Principles of Software Construction: Objects, Design, and Concurrency

Course Introduction

Christian Kästner  Charlie Garrod
Construction of Software Systems at Scale
Libraries
Reuse
Design
I/O, GUI
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
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 Airbags
13. Upfront Sensor
14. Airbag Control Unit
# How much software?

<table>
<thead>
<tr>
<th>n 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>

**Millions of Lines of Code (MLOC)**

[Bar chart showing lines of code for different operating systems: Vista, XP, Win 2000, NT 4, NT 3.5, NT 3.1]
Moore’s Law: transistors per chip

CPU Transistor Counts 1971-2008 & Moore’s Law

- Similar curve for memory, slightly steeper
The limits of exponentials

Computing capability

Human capacity

**capability**

**time**
Scaling Up: From Programs to Systems

- 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, design for change
  - Understanding: writing maintainable code
  - Correctness: testing, static analysis

- In 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: 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
After 214?

- 214 consists primarily of code-level software engineering, including the design of systems and applications
  - TOAD

- 313: Foundations of software engineering
  - Human and business aspects
  - Plan the process for and manage a software project, manage risk, coordinate teams
  - Elicit, describe, and evaluate a system's requirements
  - Design a software system and evaluate a design with regard to various quality attributes (software architecture)
  - Develop and justify a quality-assurance strategy for a software project (static analysis, inspection, ...)
  - Business models and open source

- 413: Software Engineering Practicum (a project course)

- Software Engineering Minor
Toad’s Take-Home Messages

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

• 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