

## 6.170 Lecture 1 Overview



Michael Ernst  
John Guttag  
MIT EECS

### Administrative Details

**Web site:** <http://www.mit.edu/~6.170/>

**Lecturers:**

Michael Ernst (in charge of subject)  
John Guttag

**Head TA:** Jeremy Nimmer

**Other TA's:** See web

**Schedule:** See web

**Policies:** See web

© Michael Ernst/John Guttag

Spring 2001

Slide 2

### Announcements

**To get credit for 6.170 you must**

- Be admitted by lottery
- Fill out online sign-up sheet by midnight tonight
- Provide names of preferred teammates
- Complete problem set 0 by Thursday

**Group project a big part of subject**

- Don't panic if you don't have preferred teammates

**Problem set 0**

- A diagnostic test, see web site

© Michael Ernst/John Guttag

Spring 2001

Slide 3

### Lecture and Recitation Style

**Complement class notes, not duplicate them**

- Do the reading assignments !

**Will use overheads (posted on Web)**

- Don't let us go to too fast

**Overheads not intended to serve as class notes**

- Will not make sense without lecture

**Questions welcome at any point**

- Don't mind prompted digressions

**Will occasionally request participation**

- Size of class presents a problem

**Recitations largely interactive**

- Strong set of TA's this term

© Michael Ernst/John Guttag

Spring 2001

Slide 4

### Expectations and Goals

**On entry you should**

- Be able to write small programs
- Diagnostic assignment due Thursday
- Know a small amount of discrete math
- Sets, inductive arguments
- Have successfully completed 6.001 or equivalent

**On exit you should**

- Be able to design large software systems
- Be able to write excellent medium size programs
- Be able to reason rigorously about such programs
- Know how to test and document software
- Work effectively as a member of team
- Know something about best practice in industry

© Michael Ernst/John Guttag

Spring 2001

Slide 5

### What Matters In Commercial Development

**Engineering is all about money**

- Time and cost matter
- Truth and beauty do not
- Engineers often forget this
- Business folks usually don't

**Software systems cost too much to build**

**Software systems take too long to build**

- Opportunity cost

**Software systems don't do what people want them to do**

- Or even need them to do


**Hardware systems have many of the same attributes**

- But there are important differences

© Michael Ernst/John Guttag

Spring 2001

Slide 6

 **Why Software Systems Are Different**

---

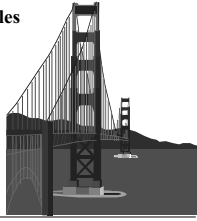
**Little separation between design and fabrication**  
Radical design changes during implementation

**Ill-defined goals**  
Enormous pressure for features


**Tight and rapidly changing schedules**  
Hard to anticipate needs

**Variety**  
Kinds of problems  
Solutions to problems  
Physics rarely intrudes  
Huge design space

**Intertwined with business strategy**



© Michael Ernst/John Guttag Spring 2001 Slide 7

 **Some Strategic Considerations**

---

**Hardware relatively stable**  
Expensive to change, especially in field


**Software more easily upgraded**  
All upgrades not equally easy

**Paradoxically, software often outlives hardware**  
Because people interact with it  
Because it can be upgraded

**Software locks in customers**  
Which monopoly will last longer, Microsoft or Intel?

**Design a critical issue**

© Michael Ernst/John Guttag Spring 2001 Slide 8

 **Developing Software Primarily a Design Activity**

---

**Process**  
Can differ greatly from project to project

**Schedules**  
Intermediate deadlines  
Staffing


**System**  
Intertwined with process -- Conway's Law

**Quality assurance -- Design plan up front; continuous**

**Maintenance -- A euphemism**  
Flexibility essential

**Risk reduction and contingency planning**  
Often neglected

© Michael Ernst/John Guttag Spring 2001 Slide 9

 **6.170 Emphasizes Specific Difficulties of Size**

---


**Goal: Make difficulty linear with respect to size**  
Rarely achieved

**Some industry studies suggest effort = length<sup>1.5</sup>**  
E.g., if 1 page takes a day, 100 pages takes 4 person years

**One page of delivered code/day considered good**  
Much code is not delivered

**Moral: Activities must proceed in parallel**

© Michael Ernst/John Guttag Spring 2001 Slide 10


 **Large Number of Programmers**

---

**Why?**  
One person would require too much elapsed time  
Need a variety of expertise

**Ramifications**  
Nobody understands whole system  
Opportunities for misunderstandings abound  
Communication costs considerable  
Management overhead

© Michael Ernst/John Guttag Spring 2001 Slide 11

 **Takes a Long Time**

---


**Why?**  
Big systems do complex things  
Live a long time and must be modified  
Too expensive to start over

**People forget things**  
And don't always realize it

**Personnel turnover (some of you will experience this)**  
People leave taking knowledge with them  
People arrive  
Must be brought up to speed  
Must live with decisions already made

**Documentation can ameliorate difficulties**

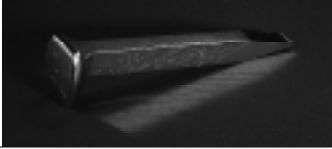
© Michael Ernst/John Guttag Spring 2001 Slide 12

 **Parallel Activities**


---

**Why?**  
Serial development requires too much elapsed time  
Sacrifice efficiency for reduced latency

**Ramifications**  
Harder to learn from mistakes  
Things may not fit together  
Testing a challenge




© Michael Ernst/John Guttag      Spring 2001      Slide 13

 **Moral**

---

**Must reduce complexity of software**  
Simplify requirements  
Find appropriate system architecture

© Michael Ernst/John Guttag      Spring 2001      Slide 14


 **Reducing and Ordering Complexity**

---

**Divide and rule is the key**


**We do this using**  
Decomposition  
Abstraction

**Decomposition creates structure**  
What kind of structure is best ?  
Big part of 6.170



**Abstraction suppresses detail**  
Trick is to suppress the right details

© Michael Ernst/John Guttag      Spring 2001      Slide 15

 **Main Topics Covered by 6.170**

---

**Abstraction and specification**  
Procedural, data, control flow  
Why they are useful and how to find them

**Writing and understanding good code**  
Examples in Java, but issues more general

**Program design**  
What makes a design good or bad  
The process of design and design tools

**Pragmatic considerations**  
Testing  
Debugging and defensive programming  
Managing software projects

© Michael Ernst/John Guttag      Spring 2001      Slide 16