

# Foundations of Software Engineering

Part 1: Overview

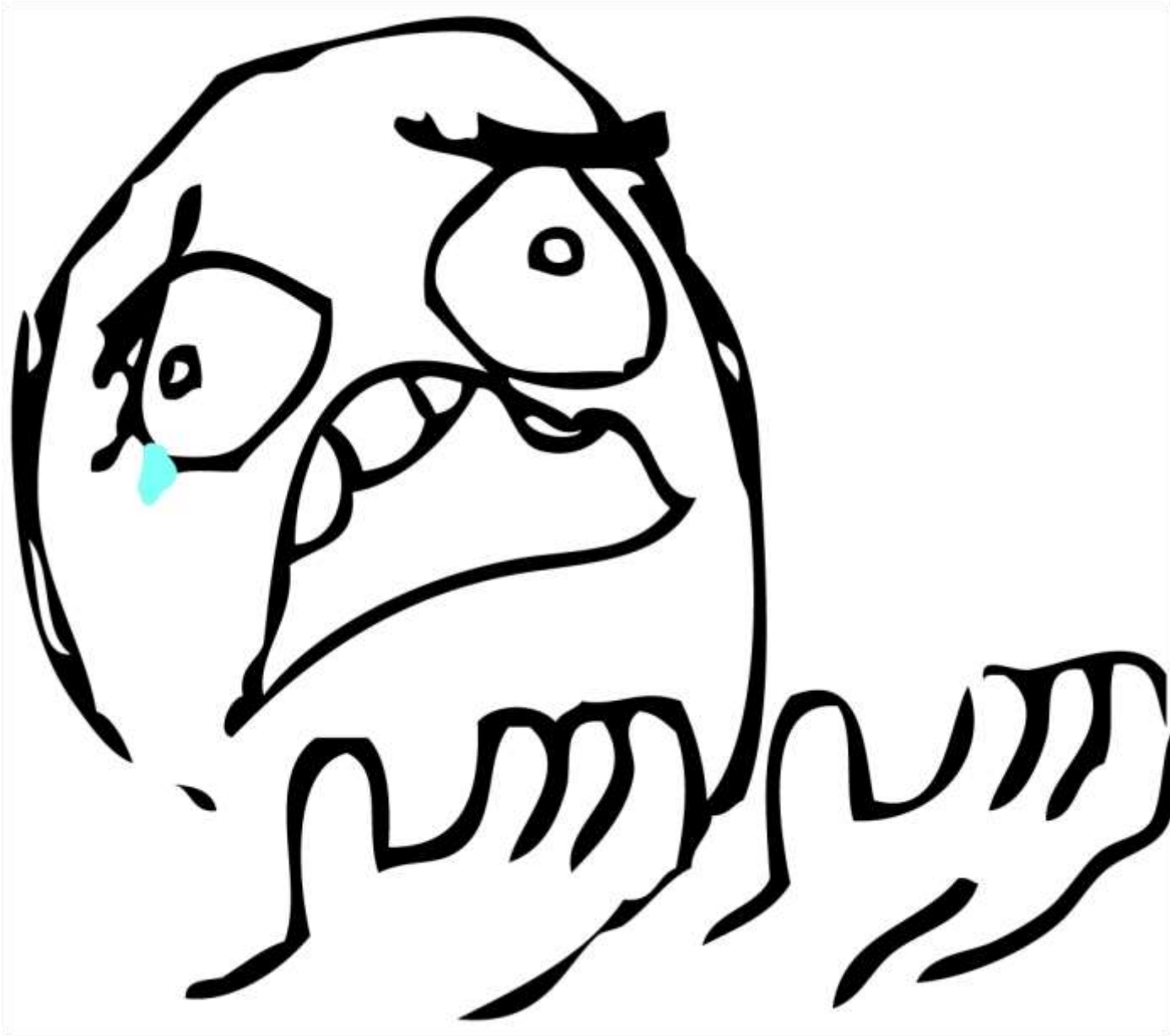
Claire Le Goues, Christian Kästner

# Learning Goals

- Broad scope of software engineering
- Importance of nontechnical issues
- Overview key challenges
  
- Syllabus, introduction and team forming







“...participants who multitasked on a laptop during a lecture scored lower on a test compared to those who did not multitask, and participants who were in direct view of a multitasking peer scored lower on a test compared to those who were not. The results demonstrate that *multitasking on a laptop poses a significant distraction to both users and fellow students and can be detrimental to comprehension of lecture content.*”

Faria Sana, Tina Weston, and Nicholas J. Cepeda. 2013. Laptop multitasking hinders classroom learning for both users and nearby peers. *Comput. Educ.* 62 (March 2013), 24-31.

“...students who took notes on laptops performed worse on conceptual questions than students who took notes longhand. We show that whereas taking more notes can be beneficial, laptop note takers’ tendency to transcribe lectures verbatim rather than processing information and reframing it in their own words is detrimental to learning.”

Psychol Sci. 2014 Jun;25(6):Epub 2014 Apr 23.




















The pen is mightier than the keyboard: advantages of longhand over laptop note taking. Mueller PA1, Oppenheimer DM2.

**Software is Everywhere**  
**Software is Important**  
(duh)



**2016** [\[ edit \]](#)

This list is up to date as of June 30, 2016. Indicated changes in market value are relative to the previous quarter.

Rank		First quarter <sup>[8]</sup>		Second quarter	Third quarter	Fourth quarter
1		<a href="#">Apple Inc</a> ▲596,988.7		<a href="#">Apple Inc</a> ▼515,590.0		
2		<a href="#">Alphabet</a> ▼514,923.5		<a href="#">Alphabet</a> ▼475,160.0		
3		<a href="#">Microsoft</a> ▼434,130.1		<a href="#">Microsoft</a> ▼399,710.0		
4		<a href="#">Amazon Inc.</a> ▲356,119.4		<a href="#">Exxon Mobil</a> ▲388,710.0		
5		<a href="#">Berkshire Hathaway</a> ▲349,813.4		<a href="#">Berkshire Hathaway</a> ▲356,810.0		
6		<a href="#">Exxon Mobil</a> ▲346,616.5		<a href="#">Amazon Inc.</a> ▼337,650.0		
7		<a href="#">Facebook</a> ▲326,357.8		<a href="#">Johnson &amp; Johnson</a> ▲333,650.0		
8		<a href="#">Johnson &amp; Johnson</a> ▲300,604.4		<a href="#">Facebook</a> ▲226,880.0		
9		<a href="#">General Electric</a> ▼285,545.7		<a href="#">General Electric</a> ▼288,488.0		
10		<a href="#">Wells Fargo</a> ▼246,035.0		<a href="#">Wells Fargo</a> ▼238,950.0		

# Gov't example: Software is integral to DoD systems.

**Table I**

**System Functionality Requiring Software (source: *PM Magazine*)**

Weapon system	Year	Functions performed in software (%)
F-4	1960	8
A-7	1964	10
F-111	1970	20
F-15	1975	35
F-16	1982	45
B-2	1990	65
F-22	2000	80

Quoting an Air Force lieutenant general, “The only thing you can do with an F- 22 that does not require software is take a picture of it.”

Crouching Dragon, Hidden Software: Software in Dod Weapon Systems (Ferguson, IEEE Software, 2001)

## Toyota Case: Single Bit Flip That Killed

Junko Yoshida

10/25/2013 03:35 PM EDT

During the trial, embedded systems experts who reviewed Toyota's electronic throttle source code testified that they found Toyota's source code defective, and that it contains bugs -- including **bugs that can cause unintended acceleration.**

"We did a few things that NASA apparently did not have time to do," Barr said. For one thing, by looking within the real-time operating system, the experts identified "unprotected critical variables." They obtained and reviewed the source code for the "sub-CPU," and they **"uncovered gaps and defects in the throttle fail safes."**

The experts demonstrated that **"the defects we found were linked to unintended acceleration through vehicle testing,"** Barr said. "We also obtained and reviewed the source code for the **black box and found that it can record false information** about the driver's actions in the final seconds before a crash."

**Stack overflow and software bugs led to memory corruption,** he said. And it turns out that the crux of the issue was these memory corruptions, which acted "like ricocheting bullets."

Barr also said more than half the dozens of **tasks' deaths** studied by the experts in their experiments **"were not detected by any fail safe."**

## Bookout Trial Reporting

[http://www.eetimes.com/document.asp?doc\\_id=1319903&page\\_number=1](http://www.eetimes.com/document.asp?doc_id=1319903&page_number=1)  
(excerpts)

**"Task X death  
in combination  
with other task  
deaths"**



003/45/7844



ISAT GeoStar 45  
23:15 EST 14 Aug. 2003

# Healthcare.gov: Government IT Project Failure at its Finest

Posted: 10/18/2013 6:33 pm



Read more > [Project Management](#), [Government](#), [Healthcare](#), [IT Projects](#), [Open Source](#), [Business News](#)

3	6	0	0	7
Share	Tweet	Linked in	Email	Comment

GET BUSINESS NEWSLETTERS:

SUBSCRIBE

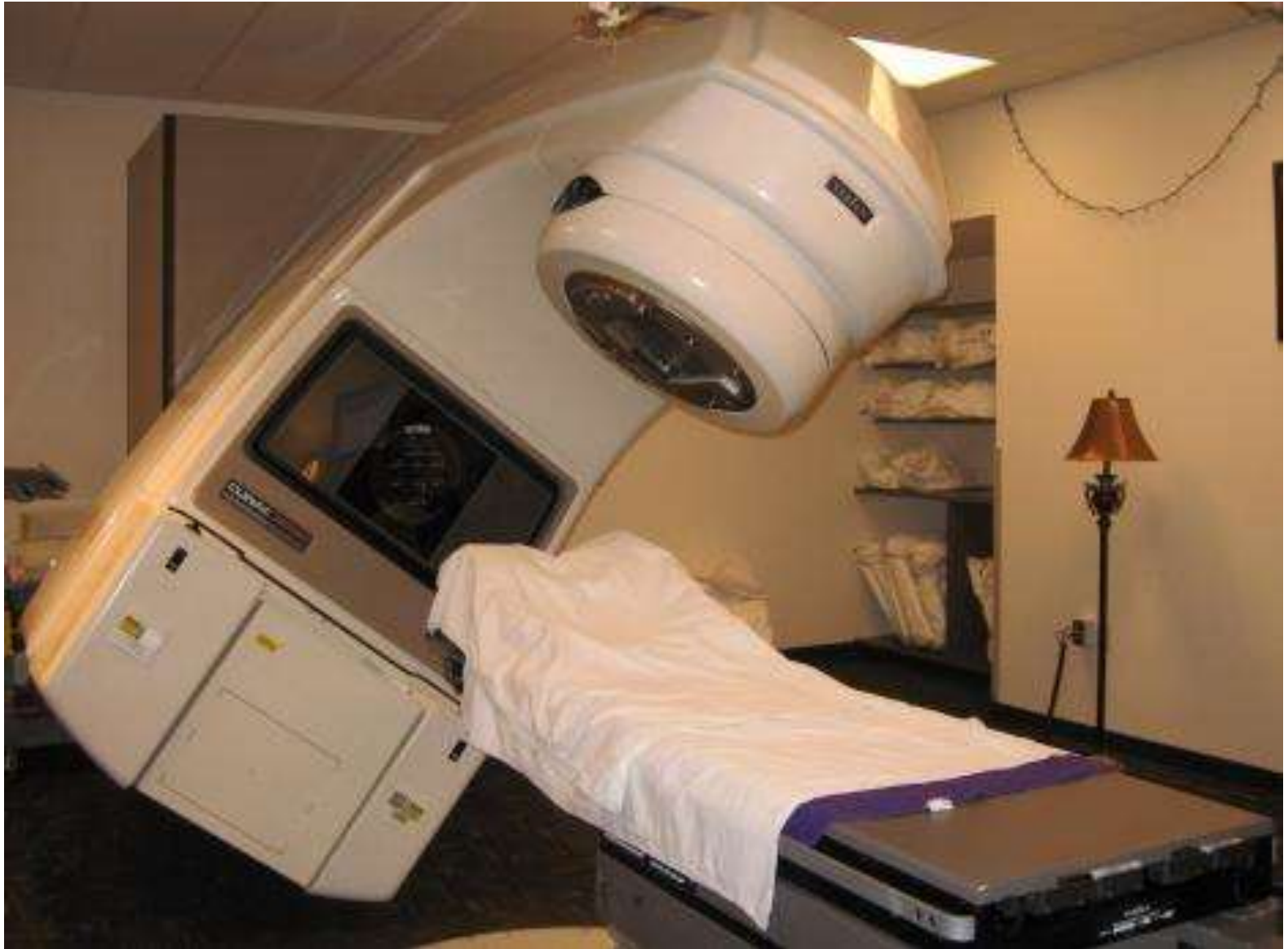
The [BusinessWeek](#) article on the [Healthcare.gov](#) failure is nothing if not instructive. From the piece:

Healthcare.gov isn't just a website; it's more like a platform for building health-care marketplaces. Visiting the site is like visiting a restaurant. You sit in the dining room, read the menu, and tell the waiter what you want, and off he goes to the kitchen with your order. The dining room is the front end, with all the buttons to click and forms to fill out. The kitchen is the back end, with all the databases and services. The contractor most responsible for the back end is CGI Federal. Apparently it's this company's part of the system that's burning up under the load of thousands of simultaneous users.

The restaurant analogy is a good one. Projects with scopes like these fail for all sorts of reasons. *Why New Systems Fail* details a bunch of culprits, most of which are people-related.

As I read the article, a few other things jumped out at me, as they virtually guarantee failure:

- The sheer number of vendors involved
- The unwillingness of key parties involved with the back-end to embrace transparency



# Failed Software Projects

- SAGE (Semi-Automatic Ground Environment); started 1951, almost obsolete when finished in 1963; higher costs than Manhattan project
- FBI Virtual Case File stopped in 2005 after 3 years and 170 M\$
- London stock exchange stopped Taurus project 1993 after 11 years when 13200% over budget



**“But we’re CMU students and we are really, really smart!”**

What is engineering? And how is it different from hacking/programming?


## ***Software Engineering?***

# 1968 NATO Conference on Software Engineering

- Provocative Title
- Call for Action
- “Software crisis”





- 
- A photograph of a circle of chairs arranged in a ring on a gray floor against a gray background. There are seven white chairs and one orange chair in the center of the ring. The chairs are modern, minimalist in design with thin metal legs.
- Name
  - Interesting software development experience?
  - Specific topic of interest?

# Case Study 1: PeopleCars

# Case Study 1: PeopleCars

- Scenario and question from last year's final
- Read scenario and question
- Discuss answers with your neighbors
  
- Keep answers until last lecture

# Syllabus and course mechanics



# Course Themes

- Software engineering as a human process
- Process
- Requirements
- Measurement
- Quality, incl. Security
- Time and team management
- Economics
- Strategic thinking about software

# Prerequisite: 214

- Assumes working knowledge of Java
- Assumes knowledge of object-oriented design (UML, design patterns, ...)
- Assumes experience with small projects (e.g., Scrabble)
  
- 313 largely focused on human issues and quality beyond functional correctness
- 313 focused on larger scale

# Active Lecture

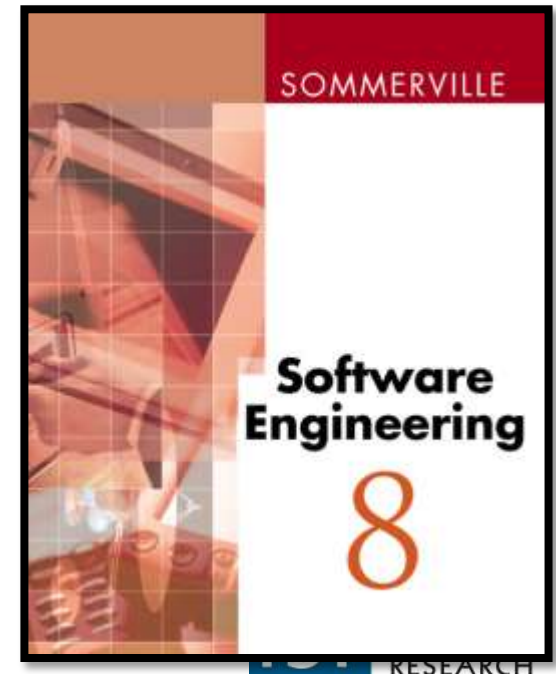
- Case study driven
- Discussion highly encouraged
- Contribute own experience
- Regular active in-class exercises
- In-class presentation
- Discussions over definitions

# Readings and Quizzes

- Reading assignments for most lectures
  - Preparing in-class discussions
  - Background material, case descriptions, possibly also podcast, video, wikipedia
  - Complement with own research
- Short and easy online quizzes on readings, due by start of lecture

# Textbook

- No single textbook
- Assigned readings from different sources
  - Book chapters (library)
  - News articles
  - Lecture notes
- Recommended supplementary reading: Sommerville, Software Engineering, edition 7 or 8
  - Aim for a used edition for <10\$



# Gaining Experience

- Case study analyses
- Team assignments
- Open source engagement
  
- No “survivor”-style projects –  
wait till 15-413

# Evaluation

- Assignments (50 %)
  - Regular homeworks, mostly in teams with individual component
  - Open source engagement
- Midterm (15 %)
- Final (20 %)
- Participation in lecture and recitation (10 %)
- Quizzes on reading assignments (5%)
- Read the learning goals!

# Participation

- Participation is important
  - Participation in in-class discussions
  - Active participation in recitations
  - Both quality and quantity are important, quality more than quantity
- Participation != Attendance



# Recitations

- Practical tasks, preparation for homework, extra material, discussions
- Please bring laptop, have github account
- This week: Good practices for collaborating with Git

# Assignments

- Planning and developing a nontrivial software project as a team
- Develop and execute a test plan
- Solicit requirements
- Implement an own static and dynamic analysis, extending FindBugs
- Contributing to an open source project of your choice

# Team Assignments

- Mirror realistic setting
- Assigned teams throughout the semester
  - Fill in team building survey before next lecture
- Peer evaluation and conflict resolution process as needed
- Most team assignments have individual components

# Late day policy

- No late days
  - (simply doesn't work with team assignments)
- Accommodations in case of health issues, travel for interviews, ... on case by case base
  - Inform us at least 2 days before deadline

# Academic Honesty

- 214-like Collaboration Policy
  - University Policy on Academic Integrity
- +
- In group work, be honest about contribution of group members; do not cover for others

# Course Infrastructure

- Course website
  - schedule, slides, syllabus, office hours
- Canvas (beta testing for CMU)
  - homeworks, grades, discussions
- Git/Github for coding and collaboration
  
- Office hours on web page, open door policy  
[staff-15313@lists.cmu.edu](mailto:staff-15313@lists.cmu.edu),

# Two Surveys



# Survey Goals

- Forming balanced groups
- Shaping the courses based on
  - your background knowledge
  - your interests
- Identifying experience in the room



# Reading Assignment Sep 1

- Sommerville Software Engineering, ed 7 or 8 – Chapter “Project Management”
- Complete quick quiz on Canvas before class

# Software *Engineering*?

# Envy of Engineers

- Producing a car/bridge
  - Estimable costs and risks
  - Expected results
  - High quality
- Separation between plan and production
- Simulation before construction
- Quality assurance through measurement
- Potential for automation



# Software Engineering?

*„The Establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.”*

*[Bauer 1975, S. 524]*

# Dangerous Analogy

- Software = Design = Plan
- Programming is design, not production
  - Production (copying/loading a program) is automated
  - Simulation not necessary
- Agile technologies possible
- Quality measurement?

