Agile Development Checklist

Manifesto for Agile Development: Individual interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan.

Agile Principles: Early and continuous release of software, welcome changes, business professionals and developers should work together, build around motivated individuals, face to face conversations, constant pace, attention to technical excellence and design, simplicity, self-organizing teams, reflection and debriefing.

Definition of Success: On-time, on budget, with all functions and features working as specified. Remember you usually only get two out of the three. Pick the two that are most important.

Mentorship: Important to find good mentors.

Tests: Write tests first.

Software cycle: Analysis, design, coding, testing, deployment. Occur simultaneously and rapidly in agile development.

Planning Game: Customer defines features, programmers provide estimates, customer prioritizes features.

On-site customers: Customers that are made easily available when programmers have a question. Ideally there should be two customers for every three programmers.

Test Driven Development: Testing, coding, design, and architecture that are all weaved together.

Deploy: Deploy something every week so that you can get quick feedback and adapt to customer response.

Product Manager: Maintains and promotes the product vision.

Domain Experts: Experts in their field.

Interaction Designers: Judge product solely based on the quality of the user experience.

Business Analyst: Refines customer needs into functional software requirements.

Testers & UI: Should stay one step ahead of programmers.

Usability Testing: Monthly testing of software with users.

Programmer Coach: Helps programmers with technical practices.

Refactoring: Changing the structure of code without changing its behavior or output. Refactor in smalls steps and use testing to insure that code behavior is not changed.

Technical Debt: Less than perfect design that “works” but requires time to cleanup, refactor, debug, improve, etc. Don’t forget to allocate time to pay down technical debt.

Timeboxing: Setting aside a specific amount of time for an activity so that the team or team members focus on that activity and stop, regardless of the progress made.

Last Responsible Moment: Delay decisions until the “moment at which failing to make a decision will cause harm or eliminates an important alternative.” Delaying gives you time to increase the amount of information you have and increases the likelihood you make a correct decision.

Stories: Self-contained elements of the projects, correspond to individual features. Criteria for story: 1. Can be completed in a week, usually half a day to 2.5 days, usually complete 2-5 per week per full-time person 2. Represents the customer value written in the customers’ terminology (customer-centric), not technical issues, 3. Describes the end result, not the implementation details, 4. Has clear completion criteria.

Iteration: Full cycle of design-code-verify-release.

Theory of Constraints: Every system has a single constraint that determines the overall throughput of the system.

Good Office Equipment: pairing stations, build machine, noise partitions, whiteboards, calendar, index cards, food.

Promise to Investors: We will make steady progress, finish the features you consider most valuable first, show you working software that reflects our progress every week, be honest and open with you about our success, challenges, and what we can reasonably provide.

Velocity: The number of story points you can complete in an iteration (usually one or two weeks, 5-10 working days). The more technical debt you have, the slower your velocity will be. Only count stories that are done done and never extend the iteration time.

Backlog: Features that have not made the current sprint.

Pair Programming: Programmers should pair together to increase quality, learning, velocity, etc. Includes a driver (details and code syntax) and a navigator (big picture strategy and quality control). Never assign partners, pairs should be fluid and shifting during the day.

Demo Progress: Make sure to demo your progress to team and stakeholders at least once a month.

Stop and Think: Biggest gains in productivity come from stopping and thinking about what you are doing.

Energized Work: Clear goals, collective responsibility, go home on time, stay home when sick, healthy snacks, frequent breaks, exercise.

Big Visible Charts: Display information so unambiguously that it can communicate even across the room.

Team Calendar: Shows important dates, releases, iteration numbers, time off, etc.

Gaming: When people get too focused on a number at the expense of overall progress.

Root Cause Analysis: Prevent mistakes by fixing process. Ask why 5 times to fix the process.

Retrospectives: Debriefing past work to find the good and bad. State that regardless of what we discover today, we understand and truly believe that everyone did the best job they could give their resources and information at hand. Brainstorming what went well and what went wrong. Put related cards together and separate unrelated cards. Vote on the top priorities. Discuss how to improve. See traction IDS.

Joint Responsibility: Train the team to say us and we, not I or me.

Eat together: Promote team unity by eating together.

Hustle: Things come to those that wait, but only the things left by those that hustle (A. Lincoln).

Avoid Overtime: Reliance on overtime indicates systemic problems.

Creative Alternatives: If stuff cannot be done in time find creative alternatives instead of trying to force the work.

Story Timing: The first 90% takes 10% of the time and the last 10% takes 90% of the time.

Osmotic Communication: Benefits from working together and overhearing each other’s conversations.

Interruptions: It takes a programmer 15 minutes to get back into flow after they have been interrupted. Chunk time to communicate with each other (first ten minutes of each hour) to avoid distractions.

Vertical Market Software: Software built for many organizations in one industry.

Horizontal Market Software: Software built for many organizations across a wide range of industries.

Speaking Language: Programmers should adapt their language to the domain experts, not the other way around. Use class names that the domain experts understand.

Daily Scrum: What did I do yesterday, what will I do today, what problems are preventing me from making progress? Should be 5-10 minutes.

Coding Standards: Development practices, tools, keybinding, IDE, file and directory layout, build conventions, error handling and assertions, approach to events and logging, design conventions, testing.

Iteration Demo: Agile teams produce working software every week. After demo ask is our work to date satisfactory, may we continue.

Source Code Length: The more lines of code a program has the more defects it will have.

Done done: Once you finish a story you should never have to come back to it. Make stories small so you can get them done done in a week.

No bugs: No bugs.

Version control: Keeping your source code in a single, authoritative place.

Repository: Master storage for all your files.

Sandbox: Working copy of code on the team members local machine, contains all files of repository from a specific point in time. Is not shared.

Check Out: To create a sandbox by checking out code from local repository.

Update: To update your sandbox by getting the latest changes from the repository.

Lock: Preventing anyone from editing a file but you.

Check in-commit: Checking in files from your sandbox to the repository.

Revert: Revert your sandbox by throwing away changes and returning to the point of your last update. Sometimes faster than debugging.

Tip or head: Part of the repository that contains the latest changes that have been checked in. When you update your sandbox you get the files at the tip.

Tag or label: Marking a particular time in the history of the repository, allowing you to easily access it again.

Roll back: Rolling back a check in to remove it from the tip of the repository.

Branch: When you split the repository into distinct alternative histories. You can edit files independently in on branch. Avoid to many branches or code customization and keep them short lived.

Merge: Process of combining multiple branches or changes and resolving any conflicts. The second person to check in a file has to merge the first person’s changes.

Concurrent Model: Allows two people to edit the same file simultaneously and version control automatically merges their changes. If two people edit the same code the version control prompt them to merge the two lines manually.

Time Travel: Going back in the repository to identify when a bug was introduced.

Binary Chop: Cutting the versions in half until you find the bug.

Spike Solutions: Code that is research based that you don’t plan to add to your source code. Keep a separate spike directory in source code.

.NET Build Tools; NAnt, MSBuild.

Build Script: Definition of how the code should be compiled.

Vision Planning: Reveals where a project is going, why it is going there, and what success looks like. Need a single vision. See page 202 of Agile Development.

Release Planning: roadmap for reaching you destination.

Planning Game: Expertise of the whole team to create achievable plans.

Risk Management Problems (& solutions): Employee turnover (positive cultures, promote good energy, no overtime, reasonable sprints, healthy environment, fair pay, company ownership), new requirements (prioritize tasks), time off (team calendar), work disruption (chunk communication, use slack, no popups), unknown technology (discover phase, spike solutions, mentors, community, slack), faulty technology (take care of technology, buy good equipment, slack), etc. Make sure to review all issues after a sprint to minimize future risks.

Iteration Planning: Structure for the teams daily activities. Stop at predetermined, unchangeable time intervals to compare reality to plan.

Slack: Allows the team padded time during an iteration to reliably deliver results every iteration.

Release early, release often: Helps with early cash flow, product feedback, etc.

Aggressive Deadlines: Extend schedules rather than reducing them.

Release Plan: Scopeboxed defines the features and leaves release date uncertain, timeboxed plan defines the release date but leaves the features uncertain. Timeboxed plans are better. To see a timeboxed plan, set a release date, define minimum marketable features.

Prioritizing Stories: Write all stories on an index card, programmers estimate time on the back of the card, customers prioritize which features should be completed first.

Kludge: An ill assorted collection of parts or a hack to get something done.

Constraint Presentation: When you cannot complete what the customer wants because of limitations of time, money or resources, give the customer two choices that are both doable and let them pick.

Batman: Deals with organizational emergencies and support requests so the other programmers can focus on programming. The batman does not work on stories or the iteration plan. Rotate a new programmer into the batman role.

Dedicated Research Time: Encourages learning and adds additional slack into your iteration.

Parkinson’s Law: Work expands to fill the time available.

Student Syndrome: Work is delayed until its deadline.

Reading Groups: Take turns reading sections of a book or an article out load and then discuss them.

Silver Stories: Less important stories that the team can set aside if they need additional team. Best practice is to instead work on technical debt instead of silver stories.

Improving Velocity: Pay down technical debt, improve customer involvement, support energized work, offload non-programmer activities to other people, provide needed resources and tools for greater efficiency, pair programmers. Be careful simply adding new programmer (read Mythical Man Month).

Cost Efficient Software: The best way to reduce the cost of writing software is to improve the internal quality of the code and design.

Practices for Clean Code: Incremental requirements (get started before customers work out all details), customer tests (to help communicate tricky domain rules), test-driven development, refactoring, simple design, incremental design architecture, spike solutions, performance optimization (using data to drive optimization), exploratory testing (identifying gas in the team’s thought process).

Requirement Storage: Put into version control instead of wiki or database.

Customer Tests: Tests to insure that tricky domain knowledge is working properly.

Test-Driven Development: Rapid cycle of automated testing, coding, and refactoring. Reduces the incidence of defects. Allows you to quickly refactor and improve design while instantly making sure that everything is still working correctly.

.NET Testing Tools: NUnit

Speed Matters: Users lose interest and switch tasks when a computer makes them wait for more than 10 seconds.

Unit Tests: Focus just on the class or method at hand.

Mock Objects: Substituting its own object for an object that talks to the outside world.

Integration Test: Test that talks to the outside world such as a database, or communicates across networks.

Divergent Change: When unrelated changes affect the same class. Occurs when your class has too many concepts and needs to be split.

Shotgun Surgery: When you have to modify multiple classes to change a single idea. Merge and create a single home for the classes.

Time Dependency: When a class’ methods must be called in a specific order.

Simple Code: communicates what you want to communicate, no duplicate code, fewest possible classes, fewest possible methods, appropriate for the intended audience, every idea is communicated to the audience, refactored.

CRC Cards: Class, Responsibility, Collaborator

Performance Story/Acceptance Tests: Need a customer valued goal, identifies how many operation should be completed in a given period of time, how much delay is acceptable between starting and completing a single operation, how much leeway is acceptable between starting an operation and getting feedback on that operation, acceptable performance, minimum satisfactory response, best possible performance, conditions under which story must perform.

Exploratory Testing: Testing code in untested conditions. Take good notes, test boundaries of data types or limitations, test none, some, all, test create, read, update, delete.

Courage: To make the right decisions, even when they are difficult, and to tell stakeholder’s the truth when they need to hear it.