Collaboration & Communication

* Daily meetings: the team has to hold short meetings on a daily basis, usually around 15 minutes. This is a good way to keep people aware of the process of the project, as well as knowing how other teams are doing. The team can discuss about what we have done, what we will do and if there are any blocks.
* Open office space: the development team should locate together in an open environment to facilitate face-to-face communication. (Lagerberg , Skude, Emanuelsson, Sandahl, & Ståhl, 2013)
* Working in a Scrum team: people will work in a Scrum team which consists of these roles:
  + Product owner: who has the authority to decide what features and functionality the team has to build and how to build them. The product owner has to meet regularly with customers to keep them aware of the team process. Besides, they ask for feedbacks and what features the customer wants next. Product owner works very closely with scrum master, designer and team members to clearly figure out what the scrum team is trying to achieve. He/she has to make sure everyone is informed of what they need, clarify any issues and helps in decision making process. Product owner will take the responsibility for the overall success of the solution being developed. Being such an important role, identifying and appointing someone who qualified enough for it is important.
  + Scrum master: who makes sure everyone understands Scrum values, principles and practices. Scrum master plans and hosts all Scrum practices which includes daily stand up, sprint planning, etc. He/she assists the team in identifying blocks and works closely with the product owner, team member and the organization to remove them. Acting like a coach, Scrum master aids team members to understand their roles and responsibilities. To sum up, the Scrum master helps the Scrum team to develop a high-performance, organization-specific approach.
  + Development team: typically involve five to nine people who have the same skillsets needed to develop a good-quality software. Development team members do testing, documenting, testing code daily besides taking part in Scrum practices. They get information needed from the Product owner and working closely with Scrum master to remove any impediments.
* Team agreements: every team should specify a team agreement. There is a number of benefits of having a team agreement. It is a foundation of creating trusting relationship which strengthens the team infrastructure. Moreover, it eliminates assumptions about what is expected of everyone which results in better performance and trust. (Thakore, 2009)
* Sprint retrospective: a meeting will be held at the end of each sprint. Only Scrum team members are involved. This is an opportunity for the team to reflect about what they have done well and what could be improved for the next sprint.
* Sprint review: a meeting which includes Scrum team members and other stakeholders at the end of the sprint. It shows the current sprint working and the demonstration of the deliverables. Sprint review is a good way to make stakeholders feel involved and informed as well as increase their interests in the project.

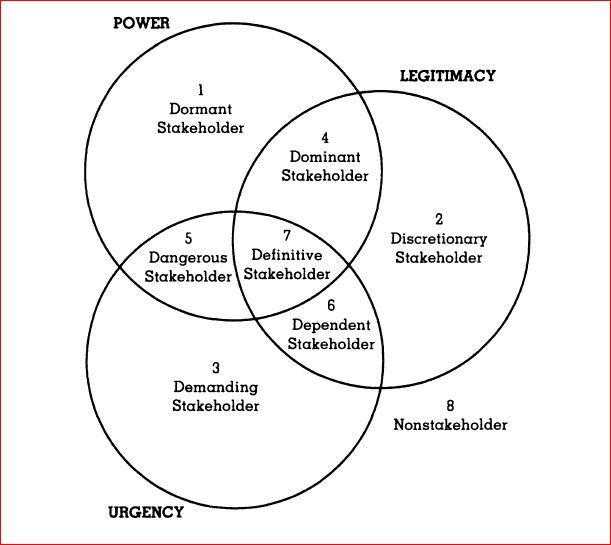
Here are a few resources that can enable the future team to upskill:

* Sutherland, J., & Schwaber, K. (2016, May 22). *ScrumGuides*. Retrieved from Scrum Guides: http://scrumguides.org/
* Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2016, May 22). *Scrum Foundation*. Retrieved from Scrum Primer: http://scrumfoundation.com/?page\_id=74
* Wang, G. (2013, Feburary 13). *Scrum Alliance*. Retrieved from Scrum: The Unity of Knowing and Doing: https://www.scrumalliance.org/community/articles/2013/february/scrum-the-unity-of-knowing-and-doing

Requirement Engineering

1) Stakeholder:

* Stakeholder analysis: identify and classify stakeholders using Stakeholder Typology (Mitchell, Agle, & Wood, 1997). There are 3 quality classes of stakeholders which are: power, legitimacy and urgency. By this way, the development team does not only make sure that they do not miss any potential stakeholders but also figure the way how to deal with different type of stakeholders.



(Picture URI: https://blackfishandseaworld.files.wordpress.com/2015/03/image02.jpg)

Here are some key stakeholders that the firm should focus on:

* Definitive Stakeholder: who have all power, legitimacy and urgency. The organization should have immediate attention to give priority to that stakeholder claims.
* Dangerous Stakeholder: who have power and urgency but no legitimacy. This kind of stakeholder can be possibly violent and be a threat to the firm.
* Dependent Stakeholder: mostly the users. They lack power but have the right to claim, since users are the ones that will bring benefits to the firm eventually. Their claims should be respected and noticed.

And some stakeholder groups that the firm can be less concern about:

* Demanding Stakeholder: those with urgent claims but having neither power nor legitimacy, could be bothersome but not dangerous.
* Dormant Stakeholder: they hold the power but yet do not have any interest or legitimate relationship with the firm. Therefore, their power remains unused. One example of dormant stakeholder is the government.
* Stakeholder Documentation: register all identified key stakeholders and specify different management strategy to them using Stakeholder Register and Stakeholder Management Strategy documents.

2) Requirements:

* Functional & Non-functional requirements: the organization should have both type of the requirements. While functional requirements are the features of the game, non-functional requirements are related to performance, security, usability and compatibility.
* User story: for software development project, user story is a good practice to manage the functional requirements should be implemented. It is a story which describes functionality that will be either valuable to user or purchase of a system or software. User story has 3 aspects: a description, a conservation and a confirmation. A good user story is independent, negotiable, valuable, estimable, small and testable.
* Product backlog: it is a prioritized, estimated list of user stories. It decides which user stories are more important so it should be implement on the first release, less important functionalities can be moved to later iterations. Product backlog should never be frozen, it is regularly updated throughout the development process. Product backlog refinement should be done in every sprint.
* Use cases: use case is a generalized description of a set of interactions between the system and one or more actors, actors are either users or systems. Use case is normally larger in scope compared to user story, usually has extensions or alternative paths to handle error. It is quite similar to a combination of user stories and acceptance tests.
* Use case diagram (UCD): UCD describes the functionalities of a system on a top down perspective at a glance. It is generally a high-level description and does not concern with details of system features as well as represent the order of system actions. UCD is suitable to describe all of the things that can be done with a database system, by all of the people who might use it.

Here are a few resources that can enable the future team to upskill:

* Mitchell, R. K., Agle, B. R., & Wood, D. J. (1997). Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts. *The Academy of Management Review*, 853-886.
* Jun , L., Qiuzhen, W., & Lin, G. (2010). Application of Agile Requirement Engineering in Modest-Sized Information Systems Development. *Software Engineering (WCSE), 2010 Second World Congress on (Volume:2 )* (pp. 207 - 210). Wuhan: IEEE.
* Hassan, S., Qamar, U., & Arslan Idris, M. (2015). Purification of requirement engineering model for rapid application development. *Software Engineering and Service Science (ICSESS), 2015 6th IEEE International Conference* (pp. 357 - 362). Beijing: IEEE.

Project Planning and Tracking

* Sprint planning: there is a sprint planning in the beginning of every sprint. It includes all Scrum team members work together to create a backlog of specific stories that have the highest priority for the product. High-priority stories are prioritized to be implement in earlier sprints. A sprint goal is defined to show what the upcoming sprint is supposed to achieve.
* Effort estimation: this is an essential part and also the hardest part of every project planning activity. Estimation is complicated because different people may have many different thinking and ways to approach a problem. There is a really high chance of giving an over or under estimation rather than an accurate estimation. In our opinion, it is nearly impossible to achieve an exactly accurate effort estimation. Subjective estimation techniques are the one that most used in ASD. For size metrics, story points and use case points are really popular. However, all current effort estimation techniques need improvement in accuracy. ( Usman, Mendes, Weidt, & Britto, 2014)
* Burn down Charts: a burn down chart can visualize the progress that you have done and what you have left in completing a user stories for a given period of time.
* Gantt Chart: the team can use Gantt chart to display project schedule information by listing project activities along with their start and finish date in a calendar format. Milestones can also be added to the Gantt chart to emphasize important events or accomplishment.
* Tracking Gantt Chart: this chart is made to keep track of project progress. Actual work percentages completed is calculated automatically. The actual start date and finish date comparison with the planned start date and finish date can be visualized. This helps the team know how well they are doing and take correlate actions if they are behind or ahead of the schedule.
* Critical Path Analysis: critical path can be easily seen through a network diagram. It is the longest path in the network diagram and at the same time determine the earliest time a project can be finished. Tasks belong to the critical path must be started or finish on time, otherwise the project completion will be delayed. Side tasks which are not part of the critical path offer some free slacks. The fact that they can be postponed for a certain amount of time without affecting the project completion date give the team some flexibilities of how and when to do these tasks.
* Source version control: having a source version control is compulsory to develop software in a team environment, this allows developers to collaborate effectively. It keeps track of the changing states of files over time and merge the contribution of multiple developers. Source version control does not only work great with code but also with documentations. There are multiple benefits of using a version control: data integrity, increased productivity, record keeping automatically, accountability. (Schwalbe, 2014)
* Task board: all the development activities are visualized in a task board. There are different kind of task board depending on the software development methods that the team used. For example, a Kanban taskboard persistently clearly visualize the flow of each stories card from development state to testing and then done. ( Grapenthin , Poggel, Book , & Gruhn, 2014)
* Program Evaluation and Review Technique (PERT): this is a technique the team can used to estimate the project duration in case of high degree of uncertainty. This is the formula:

PERT weighted average = (optimistic time + 4X most likely time + pessimistic time) / 6

* Risk register: all identified risks should be addressed in the risk register. It provides description, root courses and potential response to different project particular risks.
* Severity: a formula applied to figure out risks value. Risk which have higher Severity should be placed in a higher rank and being paid more attentions to. Severity = Likelihood x Impact.
* Expected Monetary Value (EMV): is the value based on risks that should be added to the project budget to account for project risks. EMV is not always negative, sometime it is positive indicates that the risks open some certain opportunities for the project. EMV is calculated by risk probability x Impact in Dollars.
* Risk breakdown structure (RBS): RBS is a hierarchical structure of potential risks sources. It is defined as a source-oriented grouping of risks that organizes and defines total risks exposure of the project or business. Each descending level represents an increasingly detailed definition of source of risk. (Hillson, 2003)
* Monte Carlo analysis: is a techinique which randomizes the outcome of risks and the probabilities of them occurring to help user get a better sense of how to handle these identified risks. (Mudumba & Lee, 2010)

Here are a few resources that can enable the future team to upskill for this particular category

* Britto , R., Mendes, E., & Börstler, J. (2015). An Empirical Investigation on Effort Estimation in Agile Global Software Development. *2015 IEEE 10th International Conference on Global Software Engineering* (pp. 38 - 45). Ciudad Real: IEEE.
* Grapenthin , S., Poggel, S., Book , M., & Gruhn, V. (2014). Facilitating Task Breakdown in Sprint Planning Meeting 2 with an Interaction Room: An Experience Report. *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 1 - 8). Verona: IEEE.
* Schwalbe, K. (2014). *Information Technology Project Management Seventh Edition.* Cengage Learning.
* Mudumba, V., & Lee, O.-K. (2010). A New Perspective on GDSD Risk Management: Agile Risk Management. *2010 5th IEEE International Conference on Global Software Engineering* (pp. 219 - 227). Princeton: IEEE.

Managing change

* Product backlog Grooming: refer to 3 principles: create and refining product backlog items, estimating product backlog items, prioritizing product backlog items. Product backlog grooming should be done every time a change is specified which means any new requirements will have to undertake a prioritizing and estimating process. Other existing requirements also have to be modified accordingly.
* Stakeholders actively involved: continuously receiving stakeholder feedbacks and their active participation can ensure that all product backlog items are properly chosen, avoid missing any potential requirements. Sprint review is an appropriate time to work closely with stakeholders.
* No change management during iteration: change should be avoided to happen during the middle of a sprint. If there is a chance request happens at this time, it should be left to the next iteration. Changes which are done during an iteration can result in defects due to inadequate modelling. These are regression defects cause of unforeseen side effects of architectural and code changes (Kannan, 2015)
* Documentation regularly updated: all documents have to be revised whenever a change takes place. Record different versions of document is also essential.
* Change request form: a template to document changes. It includes description of the change, what part of the project it affects, justification for changes as well as suggested implementation if the changes request is taken place. The change request form then will be reviewed by the development teams and other important stakeholders to either approve or reject the changes.

Here are a few resources that can enable the future team to upskill:

* Franklin, M. (2014). *Agile Change Management : A Practical Framework for Successful Change Planning and Implementation.* Kogan Page.
* Baruah, N. (2015). Requirement Management in Agile Software Environment. *In Proceedings of the 2015 International Conference on Soft Computing and Software Engineering (SCSE'15)* (pp. 81-83). Suta: Elsevier B.V.

Quality assurance

* Manual testing: testers play a role as an end-user and verify all features of the software by finding all possible defects. Manual testing is preliminary testing, must be carried out prior to start automating the test cases. There are different stages for manual testing such as unit testing, acceptance testing
* Automated testing: the product will be tested using scripts written by tester. Automated testing also correlates very well with the deployment of continuous integration. They boost the development team productivity, improves code quality and reduces the risk of project execution. Developers are encouraged to write and submit corresponding automated testing cases when they submit source code ( Lu, Yang, & Qian, 2014). However, automation has its own limitations:
* Automation cannot replace manual testing: not all tasks can be easily automated. Some tasks require manual testing specifically.
* Test automation could be hard to maintain due to the rapid change in technology and evolution of software products
* The needs of skilled people: writing automate tests are not an easy task. Software development skills, domain and system knowledge are required for a skilled automate test writer.
* Acceptance testing: are used to verify that user stories are developed in an expected way from customer. They are written in business domain language and developed in a conversation between the developers, testers and product owners. An example of a language used to define acceptance test is Gherkin. Acceptance tests can be done both manually and automatically. The software team will ask the user to be the actor in testing our software. Through the use of UAT it can ensure that software has practical uses in a real life situation as is functioning as intended by our users. By using UAT the team can effectively test whether or not the system can support day to day activities and is robust enough to be considered a quality and professional product by users.
* Unit testing: is a small piece of code run in isolation to test a public function or method. Unit tests are fast to write, operate and easy to maintain. Unit testing are often written by the developers during the development. The team will test every bit of software whenever a section of functionality is completed. This is to ensure that no bugs are introduced into the architecture of the software as a whole.

* Usability testing: Evaluating the usability of the software product is very important at the end of software development process. The advantage of this testing is that it will help the users satisfy when using the software, ensure no failure at performing any task by users and enhance users experience. The software team will gather some participants act as users to perform some required tasks that are believed to be critical software activities. If the users complain "can’t" or "wont" do the task correctly, they need to find the solution to that problem, and test the new design with users again to make sure it satisfies them before releasing final product to the market.
* Quality assurance checklist: a checklist can be created to ensure quality for every deliverable. The purpose is making sure nothing is overlooked and the project retains quality for all deliverables. Here is a sample of the checklist per deliverable:
* Process/Deliverable
* Quality Planning
* Quality Checking
* Quality Improvement
* Should we stop doing this (reason why)
* Behaviour Driven Development: BDD is the process of writing acceptance test before writing any code. This is the process flow of BDD:

1. Write failing acceptance tests that cover expected behaviour of the feature.
2. Write failing unit tests that cover the functionality that is being developed to allow acceptance test to pass.
3. Write the code to pass the unit tests.
4. Refactor the code and check that unit tests are still pass.
5. Ensure acceptance tests now pass.
6. The code should satisfy both unit and user level.

* Test Dirven Development: TDD is the process relies on the repetition of a very short development life cycle. The procedure that drives this cycle is called red-green-refactor:

1. Write a test (RED)
2. Run all tests (RED)
3. Write the code (GREEN)
4. Run all tests (GREEN)
5. Refactor (REFACTOR)
6. Run all test (REFACTOR)

Here are a few resources that can enable the future team to upskill:

* Watkins, J. (2009). *Agile testing : how to succeed in an extreme testing environment.* Cambridge University Press .
* Badgett, T., Sandler, C., & Myers, G. (2012). *The art of software testing third edition.* John Wiley & Sons .
* Hellmann , T., Chokshi, A., Hossein Abad, Z., Pratte, S., & Maurer, F. (2013). Agile Testing: A Systematic Mapping across Three Conferences: Understanding Agile Testing in the XP/Agile Universe, Agile, and XP Conferences. *Agile Conference (AGILE), 2013* (pp. 32 - 41). Nashville: IEEE.

# Bibliography

Lu, J., Yang, Z., & Qian, J. (2014). Implementation of continuous integration and automated testing in software development of smart grid scheduling support system. *Power System Technology (POWERCON), 2014 International Conference* (pp. 2441 - 2446). Chengdu: IEEE.

Usman, ., Mendes, E., Weidt, F., & Britto, R. (2014). Effort Estimation in Agile Software Development: A Systematic Literature Review. *Proceedings of the 10th International Conference on Predictive Models in Software Engineering* (pp. 82-91 ). Turin: ACM.

Kannan, N. (2015, May 22). *Automated change management in Agile development*. Retrieved from SearchSoftwareQuality: http://searchsoftwarequality.techtarget.com/tip/Automated-change-management-in-Agile-development

Lagerberg , L., Skude, T., Emanuelsson, P., Sandahl, K., & Ståhl, D. (2013). The Impact of Agile Principles and Practices on Large-Scale Software Development Projects: A Multiple-Case Study of Two Projects at Ericsson. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 348 - 356). Baltimore: IEEE.

Mitchell, R. K., Agle, B. R., & Wood, D. J. (1997). Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts. *The Academy of Management Review*, 853-886.

Thakore, K. (2009, December 11). *Kuntal Thakore's Blog*. Retrieved from Importance of Team Agreements…: https://kuntalthakore.wordpress.com/2009/12/11/importance-of-team-agreements/

Hillson, D. (2003). Using a Risk Breakdown Structure in project management. *Journal of Facilities Management V O L . 2 N O . 1*, 85-97.