

6.170 Spring 2000

6.170 Lecture 1 Overview



John Chapin
John Guttag
MIT EECS

Administrative Details

Lecturers:

John Chapin
John Guttag (me)

TA's: See handouts

Schedule: See handouts

Policies: See handouts

Everything else:

See web site, <http://www.mit.edu/~6.170/>

Announcements

To get credit for 6.170 you must

Be admitted by lottery

If you are a special case, see us after class

Fill out sign-up sheet tomorrow in lecture

Complete problem set 0 by Thursday

Group project a big part of subject

Provide names of preferred teammates on sign-up sheet

Staple sheets together in class

Don't worry if you don't have preferred teammates

Problem set 0

A diagnostic test, see web site

Lecture and Recitation Style

Complement class notes, not duplicate them

Do the reading assignments !

Will use overheads

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

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 about such programs

Know how to test and document software

Work effectively as a member of team

Know something about best practice in industry

Facts

Engineering is all about money

Time and cost matter

Truth and beauty do not

Unless they impact time and cost

Engineers often forget this

Business folks usually don't

Systems cost too much to build

Systems take too long to build

Opportunity cost

Systems don't do what people want them to



6.170 Spring 2000

6.170 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

Assignments will simulate issues



© John Chapin/John Guttag Spring 2000 Slide 7

6.170 Primarily a Design Activity

System
Makes an enormous difference
Intertwined with process -- Conway's Law
Not to be confused with Radin's Law

Process
Can differ greatly from project to project

Schedules -- Intermediate deadlines the key

Quality assurance -- Design plan up front; continuous

Maintenance -- A euphemism
Flexibility essential

Contingency plans

© John Chapin/John Guttag Spring 2000 Slide 8

6.170 6.170 Emphasizes Specific Difficulties of Size

Goal: Make difficulty linear with respect to size
Rarely achieved

Some industry studies suggest effort \propto length^{1.5}
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

Process qualitatively different because of
Number of people involved
Period of time required
Initial construction
Life cycle
Parallel activities

© John Chapin/John Guttag Spring 2000 Slide 9

6.170 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

© John Chapin/John Guttag Spring 2000 Slide 10

6.170 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

© John Chapin/John Guttag Spring 2000 Slide 11

6.170 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

© John Chapin/John Guttag Spring 2000 Slide 12

6.170


Spring 2000

 **The Only Solution**

Reduce complexity of software
Simplify requirements
Appropriate system architecture

Manage unavoidable complexity

© John Chapin/John Guttag Spring 2000 Slide 13


 **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



© John Chapin/John Guttag Spring 2000 Slide 14

 **Topics Covered by 6.170**

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

Programming in Java
Stylistic (not linguistic) issues

Program design
What makes a design good or bad
The process of design

Pragmatic considerations
Testing
Debugging and defensive programming
Managing software projects

© John Chapin/John Guttag Spring 2000 Slide 15