**Epic Progress Report assignment EPR2 – Creating and implementing Tasks**

Assignment authored by Prof. Grant Braught,
with edits by John MacCormick

**Introduction:**

In EPR1 you wrote an epic and broke that epic into user stories. Often user stories are still too large for individual work sessions. So, user stories can be broken down into tasks to be completed. While each user story must provide identifiable value to the product owner and may take multiple developers to implement, a task is focused on the development work necessary to implement that story. A task should be completable by an individual. In this activity you will revise your user stories (if necessary), break a few of them into tasks, and then begin implementation work on the tasks.

A note on timing: This assignment is spread over two weeks. After the first week, your team will have a standup meeting as described below during our class meeting. Your team will continue implementing tasks and identifying new ones as appropriate throughout the second week and will have a second standup meeting in class on the day the assignment is due.

**Update/Revise Stories:**

You wrote and revised your user stories in the last activity. Since then, you have had the opportunity to talk with the client, Matt Steiman. That conversation should have provided additional clarity and insight into the client’s vision for the FarmData2 application.

1. Based on the interaction with the client, you may need to revise your user stories further. If so, take a few minutes to revise and update your user stories as necessary on your team’s Kanban board in GitHub.

**Defining Tasks:**

As mentioned earlier, while epics and user stories express value from the user’s perspective, tasks are written for developers. They decompose a user story into individual pieces to be built that collectively provide the functionality of the user story. Another distinction is that user stories are often designed to have several people working on them simultaneously, whereas tasks are typically targeted at individuals. That said, you should not hesitate to work on tasks in pairs or as a full team if desired. A final distinction worth mentioning is that whereas user stories aim to provide a full vertical slice of functionality, a task need not do so. The tasks could decompose the story into smaller horizontal pieces (e.g. UI, Vue.js implementation, API call, etc.).

2. The following two short pieces do a good job of clarifying the difference between stories and tasks and the types of things that should be in tasks. Read them before continuing:

* “The Difference Between a Story and a Task” by Mike Cohn of Mountain Goat Software.
	+ <https://www.mountaingoatsoftware.com/blog/the-difference-between-a-story-and-a-task>
* “How detailed should tasks within a user story be for agile teams?” By Yvette Francino who is an Agile consultant.
	+ <https://techbeacon.com/app-dev-testing/how-detailed-should-tasks-within-user-story-be-agile-teams>

You should of course feel free to read any other sources that you find for additional information about stories, tasks and the process of breaking stories into tasks. There are about as many opinions on what these are and how to do this as there are Agile developers.

**Breaking a Story into Tasks:**

As you may have gathered from the above articles, the exact right size for a task is a matter of debate. But there is some consensus that they should be of a size you can complete in a working session. In a full-time job, that is commonly a half day or a full day. For us, we’ll try to break user stories into tasks that could be completed in an hour or two of work. Another way of thinking about this is: define your tasks so that you can complete 2 or 3 tasks per week. (Where did this number come from? Remember, you expect to spend about four hours per week working on COMP290 outside of class meeting time. If each task took exactly 1.5 hours, you could complete $4÷1.5=2.67$ tasks per week, which yields 2 or 3 per week in round numbers.)

3. **Individually**, each team member should break one user story down into tasks. Each task should have a Label, a Narrative describing what needs to be done and Completion Criteria. Keep in mind that the tasks are written for the developers, so they need not use the “As a… I want to… so that I can…” format for their Narrative.

The completion criteria for a task should be things for which automated tests could be created. Think of these as actions you could take in the browser and what you should see. For example: “When the page loads, the BarnKit has a sub-tab for the Harvest Report,” or “When the Generate Report button is clicked, a table with fake results is displayed.”

Place the cards for your tasks into the Tasks column on your team’s Kanban board. A convenient way to associate tasks to stories is to use an emoji in the card. For example:



Experiment with emojis such as “:red\_circle:” “:green\_circle:” “:large\_blue\_diamond:” and/or search online for GitHub markdown emojis.

**Scheduling Your Work:**

To work well as a team you will need to have times to both (i) coordinate and synchronize, and (ii) work independently (or in pairs). The following workflow is recommended but you may use an alternative strategy if desired.

4. **As a Team:** Set a weekly meeting time shortly after our class meeting (certainly no later than Friday). At this meeting you will define and schedule the work that you want to complete for the week:

a. Pick a user story to work on.

b. Select tasks that you plan to work on and move them to the Scheduled column. You should place into the scheduled column only tasks you think your team can complete in the current week.

c. Determine if there are dependencies between the tasks (i.e. one has to be done before the others) and make necessary adjustments. Move tasks between the Tasks and Scheduled columns so that they can be completed in the necessary order.

d. Decide who will work on which tasks and when. You can work on the tasks individually, in pairs or as a full team, whatever works best for your styles and the needs of the task.

5. **As a team, in pairs or individually**: Work on the tasks through the week.

a. When you start work on a task, move it to the In Progress column.

b. Call impromptu team meetings and have conversations with teammates and the instructor as necessary.

c. Move the card for any completed task to the Done column and start on a new task by moving it from Scheduled to In Progress.

6. **As a team:** Set a weekly meeting time shortly before the next class meeting. At this meeting you will:

 i. Review what has been accomplished.

ii. Work on merging any independently implemented parts (see Branching Workflow below).

 iii. Enumerate what is still to be done.

iv. Review and revise your stories or tasks to ensure they are up to date and accurate with your current understanding of the epic.

v. Identify any obstacles that are preventing progress and strategize as a team about how to get past them.

vi. Lightly prepare for the Standup meeting at the start of the next class (see Standup Meetings below).

**Branching Workflow for a Team:**

To work on a user story you should use a branching workflow.

7. Using one team member’s clone, create a feature branch for the story you are working on (we’ll call this the *Story Branch*) and push it to the origin repository. All other team members should then pull that branch from the origin.

8. Create a preliminary pull request to the upstream repository for your Story Branch. Be sure to include a description of the functionality that the PR will provide and link your pull request to the issue describing your Epic.

9. Each time you begin working on a task, create a feature branch (we’ll call it a *Task Branch*) from the Story Branch in the clone where you are working.

10. Implement the task in the Task Branch.

11. When the task implementation is complete and meets all of the completion criteria, commit the changes to your Task Branch. See the Commit Messages section below for more information.

12. Synchronize your Task Branch with the Story Branch.

 i. Pull the Story Branch

 ii. Merge the Story Branch into your Task Branch (resolving any conflicts).

 iii. Push your Task Branch to the origin.

iv. Create a Pull Request to the origin (your team’s GitHub) to merge the Task Branch into the Story Branch.

 v. Merge the Pull Request into the Story Branch on the Origin.

 vi. Everyone should pull the new Story Branch and merge it into any open Task Branches.

**Commit Messages:**

When creating commits, it is important to write good descriptive commit messages. The article below gives an excellent review of how to write useful commit messages:

* “On Commit Messages” by Who-T
	+ <http://who-t.blogspot.com/2009/12/on-commit-messages.html>

Also, if you have worked in a pair or as a full team on a particular task then you will want to be sure that everyone receives credit for that commit. To do so, be sure that you:

* Create a Commit with Multiple Authors
	+ <https://docs.github.com/en/github/committing-changes-to-your-project/creating-a-commit-with-multiple-authors>

**Standup:**

A standup meeting is a very brief summary of what you have done since the last meeting and an enumeration of any of the challenges or questions that you have going forward. Each team will be given 5 minutes for their standup. You may share your Kanban board during your time. Note that this is not the time to discuss deep technical details or code. However, if you are having a problem getting a piece of code to work, this would be the time to identify that as a challenge. The intention here is to get your information out there so that others in the class and the instructor can get in touch with you with suggestions and/or point you to resources that may be helpful.

**Moodle submission:**

Before the deadline for this assignment, submit to Moodle a progress report of 100-200 words, describing your personal contribution to the team’s progress on its Epic since EPR1 was completed. In your report, make sure to answer the following questions.

* Which kanban Tasks did you work on?
* Which kanban Tasks did you complete?
* How many commits to GitHub did you make?

In addition, feel free to include any other information that provides convincing evidence you have been devoting the required amount of effort to the Epic project (about four hours per week).