Agile

Alexander Zerpa

Section 1

Agile

Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility. 10.
10. Simplicity (the art of maximizing work not done) is essential.
11. The best architecture, requirements, and designs emerge from self-organized teams.
12. At regular intervals, the team reflects on how to become more efficient, then tunes and adjust its behavior accordingly.

Development Practices

Planing

- Choose tools only when necessary.
- Start with simple and free tools.
- Consider cost vs gain when deciding on Third-Party libraries.
- Short document for project standards and decisions.
- Big picture concepts not nitty-gritty details.
- Not document unless necessary.
- Light but maintain documentation.

Design

- Modularity.
- Flexibility.
- Avoid premature optimisation.

Not solving problems before they happen, but having a foundation that is adaptable enough to handle new problems and features as they come.

Velocity

How much work a team can do over a period of time.

- Baseline at 30% of sprint duration.
- Change load according to performance.
- Tends to stabilize after 6 sprints.

Section 2

Scrum

Scrum

Lightweight framework that generates value through adaptive solutions for complex problems.

Pillars

- Transparency
- Inspection
- Adaptation

Values

- Focus
- Respect
- Openness
- Courage
- Commitment

Subsection 1

Roles

Product Owner

Accountable for effective Product Backlog management.

- Developing and explicitly communicating the Product Goal.
- Creating and clearly communicating Product Backlog items.
- Ordering Product Backlog items.
- Ensuring that the Product Backlog is transparent, visible and understood.

Developers

Create any aspect of usable Increment.

- Creating a plan for the Sprint, the Sprint Backlog.
- Instilling quality by adhering to a Definition of Done.
- Adapting their plan each day towards the Sprint Goal.
- Holding each other accountable.

Scrum Master

Establish and manage Scrum framework.

Serves the Scrum Team

- Coaching the team members.
- Helping the Scrum Team focus on creating high-value Increments.
- Removing impediments to the Scrum Team's progress.
- Ensuring that all Scrum events take place.

Serves the Product Owner

- Finding techniques for effective Product Goal definition and Product Backlog management.
- Helping the Scrum Team understand the need for clear and concise Product Backlog items.
- Establishing empirical product planning for a complex environment.
- Facilitating stakeholder collaboration as requested or needed.

Serves the organization

- Leading, training, and coaching the organization in Scrum adoption.
- Planning and advising Scrum implementations within the organization.
- Helping employees and stakeholders understand and enact Scrum Methodologies
- Removing barriers between stakeholders and Scrum Teams.

Subsection 2

Events

Sprint

Fixed length event of one month or less within happens all the work necessary to achieve the Product Goal.

- No changes are are made that would endanger the Sprint Goal.
- Quality does not decrease.
- The Product Backlog is refined as needed.
- Scope may be clarified and renegotiated with the Product Owner.

Only Product Owner can cancel the Sprint if Sprint Goal becomes obsolete.

Sprint Planning

Scrum Team defines what they will work on and how to execute the plan.

Daily Scrum

Inspect progress and adapt as necessary.

Sprint Review

Inspect the outcome of the Sprint and determine future adaptations.

Sprint Retrospective

Plan ways to increase quality and effectiveness.

Subsection 3

Artifacts

Artifacts

Product Backlog

- To-do list for the scrum team.
- Single source of truth for requirements.
- May include ideas.

Sprint Backlog

- Plan composed of the Sprint Goal (why).
- The set of Product Backlog items selected for the Sprint (what).
- actionable plan for delivering the Increment (how).

Increment

- Concrete stepping stone towards the Product Goal.
- Product produce at the end of a spring.
- Functional software that meets the definition of done.

Working with Backlogs

Backlog Items

- Items closer to the top are more fine-grained and implementable.
- Items closer to the bottom are more general.
- Items tend to have title, description, order and estimates.

Definition of Done

Criteria that define what is meant what an artifact is considered done.

Backlog Refinement/Grooming

- Add more details to product backlog items.
- Clarify requirements.
- Revisit priorities.
- Add or remove items from backlog.
- Backlog item decomposition.
- Estimation.

Section 3

Extreme Programming

Subsection 1

User Stories

User Stories

Express product requirements.

Parts

- User category or role.
- Wanted functionality of the system.
- Reason for the functionality.

Acceptance Criteria

Define the criteria that must be met.

Example

```
Title: ________________

Opening sentence:
   As a ________
   I want _______
   so that _______

Additional detail:

_______________________

Acceptance criteria:
- ________________
- ________________
- ________________
```

Estimates

Apply to Backlog items.

Relative Units

Estimate task relative to others.

- Fibonacci
- Exponential
- T-shirt sizes

An estimation can be infinite if its to large to be estimated or 0 if its already done or trivial.

Planning Poker

Practice for estimation in which every developer reveals estimation at the same time, outliers clarify estimation, and then repeat until a consensus is reached.

Grouping User Stories

Epic

Represents a workflow or process to large to be estimated, as such tends to be comprised of user stories or other epics.

Themes

Group user stories, don't need to relate to the same epic.

Subsection 2

Feedback

Release Planning

Large scale feedback. Planning the next few months of work and product deliveries.

- Get aligned with each other.
- Identify what is possible.
- Remove obstacles.
- Organize the work activities.
- Commit with each other as a team.

Iterations

Medium scale feedback. Planning in weeks. Same as Sprint in Scrum.

- Begins with iteration planning.
- Continues with daily work.
- Ends with feedback.

Iteration Planing

Same as Sprint planning.

- Select user stories.
- Stories to task decomposition.
- Add or clarify information.
- Explicit agreement.

Stand-Up Meeting

Daily feedback cycle. Same as Daily Scrum.

- Every 24 hours.
- Tool for self-management.
- Brief.
- Team members typically stand up.
- Same place, Same time.

Three Questions

1. What did we get done?
2. What do we plan to get done?
3. Is there an impediment?

Subsection 3

Practices

Test-Driven Development

Automated test first, write code second.

- Write only necessary code.
- Naturally encourages good OOP practices and clean code.
- Functional test suite that grows with the production code.

Acceptance Test

Validates implementation of user story.

Unit Test

Single focus small element of the behavior of a story.

Pair Programming

- Fewer errors.
- Increases the level of competence and knowledge of every member.
- Shared credit over code.

Driver-Navigator Technique

Short time boxes. Switch roles frequently.

Driver

Controls the keyboard and mouse.

Navigator

Guides the driver.

Promiscuous Pairing

Change pair programming partners.

Mob Programming

Everyone works together.

Subsection 4

Continuous Process

Continuous Process

- Improve skills.
- Fewer mistakes.
- Becomes easier.
- Small amount of work.

Design Improvement: Refactoring

Improving the internal structure of existing code without changing its external behavior.

Continuous Integration

Merging, building, and testing several times per day.

- Merge everyone's code together.
- Automated build and test.
- Fast feedback.
- Keeps the product in a releasable state.

Small Releases

Deliver small increments of working software frequently.

Section 4

Kanban

Philosophy

Lean Thinking

- Eliminate waste.
- Amplify learning.
- Decide as late as possible.
- Deliver as fast as possible.
- Empower the team.
- Build integrity in.
- See the whole.

Kanban Principles

- Start with what you do now.
- Agree to pursue incremental, evolutionary change.
- Initially, respect current roles, responsibilities, and job titles.
- Encourage acts of leadership at every level.

Core Practices

- Visualize.
- Limit work in progress.
- Manage flow.
- Make policies explicit.
- Implement feedback loops.
- Improve collaboratively, evolve exponentially.

Practice

Work process management methodology based on Visual boards and cards to track work.

Basic Board

- 3 columns (to-do, doing, done).
- Everything start in the left most column.
- Items are move right to de board.

Step Subdivision

To differentiate completed work on a step we subdivide it into not done and a done category

Limit Work in Progress (WIP)

Team-based decision on the realistic amount of work that can be done.
Find bottlenecks or steps with overburden and set a hard work-item limit.
WIP Limit $=$ Team members $+$ Buffer

Calculate Setp WIP

1. Find slowest step.
2. Calculate WIP for that step.
3. Adjust WIP of other steps proportionally.

Project Visualization

Workflow Map

Break down process into development steps.

Cumulative Flow Diagram

Graph how work items are distributed among Kanban board columns over time.
A stable system produces a CFD that is steadily rising.

Section 5

Legacy Code and Technical Debt

Subsection 1

Set-up

Review Technical Debt

Places to Find Information

- README and other text files.
- Project wiki.
- Issue tracker.
- Same as build and run.
- Test directory.
- Same as build and run.
- Project website.
- Embedded help documentation.
- Source code commit history.
- Deployment scripts.

Dev Environment

Use Tools like Docker and Docker Compose to provide an isolated development environment.

Subsection 2

Testing

Automated Test

Test allows us to make changes to the code without worrying to much about unintended consequences.

Create a Test Suite (if one is missing)

- Start with just one test.
- Pick testing framework.
- Write a simple failing test to see how to use the framework.
- Replace the failing test with something simple about the application.

Run the Test Suite

If no documentation try using IDE test options or finding documentation about framework used.

Failing Test

When in doubt, assume the code under test is correct.

- Verify dependency versions.
- Attempt to change the test to make it pass.
- Delete the test

Build our Test Coverage

Approval Test

Compare current results against a set that are known to be good.

Dependency Behavior Tests

- Validate the behavior of a dependent library.
- Don't need to run often, just when dependencies are updated.
- Switching to a library with similar features.
- Build out a facade between project and dependency.

Subsection 3

Tool Improvement

Tool Improvement

Aging Tools

- Old language versions.
- Unsupported frameworks.
- Complicated dev environment setup.
- Limited automated testing.

Dependency Management

- Specify dependencies and their versions.
- Dependency acquisition is handled by script.
- Ats as a form of executable documentation.
- Tracks downstream dependencies.
- Can detect incompatibilities.

Update Dependencies and Tools

Before Upgrading

- Build test coverage.
- Adopt dependency management system.
- Latest IDE version.

Upgrade Procedure

- One dependency/version at a time.
- Run test before and after each upgrade.
- Check in progress to source control.
- Leave multi-dependencies upgrade to last.
- Migrate unmaintained dependencies.

Subsection 4

Pay Down Technical Debt

Where to Start

Read by Refactoring

Apply small refactoring while reading code with the goal of gaining a better understanding.

Noise and Distractions

Commented out code.

- Always delete.

Poorly formatted code.

- Formatting automatically.
- Avoid mixing logic and formatting changes.

Compiler Warnings.

- Don't address all at once.
- Address as you change files.

Runtime Warnings

- Can include deprecation messages.

External Services and Devices

Facade

Handles all communication and is the only part of a project that talks directly to the service or device.

Facade Pattern

```
.-------------------.
|      System       |
'-------------------'
         ||
.-------------------.
|      Facade       |
'-------------------'
         ||
.-------------------.
| Service or Device |
'-------------------'
```

Simulated Service or Device

```
.----------------------------.
|           System           |
'----------------------------'
              ||
.----------------------------.
|           Facade           |
'----------------------------'
              ||
.----------------------------.
| Simulated Service or Device |
'----------------------------'
```

Simulated System

```
.-------------------.
| Sumulated System  |
'-------------------'
         ||
.-------------------.
|      Facade       |
'-------------------'
         ||
.-------------------.
| Service or Device |
'-------------------'
```

Scientist Library

- Runs experiments in production.
- Designed to test a refactoring against unmodified version.
- Original code is always run.
- Experimental replacement is sometimes run.
- Results compared.
- Differences logged.
- Protections against exceptions.

Subsection 5

Keep Technical Debt at Bay

Keep Technical Debt at Bay

- Test-Driven Development.
- Refactoring.
- Clean Code.
- Monitor Code Quality.