Agile Management Plan

Ian Dennis Miller

2015-06-23

The Agile Management Plan is a pattern for how to run an Agile project. This document contains information about the conventions and processes that will be used to get this project done.

Agile Bootstrap

Every project must first be bootstrapped. To get a project started, the following documents need to be updated to ensure they are applicable to the project that will be undertaken. The Checklist can then be followed to ensure the rest of the steps happen.

Agile Documents

0. **Checklist** - all of the steps required to start and finish this project.
1. **Agile Management Plan** - the way this project is approached (i.e. this document).
2. **Team Resources Diagram** - an overview of the technical resources that are available for the team to use in completing this project.
3. **Project Guide** - the master document; an overview of the documentation related to this project.

Communication I

Most projects involve distributed team members, so this project involves communication tools that are suited to online collaboration.

Routine communication

- The project is documented with a wiki.
- The master project document is the Project Guide, which links all documentation.
- Project artifacts (e.g. images, docs) are stored in the wiki.
- Team members send direct messages through the team chat room.
- The blog communicates key project updates (e.g. meetings, sprints).
- The team calendar is used to let others know when you will be unavailable.

Communication II

Work communication

- The online Agile Board displays the Story Backlog and the current sprint.
- Source code is shared using a distributed version control system.
- The version control system is accessible via web interface.
- Continuous integration results are pushed to the team.

Meetings

- online group meetings are held using Google Hangouts
- online group presentations are performed with join.me
- IRL group meetings take place at a library meeting room or university cafeteria

Planning

External forces, such as time, budget, business requirements, and customer requirements, influence the project. These planning documents are concerned with modeling the project environment in order to communicate the vision to all the stakeholders.

0. **Charter** - this document initiates the project, articulates a vision, and determines scope.
1. **Timeline** - deliverables and releases are planned in the timeline.
2. **Organization View** - a diagram of project stakeholders and their relationships.
3. **Story Planning** - the groundwork for eliciting stories and organizing them into releases.
4. **Stories** - the backlog of stories.

Platform

- Python is a pretty good language for agile software development.
- The MVC architecture is built on top of Flask-Diamond.
- The completed software project installs to a UNIX-like system (BSD, Linux).
- The software typically runs as a non-root system daemon.
- Multiple systems are required for development and deployment:
  - development host for routine integration testing.
  - staging host for IT Ops to integrate in a clone of production environment.
  - production host for live operations.

Design I

The design process is captured in a series of documents that help to communicate design ideas to the rest of the team. It is not always necessary to use each of these documents for every project. These documents may never be complete, so it is good enough to just get them started.
A Model-View-Controller architecture is designed in terms of these documents. The ERD corresponds to the **Model**, the Wireframes correspond to the **View**, and the Process Map correspond to the **Controller**.

Design Documents

0. **Wireframes** - a mock-up of the screens in a user interface
1. **Site Map** - a diagram indicating which screens are connected to which
2. **Entity Relationship Diagram** - the things we are modeling and how they relate to one another
3. **Process Map** - a sequence of actions taken by users in our system
4. **System Map** - the users, networks, servers, and processes involved in this project

Specifications

Specifications are derived from design documentation. As with design, these specifications correspond to the Model-View-Controller architecture.

0. **Data Model Specification** - extended information about the models: attributes, data types, field description, usage notes, etc.
1. **Functional Specification** - each screen in the wireframe has a short narrative here in the functional specs.
2. **Technical Specification** - each step in the process map corresponds to a longer description here in the technical specs. If the project includes an API, it is defined here.

Working

- A story contains a unit of business functionality, and it takes work to complete a story.
- The project is completed largely by writing code.
- Story complexity is estimated in terms of points from the Fibonacci sequence: 1, 2, 3, 5, 8, 13, 21, 34.
- The stories are grouped into themes, which in turn become releases.
- During *Agile Sprints* of 1-2 weeks, we work on a certain number of stories at a time.
- The Agile Board indicates which stories are todo, in progress, and done.
- A story is done when its tests have been written and the tests are passing.

Source Code

- Whenever the Python language is used, the code style should meet PEP8.
- We use Git for controlling our source code.
- We generally follow the Git Flow branching pattern.
  - The *master* branch is stable and potentially releasable.
  - The *develop* branch is where completed work is initially integrated.
  - Each story has its own branch, which is merged from develop and after completion is merged back into develop.
- Branches are not especially important, once they have been merged, and so they may be deleted whenever it is convenient.

Testing I

- Automated tests are created directly in the project.
- A story, which provides a 'description of done,' is used as a primary source for test creation.
- When possible, tests are written before the feature is implemented.
- The project supports multiple testing methods during development
  - `make test` - run all of the tests
  - `make single` - run only tests marked with the 'single' attribute
  - `make watch` - watch for changes to the source code and repeatedly invoke 'make single'
- Integration tests, which have external dependencies, are executed by the Continuous Integration server.
- Any time code is committed to the version control repository, the Continuous Integration server executes Unit Tests.

Testing II

- Each time code is committed to the develop or master branch, the Continuous Integration server executes Integration Tests in addition to Unit Tests.
- Tests are assigned numbers, which are stored in the test code comment header.

Releases

- Each release is tagged with a version number.
- A major **X.0.0** release breaks backwards compatibility.
- A minor **0.X.0** release adds features but does not break compatibility.
- A bugfix **0.0.X** release corrects the behavior of a feature.

Release process

0. Ensure integration tests are passing in the develop branch.
1. Merge from develop to master and ensure integration tests pass in the master branch.
2. Add a new version tag to the master branch and push the tag to the repository.
3. Notify IT Ops that a new release is available.

Documentation

- The master document is the Project Guide.
- Markdown format is used for documents stored in version control.
- HTML is used for documents in the wiki.
- Code documentation is created with Python Sphinx, which uses the ReST language.
- Design documentation is shared via PDF or PNG format.
- Collaborative design work is accomplished with an online diagram editor.

Integration

Projects are integrated into the production environment for release to users. The release cycle describes the interaction between development and integration. The following documents describe IT Ops tasks during the release cycle.

0. **Installation** - the process for setting up an environment and installing the project inside it.
1. **Upgrading** - the process for getting the latest source code and applying the update.