CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Lecture 1: Software Quality, Verification, and Validation

Gregory Gay
DIT635 - January 22, 2020

Today's Goals

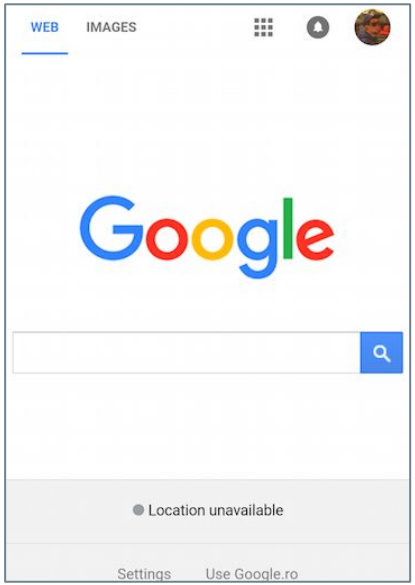
Introduce The Class

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

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

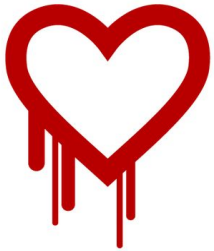
By Charles Riley @CRrileyCNN January 23, 2015: 10:10 AM ET

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?
  - **Quality Attributes and Measurement**
    - How do we assess quality?
- The key to good software?
  - **Verification and Validation**
    - Does the software do what we promised?
    - Does the software meet the needs of its users?
    - In this course, we will explore the **testing** and **analysis** activities that make up the V&V process.

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

- Describe the area of formal verification in general, including model checking and runtime verification, and its relationship to software quality
- Define metrics or monitoring quality of projects, products and processes
- 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
- Write models in at least one formal specification language 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)

- Introduction and Fundamentals (Lecture 1)
- Quality (Lectures 2-4: Dependability, Performance, Scalability, Security)
- Testing (Lectures 5-14)
  - Fundamentals
  - Unit Testing
  - Functional Testing
  - Structural Testing
  - Integration Testing
  - UI Testing
  - Acceptance Testing
  - Regression Testing
  - Model-Based Testing

Contact Info

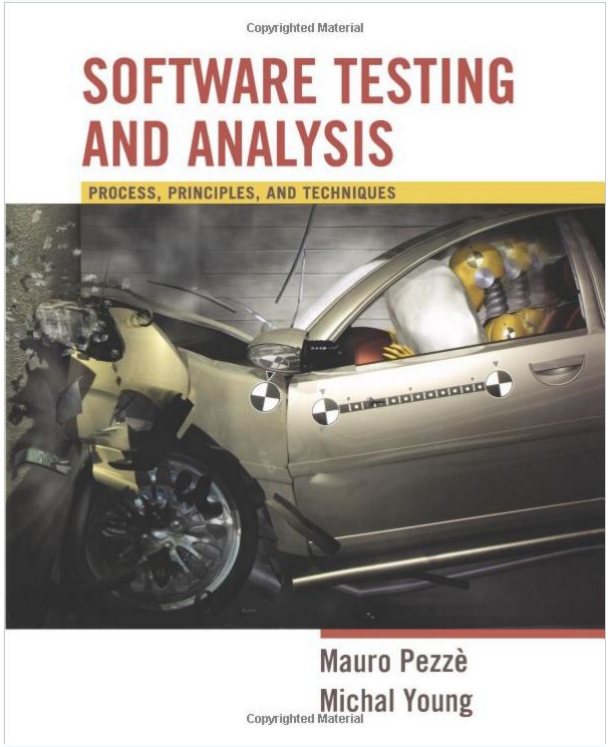
- Instructor: Greg Gay (Dr, Professor, $#*%)
  - E-mail: ggay@chalmers.se
  - Office: 481 Jupiter
    - No formal office hours, but feel free to stop by (e-mail first)
- Website:
  - https://canvas.gu.se/courses/25762
    - Pay attention to the schedule/announcements
  - https://greg4cr.github.io/courses/spring20dit635
    - Backup of Canvas page/course materials.
    - Likely out of date, but back-up if Canvas isn't working.

Teaching Team

- Teaching Assistants
  - Mohamad Drgham (gusdrgmo@student.gu.se)
  - George Sarkisian (sakogeorge@gmail.com)
- Student Representatives
  - You? E-mail ggay@chalmers.se if you want to volunteer.

Course Literature

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



Prerequisite Knowledge

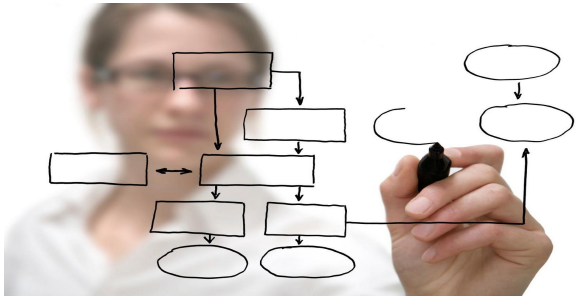
- You need to be proficient in Java
  - (and, ideally, have some knowledge of C/C++)
- You should have basic understanding of build systems and continuous integration
  - We will go over specifics later, so don’t worry.
- You need a basic understanding of logic and sets.
  - Formal verification based 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



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.**
  - **E-mail names + email addresses to ggay@chalmers.se**
  - **E-mail me if you want to be assigned to a team.**
- 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
0-59%	Fail (U)
60-85%	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)

Feedback

Problems with assignments, course questions, feedback?

- Contact me! I like feedback!
- Feel free to talk to course reps + TAs too.
- I'm new here - I **will** make mistakes.

Contact student_office.cse@chalmers.se for questions related to the course administration

- registration, signup, grades in LADOK.

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.

Classroom Climate:

All students are expected to behave as scholars at a leading institute of technology. Arrive on time, don’t talk during lecture, don’t leave before the end of lecture. Disruptive students will be warned and dismissed.

Other Policies

Diversity

Students in this class are expected to respectfully work with all other students, regardless of gender, race, sexuality, religion, etc. There is a zero-tolerance policy for any student that discriminates against other students.

Special Needs

We will provide reasonable accommodations to students that have disabilities that may affect their ability to participate in course activities or to meet course requirements. Students with disabilities should contact their instructor early in the semester to discuss their 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 easy it is to *modify*.
- Quality is hard to measure and assess objectively.

Quality Attributes

- Quality attributes describe desired properties of the system under development.
- Developers must prioritize quality attributes and design a system that meets chosen thresholds.
- Most relevant for this course: **dependability**
  - The ability of the system to *consistently* offer correct functionality, even under *unforeseen* or *unsafe* conditions.

Quality Attributes

- Performance
  - The ability of a system to meet timing requirements. When events occur, the system must respond quickly.
- Security
  - The ability of a system to protect information from unauthorized access while providing service to authorized users.
- Scalability
  - The ability to “grow” the system to process an increasing number of concurrent requests.

Quality Attributes

- Availability
  - The ability to carry out a task when needed, to minimize “downtime”, and to recover from failures.
- Modifiability
  - The ability to enhance software by fixing issues, adding features, and adapting to new environments.
- Testability
  - The ability to easily identify faults in a system. The probability that a fault will result in a visible failure.

Quality Attributes

- Interoperability
  - The ability of the software to exchange information with and provide functionality to other systems.
- Usability
  - The ability of the software to enable users to perform desired tasks and provide support to users.
  - How easy is it to use the system, learn its features, adapt the system to meet user needs, and increase confidence and satisfaction in system use?

Other Quality Attributes

- Resilience
- Supportability
- Portability
- Development Efficiency
- Time to Deliver
- Tool Support
- Geographic Distribution

Quality Attributes

- These qualities often conflict. It is hard to achieve multiple qualities at once.
  - Using fewer subsystems improves performance, but hurts modifiability.
  - Introducing redundant data improves availability, but makes security more difficult.
  - Localizing safety-related features usually introduces more communication between subsystems, degrading 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.
- The primary process of making software dependable (and providing evidence of dependability) is **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?
  - “Specification” and “implementation” are roles.
    - Source code and requirement specification.
    - Detailed design and high-level architecture.
    - Test oracle and requirement specification.
- Verification is an experiment.
  - Does the software work under conditions we set?
  - We can perform trials, evaluate the software, and provide evidence for verification.

Validation

- Does the product work in the real world?
  - Does the software fulfill the users' actual requirements?
- Not the same as conforming to a specification.
  - If we specify and implement all behaviors related to two 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?
- Validation
  - Does the software meet the needs of your users?
  - **This is much harder.**

Validation shows that software is useful. Verification shows that it is dependable. Both are needed to be ready for release.

Verification and Validation: Motivation

- 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.
  - How can we ensure that the software we build is dependable.
  - Testing is the primary activity of verification, and our main focus in this class.

Required Level of V&V

The goal of V&V is to establish confidence that the system is “fit for purpose.”

How confident do you need to be? Depends on:

- **Software Purpose:** The more critical the software, the more important that it is reliable.
- **User Expectations:** When a new system is installed, how willing are users to tolerate bugs because benefits outweigh cost of failure recovery.
- **Marketing Environment:** Must take into account competing products - features and cost - and speed to market.

Basic Questions

1. When do verification and validation start? When are they complete?
2. What techniques should be applied to obtain acceptable quality at an acceptable cost?
3. How can we assess readiness for release?
4. How can we control the quality of successive releases?
5. How can the development process be improved to make verification more effective (in cost and impact)?

When Does V&V Start?

- V&V starts as soon as the project starts.
- Feasibility studies must consider quality assessment.
- Requirement specifications 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.

Types of Verification

Static Verification

- Analysis of system artifacts to discover problems.
  - Proofs: Posing hypotheses and making a logical argument for their validity using specifications, system models, etc.
  - Inspections: Manual “sanity check” on artifacts (such as source code) by other people or tools, searching for issues.

Advantages of Static Verification

- During execution, errors can hide other errors. Hard to find all problems or trace back to a single source.
  - Inspections not impacted by program interactions.
- Incomplete systems can be inspected without additional costs. If a program is incomplete, special code is needed to run the part that is to be tested.
- Inspection can also assess quality attributes such as maintainability, portability, poor programming, inefficiencies, etc.

Dynamic Verification

- Exercising and observing the system to argue that it meets the requirements.
  - Testing: Formulating controlled sets of input to demonstrate requirement satisfaction or find faults.
  - Fuzzing: Spamming the system with random input to locate security vulnerabilities, memory leaks, buffer overruns, etc.
  - Taint Analysis: Assigning a bad value to a variable and monitoring which system variables it corrupts and how it corrupts them.

Dynamic Verification

- Static verification is not good at discovering problems that arise from runtime interaction, timing problems, or performance issues.
- Dynamic verification is often cheaper than static - easier to automate.
  - However, it 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.

We are 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”

The Role of Software Engineers

Software engineers aren't just responsible for designing, constructing, and maintaining software.

They are the people we look to ***plan***, ***make***, and ***justify well-informed decisions*** about ***trade-offs*** throughout the development process.

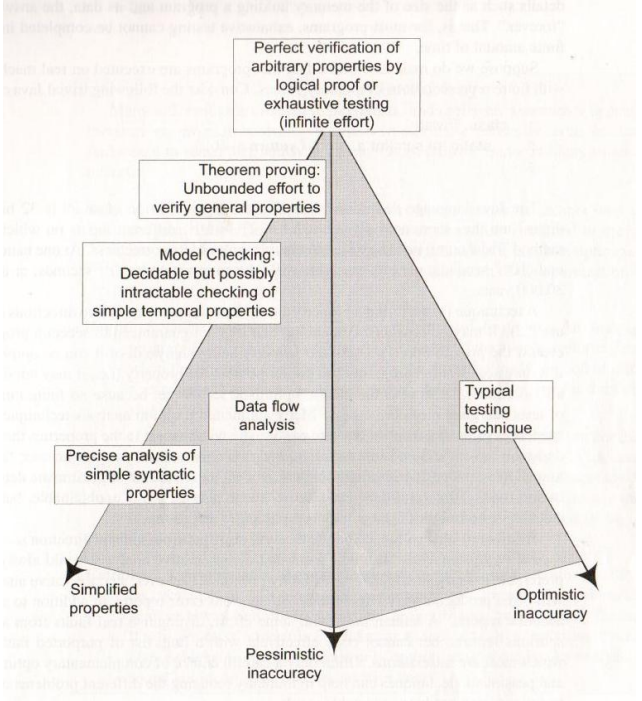
Perfect Verification

- For physical domains, verification consists of calculating proofs of correctness.
- Given a precise specification and a program, we should be able to do the same… Right?
  - 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.
  - We must accept some degree of inaccuracy.

Verification Trade-Offs

Three dimensions of inaccuracy:

- **Pessimistic Inaccuracy** - not guaranteed to accept a program even if the program possesses the property.
- **Optimistic Inaccuracy** - may accept a program that does not possess a property.
- **Property Complexity** - if one property is too difficult to check, substitute one that is easier to check or constrain the types of programs checked.



Assessing Verification Techniques

- Safe
  - No optimistic inaccuracy - it only accepts programs that are correct with respect to that property.
- Sound
  - An analysis of a program with respect to property is *sound* if the technique returns true ONLY when the program does meet the property.
  - If true = correct and the technique is *sound,* then the technique is also *safe*.
  - If true = incorrect and the technique is sound, you allow *optimistic* but disallow *pessimistic inaccuracy*.

Assessing Verification Techniques

- Complete
  - An analysis of a property on a program is *complete* if it always returns true when the program does satisfy the program.
  - If true = correct, then *complete* admits only *optimistic inaccuracy*.
- Often a trade-off between safe, sound, and complete.

How Can We Assess Readiness?

- Identifying faults is useful, but finding all faults is nearly impossible.
- Instead, need to decide when to stop verification and validation.
- Need to establish criteria for acceptance.
  - How good is “good enough”?
- One option is to measure dependability (availability, mean time between failures, etc) and set a “acceptability threshold”.

Product Readiness

- Another option is to put it in the hands of human users.
- Alpha/Beta Testing - invite a small group of users to start using the product, have them report feedback and faults. Use this to judge product readiness.
  - Can make use of dependability metrics for a quantitative judgement (metric > threshold).
  - Can make use of surveys as a qualitative judgement (are the users happy with the current product?)

Ensuring Quality of Successive Releases

- Verification and validation 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.
- Faults have not always been fixed before release. Do not forget those.
  - Regression Testing - when code changes, rerun tests to ensure that it still works.
    - As faults are repaired, add tests that exposed them to the suite.

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.

We Have Learned

- Verification can be static or dynamic.
  - Pessimistically or optimistically inaccurate
  - Level of inaccuracy can be controlled by simplifying properties.
  - Desire safe, sound, and complete.
    - Obtaining one often involves losing other.

Next Time

- More on quality
  - Quality Scenarios (high-level test scenarios)
- Measuring and assessing dependability

- Plan your team selection.
  - The earlier, the better! Due January 30, 11:59 PM.
  - Three people, e-mail names/e-mails/team name to ggay@chalmers.se
    - Let me know if you want assigned to a team.

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY