FACULDADE ENGENHARIA
Departamento de Informática

Qualidade de Software (14450)

Getting Started

(adapted from lecture notes of the “DIT 635 - Software Quality and Testing” unit, delivered by Professor Gregory Gay, at the Chalmers and the University of Gothenburg, 2022)

Today's Goals

- Introduce the idea of "quality"
- Cover the basics of verification and validation
- Software testing myths

When is software ready for release?

Flawed Software Will Be Exploited

40 Million Card Accounts Affected by Security Breach at Target

Sony: Hack so bad, our computers still don't work

By Charles Riley @CRrileyCNN January 23, 2015: 10:10 AM ET

Recommend 182

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

Flawed Software Will Be Exploited

Unpatched Critical Flaw Disclosed in Zoom Software for Windows 7 or Earlier

July 10, 2020 Swati Khandelwal

Tesla to recall 135,000 vehicles over computer memory failure

BY FRENCH PRESS AGENCY - AFP | WASHINGTON DC | FEB 02, 2021 - 8:54 PM GMT+3 |

Flawed Software Will Hurt People

In 2010, software faults were responsible for **26% of medical device recalls**.

“There is a reasonable probability that use of these products will cause serious adverse health consequences or death.”

-US Food and Drug Administration

When is software ready for release?

The short (and not so simple) answers...

- ✧ We release **when we can’t find any bugs…**
- ✧ We release **when we have finished testing…**
- ✧ We release **when quality is high...**

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

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”.

Quality Assurance

General Principles of QA:

✧ Know what you are doing

✧ Know what you should be doing

✧ Know how to measure the difference

When is Software Ready for Release?

Software is ready for release when you can argue that it is ***dependable***.

✧ Correct, reliable, safe, and robust.

✧ Shown through **Verification and Validation**.


Software System
Brilliant quality assurance engineer

Verification and Validation

Barry Boehm, inventor of the term “software engineering”, describes them as:

✧ **Verification:**

- “Are we building the product right?”

✧ **Validation:**

- “Are we building the right product?”

Verification and Validation

Verification

Validation

Verification and Validation

Activities that must be performed to consider the software "done."

- **Verification:** The process of proving that the software conforms to its specified functional and non-functional requirements.

- **Validation:** The process of proving that the software meets the customer's true requirements, needs, and expectations.

Verification

- ✧ Is the implementation consistent with its specification?
  - Does the software work under conditions we set?
  - (usually based on requirements)
- ✧ **Verification is an experiment.**
  - Perform trials, evaluate results, gather evidence.

Verification

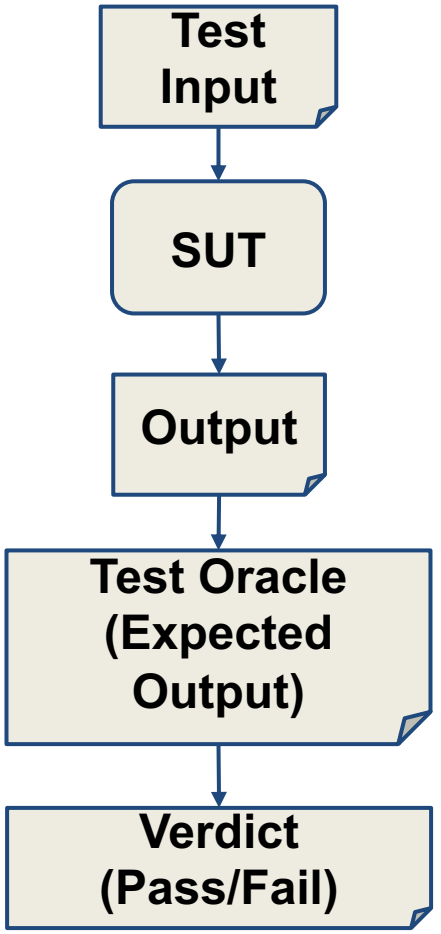
- ✧ Is a implementation consistent with a specification?
- ✧ “Specification” and “implementation” are roles.
  - Usually source code and requirement specification.
  - But also…
    - Detailed design and high-level architecture.
    - Design and requirements.
    - Test cases and requirements.
    - Source code and user manuals.

Software Testing

✧ An investigation into system quality.

✧ Based on sequences of **stimuli** and **observations**.

- **Stimuli** that the system must react to.
- **Observations** of system reactions.
- **Verdicts** on correctness.



Validation

- ✧ Does the product work in the real world?
  - Does the software fulfill the users' **actual needs**?
- ✧ Not the same as conforming to a specification.
  - If we specify **two buttons** and implement all behaviors related to those buttons, we can achieve verification.
  - If the user expected **a third button**, we have not achieved validation.

Verification and Validation

- ✧ Verification
  - Does the software work as intended?
  - Shows that software is dependable.
- ✧ Validation
  - Does the software meet the needs of your users?
  - Shows that software is useful.
  - **This is much harder.**

Verification and Validation

- ✧ Both are important.
  - A well-verified system might not meet the user's needs.
  - A system can't meet the user's needs unless it is well-constructed.
- ✧ This class largely focuses on verification.
  - **Testing is the primary activity of verification**.

Required Level of V&V

✧ Depends on:

- **Software Purpose:** The more critical, the more important that it is reliable.
- **User Expectations:** Users may tolerate bugs because benefits outweigh cost of failure recovery.
- **Marketing Environment:** Competing products - features and cost - and speed to market.

Basic Questions

1. When do verification and validation start and end?
2. How do we obtain acceptable quality at an acceptable cost?
3. How can we assess readiness for release?
4. How can we control quality of successive releases?
5. How can the development process be improved to make verification more effective?

When Does V&V Start?

✧ V&V can start **as soon as the project starts**.

- Feasibility studies must consider quality assessment.
- Requirements can be used to derive test cases.
- Design can be verified against requirements.
- Code can be verified against design and requirements.
- Feedback can be sought from stakeholders at any time.

Static Verification

✧ Analysis of system artifacts to discover problems.

- **Proofs:** Posing hypotheses and making arguments for their validity using specifications, system models, etc.
- **Inspections:** Manual “sanity check” on artifacts (e.g., source code) by people or tools, searching for issues.

Advantages of Static Verification

- One error can hide other errors. Inspections not impacted by program interactions.
- Incomplete systems can be inspected without special code to run partial system.
- Inspection can assess quality attributes such as maintainability, portability, code style, program inefficiencies, etc.

Dynamic Verification

- Exercising and observing the system to argue that it meets the requirements.
  - **Testing:** Formulating sets of input to demonstrate requirement satisfaction or find faults.
  - **Fuzzing:** Generating semi-random input to locate crashes and other anomalies.
  - **Taint Analysis:** Monitoring how faults spread by corrupting system variables.

Advantages of Dynamic Verification

- ✧ Discovers problems from runtime interaction, timing problems, or performance issues.
- ✧ Often cheaper than static verification.
  - Easier to automate.
  - However, cannot prove that properties are met - cannot try all possible executions.

The Trade-Off Game

Software engineering is the process of designing, constructing and maintaining the best software possible given the available resources.

Always trading off between what we want, what we need, and what we've got.

As a NASA engineer put it,

✧ “Better, faster, or cheaper - pick any two”

Perfect Verification

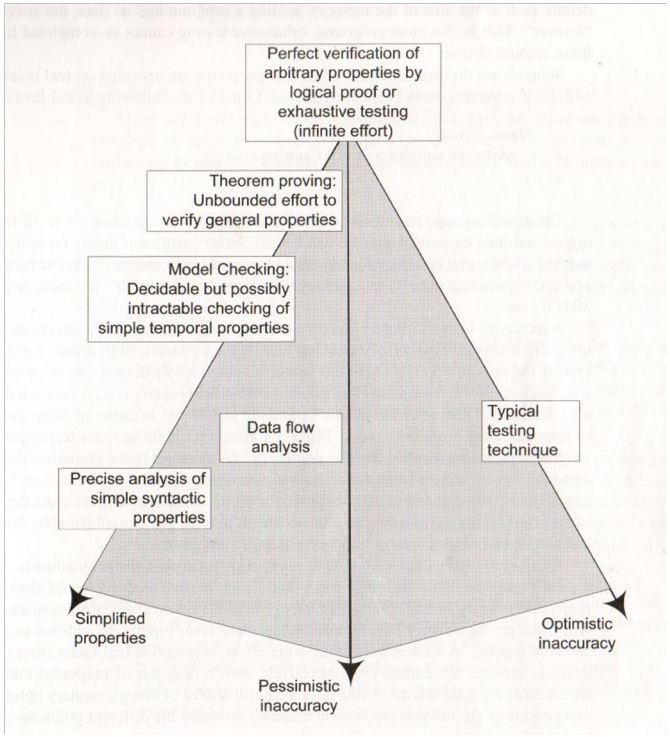
✧ Verification is an instance of the **halting problem**.

- There is at least one program for which any technique cannot obtain an answer in finite time.
- Testing - cannot exhaustively try all inputs.
- Must accept some degree of inaccuracy.

Verification Trade-Offs

We are interested in proving that a program demonstrates property X

- **Pessimistic Inaccuracy** - not guaranteed to program even if the it possesses X.
- **Optimistic Inaccuracy** - may accept program that does not possess X.
- **Property Complexity** - if X is too difficult to check, substitute simpler property Y.



How Can We Assess Readiness?

- ✧ Finding all faults is nearly impossible.
- ✧ Instead, decide when to stop V&V.
- ✧ Need to establish criteria for acceptance.
  - How good is “good enough”?
- ✧ Measure dependability and other quality attributes and set threshold to meet.

Product Readiness

- ✧ Put it in the hands of human users.
- ✧ **Alpha/Beta Testing**
  - Small group of users using the product, reporting feedback and failures.
  - Use this to judge product readiness.
  - Make use of dependability metrics for quantitative judgement (metric > threshold).
  - Make use of surveys as a qualitative judgement.

Ensuring Quality of Successive Releases

- ✧ V&V do not end with the release of the software.
  - Software evolves - new features, environmental adaptations, bug fixes.
  - Need to test code, retest old code, track changes.
  - When code changes, rerun tests to ensure tested elements still works.
  - Retain tests that exposed faults to ensure they do not return.

Improving the Development Process

- ✧ Try to learn from your mistakes in the next project.
  - Collect data during development.
    - Fault information, bug reports, project metrics (complexity, # classes, # lines of code, test coverage, etc.).
  - Classify faults into categories.
  - Look for common mistakes.
  - Learn how to avoid such mistakes.
  - Share information within your organization.

Software Testing Myths

Myth	Fact
Quality Control = Testing.	Testing is just one component of software quality control, which includes other activities such as Reviews.
The objective of Testing is to ensure a 100% defect- free product.	The objective of testing is to uncover as many defects as possible while ensuring that the software meets the requirements. Identifying and getting rid of all defects is impossible.
Testing is easy.	Testing can be difficult and challenging (sometimes, even more so than coding).
Anyone can test.	Testing is a rigorous discipline and requires many kinds of skills.
Automated testing eliminates the need for manual testing.	100% test automation cannot be achieved. Manual Testing, to some level, is always necessary.

Testing Requires Writing Code

✧ Testing cannot wait for the system to be complete.

- The component to be tested must be isolated from the rest of the system, instantiated, and *driven* using method invocations.
- Untested dependencies must be *stubbed out* with reliable substitutions.
- The deployment environment must be simulated by a controllable *harness*.

Test scaffolding

- **Test scaffolding** is a set of programs written to support test automation.
  - Not part of the product
  - Often temporary
- Allows for:
  - Testing before all components complete.
  - Testing independent components.
  - Control over testing environment.

Test scaffolding

- A **driver** is a substitute for a main or calling program.
  - Test cases are drivers.
- A **harness** is a substitute for all or part of the deployment environment.
- A **stub** (or **mock object**) is a substitute for system functionality that has not been completed.
- Support for recording and managing test execution.

Scaffolding

- Stubs and drivers are code written as replacements other parts of the system.
  - May be required if pieces of the system do not exist.
- Scaffolding allows greater control over test execution and greater observability to judge test results.
  - Ability to simulate dependencies and test components in isolation.
  - Ability to set up specialized testing scenarios.
  - Ability to replace part of the program with a version more suited to testing.

Generic vs Specific Scaffolding

- ✧ Simplest driver - one that runs a single specific test case.
- ✧ More complex:
  - Common scaffolding for a set of similar tests cases,
  - Scaffolding that can run multiple test suites for the same software (i.e., load a spreadsheet of inputs and run then).
  - Scaffolding that can vary a number of parameters (product family, OS, language).
- ✧ Balance of quality, scope, and cost.

Key Points (1 of 2)

- Quality attributes describe desired properties of the system under development.
  - Dependability, scalability, performance, availability, security, maintainability, testability, ...
- Developers must prioritize quality attributes and design a system that meets chosen thresholds.
- Quality is often subjective. Choose a definition, and offer objective thresholds.

Key Points (2 of 2)

- Software should be dependable and useful before it is released into the world.
- Verification is the process of demonstrating that an implementation meets its specification.
  - This is the primary means of making software dependable (and demonstrating dependability).
  - Testing is most common form of verification.


HOW
WHY
WHERE
WHEN
WHERE
WHY
WHERE
WHO
WHEN
WHAT
WHAT
WHY
WHAT
WHEN
WHO
WHERE
WHY
HOW