MODELS FOR THE EXECUTION OF SOME EFFECTIVE MANAGEMENT TECHNIQUES FOR MANAGING GLOBAL TEAMS

Predictive SDLC Models

Predictive (heavyweight) models include:

- Waterfall
- Iterative
- Spiral
- V-shaped

There are many more options, but these are the most common ones. Let's discover the main characteristics of each. We won't dive deep into the phases of each model because they are pretty similar. So, let's find out each model's peculiarities and pros and cons.

Adaptive SDLC Models

Among different SDLC models and methodologies, adaptive (agile) are the brightest candidates nowadays. The agile approach opens up new possibilities for specialists, enables more flexibility, and puts the communication between people ahead of the blind plan following. Realizations of Agile models include:

- Scrum
- XP
- Kanban

SDLC MODELS

Waterfall SDLC Model

The waterfall model is a linear, sequential approach to the software development lifecycle (SDLC) that is popular in software engineering and product development. The waterfall model uses a logical progression of SDLC steps for a project, similar to the direction water flows over the edge of a cliff. It sets distinct endpoints or goals for each phase of development. Those endpoints or goals can't be revisited after their completion.

Phases of the waterfall model

When used for a software development process, the waterfall methodology has seven stages:

1. Requirements. Potential requirements, deadlines and guidelines for the project are analyzed and placed into a formal requirements document, also called a functional specification. This stage of development defines and plans the project without mentioning specific processes.

2. Analysis. The system specifications are analyzed to generate product models and business logic to guide production. This is also when financial and technical resources are audited for feasibility.
3. Design. A design specification document is created to outline technical design requirements, such as the programming language, hardware, data sources, architecture and services.
4. Coding and implementation. The source code is developed using the models, logic and requirement specifications designated in the prior phases. Typically, the system is coded in smaller components, or units, before being put together.
5. Testing. This is when quality assurance, unit, system and beta tests identify issues that must be resolved. This may cause a forced repeat of the coding stage for debugging. If the system passes integration and testing, the waterfall continues forward.
6. Operation and deployment. The product or application is deemed fully functional and is deployed to a live environment.
7. Maintenance. Corrective, adaptive and perfective maintenance is carried out indefinitely to improve, update and enhance the product and its functionality. This could include releasing patch updates and new versions.

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.

Iterative SDLC Model

The iterative model resembles a waterfall model, but there is quite a considerable difference between them. For example, let's suppose there's an app that contains ten core features. In the waterfall case, all ten functions will be thoroughly planned during the requirement analysis and design phases and then steadily implemented during the development stage. The iterative model is quite different. It implies that the whole process is divided into a particular number of iterations, and during each of them, developers build a limited number of features.

So, the Iterative SDLC model does not require a complete list of requirements before the project starts. 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. Speaking with math terminology, the iterative model is a realization of the sequential approximation method; that means a gradual closeness to the planned final product shape. For example, during the first iteration, the team has decided to work on three features out of 10. While creating them, developers pass all stages of the software development process, starting from the requirement gathering to the deployment and maintenance. When they move to the next set of functions, the development cycle starts over.

Spiral SDLC Model

Spiral model is a combination of the Iterative and Waterfall SDLC models with a significant accent 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.

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. Let's clear the terms in a few words, so there's no misconception. Verification and validation mean different things, though they seem pretty similar. The goal of verification is to determine whether the software is consistent with the initial technical requirements. Validation, in turn, should confirm whether the product corresponds to the business needs, whether it serves its intended purpose, whether it acts as planned. To summarize, verification accounts for aligning features with the technical requirements based on the business requirements. Validation manages the last ones.

These concepts include different types of product testing. These methods are located along the respective axes. One on the left side necessarily has an associated one on the right. For example, the requirement analysis stage corresponds to acceptance testing, system design to system testing, architecture design to integration testing, etc. 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. This is a rigorous model, and the next step is started only after the previous one is over. Every phase includes the current process control to ensure that the conversion to the next stage is possible.

Agile SDLC Model

Agile is a philosophy, not a specific development approach. It is a whole family of methodologies. Scrum, Kanban, or XP (extreme programming) are among the most common realizations of the Agile SDLC. Let's find out the core principles of Agile in general and then take a brief look at some of its realizations. Its first peculiarity is that 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. Sprint reviews have such a structure – the first half of the meeting is dedicated to the performed work, and the second half is about planning the next Sprint.

One more aspect of the Agile software life cycle is face-to-face communication. It is one of 12 Agile principles. Considering that companies often outsource, meeting a customer face-to-face is problematic, but Agile offers video conferencing instead of audio calls to observe the body language. It helps to avoid some misunderstandings and build better relationships with clients. A lot more can be said on Agile SDLC, but let's better look at specific numbers. Research on Agile methodology shows incredible results. Here are the main key points:

- The adoption of Agile has grown from 37% in 2020 to 86% in 2021
- 50% more achieved business objectives with Agile

- 70% of people choose Agile due to the possibility of changing project priorities at any moment
- 60% of programmers working with Agile highlight noticeably faster time to market

Scrum/XP

People consider Scrum and Extreme programming the two different implementations of Agile. Together, they make an excellent approach to software development, but it's apples and oranges separately. Scrum is a project management methodology, and XP is a development technique. Extreme programming contains methods targeting specifically the technical side of the software development process. They include continuous integration, pair programming, test-driven development, etc. Scrum, in turn, is oriented towards the organization of the process. Sprints lie at the core of Scrum. A sprint is an iteration that lasts from 2 to 4 weeks. It starts with the planning of the product and an iteration itself. Then, the product passes the stages of a chosen software development methodology (XP is the most popular option in terms of Scrum). At the end of the sprint, the team delivers the developed part of the product, and then it's time for the retrospective – sprint review and analysis. The core methods of Scrum contain sprint planning, daily meet-ups, demos and reviews, and a retrospective at the end of each sprint. Its main principles are courage, commitment, focus, openness, respect. Also, Scrum, as a realization of Agile, is subject to 12 Agile principles. The visibility and obviousness of the process are what make Scrum stand out. Sprint backlogs (SB) and product backlogs (PB) would be excellent proof of it. PB is a set of objectives that should be achieved at the end of development. Product backlog includes the names of items, their priority, and a number of a sprint when they need to be completed. In sprint backlog, each item is divided into small steps called tasks. SB defines all tasks that developers undertake to complete during a particular sprint.

Kanban

Kanban is another realization of Agile. It somehow reminds Scrum, but still, they have some distinct differences. Both approaches use boards to track progress and have similar sections. The ones of Kanban are:

- Requested
- In progress
- Done

The range of statuses in Kanban is less extensive, but it is sufficient to keep up with the software development process. The key differences lie in the work approach and main principles of the method. Let’s start with planning. Kanban doesn’t obligate devs to follow the plan, unlike Scrum strictly. It is open to constant changes. With Kanban, team members don’t track the exact time spent on a task. Instead, the board includes the “Recommended timeframe” section. It allows estimating the approximate development time in the end. Also, Scrum has strictly defined roles, such as scrum master, product owner, and a development team. In Kanban, no roles are set, so the flexibility is much higher. Iterations in Kanban don’t have predefined timeframes – they can change depending on the amount of work. Scrum seems very flexible before you find out about Kanban, enabling even more flexibility. However, it’s questionable how effective such an approach will be because an undisciplined team is as destructive as an overdisciplined. So, it’s essential to consider all pros and cons before deciding on a software development methodology.