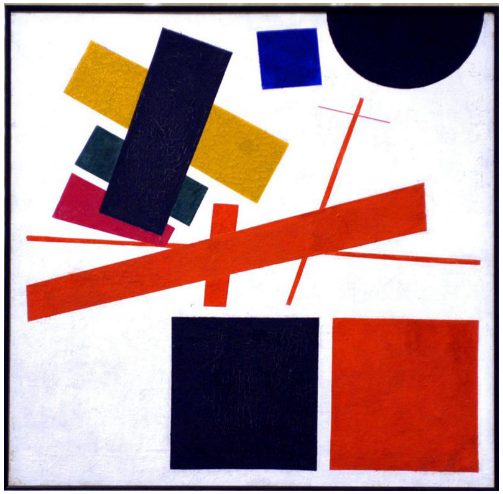
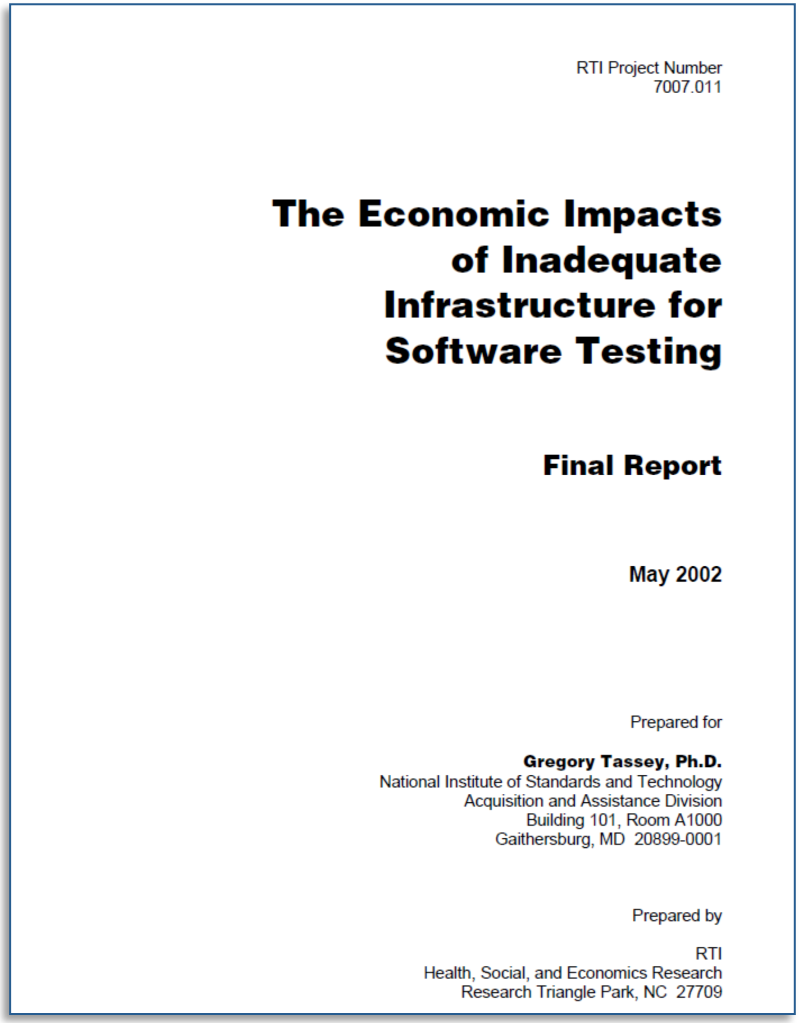
CPEN 422
Software Testing and Analysis



Lecture 02

Can you estimate the cost of failing software systems?

Cost of inadequate testing (US alone)



RTI Project Number
7007.011

The Economic Impacts of Inadequate Infrastructure for Software Testing

Final Report

May 2002

Prepared for

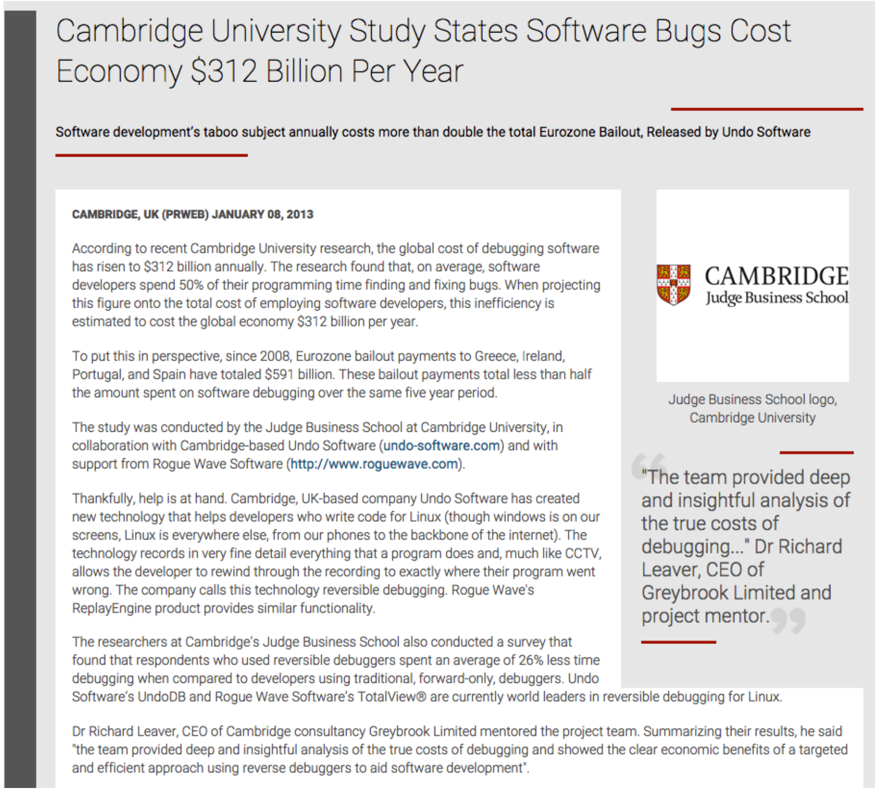
Gregory Tassey, Ph.D.
National Institute of Standards and Technology
Acquisition and Assistance Division
Building 101, Room A1000
Gaithersburg, MD 20899-0001

Prepared by

RTI
Health, Social, and Economics Research
Research Triangle Park, NC 27709

US alone:
59 billion per year

New study in 2013: Cost of Software Bugs



Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year

Software development's taboo subject annually costs more than double the total Eurozone Bailout, Released by Undo Software

CAMBRIDGE, UK (PRWEB) JANUARY 08, 2013

According to recent Cambridge University research, the global cost of debugging software has risen to $312 billion annually. The research found that, on average, software developers spend 50% of their programming time finding and fixing bugs. When projecting this figure onto the total cost of employing software developers, this inefficiency is estimated to cost the global economy $312 billion per year.

To put this in perspective, since 2008, Eurozone bailout payments to Greece, Ireland, Portugal, and Spain have totaled $591 billion. These bailout payments total less than half the amount spent on software debugging over the same five year period.

The study was conducted by the Judge Business School at Cambridge University, in collaboration with Cambridge-based Undo Software (undo-software.com) and with support from Rogue Wave Software (http://www.roguewave.com).

Thankfully, help is at hand. Cambridge, UK-based company Undo Software has created new technology that helps developers who write code for Linux (though windows is on our screens, Linux is everywhere else, from our phones to the backbone of the internet). The technology records in very fine detail everything that a program does and, much like CCTV, allows the developer to rewind through the recording to exactly where their program went wrong. The company calls this technology reversible debugging. Rogue Wave's ReplayEngine product provides similar functionality.

The researchers at Cambridge's Judge Business School also conducted a survey that found that respondents who used reversible debuggers spent an average of 26% less time debugging when compared to developers using traditional, forward-only, debuggers. Undo Software's UndoDB and Rogue Wave Software's TotalView® are currently world leaders in reversible debugging for Linux.

Dr Richard Leaver, CEO of Cambridge consultancy Greybrook Limited mentored the project team. Summarizing their results, he said "the team provided deep and insightful analysis of the true costs of debugging and showed the clear economic benefits of a targeted and efficient approach using reverse debuggers to aid software development".

CAMBRIDGE Judge Business School

Judge Business School logo, Cambridge University

> "The team provided deep and insightful analysis of the true costs of debugging..." Dr Richard Leaver, CEO of Greybrook Limited and project mentor.

Globally
$312 billion per year

http://www.prweb.com/releases/2013/1/prweb10298185.htm

Can testing help?

- Yes, definitely!
- Read “Goto Fail, Heartbleed, and Unit Testing Culture” by Martin Fowler
  - http://martinfowler.com/articles/testing-culture.html

Software Testing

- An activity to assess the quality of a system
- Using simple scenarios that can be understood

Software Testing: Broad Definition

is an

- Empirical technical *investigation*
- conducted to provide *stakeholders* with
- information about the *quality* of
- the software or service under test

Software Testing: Technical Definition

Testing consists of

- the *dynamic* verification of the behavior of a program
- on a finite *set of test cases*
- suitably *selected* from the usually infinite executions domain
- against the specified *expected* behavior

Bertolino,
www.swebok.org

- Each test case is an executable example of system behavior
- Each example can help in stakeholder communication
- Throughout the full development cycle

Today's Objectives

- – Why is testing challenging?
- – Different levels of testing

Software is everywhere

- So are software bugs!

- But why?

- What are **sources of problems**?

Sources of Problems

- **Requirements Definition:** Erroneous, incomplete, inconsistent requirements.
- **Design:** Fundamental design flaws in the software.
- **Implementation:** Programming faults, typos, off-by-one, vulnerable code.
- **Support Systems:** Poor programming languages, faulty compilers and debuggers, misleading development tools (IDEs).

Sources of Problems (Cont’d)

- **Inadequate Testing of Software:** No or incomplete testing, poor verification, mistakes in debugging.

- **Evolution:** Sloppy redevelopment or maintenance, ***introduction of new flaws in attempts to fix old flaws or add new features***.

Adverse Effects of Faulty Software

- **Communications:** Loss of communication media, non delivery of data.
- **Space Applications:** Lost lives, launch delays.
- **Defense and Warfare:** Misidentification of friend (friendly fire!).

Adverse Effects of Faulty Software (Cont’d)

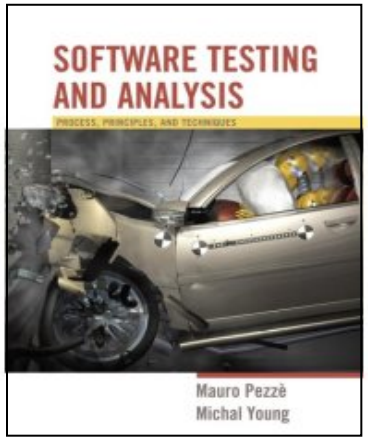
- **Money Management:** Fraud, violation of privacy, shutdown of stock exchanges and banks, negative interest rates.
- **Control of Elections:** Wrong results (intentional or non-intentional).
- **Transportation:** Deaths, delays, sudden acceleration, inability to brake.

Discussion …

- Do you think bug free software is achievable?
  - Are their technical barriers that make this impossible?
  - Is it just a question of time before we can do this?
  - Are we missing technology or processes?

Chapter 2: Verification and Validation

2.1


SOFTWARE TESTING AND ANALYSIS
PROCESS, PRINCIPLES, AND TECHNIQUES
Mauro Pezzè
Michal Young

- Validation: does the software system meet the user's real needs?

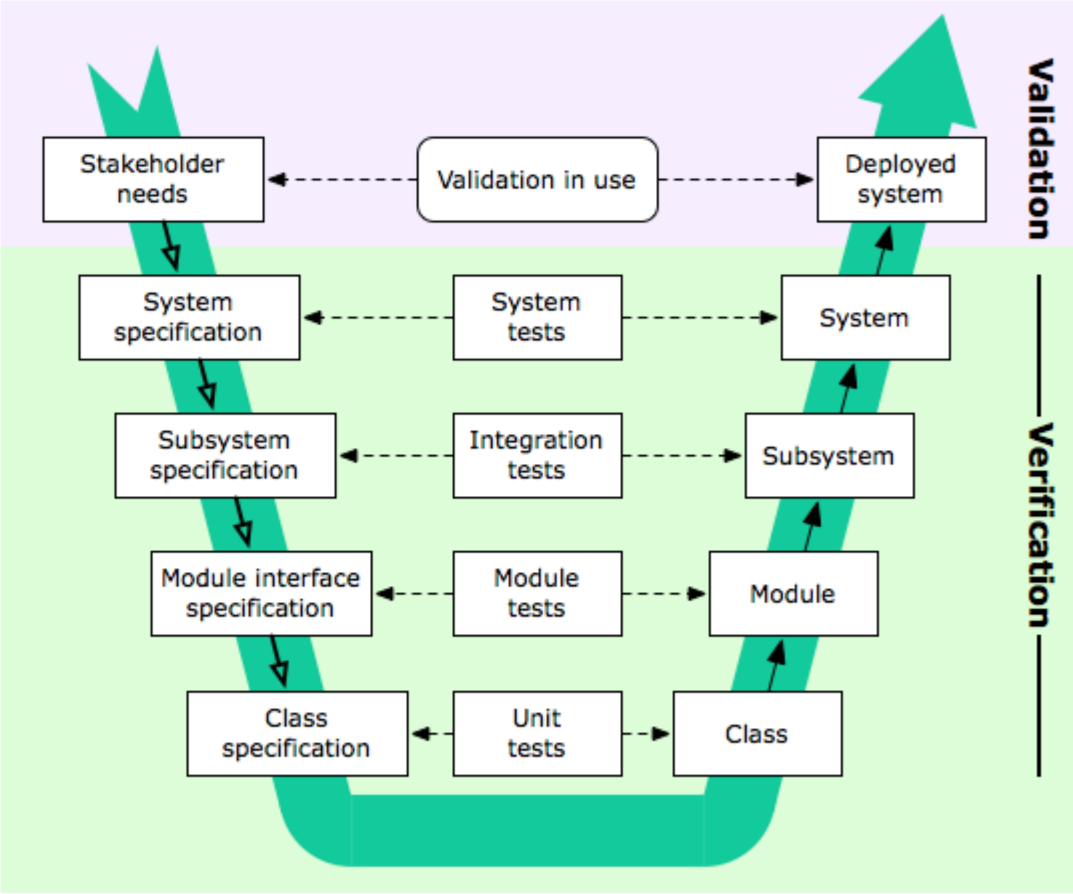
  Are we building the right software?

- Verification: does the software system meet the requirements specifications?

  Are we building the software right?

Validation

Testing Spectrum



Verification

Key

A ↓ **B** — **B** elaborates the means by which **A** can be satisfied

S <- - **T** - -> **C** — If **C** passes **T**, then **C** meets **S** sufficiently well

A ↑ **B** — **B** must meet its specification before **A** can

Static versus Dynamic Analysis

Static Analysis:

- Determines or estimates software quality ***without*** reference to actual executions
- Techniques: code inspection, program analysis, symbolic analysis, and model checking.

Dynamic Analysis:

- Ascertains and/or approximates software quality ***through*** actual executions, i.e., with real data and under real (or simulated) circumstances
- Techniques: synthesis of inputs, testing procedures, and the automation of testing environment generation.

Functional versus Structural testing

- Differences?

- AKA: Black box versus White box testing

Functional versus Structural Testing

Functional Testing:

- software program or system under test is viewed as a “**black box**”.
- emphasizes on the **external** behavior of the software entity.
- selection of test cases for functional testing is based on the **requirement** or design **specification** of the software entity under test.

Functional versus Structural Testing

Structural Testing:

- the software entity is viewed as a “**white box**”.
- emphasizes on the **internal** structure of the software entity.
- goal of selecting such test cases is to cause the execution of specific **statements**, program **branches** or **paths**.
- expected results are evaluated on a set of **coverage criteria**. Examples: path coverage, branch coverage, and data-flow coverage.

Levels of Automation

- Manual Testing (no automation)
  - Code review, inspection, exploratory testing
- Test Scripting (automated test execution)
  - Writing unit tests
- Test Generation (Automated test creation)
  - Using a tool to generate test cases

Testing Terminology

- **Failure/fault/error** (or bug)
- **Test plan**: test specification
- **Test case**: a single unique unit of testing code
- **Test suite**: collection of test case
- **Test oracle**: expected behavior
- **Test fixture** (or test data)
- **Test harness:** collection of all the above + conf.

Test Oracles

Common types of oracles:

- an expert human being
- Assertions: assertEquals(x, 34);
- specifications and documentation (e.g., invariants)
- other programs (e.g., program that uses a different algorithm to evaluate the same expression as the product under test)
- a heuristic oracle that provides approximate results or exact results for a set of test inputs,
- a consistency oracle that compares the results of one test execution to another for similarity (regression testing),

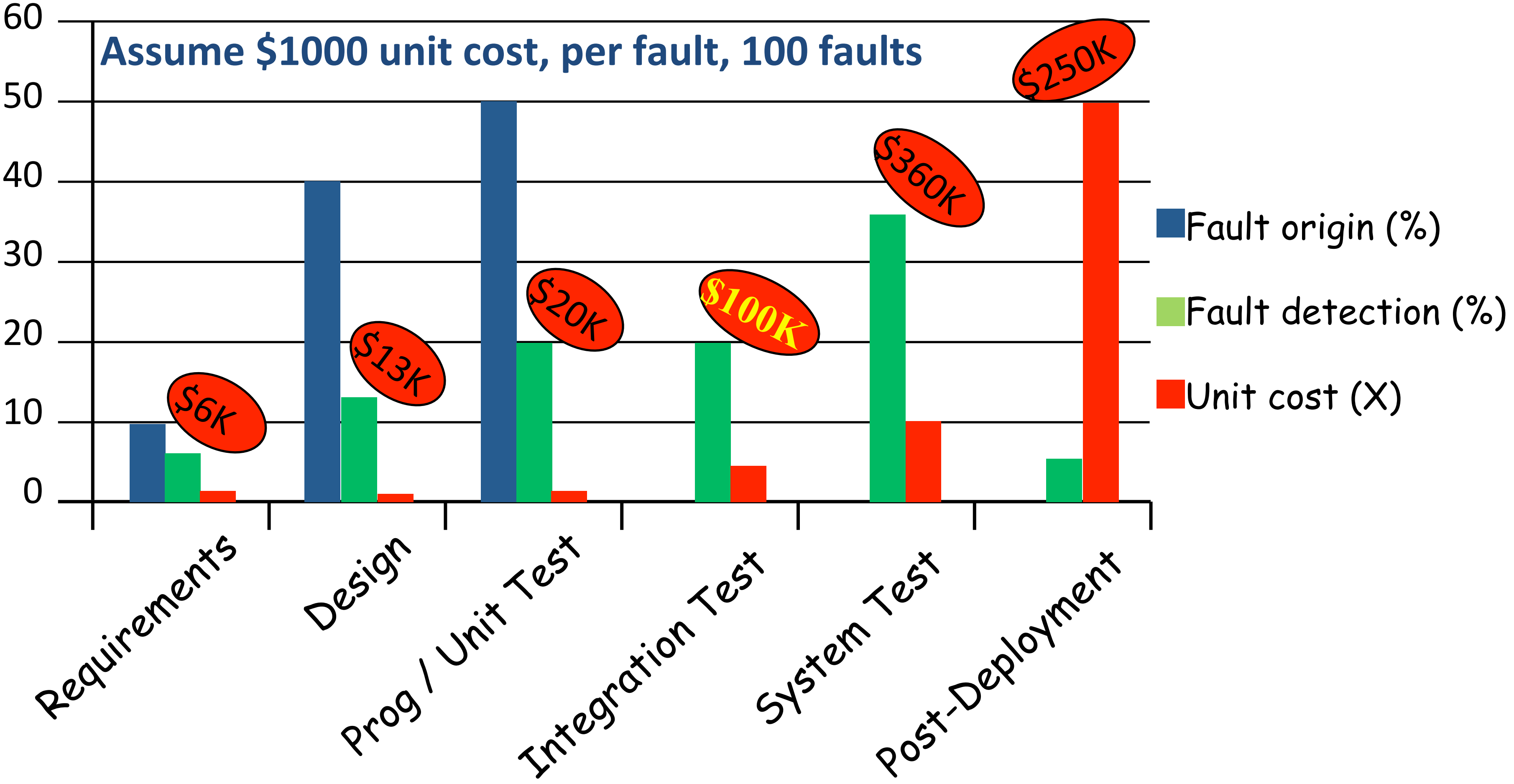
Given that:

- Each test case is one executable **example** of system behavior
- Either functional or structural

Identify 3 main testing challenges

Discuss with your neighbor

“It's all done. I just need to test it.”



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

Questions?