CS1530 - LECTURE 2

THE SOFTWARE DEVELOPMENT LIFECYCLE AND ITS DISCONTENTS

THE TAO OF PROGRAMMING

A manager went to the Master Programmer and showed him the requirements document for a new application. The manager asked the Master: "How long will it take to design this system if I assign five programmers to it?"

"It will take one year," said the Master promptly.

"But we need this system immediately or even sooner! How long will it take if I assign ten programmers to it?"

The Master Programmer frowned. "In that case, it will take two years."

"And what if I assign a hundred programmers to it?"

The Master Programmer shrugged. "Then the design will never be completed," he said.

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

OTHER POSSIBLE ADDITIONS TO THE SDLC

1. External Acceptance or Evaluations
2. Regulatory Checks
3. Deployment / DevOps
4. External integration
5. EOL Preparation
6. Customer Support

NOTE THAT THIS IS THE “CLASSIC” VIEW OF SOFTWARE DEVELOPMENT

- Consider it a “Platonic ideal”
- Very few modern software development processes will look exactly like this, but the same kinds of events must take place

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

ANALYSIS

- Determining the needs of the customer
- Thinking about possible technologies to use, possible trade-offs, limitations, etc.
- Determining team size and needs

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

REQUIREMENTS

- Eliciting and setting down requirements
- Determining plans for verification and validation (ensuring that you have created the software right, and that you have created the right software)
- Eliminating ambiguity and contradiction in requirements

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

DESIGN

- Laying out structure of software system
- Determining API, IPC, UI, and other external interfaces
- Plan architectural paradigms to be used, code style, and other rules for development and further stages
  - Tabs vs spaces, anyone?

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

DEVELOPMENT

- Writing the code
- An important part of the process, but not all of it!

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

INTEGRATION AND TEST

- Integrate different software subsystems and code
- Integrate with external software and interfaces
- Perform testing to ensure that software system works "correctly" (sometimes determining the definition of "correct" is the most difficult part)

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

RELEASE

- Releasing to customers and users
- May take place…
  - once (e.g. some embedded software)
  - multiple times (e.g. an operating system)
  - almost constantly (e.g. SaaS or other online apps)

THE SOFTWARE DEVELOPMENT LIFECYCLE

-> Analysis

-> Requirements

-> Design

-> Development

-> Integration and Test

-> Release

-> Maintenance

MAINTENANCE

- Prioritization and fixing of defects
- Addition of features
- Modifications to keep up with ever-changing interfaces of other systems

WHO'S INVOLVED?

1. Systems Engineers
2. Project Managers
3. Product Managers
4. Quality Assurance
5. UI/UX Designers
6. System Administrators
7. Customer Support
8. Database administrators
9. Users and Customers
10. Software Developers, too, I guess…

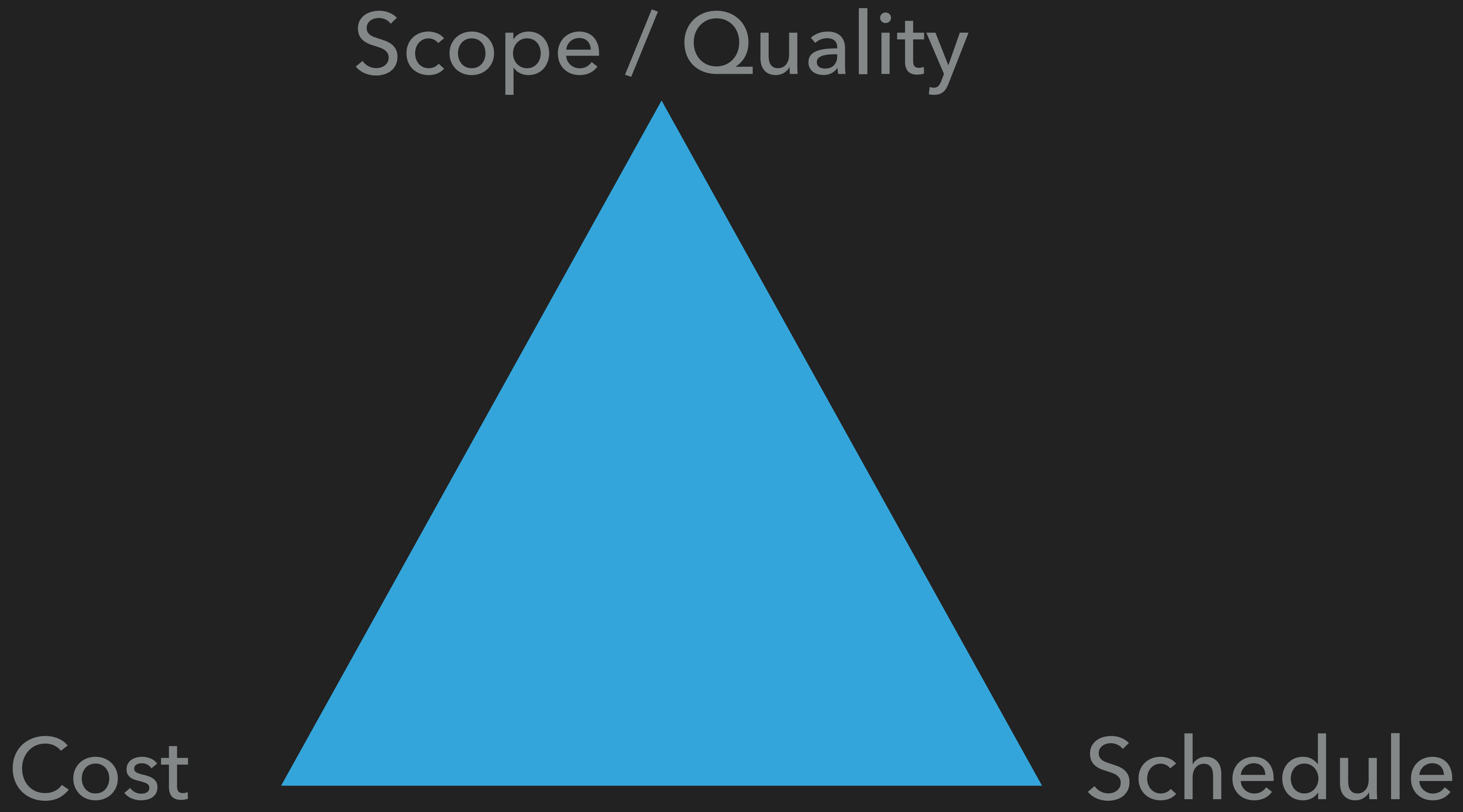
REMINDER

The software developer is an essential piece of software development, but not the only part.

SO WE KNOW HOW TO WRITE SOFTWARE, RIGHT?

- There’s the plan, just follow it!
- We know this is not the case.
  - Software often has defects
  - Software often runs late
  - Software often does not meet the needs of the user

THE IRON TRIANGLE OF PROJECT MANAGEMENT



DEFINITIONS

- **Scope:** Features that are included in the project
- **Quality:** Is the program performant, bug-free, etc?
- **Cost:** How much does it cost to develop (in terms of team members and other resources)?
- **Schedule:** How long will it take?

Effort spent on improving one aspect will inevitably have impact on the others, usually negative.

TRADE-OFFS: THE BUGBEAR OF ALL ENGINEERS

- "Good, fast, cheap - pick two"
- "Grades, social life, sleep: pick two”
- Design it quickly and to a high standard, but it will be expensive. Design it quickly and cheaply, but it will be of low quality and/or be missing features. Design it well and cheaply, but it will take a long time.

THE IMPACT OF THE IRON TRIANGLE

- Software development is an immature field compared to other engineering disciplines
  - We've been building bridges since pre-history
  - We've been building software on a large scale for less than 70 years (ignoring Ada Lovelace, Alonzo Church, Alan Turing, etc.)
  - And it's complex and unforgiving!

WE OFTEN DON'T KNOW HOW THE IRON TRIANGLE WILL BE IMPACTED

- … and it is often impacted in ways different than in other disciplines!
- Brooks' Law
  - *"Adding manpower to a late project will make it later"*
- Why?

COMMUNICATION

- Necessary communications increase as (n * (n-1)) / 2.
- 1 = 0 comms path
- 5 = 10 comms path
- 10 = 45 comms paths
- 20 = 190 comms paths
- 100 =4,950 comms paths

ESTIMATING SCHEDULE

- Time to completion has almost always been underestimated in software development
- Why?

DONALD RUMSFELD, SECRETARY OF DEFENSE UNDER PRESIDENTS FORD AND BUSH

“[T]here are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns – the ones we don't know we don't know. And if one looks throughout the history of our country and other free countries, it is the latter category that tend to be the difficult ones.”

SOFTWARE ESTIMATION IS HARD!

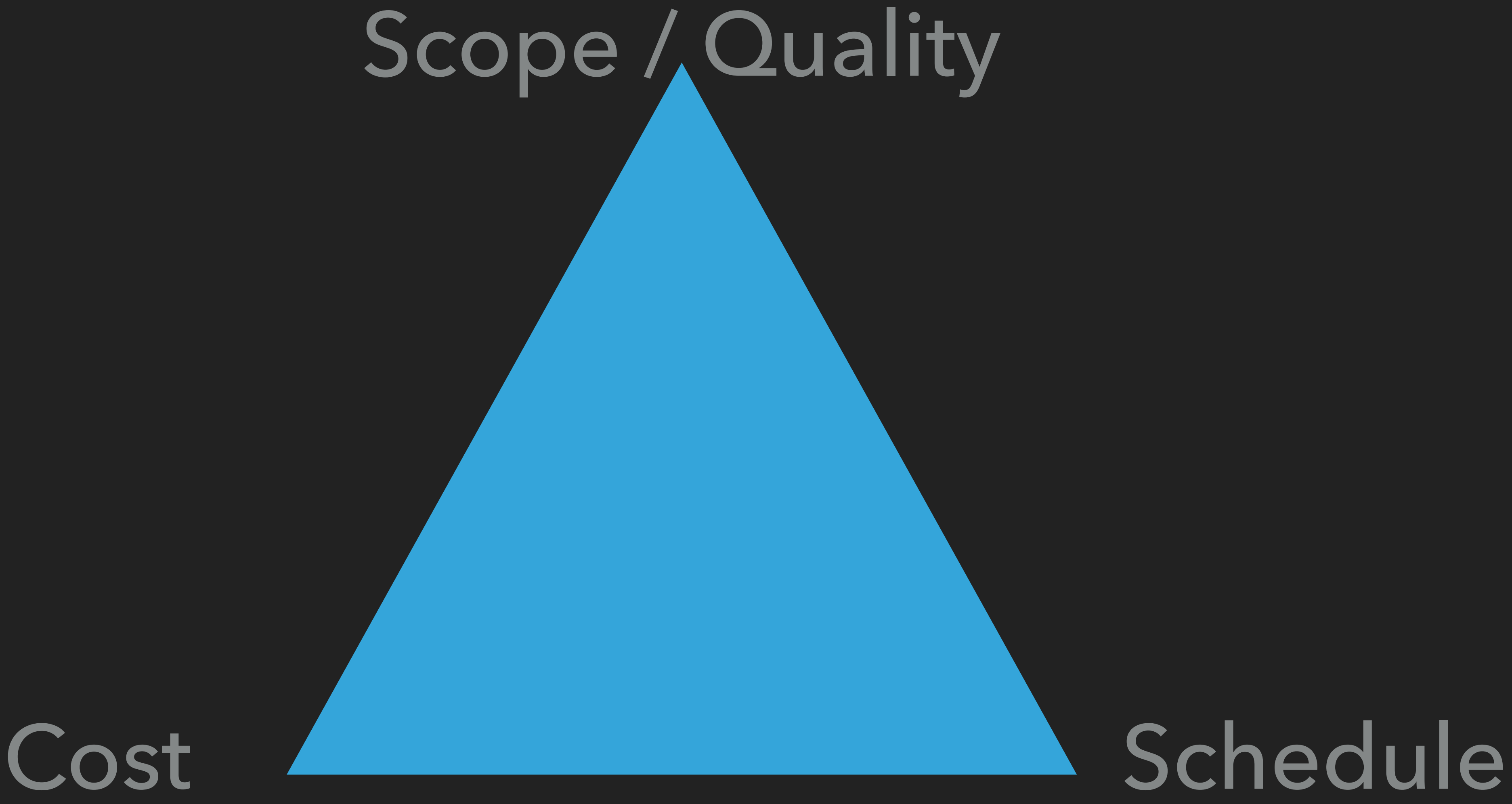
- Every software project is new - if it was something we'd done before, we'd have used the original project
- We don't know what challenges we will face, and it is often difficult to know beforehand
- Not every engineer is interchangeable
    - Skill sets and experience differ
    - Teams and management differ

DETERMINING ACCEPTABLE SCOPE AND QUALITY ARE DIFFICULT

- Users and customers may not know what they want, or may be unable to explain it
- Users and customers may provide inconsistent or contradictory requirements
- Engineers may misunderstand ambiguous or poorly-worded requirements
- Non-functional requirements (e.g. performance, security, scalability) may be poorly specified, or not specified

THE IRON TRIANGLE OF PROJECT MANAGEMENT

Software development has difficulties with every aspect of the iron triangle!



SOFTWARE SYSTEM != PROGRAM

- Program
- Programming System
- Programming Product
- Programming System Product

FRED BROOKS SAW ALL OF THIS…

- Sadly, many of the problems in The Mythical Man-Month are still with us.
    - Productivity increases sub-linearly with number of developers
    - Estimating is hard
    - People don’t think about “unknown unknowns”
    - … but we are finding better ways!