**SOFTWARE ENGINEERING G6046**

**Agile process model**

**Guidance for organising and documenting a sprint cycle**

**Summary**

For your group coursework, the development process should follow an Agile-like process model. We will discuss the nature of Agile in lecture classes. There is no “gold standard” or “standard methodology” for doing a project the Agile way. Every organisation has its own flavour that suits its business. But we can draw upon the core ideas to create a process model that is suitable for this group coursework activity. Just because we are embracing Agile, there is no excuse for zero or poor quality documentation. We need to develop an appropriate type of documentation standard. The guidance document sets out what should be documented for each intensive iteration of development work (“sprint”). You can adapt the template given in this document, as long as you can justify your reasons, and document them convincingly in your final submission.

Remember that the group coursework is as much about the final codebase you produce as it is the journey that gets your there. There is a strong emphasis on team working, and discovering what works and what does not.

**The Agile process model.**

Agile places an emphasis on creativity over complex documentation, but that does not mean zero documentation. It is an iterative process model that typically sees development over a series of sprints, until the product is seen to meet the needs of the customer. Remember that:

* A sprint is typically of fixed duration. In industry 2-4 weeks is not unusual. For this module, 1-2 weeks is going to be more appropriate.
* A sprint generally involves focussed working on a problem, to the exclusion of all else. In industry a Scrum master is responsible for isolating the team from external distractions. For this module, this not practical as you have other modules and coursework to work on.
* Each sprint should give rise to a working prototype. Code delivered at the end of a sprint that does not work, or does not deliver on the objectives of the sprint has no real value. The objectives of the sprint should be achievable within the sprint. You can always re-plan so that an objective is moved to a later sprint.
* At the end of each sprint, the customer would provide feedback on what has been achieved and this would be used to inform the next sprint. In industry, this engagement with the customer is critical to making Agile work. You can get customer feedback by showing your work at a seminar session. You do not have to do this, but getting feedback is useful. You can use the user requirements document you have been given, and the updates you receive from the customer via the Canvas discussions to assess whether your sprint has met its objectives.

So a typical sprint runs like this:

1. Determine the “user stories” that are to be addressed. This can take the form of a user story narrative, and/or a set of “task cards” that describe some useful part of the overall problem that you intend to solve. You should be able to show that the stories or cards reflect what the customer has asked for in their user requirements document. If the customer did not ask for it, why would you be developing it? Consider prioritising task cards. This may reflect the significance of the task, or how complex you think it may be to achieve.
2. Determine whether there are any specific system requirements (functional, non-functional and domain requirements) that naturally emerge from your user stories or task cards. They do not need to be exhaustive, but they will focus your mind on exactly what needs to be done, and what needs to be tested.
3. Assign team members and resources to deliver the sprint. Some simple project planning is appropriate here. Consider any key risk factors and what strategies you have in place for them.
4. Make any design decisions that are essential for the delivery of the sprint. This can take the form of class analysis (for OO implementations), system or other high level diagrams, UML diagrams, including although not limited to, use cases, sequence diagrams and state diagrams. You only need to do sufficient design to support the sprint objectives.
5. Develop a test plan. Once you run your sprint, you should test your prototype. This will likely require some mix of automated unit testing and human-run system testing. To figure out what needs to be tested, look at any specific system requirements you set out for the sprint.
6. Get feedback on your work. Compare what you have done with the user stories and task cards. If you need more feedback, show your work to the customer.
7. Set your priorities for the next sprint, using knowledge and experience gained from your last sprint. The customer can provide help here – just email them if you need to.

Each sprint cycle needs to be documented to provide the evidence necessary for your final submission. The documentation is more lightweight than would be the case for a Waterfall development process model. You may find it helpful to have one person in your team co-ordinating the documentation process. You should keep backups of your documentation and look after it with the same care you should afford your codebase. A working program with poor quality documentation will not grade very well.

You can organise as many sprints as you think appropriate. But each sprint will need to be documented, so that markers can get a clear sense of the progression of your group throughout this coursework activity. It is likely that most groups will have between 4-6 sprints. Some might have more, it’s really down to you to decide as a group how you will make this process work.

**Sprint cycle template documentation model.**

Use the template at **Appendix A** as a basic model for what the documentation should look like for each sprint. You can adapt this documentation model if you think appropriate. You should be prepared to justify any such changes in your final submission.

Kingsley Sage

[Khs20@sussex.ac.uk](mailto:Khs20@sussex.ac.uk)

February 2020

**APPENDIX A: SPRINT DOCUMENTATION TEMPLATE**

|  |  |
| --- | --- |
| 1. **Summary data** | |
| Team number |  |
| Sprint technical lead(s) |  |
| Sprint start date |  |
| Sprint end date |  |

*The technical lead may vary from one sprint to the next. This is down to how you collectively organise your team.*

|  |  |
| --- | --- |
| 1. **Individual key contributions** | |
| **Team member** | **Key contribution(s)** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

*This data should help you to agree your peer assessment at the end of the project. If there is a dispute over your peer assessment, the markers will refer to this section as evidence to support a final decision.*

|  |
| --- |
| 1. **User stories / task cards** |
| *Provide text descriptions of any user stories or task cards you have selected for this sprint. These should naturally emerge from the user requirements document and discussion on Canvas. If you produce task cards, they should show the relative priority of the task for this sprint.* |

|  |
| --- |
| 1. **Requirements analysis** |
| *For the user stories/task cards selected, set out what key functional, non-functional and domain requirements you have identified. Remember that functional and non-functional requirements can be further categorised as mandatory (“shall”) and desirable (“should”). You can use free text descriptions or tabular formats. Remember that domain requirements cannot be acted upon directly. They require domain expertise to refine them into meaningful functional and non-functional requirements. All requirements should be SMART (Specific, Measurable, Achievable, Realistic and Time-Bounded). The requirements analysis does not need to be exhaustive, but should focus on things that are important for this sprint. They should also form a basis for testing.* |

|  |
| --- |
| 1. **Design** |
| *Remember that you only need to do enough design to support the objectives of the sprint. For teams working with OO implementation languages (likely most of you), this would include a class diagram. You may find it useful to develop simple Application Programming Interfaces (APIs) for key classes. This will focus your attention on what each class needs to make available for other classes to use. It also supports good documentation practice and helps coders work together.* |

|  |
| --- |
| 1. **Test plan and evidence of testing** |
| *You should consider:*   * *Unit/component level testing – typically achieved using automated test procedures such as Junit in Java. This level of testing demonstrates that individual classes are working as you intend.* * *System level testing – typically a human lead and documented test process that shows the prototype working as a whole entity.*   *Testing should show that the requirements you set out are being delivered on. They provide a means of showing that we have delivered what the user stores and task cards set out. Remember to identify a useful set of boundary test conditions.*  *Evidence of testing should demonstrate that the prototype achieved has been tested according to the test plan. If there are deficiencies, then these should be documented, as they will need further work in a subsequent sprint.* |

|  |
| --- |
| 1. **Summary of sprint** |
| *You should consider and discuss:*   * *Did you achieve you objectives for this sprint?* * *Is there a working prototype?* * *What went well, and what did not go well? If things did not go well, what have you learned and what will you do differently for the next sprint?* * *Is there any feedback from the customer?* |