Software development process

In software engineering, a **software development process** is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a **software development life cycle**. The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.[1]

Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming.

Some people consider a life-cycle "model" a more general term for a category of methodologies and a software development "process" a more specific term to refer to a specific process chosen by a specific organization. For example, there are many specific software development processes that fit the spiral life-cycle model. The field is often considered a subset of the systems development life cycle.

Contents

History

The software development methodology (also known as SDM) framework didn't emerge until the 1960s. According to Elliott (2004) the systems development life cycle (SDLC) can be considered to be the oldest formalized methodology framework for building information systems. The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle––from inception of the idea to delivery of the final system––to be carried out rigidly and sequentially"[2] within the context of the framework being applied. The main target of this methodology framework in the 1960s was "to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines".[2]

Methodologies, processes, and frameworks range from specific proscriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a "sponsor" or "maintenance" organization distributes an official set of documents that describe the process. Specific examples include:

1970s

- Structured programming since 1969
- Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology

1980s

- Structured systems analysis and design method (SSADM) from 1980 onwards
- Information Requirement Analysis/Soft systems methodology

1990s

- Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s
- Rapid application development (RAD), since 1991
- Dynamic systems development method (DSDM), since 1994
- Scrum, since 1995
- Team software process, since 1998
- Rational Unified Process (RUP), maintained by IBM since 1998
- Extreme programming, since 1999

2000s

- Agile Unified Process (AUP) maintained since 2005 by Scott Ambler
- Disciplined agile delivery (DAD) Supersedes AUP

2010s

- Scaled Agile Framework (SAFe)
- Large-Scale Scrum (LeSS)

It is notable that since DSDM in 1994, all of the methodologies on the above list except RUP have been agile methodologies - yet many organisations, especially governments, still use pre-agile processes (often waterfall or similar). Software process and software quality are closely interrelated; some unexpected facets and effects have been observed in practice [3]

Since the early 2000s scaling agile delivery processes has become the biggest challenge for teams using agile processes.[4]

Among these another software development process has been established in open source. The adoption of these best practices known and established processes within the confines of a company is called inner source.

Practices

Several software development approaches have been used since the origin of information technology, in two main categories. Typically an approach or a combination of approaches is chosen by management or a development team.

"Traditional" methodologies such as waterfall that have distinct phases are sometimes known as **software development life cycle** (SDLC) methodologies [5], though this term could also be used more generally to refer to any methodology. A "life cycle" approach with distinct phases is in contrast to Agile approaches which define a process of iteration, but where design, construction, and deployment of different pieces can occur simultaneously.

Continuous integration

Continuous integration is the practice of merging all developer working copies to a shared mainline several times a day.[6] Grady Booch first named and proposed CI in his 1991 method,[7] although he did not advocate integrating several times a day. Extreme programming (XP) adopted the concept of CI and did advocate integrating more than once per day – perhaps as many as tens of times per day.

Prototyping

Software prototyping is about creating prototypes, i.e. incomplete versions of the software program being developed.

The basic principles are:[1]

- Prototyping is not a standalone, complete development methodology, but rather an approach to try out particular features in the context of a full methodology (such as incremental, spiral, or rapid application development (RAD)).
- Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
- The client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.
- While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problems, but this is true for all software methodologies.

Incremental development

Various methods are acceptable for combining linear and iterative systems development methodologies, with the primary objective of each being to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

There are three main variants of incremental development:[1]

1. A series of mini-Waterfalls are performed, where all phases of the Waterfall are completed for a small part of a system, before proceeding to the next increment, or
2. Overall requirements are defined before proceeding to evolutionary, mini-Waterfall development of individual increments of a system, or
3. The initial software concept, requirements analysis, and design of architecture and system core are defined via Waterfall, followed by incremental implementation, which culminates in installing the final version, a working system.

Rapid application development

Rapid application development (RAD) is a software development methodology, which favors iterative development and the rapid construction of prototypes instead of large amounts of up-front planning. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

Rapid Application Development (RAD) Model

The rapid development process starts with the development of preliminary data models and business process models using structured techniques. In the next stage, requirements are verified using prototyping, eventually to refine the data and process models. These stages are repeated iteratively; further development results in "a combined business requirements and technical design statement to be used for constructing new systems".[8]

The term was first used to describe a software development process introduced by James Martin in 1991. According to Whitten (2003), it is a merger of various structured techniques, especially data-driven information technology engineering, with prototyping techniques to accelerate software systems development.[8]

The basic principles of rapid application development are:[1]

- Key objective is for fast development and delivery of a high quality system at a relatively low investment cost.
- Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
- Aims to produce high quality systems quickly, primarily via iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include Graphical User Interface (GUI) builders, Computer Aided Software Engineering (CASE) tools, Database Management Systems (DBMS), fourth-generation programming languages, code generators, and object-oriented techniques.
- Key emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.
- Project control involves prioritizing development and defining delivery deadlines or “timeboxes”. If the project starts to slip, emphasis is on reducing requirements to fit the timebox, not in increasing the deadline.

- Generally includes joint application design (JAD), where users are intensely involved in system design, via consensus building in either structured workshops, or electronically facilitated interaction.
- Active user involvement is imperative.
- Iteratively produces production software, as opposed to a throwaway prototype.
- Produces documentation necessary to facilitate future development and maintenance.
- Standard systems analysis and design methods can be fitted into this framework.

Methodologies

Agile development

"Agile software development" refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve via collaboration between self-organizing cross-functional teams. The term was coined in the year 2001 when the Agile Manifesto was formulated.

Agile software development uses iterative development as a basis but advocates a lighter and more people-centric viewpoint than traditional approaches. Agile processes fundamentally incorporate iteration and the continuous feedback that it provides to successively refine and deliver a software system.

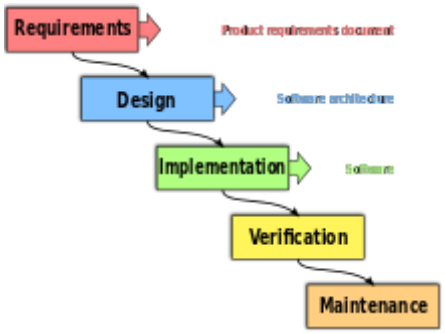
There are many agile methodologies, including:

- Dynamic systems development method (DSDM)
- Kanban
- Scrum

Waterfall development

The waterfall model is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall) through several phases, typically:

- Requirements analysis resulting in a software requirements specification
- Software design
- Implementation
- Testing
- Integration, if there are multiple subsystems
- Deployment (or Installation)
- Maintenance



The activities of the software development process represented in the waterfall model. There are several other models to represent this process.

The first formal description of the method is often cited as an article published by Winston W. Royce[9] in 1970 although Royce did not use the term "waterfall" in this article. Royce presented this model as an example of a flawed, non-working model.[10]

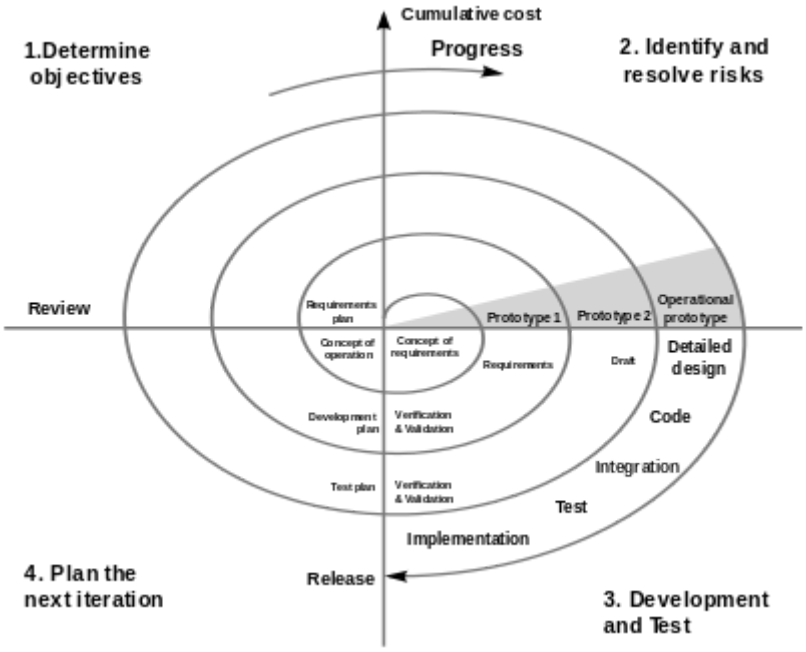
The basic principles are:[1]

- Project is divided into sequential phases, with some overlap and splashback acceptable between phases.
- Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
- Tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase. Written documentation is an explicit deliverable of each phase.

The waterfall model is a traditional engineering approach applied to software engineering. A strict waterfall approach discourages revisiting and revising any prior phase once it is complete. This "inflexibility" in a pure waterfall model has been a source of criticism by supporters of other more "flexible" models. It has been widely blamed for several large-scale government projects running over budget, over time and sometimes failing to deliver on requirements due to the Big Design Up Front approach. Except when contractually required, the waterfall model has been largely superseded by more flexible and versatile methodologies developed specifically for software development. See Criticism of Waterfall model.

Spiral development

In 1988, Barry Boehm published a formal software system development "spiral model," which combines some key aspect of the waterfall model and rapid prototyping methodologies, in an effort to combine advantages of top-down and bottom-up concepts. It provided emphasis in a key area many felt had been neglected by other methodologies: deliberate iterative risk analysis, particularly suited to large-scale complex systems.



Spiral model (Boehm, 1988)

The basic principles are:[1]

- Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments and providing more ease-of-change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle.
- "Each cycle involves a progression through the same sequence of steps, for each part of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program."[11]
- Each trip around the spiral traverses four basic quadrants: (1) determine objectives, alternatives, and constraints of the iteration; (2) evaluate alternatives; Identify and resolve risks; (3) develop and verify deliverables from the iteration; and (4) plan the next iteration.[12]
- Begin each cycle with an identification of stakeholders and their "win conditions", and end each cycle with review and commitment.[13]

Other

Other high-level software project methodologies include:

- Behavior-driven development and business process management[14]
- Chaos model - The main rule is always resolve the most important issue first.
- Incremental funding methodology - an iterative approach
- Lightweight methodology - a general term for methods that only have a few rules and practices
- Structured systems analysis and design method - a specific version of waterfall
- Slow programming, as part of the larger Slow Movement, emphasizes careful and gradual work without (or minimal) time pressures. Slow programming aims to avoid bugs and overly quick release schedules.
- V-Model (software development) - an extension of the waterfall model
- Unified Process (UP) is an iterative software development methodology framework, based on Unified Modeling Language (UML). UP organizes the development of software into four phases, each consisting of one or more executable iterations of the software at that stage of development: inception, elaboration, construction, and guidelines. Many tools and products exist to facilitate UP implementation. One of the more popular versions of UP is the Rational Unified Process (RUP).

Process meta-models

Some "process models" are abstract descriptions for evaluating, comparing, and improving the specific process adopted by an organization.

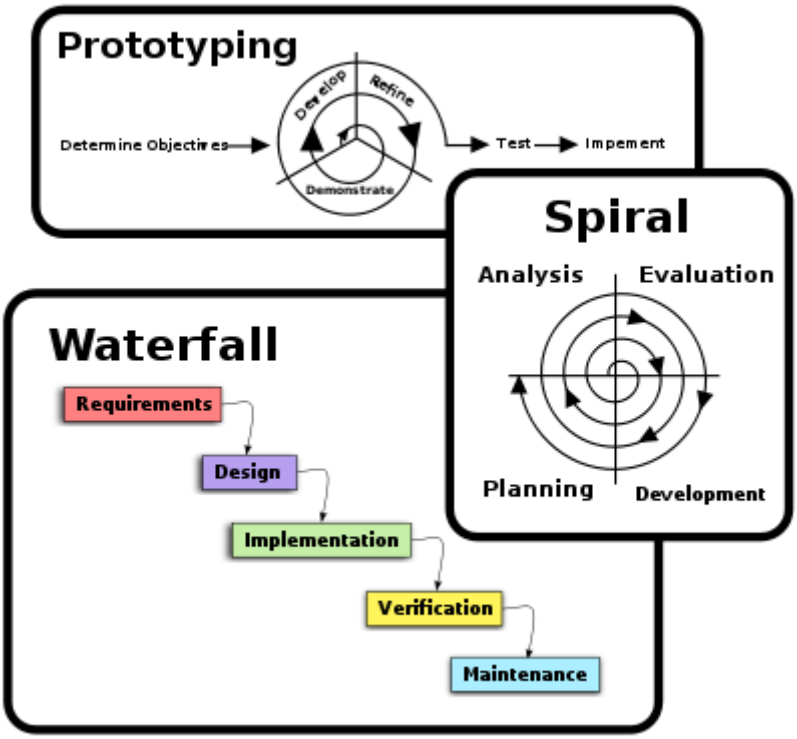
- ISO/IEC 12207 is the international standard describing the method to select, implement, and monitor the life cycle for software.
- The Capability Maturity Model Integration (CMMI) is one of the leading models and based on best practice. Independent assessments grade organizations on how well they follow their defined processes, not on the quality of those processes or the software produced. CMMI has replaced CMM.
- ISO 9000 describes standards for a formally organized process to manufacture a product and the methods of managing and monitoring progress. Although the standard was originally created for the manufacturing sector, ISO 9000 standards have been applied to software development as well. Like CMMI, certification with ISO 9000 does not guarantee the quality of the end result, only that formalized business processes have been followed.
- ISO/IEC 15504 *Information technology — Process assessment* also known as Software Process Improvement Capability Determination (SPICE), is a "framework for the assessment of software processes". This standard is aimed at setting out a clear model for process comparison. SPICE is used much like CMMI. It models processes to manage, control, guide and

monitor software development. This model is then used to measure what a development organization or project team actually does during software development. This information is analyzed to identify weaknesses and drive improvement. It also identifies strengths that can be continued or integrated into common practice for that organization or team.

- SPEM 2.0 by the Object Management Group
- Soft systems methodology - a general method for improving management processes
- Method engineering - a general method for improving information system processes

In practice

A variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. One software development methodology framework is not necessarily suitable for use by all projects. Each of the available methodology frameworks are best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.[1]



The three basic approaches applied to software development methodology frameworks.

Software development organizations implement process methodologies to ease the process of development. Sometimes, contractors may require methodologies employed, an example is the U.S. defense industry, which requires a rating based on process models to obtain contracts. The international standard for describing the method of selecting, implementing and monitoring the life cycle for software is ISO/IEC 12207.

A decades-long goal has been to find repeatable, predictable processes that improve productivity and quality. Some try to systematize or formalize the seemingly unruly task of designing software. Others apply project management techniques to designing software. Large numbers of software projects do not meet their expectations in terms of functionality, cost, or delivery schedule - see List of failed and overbudget custom software projects for some notable examples.

Organizations may create a Software Engineering Process Group (SEPG), which is the focal point for process improvement. Composed of line practitioners who have varied skills, the group is at the center of the collaborative effort of everyone in the organization who is involved with software engineering process improvement.

A particular development team may also agree to programming environment details, such as which integrated development environment is used, and one or more dominant programming paradigms, programming style rules, or choice of specific software libraries or software frameworks. These details are generally not dictated by the choice of model or general methodology.

See also

Software development life cycle (SDLC)

- Systems development life cycle
- Computer-aided software engineering (some of these tools support specific methodologies)
- List of software development philosophies
- Outline of software engineering
- OpenUP
- Project management
- Software development
- Software development effort estimation
- Software release life cycle
- Top-down and bottom-up design#Computer science

References

1. Centers for Medicare & Medicaid Services (CMS) Office of Information Service (2008). *Selecting a development approach (http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf)*. Webarticle. United States Department of Health and Human Services (HHS). Re-validated: March 27, 2008. Retrieved 27 Oct 2008.
2. Geoffrey Elliott (2004) *Global Business Information Technology: an integrated systems approach*. Pearson Education. p.87.
3. Suryanarayana, Girish (2015). "Software Process versus Design Quality: Tug of War?" (http://ieeexplore.ieee.org/document/7140652/?arnumber=7140652). *IEEE Software*. **32** (4): 7–11. doi:10.1109/MS.2015.87 (https://doi.org/10.1109%2FMS.2015.87).
4. saeeda, Hina; Khalid, Hannan; Ahmed, Mukhtar; Sameer, Abu; Arif, Fahim (2015-09-01). "Systematic Literature Review of Agile Scalability for Large Scale Projects" (https://www.researchgate.net/publication/283527859_Systematic_Literature_Review_of_Agile_Scalability_for_Large_Scale_Projects?). *ResearchGate*. **6** (9). CiteSeerX 10.1.1.695.4994 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.695.4994). doi:10.14569/IJACSA.2015.060908 (https://doi.org/10.14569%2FIJACSA.2015.060908). ISSN 2156-5570 (https://www.worldcat.org/issn/2156-5570).
5. "Software Development Process" (https://courses.lumenlearning.com/zeliite115/chapter/reading-software-development-process/).
6. "Continuous Integration" (https://www.thoughtworks.com/continuous-integration).
7. Booch, Grady (1991). *Object Oriented Design: With Applications* (https://books.google.com/books?id=w5VQAAAAMAAJ&q=continuous+integration+inauthor:grady+inauthor:booch&dq=continuous+integration+inauthor:grady+inauthor:booch&hl=en&sa=X&ei=0_TxU6TqIMOZyASJ3ICYCQ&ved=0CEQQ6AEwAg). Benjamin Cummings. p. 209. ISBN 9780805300918. Retrieved 18 August 2014.
8. Whitten, Jeffrey L.; Lonnie D. Bentley, Kevin C. Dittman. (2003). *Systems Analysis and Design Methods*. 6th edition. ISBN 0-256-19906-X.
9. Wasserfallmodell > Entstehungskontext (http://cartoon.iguw.tuwien.ac.at/fit/fit01/wasserfall/entstehung.html), Markus Rerych, Institut für Gestaltungs- und Wirkungsforschung, TU-Wien. Accessed on line November 28, 2007.
10. Conrad Weisert, Waterfall methodology: there's no such thing! (http://www.idinews.com/waterfall.html)
11. Barry Boehm (1996)., "A Spiral Model of Software Development and Enhancement (http://doi.acm.org/10.1145/12944.12948)". In: *ACM SIGSOFT Software Engineering Notes* (ACM) 11(4):14-24, August 1986
12. Richard H. Thayer, Barry W. Boehm (1986). *Tutorial: software engineering project management*. Computer Society Press of the IEEE. p.130
13. Barry W. Boehm (2000). *Software cost estimation with Cocomo II: Volume 1*.
14. Lübke, Daniel; van Lessen, Tammo (2016). "Modeling Test Cases in BPMN for Behavior-Driven Development" (http://ieeexplore.ieee.org/document/7548916/). *IEEE Software*. **33** (5): 15–21. doi:10.1109/MS.2016.117 (https://doi.org/10.1109%2FMS.2016.117).

External links

- Selecting a development approach (http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf) at cms.hhs.gov.
- Gerhard Fischer, "The Software Technology of the 21st Century: From Software Reuse to Collaborative Software Design" (http://l3d.cs.colorado.edu/~gerhard/papers/isfst2001.pdf), 2001
- Subway map of agile practices at Agile Alliance (https://www.agilealliance.org/agile101/subway-map-to-agile-practices/)