DEVELOPMENT LIFE CYCLE MODEL
IT 314:Team#23

Prepared by Ishani Tiwari, Vivek Vaish and Himanshu Patel
Reviewed by the rest of team #23

The systems development life cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project, from an initial feasibility study through maintenance of the completed application. The various available development life cycles are discussed here:

1.**Waterfall Model**

The waterfall model is a sequentially designed model comprising of several steps each of which must be completed before the next starts.

- **Systems analysis, requirements definition: :** Establishes a high-level view of the intended project and determines its goals. Refines project goals into defined functions and operation of the intended application. Analyzes end-user information needs.

- **Systems design:** Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation.

- **Implementation:** The real code is written here.

- **Integration and testing:** Brings all the pieces together into a special testing environment, then checks for errors, bugs and interoperability..

- **Maintenance:** What happens during the rest of the software's life: changes, correction, additions, moves to a different computing platform and more. This, the least glamorous and perhaps most important step of all, goes on seemingly forever.

MERITS:

1. This model is very simple and most easy to understand.
2. For well understood problems this is the most suitable model.
3. It has fixed start and end dates so it enforces discipline.

DEMERITS:

1. The model requires that the requirements should be stated explicitly which might not be feasible.
2. Not very good for improving product visualization
3. The model has difficulty in accommodating the natural uncertainty that exists in many projects. Changes can cause confusion as the project proceeds.
4. A working version of the product is available only in the later stages. Till then, the customer must have patience.
5. A major blunder undetected in the earlier stages can be disastrous later on.
6. This model is the oldest paradigm for software engineering so it has not the capacity to produce an effect.

Justification for not choosing this model :

The waterfall model assumes that the only role for users is in specifying requirements, and that all requirements can be specified in advance. Unfortunately, requirements grow and change throughout the process and beyond, calling for considerable feedback and iterative consultation. This model won't give us a good visualisation and hence it will be difficult to manage. In addition to this, this model doesn't provide scope for correcting mistakes.

2. ITERATIVE WATERFALL MODEL

Many engineers recommend modified versions of the waterfall model. In the traditional waterfall model, the different stages of development are not allowed to overlap. One common type of modification allows some of the stages to overlap, which results in reduced documentation requirements and a reduced cost of returning to earlier stages to make changes.

MERITS:

1. This model allows some of the stages to overlap. Overlapping stages, such as the requirements stage and the design stage, allow the development team to integrate feedback from the design phase into the requirements

DEMERITS:

1. Process becomes time consuming and costly.
2. The Iterative Waterfall model complicates the measurement of progress.

3.PROTOTYPING MODEL

Prototyping can be useful in determining how a design meets a set of requirements. You can build a prototype, adjust the requirements, and revise the prototype several times, gaining an understanding of the project's overall goals.

MERITS:

1. This paradigm assists to understand what is to be built when requirements are fuzzy.
2. Provides a good mechanism for understanding the customers requirements
3. Illustrates data formats, messages, reports, interactive dialogues.
4. Especially useful for GUI (Graphical User Interface) development.
5. Facilitates critical examination of technical issues associated with software development.
6. Performance related issues e.g. response time and efficiency of algorithm.
7. Reduces wasted development effort
8. Difficulty of "getting a product right first time".
9. Inevitable discarding of initial product.
10. Situations warranting prototyping.
11. User requirements are not well understood.
12. Technical aspects are not well understood.

DEMERITS:

1. Usually the prototype is built without adequately considering overall software quality or long-term maintainability.
2. Lack of appreciation that Prototype building is very different from product building.
3. Compromises on the choices like operating system or programming language, are often made during prototyping.
4. Customer demands that the prototype be "quickly" converted to a working product. The prototype is confused to be the actual product.

Justification for not using this model:

In our requirement gathering phase we have a pretty clear idea about our requirements for the software. Hence, to spend more and more time on prototype evaluation would not be necessary in our case and furthermore it will consume a lot of time.

4.**RAPID APPLICATION DEVELOPMENT MODEL**

Rapid Application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. It enables a development team to create a "fully functional system" within very short time periods, provided the requirements are well understood and project scope is constrained.

MERITS:

1. All functions are modularized so it is easy to work with.
2. High speed process.
3. Constant integration isolates problems and encourages customer feedback.
4. Increases quality through the involvement of the user in the analysis and design stages.
5. Quality is both the degree to which a delivered application meets the needs of users as well as the degree to which a delivered system has low maintenance costs.
6. It is flexible and adaptable to changes.

DEMERITS:

1. For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
2. RAD requires equal commitment from the developers and customers towards the rapid development process.
3. RAD cannot be applied to projects that cannot be properly modularized.
4. RAD is not appropriate for areas using new technology (high technical risk).
5. RAD requires that the requirements are "well understood" and that the project scope is "constrained".

Justification for not using this model:

RAD requires proper commitment from all the project members in order to have a small development cycle. Since this is our first such big project it is not reasonable to expect equal contribution from every group member. Also full efficiency might also not be possible which this

model requires.

5.**EVOLUTIONARY MODEL**

MERITS:

1. This model enables software engineers to develop increasingly more complete versions of the software.
2. User gets an opportunity to experiment/use the partial system much before the fully developed version is released.
3. Helps in eliciting requirements.
4. Core module gets tested very thoroughly (since it gets tested at the time of each release)
5. Entire resource requirements need not be committed to the project at the same time

DEMERITS:

1. Not suitable for small projects
2. Management Complexity is more
3. Highly skilled resources are required for risk analysis.

Examples of Evolutionary Models:

- The incremental Model
- The Spiral Model
- The Concurrent development model

5a. INCREMENTAL MODEL

Construct a partial implementation of a total system then slowly add increased functionality. The incremental model prioritizes requirements of the system and then implements them in groups.

MERITS:

1. This model focuses on the delivery of an operational product with each increment.
2. Useful when staffing is unavailable for a complete implementation by the business deadline of the project.
3. Early increments can be implemented by fewer people, and depending on the acceptability of the core product, more resources can be added to implement subsequent stages.
4. Increments can be planned to manage technical risks.

DEMERITS:

1. Agreement on core functionality is not easy.
2. The process can be time consuming.
3. Depends upon client agreeing for it.

5b. SPIRAL MODEL

The spiral model is an iterative model that attempts to combine advantages of the top-down and bottom-up models of software design. The goal is to reduce, as much as possible, an application's time-to-market; in the traditional waterfall model, since each step must be completed before the next one starts, the time-to-market can be much longer.

MERITS:

1. Provides potential for rapid development of increasingly more complete versions of the software.
2. Useful when we are uncertain about requirements, technology to be used.
3. Provides direct support for coping with project risk

DEMERITS:

1. It maintains the systematic stepwise approach suggested by the classical life cycle but incorporates it into an iterative framework that more realistically reflects the real world.
2. Difficult to convince the customer that the evolutionary model is controllable.
3. Demands considerable risk assessment expertise.
4. Demands a direct consideration of technical risk at all stages of the project and, if properly applied, should reduce risks before they become problematic.

Justification for not using this model:

This model is generally suitable for large and complicated projects. It will be hard to break the project down into separate modules. The overall software design procedure we plan to follow is quite clear to us so we are not applying spiral model as a whole to our overall project. The scope of the project is not so big so it doesn't require an evolutionary model.

5c. CONCURRENT DEVELOPMENT MODEL

Development and testing are performed concurrently and multiple baselines are included as a part of the project.

MERITS:

1. Provides an accurate picture of the current state of project.
2. Defines a networks of activities
3. Involved in activities typically associated with many phases of development simultaneously.
4. Immediate feedback from testing
5. New features can be added late in the project
6. No surprises during formal validation because testing has been continuous

DEMERITS:

1. The SRS must be continually updated to reflect changes
2. It requires discipline to avoid adding too many new features too late in the project.
3. It requires each member to work efficiently as monitoring will still be hard. This has been overcome with each member committing towards the project.

Justification for not using this model:

In our requirement gathering phase we found out that the work cannot be divided exclusively in phases that it can be done concurrently. Each phase must be completed before the next phase begins. therefore we cannot use this model.

SELECTED LIFE CYCLE MODEL FOR OUR PROJECT :

We have chosen **"Agile software development model "** for our polling website.

Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.

It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

Agile methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames (timeboxes) that typically last from one to four weeks.

Each iteration involves a team working through a full software development cycle, including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. Stakeholders produce documentation as required.

An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration. Multiple iterations might be required to release a product or new features.

Advantages of Agile software development

1. Agile methodology has an adaptive team which is able to respond to the changing requirements.
2. The team does not have to invest time and effort and finally find that by the time they delivered the product, the requirement of the customer has changed.

3. Face to face communication and continuous inputs from customer representative leaves no space for guesswork.
4. The documentation is crisp and to the point to save time.
5. The end result is the high quality software in least possible time duration and satisfied customer.
6. In a nutshell this means that you can get development started fast, but with the caveat that the project scope statement is "flexible" and not fully defined. Hence this can be one of the major causes of scope creep if not managed properly.

Disadvantages of Agile software development

1. In case of some software deliver-ables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
2. The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
3. Only senior programmers are capable of taking the kind of decisions required during the development process. Hence newbie programmers need to be combined with experienced resources.

Why Agile development suits our project?

1. This app needs to adapt as we get more users. There are not many polling websites, so experience on ours will let us know how to further develop the project.
2. A lot of the quality of software depends on the user interface, to keep in mind the look and usability. This will need many and frequent iterations to waste minimum work and get a good UI.
3. In our design we are realising how we need several prototypes to get a particular feature or interface correct.
4. Since the team has such skill set that one member might have to contribute to more than one area of the project, he/she can move from one area to another if we have short-term and specific goals.
5. We have listed some features which will be implemented based on priority. As this is an ‘incremental’ model, we will add the features, according to time available.
6. The regular iterations will work well when we need regular feedback from mentors or testers.
7. The team is enthusiastic about the idea and is working hard to put together a list of features, the motivation matches with the agile methodologies.