The Software Lifecycle

Examining the phases of large-scale software development projects

Jeff Stephenson

Software Lifecycles

- Software Engineering vs. Programming
  - What you have done for our past assignments is mostly *programming -- (write a method to do this, etc.)*
  - Programming is only one part of creating a software system
    - An incredibly important part!!!
    - In real systems, a **smaller part** than you would imagine.
  - For real systems, there is a lot of work to be done both before programming and afterwards.

Software Lifecycles

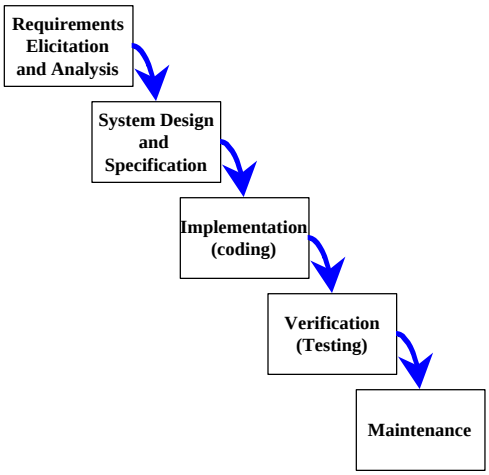
- Adopting an engineering approach to software development means taking a “holistic” view.
- A software system has a lifecycle.
  - Extending from the conception of the idea (hey, wouldn’t it be great if…..)
  - Until the system is no longer needed (always longer than the engineers anticipated -- Y2K bugs, etc).
- *Software Engineering* addresses all of the parts of the lifecycle.

Software Lifecycles

- Software projects need to be planned! (obviously)
- We’ll want to follow a *process* for developing software.
- Each published “process” for software development assumes a certain lifecycle for a proposed project.
- The process will take the project through a series of phases according to a lifecycle model.

Waterfall Model

- **This is one type of lifecycle model. Each phase is completed before moving on to the next phase.**
  - Requirements
  - Design/Specification
  - Implementation
  - Verification
  - Maintenance



Requirements Elicitation and Analysis

- Step 1: Requirements Elicitation/Analysis
  - The initial step of the process is defining (in as precise terms as possible) what problems the user needs the proposed system to solve.
  - The focus is on the PROBLEM, not the system. You should not worry about the constraints of technology (or your skill) when establishing the user requirements.

Requirements Elicitation and Analysis

- Step 1: Requirements Elicitation/Analysis (2)
  - The fundamental challenge of this stage is gaining an understanding of what the user truly wants.
  - Natural language descriptions are ambiguous, and users often operate on a different set of assumptions than developers.
  - Often, software engineers employ techniques such as:
    - Over the Shoulder Requirements Gathering
    - Use Cases/Scenarios
    - Prototypes/Mock-ups

Requirements Elicitation and Analysis

- Step 1: Requirements Elicitation/Analysis (3)
  - The deliverable of this phase is a document specifically describing the user requirements that the system must satisfy (SRS).
  - Traditionally, both the customer and the development team sign off on this document.
  - Errors at this stage are the most costly to fix (such as omitted requirements or misunderstood requirements).
  - Approx. 10% of total project time, and 5% of project cost expended during this phase (as high as 20% of time).

System Design and Specification

- Step 2: Design and Specification
  - We now know WHAT the system will do, but HOW will it be done?
  - In this phase we will describe the software architecture that we will later build.
  - We will most likely use both textual and graphical notations that correspond with a design methodology.

System Design and Specification

- Step 2: Design and Specification (2)
  - Structured Design (Flow Charts, Data Flow Diagrams)
  - Entity-Relationship Diagrams (Data Driven)
  - Object Oriented Design (Object Models, Class Diagrams, UML)
    - Show the classes to be created
    - What services (methods) each class will offer.
    - The relationship between classes (inheritance trees, association diagrams).
  - We may also use formal specification languages to specifically describe the behavior of the system and prove properties about

System Design and Specification

- Step 2: Design and Specification (3)
  - We may also use formal specification languages to specifically describe the behavior of the system
  - These languages are mathematically based and are often used when you need to prove some invariant of the system (such as “Our system will never deadlock”).
  - Examples of formal languages include:
    - Z (based on set theory)
    - CSP (specifically for concurrent systems).
    - NP (used along with a formal model checking program)

System Design and Specification

- Step 2: Design and Specification (4)
  - Errors at this stage are also very expensive to fix and a lot of effort is put into verifying designs (with CASE tools, etc).
  - Poor designs often lead to maintenance nightmares, and band-aided systems.
  - Typically around 30% of project time, and 20% of project cost will be expended in this phase.
  - The deliverable of this phase is a document which describes the software architecture of the proposed system in detail (SDD, etc.).

Implementation

- Step 3: Implementing the System
  - This part involves translating each of the modules described in the system design into functioning code.
  - 90% of a typical computer science education focuses solely on this phase
  - Implementation is made much easier if your choice of language supports your design methodology.
    - You won't have an easy time implementing an object-oriented design in a non-object-oriented language!
  - Typically, 30% of project time and 35% of project cost expended in this phase.

Verification

- Step 4: Testing the System
  - This part results from a larger concern of "quality assurance."
  - Several levels of testing occur.
    - Unit Testing -- Testing each module to see that it functions as specified
    - Integration Testing -- Testing the integration of several modules
    - System Testing -- Testing that the system meets the user requirements!!!
  - Typically, 30% of project time and 40% of project cost expended in this phase. (Yes, that is a huge amount.)

Maintenance

- Step 5: Maintaining the System
  - So, you have delivered the system. Congratulations!
  - However, in most cases, you are nowhere near finished.
  - Customers are good testers, and their needs change over time.
    - You may perform three types of maintenance:
      - Perfective Maintenance (improving the quality over time without changing functionality).
      - Adaptive Maintenance (changing the system to react to changing environments).
      - Corrective Maintenance (correcting errors found in the system).

Maintenance

- Step 5: Maintaining the System (2)
  - Depending on the project, maintenance costs can rise far above the original cost of developing the system.
    - IT systems for large companies (HR systems, Traveler's Insurance Claim-Agent system, POS systems, etc) require large amounts of maintenance.
    - Maintenance for commercial products (like Microsoft Word, etc) is often rolled into a new release.
  - The future maintenance needs of a system should be a major concern during system design and development.

Software Lifecycles

- Now you have seen the phases that most large-scale software projects go through.
- What do you think are the problems with using the waterfall model of a software lifecycle?

(: no looking ahead :)

Software Lifecycles

- The waterfall model was used for many years, and has proven to be a terrible model for software development.
  - Reason 1: All of the planning and design is done at the beginning of the project.
    - The problem is that during a two-year project (or even a two-month project) user requirements often change.
    - In the middle of development, a customer is likely to say "we don't need that any more, what we would really like is this…."
    - This will happen! And the waterfall model has no way to deal with this problem.

Software Lifecycles

- Reason 2: All of the testing is at the end of the project.
  - A scenario: One entire year is spent coding a new system for a new target platform. At the end of the implementation phase, the work of 10 developers is combined to form a system with 500,000 lines of code. This code is largely untested.
  - Result: CHAOS. Even good developers create bugs, and combining code with other people always causes unanticipated bugs. Debugging 500,000 lines of untested code is a nightmare that may never end.
  - There are tons of case studies of large software systems that get to this stage and are cancelled or restarted from scratch.

Software Lifecycles

- Thus, we can't realistically save all of the testing until the end, and we also need to revisit the requirements and design phases throughout the product.

Software Lifecycles

- A number of software lifecycle models have been introduced to deal with this problem
  - Prototyping
  - Iterative Development
  - Spiral Model
- In each of these models, you cycle through multiple iterations of requirements gathering, design, implementation and testing. Like several instances of the waterfall model!