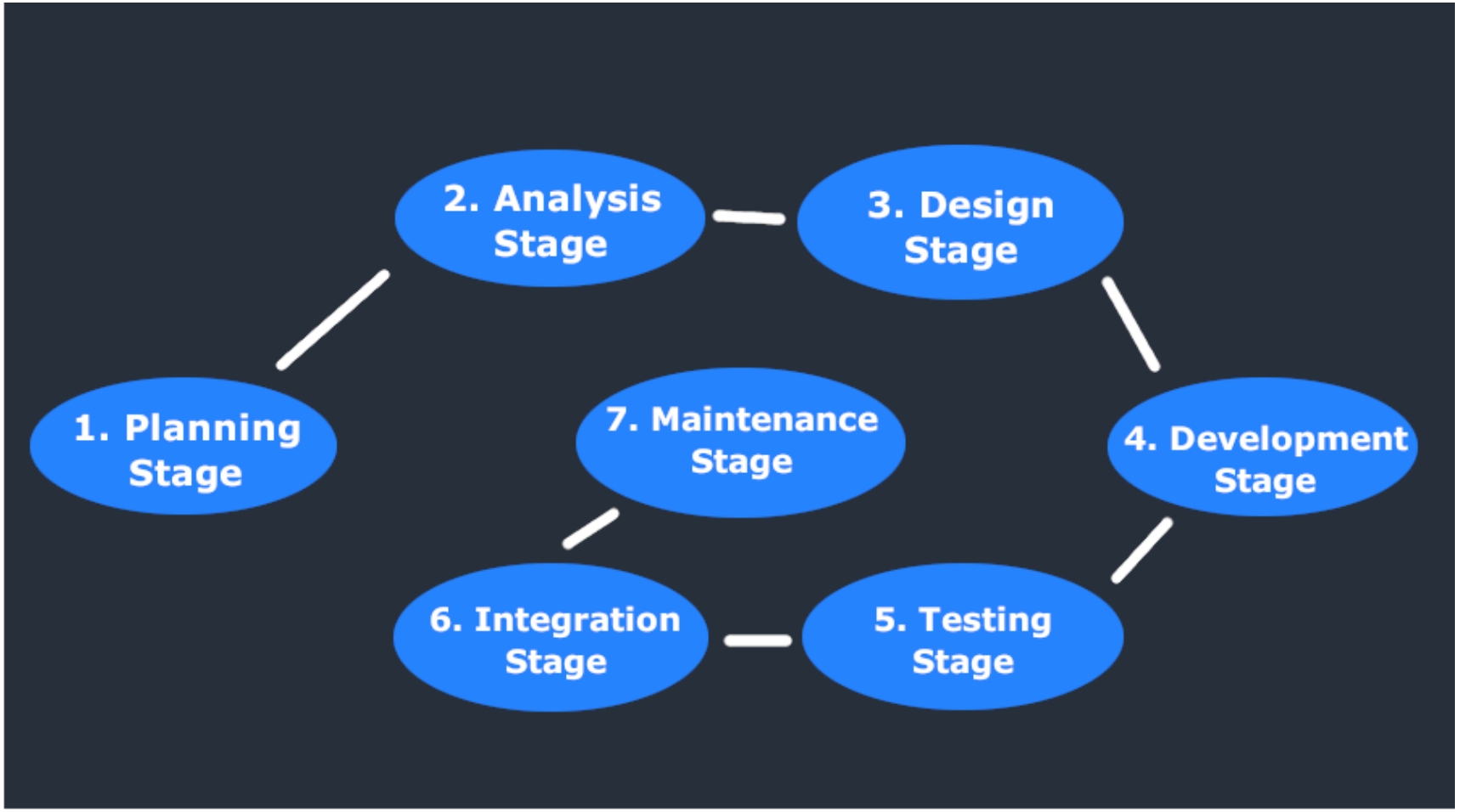
System Development Life Cycle Guide



The software development process is normally long and tedious. But project managers and system analysts can leverage software development life cycles to outline, design, develop, test, and eventually deploy information systems or software products with greater regularity, efficiency, and overall quality.

What is System Development Life Cycle?

A system development life cycle or SDLC is essentially a project management model. It defines different stages that are necessary to bring a project from its initial idea or conception all the way to deployment and later maintenance.

System Development Life Cycle US Guide

In this guide, we'll break down everything you need to know about the system development life cycle, including all of its stages. We'll also go over the roles of system analysts and the benefits your project might see by adopting SDLC.

7 Stages of the System Development Life Cycle

There are seven primary stages of the modern system development life cycle. Here's a brief breakdown:

- Planning Stage
- Feasibility or Requirements of Analysis Stage
- Design and Prototyping Stage
- Software Development Stage
- Software Testing Stage
- Implementation and Integration
- Operations and Maintenance Stage

Now let’s take a closer look at each stage individually.

Planning Stage

Before we even begin with the planning stage, the best tip we can give you is to take time and acquire proper understanding of app development life cycle.

The planning stage (also called the feasibility stage) is exactly what it sounds like: the phase in which developers will plan for the upcoming project.

It helps to define the problem and scope of any existing systems, as well as determine the objectives for their new systems.

By developing an effective outline for the upcoming development cycle, they'll theoretically catch problems before they affect development.

And help to secure the funding and resources they need to make their plan happen.

Perhaps most importantly, the planning stage sets the project schedule, which can be of key importance if development is for a commercial product that must be sent to market by a certain time.

Analysis Stage

The analysis stage includes gathering all the specific details required for a new system as well as determining the first ideas for prototypes.

Developers may:

- Define any prototype system requirements
- Evaluate alternatives to existing prototypes
- Perform research and analysis to determine the needs of end-users

Furthermore, developers will often create a software requirement specification or SRS document.

This includes all the specifications for software, hardware, and network requirements for the system they plan to build. This will prevent them from overdrawing funding or resources when working at the same place as other development teams.

Design Stage

The design stage is a necessary precursor to the main developer stage.

Developers will first outline the details for the overall application, alongside specific aspects, such as its:

- User interfaces
- System interfaces
- Network and network requirements
- Databases

They'll typically turn the SRS document they created into a more logical structure that can later be implemented in a programming language. Operation, training, and maintenance plans will all be drawn up so that developers know what they need to do throughout every stage of the cycle moving forward.

Once complete, development managers will prepare a design document to be referenced throughout the next phases of the SDLC.

Development Stage

The development stage is the part where developers actually write code and build the application according to the earlier design documents and outlined specifications.

This is where Static Application Security Testing or SAST tools come into play.

Product program code is built per the design document specifications. In theory, all of the prior planning and outlined should make the actual development phase relatively straightforward.

Developers will follow any coding guidelines as defined by the organization and utilize different tools such as compilers, debuggers, and interpreters.

Programming languages can include staples such as C++, PHP, and more. Developers will choose the right programming code to use based on the project specifications and requirements.

Testing Stage

Building software is not the end.

Now it must be tested to make sure that there aren't any bugs and that the end-user experience will not negatively be affected at any point.

During the testing stage, developers will go over their software with a fine-tooth comb, noting any bugs or defects that need to be tracked, fixed, and later retested.

t's important that the software overall ends up meeting the quality standards that were previously defined in the SRS document.

Depending on the skill of the developers, the complexity of the software, and the requirements for the end-user, testing can either be an extremely short phase or take a very long time. Take a look at our top 10 best practices for software testing projects for more information.

Implementation and Integration Stage

After testing, the overall design for the software will come together. Different modules or designs will be integrated into the primary source code through developer efforts, usually by leveraging training environments to detect further errors or defects.

The information system will be integrated into its environment and eventually installed. After passing this stage, the software is theoretically ready for market and may be provided to any end-users.

Maintenance Stage

The SDLC doesn't end when software reaches the market. Developers must now move into a maintenance mode and begin practicing any activities required to handle issues reported by end-users.

Furthermore, developers are responsible for implementing any changes that the software might need after deployment.

This can include handling residual bugs that were not able to be patched before launch or resolving new issues that crop up due to user reports. Larger systems may require

longer maintenance stages compared to smaller systems.

Role of System Analyst

An SDLC's system analyst is, in some ways, an overseer for the entire system. They should be totally aware of the system and all its moving parts and can help guide the project by giving appropriate directions.

The system analyst should be:

- An expert in any technical skills required for the project
- A good communicator to help command his or her team to success
- A good planner so that development tasks can be carried out on time at each phase of the development cycle

Thus, systems analysts should have an even mix of interpersonal, technical, management, and analytical skills altogether. They're versatile professionals that can make or break an SDLC.

Their responsibilities are quite diverse and important for the eventual success of a given project. Systems analysts will often be expected to:

- Gather facts and information
- Make command decisions about which bugs to prioritize or what features to cut
- Suggest alternative solutions
- Draw specifications that can be easily understood by both users and programmers
- Implement logical systems while keeping modularity for later integration

- Be able to evaluate and modify the resulting system as is required by project goals
- Help to plan out the requirements and goals of the project by defining and understanding user requirements

6 Basic SDLC Methodologies

Although the system development life cycle is a project management model in the broad sense, six more specific methodologies can be leveraged to achieve specific results or provide the greater SDLC with different attributes.

Waterfall Model

The waterfall model is the oldest of all SDLC methodologies. It's linear and straightforward and requires development teams to finish one phase of the project completely before moving on to the next.

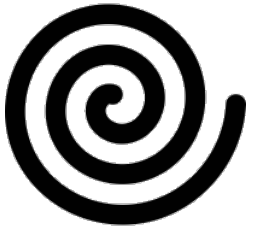
Each stage has a separate project plan and takes information from the previous stage to avoid similar issues (if encountered). However, it is vulnerable to early delays and can lead to big problems arising for development teams later down the road.

Iterative Model

The iterative model focuses on repetition and repeat testing. New versions of a software project are produced at the end of each phase to catch potential errors and allow developers to constantly improve the end product by the time it is ready for market.

One of the upsides to this model is that developers can create a working version of the project relatively early in their development life cycle, so implement the changes are often less expensive.

Spiral Model



Spiral models are flexible compared to other methodologies. Projects pass through four main phases again and again in a metaphorically spiral motion.

It's advantageous for large projects since development teams can create very customized products and incorporate any received feedback relatively early in the life cycle.

V-Model

The V-model (which is short for verification and validation) is quite similar to the waterfall model. A testing phase is incorporated into each development stage to catch potential bugs and defects.

It's incredibly disciplined and requires a rigorous timeline. But in theory, it illuminates the shortcomings of the main waterfall model by preventing larger bugs from spiraling out of control.

Big Bang Model

The Big Bang model is incredibly flexible and doesn't follow a rigorous process or procedure. It even leaves detailed planning behind. It's mostly used to develop broad ideas when the customer or client isn't sure what they want. Developers simply start the project with money and resources.

Their output may be closer or farther from what the client eventually realizes they desire. It's mostly used for smaller projects and experimental life cycles designed to inform other projects in the same

company.

Agile Model

The agile model is relatively well-known, particularly in the software development industry.

The agile methodology prioritizes fast and ongoing release cycles, utilizing small but incremental changes between releases. This results in more iterations and many more tests compared to other models.

Theoretically, this model helps teams to address small issues as they arise rather than missing them until later, more complex stages of a project.

Benefits of SDLC

SDLC provides a number of advantages to development teams that implement it correctly.

Clear Goal Descriptions

Developers clearly know the goals they need to meet and the deliverables they must achieve by a set timeline, lowering the risk of time and resources being wasted.

Proper Testing Before Installation

SDLC models implement checks and balances to ensure that all software is tested before being installed in greater source code.

Clear Stage Progression

Developers can't move on to the next age until the prior one is completed and signed off by a manager.

Member Flexibility

Since SDLCs have well-structured documents for project goals and methodologies, team members can leave and be replaced by new members relatively painlessly.

Perfection Is Achievable

All SDLC stages are meant to feed back into one another. SDLC models can therefore help projects to iterate and improve upon themselves over and over until essentially perfect.

No One Member Makes or Breaks the Project

Again, since SDLCs utilize extensive paperwork and guideline documents, it's a team effort and losing one even major member will not jeopardize the project timeline.

What You Need to Know About System Development Life Cycle

Where is SDLC Used?

System development life cycles are typically used when developing IT projects.

Software development managers will utilize SDLCs to outline various development stages, make sure everyone completes stages on time and in the correct order, and that the project is delivered as promptly and as bug-free as possible.

SDLCs can also be more specifically used by systems analysts as they develop and later implement a new information system.

What SDLC Model is Best?

It largely depends on what your team's goals and resource requirements are.

The majority of IT development teams utilize the agile methodology for their SDLC. However, others may prefer the iterative or spiral methodologies.

All three of these methods are popular since they allow for extensive iteration and bug testing before a product is integrated with greater source code or delivered to market.

DevOps methodologies are also popular choices. And if you ever need a refresher course on what is DevOps, you needn't worry as our team at CloudDefense has got you covered!

What Does SDLC Develop?

SDLC can be used to develop or engineer software, systems, and even information systems. It can also be used to develop hardware or a combination of both software and hardware at the same time.

FAQs

What Were the 5 Original Phases of System Development Life Cycle?

The systems development life cycle originally consisted of five stages instead of seven. These included planning, creating, developing, testing, and deploying. Note that it left out the major stages of analysis and maintenance.

What Are the 7 Phases of SDLC?

The new seven phases of SDLC include planning, analysis, design, development, testing, implementation, and maintenance.

What is System Development Life Cycle in MIS?

In the greater context of management information systems or MIS, SDLC helps managers to design, develop, test, and deploy information systems to meet target goals.

Conclusion

Ultimately, any development team in both the IT and other industries can benefit from implementing system development life cycles into their projects. Use the above guide to identify which methodology you want to use in conjunction with your SDLC for the best results.