

Healthy Software Project Checklist

How to use this checklist?

These are the "Essential Development and project management Practices" for Healthy Project. It is prepared as checklist. Take print and tick how many practices you are following. Create a roadmap/plan to implement practices that you have not 'ticked'.

(Shameless Plug: A checklist is only as powerful as its execution. If you'd like expert guidance to embed these principles into your project culture — I offer consulting designed to drive adoption, reduce friction, and deliver outcomes)

Managing Requirements

Make sure following items are documented.

Business problem

- ✓ Define what problem that you are trying to solve?
- ✓ Who are your users?

Specifications from User Point of View

- ✓ Solution that you are going to implement
- ✓ Known Limitations of solution
- ✓ End user performance expectations
 - ✓ Memory
 - ✓ Execution time
- ✓ expected input data sizes
- ✓ Most frequent usage scenarios
- ✓ Assumptions (especially assumptions about the User)

Other Requirements

- ✓ Hardware and Cloud requirements
- ✓ Third party software requirements/dependencies
- ✓ Integrations (if any)
- ✓ Deployment scenario/specs (e.g. Cloud, desktop, SaaS, etc)

Make sure There is single spec document (as much as possible)

- ✓ change requests don't live in emails/chats etc.
- ✓ define who updates spec. doc.
- ✓ make sure every change request in updated in spec doc.
- ✓ Make sure spec documents are committed in same version control repo as code.



Managing and documenting Design/Architecture

What to document in design ??

- ✓ what are your targets for design and how you will take care of these design aspects
 - maintainability
 - o debuggability
 - o extendibility
- ✓ Design Intent
- ✓ assumptions (about end user, developers, expertise, deployment etc)

Technology choices

- ✓ Document Technology choices and their reasons for every technology, programming language, web framework, database etc.
- ✓ Reason cannot be "everyone is using it".

Which diagrams to use for documenting the design

- ✓ Module dependency diagram (Layer cake diagram). Make sure there are no circular dependencies between modules. This is MANDATORY DIAGRAM
- √ deployment diagram
- √ class diagram
- ✓ Sequence diagram or Swimlane diagrams. Avoid use of flow charts

Coding Practices

- ✓ Ensure 0 warnings at max warning level
- ✓ Enable compiler flags to "treat warnings as errors"
- ✓ Use Asserts
 - ✓ on average at least one assert every function
- ✓ Fix static analysis bugs and warnings first
- ✓ Write automated unit tests.
 - ✓ it should be possible for Any developer to run any unit tests.
 - ✓ Unit tests should be committed in the same repository as code.
- \checkmark ensure zero setup builds for developers. The process to introduce a new developer to the team should be as easy as.
 - ✓ checkout from Version control
 - ✓ open project in IDE (Visual Studio, VS Code etc)
 - ✓ Trigger the build
- ✓ Source code Is Your Best Documentation.
 - ✓ Write comments to critical functions. Write 'Why comments'
 - ✓ Use appropriate , clear names for classes, functions, variables etc.



QA and Testing Practices

- ✓ Prepare and document Test plans
- ✓ Identify
 - ✓ Acceptance tests
 - ✓ ensure customer 'formally' agrees with those Acceptance Tests
 - √ identify smoke tests/sanity tests.
 - √ identify regression tests
- ✓ Regularly update/add to the tests plans. Any time spec document is updated. Test plans should be reviewed and updated.
- ✓ Test plans are committed in the same repository as source code.
- ✓ Use docker/Virtual Machines for setting up test environment.
- \checkmark Testers pickup builds from build server ONLY. And not from developer's machine.

Build and Release Management

- ✓ Document the build process
 - ✓ Define version naming convention. Preferably use semantic versioning(http://semver.org)
 - ✓ Document branching and Tagging strategy for builds
 - ✓ Document daily build steps
 - ✓ Document QA Build steps
 - ✓ Document customer release build steps
 - ✓ Document Hotfixes process.
 - ✓ Document the deployment process.
- ✓ Document Customer Release procedure
 - ✓ Contents of Release Package
 - Release notes
 - new features
 - Known bugs
 - Known limitations
 - special installation/deployment/upgrade instructions
 - acceptance test results/test logs
 - √ How to Prepare installer/binaries or Docker
- ✓ Setup a Build server.
 - ✓ Automate Build and Release based on the build release process documents
- ✓ Prepare customer release and QA builds on Build server only



Development Tools and DevOps

- ✓ Build Server
 - ✓ Daily Build
 - Ensure Daily build is working. Escalate if Daily Build failing regularly
 - Run static analysis and code metrics
 - Run code duplication checks
 - Run automated unit and regression tests
 - ✓ Automate Release Builds
 - make installer or docker image
 - automatically tag
 - automatically update the semantic version
- ✓ SCM/Version Control
 - ✓ Commit Early/Commit often
 - ✓ Ensure no un-committed code on your machine at end of day
 - ✓ take update (git pull/svn update) at least twice in a day
 - ✓ use branching/merging as per the defined process.
 - ✓ Use 'feature branches' sparingly
 - ✓ use tagging

Project Management

- ✓ Define who 'Technical Owner' of the project is?
 - ✓ For all design/coding related issues, decision of 'technical owner' is final.
 - ✓ Project Manager will NOT override "Technical Owner's decision on this
 - ✓ Define who is responsible for updating the specs with any change requests from customer.
- ✓ QA Lead/Owner is empowered.
 - ✓ QA Lead/Owner must decide if the release should be given to customer or not.
 - ✓ Project manager or Technical Lead should not override the decision of QA Lead/owner.
- ✓ QA Reports the feature completion status and not the development team.
 - ✓ Based on acceptance tests that are passing
 - ✓ A Use case is DONE when the QA says it is DONE and not when Developer says it
 is DONE
- ✓ Project should be in 'shippable' state EVERY DAY.
 - ✓ There is no 'stabilization' phase or 'warning fixes' phase.