Principles of Software Construction: Objects, Design, and Concurrency

Course Introduction

Charlie Garrod     Christian Kästner
Construction of

Software Systems

at Scale
Libraries
Reuse
Design
Analysis
Concurrency
Software and automobiles

<table>
<thead>
<tr>
<th>Air-bag system</th>
<th>Antilock brakes</th>
<th>Automatic transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm system</td>
<td>Climate control</td>
<td>Collision-avoidance system</td>
</tr>
<tr>
<td>Cruise control</td>
<td>Communication system</td>
<td>Dashboard instrumentation</td>
</tr>
<tr>
<td>Electronic stability control</td>
<td>Engine ignition</td>
<td>Engine control</td>
</tr>
<tr>
<td>Electronic-seat control</td>
<td>Entertainment system</td>
<td>Navigation system</td>
</tr>
<tr>
<td>Power steering</td>
<td>Tire-pressure monitoring</td>
<td>Windshield-wiper control</td>
</tr>
</tbody>
</table>

1. Adaptive Cruise Control
2. Electronic Brake System MK60 E
3. Sensor Cluster
4. Gateway Data Transmitter
5. Force Feedback Accelerator Pedal
6. Door Control Unit
7. Sunroof Control Unit
8. Reversible Seatbelt Pretensioner
9. Seat Control Unit
10. Brakes
11. Closing Velocity Sensor
12. Side Satellites
13. Upfront Sensor
14. Airbag Control Unit
Moore’s Law: transistors per chip

Curve shows ‘Moore’s Law’: transistor count doubling every two years

Similar curve for memory, slightly steeper
How much software?

<table>
<thead>
<tr>
<th>System</th>
<th>Year</th>
<th>% of Functions Performed in Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-4</td>
<td>1960</td>
<td>8</td>
</tr>
<tr>
<td>A-7</td>
<td>1964</td>
<td>10</td>
</tr>
<tr>
<td>F-111</td>
<td>1970</td>
<td>20</td>
</tr>
<tr>
<td>F-15</td>
<td>1975</td>
<td>35</td>
</tr>
<tr>
<td>F-16</td>
<td>1982</td>
<td>45</td>
</tr>
<tr>
<td>B-2</td>
<td>1990</td>
<td>65</td>
</tr>
<tr>
<td>F-22</td>
<td>2000</td>
<td>80</td>
</tr>
</tbody>
</table>

Source: PM Magazine

Table 3.3a – System functionality requiring software

(informal reports)
The limits of exponentials

Computing capability

Human capacity
You’ve written small- to medium-size programs in 15-122

This course is about managing software complexity

- **Scale** of code: KLOC -> MLOC
- Worldly **environment**: external I/O, network, asynchrony
- Software **infrastructure**: libraries, frameworks, components
- Software **evolution**: change over time

- Contrast: algorithmic complexity
- Not an emphasis in this course
Our goal: understanding both the **building blocks** and also the **principles** for construction of software systems at scale
A framework for mobile app software (IOS)
The four course themes

- **Threads and Concurrency**
  - Concurrency is a crucial system abstraction
  - E.g., background computing while responding to users
  - Concurrency is necessary for performance
  - Multicore processors and distributed computing
  - *Our focus*: application-level concurrency
  - Cf. functional parallelism (150, 210) and systems concurrency (213)

- **Object-oriented programming**
  - For flexible designs and reusable code
  - A primary paradigm in industry – basis for modern frameworks
  - Focus on Java – used in industry, some upper-division courses

- **Analysis and Modeling**
  - *Practical* specification techniques and verification tools
  - Address challenges of threading, correct library usage, etc.

- **Design**
  - Proposing and evaluating alternatives
  - Modularity, information hiding, and planning for change
  - Patterns: well-known solutions to design problems
Motivating example #1: GraphLayout

Source code: http://java.sun.com/applets/jdk/1.4/demo/applets/GraphLayout/example1.html
Screenshot from http://stackoverflow.com/questions/1318770/impressive-examples-in-java
Discussion: GraphLayout

What does the design of GraphLayout look like, conceptually?

What is most important about the design?

How should the GUI be organized? Why?
Motivating example #2: Virtual Worlds
Discussion: Virtual Worlds

- How can the virtual world to scale to thousands of users?

- How can we organize the system to easily add new things?

- How can we support different kinds of things, while taking advantage of their similarities? (can you think of an example?)
Considering the examples

**Threads and Concurrency**
- In the GUI-based app
- On game clients
- On the game servers

**Object-oriented programming**
- Organizing by object types, then actions

**Analysis and Modeling**
- How to gain confidence regarding all possible executions

**Design**
- How to organize systems that grow and evolve
- How to define the interfaces between infrastructure and our code
Principles of Software Construction: Objects, Design, and Concurrency

Course Organization

Christian Kästner

Charlie Garrod
Course preconditions

• **15-122 or equivalent**
  ▪ 2 semesters of programming, knowledge of C-like languages

  ▮ **Specifically:**
  ▪ Basic programming skills
  ▪ Basic reasoning about programs
  ▪ Basic algorithms and data structures
Course postconditions

- **OO understanding**
  - Objects, classes, types
  - Java development skills

- **Understanding larger-scale software**
  - Design patterns
  - Design and use of libraries and frameworks

- **Modeling and analysis**
  - Use of development, testing, and analysis tools

- **Concurrent and distributed systems**
  - Scaling and performance
  - Safe programming practices for explicit concurrency
Important features of this course

• The team
  ▪ Instructors
    • Christian Kästner kaestner@cs.cmu.edu
    • Charlie Garrod charlie@cs.cmu.edu
  ▪ TAs
    • Daniel Lu dylu@andrew.cmu.edu [Section A]
    • Alex Lockwood alockwoo@andrew.cmu.edu [Section B]
    • Shannon Lee sjl1@andrew.cmu.edu [Section C]
    • Michael Maass mmaass@cs.cmu.edu [Section D]

The schedule

• Lectures
  • Tues, Thurs 3:00 – 4:20pm PH 100
• Recitations
  • A: Weds 9:30-10:20am WEH 5310
  • B: Weds 10:30-11:20am WEH 5310
  • C: Weds 11:30-12:20pm WEH 5310
  • D: Weds 12:30-1:20pm WEH 5310
• Office hours
  • To be announced – see course web page

Recitations are required
Important features of this course

• Course website
  ▪ Schedule, assignments, lecture slides, policy documents
    http://www.cs.cmu.edu/~charlie/courses/15-214

▷ Tools
  ▪ Subversion
    • Assignment distribution, handin, and grades
  ▪ Piazza
    • Discussion site – link from course page
  ▪ Eclipse
    • Recommended for developing code

▷ Assignments
  ▪ Homework 0 available tonight
    • Ensure all tools are working together
    • Subversion, Java, Eclipse

▷ First recitation is tomorrow
  ▪ Introduction to Java and the tools in the course
  ▪ Bring your laptop, if you have one!
    • Install Subversion, Java, Eclipse beforehand – instructions on Piazza
Course policies

• Grading (*subject to adjustment*)
  ▪ 60% assignments
  ▪ 15% midterm
  ▪ 20% final exam
  ▪ 5% participation

 Collaboration policy is on the course website
  ▪ We expect your work to be your own
  ▪ Ask if you have any questions
  ▪ If you are feeling desperate, please reach out to us
    • Always turn in any work you've completed before the deadline

 Texts
  ▪ Alan Shalloway and James Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design* (2nd Ed).
  ▪ Several free online texts (Java, etc.)
Course policies

• Late days for homework assignments
  ▪ 5 total late days for the semester
  ▪ May use a maximum of 2 late days per assignment
  ▪ No other late work accepted, except under extreme circumstances

Recitations
  ▪ Practice of lecture material
  ▪ Discussion, presentations, etc.
  ▪ Attendance is required
  ▪ In general, bring a laptop if you can
Principles of Software Construction: Objects, Design, and Concurrency

Objects

Charlie Garrod  Christian Kästner
Object orientation (OO)

- **History**
  - Simulation – Simula 67, first OO language
  - Interactive graphics – SmallTalk-76 (inspired by Simula)

- **Object-oriented programming (OOP)**
  - Organize code bottom-up rather than top-down
  - Focus on **concepts** rather than **operations**
  - Concepts include both **conventional data types** (e.g. List), and **other abstractions** (e.g. Window, Command, State)

- **Some benefits, informally stated**
  - Easier to reuse concepts in new programs
    - Concepts map to ideas in the target domain
  - Easier to extend the program with new concepts
    - E.g. variations on old concepts
  - Easier to modify the program if a concept changes
    - Easier means the changes can be **localized** in the code base
Objects

• **Object**
  • A package of state (data) and behavior (actions)

• **Data and actions**
  • **Fields** in the object hold data values
    • Like the fields of a struct in C
    • Access to fields can be restricted
  • **Methods** describe operations or actions on that data
    • Like functions associated with an abstract data type
    • They have access to all fields
    • Method calls can be thought of as "messages" to the object

• **Thus...**
  • **Methods** can control access to the fields
    • Best practice: Don’t allow fields to be seen from outside
  • The **object** can be thought of as a *service* that is accessed through a managed interface. The **class** described a family of similar services.
    • E.g., a particular button (object) vs. buttons in general (class)
Example: Concept of a Rectangle

• What do you need to **know** about a rectangle?

• What might you want to **do** with a rectangle?
Example: Points and Rectangles

class Point {
    int x, y;
    int getX() { return this.x; } // a method; getY() is similar
    Point(int px, int py) {this.x = px; this.y = py;} // constructor for creating the object
}

class Rectangle {
    Point origin;
    int width, height;
    Point getOrigin() { return this.origin; }
    int getWidth() { return this.width; }
    void draw() {
        this.drawLine(this.origin.getX(), this.origin.getY(), // first line
                      this.origin.getX()+this.width, this.origin.getY());
        ... // more lines here
    }
    Rectangle(Point o, int w, int h) {
        this.origin = o; this.width = w; this.height = h;
    }
}
Toad’s Take-Home Messages

• 214: managing complexity, from programs to systems
  ▪ Threads and concurrency
  ▪ Object-oriented programming
  ▪ Analysis and modeling
  ▪ Design

❖ GraphLayout and virtual worlds illustrate some challenges

❖ Object-oriented programming organizes code around **concepts**
  ▪ Methods capture behavior, fields capture state
  ▪ As we will see, this organization allows
    • Greater reuse of concepts
    • Better support for change when concepts vary