Handout 01: Creating a Program versus Building a System

Required Reading

- Chapter 1: Creating a Program, from Tsui, Karam, & Bernal, 2018
- Chapter 2: Building a System, from Tsui, Karam, & Bernal, 2018

Overview

Chapters 1 and 2 demonstrate the difference between a small programming project and the effort required to construct a mission-critical software system. [The authors] purposely took two chapters to demonstrate this concept, highlighting the difference between a single-person “garage” operation and a team project required to construct a large “professional” system. The discussion in these two chapters delineates the rationale for studying and understanding software engineering. (Tsui, Karam, & Bernal, 2018, p. xiv)

Objectives for Chapter 1: Creating a Program

- Analyze some of the issues involved in producing a simple program
    - Requirements (functional, nonfunctional)
    - Design constraints and design decisions
    - Testing
    - Effort estimation
    - Implementation details
- Understand the activities involved in writing even a simple program
- Preview many additional software engineering topics found in the later chapters

Objectives for Chapter 2: Building a System

- Characterize the size and complexity issues of a system
- Describe the technical issues in development and support of a system
- Describe the nontechnical issues of developing and supporting a system
- Demonstrate the concerns in the development and support activities of large application software
- Describe the coordination efforts needed for process, product and people; these software engineering topics are expanded in later chapters

Terms and Concepts to Know

- Program requirements: functional requirements versus nonfunctional requirements
- Design constraints
- Design decisions
- Verification testing versus validation testing
- Acceptance testing versus unit testing
- Black-box versus white-box testing
- Effort estimation: cost estimate versus schedule; “How much effort?” (total person hours) versus “How long?” (elapsed time)
- Software development process

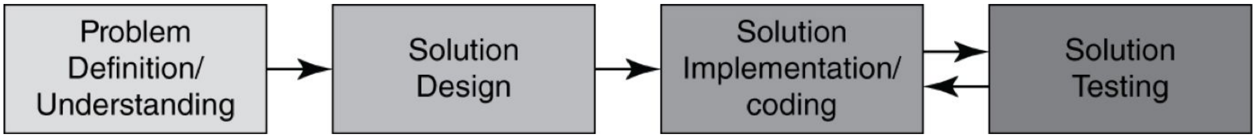
Consider a “Simple” Problem...

A “Simple Set” of Steps (Sequence of Activities) for Creating a Program

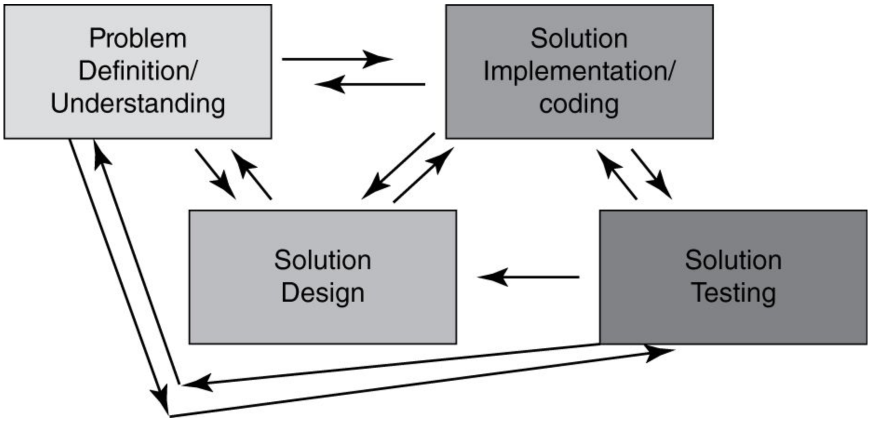
1. **Understand the problem** - requirements
   - Functionalities
   - Non-functionalities: performance, security, modifiability, marketability, etc.
2. **Perform some design** - based on requirements
   - Organize the functionalities in some sequence; possibly using some diagrams.
   - Focus on input/output (data, formats, organization).
   - Think about some constraints (non-functionalities) such as speed, UI looks, programming language, dependencies, etc.
   - Any specific algorithm and improvements on sequence of functionalities.
3. **Code/Implement** - turning the design into actual code
   - Depending on how much design is completed, one may either directly engage in conversion to code (language dependent) or do some more designing.
4. **Verify/Test the program** - check the program results (via output) with some predetermined expected set of inputs
   - The pre-determined inputs are *test cases* and require some thinking.
   - If the results do not match what is expected then:
     - “Debug”
     - Fix
     - Retest — reverify
   - *Stop* when all test cases produce the expected results.
   - Question: How many test cases should we develop and run?

What Really Happens?

“Imagined” - Ideal Situation


Problem Definition/ Understanding
Solution Design
Solution Implementation/ coding
Solution Testing

“Actual” - Real Situation


Problem Definition/ Understanding
Solution Implementation/ coding
Solution Design
Solution Testing

It’s Done! What else matters?

- How long (elapsed time) did it take to complete the work?
- How much effort (total person hours) is expended to do the work?
- Does the solution solve the complete problem?
- How “good” is the work — (code, design, documentation, testing, etc.)?
  - And who determines what is “good”?

Consider a “Simple” Problem…“Write a program in your favorite language that will accept numerical numbers as inputs, compute the average, and output the answer.”

- How long (in elapsed time) would it take you to implement this solution?
- How much overall effort (in person hours) will this take?
- How well will your solution match the problem?
- How good is your code/design/documentation/testing?

Building a System

- Moving from writing a program to building a system. What’s the difference?
- What is a **system**? What is a system involving software?
- What are some characteristics of building a system?
  - Size and Complexity
    - Breadth of complexity
    - Depth of complexity
  - Technical Considerations of Development and Support
    - Problem and design decomposition
    - Technology and tool considerations
    - Process and methodology
  - Nontechnical Considerations of Development and Support
    - Effort estimation and schedule
    - Assignments and communications

Consider a Large, Complex System

Building a mission-critical or business-critical system requires (1) several separate activities performed by (2) more than one person (e.g., 50 ~ 100):

- **Requirements**: gathering, analysis, specification, and agreement
- **Design**: abstraction, decomposition, cohesion, interaction, and coupling analysis
- **Implementation**: coding and unit testing
- **Integration** and tracking of pieces and parts
- **Separate testing**: functional testing, component testing, system testing, and performance testing
- **Packaging** and **releasing** the system

Also, Need to Support (and Maintain) the System (because it is complex)

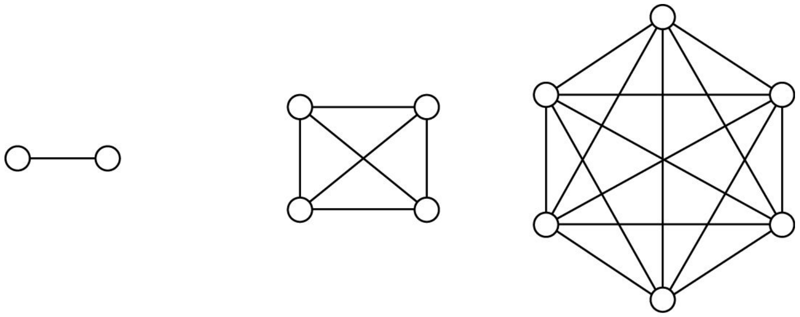
- Pre-release: preparation for education and support:
  - Number of expected users
  - Number of “known problems” and expected quality
  - Amount of user and support personnel training
  - Number of fix and maintenance cycle
- Post-release: preparation for user and customer support:
  - Call center and problem resolutions
  - Major problem fixes and code changes
  - Functional modifications and enhancements

Coordination Efforts Required in Systems Development and Support

Because there are (1) more parts, (2) more developers, and (3) more users to consider in large systems than a single program developed by a single person for a limited number of users, there is the need for coordination of 3Ps:

- **Processes** and methodologies to be used
- Final **product** and intermediate artifacts
- **People** (developers, support personnel, and users)

For n people, the number of potential communication paths = n * (n − 1) / 2.



2 people:
1 path

4 people:
possibly 6 paths

6 people:
increase to
potentially 15 paths

As the number of people increase, the necessary amount of communications increases, and the number of communications errors also tends to increase.

Question: What is a software development process or methodology that you have used to help coordinate the 3Ps?

References

Tsui, F., Karam, O, & Bernal, B. (2018). *Essentials of Software Engineering* (4th ed.). Burlington, MA: Jones & Bartlett Learning.