EECS4314 week 3

Jerry Wu

2024-01-24

Contents

Lecture 1

Software Architecture

Evolution of programming abstractions

- First computers used punchcards and were hardwired
- First software was written in assembly
- In the 1960s, high level programming languages like Fortran, Cobol, Pascal, C, etc. were introduced to make code more readable and easier to use/understand
- ADTs and OOP in the mid 1970s
- Design patterns first introduced in 1990s, allows object reuse

Why software architecture?

- As software size and complexity increases, the design problem goes beyond data structures and algorithms
- Designing and specifying the overall system structure emerges as a new kind of problem
- Recall reference architecture and product line architecture

Software architecture issues

- Organization and global control structure
- Protocols of communication, synchronization, and data access
- Assignment of functionality to design elements

- Physical distribution of product
- Selection among design alternatives

State of practice

- Currently, there is no well defined terminology or notation to characterize architectural structures and systems
- Good software engineers will make common use of architectural principles when designing complex software systems
- These are simply principles or idiomatic patterns that have emerged over time

Descriptions of software architectures

- **Example 1**
  - Camelot is based on the **client server model** and uses remote procedure calls both locally and remotely to provide communication among apppplications and servers
  - We have chosen a distributed **object oriented** approach to managing information
- **Example 2**
  - **Abstraction layering** (layered architecture) and system decomposition provide the appearance of system uniformity to clients, yet allow HeliX to accomidate a diversity of autonomous devices (self managing systems like server management, not self driving cars)
  - The architecture encourages a **client server model** for structuring of applications
- **Example 3**
  - The easiest way to make a canonical **sequential** compiler into a **concurrent** compiler is to **pipeline** the execution of the compilar phases over a number of processors
  - A more effective way is to **split the source code into many segments which are concurrently processed through the various phases of compilation** (by multiple compiler processes) before a final **merging** pass recombines the object code into a single program

Some standard software architectures

- **ISO/OSI reference model** is a layered network architecture
- **X Window System** is a distributed windowed user interface architecture based on event triggering and callbacks. Linux can run without a GUI, but it can have a windowed GUI thanks to this.
- **NIST/ECMA reference model** is a generic software engineering environment architecture based on layered communication substrates