CS409
Software Testing

TAN, Shin Hwei
陈馨慧
Southern University of Science and Technology

Slides adapted from cs498dm (UIUC) and cs4218 (NUS)

Teaching Staff

- Instructor: Shin Hwei Tan (陈馨慧)
- Email: tansh3@sustc.edu.cn (subject cs409)
  - Office: 创园605
  - Office hours: after lecture (Monday 2pm)
- TA/Grader:
  - Ziqiang Li（李子强）

Administrative Info

- Lecture: Monday 10:20am-12:10pm
- Lab: Monday 4:20pm-6:10pm
- Credit:
  - 32 Lectures + 32 Lab
- Prerequisites: CS304. 软件工程 Software Engineering

Course Overview

- Introduction to software testing
  - Systematic, organized approaches to testing
  - Based on models and coverage criteria
  - Testing is not (only) about finding “bugs”
  - Improve your testing (and development) skills
- Several homeworks
- Project: testing some real software
  - Open-source Android apps: Google I/O. F-droid
- Exams

Syllabus

Week 1: Introduction
Week 2: Example Testing Session
Week 3: Unit Testing (JUnit)
Week 4: Test Driven Development (TDD)
Week 5: Statement Coverage
Week 6: Graph Coverage
Week 7: Data flow coverage
Week 8: Graph Coverage for Specifications
Week 9: Logic Testing
Week 10: Logic Coverage
Week 11: Input Space Partitioning
Week 12: Mutation Testing
Week 13: Regression Testing / Z3 Solver
Week 14: Continuous Integration/ Program Synthesis
Week 15: Finding bugs in Java programs with Automated Testing Tools
Week 16: Finding bugs in Android apps with Automated Testing Tools

May Change to new materials

Grading

- Points
  - Assignment (40%)
  - Exams (20%)
  - Project (20%)
  - Project Final Presentation (20%)
- Grades
  - A*(90%), B*(80%), C*(70%), D*(60%), F(<60%)
  - For more details, see the syllabus
  - The instructor may lower the point limits

Project

- Testing some real software
  - Groups of 3 students
  - Select from the list of open-source Android apps
- Deliverables
  - Proposal (due in three weeks)
  - Progress report (around mid of semester)
  - Final presentation and report (end of semester)
  - Bug reports (hopefully you'll find some bugs)
    - Extra bonus points for reporting NEW bugs!

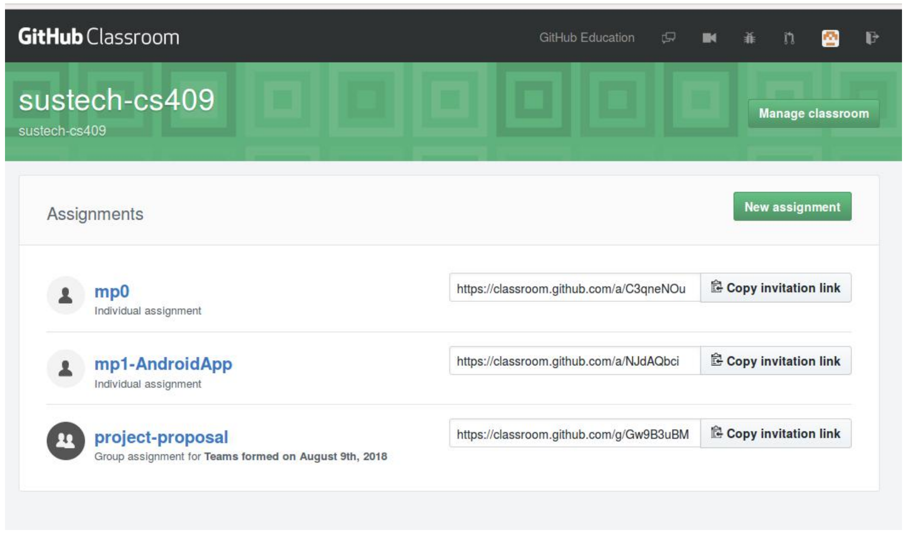
Collaboration

- You must individually solve homework assignments
- You can collaborate on everything else (unless explicitly stated not to collaborate!)
  - Discuss homework assignments (but **mention** you discussed and **don’t code/write together**)
    - Plagiarism is a serious breach of ethics
  - Do projects in groups
- Testing is a social activity
  - Communication matters

Course Communication

- GitHub Classroom
  - Share the bugs that your find! (Remember to document when you find a bug through knowledge shared in GitHub discussion!)
  - Helps each other with tools installation
- Instructor: Shin Hwei Tan
  - Email: tansh3@sustc.edu.cn (subject cs409)
- Office: 创园605
  - Office hours: after lecture

GitHub Classroom



sustech-cs409
sustech-cs409

Assignments

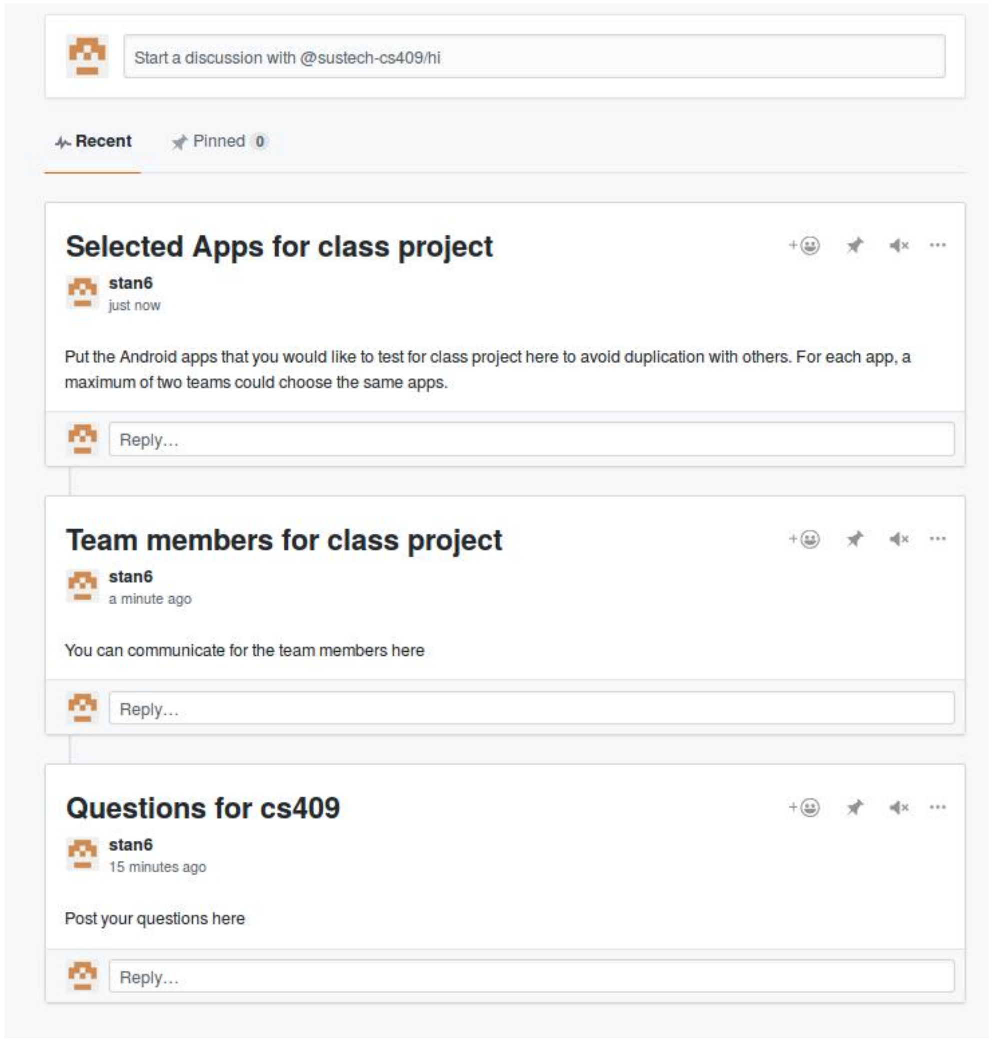
mp0
Individual assignment
https://classroom.github.com/a/C3qneNOu

mp1-AndroidApp
Individual assignment
https://classroom.github.com/a/NJdAQbci

project-proposal
Group assignment for **Teams formed on August 9th, 2018**
https://classroom.github.com/g/Gw9B3uBM

- Automates repository creation and access control
  - Easy to distribute starter code and collect assignments on GitHub.

Discussion through GitHub



Start a discussion with @sustech-cs409/hi

Selected Apps for class project

stan6
just now

Put the Android apps that you would like to test for class project here to avoid duplication with others. For each app, a maximum of two teams could choose the same apps.

Team members for class project

stan6
a minute ago

You can communicate for the team members here

Questions for cs409

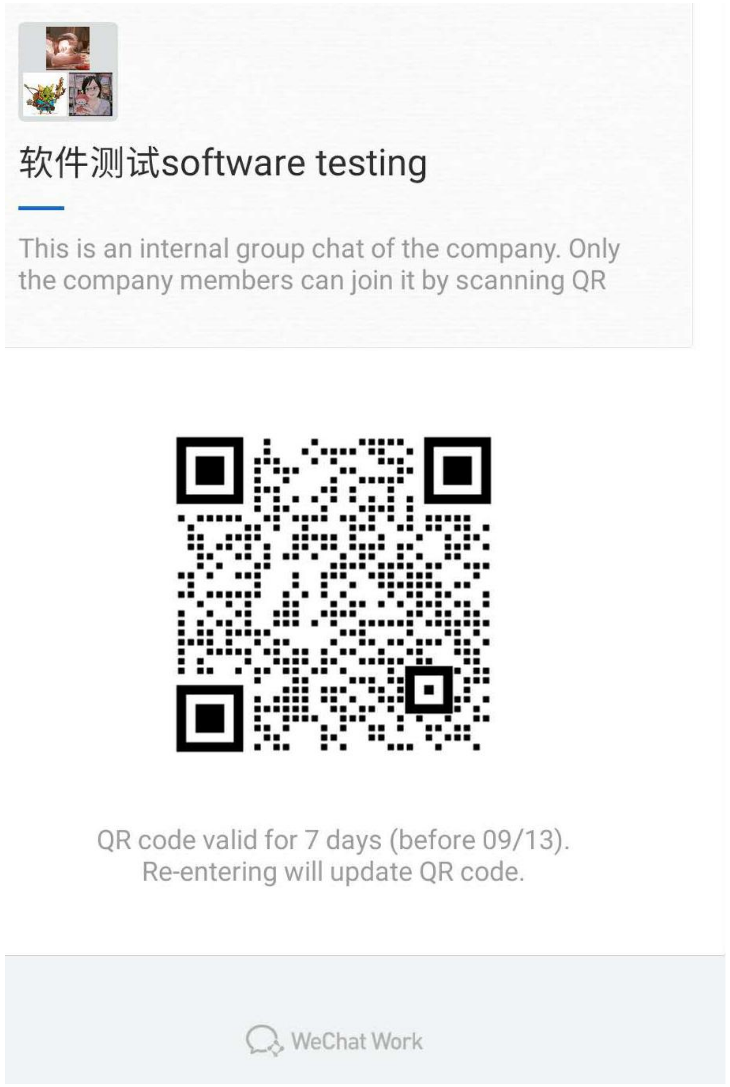
stan6
15 minutes ago

Post your questions here

Signup Sheet

- Name
- Email address (ID)
- Program/Year
- Interests: what would you like to learn about testing?
- Experience: what testing did you do?

Group Wechat



软件测试software testing

This is an internal group chat of the company. Only the company members can join it by scanning QR

QR code valid for 7 days (before 09/13).
Re-entering will update QR code.

WeChat Work

Textbook

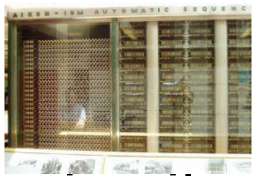
- "Introduction to Software Testing"
  by Paul Ammann and Jeff Offutt
  Cambridge University Press, Jan. 2008
  http://cs.gmu.edu/~offutt/softwaretest/

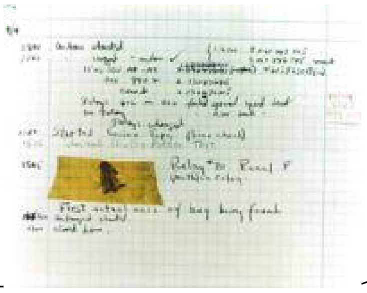
- Book should be available in the library

What is a computer bug?

What is a computer bug?

- In 1947 Harvard University was operating a room-sized computer called the Mark II.
  - mechanical relays
  - glowing vacuum tubes
  - technicians program the computer by reconfiguring it
  - Technicians had to change the occasional vacuum tube.
- A moth flew into the computer and was zapped by the high voltage when it landed on a relay.
- Hence, the first computer bug!
  - I am not making this up :-)




[illegible]
Relay #70 Panel F
[illegible]
First actual case of bug being found.
[illegible]

Bugs a.k.a. …

- Defect
- Fault
- Problem
- Error
- Incident
- Anomaly
- Variance
- Failure
- Inconsistency
- Product Anomaly
- Product Incidence
- Feature :-)

This Lecture: Introduction to “Bugs”

- Why look for bugs?
- What are bugs?
- Where they come from?
- How to detect them?

Some Costly "Bugs"

- NASA Mars space missions
  - Priority inversion (2004)
  - Different metric systems (1999)
- BMW airbag problems (1999)
  - Recall of 15,000+ cars
- Ariane 5 crash (1996)
- Your own favorite examples?

Where is the bug?

- What will be printed?
- What type of bug is this?

```
 public static void main(String[] args) {

        final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 *
1000;
        System.out.println("MICROS_PER_DAY : " +
MICROS_PER_DAY);

    }
```

Where is the bug?

- What will be printed?
- What type of bug is this?

```
 public static void main(String[] args) {

       final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 *
1000;
       System.out.println("MICROS_PER_DAY : " +
MICROS_PER_DAY);

   }
```

It will print 500654080 but the correct answer should be 86,400,000,000!

- ➢ Integer Overflow
- ➢ change 24 to 24L

Ariane 5 crash (1996)

- Uncaught exception of numerical overflow

Fatal Uber crash (2018)

- Software error in autonomous vehicles

Defective Software

- We develop programs that contain defects
  - How many? What kind?
- Hard to predict the future, however…
  it is highly likely, that the software we (including you!) will develop in the future will not be significantly better.

Sources of Problems

- **Requirements Definition:** Erroneous, incomplete, inconsistent requirements.
- **Design:** Fundamental design flaws in the software.
- **Implementation:** Mistakes in chip fabrication, wiring, programming faults, malicious code.
- **Support Systems:** Poor programming languages, faulty compilers and debuggers, misleading development tools.

Sources of Problems (Cont'd)

- **Inadequate Testing of Software:** Incomplete testing, poor verification, mistakes in debugging.
- **Evolution:** Sloppy redevelopment or maintenance, introduction of new flaws in attempts to fix old flaws, incremental escalation to inordinate complexity.

Adverse Effects of Faulty Software

- **Communications:** Loss or corruption of communication media, non delivery of data.
- **Space Applications:** Lost lives, launch delays.
- **Defense and Warfare:** Misidentification of friend or foe.

Adverse Effects of Faulty Software (Cont'd)

- **Transportation:** Deaths, delays, sudden acceleration, inability to brake.
- **Safety-critical Applications:** Death, injuries.
- **Electric Power:** Death, injuries, power outages, long-term health hazards (radiation).

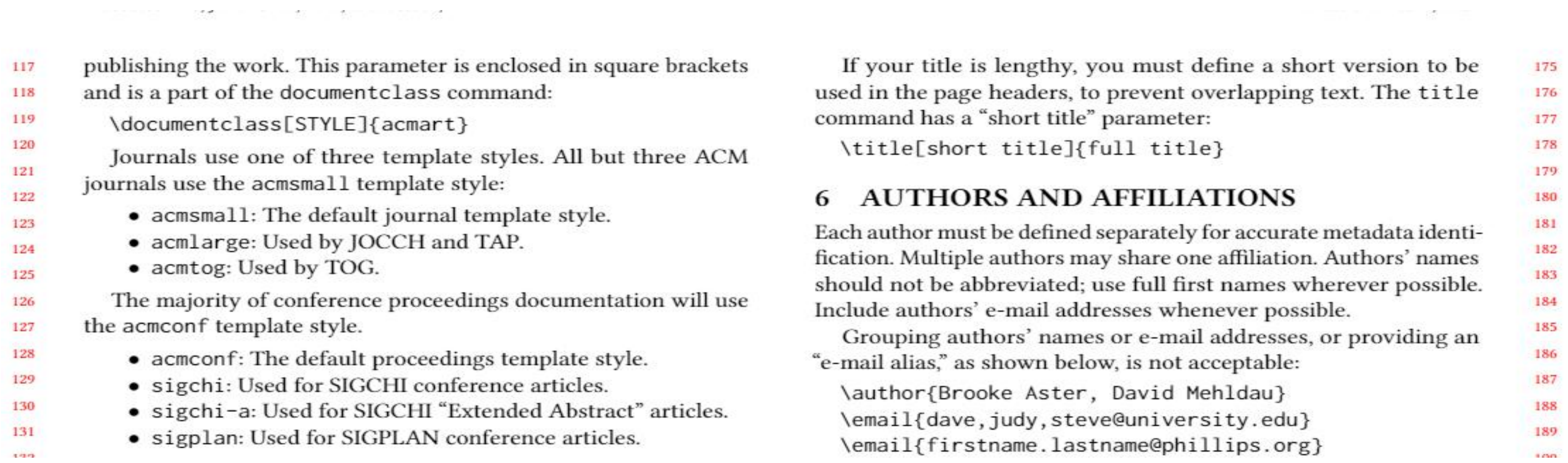
Adverse Effects of Faulty Software (Cont'd)

- **Money Management:** Fraud, violation of privacy, shutdown of stock exchanges and banks, negative interest rates.
- **Control of Elections:** Wrong results (intentional or non-intentional).
- **Control of Jails:** Technology-aided escape attempts and successes, accidental release of inmates, failures in software controlled locks.
- **Law Enforcement:** False arrests and imprisonments.

Some “Bugging” Bugs

- Smaller issues that give unexpected results
  - My example bug(s)
- Your own favorite examples?
- Rule(?) for electronics in the classroom: you can use your laptop/tablet/phone (/game console) if you show some bug

A recent bug that I found: What is the difference between these pictures?



publishing the work. This parameter is enclosed in square brackets and is a part of the documentclass command:

```
\documentclass[STYLE]{acmart}
```

Journals use one of three template styles. All but three ACM journals use the acmsmall template style:

- acmsmall: The default journal template style.
- acmlarge: Used by JOCCH and TAP.
- acmtog: Used by TOG.

The majority of conference proceedings documentation will use the acmconf template style.

- acmconf: The default proceedings template style.
- sigchi: Used for SIGCHI conference articles.
- sigchi-a: Used for SIGCHI "Extended Abstract" articles.
- sigplan: Used for SIGPLAN conference articles.

If your title is lengthy, you must define a short version to be used in the page headers, to prevent overlapping text. The title command has a "short title" parameter:

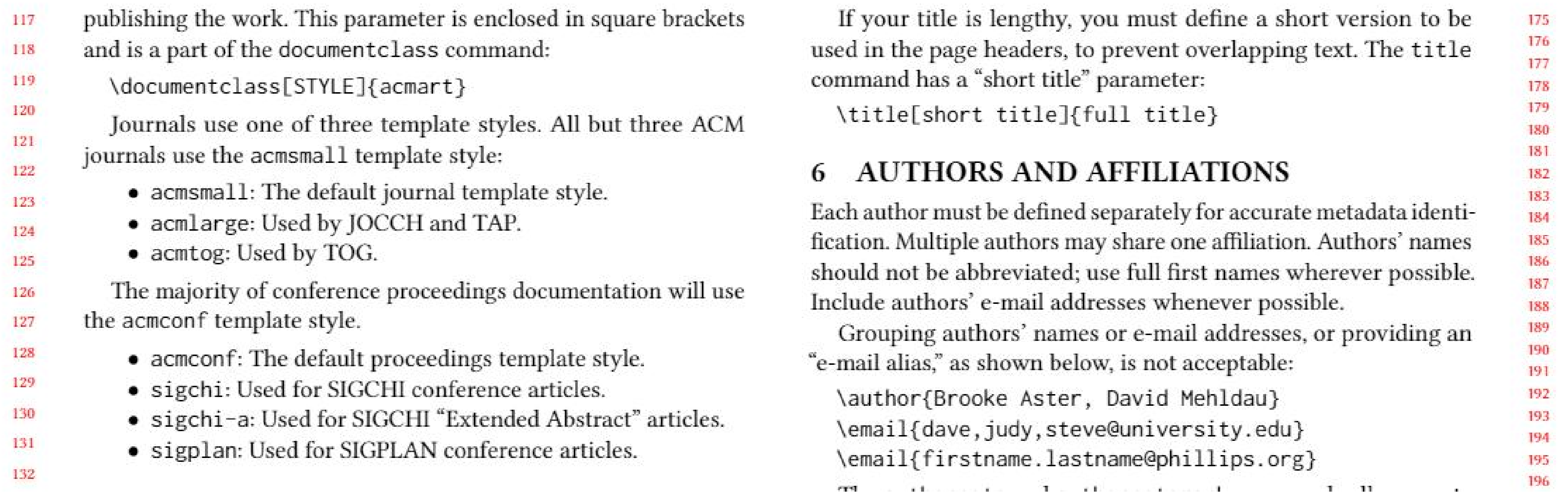
```
\title[short title]{full title}
```

6 AUTHORS AND AFFILIATIONS

Each author must be defined separately for accurate metadata identification. Multiple authors may share one affiliation. Authors' names should not be abbreviated; use full first names wherever possible. Include authors' e-mail addresses whenever possible.

Grouping authors' names or e-mail addresses, or providing an "e-mail alias," as shown below, is not acceptable:

```
\author{Brooke Aster, David Mehldau}
\email{dave,judy,steve@university.edu}
\email{firstname.lastname@phillips.org}
```



publishing the work. This parameter is enclosed in square brackets and is a part of the documentclass command:

```
\documentclass[STYLE]{acmart}
```

Journals use one of three template styles. All but three ACM journals use the acmsmall template style:

- acmsmall: The default journal template style.
- acmlarge: Used by JOCCH and TAP.
- acmtog: Used by TOG.

The majority of conference proceedings documentation will use the acmconf template style.

- acmconf: The default proceedings template style.
- sigchi: Used for SIGCHI conference articles.
- sigchi-a: Used for SIGCHI "Extended Abstract" articles.
- sigplan: Used for SIGPLAN conference articles.

If your title is lengthy, you must define a short version to be used in the page headers, to prevent overlapping text. The title command has a "short title" parameter:

```
\title[short title]{full title}
```

6 AUTHORS AND AFFILIATIONS

Each author must be defined separately for accurate metadata identification. Multiple authors may share one affiliation. Authors' names should not be abbreviated; use full first names wherever possible. Include authors' e-mail addresses whenever possible.

Grouping authors' names or e-mail addresses, or providing an "e-mail alias," as shown below, is not acceptable:

```
\author{Brooke Aster, David Mehldau}
\email{dave,judy,steve@university.edu}
\email{firstname.lastname@phillips.org}
```

A bug with the TeX template? If you agree, should this be forwarded to Boris then?

From: Shin Hwei Tan <tansh3@sustech.edu.cn>
Sent: Wednesday, August 21, 2019 9:05 AM
To: production <production@hq.acm.org>
Subject: Important Bugs in the formatting of line numbers for the latest latex template

Dear ACM officer,

I have found a bug in the recently updated acm-sigconf Latex template (last update August 6, 2019)LATEX (Version 1.63a).

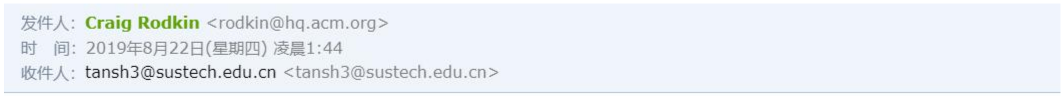
Below is the description of the bug:

When I add the review options in the sample-sigconf.tex (using \documentclass[sigconf,review]{acmart}), I realized that starting from the second page of the compiled PDF, the line numbers in the left of the PDF and the line numbers in the right of the PDF are not the same. Particularly, as shown in the attached image 1174580209.jpg, the spacing between the line numbers at the right side of the PDF is very narrow compared to the spacing between the line numbers at the left side of the PDF.

Moreover, I find that this problem doesn't exist in the overleaf version of the Latex template. If I use the built-in Latex template in overleaf without adding the new acmart.cls, then the line numbers are rendered correctly. I realized that others users of the template also faced the same problems so this issue have affected many users who are rushing for a conference deadline.

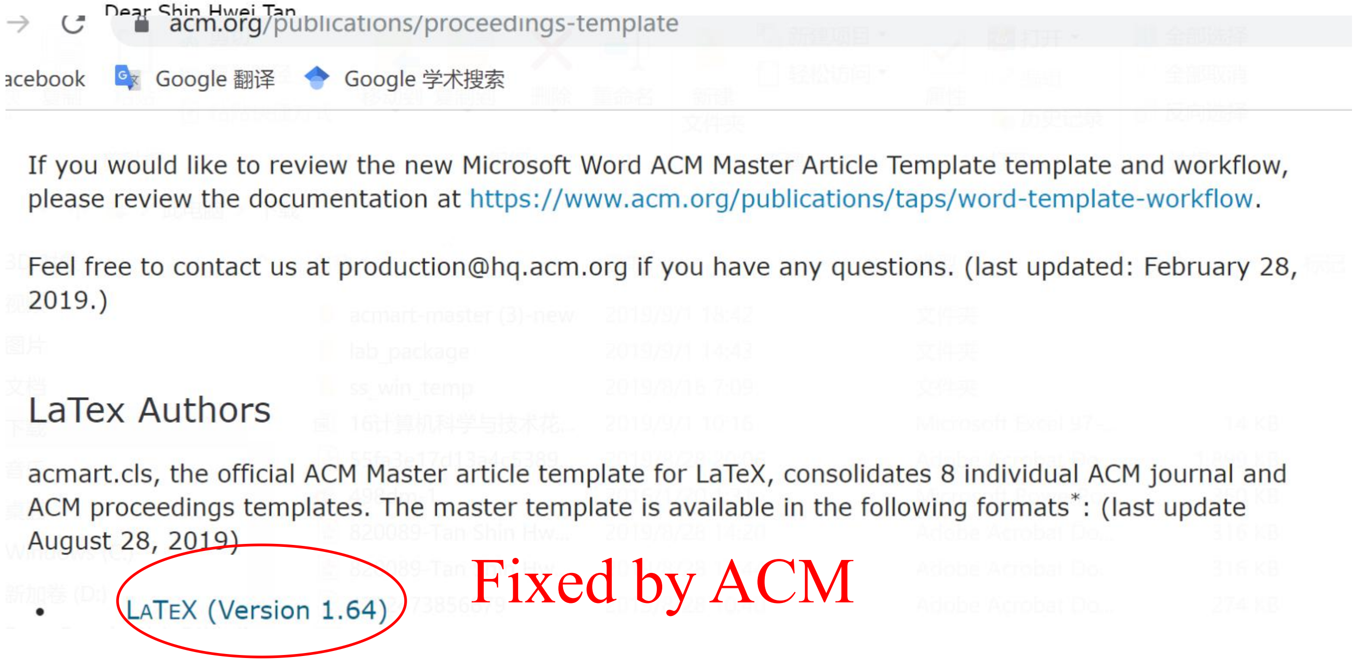
Could you please help me to fix this issue?

Thank you.



发件人: **Craig Rodkin** <rodkin@hq.acm.org>
时　间: 2019年8月22日(星期四) 凌晨1:44
收件人: tansh3@sustech.edu.cn <tansh3@sustech.edu.cn>

Dear Shin Hwei Tan



If you would like to review the new Microsoft Word ACM Master Article Template template and workflow, please review the documentation at https://www.acm.org/publications/taps/word-template-workflow.

Feel free to contact us at production@hq.acm.org if you have any questions. (last updated: February 28, 2019.)

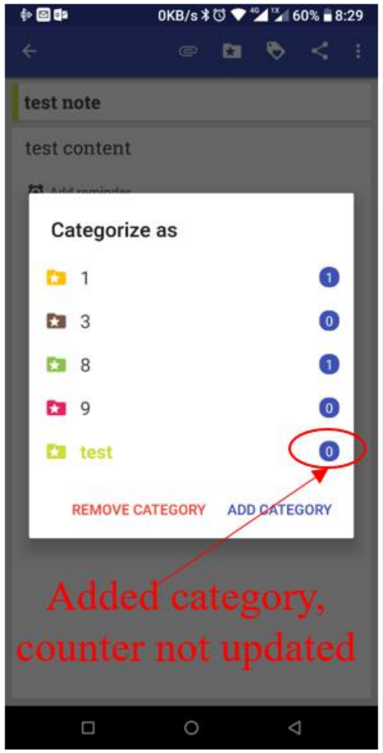
LaTex Authors

acmart.cls, the official ACM Master article template for LaTeX, consolidates 8 individual ACM journal and ACM proceedings templates. The master template is available in the following formats*: (last update August 28, 2019)

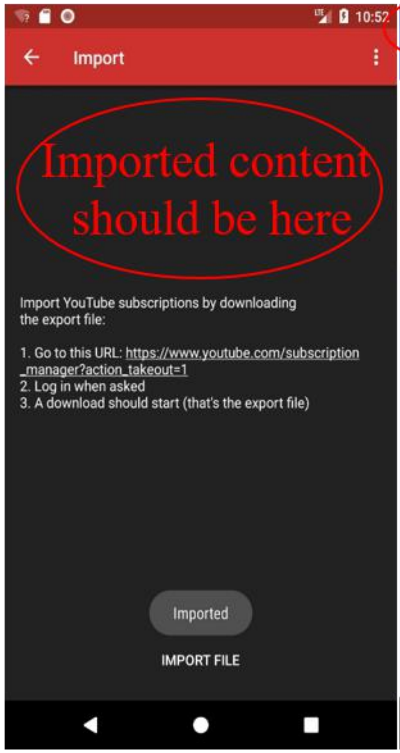
- LATEX (Version 1.64)

Fixed by ACM

Bugs found by students in cs409 (2018)



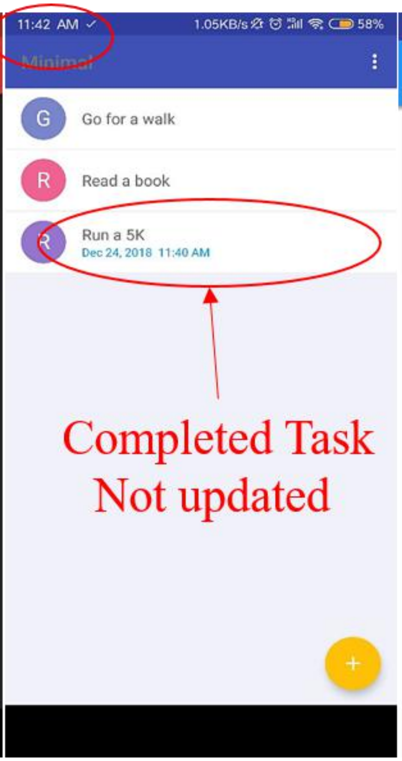
Added category, counter not updated



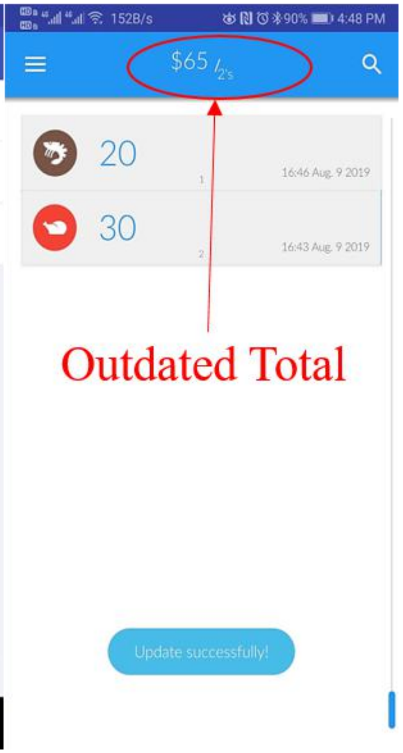
Imported content should be here

Import YouTube subscriptions by downloading the export file:

1. Go to this URL: https://www.youtube.com/subscription_manager?action_takeout=1
2. Log in when asked
3. A download should start (that's the export file)



Completed Task Not updated



Outdated Total

Outdated view!

Where is the bug?

- Where is the bug?
- What type of bug is this?

```
char exampleArray[] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd' };

for(int i = 0; i <= 11; i++)
{
  print(exampleArray[i])
}
```

Motivation for Testers

- A slide from Debra Richardson, a professor at UC Irvine: “Analysis and Testing are Creative”

Analysis and Testing are Creative

- Testing and analysis are important, difficult, and stimulating
- Good testing requires as much skill and creativity as good design, because testing is design.
- Testers should be chosen from the most talented employees
  - It is a competitive advantage to produce a high-quality product at acceptable, predictable cost
- Design the product and process for test
  - The process: for visibility, improvement
  - The product: for testability at every stage

Terminology

- Anomaly
- Bug
- Crash
- Defect
- Error
- Failure, fault
- Glitch
- Hang
- Incorrectness
- Jitter

Dynamic vs. Static

- Incorrect (observed) behavior
  - Failure, fault
- Incorrect (unobserved) state
  - Error, latent error
- Incorrect lines of code
  - Fault, error

“Bugs” in IEEE 610.12-1990

- Fault
  - Incorrect lines of code
- Error
  - Faults cause incorrect (unobserved) state
- Failure
  - Errors cause incorrect (observed) behavior

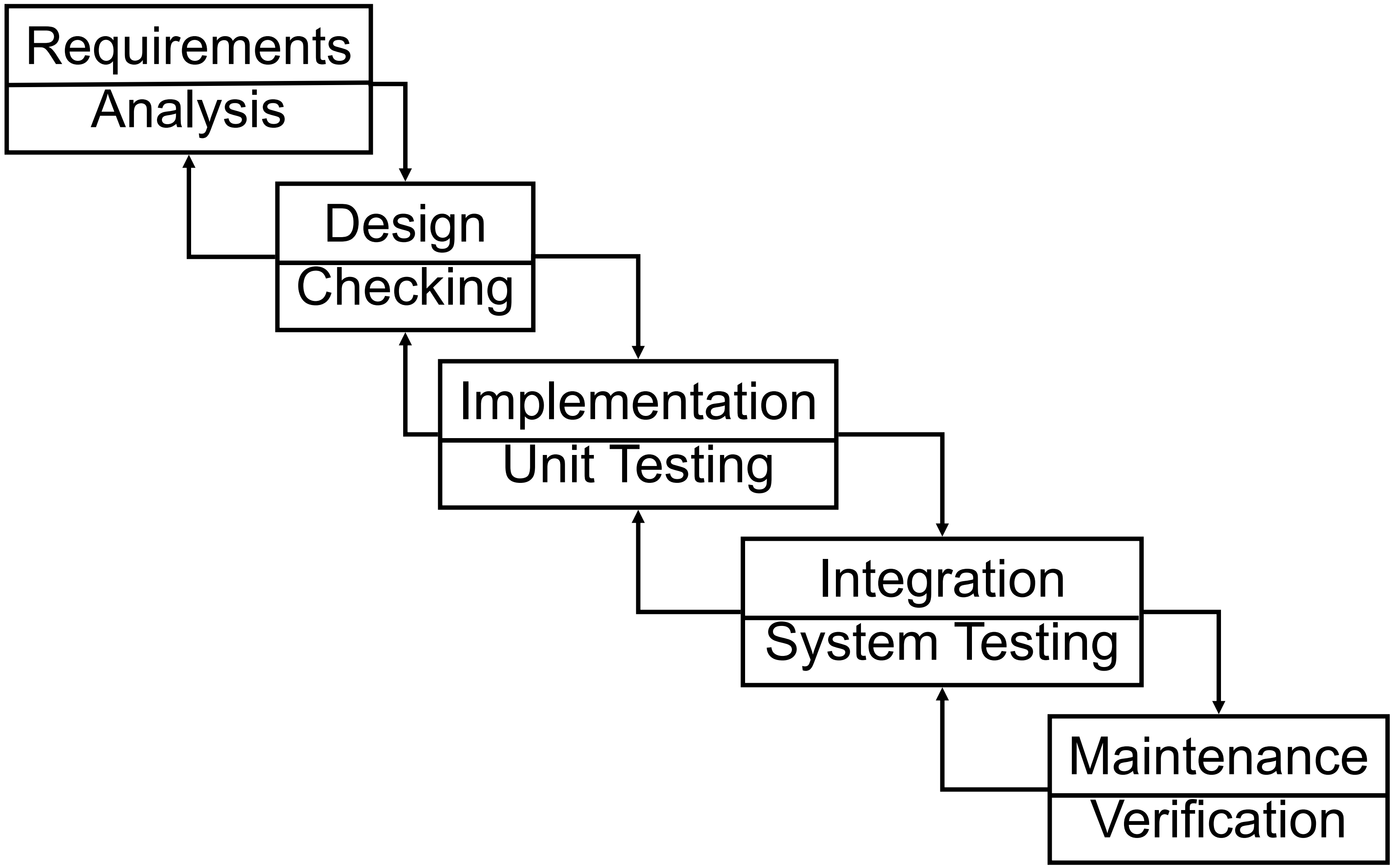
- Not used consistently in literature!

Correctness and Quality

- Common (partial) properties
  - Segfaults, uncaught exceptions
  - Resource leaks
  - Data races, deadlocks
  - Statistics based

- Specific properties
  - Requirements
  - Specification

Traditional Waterfall Model



Phases (1)

- Requirements
  - Specify what the software should do
  - Analysis: eliminate/reduce ambiguities, inconsistencies, and incompleteness
- Design
  - Specify how the software should work
  - Split software into modules, write specifications
  - Checking: check conformance to requirements, using for example conformance testing

Phases (2)

- Implementation
  - Specify how the modules work
  - Unit testing: test each module in isolation
- Integration
  - Specify how the modules interact
  - Integration testing: test module interactions
  - System testing: test the entire system
- Maintenance
  - Evolve software as requirements change
  - Regression testing: test changes

Testing Effort

- Reported to be >50% of development cost [e.g., Beizer 1990]

- Microsoft: 75% time spent testing
  - 50% testers who spend all time testing
  - 50% developers who spend half time testing

When to Test

- The later a bug is found, the higher the cost
  - Orders of magnitude increase in later phases
  - Also the smaller chance of a proper fix

- Old saying: test often, test early
- New methodology: test-driven development (write tests before code)

Software is Complex

- Malleable
- Intangible
- Abstract
- Solves complex problems
- Interacts with other software and hardware
- Not continuous

Software Still Buggy

- Folklore: 1-10 (residual) bugs per 1000 NBNC (LOC, no blank lines of code and no comments) lines of code (after testing)

- Consensus: total correctness impossible to achieve for (complex) software
  - Risk-driven finding/elimination of bugs
  - Focus on specific correctness properties

Approaches for Detecting Bugs

- Software testing
- Model checking
- (Static) program analysis

Software Testing

- Dynamic approach
- Run code for some inputs, check outputs
- Checks correctness for some executions

- Main questions
  - Test-input generation
  - Test-suite adequacy
  - Test oracles

Verification and validation

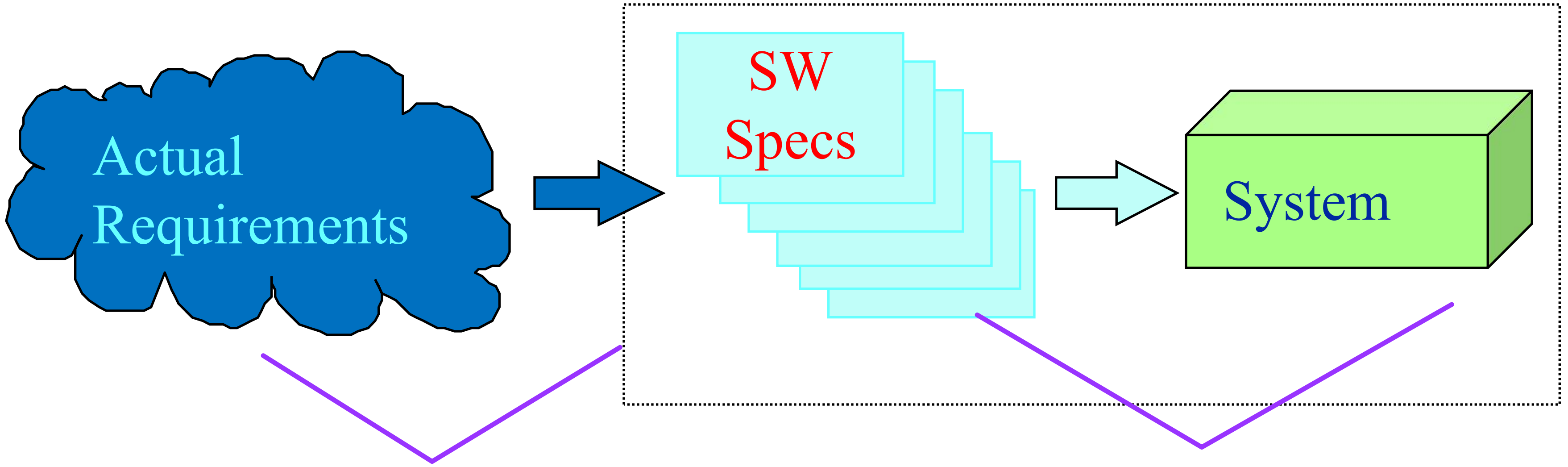
- Validation:
  does the software system meets the user's real needs?

  are we building the right software?

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

  are we building the software right?

Validation and Verification



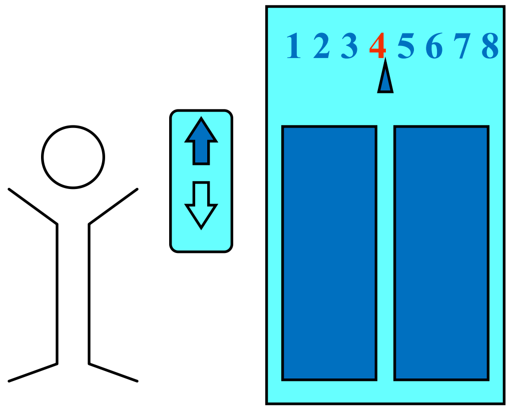
Validation

Includes usability testing, user feedback

Verification

Includes testing, inspections, static analysis

Verification or validation depends on the specification

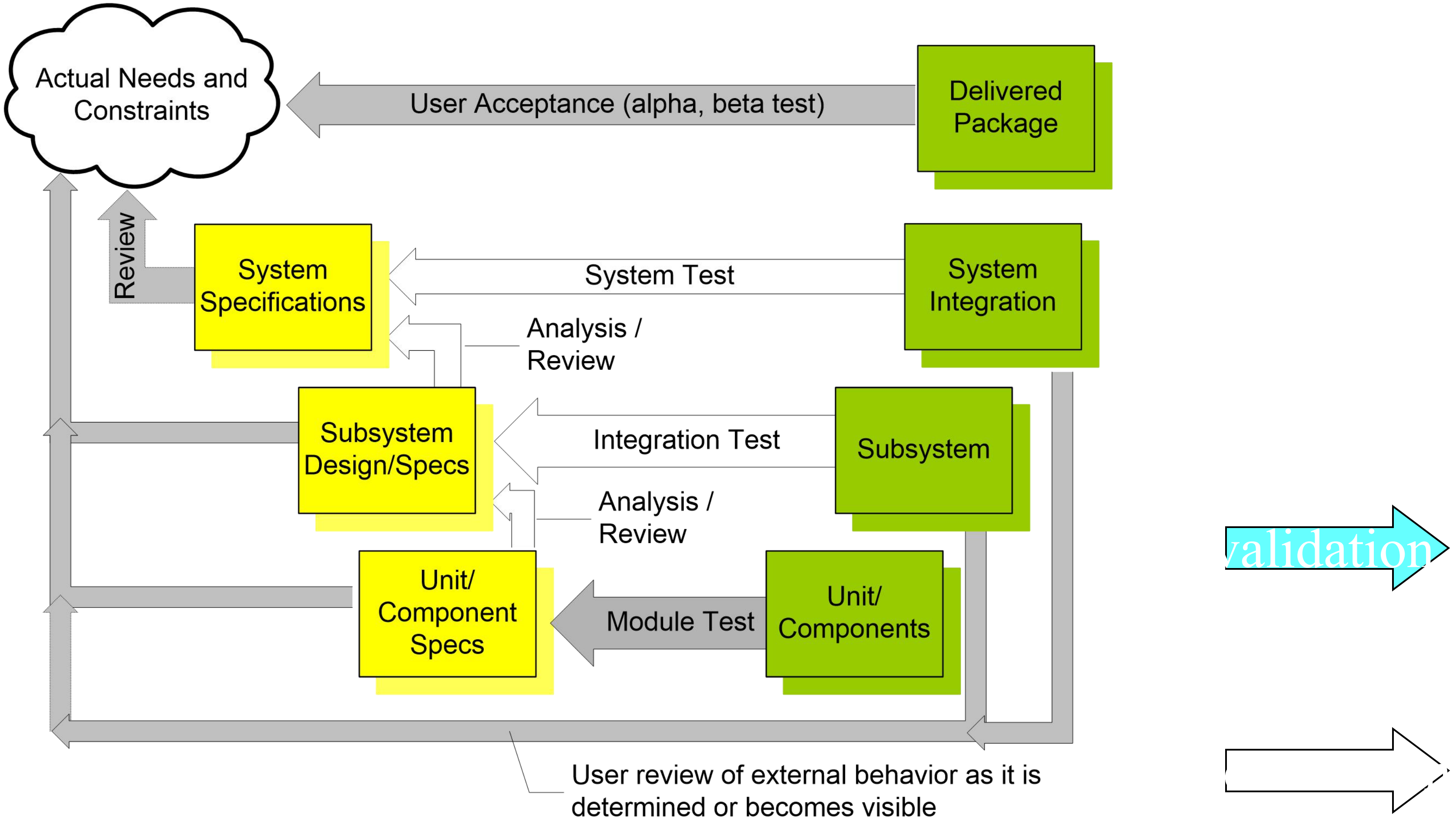


Example: elevator response

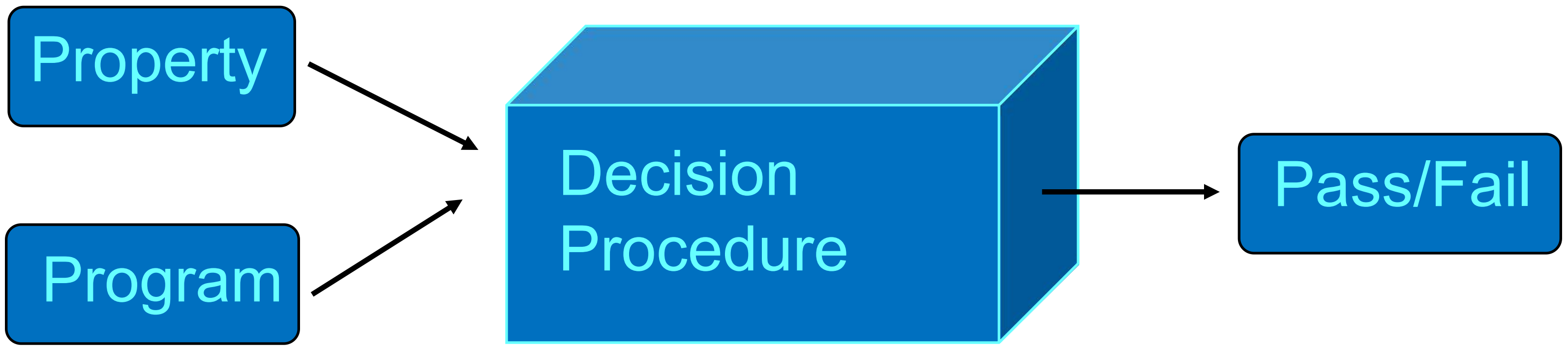
Unverifiable (but validatable) spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i soon...

Verifiable spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i within 30 seconds

Validation and Verification Activities



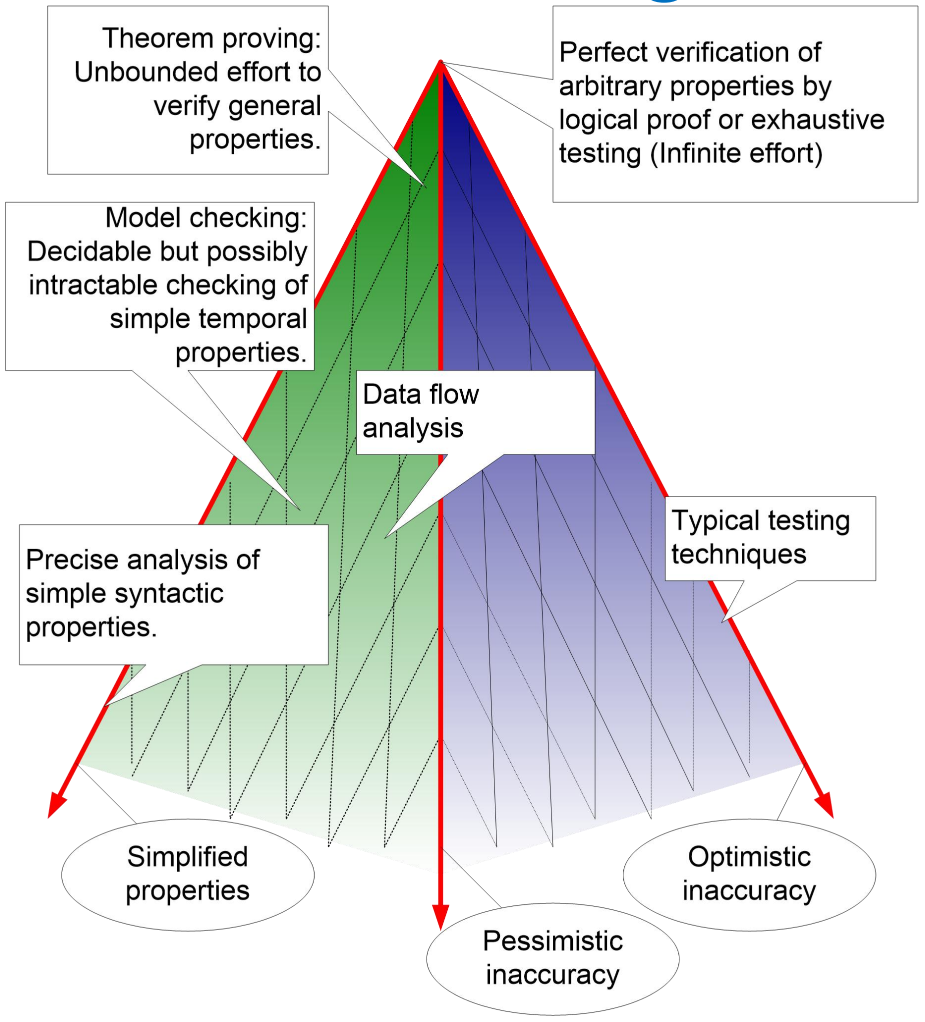
You can't ~~always~~ *ever* get what you want



Correctness properties are undecidable

the halting problem can be embedded in almost every property of interest

Getting what you need ...


Theorem proving: Unbounded effort to verify general properties.
Perfect verification of arbitrary properties by logical proof or exhaustive testing (Infinite effort)
Model checking: Decidable but possibly intractable checking of simple temporal properties.
Data flow analysis
Typical testing techniques
Precise analysis of simple syntactic properties.
Simplified properties
Pessimistic inaccuracy
Optimistic inaccuracy

- optimistic inaccuracy: we may accept some programs that do not possess the property (i.e., it may not detect all violations).
  - testing
- pessimistic inaccuracy: it is not guaranteed to accept a program even if the program does possess the property being analyzed
  - automated program analysis techniques
- simplified properties: reduce the degree of freedom for simplifying the property to check

Other Testing Questions

- Maintenance
- Selection
- Minimization
- Prioritization
- Augmentation
- Evaluation
- Fault Characterization
- …

Current Status

- Testing remains the most widely used approach for finding bugs
  - Testing is gaining importance with test-first development and increased reliability needs
- A lot of recent progress (within last decade) on model checking and static analysis
  - Model checking: from hardware to software
  - Static analysis: from sound to practical
- Vibrant research in the area
  - Gap between research and practice

Topics Related to Finding Bugs

- How to eliminate bugs?
  - Debugging
- How to prevent bugs?
  - Programming language design
  - Software development processes
- How to show absence of bugs?
  - Theorem proving
  - Model checking, program analysis

Testing Topics to Cover

- Test coverage and adequacy criteria
  - Graph, logic, input domains, syntax-based
- Test-input generation
- Test oracles
- Model-based testing
- Testing software with structural inputs
- Test automation
- Testing in your domain of interest?

Summary of the Introduction

- Eliminate bugs to save lives and money
- “Bugs” may mean faults, errors, failures
- Several approaches for detection: software testing, model checking, static analysis…
- Software testing is the most widely used approach for validation and verification
  - We will cover systematic approaches to testing, based on coverage criteria for various models
  - Testing is not (only) about revealing faults