The Software Problem

- Scale
- Users as bugs
- Users as evolution
- Evolution yields complexity and bugs
- SE matters

“In 2000, total sales of software reached approximately $180 billion, supported by a large workforce encompassing 697,000 software engineers and 585,000 computer programmers.”
Scale

U.S. MANNED SPACEFLIGHT PROGRAM

MILLIONS OF INSTRUCTIONS

100
75
50
25
0

SPACE STATION
SPACE SHUTTLE
APOLLO
GEMINI
MERCURY

SOURCE: Barry W. Boehm

PROJECT SIZE (FUNCTION POINTS)

10,240
5,120
2,560
1,280
640
320
160
80
40
20
10
0
0 10 20 30 40 50 60

MONTHS

PLANNED
ACTUAL

U.S. AVERAGE PROJECT SCHEDULE

SOURCE: Software Productivity Research

PROJECT SIZE (FUNCTION POINTS)

10,240
5,120
2,560
1,280
640
320
160
80
40
20
10
0
0 10 20 30 40 50

PERCENT

U.S. AVERAGE CANCELLATION PROBABILITY

SOURCE: Software Productivity Research

"Studies have found that reworking defective requirements, design, and code typically consumes 40 to 50 percent of the total cost of software development (Jones 1986). As a rule of thumb, every hour you spend on defect prevention will reduce your repair time from three to ten hours. In the worst case, reworking a software requirements problem once the software is in operation typically costs 50 to 200 times what it would take to rework the problem in the requirements stage (Boehm and Papaccio 1988). It’s easy to understand why. A 1-sentence requirement can expand into 5 pages of design diagrams, then into 500 lines of code, 15 pages of user documentation, and a few dozen test cases. It’s cheaper to correct an error in that 1-sentence requirement at requirements time than it is after design, code, user documentation, and test cases have been written to it.”

Steve McConnell, Software Quality at Top Speed, Software Development, August 1996
Users as Evolution

EMERGENCY PROGRAM FIXES 12.4
ROUTINE DEBUGGING 9.3
ACCOM CHANGES TO INPUT DATA, FILES 17.4
ACCOM CHANGES TO HARDWARE, OS 6.2
ENHANCEMENTS FOR USERS 41.8
IMPROVE DOCUMENTATION 5.5
IMPROVE CODE EFFICIENCY 4.0
OTHER 3.4

LIENTZ -- SWANSON
DPMA SURVEY, 1978
487 INSTALLATIONS

PERCENT OF SOFTWARE MAINTENANCE EFFORT

Boehm, SE as it is, ICSE'79
Evolution yields Complexity/Bugs

Figure 4 Serial and average growth trends of a particular attribute

Figure 7 Complexity growth during the interval prior to each release
RAYTHEON HAS SAVED $17.2 million in software costs since 1988, when its equipment division began using rigorous development processes that doubled its programmers' productivity and helped them to avoid making expensive mistakes.
What’s Modularity Got to Do with It?

• An old idea
• Sometimes mocked
• Sometimes modified

Do we rewrite the rules, or just reinterpret them?
A Brief History of Modularity

In various forms is ancient, where components can be reliably composed & interchanged.

Software is unique for its highly malleable form and support for customizable abstraction

The key issue isn’t modularity, it’s how to modularize: what should the components be? How should they interface?
Benefits of Software Modularity

Maintainability
Extensibility/Incrementality
Testability
Debugability
Reusability
Division of labor / parallel work / specialization
Interchangeability
Understandability/Comprehensibility
It's all about “hands on”

THE COURSE
Organization of the Course

Alternating readings and labs (50% each)
Readings on Thursday
Lab on Tuesday
  • Do “lab” work at home in groups (of 4)
    – Upload via Google Forms; each lab a branch in Git
  • Present work in class for critique
  • Idea is to learn from each other, hands on

Install Android Studio and app quickly
  • TA Sander Valstar will send instructions via Piazza

Need to find/form a group quickly, too

We will communicate over Piazza:
  • https://piazza.com/ucsd/spring2019/cs218
Code **refactoring** is the process of restructuring existing computer code—changing the factoring—without changing its external behavior. Refactoring improves nonfunctional attributes of the software. Advantages include improved code readability and reduced complexity; these can improve source-code maintainability and create a more expressive internal architecture or object model to improve extensibility. Typically, refactoring applies a series of standardised basic *micro-refactorings*, each of which is (usually) a tiny change in a computer program's source code that either preserves the behaviour of the software, or at least does not modify its conformance to functional requirements. -- *Wikipedia*
So Many Kinds

Rename

Abstraction
  extract variable/temp
  extract method
...

Class hierarchy manipulations
  push up / pull down method
...

Convert branching to polymorphic dispatch
...
Leverage Resources

Refactoring.com

Wikipedia: Refactoring

Youtube videos, etc.

Android Studio has built-in refactoring operations but limited to simpler “microrefactorings”
try them out!

Fowler’s book is decent:
  *Refactoring: Improving the Design of Existing Code*
Goals of the Course

Deep understanding of software modularity
Exposure to foundational literature
Improve critical reading of literature
Improve technical discussion skills
Take ideas and skills into your own practice
Rest of Today

Structure of course
Grading
How to read and discuss papers
Project
Questions (at any time)
The Computer for the 21st Century

Max A. Weiser
Palo Alto Research Center, Menlo Park, CA, USA

SPECIALIZED ELEMENTS OF HARDWARE AND SOFTWARE, CONTAINED ON SILICON, WIRE, AND RESISTIVE NANO-METAL-COMPUTER-CIRCUITS.

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. Consider writing, perhaps the first information technology. The ability to represent spoken language symbolically for longer-term storage and transmission from the limits of individual memory. Today this technology is ubiquitous in industrialized countries. Not only in books, newspapers, and television, but in our own bodies. In our speech, handwriting, handwriting, music, and art. Writing is a form of communication that is both personal and public. It is a way for individuals to express their thoughts and ideas, and it is a way for communities to share information and knowledge. Writing is a powerful tool for learning, for thinking, and for creativity. It is a medium that has allowed us to record and transmit ideas and information across generations and cultures. Writing is a fundamental part of human culture, and it is a key component of our ability to learn and to understand the world around us.

However, although ubiquitous computers may seem to be everywhere, there are still many aspects of the way we interact with them that we take for granted. For example, we may not be aware of the effort that goes into creating the software that powers our devices, or the complex algorithms that make it possible for us to search the internet, send emails, or make purchases online.

Perhaps most strikingly, we do not see the computer as an object. This is not because we do not recognize its importance, but because it has become so integrated into our daily lives. The computer is a tool that we use to access information, to communicate with others, and to perform a wide range of tasks.

This is a fundamental shift in our relationship with technology. We have moved from seeing the computer as a tool that we use to see the world as a tool that we interact with. This change has been driven by a number of factors, including the increasing availability of powerful and affordable computing devices, the development of user-friendly software, and the growing awareness of the importance of human-computer interaction.

As we continue to develop new technologies, it is important that we consider their impact on society, and that we design systems that are not only effective, but also ethical and empowering. We should strive to create technology that is accessible to all, and that respects the dignity and autonomy of every individual. Only then can we truly say that we have created a world where technology is a tool for humanity, and not a master over it.
In-Class Discussion – Thursdays

“Socratic Circles” round-table discussion
More dynamic, less controlled, more open-ended
“Peer learning”
Lab + Readings

THE COMING WEEK
Readings - Wednesday


ACM papers always available for download from UCSD or through the UCSD VPN; other papers available on Piazza

Be prepared to discuss, not just listen.

Also:
- Web site is subject to change; I'll announce any late-breaking changes.