Spring 2021

CS 424/624 Reliable Software Systems

Lecture 1: Introduction

Prof. Ryan Huang

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

One Request…

- **Turn on your video, *if you can*, this time and through the semester**
  - It is not fun to teach to black screens, esp. since this is discussion-oriented course

Instructor

- **Prof. Ryan Huang**
  - Assistant Professor joined Hopkins in 2017
  - Lead the Ordered Systems Lab (https://orderlab.io)
- **Research Area**
  - Operating Systems
  - Cloud and Mobile Computing
  - Systems Reliability
- **Office Hour:**
  - Thursdays 4-5pm (or by appointment), Zoom

Teaching Assistant

- **Yigong Hu**
  - PhD student in Order Lab
  - Published research work on energy-efficient mobile OS (ASPLOS ’19) and performance misconfiguration detection (OSDI ‘20)

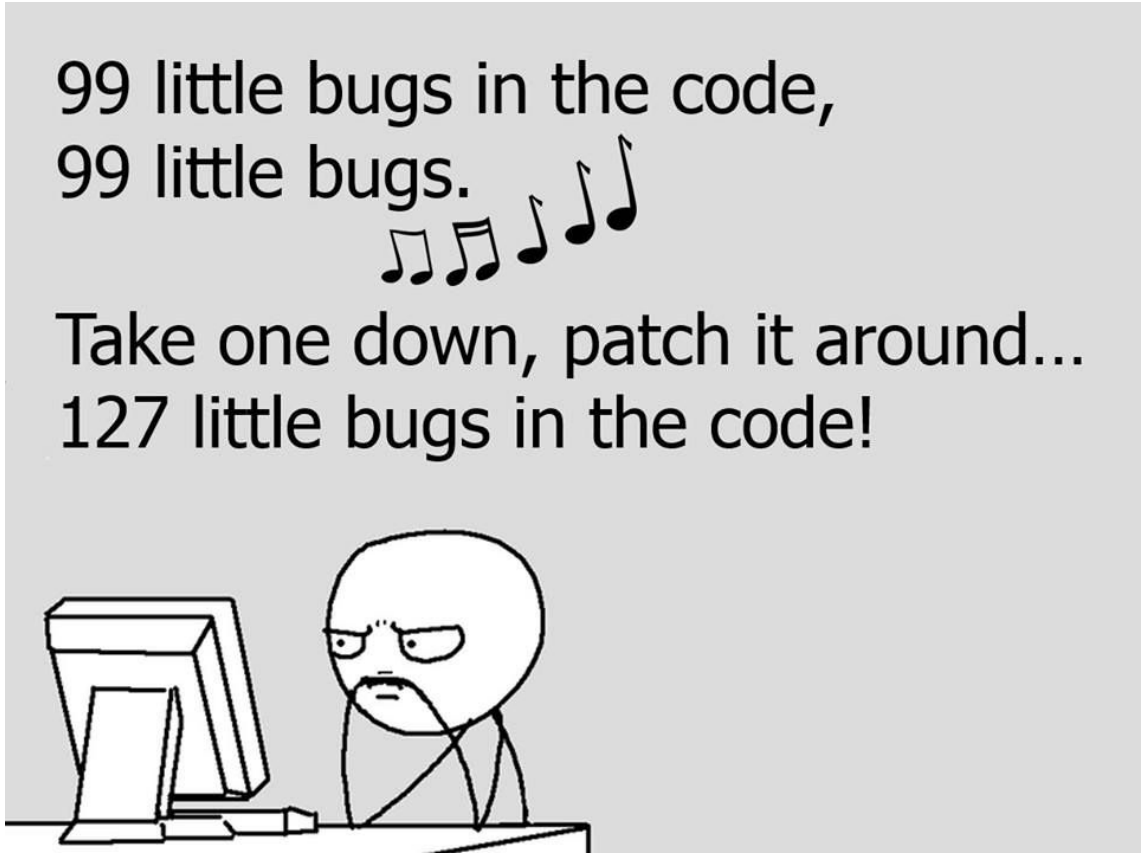
- **Office Hour:**
  - TBD

Quick Survey

- **How many undergraduate students?**
  - Masters students
  - PhD students?
  - Non-CS majors?
- **How many have taken compiler/PL/OS course?**
- **Why take this course?**

Why Take This Course?

- **Degree requirement…**
- **As both programmers and users, you wrestled with buggy software**
- **Learn methods to improve reliability**
- **Become better developers**

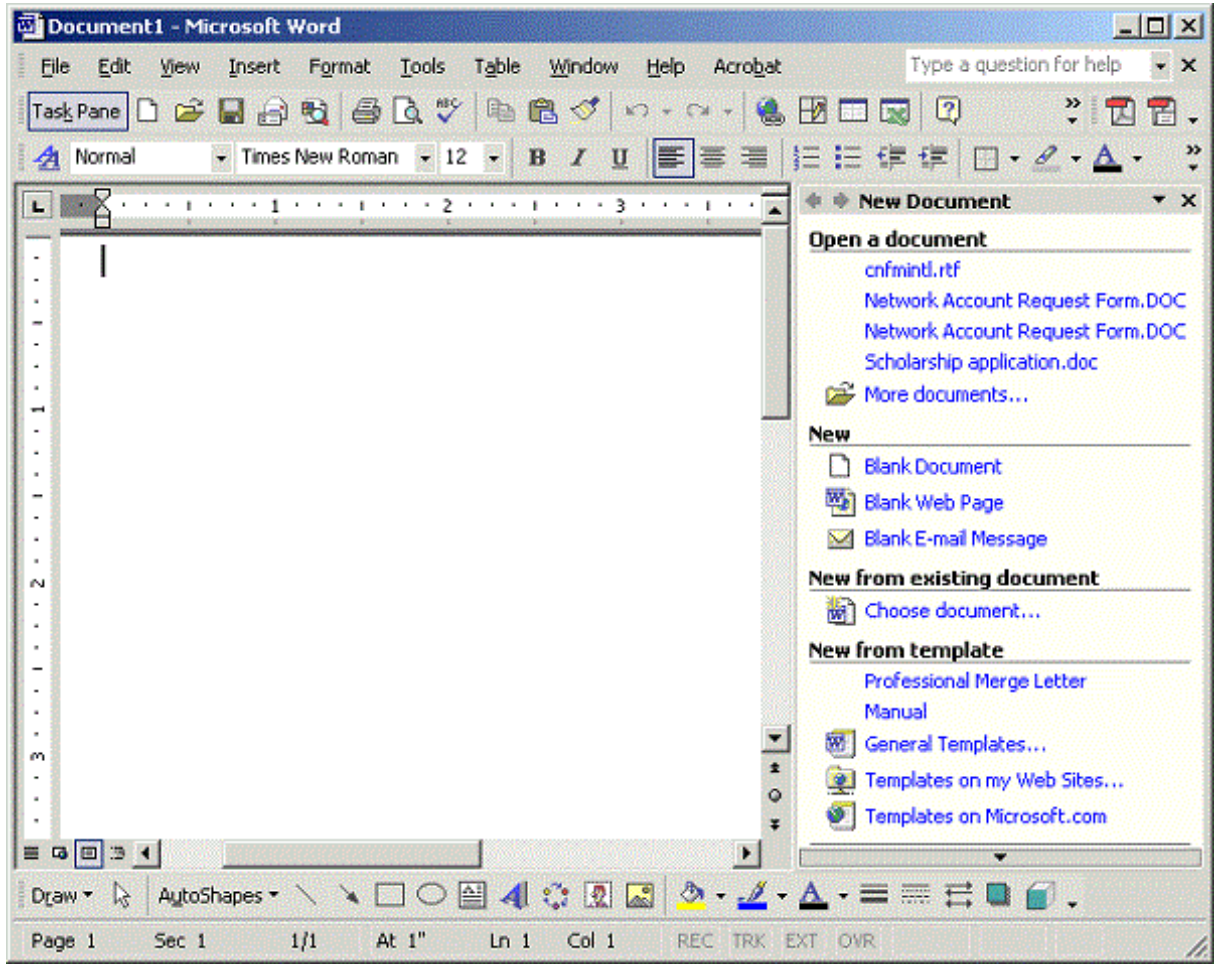


Course Characteristics

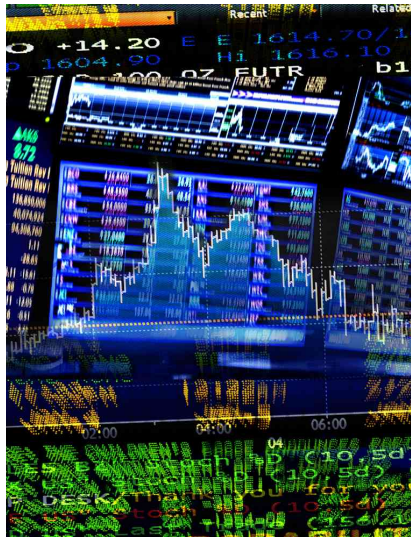
- **This is a new course!**
  - Not offered often (about every other year)
  - We'll be tweaking and improving the course as it progresses
- **Reading intensive**
  - Need to read three or four papers a week
- **Discussion-oriented**
- **Exploratory flavor**
  - Many topics we discuss are not done deals

Motivation for This New Course (1)

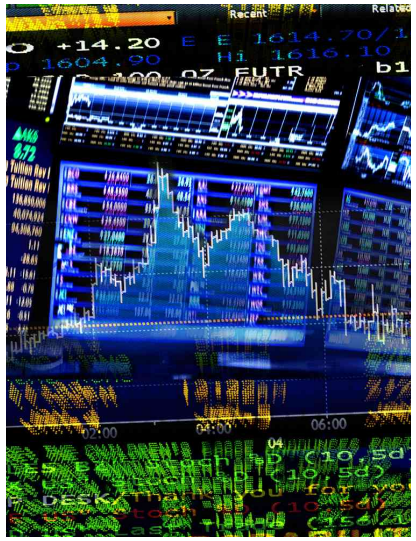
Circ. 2000



Motivation for This New Course (1)

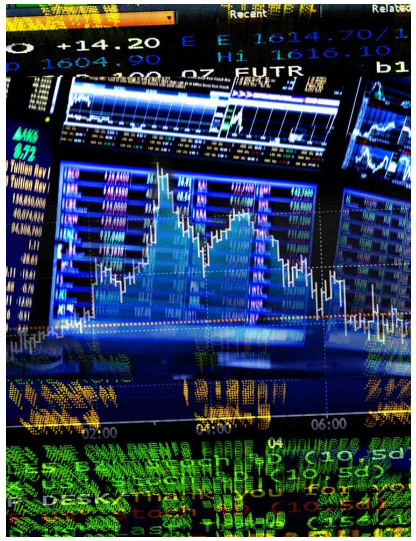


Motivation for This New Course (1)



“Software is eating the world”

Motivation for This New Course (1)

"Software is eating the world"

- **Cost of software failure is extremely high**
  - Millions of dollars of financial loss
  - Even life & death
- **Software reliability is ever more important**

Motivation for This New Course (2)

- **Industry spends significant time & resources on reliability**
  - Testing, finding bugs, debugging, patching, etc.

> "We have as many testers as we have developers. And testers spend all their time testing, and developers spend half their time testing. We're more of a testing, a quality software organization than we're a software organization."
>
> **-- Bill Gates**

Motivation for This New Course (2)

- **Industry spends significant time & resources on reliability**
  - Testing, finding bugs, debugging, patching, etc.
  - Many tech companies have dedicated teams working on it
    - e.g., Site Reliability Engineering (SRE) team

Motivation for This New Course (3)

- **Students are taught programming, but not much on reliability skills**
  - There are courses about testing, but reliability requires much more than testing

> "Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often."

- **Current ways of software dev and operation are often *ad-hoc***

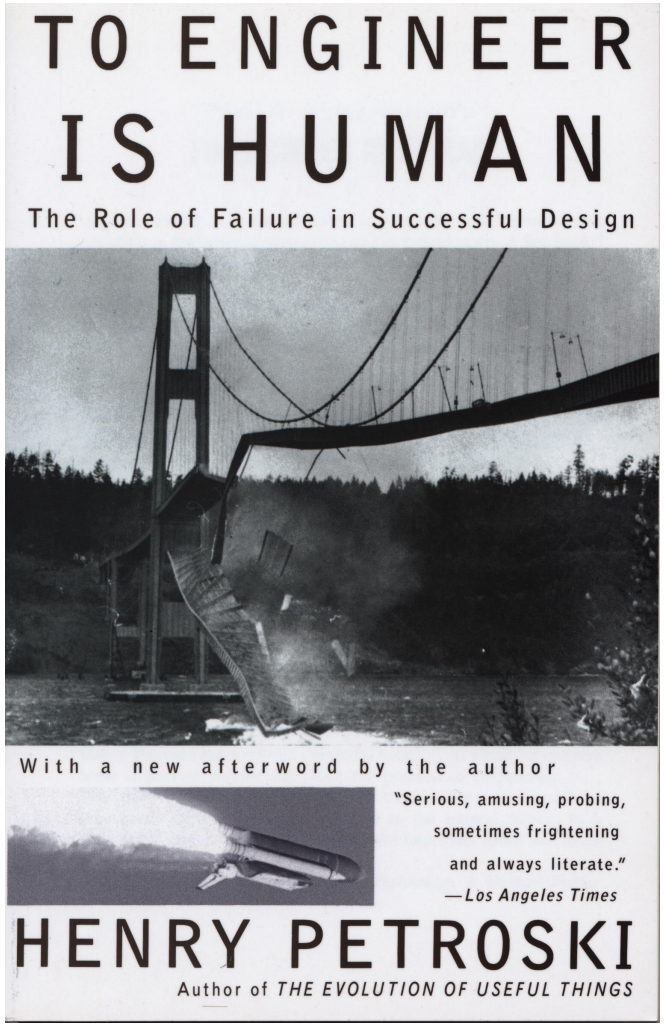

IT DEPARTMENT
HAVE YOU TRIED TURNING IT OFF AND ON AGAIN?
memegenerator.net


It doesn't work..... why?
It works..... why?

Fixing a bug in production

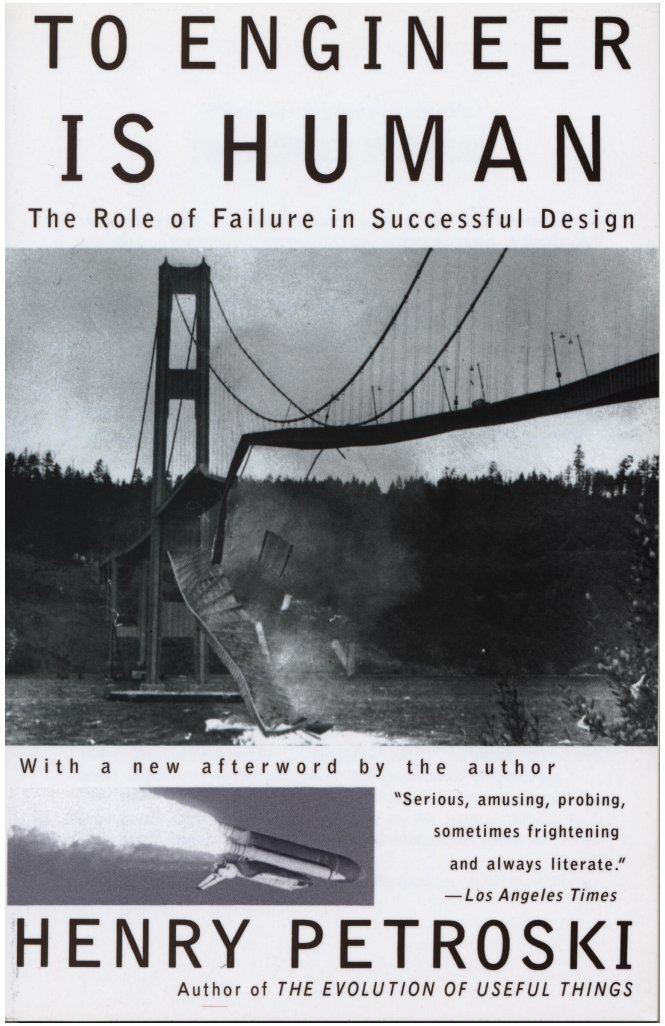
Learn from Other Disciplines

- **Safety is treated much more rigorously in other engineering disciplines (e.g., Civil Engineering)**
- **But not always the case!**
  - Arrival of railroads in 19th century
  - Not very successful: ~¼ of all iron truss railroad bridges failed between 1850s and 1890s.
  - Engineers started studying failures and learned from the failure
    - E.g., introducing concept of *margin of safety*
- **A whole Reliability Engineering sub-discipline**



Learn from Other Disciplines

- **Safety is treated much more rigorously in other engineering disciplines (e.g., Civil Engineering)**
- **But not always the case!**
  - Arrival of railroads in 19th century
  - Not very successful: ~¼ of all iron truss railroad bridges failed between 1850s and 1890s.
  - Engineers started studying failures and learned from the failure
    - E.g., introducing concept of *margin of safety*
- **A whole Reliability Engineering sub-discipline**



Course Objectives

- **Introduction and overview about this field**
  - Reliability is a deep and challenging field, impossible to cover in one course
- **Expose students to an array of systematic methods**
- **Learn the latest progress**
- **Build practical reliability tools**
- **A stretch goal: inspire you to advance the state of the art**

Course Topics

Find Bugs

- Static analysis
- Dynamic analysis
- Binary analysis
- Symbolic execution
- Fuzzing
- Misconfiguration

Formal Method

- Model checking
- Verification
- PCC

Understand Failure

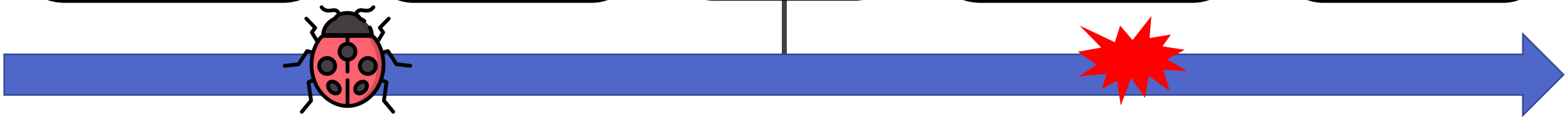
- Empirical study

Diagnosis

- Debugging
- Logging
- Taint tracking
- Record & replay

Survival

- Fault isolation
- Failure-oblivious computing
- Recovery



Scope of Focus

- **Principled methods, but emphasize on their practices**
  - Less on the theoretical aspects
    - e.g., we will discuss static analyses, but not much on their theories
- **Solutions that target large system software**
  - Rather than toy programs or small benchmarks
- **Analyses and reasoning of systems**
  - Versus development process in Software Engineering courses
- **Mainly standalone software**
  - Rather than distributed systems

Course Materials

- ***No textbook***
- **Content based on research papers**
  - Mostly from top systems conferences
    - OSDI, SOSP, EuroSys, ASPLOS, etc.
  - Others from top conferences in adjacent fields,
    - e.g., PLDI (programming language), FSE (software engineering), CCS (security)
- **A curated reading list that covers representative works**
- **Tool documentation or tutorials**

Course Logistics

Important Links (1)

- **Course Website**
  - https://www.cs.jhu.edu/~huang/cs624/spring21
  - The syllabus is actively updated, check it often!
  - Lecture slides, homework exercises, project requirements
- **Q&A forum**
  - https://piazza.com/jhu/spring2021/en601424624 (access code in email+slack)
  - Ask questions
- **Discussion channel**
  - https://jhu-cs624-sp21.slack.com
  - Paper discussions, pop quizzes, team communications

Important Links (2)

- **Gradescope**
  - http://gradescope.com
  - EntryCode posted on Slack
  - Submit paper reviews
- **Staff mailing list**
  - cs624-staff@cs.jhu.edu
  - Private administrative requests

Format

- **Each lecture typically has two papers to discuss**
  - May take different approaches or focus on different aspects on the same topic
  - Comparing and contrasting them gives you better understandings
- **I will lead the first few lectures**
- **For the remaining lectures, students will present the papers**
- **The audience should actively participate in discussions**

Paper Reading

- **All students must read the required papers *before* the class**
  - There will be pop quiz to check if you have read the papers
- **Most papers are dense and take time to read**
  - It will get better as you read more
  - Reading technical papers is a useful skill you'll practice in this course

Tips on Paper Reading

- **Two good meta papers on how to effectively paper**
  - http://cseweb.ucsd.edu/~wgg/CSE210/howtoread.html
  - "How to Read a Paper" by S. Keshav (link in course website)
- **Multi-pass method**
  - Understand "big questions" first
    - What is the motivation? Is the problem important? Why existing solution is insufficient? What is the proposed idea? What is it insight behind the idea? Etc.
  - Zoom into the "nuts and bolts"
    - How does the solution decide X? What if there is a Y? What workloads are used? Etc.
  - Evaluate the paper as a whole
- **Encourage you to form a reading group with a few classmates**
  - Discussing among the group can significantly help you understand the papers better

Paper Reviews

- **Required to write a review and submit it *by noon* for each lecture**
  - A review template is provided, but you don't have to follow the structure
  - Must capture your own understanding & critiques
    - Copying a large amount of text from the paper does not count as a review
  - Be concise
  - If there are two readings, write review for *at least* one of them
- **Submit reviews on Gradescope**

Paper Presentations

- **Each student needs to present at least twice a semester**
  - We may pair two students to present one paper if there are many students
- **Sign up presentation slots (link in Slack/Piazza)**
  - Enter your preferences within the first three weeks
  - Each slot has three candidate cells
    - You can enter your name if there is an empty cell
    - Do not overwrite a cell if it has been taken
  - I will decide the presenter
  - You can request to change preference
- **Presenters for the first eight slots get bonus credits**

Presentation Requirements (1)

- **For many of the required papers, the authors publish the slides**
  - You can use the slides as a reference, but you must make your own slides
- **Do not have to be super fancy, but must clear and well structured**
  - E.g., putting excessive texts in one slide is bad (this slide isn't a good example!)
  - Should use visualizations and illustrations to more effectively explain content
  - The slides should have a coherent logic flow

Presentation Requirements (2)

- **Informative and detailed**
  - Can *not* be vague and just describing the high-level idea
  - Describe the design and implementation in detail
    - Rule of thumb, 10-20 slides explaining the technical solution
- **Include background knowledge when necessary**
  - E.g., the solution may be using a known technique and does not explain that technique in the paper; do the homework and explain in your presentation
- **Consider doing a demo (not too long, though)**
- **Prepare some questions to trigger discussions**

Presentation Requirements (3)

- **Each student presentation will be rated by the course staff based on its clarify, informativeness, communication, etc.**
  - The rating will be counted in the overall grade
- **Some paper may be assigned with two presenters**
  - The two presenters should coordinate in making the presentation
  - The final presentation should be a coherent talk
    - Each can be in charge of different sections
  - The two presenters receive the same grade for that presentation

Homework Exercises

- **Four homework exercises for different topics**
  - Static analysis
  - Symbolic execution
  - Binary analysis & instrumentation
  - Fuzzing
- **Give you basic hands-on experience on some widely-used tools**
  - relatively lightweight, some task will be open-ended
- **Helps warm you up for your course project**

Course Project

- **Design a solution that improves s/w reliability**
  - Can be on any of the topics covered in the course
  - Topics outside the course syllabus may also work, but should check with me
- **Work in a team**
  - 2 or 3 persons each team, start looking for potential teammates now!
- **Build a non-trivial artifact and write a paper-like report**
  - Artifact should be evaluated with real-world software
- **Deliverables**
  - #1 proposal, #2 checkpoint report, #3 final report

Course Project Ideas

- **I will provide some example topics**
- **Encourage you to come up with your own ideas**
  - If you are unsure about the ideas, can schedule appointments with to discuss
- **The course staff will meet with each group along the way**
  - Check progress, give suggestions
- **The project should contain some "novelty"**
  - It is OK to start with reproducing a related paper we have discussed, but must explore something not described in the paper

Final Presentation

- **We will hold a "mini-research" conference in the end of the semester**
- **Each team will do a final presentation on their projects**
  - Structure the presentation like a technical talk
- **Expect some demo of the developed solution**

Grading

- **Paper Reviews: 10%**
  - Staff will randomly choose three of your reviews to grade
- **Presentation: 10%**
  - Graded based on clarity, informativeness, structure, etc.
- **Class Participation: 15%**
  - Discussions, answer to pop quizzes
- **Homework Exercises: 15%**
  - Allowed to choose three out of the four exercises to submit
- **Project: 50%**

Policies for Auditing Students

- **Paper presentation**
  - Students who audit the class are allowed to sign up for paper presentation
  - However, preferences will be given to students who take the class for credits
- **Paper reviews, homework exercises**
  - Auditing students are welcome to submit the paper reviews and do homework exercises
  - However, the submissions will *not* be graded
- **Course project**
  - Auditing students in general should not team up with non-auditing students in course project, unless with explicit permission
  - Auditing students can work in course projects, but receiving help and feedback from the course staff is not guaranteed.

Academic Integrity Policies

- **Individual assignments must be completely by yourself**
  - Including paper reviews and exercises
- **Do not copy reviews/code/solutions from the Internet or other classmates**
  - The consequence of cheating is very high
    - Do not have wishful thinking, it will be caught
    - Nobody likes it when it happens, especially the course staff
- **Do not publish your own reviews/code/solutions**
  - Unless the instructor explicitly allows you to do so
- **Cite any code that inspired your code**

Collaboration Policies

- **Collaboration is allowed and encouraged**
  - Discuss papers together
    - But reviews must be written independently
  - Help explain concepts to others
  - Discuss project ideas, algorithms with other groups

Late Policies

- **Four late days (96 hours) you can use throughout the semester**
- **Email the staff mailing list to request using the late hours**
- **Late submissions without or exceeding grace period will receive penalties as follows:**
  - 1 day late, 15% deduction
  - 2 days late, 30% deduction
  - 3 days late, 60% deduction
  - after 4 days, no credit.

A Few Basic Concepts

What Is Reliability?

- **Reliability is an important metric about a system's quality**
  - Other metrics: efficiency, security, usability, maintainability, etc.

Definition

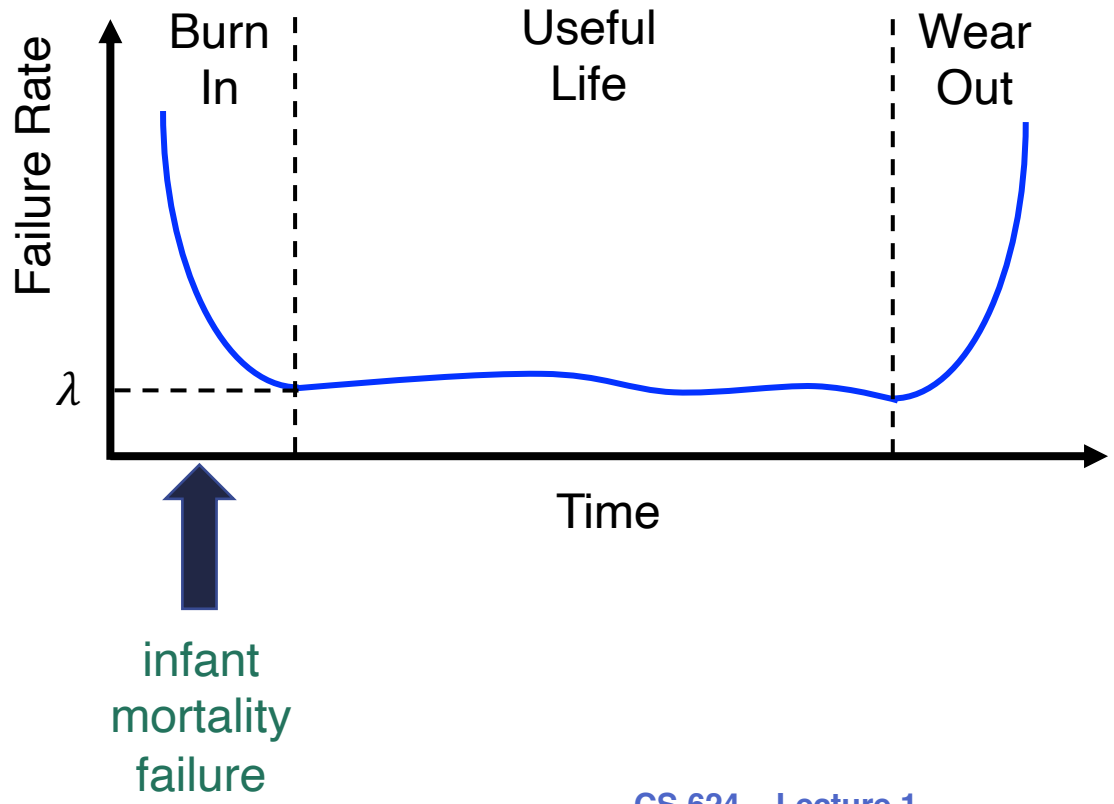
The probability that a system operates without failure in a given period of time.

$$Reliability = 1 - Probability(Failure)$$

- **Can be expressed as failure rate λ**
- **Mean Time Between Failure (MTBF, $1/\lambda$) is often reported**
  - MTBF = 2000 hours => λ = 0.0005/hour

Reliability Curve

- **The failure rate of a system usually depends on time**
  - Hard disk's failure rate in its fifth year > the rate in the first year



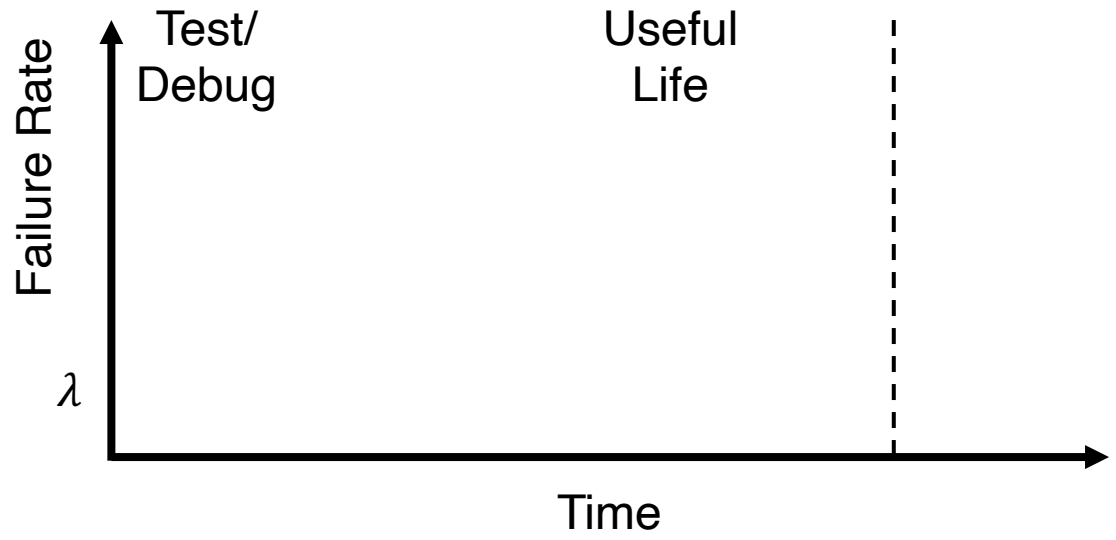
The bathtub curve

Reliability of Software vs. Hardware

- **Hardware often exhibits the bathtub curve, but software doesn't**
  - Why?
  - Hardware faults are mostly *physical faults*
  - Software faults are *design & implementation faults*
    - Hard to visualize, classify, detect, and correct
    - Related to human factors, which we often don't understand well
  - Software does not need "manufacturing"
    - Its quality does not change much once it's deployed

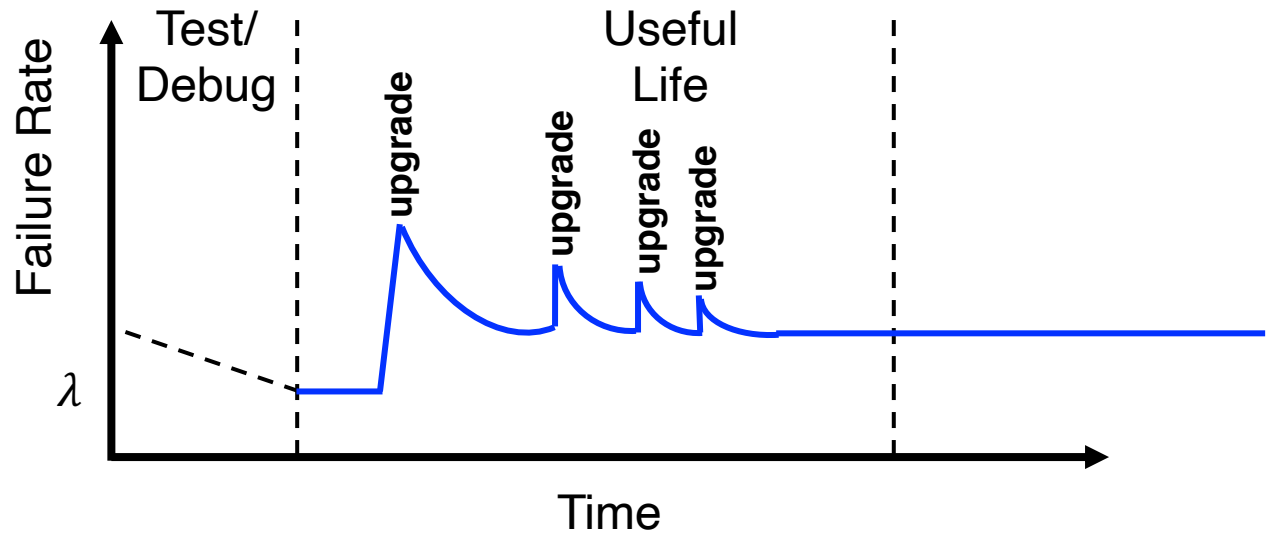
What Is The "Bathtub Curve" For Software?

- **What is the one major reason software fails?**
- **Upgrades!**



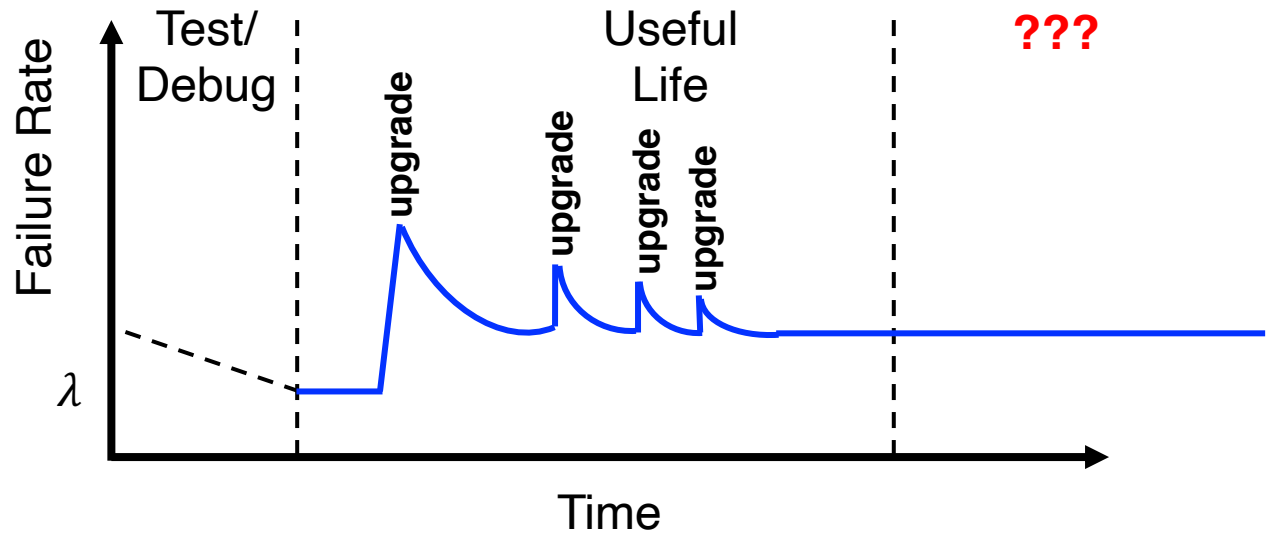
What Is The "Bathtub Curve" For Software?

- **What is the one major reason software fails?**
- **Upgrades!**



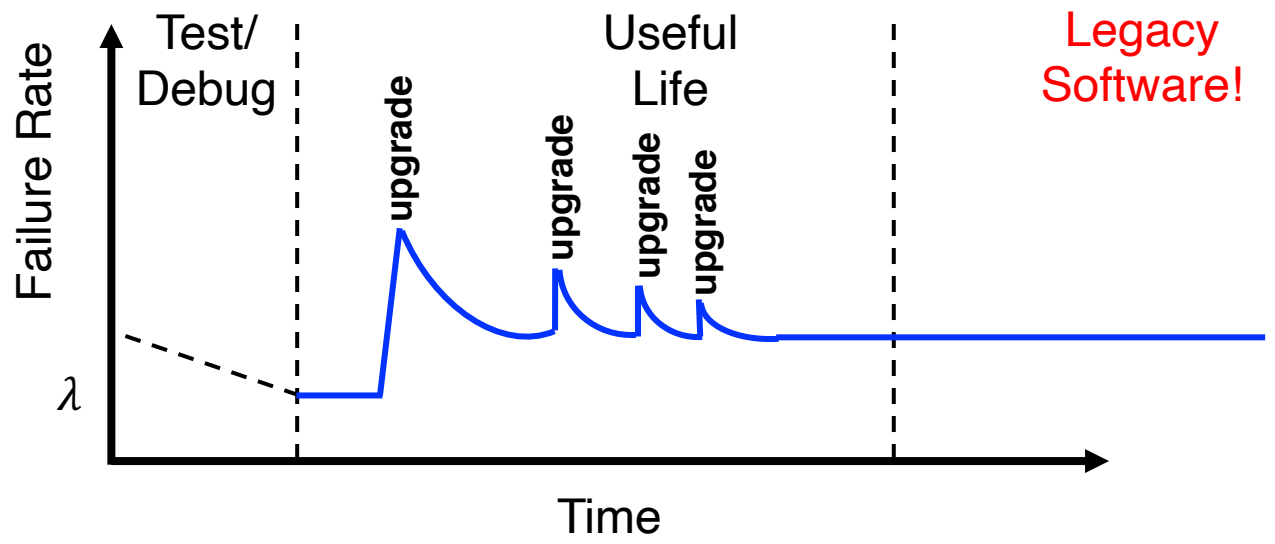
What Is The "Bathtub Curve" For Software?

- **What is the one major reason software fails?**
- **Upgrades!**

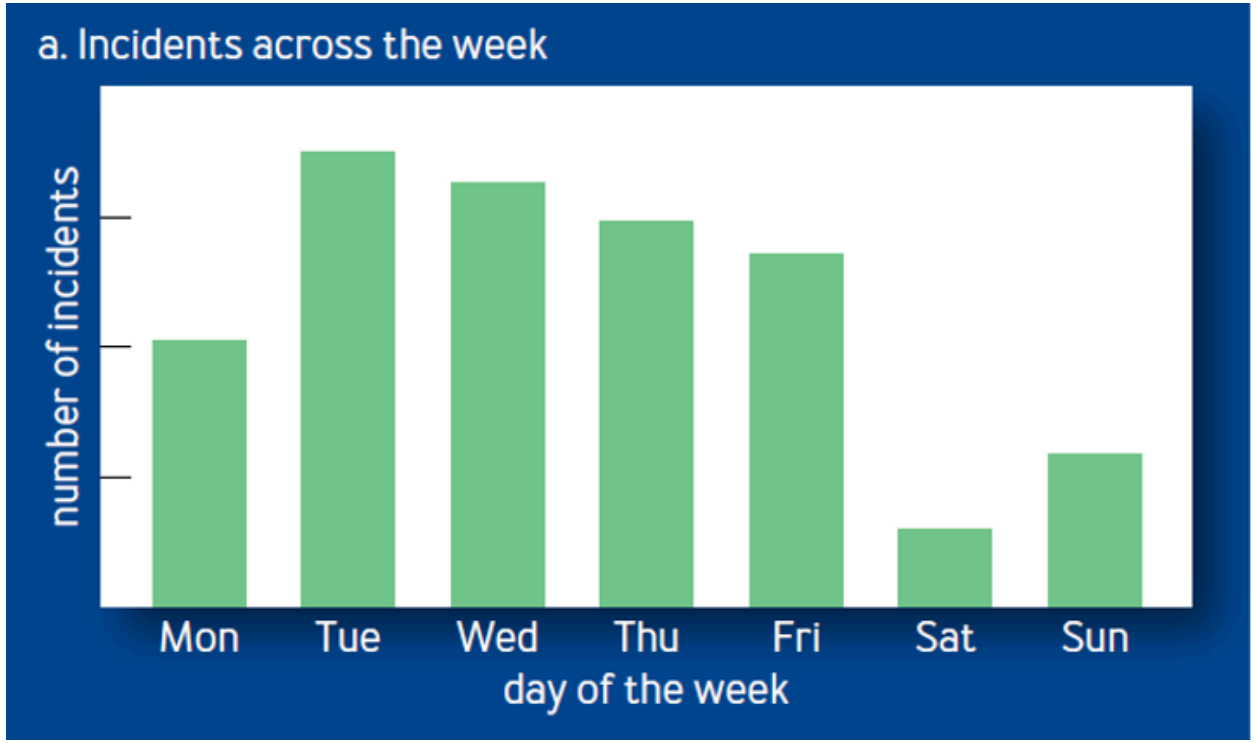


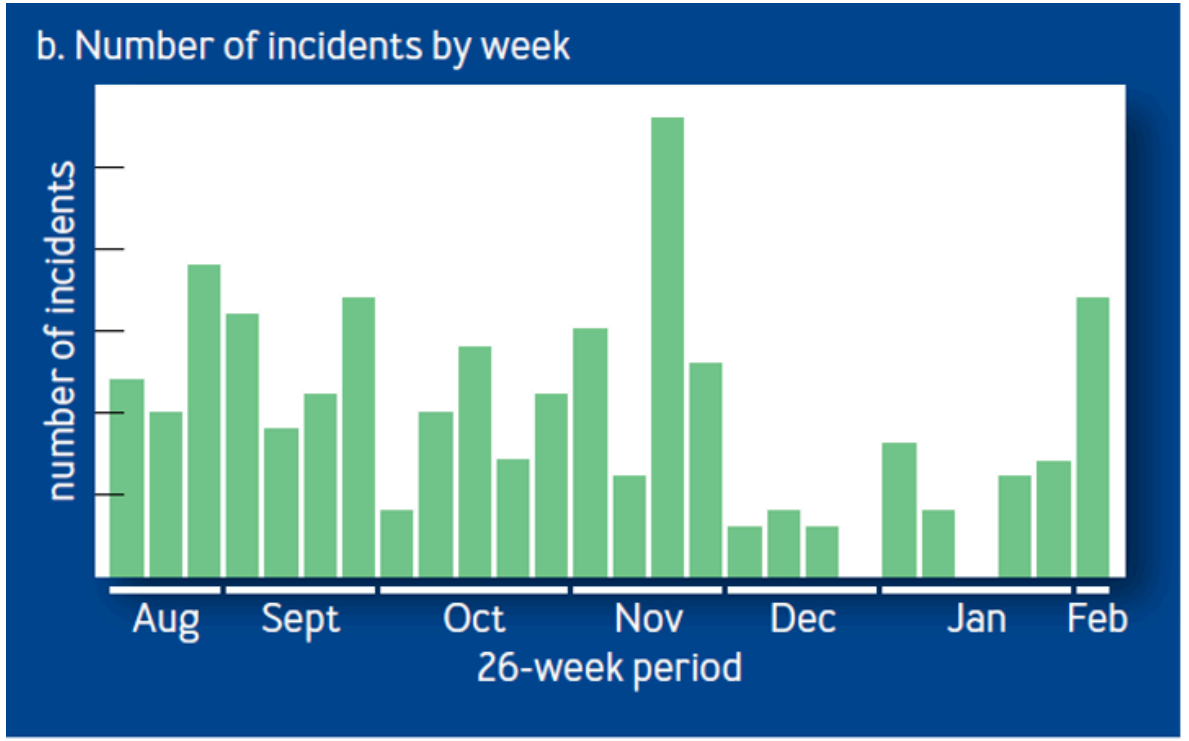
What Is The "Bathtub Curve" For Software?

- **What is the one major reason software fails?**
- **Upgrades!**



Number of Facebook System Incidents





"Fail at Scale" [ACM Queue]

Reliability vs. Availability

- **Another quality metric for software system is *Availability***
  - Often used in distributed systems, but applies to regular software as well

Definition

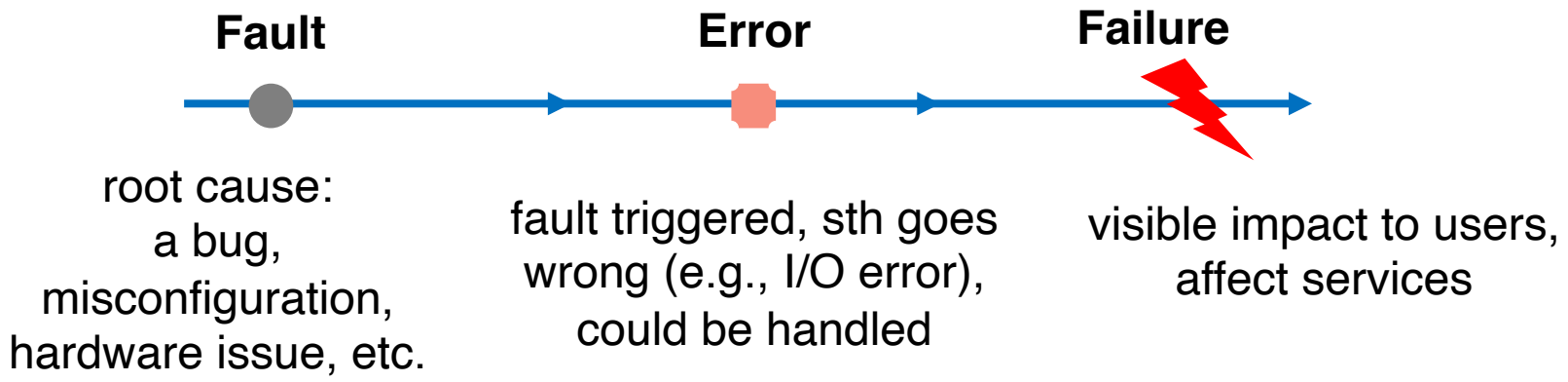
The percentage of time that the system operates satisfactorily.

$$Avalability = \frac{E[uptime]}{E[uptime] + E[downtime]}$$

- **Reliability and availability are closely related, but different**
- **We can build a system that is unreliable but highly available!**
  - Key idea behind the success of modern distributed systems like GFS
    - many unreliable commodity servers + fault-tolerance ≫ a few ultra-reliable workstations

Fault, Error, Failure

- **The three terms are often mis-used interchangeably and may mean different things to different people**
- **A classic definition**



What Is A Failure Anyway?

- **A very simple question but surprisingly not easy to answer**
  - Some are obvious, e.g.,
  - But modern software is extremely complex
    - E.g., Google doc still displays the content but no long allow users to make edits
    - E.g., you can login to Facebook, but news feeds are empty
      - Or it takes a very long time to load, e.g., 1s?, 10s? 30s? 5min?
      - Or some users can view their feeds but a few others cannot
    - E.g., Chrome is running, but slowly leaking memory
    - E.g., your Android phone is usable, but runs out of battery in 6 hours
- **One driving research question behind research in my group**

What Is A Failure Anyway?

- **A very simple question but surprisingly not easy to answer**
  - Some are obvious, e.g.,

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

What Is A Failure Anyway?

- **A very simple question but surprisingly not easy to answer**
  - Some are obvious, e.g.,
  - But modern software is extremely complex
    - E.g., Google doc still displays the content but no long allow users to make edits
    - E.g., you can login to Facebook, but news feeds are empty
      - Or it takes a very long time to load, e.g., 1s?, 10s? 30s? 5min?
      - Or some users can view their feeds but a few others cannot
    - E.g., Chrome is running, but slowly leaking memory
    - E.g., your Android phone is usable, but runs out of battery in 6 hours
- **One driving research question behind research in my group**

Common Approaches to Improve SW Reliability

- **Static analysis**
  - analyze source code of a program statically
- **Dynamic analysis**
  - analyze program behavior in its dynamic execution
- **Symbolic execution**
  - executes program with symbolic input
- **Model checking**
  - construct a model from specification or implementation, systematically enumerate the software behavior under this model, and check if some path violates a desired property
- **Formal verification**
  - prove a program satisfies a formal specification of its behavior

Interaction Time

Introduction

- **Introduce yourself**
- **Describe some unforgettable bug/issue**
  - either in your code and someone else's software

For Next Class...

- Browse the course web
- Sign up the paper presentation
- Read assigned papers and write reviews
- Search for teammates