1 Agile Development

Project

- Temporary endeavor to create a unique product or service
- Quality is determined by: scope, cost, time

Waterfall

- Back in 1970
- Goes through each stage of software developement in series

Agile

- Around 2000
- Aim to address project issues during development, changing emphasis by clients, reducing reliance on documentations
- Flexible and lightweight

Ideas in agile

- Project inception
- Sprints
- Roles
- Standups
- Retrospectives
- Scope

Roles

- Scrum master, provides leadership and guidance, ensuring teamwork, facilitates discussion
- Product owner, deep knowledge of business, priority backlog, communicates with clients
- Developers, building products

Agile methods

- Kanban board (todo, doing, done)
- Scrum: divide development into sprints, product owner place client inputs into sprint backlog in sprint planning meeting, within each sprint complete the stuff in the backlog, after each sprint do sprint retrospectives
- Or any broad practical idea

Ceremonies

- Standups
- Sprints

Sprint

- Sprint planning, why sprint, what to do, who to do

- Standups, inspect progress, adapt if needed
- Sprint review, inspect outcome, present to stakeholder
- Retrospective, increase quality for future, what worked what didnt, focus on process

Sprint planning

- Adjust backlog
- Decide task to do
- Check with client
- Allocate work equally by estimating task sizes
- Metrics to estimate task sizes: t-shirt sizes, fibonacci numbers, poker

Standup

- Keep everyone on track
- Stop blockers
- Improve communication
- Everyone speaks
- What did I do since last? What will I do before next? Are there any blockers?

Sprint retrospective

- Focus on the process, not the outcome
- What did we do well, what could we have done better
- Brainstorming style or combining indep answers

Sprint review

- Focus on the outcome/product, often with client
- What is done, what isnt done

2 Requirements

Clients

- Important component
- Project must meet their needs/interests
- Client may not clearly express their needs

Requirement type

- Functional requirements, what the software needs to do
- Quality (non-functional) requirements, security/scalability/portability/maintainability, ease-of-use, performance
- Emotional requirements, what using the software should feel like

Requirement trajectory

- Inception: determines what to build, elicit client wants and needs, document client requests as requirements
- Sprints: updating requirements and priorities
- Conclusion: when the project is complete

Requirements in subject

- No fixed process due to varying projects, backgrounds, team capabilities
- Will give some recommended approaches so you can practise learning things

User stories

- Short description of a feature told from the perspective of the person who desires the new capability (usually a customer of the system.)
- Template of: as a user, I want goal so that this reason
- Organized into epics, which are collections of user stories. Each epic has a description summarizing the user stories inside it
- Each user story has a priority. Priority can be: essential/optional, high/medium/low, must/should/could/won't.

Persona

- Depicting a typical user of the software. From HCI to understand the customer

Motivational modelling

- Method to share a common understanding of the problem between client and developers
- Easy to connect to (verify with) requirements/designs.
- Symbols: emotional goals (how it should feel), functional goal (what it should do), quality goal (how it should be), roles (who achieves goal)
- A model contains functional goals each surrounded by roles/emotions-goals/quality-goals. Edges between functional goals implies relationship
- Three-stage process:
- Brainstorming to produce 4 lists of who/do/be/feel
- Transform list into single page model (using AMMBER)
- Get feedback from individual/clients and tweak model
- Produce diagram and ongoing monitor its relevancy

3 Design

Double diamond design model

- Discover (divergent thinking)
- Define (convergent thinking)

- Develop (divergent thinking)
- Deliver (convergent thinking)

Architecture design

- A communication exercise so people can understanding your thinking.
- No fixed method
- Contains: detailed architecture, high level architecture, ui design (templates, prototypes). Requires design diagrams
- Design diagrams are intended for: other team members, clients, future developers on the same projects

High level architecture

- Frontend is about the user interface
- Backend is about storing and retrieving information

4+1 architecture model

- Defines a set of views, each relevant to different stakeholders. They are:
- Logical view, for end-user functionality
- Development view, for programmers
- Process view, for integrators
- Physical view, for system engineers
- Scenarios, user stories and use-cases

Logical view

- Describes functional requirements of the system. Shows components of the system and their relationship
- Domain model, database ER model.

Process view

- Deals with the dynamic aspects of the system, how processes communicate at runtime
- Sequence diagrams

Development view

- The components from the programmer's perspective
- Architecture goals and constraints, package diagram, system diagrams (architecture diagram), api description

Physical view

- System from an engineer's perspective. Topology of software components at the physical layer
- Deployment diagram (flowchart for deployment, CI/CD)

Scenario

- Subset of important usecase
- Use case diagram

Low fidelity prototypes

- Ensures that interface is correct
- Basic visual elements, only key elements, no colors
- Test interaction with user

High fidelity prototype

- As similar to the product as possible
- Only develop after there is a solid understanding of the UI
- Realistic interactions, content, colors

4 Coding

Coding ideas

- Use coding standards
- Linters
- Careful about open source licenses

Repository

- Elements required: Readme, license, branch protection, notification/integration
- Development using meaningful pull requests
- Releases and tags

Git workflow

- Master branch contains only stable releases
- Feature branches contain commits for a specific feature. Merged into master when completed
- Develop branch used to protect master branch from unstable releases

Tech stack

- Key considerations are: client preference, team expertise, learning, resource constraint
- Freedom to choose any

Web Framework components

- Front end
- Back end
- Server

- Database

Pair programming

- Two developers working on the same task at the same time
- One is driver, one is reviewer
- Improves code quality, collaborator, productivity

Code review

- One person reviewing anther person's code during PR
- Helps to reduce bugs, improve quality, encourages discussion
- Tightly integrated with version control system
- Exact process depends on the team and negotiation

5 Testing

Software testing

- A process to identify the correctness, completeness, and quality of software.
- Contains a set of activities conducted with the intent of finding errors in software to correct before release
- Software bugs can be costly

Testing principles

- Exhaustive testing is not possible (testing all combinations is costly)
- Defect clustering (a small number of modules contain most of the defects)
- Pesticide paradox (if the same tests (testcases) are repeated, eventually they will not find new bugs)
- Testing shows presence of defects (hard to argue that software is defect free)
- Absence of errors is a fallacy (bug free code may not meet requirements)
- Testing is context dependent (cheaper to fix a defect in the early stage)

Testing methods

- Whitebox testing, testing your own code with knowledge of code
- Blackbox testing, testing others' code without looking
- Test-driven development

Test plan

- Document strategies, objectives, and tools
- Ensure everyone understands the objectives. Testing should not be neglected
- Includes: objectives, tools, scope, type of test completed, bug-report strategy

Testing types

- Unit tests, whether each software unit performs as expected
- Integration tests, checking whether multiple software components operate correctly
- Acceptance testing, if the application meets requirements (user-stories)
- Load testing
- UX testing

Manual or Automated

- Manual testing requires documentation detailing step by step instructions. Confirmed by another reviewer
- Automated testing allows automated running of testcases

Integration testing

- Big bang. All units combined and tested in one go
- Top down, top level units tested first, lower level tested later
- Bottom up, reverse of top down

Code coverage

- Common metric for examining thoroughness of test
- Proportion of code covered
- Level: function, statement, branches, condition, line coverage

Acceptance testing

- Ensure that system meets the user requirements
- Criteria written before implementation. Independently testable with defined pass or fail.

When to write tests

- Within Agile sprints, parallel to development if defined in the sprint planning
- Helps team collaboration and improve quality

Test case templates

- Use lecture slides

Load testing

- How well the program can cope with scalability
- Type of performance testing

User experience testing

- Client testing and using the program
- Check if the tool is easy to use

V model of testing

- Each stage of software development corresponds to a stage of testing

When should tests be ran

- Everytime when there is a change to the codebase

6 Ethics and professionalism

Professional

- Professional workers have standards they need to respect. Professional behaviors follows the standards of the profession
- IT workers are professional workers according to the lecturer
- Bad things can happen if one doesn't follow professional behaviors.

Code of ethics

- A list of rules that should be followed (adhered to)
- Examples: australian computer society code, engineers australia, software engineering

Law Compliance

- Illegal downloading of music and films
- Should not build software that violates the law
- Companies have minimum control unless you are profitable

7 Deployment

Before Devops

- Two teams: developer teams writing code, operations team deploys and monitors production
- Teams worked in isolation
- Lots of problems integrating code and managing release

Devops

- Merge developer and operations teams. Increases their collaboration and allows release through automated workflows
- Automation is using technology to perform tasks in a reproducible way. Minimum human intervention and detailed feedbacks
- Devops pipeline: a set of automated processes including CI/CD

CI/CD

- Continuous integration, integrating code changes onto master branch. Automated building and tests
- Continuous delivery, extends CI. Automates the release process so that the application can be deployed at anytime.

- Continuous deployment, extends CD. Automated deployment to production if all other stages are successful.
- Continuous delivery still requires a manual deployment stage (but everything is setup to make it easy). Continuous deployment automates the entire stage.

Why CI/CD

- Fast releases through automating the workflow
- Simple integration
- Fewer errors due to less human intervention
- Isolated failure from detailed feedback where in the pipeline has failed

Devops pipeline

- Involves a set of steps executed when a condition is met (PR)
- Need to perform checks before deployment
- Continuous integration: Push code, Automated build and test
- Continuous deployment: Review and Approve, Automated deploy.

Expectation

- Consider devops and deployment options
- Depends on client needs
- Webapps should have automated CI/CD
- Github actions too

Deployment

- SaaS, vendor manages everything: salesforce
- PaaS, vendor manages architecture: heroku
- IaaS, you manage everything: digitalocean

8 Handover

Conducting the handover meeting

- Demonstrate the application: initial project requirements, specific functionalities that were a focus
- Discuss the code and documentation: guide client through repo, highlight structure and key components
- Review outstanding items: communicate unimplemented features and reasons and suggestions for future
- Discuss accounts/api-key
- Arrange deployment on client infrastructure

Handover checklist

- Codebase
- Licenses and thirdparty licenses
- Readme files on repo structure
- Database access and credientials
- Hosting service access
- Database structure (ER diagram)
- Uh
- Key classes and application layers
- Key algorithms
- Project documentation (user stories, personas, motivational model, design document, meeting minutes, development log)
- User Manual, highlighting system use cases leveraging acceptance testing doc
- Startup and shutdown instructions
- Deployment guidelines, how to deploy the code

Post handover

- Provide contact information
- Follow up email