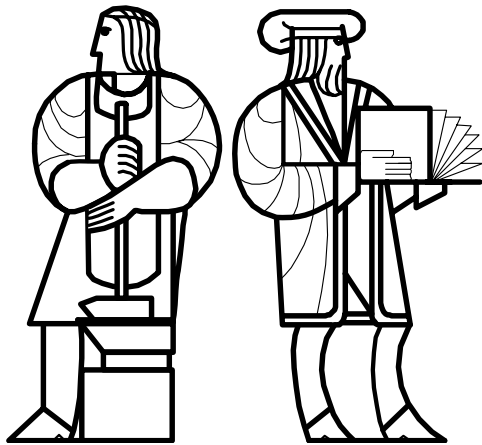




---

# 6.170 Lecture 23

## Wrapup



**MIT EECS**



# LESSONS of 6.170

---

**Modularity**

**Abstraction**

**Specification**

**Design**

**Documentation**

**Correctness**

**Teamwork**



**DIVIDE and CONQUER:**

**Modularity, abstraction, specs**

**No one person can understand all of a realistic system**

**Modularity permits focusing on just one part**

**Abstraction enables ignoring detail**

**Specifications (and documentation) formally describe behavior**

**Reasoning relies on all three to understand/fix errors**

Or to avoid them in the first place



# Getting it right ahead of time

---

**Design: predicting implications**

**Example: understanding interconnections**

**Understanding the strengths and weaknesses**

**If you don't understand a design, you can't use it**

**Documentation matters!**



# Documentation

---

**Everyone wants good documentation when using a system**

Not everyone likes writing documentation

**Documentation is the most important part of a user interface**

**What's obvious to you probably isn't obvious to others**

**“An undocumented software system has zero commercial value.” –John Chapin (CTO of Vanu, Inc.)**



# Maintenance

---

**Maintenance accounts for most of the effort (often 90% or more) spent on a successful software system**

**A good design enables the system to adapt to new requirements while maintaining quality**

Think about the long term, but don't prematurely optimize

**Good documentation enables others to understand the design**



# Correctness

---

## **In the end, only correctness matters**

*Near-correctness* is often easy!

*Correctness* can be difficult

## **How to determine the goal?**

Requirements elicitation

Design documents for the customer

## **How to increase the likelihood of getting there?**

You are unlikely to achieve it without use of modularity, abstraction, specification, documentation, design, ...

Doing the job right is usually justified by return on investment (ROI)

## **How to verify that you achieved it?**

Testing

Reasoning (formal or informal) helps!

## **Returnin gave a little practice**



# Working in a team

---

**No one person can understand all of a realistic system**

Break the system into pieces

Use modularity, abstraction, specification, documentation

**Different points of view bring value**

**Work effectively with others**

Sometimes challenging, usually worth it

**Both technical and management contributions are critical**



# Go forth and conquer

---

## **System building is fun!**

It's even more fun when you build them successfully

## **Pay attention to what matters**

Use the techniques and tools of 6.170 effectively