CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Lecture 1: Software Quality, Verification, and Validation

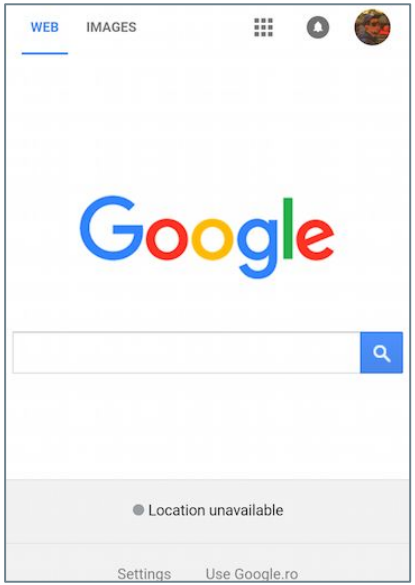
Gregory Gay
DIT636/DAT560 - January 15, 2024

menti.com, code **7590 0097**

When is software ready for release?

Our Society Depends on Software

This is software:



So is this:

Flawed Software Will Hurt Profits

- 2002: “Bugs cost the U.S. economy $60 billion annually… and testing would relieve one-third of the cost.” **(NIST)**
  - “In 2016, that number jumped to *$1.1 trillion*” **(Cohane)**

“Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it before.” **- Barry Boehm (Emeritus Prof., USC)**

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

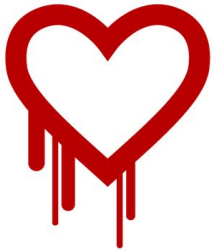
By Charles Riley @CRrileyCNN January 23, 2015: 10:10 AM ET

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 Hurt People

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

- 2011-2015: 627 devices (1.4 million units) recalled, 12 devices (191K units) in “highest-risk” category.

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

- **US Food and Drug Administration**

This Course

- What is “good” software?
  - Determined through **quality metrics** (dependability, performance, scalability, availability, security, ...)
- The key to good software?
  - **Verification and Validation**
- We will explore **testing** and **analysis** activities of the V&V process.

Today's Goals

Introduce The Class

- AKA: What the heck is going on?
- Go over course PM
- Clarify expectations
- Assignments/grading
- Answer any questions
- Introduce the idea of "quality"
- Cover the basics of verification and validation

Contact Details

- Instructor: Greg Gay (Dr, Professor, $#*%)
  - E-mail: ggay@chalmers.se
- Website:
  - https://chalmers.instructure.com/courses/27815
    - Pay attention to the schedule/announcements
  - https://greg4cr.github.io/courses/spring24dit636
    - Backup of Canvas page/course materials.
    - May be out of date, but good if Canvas isn't working.

Teaching Team

- Teaching Assistants
  - Afonso Fontes (afonso.fontes@chalmers.se)
  - Burak Askan (gusaskbu@student.gu.se)
  - Labiba Karar Eshaba (guseshka@student.gu.se)
  - Kanokwan Haesatith (kanokwan.haesatith@outlook.com)
  - Georg Zsolnai (guszsoge@student.gu.se)
- Student Representatives
  - Adrian Hassa (gushasade@student.gu.se)
  - Abhimanyu Kumar (guskumab@student.gu.se)
  - Marko Mojsov (gusmojsma@student.gu.se)
  - Marcelo Santibáñez (gussanmaed@student.gu.se)
  - Michal Spano (gusspanomi@student.gu.se)

Seeking one Chalmers student - email ggay@chalmers.se

Communication and Feedback

- Post questions to Canvas discussion forum (preferred) or e-mail to myself/TAs.
- Send me private or sensitive questions!
- Send feedback to course reps or me.
- Contact studentoffice@cse.gu.se for questions related to registration, sign-up, LADOK.

Desired Course Outcomes

Knowledge and understanding

- Explain quality assurance models in software engineering and the contents of quality assurance plans
- Describe the distinction between verification and validation
- Name and describe the basic concepts on testing, as well as different testing techniques and approaches
- Describe connection between development phases and kinds of testing
- Exemplify and describe a number of different test methods, and be able to use them in practical situations
- Exemplify and describe tools used for testing software, and be able to use them and interpret their output

Desired Course Outcomes

Competence and skills

- Define metrics required for monitoring the quality of projects, products and processes in software engineering
- Construct appropriate and meaningful test cases, and interpret and explain (to stakeholders) the results of the application of such test cases (using appropriate tools) to practical examples
- Develop effective tests for systems at differing levels of granularity (e.g., unit and system level)
- Plan and produce appropriate documentation for testing
- Apply different testing techniques on realistic examples

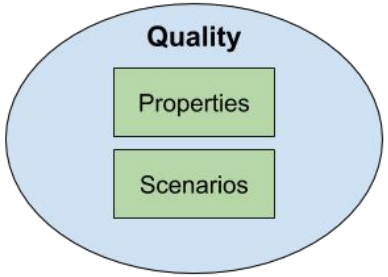
Desired Course Outcomes

Judgement and approach

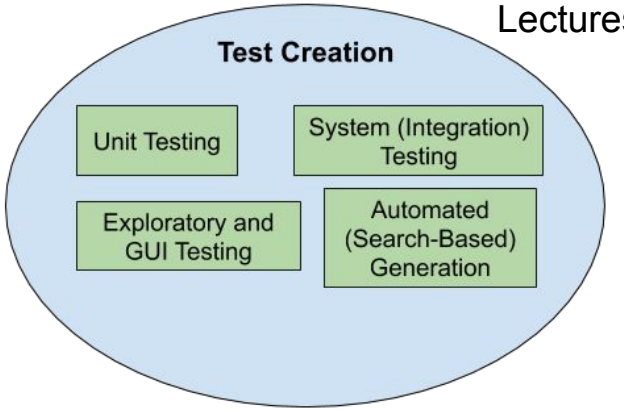
- Identify emerging techniques and methods for quality management using relevant sources
- Identify and hypothesize about sources of program failures, and reflect on how to better verify the correctness of such programs

Lecture Plan (approximate)

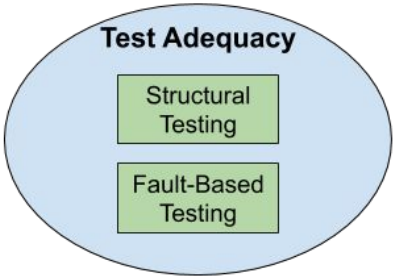
Lectures 2-3



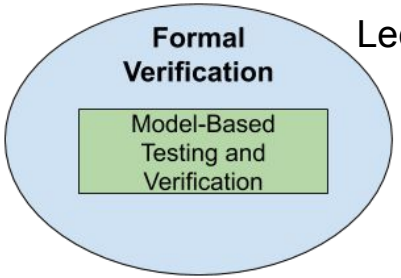
Lectures 4 - 8, 12



Lectures 9 - 11



Lectures 13 - 14

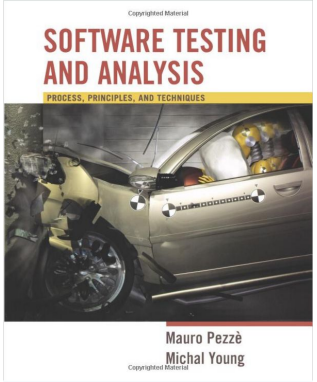


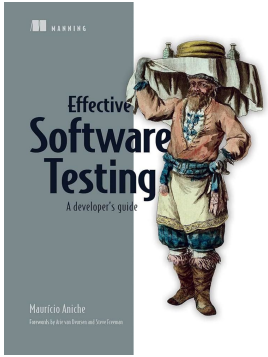
Changes from Last Time

- Slides have been updated and some examples have been added or reworked.
- Assignments have been updated and reworked.

(Optional) Course Literature

- Software Testing and Analysis, Mauro Pezze and Michal Young.
  - Free: https://ix.cs.uoregon.edu/~michal/book/free.php
- Effective Software Testing: A Developer's Guide, Maurício Aniche.
  - $25.99 (eBook), $32.49 (physical).
  - https://www.effective-software-testing.com/





Prerequisite Knowledge

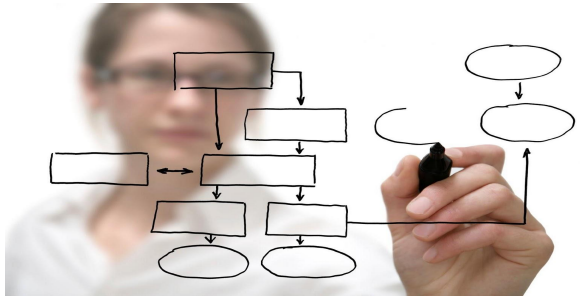
- You need to be proficient in Java.
  - Some code examples also in Python, C/C++.
- Basic understanding of build systems and continuous integration.
  - We will go over specifics later.
- Basic understanding of logic and sets.
  - Formal verification based on logical arguments.

Course Design

Lectures

Exercise Sessions

Group Assignments



Examination Form

Sub-Courses

- Written examination (Skriftlig tentamen), 4.5 higher education credits
- Assignments (Inlämningsuppgifter), 3 higher education credits
  - Grading scale: Fail (U), 3-5

Assessment

- Individual hall exam at end of course
- Written assignments in teams of **three**.
  - **You may choose your own team. See Assignment 0 on Canvas. Due next Monday.**
- Three written assignments.
  - Equally weighted.
  - Final grade is average of three assignment grades.

Assessment

- Self and peer-evaluation due with each assignment
  - May be used to adjust individual assignment grades.
  - **AKA: don’t slack off!**
- Late assignments, -20% per day, 0% after two days
- If final assignment average is failing, all three assignments must be redone/resubmitted.

Grading Scale

- Score of 1-100, converted to Fail, 3-5:
- Final course grade:

% Grade	Grading Scale
0-49%	Fail (U)
50-69%	3
70-85%	4
86-100%	5

(right) Exam Grade (down) Assignment Grade	U	3	4	5
U	U	U	U	U
3	U	3	4	4
4	U	3	4	5
5	U	4	4	5

Expected Workload

- This class can be time consuming.
  - Understanding the material takes time.
  - Project work requires team coordination.
- Do not underestimate the project work.
  - Good engineering is hard.
  - Planning and scheduling your time is essential.
  - Do NOT delay getting started.
  - Appoint a team leader (and rotate the role)

Other Policies

Integrity and Ethics:

- Work you submit must be your own.
  - Generative AI: Can be used to brainstorm, summarize concepts, fix grammar, generate plots (i.e., to support learning), but all text and code must be written by **YOU**.
  - If you want to use GenAI, **ask me first**, disclose in submission.
- Collaboration is not permitted on assignments.
- Violation = failing grade and reporting.

Other Policies

Classroom Climate:

Arrive on time, don't talk during lecture, don't use chat unless asking or answering questions. Disruptive students will be warned and dismissed.

Diversity

Students in this class are expected to work with all other students, regardless of gender, race, sexuality, religion, etc. Zero-tolerance policy for discrimination.

Special Needs

We will provide reasonable accommodations to students that have disabilities. Contact teaching team early to discuss individual needs.

Let’s take a break!

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 **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 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”.

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**.

Verification and Validation

Activities that must be performed to consider the software “done.”

- **Verification**: Proving that software conforms to its functional and non-functional requirements.
- **Validation**: Proving that software meets customer’s true requirements, needs, and expectations.

Verification and Validation

Barry Boehm:

- **Verification**:
  - “Are we building the product right?”
- **Validation**:
  - “Are we building the right product?”

Verification

- Is the implementation correct?
  - Judged by asking: “Is the implementation consistent with its specification?”
- **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.

How do we know a system is correct?

Rationalists

"It is correct because I **proved** that certain errors do not exist in the system."

Empiricists

"It is correct because I never **observed** incorrect behaviors."

Adapted from Shin Yoo (KAIST)

Static Verification

- Analysis of code and other development artifacts.
  - **Proofs**: Posing hypotheses and making arguments using specifications, models, etc.
  - **Inspections**: Manual “sanity check” on artifacts (e.g., source code), searching for issues.

“It is correct because I **proved** that certain errors do not exist in the system.”

Advantages of Static Verification

- Proofs offer conclusive evidence of problems.
- One error can hide other errors. Inspections not impacted by program interactions.
- Incomplete systems can be inspected.
- Code inspections can assess subjective quality attributes (maintainability, portability, usability).

Dynamic Verification

- Exercising and observing the system.
  - **Testing:** Executing input and checking whether the resulting output meets expectations.
  - **Fuzzing:** Generating semi-random input to detect crashes, memory leaks, buffer overflows, etc.
  - **Taint Analysis:** Monitoring how corrupted data spreads through system.

"It is correct because I never **observed** incorrect behaviors."

Advantages of Dynamic Verification

- Discovers problems from runtime interaction, timing problems, or performance issues.
- Cheaper, more scalable than static verification.
  - Much easier to achieve volume.
  - Works on much more complex systems.
  - However, cannot prove that properties are met
    - Cannot try all possible executions.

The Trade-Off

Static Analysis:

- Overapproximation
- Naive analysis:
  - There is a division-by-zero error here.
- Not being naive is expensive.

```
def foo(n):
    if n > 0:
        print(bar(n))
    else:
        return

def bar(a):
    return 42 / a
```

Dynamic Analysis:

- Underapproximation
- Only detect faults if we select the right input.

```
def test_bar():
    assert bar(42) == 1
```

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 failed 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**.

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.

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.

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.

Ensuring Quality of Successive Releases

- V&V do not end with release.
  - New features, environmental adaptations, bug fixes.
  - Test new code, retest old code, track changes.
    - When code changes, rerun tests to ensure old code 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.

We Have Learned

- 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.

We Have Learned

- 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.

Next Time

- Measuring and assessing quality.
- No exercise session this week.

- Plan your team selection.
  - The earlier, the better! Due January 21, 23:59.
  - See Assignment 0 on Canvas

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY