EECS4314 week 1

Jerry Wu

2024-01-10

Contents

Lecture 1

Introduction, requirements

"It's software engineering, but advanced" - H.V. Pham 2024

SCRUM roles

- Product owner: Person who's responsible for the success of the product
- Stakeholders: Investors
- Users: general userbase population
- Team members: developers, testers, etc.
- Scrum master: overseer and manager of the product development as a whole

Software development life cycle

In general, we follow these criteria in order when developing a new software product using any process model (waterfall, AGILE, etc.):

1. **Requirements** - what do we need the software to do?
2. **Architecture** - blueprint for the design based on the hardware of the system
3. **Design** - how should we structure the code?
4. **Implementation** - actual coding
5. **Testing** - identify problems in the current cycle
6. **Maintenance** - continuously evolve and improve the product

Question

Which phase of this process lasts the longest and is most costly to make fixes and changes to?

The answer

Maintenance is the most costly and takes the longest, since it is an ongoing process. Requirements is the area that is most costly to fix and make changes to, since the requirements are the very foundations of the product idea.

- Statistically speaking, as the years have gone by, the percentage of project costs devoted to maintenance have also increased.

Types of software maintenance activities

- **Perfective**: add new functionality/features
- **Corrective**: fix faults, errors, and failures (MOST IMPORTANT ONE)
- **Adaptive**: new support for file formats, refactoring of code, etc

Common software development problems

- The largest problems appearing in ≈50% of responses
  - Requirements and specifications
  - Managing customer requirements
- Coding is usually a non issue relatively speaking (especially when there is a group of developers working together).

Costs of incorrect or incomplete requirements

- In 1981, 75-85% of all issues found in software can be traced back to:
  - Requirements
  - Design
- In 2000, out of 500 major projects, 70-85% of costs are due to:
  - Requirement errors
  - New requirements

High cost of requirement errors

- Errors during the design phase could fall into one of the following categories:
  - Errors that occured when a technical design was created from a correct set of requirements
  - Requirement errors that should have been detected earlier in the proess but "leaked" into the design phase of the project
- The latter category turns out to be particularly expensive, since:
  - The errors are misleading:

        * Developers are looking for design errors, but they are in fact in the requirements.
    - By the time the errors are discovered:
        * These issues lay low within the software and go undetected for an extended period of time, all the while causing errors. So as a result, we lose time.
        * There is also an administrative side to this, since we would have to go all the way back to the requirements in order to see where we went wrong.

- To repair an error, costs are incurred in:
    - Rework
    - Change orders
    - Corrective action
    - Scrap
    - Recall
    -

Requirements

Where do requirements come from?

Requirements for software can come from one of the following places:

- Users and stakeholders who have demands/needs (**raw requirements**)
- Analysts/requirement engineers
  - Elicit these demands and needs
  - Analyze them for consistency, completeness, and feasability
  - Formulate the requirements and write a specification list in formal language (REQUIRED BY LAW BY CRITICAL SYSTEMS; medical, vehicle etc)
  - Validate the gathered requirements and reflect the needs/demands of the stakeholders. Look for these responses:
    * Yes, this is what I'm looking for
    * This system will solve the problem

Types of requirements

- Functional requirements
  - Specify the function of the system.
  - Usually follows a form of a mathematical function:

$$f(i, state) \rightarrow (output, state_{new})$$

- Non-functional requirements
  - Quality requirements
    * Specify how well the system performs its intended functions
    * Performance, usability, maintenance, reliability, and portability
  - Managerial requirements
    * When will the product be delivered?
    * Verification - is everything that's needed present at the time of release?
    * What happens if things go wrong? Legal requirements.
  - Context/Environment requirements
    * Legal stuff, range of consitions in which the system should operate (VERY IMPORTANT FOR CRITIAL SYSTEMS)

Design and architecture

- Design
  - Inner structure of components
  - Low level, information hiding and interfaces make it easier to change
  - Mostly technical stuff like code
  - Makes sense for systems with KLOCs
  - Late in SWE life cycle
- Architecture
  - Structure of system (components/connectors)
  - High level and hard to change (better get it right!)
  - Concerned with technical and non technical requirements (security, legal, outsourcing, etc.)
  - Makes sense for systems with MLOCs
  - Early in SWE life cycle

Software Architecture

Official definition by IEEE

Architecture is the fundamental organization of a system embodied in its components, relationships to each other, and to the environment as well as the principles guiding its design and evolution.

- *System*: A collection of components organized to accomplish a specific function or set of functions
- For example:
  - An individual application
  - Systems in traditional sense
  - Subsystems, systems of systems, etc.
- Systems exist to fulfill one or more **missions** in its environment

Environment, Missions and Stakeholders

- Environment: Determines the setting and circumstances of developmenta, operational, political, and other influences upon that system
- Mission: a use or operation for which a system is intended by one or more stakeholders to meet some set of objectives
- Stakeholders

Kruchten's definition

An architecture is the set of **significant decisions** about the organization of a software system.

- structural elements of which the system is composed of together with their behavior

Architectural styles

Types of styles

- Live architecture
  - Just an **idea**, can be mostly concrete or completely abstract.
  - A "mental model" or wetware of sorts; may be fuzzy, inaccurate, incorrect, or incomplete.
- Complexity
  - Number talk. Simplifies the system by concentrating on structure, not content or semantics.
  - Cognitive complexity: How hard is it to understand or visualize?
- Reverse Engineering
  - Extraction of design or architecture from existing implementations and from developers
  - Design recovery, code reuse.

Determining style