Software Development Modules

A software process model is an abstraction of the software development process. The models specify the stages and order of a process.[1]

Examples of SDLC Models;

Waterfall SDLC Model

The waterfall is a cascade SDLC model that presents the development process like the flow, moving step by step through the phases of analysis, projecting, realization, testing, implementation, and support. This SDLC model includes gradual execution of every stage. Waterfall implies strict documentation. The features expected of each phase of this SDLC model are predefined in advance.

The waterfall life cycle model is considered one of the best-established ways to handle complex projects. This approach allows avoiding many mistakes that may appear because of insufficient control over the project. However, it results in pervasive documentation development. It is beneficial to the developers who may be working with the product in the future, but it takes a long time to write everything down.

In some cases, the feedback loop is included. It allows making short reviews of each stage's result and applying some minor amendments. This loop enables specialists to return to the previous phase for a short period.

If something significant changes in the initial plan, a team should wait until the very last stage to return to the beginning and pass all software life cycle phases again.

ADVANTAGES

- Simple to use and understand
- Management simplicity thanks to its rigidity: every phase has a defined result and process review
- Development stages go one by one
- Perfect for the small or mid-sized projects where requirements are clear and not equivocal
- Easy to determine the key points in the development cycle

DISADVANTAGES

- The software is ready only after the last stage is over
- High risks and uncertainty
- Not the best choice for complex and object-oriented projects
- Inappropriate for the long-term projects
- The progress of the stage is hard to measure while it is still in the development
- Integration is done at the very end, which does not give the option of identifying the problem in advance

[1] https://existek.com/blog/sdlc-models/

Iterative SDLC Model

The iterative model means that the whole process is divided into a particular number of loops, and during each of them, developers build a limited number of features.

The development process may start with the requirements to the functional part, which can be expanded later. The process is repetitive, allowing to make new versions of the product for every cycle. Every iteration (that lasts from two to six weeks) includes the development of a separate component of the system. After that, this component is added to the features developed earlier.

ADVANTAGES

- Some functions can be quickly developed at the beginning of the development lifecycle
- The paralleled development can be applied
- The progress is easy measurable
- The shorter iteration is - the easier testing and debugging stages are
- It is easier to control the risks as high-risk tasks are completed first
- Problems and risks defined within one iteration can be prevented in the next sprints

DISADVANTAGES

- Iterative model requires more resources than the waterfall model
- Constant management is required
- Issues with architecture or design may occur because not all the requirements are foreseen during the short planning stage
- Bad choice for the small projects
- The process is difficult to manage
- The risks may not be completely determined even at the final stage of the project
- Risks analysis requires involvement of the highly-qualified specialists

Spiral SDLC Model

Spiral model is a combination of the Iterative and Waterfall SDLC models with a emphasis on the risk analysis. The main issue of the spiral model is defining the right moment to take a step into the next stage. The preliminary set timeframes are recommended as the solution to this issue. The shift to the next stage is done according to the plan, even if the work on the previous step isn’t done yet. The plan is introduced based on the statistical data received in the last projects and even from the personal developer’s experience.

ADVANTAGES

- Lifecycle is divided into small parts, and if the risk concentration is higher, the phase can be finished earlier to address the treats
- The development process is precisely documented yet scalable to the changes
- The scalability allows to make changes and add new functionality even at the relatively late stages

DISADVANTAGES

- Can be quite expensive
- The risk control demands involvement of the highly-skilled professionals
- Can be ineffective for the small projects
- Big number of the intermediate stages requires excessive documentation

V-shaped SDLC Model

The V-shaped algorithm differs from the previous ones by the work approach and the architecture. If we visualize this model, we'll see that there appears one more axis, unlike the waterfall and iterative models. Along with the first one, they constitute the V letter.

The V-model is called this way because of the scheme's appearance and because its primary priorities are Verification and Validation. Stages positioned along the left axis display the verification phases, and the ones on the right are responsible for validation.

To summarize, the V-shaped SDLC model is an expansion of the classic waterfall model, and it's based on the associated test stage for every development stage. Every phase includes the current process control to ensure that the conversion to the next stage is possible.

ADVANTAGES

- Every stage of V-shaped model has strict results so it's easy to control
- Testing and verification take place in the early stages
- Good for the small projects, where requirements are static and clear

DISADVANTAGES

- Lack of the flexibility
- Bad choice for the small projects
- Relatively big risks

Agile SDLC Model

All work is split into iterations like the iterative model. These iterations are named sprints. The team initially defines what actions they'll need to perform in a particular timeframe. The main difference with the iterative approach is that this amount of work is not strict and can be changed in the middle of the process.

The following distinction is that Agile doesn't ever leave customers in ignorance. Specialists on a provider's side constantly stay in contact with the client. They give him updates on the performed work and familiarize him with the plan. All changes are also discussed with the customer and approved by him. Each stage in Agile should be analyzed and accepted by all sides before the development team can move on to the next one.

Agile includes daily or weekly calls and Sprint reviews.

ADVANTAGES

- Corrections of functional requirements are implemented into the development process to provide the competitiveness
- Project is divided by short and transparent iterations
- Risks are minimized thanks to the flexible change process

DISADVANTAGES

- Difficulties with measuring the final cost because of permanent changes
- The team should be highly professional and client-oriented
- New requirements may conflict with the existing architecture
- With all the corrections and changes there is possibility that the project will exceed expected time

History of Software Construction

Software construction is a software engineering discipline. It is the detailed creation of working meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging. It is linked to all the other software engineering disciplines, most strongly to software design and software testing.[2]

The term software engineering was suggested at conferences organized by NATO in 1968 and 1969 to discuss the 'software crisis'. The software crisis was the name given to the difficulties encountered in developing large, complex systems in the 1960s. It was proposed that the adoption of an engineering approach to software development would reduce the costs of software development and lead to more reliable software.[3] A search for solutions began. It concentrated on better methodologies and tools. The most prominent were programming languages reflecting the procedural, modular, and then object-oriented styles. Software engineering is intimately tied to their emergence and improvement. Also of significance were efforts of systematizing, even automating program documentation and testing. Ultimately, analytic verification and correctness proofs were supposed to replace testing.[4]

---

[2] https://en.wikipedia.org/wiki/Software_construction

[3] https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/History/index.html

[4] https://ieeexplore.ieee.org/iel7/52/8474478/08474489.pdf