Software Development Methods

Introduction

Tsay, Yih-Kuen and Wang, Bow-Yaw

{tsay,bywang}@im.ntu.edu.tw

Dept. of Information Management
National Taiwan University
What Do We Ask of a Program/Software?

1. Does what it is supposed to do, i.e., correct
2. Carries out its tasks efficiently
3. Friendly to the user
4. Well-structured and easy to maintain
5. Cheap to develop
Why This Course?

🌍 From our personal (perhaps biased) observation, students (and hence practicing programmers)

Solar 🌞 rarely care about writing “good” programs,

Solar 🌞 know few useful programming techniques, and

Solar 🌞 cannot use development tools effectively.

🌍 Result: low-quality commercial software

🌍 Question: Shouldn’t you start to care (if you haven’t)?

Note: in this course, a good program is one that is at least correct and well-structured.
Course Objectives

Learn how to develop correct and high-quality software with better engineering skills:
- The UML
- Design patterns
- Verification tools

Get exposed to a bit of formality so that you will be able to describe and reason about programs more precisely.

Note: there are numerous other software development methods. You are encouraged to explore them through course taking or self-study.
Programming in Class

- Environment is controlled
- Problems are well-defined (sorting, BFS, etc.)
- Solutions are well-defined (in your algorithm textbooks)
- Programs seldom change (write once, use once)
- Correctness may not be an issue
- Robustness has never been an issue
Programming in Real World

- Environment is open
- Problems are *not* well-defined
- There may be multiple options available
- Programs change all the time
- Correctness is most important
- Robustness is necessary
Example – an Inventory System

🌍 A 24-hour store asks you to develop an inventory system

🌟 The system will be used by many people.
🌟 It is impossible to know what goods or categories the store will have.
🌟 What database and user interface packages would you use?
🌟 What if they ask you to add new features?
🌟 Your system’d better not be confused by different calendar systems.
🌟 Your system’d better be able to be working all year long.
What about Software Project Management?

- Software development, after all, will be done by engineers.
- Project leaders need to know what engineering options they have.
- We will look at the problem from an engineer’s point of view.
- The material should be complementary to related software project management courses.
After several meetings with your client, you have an informal idea of what your client wants.

You bring the informal idea back and start developing the system with your colleagues.

But your colleagues did not participate the meetings. They are not so familiar with domain knowledge as you are.

What would you do?
Suppose you would like to develop a sorting algorithm for any totally ordered set.

A set $S$ is totally ordered if either $a < b$, $a = b$, or $a > b$ for any $a, b \in S$.

How do you convey the idea to your colleague?
Modeling Totally Ordered Sets

🌐 An element of a totally ordered set is an object of class TOSet.

🌐 The class TOSet has a static member function compare (TOSet &, TOSet &) that compares two elements.

🌐 We can create an object and assign its value.
The sorting function accepts an array of `TOSet` objects as inputs.

It uses `compare (TOSet &, TOSet &)` to compare elements in the array.

It outputs a permutation of the input array such that the elements in the permutation are ordered by the `compare (TOSet &, TOSet &)` function.
Problems

- It is still ambiguous. (What do you mean by “ordered by the compare (TOSet &, TOSet & ) function?”)
- It is not complete. (What is a permutation?)
- It is written in natural language.
- It is already too complicated. (What if you have 30 classes in your system?)
Unified Modeling Language

- UML is designed for program specification.
- It is a graphical language.
- It tells you the relation among different classes.
- It has rigorous semantics (that is, not ambiguous).
- There are tools that can simulate your UML designs.
From Specification to Design

Software development is more than specification

🌱 UML Specification is a way of communication

🌱 Like natural languages, you may know the words and grammar of English. But you still may not compose a good essay in English.

🌱 After learning UML, we will discuss useful programming techniques for system design.
An Exercise

Compute

\[ \int x^3 \ln^3 x \, dx = ? \]
\[
\int x^3 \ln^3 x \, dx = \frac{x^4}{4} \ln^3 x - \int \frac{x^4 \ln^2 x}{4} \, dx
\]
\[
= \frac{x^4}{4} \ln^3 x - \frac{3}{4} \int x^3 \ln^2 x \, dx
\]
\[
= \frac{x^4}{4} \ln^3 x - \frac{3}{4} \left[ \frac{x^4}{4} \ln^2 x - \int \frac{x^4 \ln x}{4} \, dx \right]
\]
\[
= \frac{x^4}{4} \ln^3 x - \frac{3}{16} x^4 \ln^2 x + \frac{3}{8} \int x^3 \ln x \, dx
\]
\[
= \frac{x^4}{4} \ln^3 x - \frac{3}{16} x^4 \ln^2 x + \frac{3}{8} \left[ \frac{x^4}{4} \ln x - \int \frac{x^4}{4} \, dx \right]
\]
\[
= \frac{x^4}{4} \ln^3 x - \frac{3}{16} x^4 \ln^2 x + \frac{3}{32} x^4 \ln x - \frac{3}{32} \int x^3 \, dx
\]
\[
= \frac{x^4}{4} \ln^3 x - \frac{3}{16} x^4 \ln^2 x + \frac{3}{32} x^4 \ln x - \frac{3}{128} x^4
\]
Strategies and Patterns

- What strategies do we have?
  - polynomial integration
  - integral of $\ln x$
  - variable substitution
  - integration by parts

- The problem is solved by choosing combinations of strategies.

- What about program development? Is there any strategy or pattern for programming?
Suppose you want to implement a database.
The user may ask you to search or sort by field.
You may use sorting algorithms, search algorithm, even balanced tree data structures.
For different situations, you may use different sorting algorithms (e.g., memory- versus disk-based).
You do not develop your program from scratch.
What about System Architecture?

Suppose you want to develop a system for

- vehicle controller
- user interface
- data management

Is there any known strategy or pattern that could be applied?
Example – Vehicle

Let’s suppose we want to define a vehicle rental system at seashore resorts. They have bikes, cars, sailboats, and yachts

- Class **LandVehicle** for bikes and cars
- Class **WaterVehicle** for sailboats and yachts

One day, a resort management team decides to introduce hovercrafts. How would you modify the class hierarchy to include the new product?
Design Patterns

- An objected-oriented programming technique for system design
- A collection of class hierarchies
- Used in commercial tools and systems
A software developed by proper methodologies does not necessarily entail quality.

🌍 UML specifications allow clients, system architects, and programmers to communicate.

🌍 Design patterns help system architects and programmers to deploy software structures sensibly.

But they do not imply the system cannot go wrong.
Damages

- Device drivers
- Stock Exchange
- Medical instruments
What are the Problems?

- Programming errors
- Design flaws
A Lesson from the Hardware Industry

- The first Pentium was discovered to have the infamous F00F bug.
- IC manufacturing costs lots of money.
- No company would like to have a buggy design to be sent to the foundry.
- But how?

Note: the “Pentium floating point divide” bug (in 1993) ultimately costs Intel US$475 million.
Testing and Verification

IC design houses use tools to help them catch bugs.

💡 Testing: Run simulation on designs to find bugs

💡 Verification: Analyze designs to prove they are correct
Testing

- It can check the system before it is implemented.
- Simulator generates random inputs.
- Erroneous behavior can be observed if proper inputs are generated.
Verification

- It can check the system before it is implemented.
- Verification tools try all possible inputs.
- Erroneous behavior can be observed if proper inputs are generated.
- Correctness can be ensure if all inputs have been tested.
Ingredients of Verification

- Behavior Modeling
- Property Specification
- Verification Algorithm/Tool
  (or, if that fails, Proof and Proof Checker)
Behavior Modeling

- It describes system behavior.
- We need a formal language to avoid ambiguity.
- Unlike typical programming languages, the control flow of programs is of concern.
- Users specify their systems as models in modeling languages.
Property Specification

- It specifies what properties are of interest.
- Another formal language is needed.
- High-level properties are independent of the implementation.
- Users specify the requirements in property specification languages.
Automatic Verification Tools

- A verification tool takes the model and property specification as input.
- It checks whether the model has the property or not.
- Many verification problems are undecidable and some work-around techniques (e.g., abstraction) may help.
Correctness Proofs and Proof Checking

- Correctness proofs are the last resort, when everything else fails.
- Unfortunately, proofs are usually hard to produce.
- Even worse, you can make mistakes in a proof.
- Fortunately, checking if a proof is really a proof can be automated.
Programming in the Small

- We will also study development methods that probably only work for smaller programs.
- However, a larger program is composed of smaller ones.
- Making the smaller programs correct helps improve the overall quality of the larger one.
Conclusion

This is a course that views software development from an engineer’s viewpoint.

It covers design and programming techniques for software development.

It introduces you to useful verification tools.

We hope you’ll appreciate the methodology and improve software quality with better engineering skills.