**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 | 2 |
| Sprint technical lead(s) | Jonathan Benjamin Morris |
| Sprint start date | 27/04/2020 |
| Sprint end date | 14/05/2020 |

*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)** |
| Alex H. | Parsing JSON files for the two card decks, |
| Ben M. | Implementation of Cards, adding to player movement, Javadoc of: card, board, cardDraw, GameMaster |
| Joe C. | Frontend development |
| Joe L. | Testing documents and Junit files |
| Tom M. | Next turn and Roll Dice sprites |
|  |  |

*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) The game is for 2-6 players. Each player is assigned one of the game tokens. The tokens are: boot, smartphone, goblet, hatstand, cat and spoon. Each player takes a turn by rolling two dice to determine how they move around the board. At the outset, all players start on the board space labelled Go and move clockwise around the board.  2) At the outset of the game, each player has £1,500 in cash. One player is designated the banker and is responsible for distributing the correct amount of cash to each player. The bank has a total of £50,000 cash. Players may not borrow additional money from the bank, but they can trade game items with the bank.  3) At the outset of the game, the two packs of cards labelled “pot luck” or “opportunity knocks” are shuffled and placed on the board. When cards are taken, they must be replaced at the bottom of the corresponding pile.  4) For each turn, the player rolls the two dice. They move the number of spaces shown on the dice and arrive at a board space. Players move clockwise around the board.  5) If a player throws a double, then they take another turn. If a player throws another double at the third turn, then they “go to jail”. When a player goes to jail, they go directly and do not pass Go.  6) Board spaces may consist of properties, a “pot luck” space, an “opportunity knocks” space, “free parking”, the jail/just visiting space or a space with specific instructions that must be followed by the player.  7) If a player lands on a “pot luck” or “opportunity knocks” space, they take a card for the top of the corresponding pile and carry out the instructions on the card. When this is complete, the card is replaced at the bottom of the corresponding pile.  8) Players make progress in the game by buying property as they move around the board. Players may not purchase property until they have completed one complete circuit of the board by passing the Go space. When a player passes Go, they receive £200 from the bank.  9) All properties are initially the property of the bank. When a player purchases a property, the card is transferred from the bank to that player and the amount shown on the card is paid to the bank.  10) Once a player has made their move, if they land on a property that has not yet been purchased, they have the opportunity to buy that property. If they decide not to buy that property then the property is auctioned by the bank. Each player makes a bid to the bank. The bank sells the property to the highest bidder. If there are no bids, then the property remains unsold. All bidding players must have completed one circuit of the board.  11) If a player lands on a property owned by another player, they must pay the player who owns the property the value of the rent shown on the card.  12) If a player owns all of the properties in a colour coded group, but the properties are otherwise not developed further with houses and hotels, then the rent due is doubled.  13) If a property is improved with houses or hotels, then the rent to be paid is as shown on the card.  14) All rents must be paid for in cash. If a player is unable to pay the rent for a property they have landed on, they must sell game assets to make good on the rent. If they are unable to pay the rent after selling all of their game assets, then they are bankrupt and must leave the game. Their game token is then removed from the board.  15) Players may not borrow or lend money from each other, and may not borrow money from the bank.  16) When a player has finished moving their token, and has completed any property purchase activity, they have the option to buy houses and hotels to improve their properties. Players are not permitted to improve their properties at any other time.  17) Houses and hotels may only be purchased for properties where a player owns all of the properties in a particular colour coded group.  18) Houses and hotels are purchased for the amount shown on the game card.  19) If a player needs to raise funds, they can sell a property back to the bank for its original value as shown on the game card. A property can only be sold when there are no houses or hotels on the property. A player may also sell houses and hotels back to the bank for the original purchase price.  20) Where a coloured set of properties is owned and developed by a player, there may never be a difference of more than 1 house between the properties in that set. If a player wishes to buy a hotel, that is the equivalent of 5 houses in cost. A player may have 4 houses on one set and a hotel on another in that set.  21) The maximum development permitted on any one property is one hotel.  22) If a player needs to raise funds, they may mortgage a property with the bank. The bank will pay the player one half of the value of the property as shown on the game card. No rents may be collected for that property whilst it is under mortgage.  23) If a mortgaged property is then sold back to the bank, it is sold for one half of the property price as shown on the card.  24) Where fines are to be paid, the proceeds accumulate on the free parking space in the centre of the board. When a player lands on free parking, they collect all of the funds currently on the free parking space.  25) If a player is sent to the jail, they may pay £50 to be released from jail. The £50 is added to the free parking fines. The player token is then moved to “just visiting” and the players turn ends. The player takes a normal turn in the next round.  26) If a player opts to stay in jail, they give up their turn for the next 2 rounds. Whilst in jail, a player may not collect any rents from other players. At the end of the next 2 rounds, the player token is moved to “just visiting” and the players turn ends. The player takes a normal turn in the next round.  27) If a player has a “get out of jail free” card, then they place the card at the bottom of the “pot luck” or “opportunity knocks” pile as appropriate, the player token is moved to “just visiting” and the players turn ends. The player takes a normal turn in the next round.  The abridged game: In the abridged version, a time limit is agreed at the outset by all players. When the time limit is reached, and the players have all taken the same number of turns, the game ends. Each player then calculated the value of their game assets. The player with the greatest value of game assets is declared the winner. |

|  |
| --- |
| 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.*  Domain      Non-functional   |  |  | | --- | --- | | Property | Measure | | Speed | User/event response time   * should take less than a second to respond to user input | | Size | Mbytes   * program shall be less than 15,000 Mbytes in size * the program should be less 400Mbytes in size. | | Ease of Use | Time for the user to understand the game   * it should take less than five minutes for the user to be able to understand the game | | Portability | Number of target systems   * The system shall be able to be usable on Window’s computers * The system should be able to be usable on MAC’s. |   Functional   |  |  | | --- | --- | | F1 | | | Description | The software shall allow players to buy property that they land on | | Inputs | Int: playerPosition | | Outputs |  | | Error conditions | Tile already bought bought someone else  Tile not existing | | F2 | | | Description | When bought the system shall store that the property has been bought by the player | | Inputs | Player: thePlayer | | Outputs |  | | Error conditions | Player doesn’t exist (nullReference) | | F3 | | | Description | When bought the system shall store that the tile has been bought. | | Inputs | Player: thePlayer  Tile: theTile  Cost: theCost | | Outputs |  | | Error conditions | The player doesn’t exist  The tile doesn’t exist | | F4 | | | Description | If a player decides not to buy a property then it shall be auctioned to the highest bidder. | | Inputs | Int[] : player bids | | Outputs |  | | Error conditions |  | | F5 | | | Description | The system shall allocate the sum of fines payed to players to the free parking | | Inputs | Int: fine | | Outputs |  | | Error conditions |  | | F6 | | | Description | The system shall pay the player the sum of free parking when the player lands on free parking | | Inputs | Player | | Outputs |  | | Error conditions |  | | F7 | | | Description | The system shall pay the player half the properties value if they mortgage it, but until it is unmortgaged it will not be allowed to collect rent when other players land on it | | Inputs | Player | | Outputs |  | | Error conditions |  | | F8 | | | Description | The system shall pay the player the whole amount of the properties value if they sell it, but will remove that property from the number the player owns. | | Inputs | Player | | Outputs |  | | Error conditions |  | | F9 | | | Description | Landing on To Jail, or drawing a card that sends a player to jail will move them backwards to jail. | | Inputs | Player | | Outputs |  | | Error conditions |  | | F10 | | | Description | When a player lands on a property where all the properties are owned by the same player, they pay double the base rent. | | Inputs | Player | | Outputs |  | | Error conditions |  | | F11 | | | Description | When the player lands on jail they can choose to pay a £50 fine to leave immediately or stay for 2 turns. | | Inputs | Player | | Outputs |  | | Error conditions |  | |  |  | | F - Players shall start the game with 1500 of in game Money  F – Players shall each have a unique sprite to display.  F – The bank shall have a limitless amount of money to give to the players | | |

|  |
| --- |
| 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.*  See design folder of sprint 3 |

|  |
| --- |
| 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.*  See Code>scr>Junit Tests for the junit tests  See Testing for the systems testing and bug report |

|  |
| --- |
| 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?*   We did achieve objectives for this sprint  There is a working prototype  Things that went well   * Development team got to work quickly through this sprint. * Sprint was wrapped up quicker than sprint 2 * Communication between frontend developers and backend developers to fix issues and solve problems.   Things that did not go well   * No Junit tests until sprint was nearly finished. * Junit tests incomplete as development team is having to deal with bugs that aren’t caught. * Little communication from some members of the team. Specifically: Tom and Joe L.   No feedback from customer |