Software Engineering Concepts and Models

*** Meenakshi Rathod, Assistant Professor, Dept of Computer Science, Govt First Grade College, Shapur.**

****Sampatkumari.M.Bandagar, Assistant Professor and HOD of Computer Science, Govt First Grade College, Humanabad.**

*****Dr.Basavaprasad.B. Assistant Professor and HOD of Computer Science, Govt First Grade College, Raichur.**

Abstract

This paper attempts to study **software engineering concepts,** which are developed in to **models for better product deployment**. There are many different types of software systems from simple to complex systems. These systems may be developed for a particular customer, like systems to support a particular business process, or developed for a general purpose, like any software for our computers such as word processors.

Many systems are now being built with a generic product as base, which is then adapted to suit the requirements of a customer such as SAP system. A good software should deliver the main required functionality. Other set of attributes — called quality or non-functional — should be also delivered. Examples of these attributes are, the software is written in a way that can be adapted to changes, response time, performance (less use of resources such as memory and processor time), usable; acceptable for the type of the user it's built for, reliable, secure, safe, …etc. Software engineering is an engineering discipline that's applied to the development of software in a *systematic* approach (called a software process). It's the application of theories, methods, and tools to design build a software that meets the specifications efficiently, cost-effectively, and ensuring quality. It's not only concerned with the technical process of building a software, it also includes activities to manage the project, develop tools, methods and theories that support the software production. Not applying software engineering methods results in more expensive, less reliable software, and it can be vital on the long term, as the changes come in, the costs will dramatically increase. Different methods and techniques of software engineering are appropriate for different types of systems. For example, games should be developed using series of prototypes, while critical control systems require a complete analyzable specification to be developed. Computer science focuses on the theory and fundamentals, like algorithms, programming languages, theories of computing, artificial intelligence, and hardware design, while software engineering is concerned with the activities of developing and managing a software. Model driven design develops textual and graphical models as primary design artifacts. Development tools are available that use model transformation and code generation to generate well-organized code fragments that serve as a basis for producing complete applications.

Key words: Software engineering, concepts, models , design artifacts, deployment

Introduction

The first theory about software was proposed by Alan Turing in his 1935 essay *Computable numbers with an application to the Entscheidungs problem (Decision problem)*. The term "software" was first used in print by John W. Tukey in 1958. Colloquially, the term is often used to mean application software. In computer science and software engineering, software is all information processed by computer system, programs and data. The academic fields studying software are computer science and software engineering. From its beginnings in the 1940s, writing software has evolved into a profession concerned with how best to maximize the quality of software and of how to create it. Quality can refer to how maintainable software is, to its stability, speed, usability, testability, readability, size, cost, security, and number of flaws or "bugs", as well as to less measurable qualities like elegance, conciseness, and customer satisfaction, among many other attributes. How best to create high quality software is a separate and controversial problem covering software design principles, so-called "best practices" for writing code, as well as broader management issues such as optimal team size, process, how best to deliver software on time and as quickly as possible, work-place "culture," hiring practices, and so forth. All this falls under the broad rubric of software engineering. Software engineering is a young discipline, and is still developing. The directions in which software engineering is developing include:

Aspects: Aspects help software engineers deal with quality attributes by providing tools to add or remove boilerplate code from many areas in the source code. Aspects describe how all objects or functions should behave in particular circumstances. For example, aspects can add debugging, logging, or locking control into all objects of particular types. Researchers are currently working to understand how to use aspects to design general-purpose code. Related concepts include generative programming and templates.

Agile: Agile software development guides software development projects that evolve rapidly with changing expectations and competitive markets. Proponents of this method believe that heavy, document-driven processes (like TickIT, CMM and ISO 9000) are fading in importance. Some people believe that companies and agencies export many of the jobs that can be guided by heavy-weight processes. Related concepts include extreme programming, scrum, and lean software development.

Objective:

This paper intends to explore and analyze **Software engineering (SE) that** is concerned with developing and maintaining software systems that behave reliably and efficiently Also engineering **practices** for software development, and typically handles the overall system design of the software

Software engineering concepts

Separation of concerns is a recognition of the need for human beings to work within a limited context. As descibed by G. A. Miller [Miller56], the human mind is limited to dealing with approximately seven units of data at a time. A unit is something that a person has learned to deal with as a whole - a single abstraction or concept. Although human capacity for forming abstractions appears to be unlimited, it takes time and repetitive use for an abstraction to become a useful tool; that is, to serve as a unit.

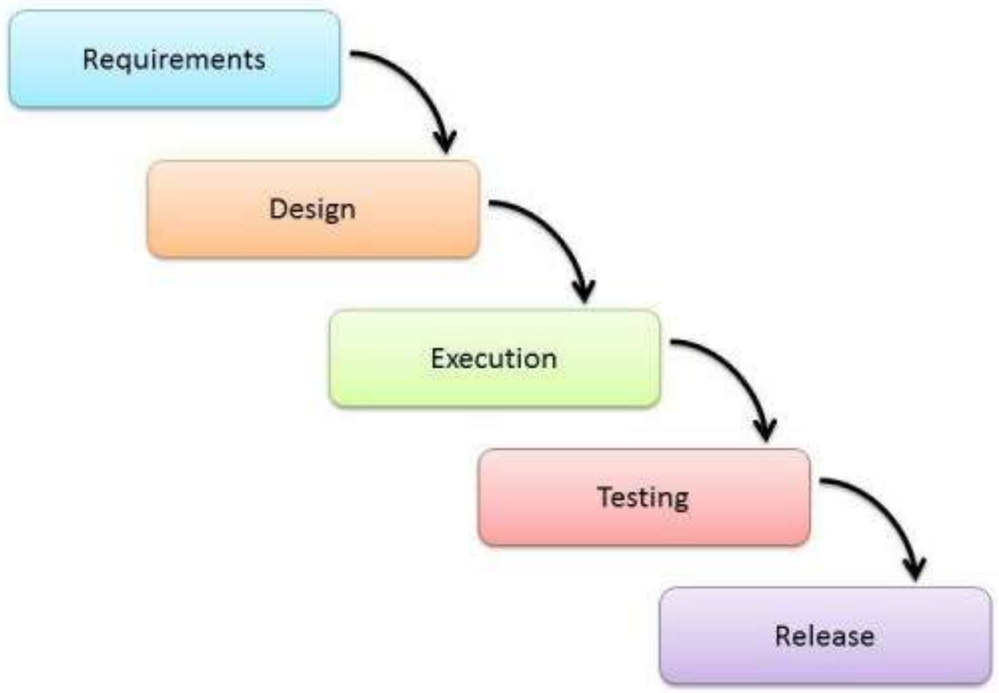
When specifying the behavior of a data structure component, there are often two concerns that need to be dealt with: basic functionality and support for data integrity. A data structure component is often easier to use if these two concerns are divided as much as posible into separate sets of client functions. It is certainly helful to clients if the client documentation treats the two concerns separately. Further, implementation documentation and algorithm descriptions can profit from separate treatment of basic algorithms and modifications for data integrity and exception handling.

There is another reason for the importance of separation of concerns. Software engineers must deal with complex values in attempting to optimize the quality of a product. From the study of algorithmic complexity, we can learn an important lesson. There are often efficient algorithms for optimizing a single measurable quantity, but problems requiring optimization of a combination of quantities are almost always NP-complete. Although it is not a proven fact, most experts in complexity theory believe that NP-complete problems cannot be solved by algorithms that run in polynomial time.

In view of this, it makes sense to separate handling of different values. This can be done either by dealing with different values at different times in the software development process, or by structuring the design so that responsibility for achieving different values is assigned to different components.

Waterfall Model

The Waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach and most widely known that was used for software development.



Waterfall Model

The usage

Projects which not focus on changing the requirements, for example, projects initiated from a request for proposals (RFPs), the customer has a very clear documented requirements

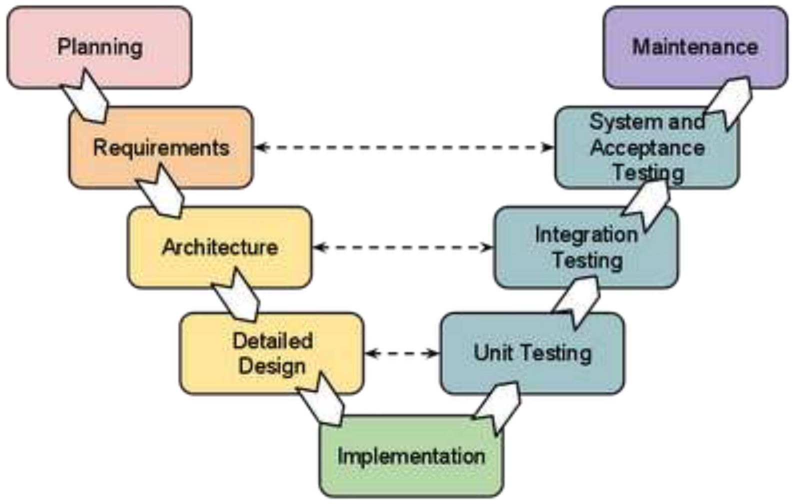
Advantages and Disadvantages

Advantages	Disadvantages
• Easy to explain to the users. • Structures approach. • Stages and activities are well defined. • Helps to plan and schedule the project. • Verification at each stage ensures early detection of errors/misunderstanding. • Each phase has specific deliverables.	• Assumes that the requirements of a system can be frozen. • Very difficult to go back to any stage after it finished. • A little flexibility and adjusting scope is difficult and expensive. • Costly and required more time, in addition to the detailed plan.

V-Shaped Model

Description

It is an extension of the waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the implementation and coding phase, to form the typical V shape. The major difference between the V-shaped model and waterfall model is the early test planning in the V-shaped model.



The usage

- Software requirements clearly defined and known
- Software development technologies and tools are well-known

Advantages and Disadvantages

Advantages	Disadvantages
• Simple and easy to use • Each phase has specific deliverables. • Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle. • Works well for where requirements are easily understood. • Verification and validation of the product in the early stages of product development.	• Very inflexible, like the waterfall model. • Adjusting scope is difficult and expensive. • The software is developed during the implementation phase, so no early prototypes of the software are produced. • The model doesn’t provide a clear path for problems found during testing phases. • Costly and required more time, in addition to a detailed plan

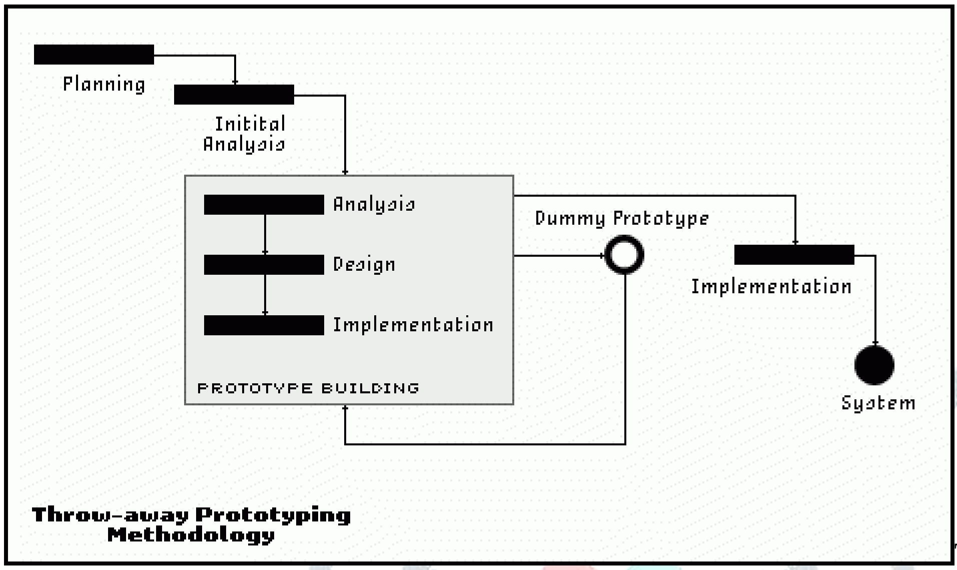
Prototyping Model

Description

It refers to the activity of creating prototypes of software applications, for example, incomplete versions of the software program being developed. It is an activity that can occur in software development and It used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team. This also will reduce the iterations may occur in the waterfall approach and hard to be implemented due to the inflexibility of the waterfall approach. So, when the final prototype is developed, the requirement is considered to be frozen.

It has some types, such as:

- Throwaway prototyping: Prototypes that are eventually discarded rather than becoming a part of the finally delivered software



Throwaway prototyping

- Evolutionary prototyping: prototypes that evolve into the final system through an iterative incorporation of user feedback.

  Evolutionary prototyping
- Incremental prototyping: The final product is built as separate prototypes. In the end, the separate prototypes are merged in an overall design.

  Incremental prototyping
- Extreme prototyping: used in web applications mainly. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase, the services are implemented

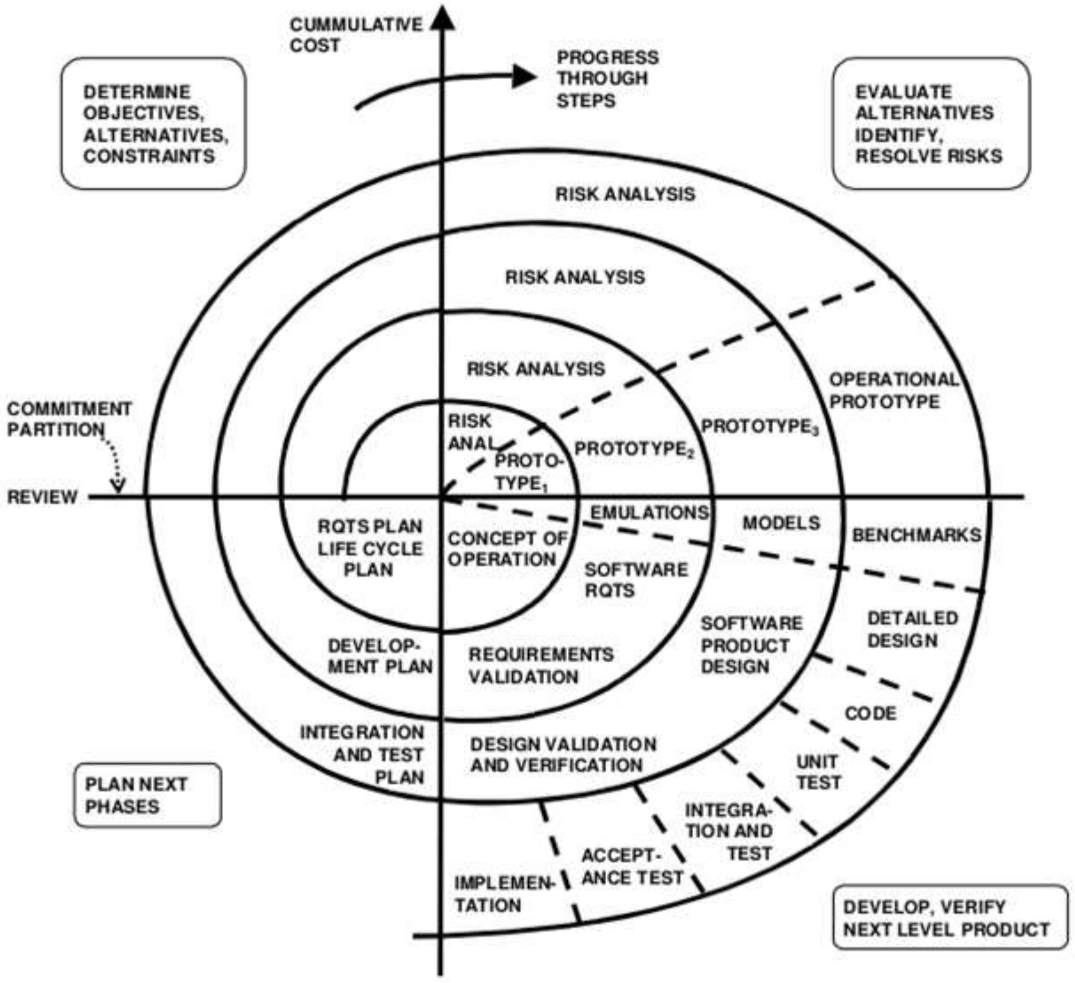
  The usage
- This process can be used with any software developing life cycle model. While this shall be chosen when you are developing a system has user interactions. So, if the system does not have user interactions, such as a system does some calculations shall not have prototypes.

  Advantages and Disadvantages

Spiral Model (SDM)

Description

It is combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. This model of development combines the features of the prototyping model and the waterfall model. The spiral model is favored for large, expensive, and complicated projects. This model uses many of the same phases as the waterfall model, in essentially the same order, separated by planning, risk assessment, and the building of prototypes and simulations.



Spiral Model

The usage

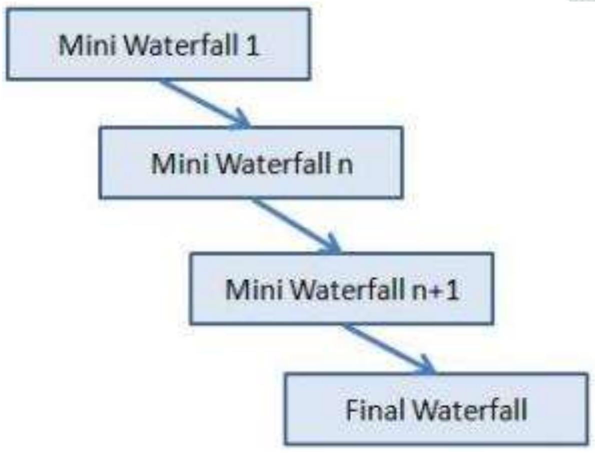
It is used in the large applications and systems which built-in small phases or segments.

Advantages	Disadvantages
• Estimates (i.e. budget, schedule, etc.) become more realistic as work progressed because important issues are discovered earlier. • Early involvement of developers. • Manages risks and develops the system into phases.	• High cost and time to reach the final product. • Needs special skills to evaluate the risks and assumptions. • Highly customized limiting re-usability

Iterative and Incremental Model

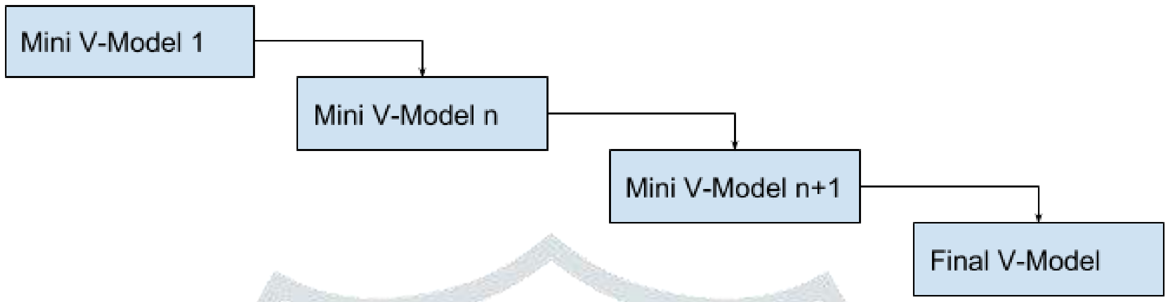
Description

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during the development of earlier parts or versions of the system. It can consist of mini waterfalls or mini V-Shaped model



The usage

It is used in shrink-wrap application and large system which built-in small phases or segments. Also, can be used in a system has separated components, for example, ERP system. Which we can start with the budget module as a first iteration and then we can start with the inventory module and so forth.



Iterative and Incremental Model

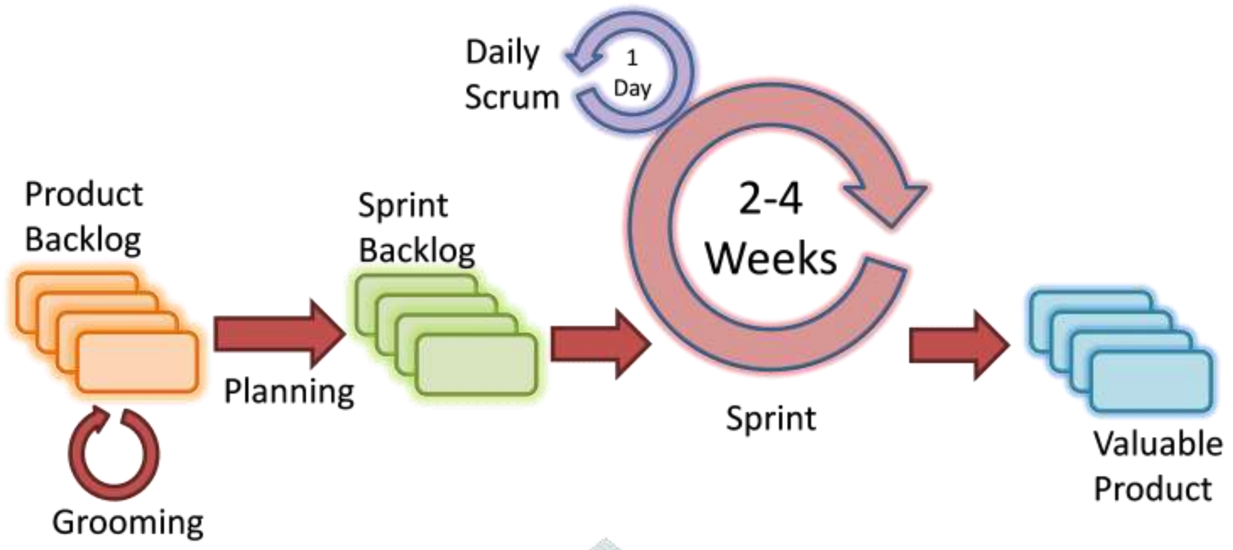
Advantages and Disadvantages

Advantages	Disadvantages
• Produces business value early in the development lifecycle. • Better use of scarce resources through proper increment definition. • Can accommodate some change requests between increments. • More focused on customer value than the linear approaches. • We can detect project issues and changes earlier.	• Requires heavy documentation. • Follows a defined set of processes. • Defines increments based on function and feature dependencies. • Requires more customer involvement than the linear approaches. • Partitioning the functions and features might be problematic. • Integration between the iterations can be an issue if it is not considered during the development and project planning.

Agile Model

Description

It is based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams.



Scrum Agile Model

The usage

It can be used with any type of the project, but it needs more engagement from the customer and to be interactive. Also, we can use it when the customer needs to have some functional requirement ready in less than three weeks and the requirements are not clear enough. This will enable more valuable and workable piece for software early which also increase the customer satisfaction.

Advantages and Disadvantages

Advantages	Disadvantages
• Decrease the time required to avail some system features. • Face to face communication and continuous inputs from customer representative leaves no space for guesswork. • The end result is the high-quality software in the least possible time duration and satisfied customer.	• Scalability. • The ability and collaboration of the customer to express user needs. • Documentation is done at later stages. • Reduce the usability of components. • Needs special skills for the team.

Here is a consolidated presentation to illustrate most of the popular software development models.

The principle of generality is closely related to the principle of anticipation of change. It is important in designing software that is free from unnatural restrictions and limitations. One excellent example of an unnatural restriction or limitation is the use of two digit year numbers, which has led to the "year 2000" problem: software that will garble record keeping at the turn of the century. Although the two-digit limitation appeared reasonable at the time, good software frequently survives beyond its expected lifetime.

For another example where the principle of generality applies, consider a customer who is converting business practices into automated software. They are often trying to satisfy general needs, but they understand and present their needs in terms of their current practices. As they become more familiar with the possibilities of automated solutions, they begin seeing what they need, rather than what they are currently doing to satisfy those needs. This distinction is similar to the distinction in the principle of abstraction, but its effects are felt earlier in the software development process.

Conclusion

Software engineering and development has generated a multitude of methodologies, tools, and practices over the years, but few (if any) have managed to deliver on the value that was promised and more often than not are just shifting complexity or cost from one part of the SDLC to another. A software development life cycle (SDLC) is a framework that provides the process consumed by organizations to build an application from its inception to its end.

The software development life cycle abbreviated as SDLC framework. SDLC framework is a process used for developing software applications or products. It is a procedure performed by the organization and followed step by step to develop and design the right quality product for promoting right from development to production environment in a timely manner.

References

1. Kaner, Cem (November 17, 2006). Exploratory Testing (PDF). Quality Assurance Institute Worldwide Annual Software Testing Conference. Orlando, FL. Retrieved November 22, 2014.
2. 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990, doi:10.1109/IEEESTD.1990.101064, ISBN 9781559370677
3. Pan, Jiantao (Spring 1999). "Software Testing" (coursework). Carnegie Mellon University. Retrieved November 21, 2017.
4. Leitner, Andreas; Ciupa, Ilinca; Oriol, Manuel; Meyer, Bertrand; Fiva, Arno (September 2007). Contract Driven Development = Test Driven Development – Writing Test Cases (PDF). ESEC/FSE'07: European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering 2007. Dubrovnik, Croatia. Retrieved December 8, 2017.
5. Kaner, Cem; Falk, Jack; Nguyen, Hung Quoc (1999). Testing Computer Software (2nd ed.). New York: John Wiley and Sons. ISBN 978-0-471-35846-6.

6. Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. ISBN 978-0-470-04212-0.
7. "Certified Tester Foundation Level Syllabus" (pdf). International Software Testing Qualifications Board. March 31, 2011. Section 1.1.2. Retrieved December 15, 2017.
8. "Certified Tester Foundation Level Syllabus" (PDF). International Software Testing Qualifications Board. July 1, 2005. Principle 2, Section 1.3. Retrieved December 15, 2017.
9. Ramler, Rudolf; Kopetzky, Theodorich; Platz, Wolfgang (April 17, 2012). Combinatorial Test Design in the TOSCA Testsuite: Lessons Learned and Practical Implications. IEEE Fifth International Conference on Software Testing and Validation (ICST). Montreal, QC, Canada. doi:10.1109/ICST.2012.142.
10. "The Economic Impacts of Inadequate Infrastructure for Software Testing" (PDF). National Institute of Standards and Technology. May 2002. Retrieved December 19, 2017.
11. Sharma, Bharadwaj (April 2016). "Ardentia Technologies: Providing Cutting Edge Software Solutions and Comprehensive Testing Services". CIO Review (India ed.). Retrieved December 20, 2017.
12. Gelperin, David; Hetzel, Bill (June 1, 1988). "The growth of software testing". Communications of the ACM. 31 (6): 687–695. doi:10.1145/62959.62965. S2CID 14731341.