Software Development Life Cycle (SDLC)

Date No.

Verification Phase — Validation Phase

Requirements Analysis ← System Testing — System Test

Software Architecture ← Integration Testing — Integration Testing

Software Development ← Software Unit Testing — Unit Testing

Software Code

Requirements Analysis
Describe high level features

AIOT Coffee Maker
- Black Coffee
- Latte
- Cappuccino

Software Architecture
How the software should be layered & structured
1 software unit → 1 source code file

Black Coffee → black_coffee.py
Latte → latte_coffee.py
Cappuccino → cappuccino_coffee.py

Software Development
Implementation of Software Code
Implementation to follow Software Architecture structure

Software Unit testing
Test to ensure correct implementation

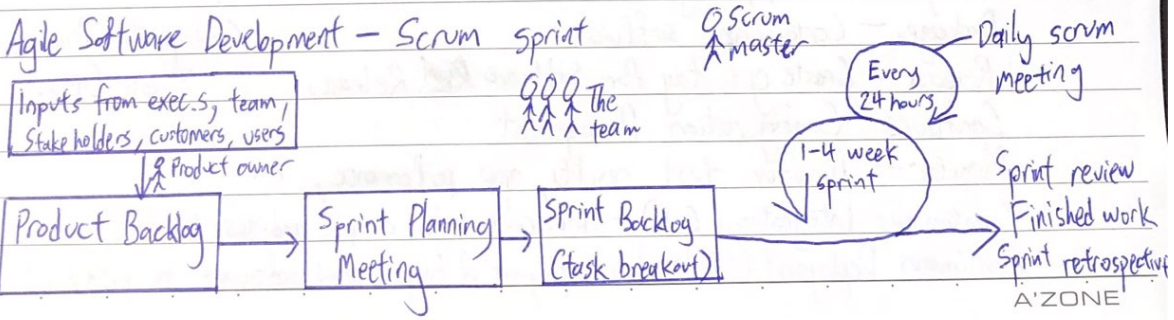
pytest

Integration testing
Test several software units to see if they work together

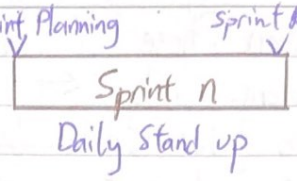
SDLC can be more efficient → Agile Software Development & Waterfall Development

Waterfall Development: Requirements → Analysis → Design → Code → Test → Maintenance
Project tasks pre-planned like in gantt chart
Waterfall assumes stable project requirements → Difficult to adapt when changes

Agile Software Development – Scrum



Scrum Master
Helps team understand the scrum method



Product owner
Plan User Stories for each sprint based on Priority and team capacity

Stand up meetings - 15 mins daily
3 questions shared by each scrum member
- What did I work on yesterday?
- What am I planning on working on tommorrow?
- Do I have any impediments that is blocking me?

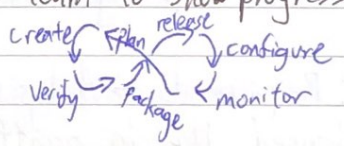
Opportunity for scrum team members to understand if anyone needs additional support and discuss how best the team can help each other

Sprint Planning
Product owner define priority of User stories in Backlog
Scrum team estimates complexity of each User story
Final list of User stories to be decided + assignment of user stories to each member

Sprint Review / Demo
Review and reflect on recently completed sprint - any improvements?
Is User Story completed based of User "definiton of done"?

Jira tool - Used by scrum team to show progress on all user stories

Devops framework



Create - Implement code
Plan - Sprint planning
Verify - Unit tests, Integration tests
Package - Containerize software
Release - Create git tag for Software Releases
Configure - Configuration Management
Monitor - Monitor test results, app performance, etc

Configuration Management System
eg Github

Continuous Integration (CI) - Auto. cont. build, compile, and test the latest code pushed into a CM syst.
Continuous Deployment (CD) - Auto. deployment of fully verified software to predefined targets

Date No.

Jenkins – CI – Automation of Build, Test and Deployment
Git / Github —— Configuration Management tool for archiving code
PyTest – CI – Automated Software Unit Test for python code
Docker – CD – Containerization tool
Kubernetes – CD – Deployment tool
K3S – CD – Lightweight version of Kubernetes ported for ARM microprocessors

---

The stupid main() implementation

```
def main():
    ...


if __name__ == '__main__':
    main()
```

```
strings = ["string", "word", "buh"] # list of strings
# sort ascending
sorted_strings = sorted(strings)          | strings.sort()          Output: ['buh', 'string', 'word']
print(sorted_strings)                     | print(strings)
# sort decending                                                    Output: ['word', 'string', 'buh']
sorted_strings = sorted(strings, reverse = True) | strings.sort(reverse = True)
```

```
def test_desc_on_test():
    result = []
    expected = []
    ...          # remember import filename (no .py ext) and when robbing functions from file
    result = [] # need put filename.funct()
    assert(result == expected)
```

cd /. → to root directory cd "path" → to specified path
git init → init git repos in current directory
git status → shows staged, unstaged and untracked files
git add * → adds all new or changed files to stage, for specific replace * with "<file>"
git commit -m "desc" → records changes in staging area to repos
git diff → shows differences between working dir and stage area
git branch "branchname" → create branch with name
git checkout "branchname" → switches to specified branch
git merge "branchname" → merge specified branch into current branch
git remote -V → displays current remote repos URL
git remote add origin <github link> → adds new remote repos under URL
git push --set-upstream origin master → set default push to origin URL
git push -u origin → push changes to origin URL
git tag -a v<no.> -m "desc" → create tag v<no.> with desc
git push origin v<no.> → pushes v<no.> tag to origin URL
git submodule add <github link> → adds submodule to URL repos
git clone <github link> → clones URL repos to local workspace
git log → shows commit history for current repos
git tag -d <tag v.no> → delete tag
git remote set-url origin <url> → replace origin url
file .gitignore → specify what file or folder to ignore in add and commit.
example in file, for file <filename.txt>, for folder __pycache__/