be used at any stage of development
  - Example: manual code inspection

Example 1

- Estimating correctness
  - Identify software micro-architectures in software architectures that
  - 1. frequently change over versions & are prone to defects:
    - Artefacts: classes and their calls
  - 2. or frequently change over versions & change together (e.g., to fix a bug)
    - Artefact: bug report, code commit

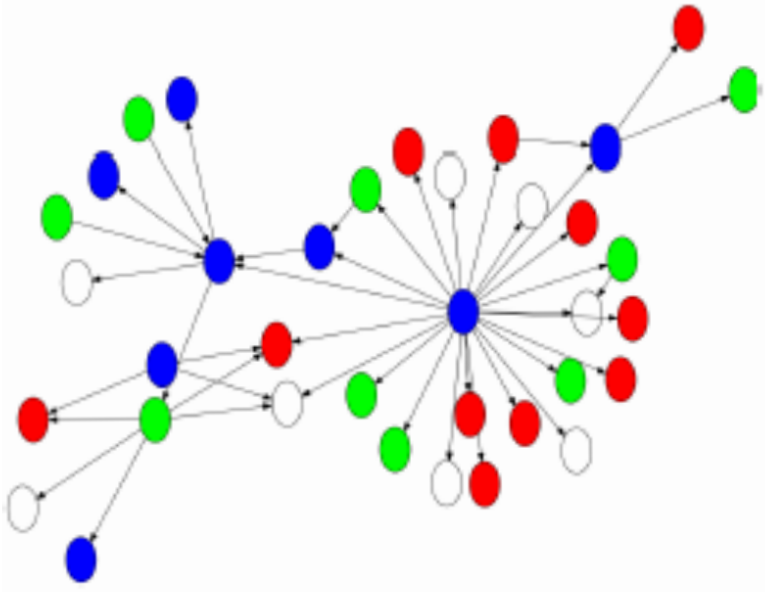
Example

- Are these micro-structures intentional (design patterns) or residual of bad code?



Example 2

- Representing the commit process as a graph, where nodes are commits and edges occur when two commits change the same files
- Mining commits' graphs (subgraph isomorphisms) to investigate the presence of large architectural changes and their likelihood of occurrence in bug fixing.
- Artefact: code commit, changed files



Example 3

- To predict the anomalies of systems
  - **Mining systems logs** to model and predict system mis-behavior (anomalies).
  - Artefacts: log events

```
<INFO
    TimeStamp= 20170605T05:54:05.520Z
    File= ObservingModeBaseImpl.java
    Line= 260
    Routine= beginSubscan
    Host= gas01
    Process= CONTROL/ACC/javaContainer
    SourceObject= CONTROL/Array001
    Thread= Thread771
    LogId= 20375
    Audience= Operator >
<![CDATA['Scan 5, subscan 3 has an intent of HOT, takes
5.760 seconds from 05:54:05.520 to 05:54:11.280']]>
</INFO>
```

Review

- The goal is to examine a software/system artefact and to approve it
  - Systematic inspection of software/system to find and resolve defects
  - Typically, performed manually
  - Documents like requirements, system designs, codes, test plans and test cases

Testing

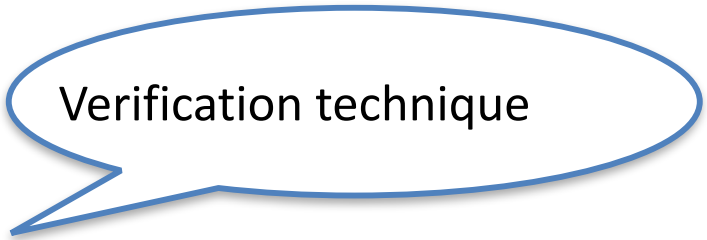
- Analysis and review are typically static
- Testing is a **dynamic process**
  - **It requires to execute the software or run the system**
  - It can be done only when the artefacts to be tested are “executable”

Exercise

```
[1] int foo (int a, int b, int c, int d, float e) {
[2]    if (a == 0) {
[3]        return 0;
[4]    }
[5]    int x = 0;
[6]    if ( (a==b) II ( (c == d) && bug(a) ) ) {
[7]        x=1;
[8]    }
[9]    e = 1/x;
[10]    return e;
[11] }
```

bug(a) = TRUE if !a==0 else 0

Exercise



- Statement coverage
- If 100% of statement are covered by tests then the method is correct

Exercise - Statement coverage

- Identify input values that execute all statements
  - How many t-uples of input values? What is the output for each of them?
  - I/O and pass and fail criterion define a test case

Test case - notation

- **$T_{\text{Output value}}$ (input values)**

- $T_{\text{FileNotFoundException}}$(0, “Hello”, 0.3)
- T_3(0, “Home”, 3)
- $T_{(3,4)}$(1, “Home”, “Layout”)

Exercise - program analysis

- Property: correctness
  - First check manually if the method is correct
  - Then use statement coverage
- Do the selected inputs catch the failure?
- Discuss accuracy of statement coverage with the t-uples of input values you chose: FP? FN?