Software Engineering

CS305, Autumn 2020

Week 5

Class Progress…

- Last week:
  - Unified Modeling Language (UML) behavioral diagrams
  - Testing Overview and System Testing
  - Generating Test Cases from Software Specifications
    - General Approach
    - Partition Category Method
    - Tool: TSL Generator for generating test specs using Partition Category method

Class Progress…

- This week:
  - Feedback on PA0 (SRS)
  - Software System Design
    - Architectural (high-level) design

Feedback on PA0

- What went well
  - Identifying what each of the sections in SRS were asking for.
    - Purpose, Scope, User and Software Interfaces, User Characteristics, Functional/Non-Functional Requirements
  - Very creative functional and non-functional requirements:
    - E.g. Demo of tool, Data cleanup on uninstallation,
    - E.g. DBMS interfacing, GUI addition, spell-checkers, extended statistics

Feedback on PA0

- What did not go well
  - 82% participation
  - Not asking questions (to the customer) about what is required and what is not required as part of deliverables

    who is the customer?
    - E.g. ignoring the requirement that the program should be able to run on the command line
  - Not specifying OS name and the command line software in "software interfaces"
  - Combining multiple functional requirements
  - Overlooking mandatory requirements
  - Forgetting to **rename** `SRS_Template.md`

Feedback on PA0

- Essential stuff:
  1. What is the software supposed to do?

     e.g. compute average length of sentences (in words)
  2. Why is the software required?

     e.g. to enable college students to analyze their essays
  3. How is the software supposed to be used (by the user)?

     e.g. run the software using a terminal (another piece of software), pass certain parameters, observe the output on the terminal
- Guideline:
  - Scope, Purpose: 1, 2
  - User Requirements: 1, 2, 3
  - System Requirements: 1, 2, 3

Software Design

Software Design

- An activity focusing on organizing the system to satisfy functional and non-functional requirements
- **Input** to this activity:
  - SRS document (focused on what to do)
- **Output** from this activity: a *blueprint*
  - Design document(s) (focusing on how to do it)
  - Should capture:
    - Structure
    - Behavior
    - Interaction
    - Non-functional properties

Why is Design Important?

- Good design ➡
  - easier to implement / code
  - easier to make changes to the code
  - easier to test
  - easier to maintain
  - easier to understand the impact of requirements changes

Does good design mean successful software?

Design Decisions and Impact

- Face thousands of design decisions in a large project
  - E.g. what data structure to use? Whether to allocate memory on stack or heap? What and how many parameters should a function accept? etc.
- Most decisions do not have an impact on the software success
- Some do have an impact – *architectural decisions*
  - Changes to these affect a large part or the whole system

Design Overview

- Architectural (high-level) design
  - Decompose the system into modules / components
  - Identify connections / interactions between them
- Detailed (low-level) design
  - Choose data structures
  - Select algorithms, protocols
- *(when applicable)* User Interface (UI) design
- Test the Design – make sure that the design meets functional and non-functional requirements

Software Architecture - Distinctions

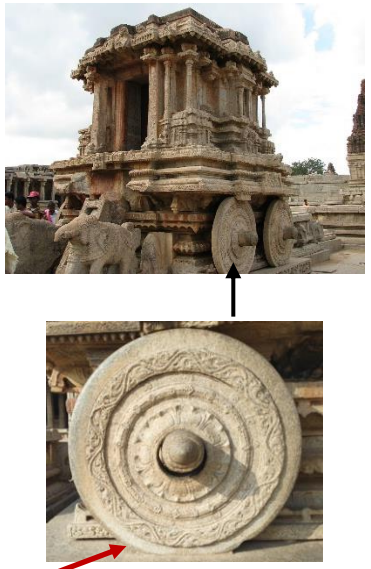
- Prescriptive:
  - as-conceived Software Architecture (SWA)
  - Captures design decisions made prior to software construction
- Descriptive
  - as-implemented SWA
  - Describes how the system has been built actually
- Often there is a gap / inconsistency between Prescriptive and Descriptive SWA

SWA Evolution

- SWA is not defined once. You do it iteratively, over time
- Ideally, prescriptive architecture should be modified first (e.g. changing the blueprint of a building)
- In software, often, descriptive architecture changes first and then the prescriptive architecture
  - Developer sloppiness
  - Tight deadlines
  - Non-existent prescriptive architecture etc.

SWA Evolution

- Important related concepts:
  - architectural drift
    - Introducing architectural design decisions orthogonal to system's prescriptive architecture
      - E.g. erecting ad-hoc structures
  - architectural erosion
    - Introducing architectural design decisions that violate a system's prescriptive architecture
      - E.g. cementing the chariot wheel



pic source: https://en.wikipedia.org/wiki/Hampi

SWA Ideal Characteristics

- Scalability
  - Ability of the software to handle growth e.g. adding more web servers to handle increased load in a web-based architecture.
- Cohesion
  - Measure of how strongly related are the elements of a module. Desired: high cohesion (e.g. a module should have a bunch of highly cooperating elements rather than independent, unrelated pieces)
- Coupling
  - Measure of how strongly related are different modules in the system. Desired: low coupling (e.g. to understand a module, one should not have to look at several modules)

SWA Elements

- SWA captures the composition and interplay of different elements:
  - Processing elements
    - Perform transformation on data
  - Data elements
    - Contain the data or information. Also called state.
  - Interaction elements
    - Glue that holds different pieces of SWA
- Components contain (Processing + Data) elements
- Connectors maintain and control interaction elements

Systems configuration