CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

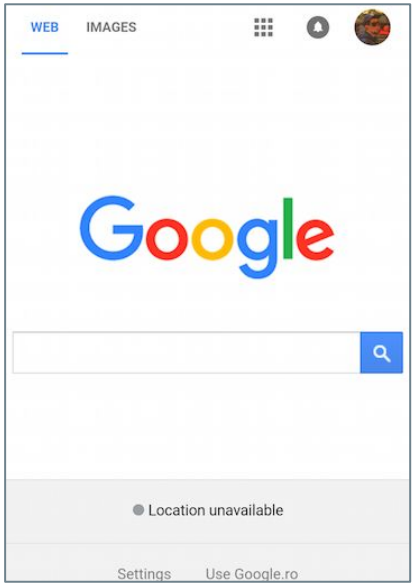
Lecture 1: Software Quality, Verification, and Validation

Gregory Gay
DIT635 - January 19, 2022

When is software ready for release?

Our Society Depends on Software

This is software:



So is this:

Flawed Software Will Hurt Profits

"Bugs cost the U.S. economy $60 billion annually… and testing would relieve one-third of the cost."

- NIST

"Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it before."

- Barry Boehm (TRW Emeritus Professor, 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

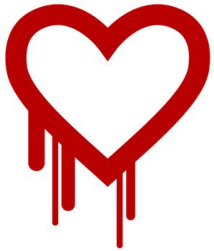
By Charles Riley @CRrileyCNN January 23, 2015: 10:10 AM ET

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**.

“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://canvas.gu.se/courses/51646
    - Pay attention to the schedule/announcements
  - https://greg4cr.github.io/courses/spring22dit635
    - 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)
  - Sandra Smoler Eisenberg (gussmosa@student.gu.se)
- Student Representatives
  - You?
  - E-mail ggay@chalmers.se if you want to volunteer.

Communication and Feedback

- Post questions to Canvas discussion forum (preferred) or e-mail.
- Send me private or sensitive questions!
- Send feedback to course reps or me.
- Contact student_office.cse@chalmers.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

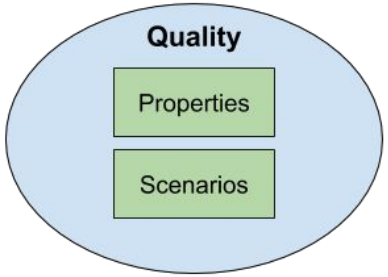
Desired Course Outcomes

Judgement and approach

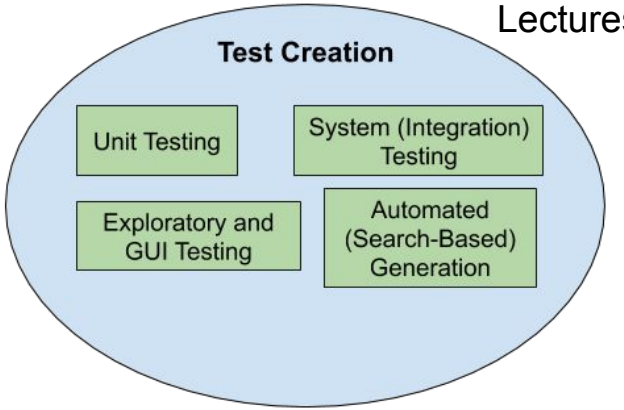
- Identify emerging techniques and methods for quality management using relevant sources
- Identify and hypothesize about sources of program failures, and reflect on how to better verify the correctness of such programs

Lecture Plan (approximate)

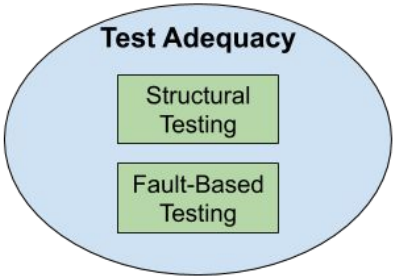
Lectures 2-3



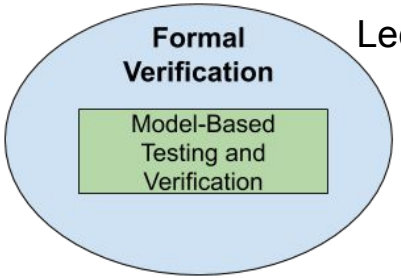
Lectures 4 - 8, 12



Lectures 9 - 11



Lectures 13 - 14

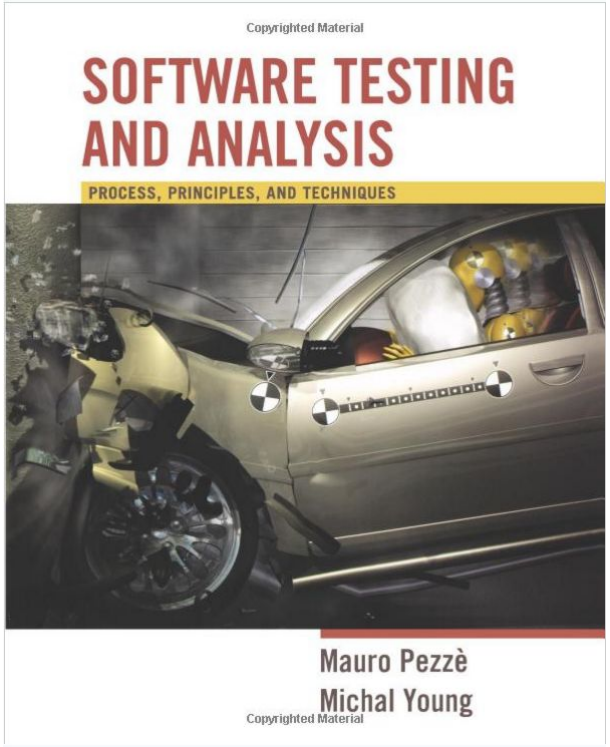


Changes from Last Time

- Small changes to lecture contents to fix typos, update materials.
- Adjustments to homework assignments based on feedback.

Course Literature

- Software Testing and Analysis, Mauro Pezze and Michal Young.
  - Free from https://ix.cs.uoregon.edu/~michal/book/free.php
  - Gives more information on many of the topics covered
- Others posted to Canvas



Prerequisite Knowledge

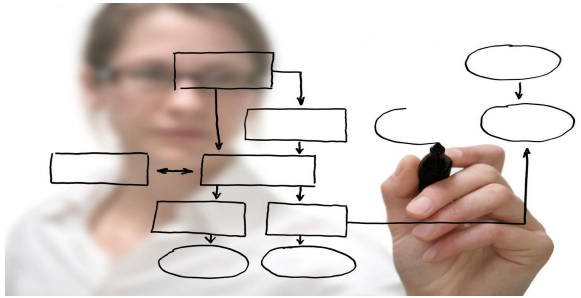
- You need to be proficient in Java
  - (ideally, some knowledge of C/C++)
- Basic understanding of build systems and continuous integration
  - We will go over specifics later.
- Basic understanding of logic and sets.
  - Formal verification based on logical arguments.

Course Design

Lectures (Wed 8:15-10:00, Friday 10:15-12:00)

Exercise Sessions
(Friday, 13:15-15:00)

Group Assignments



Online Lectures

- **For now:**
  All lectures and exercise sessions on Zoom.
  - **Will switch to hybrid if covid situation improves.**
- Lectures will recorded, exercise sessions not.
- In lectures: use real names, mute/no camera, ask questions via chat.
  - (exercise sessions are more interactive - feel free to use camera/mic there if you want)

Examination Form

Sub-Courses

- Written examination (Skriftlig tentamen), 4.5 higher education credits
  - Grading scale: Pass with Distinction (VG), Pass (G) and Fail (U)
- Assignments (Inlämningsuppgifter), 3 higher education credits
  - Grading scale: Pass (G) and Fail (U)

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 Tuesday.**
- 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

- Assignments: Pass (G), Fail (U)
- Exam: Pass w/ Distinction (VG), Pass (G), Fail (U)

Grading Scale for Assignments:

% Grade	Grading Scale
0-59%	Fail (U)
60-100%	Pass (G)

Grading Scale for Exams:

% Grade	Grading Scale
0-49%	Fail (U)
50-85%	Pass (G)
86-100%	Pass with Distinction (VG)

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:

Homework and programs you submit for this class must be entirely your own. If this is not absolutely clear, then contact me. Any other collaboration of any type on any assignment is not permitted. It is your responsibility to protect your work from unauthorized access. Violation = failing grade and reporting.

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.

Other Policies

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 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”.

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:** 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 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

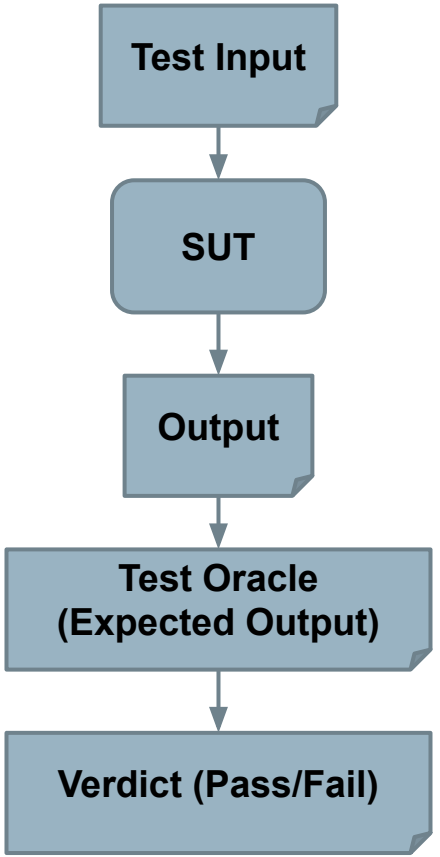
- Is the implementation consistent with its specification?
  - Does the software work under conditions we set?
  - (usually based on requirements)
- **Verification is an experiment.**
  - Perform trials, evaluate results, gather evidence.

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 using specifications, models, etc.
  - **Inspections:** Manual “sanity check” on artifacts (e.g., source code), 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.

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

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

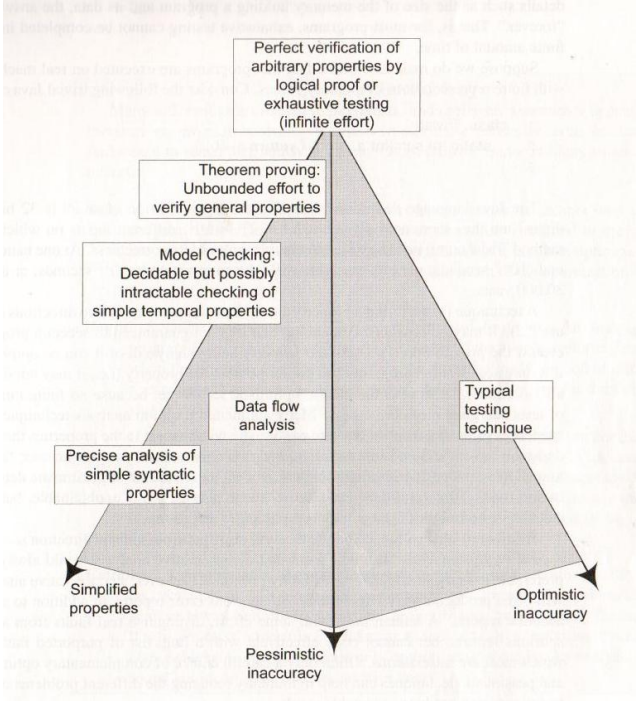
Perfect Verification

- 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.

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.
- Quality is often subjective. Choose a definition, and offer objective 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.
  - Pezze & Young - Chapter 4
  - Other reading on Canvas

- Plan your team selection.
  - The earlier, the better! Due January 25, 11:59 PM.
  - See Assignment 0 on Canvas

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY