# *Assignment 1 – Systems Development Methodologies and UML Overview*

Date assigned: Monday, January 30, 2017

Date due: **Wednesday, February 8, 2017 11:50pm**

**Learning Objectives**

Upon successful completion of this assignment, the student will be able to:

* Understand the differences between the various Systems Development Methodologies.
* Understand the different purposes and roles of the different types of UML diagrams

To do:

**Part A**

Which Systems Development Methodology will work best for a project within an organization is an important strategic question that needs to be answered before work starts on the project. Making the wrong choice could lead to disastrous results for the organization such as delayed timelines, unhappy clients, project overruns, and cancelled projects.

Assume the Systems Development Life Cycle was classified broadly into the following two categories:

1. Waterfall, being sequential, with clear and strict cut-off between phases;
2. Iterative or Agile, being repetitive with flexible cut-off rules.

List **six most important factors** an organization should consider while choosing one of the Systems Development Methodology listed above.

For each of the factors listed, describe clearly the reasons for preferring one methodology over the other.

Time sensitivity:

If the project is time sensitive, agile development would probably be the better choice. You can get a start product out much sooner and build off it to have something functional much sooner than you could manage with a Waterfall development.

System complexity:

For large and complex systems, waterfall development might be better. With a larger and more complex system you might want to begin planning and get an idea of what the system will look like before beginning to implement it. Especially with high risk systems that has people’s lives at stake.

Clarity of user requirements:

With unclear user requirements, agile will be better. You can get a vague idea of what the user wants, then start building. When you make significant changes, get the client to see what you’ve done and see what they think of it. Make changes based on their immediate feedback.

Familiarity with technology:

If you’re unfamiliar with technology, again, agile will be better. Time won’t be wasted learning the technology on small side projects and instead the effort spent learning the new technology will be put directly into the product. During later sprints, you can dedicate time to go back and fix the earlier mistakes.

System Reliability:

If you need a system that is reliable and robust, Waterfall might be slightly better. With waterfall development, you plan everything well ahead to be able to clear out any potential mistakes before they ever get implemented. With Agile though, you’ll be able to become more familiar with the product early on and notice potential issues as you become very familiar with the system.

Schedule visibility:

When the user wants to see the system development, agile will always be better. The owner gets to see changes made to the system very frequently and gives the owner ample opportunity to request changes.

**Part B**

Assume that you are a Project Manager using a waterfall development based methodology on a large and a complex project. The Chief Executive Officer of your organization recently attended a conference on prototyping methodologies. He wants you to replace the existing methodology with prototyping. What would be your decision and Why? Describe your views clearly and in detail.

* For a large and complex system, prototyping is good because there are lots of different possible issues that can occur in a large system. Using prototyping means that you can work out these problems between different iterations. If you’re prototyping properly, things should be well designed and easily able to accommodate extra subsystems getting added onto what you currently have. However, waterfall development isn’t meant to work like that. Once the system is in development, it will be built to be integrated with a few other specific parts of the system, and not with multiple other components. With a prototyped system, you’re bolting a bunch of pieces onto something that already works, but you haven’t prepared for that in a system that began being developed under a waterfall methodology. If you haven’t begun the actual system implementation, then I think it would be more feasible, but then you’ve wasted a bunch of time getting all the documentation prepared and your system analysis will have been done thinking of the system from a waterfall perspective and will need to be partially scraped and re-done. If you’ve already started the development, I don’t think switching should really be considered because you’re creating a lot of potential problems in the system. If you haven’t started yet, then it’s a matter of determining costs and time required to restart the analysis and implementation. So, I guess it depends really on where you are in the SDLC and if you’re in a place where it’s more feasible, it depends on what your system analysts say about the cost and time required to change in comparison to keeping on track.

**Part C**

Both Rapid Application Development and Agile Development are short iterative processes that deliver a reduced feature set. What are the essential differences between the two methodologies? Is one methodology better than the other in your opinion? Explain.

* RAD focuses on getting a product out there and not really going back to fix changes, unless you’re changing everything. It gets a system out there without looking at the functional requirements that the system owner wants. Agile on the other hand looks at the non-functional requirements as well and will go back and make changes to implemented subsystems to meet changes in functional requirements. Part of the sprints is going back and optimizing code, making sure the system is fast, secure, scalable, etc. RAD doesn’t really do this. It doesn’t go back and look at old code in the same way that agile does.

**Part D**

In Agile Development, what is the Product Backlog, the Sprint backlog and what are the sources of the items in the different Backlogs?

Explain how the Backlog is used in terms of sprint and release planning, and how items are prioritized and sized. Clearly specify the team roles and who prioritizes and who sizes the tasks.

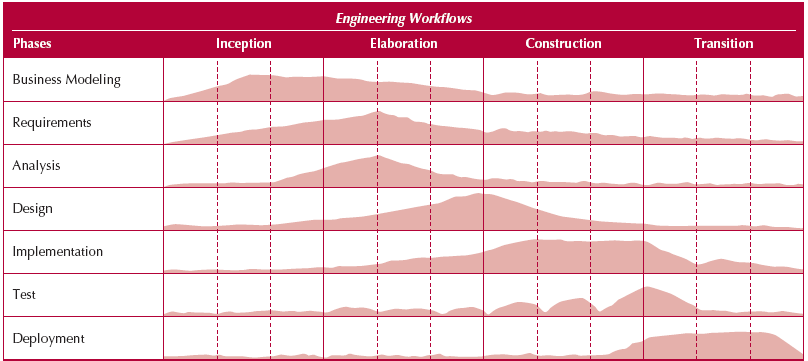
* The product backlog is the entirety of what must still be implemented in the system. Before development begins, the product backlog is an image of what the entire system must look like in terms of user stories.
* The sprint backlog is a small portion of the product backlog and contains the things that you want to accomplish in a single sprint. After each day, you can take things off of the sprint backlog as they get added to the system. Before each sprint cycle, a new sprint backlog is created to figure out what all is going to get completed in the sprint.
* The product backlog is created by the user stories generated from the requirements that the product owner has described. The product backlog is broken down into releases by the scrum master and what they think is the most important to the system and the owner. It’s broken down by what they think will bring the system owner the most value. The scrum master then breaks it down again into individual items that different developers pick and they work on them for the sprint cycle.

**Part E**

Compare the involvement of the end user (or someone representing the end user, such as the business owner) in Waterfall vs Agile Development showing which disciplines and phases the end user would be involved in. Typically, over the duration of a project, when is the end user participating/consulted? Which methodology would require more user involvement and why?

* In waterfall, the end user/system owner is heavily invested in communication early on in the project, but once the documentation has been completed and they have what they need, the team breaks off from them and goes off to work on their own until they can hand the owner the system in the future. At that point, the owner is again invested in the integration process, but they’re uninvolved throughout the development of the system.
* In Agile development, the product owner stays heavily invested in communication throughout the entire project. At each new release, the team get together with system owners for feedback and they share what they like and what they need to change. They keep doing this throughout the entire development of the system, constantly telling them what they want and constantly changing things.
* I don’t think that in one methodology the end user is necessarily consulted more or less, they’re just consulted at different times. Someone can participate a lot in the beginning of a waterfall project and stay very involved every day, until they don’t need them anymore and someone else can be involved in an agile project as little as possible and only offer feedback when requested and make their feedback short and unclear. But the two roles can also be revered. In waterfall, those doing the requirements gathering can spend all their time probing information out of the clients and the people involved in agile can be super involved and offer tons of critical feedback. It depends more on the people involved I think than the methodology that’s been chosen.

**Part F**

Given the following workflow: 

Activity Diagram

Use Case Diagram

Sequence Diagram

State Machine Diagram

Deployment Diagram

Package diagrams

Class Diagram

Diagrammatically map in where you would typically see:

1. Activity Diagrams
2. Use case diagrams
3. Package diagrams
4. Deployment diagrams
5. State machine diagrams
6. Sequence diagrams
7. Class diagrams

For each diagram, define the roles of who would author, who would review, and who would typically approve the diagrams.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Product owner | Development lead | Test lead | It/support lead | Developer | tester |
| Activity diagram | C | R/A | C | C | I | I |
| Use case diagram | C | R/A | C | C | I | I |
| Package diagram | C | A | C | R | I | C |
| Deployment diagram | C | C | A | R | C | C |
| State machine diagram | I | C | C | R/A | I | I |
| Sequence diagram | I | A | C | I | R | I |
| Class diagram | I | A | C | I | R | I |

Assume the roles are:

1. product owner – the person funding the project and speaks for the end customer
2. development lead – responsible for the programmers and is part of the delivery team
3. test lead – responsible for all the testers and is part of the delivery team
4. IT/Support lead – responsible for the deployed product after it is released. Not part of the delivery team.
5. Developer – part of the delivery team. Typically does program design and coding tasks.
6. Tester – part of the delivery team. Typically, does test design, setup and execution.

Format this in a RACI chart. See [here](https://en.wikipedia.org/wiki/Responsibility_assignment_matrix), for research on what a RACI chart is.

**Mark Breakdown:**

|  |  |
| --- | --- |
| **Section** | **Marks** |
| **Part A** | **30** |
| **Part B** | **20** |
| **Part C** | **20** |
| **Part D** | **10** |
| **Part E** | **10** |
| **Part F** | **20** |
| **Organization and English** | **5** |

**To submit**

Submit your Word document into Moodle.