Software Design Primer
The SDLC

• Software Design Life Cycle
  • Requirements (elicitation and specification)
  • Design
  • Construction
  • Testing
  • Maintenance
Requirements

• Functional
  • What the software must do
  • Testable

• Non-functional
  • Constraints or quality requirements
  • Types: performance, maintainability, safety, reliability, security, interoperability, ...

• Quantifiable
  • avoid vague requirements – use quantitative where appropriate
Requirements

• Elicitation techniques
  • Interviews, scenarios, prototypes, facilitated meetings, observation, user stories
Design

“the process of defining the architecture, components, interfaces, and other characteristics or a system or component” ISO/IEC/IEEE 24765:2010 Systems and software engineering – Vocabulary

Two activities

- Architectural design: top-level structure and identifies components
- Detailed design: enough detail to build each component

Key issues

- Concurrency, event handling, data persistence, component distribution across hardware, exception handling and fault tolerance, interaction with users, security (authorization, attacks, cryptography)
Design

• UI design
  • Principles: learnability, familiarity, consistency, low surprise, recoverability, user guidance, user diversity
  • Modalities: question-answer, direct manipulation, menu selection, form fill-in, command language, natural language

• Design notations
  • Structural: class and object diagrams, entity-relationship diagrams, …
  • Behavioural: DFDs, flowcharts, state charts, …
  • Strategies: function-oriented, object-oriented, component-based, …
Construction

• Fundamentals
  • Minimize complexity, anticipate change, construct for verification, reuse, standards

• Test-driven development
  • Writing test cases before writing code

• Tools
  • IDEs, GUI builders, test frameworks, profiling
Testing

• Levels
  • Unit, integration, system

• Objectives
  • Acceptance, regression, performance, security, stress, recovery, usability
Maintenance

• Needed to
  • Correct faults
  • Improve design
  • Implement enhancements
  • Interface to new systems
  • Migration
  • Retirement

SWEBOK v3 Chapter 5, Section 1
SDLC Processes: Waterfall

- One stage finishes before next starts
- Requires stable, comprehensive requirements
  - A late change in requirements requires revisiting all stages
- Heavy on documentation
- System testing happens late
- Appropriate for safety-critical applications

https://existek.com/blog/sdlc-models/
SDLC Processes: Waterfall

- Lends itself to structured project planning
  - Work Breakdown Structures (WBS)
  - 100% rule

https://www.workbreakdownstructure.com/img-content/work-breakdown-structure-example.png
SDLC Processes: Waterfall

• Lends itself to structured project planning
  • Work Breakdown Structures (WBS)
    • 100% rule
  • PERT schedule analysis
    • Nodes connected by dependencies
    • Earliest start times calculated starting a nodes with no predecessors
    • Latest finish times back-propagated from nodes with no successors
    • Identify critical path and slack

SDLC Processes: Waterfall

- Lends itself to structured project planning
  - Work Breakdown Structures (WBS)
    - 100% rule
  - PERT schedule analysis
    - Identify critical path and slack
  - **Gantt chart**
    - Graph project timeline and track progress
- Can do all this with Microsoft Project (free download)
SDLC Processes: Agile

- [https://dzone.com/articles/3-styles-agile-iterative](https://dzone.com/articles/3-styles-agile-iterative)
  1. Iterative
  2. Incremental
  3. Evolutionary
SDLC Processes: Agile - Iterative

- Requirements may all be known a priori but they are added in stages
  - New requirements are added each iteration
  - High priority / high risk requirements are added early
- Useful for big corporations, banks

https://existek.com/blog/sdlc-models/
SDLC Processes: Agile - Incremental

- Similar to iterative but each increment is released to customer for feedback

http://testingfreak.com/incremental-model-software-testing-advantages-disadvantages-incremental-model/
SDLC Processes: Agile - Evolutionary

- No requirements document
- Start with a goal – single sentence or paragraph
- Prototypes are shown to customers for feedback
- Common in start-ups
Wireframing

- Designing an apps navigation and user interface
  - Ranges from paper prototype to interactive prototype

- Paper prototyping
  - Can be made interactive e.g. marvelapp.com

- Opensource GUI prototyping software
  - pencil.evolus.vn

- Useful for early customer feedback
Tools used in F18/W19

- user stories, issue tracking - PivotalTracker
  - Only 30 days free
- interface design - Figma
- frontend – ReactJs, ReactNative, Flutter
- backend – Firebase
- version control – github.com, git.uwaterloo.ca
AWS credits

- One student per team fill out [https://forms.gle/quPQaXzApuzbkGLD6](https://forms.gle/quPQaXzApuzbkGLD6)
Git

- **Documentation**
  - [https://git-scm.com/doc](https://git-scm.com/doc)

- **Distributed version control**
  - Clients mirror the repo

Git

- Clone – makes a local copy of the repository
  - `git clone repoURL`

- Edit – modify files in working tree

- Selectively add changes for next commit
  - `git add .` (changes in current dir)
  - `git add -A` (changes throughout tree)

- Commit to Git directory
  - `git commit -m "my changes"`
Git

- Checking changes
  - `git status`
  - `git diff` (only shows unstaged changes)

- Reverting an unstaged change
  - `git checkout modified_filename`

- Reverting a staged change
  - `git reset HEAD staged_filename`
Git

- See remote server (often called origin)
  - git remote
- Fetch all data from remote project
  - git fetch
Git

- Use branches to add features, fix bugs
- Branches
  - `git branch branchname` (creates local branch)
  - `git checkout branchname`
  - `git checkout master` (switch to master branch)
  - `git pull` (get updates from remote)
  - `git merge branchname` (merge branch changes into master)
  - `git branch -d branchname` (delete branch)
  - `git push origin branchname` (push to server)
  - `git push` (push changes to remote)
Git

- Pull requests
  - Fork a repo
  - Edit your copy and push to your remote master
  - Issue pull request to original repo
  - Owner can choose to pull changes into their repo