Waterfall model

The **waterfall model** is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks.[1] The approach is typical for certain areas of engineering design. In software development,[1] it tends to be among the less iterative and flexible approaches, as progress flows in largely one direction ("downwards" like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, deployment and maintenance.

The waterfall development model originated in the manufacturing and construction industries; where the highly structured physical environments meant that design changes became prohibitively expensive much sooner in the development process. When first adopted for software development, there were no recognised alternatives for knowledge-based creative work.[2]

Contents

History

The first known presentation describing use of such phases in software engineering was held by Herbert D. Benington at the Symposium on Advanced Programming Methods for Digital Computers on 29 June 1956.[3] This presentation was about the development of software for SAGE. In 1983 the paper was republished with a foreword by Benington explaining that the phases were on purpose organised according to the specialisation of tasks, and pointing out that the process was not in fact performed in a strict top-down fashion, but depended on a prototype.[2]

Although the term "waterfall" is not used in the paper, the first formal detailed diagram of the process later known as the "waterfall model" is often cited as a 1970 article by Winston W. Royce.[4][5][6] However he also felt it had major flaws stemming from the fact that testing only happened at the end of the process, which he described as being "risky and invites failure".[4] The rest of his paper introduced five steps which he felt were necessary to "eliminate most of the development risks" associated with the unaltered waterfall approach.[4]

Royce's five additional steps (which included writing complete documentation at various stages of development) never took mainstream hold, but his diagram of what he considered a flawed process became the starting point when describing a "waterfall" approach.[7]

The earliest use of the term "waterfall" may have been in a 1976 paper by Bell and Thayer.[8]

In 1985, the United States Department of Defense captured this approach in DOD-STD-2167A, their standards for working with software development contractors, which stated that "the contractor shall implement a software development cycle that includes the following six phases: Software Requirement Analysis, Preliminary Design, Detailed Design, Coding and Unit Testing, Integration, and Testing".[9]

Model

In Royce's original waterfall model, the following phases are followed in order:

1. System and software requirements: captured in a product requirements document
2. Analysis: resulting in models, schema, and business rules
3. Design: resulting in the software architecture
4. Coding: the development, proving, and integration of software
5. Testing: the systematic discovery and debugging of defects
6. Operations: the installation, migration, support, and maintenance of complete systems

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is reviewed and verified.

Various modified waterfall models (including Royce's final model), however, can include slight or major variations on this process.[4] These variations included returning to the previous cycle after flaws were found downstream, or returning all the way to the design phase if downstream phases deemed insufficient.

Supporting arguments

Time spent early in the software production cycle can reduce costs at later stages. For example, a problem found in the early stages (such as requirements specification) is cheaper to fix than the same bug found later on in the process (by a factor of 50 to 200).[10]

In common practice, waterfall methodologies result in a project schedule with 20–40% of the time invested for the first two phases, 30–40% of the time to coding, and the rest dedicated to testing and implementation. The actual project organisation needs to be highly structured. Most medium and large projects will include a detailed set of procedures and controls, which regulate every process on the project.[11]

A further argument for the waterfall model is that it places emphasis on documentation (such as requirements documents and design documents) as well as source code. In less thoroughly designed and documented methodologies, knowledge is lost if team members leave before the project is completed, and it may be difficult for a project to recover from the loss. If a fully working design document is present (as is the intent of Big Design Up Front and the waterfall model), new team members or even entirely new teams should be able to familiarise themselves by reading the documents.[12]

The waterfall model provides a structured approach; the model itself progresses linearly through discrete, easily understandable and explainable phases and thus is easy to understand; it also provides easily identifiable milestones in the development process. It is perhaps for this reason that the waterfall model is

used as a beginning example of a development model in many software engineering texts and courses.[13]

Criticism

Clients may not know exactly what their requirements are before they see working software and so change their requirements, leading to redesign, redevelopment, and retesting, and increased costs.[14]

Designers may not be aware of future difficulties when designing a new software product or feature, in which case it is better to revise the design than persist in a design that does not account for any newly discovered constraints, requirements, or problems.[15]

Organisations may attempt to deal with a lack of concrete requirements from clients by employing systems analysts to examine existing manual systems and analyse what they do and how they might be replaced. However, in practice, it is difficult to sustain a strict separation between systems analysis and programming.[16] This is because implementing any non-trivial system will almost inevitably expose issues and edge cases that the systems analyst did not consider.

In response to the perceived problems with the *pure* waterfall model, modified waterfall models were introduced, such as "Sashimi (Waterfall with Overlapping Phases), Waterfall with Subprojects, and Waterfall with Risk Reduction".[10]

Some organisations, such as the United States Department of Defense, now have a stated preference against waterfall-type methodologies, starting with MIL-STD-498, which encourages *evolutionary acquisition* and *Iterative and Incremental Development*.[17]

While advocates of agile software development argue the waterfall model is an ineffective process for developing software, some sceptics suggest that the waterfall model is a false argument used purely to market *alternative* development methodologies.[18]

Rational Unified Process (RUP) phases acknowledge the programmatic need for milestones, for keeping a project on track, but encourage iterations (especially within Disciplines) within the Phases. RUP Phases are often referred to as "waterfall-like".

Modified waterfall models

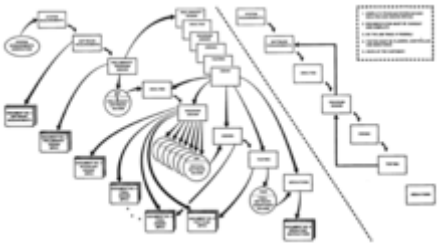
In response to the perceived problems with the "pure" waterfall model, many **modified waterfall models** have been introduced. These models may address some or all of the criticisms of the "pure" waterfall model.

These include the Rapid Development models that Steve McConnell calls "modified waterfalls":[10] Peter DeGrace's "sashimi model" (waterfall with overlapping phases), waterfall with subprojects, and waterfall with risk reduction. Other software development model combinations such as "incremental waterfall model" also exist.[19]

Royce's final model

Winston W. Royce's final model, his intended improvement upon his initial "waterfall model", illustrated that feedback could (should, and often would) lead from code testing to design (as testing of code uncovered flaws in the design) and from design back to requirements specification (as design problems may necessitate the removal of conflicting or otherwise unsatisfiable / undesignable requirements). In the same

paper Royce also advocated large quantities of documentation, doing the job "twice if possible" (a sentiment similar to that of Fred Brooks, famous for writing the Mythical Man Month, an influential book in software project management, who advocated planning to "throw one away"), and involving the customer as much as possible (a sentiment similar to that of extreme programming).



Royce final model

Royce notes to the final model are:

1. Complete program design before analysis and coding begins
2. Documentation must be current and complete
3. Do the job twice if possible
4. Testing must be planned, controlled and monitored
5. Involve the customer

See also

- List of software development philosophies
- Agile software development
- Big Design Up Front
- Chaos model
- DevOps
- Iterative and incremental development
- Object-oriented analysis and design
- Rapid application development
- Software development process
- Spiral model
- Structured Systems Analysis and Design Method (SSADM)
- System development methodology
- Traditional engineering
- V-model

References

1. Petersen, Kai; Wohlin, Claes; Baca, Dejan (2009). Bomarius, Frank; Oivo, Markku; Jaring, Päivi; Abrahamsson, Pekka (eds.). "The Waterfall Model in Large-Scale Development" (https://link.springer.com/chapter/10.1007/978-3-642-02152-7_29). *Product-Focused Software Process Improvement*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer: 386–400. doi:10.1007/978-3-642-02152-7_29 (https://doi.org/10.1007%2F978-3-642-02152-7_29). ISBN 978-3-642-02152-7.
2. Benington, Herbert D. (1 October 1983). "Production of Large Computer Programs" (http://sunset.usc.edu/csse/TECHRPTS/1983/usccse83-501/usccse83-501.pdf) (PDF). *IEEE Annals of the History of Computing*. IEEE Educational Activities Department. **5** (4): 350–361. doi:10.1109/MAHC.1983.10102 (https://doi.org/10.1109%2FMAHC.1983.10102). S2CID 8632276 (https://api.semanticscholar.org/CorpusID:8632276). Retrieved 2011-03-21. Archived (https://web.archive.org/web/20110718084251/http://sunset.usc.edu/csse/TECHRPTS/1983/usccse83-501/usccse83-501.pdf) July 18, 2011, at the Wayback Machine

3. United States, Navy Mathematical Computing Advisory Panel (29 June 1956), *Symposium on advanced programming methods for digital computers*, [Washington, D.C.]: Office of Naval Research, Dept. of the Navy, OCLC 10794738 (https://www.worldcat.org/oclc/10794738)
4. Royce, Winston (1970), "Managing the Development of Large Software Systems" (http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf) (PDF), *Proceedings of IEEE WESCON*, **26** (August): 1–9
5. "Waterfall" (http://www.informatik.uni-bremen.de/uniform/vm97/def/def_w/WATERFALL.htm). *Bremen University - Mathematics and Computer Science*.
6. Abbas, Noura; Gravell, Andrew M.; Wills, Gary B. (2008). Abrahamsson, Pekka; Baskerville, Richard; Conboy, Kieran; Fitzgerald, Brian; Morgan, Lorraine; Wang, Xiaofeng (eds.). "Historical Roots of Agile Methods: Where Did "Agile Thinking" Come From?" (https://link.springer.com/chapter/10.1007/978-3-540-68255-4_10). *Agile Processes in Software Engineering and Extreme Programming*. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer: 94–103. doi:10.1007/978-3-540-68255-4_10 (https://doi.org/10.1007%2F978-3-540-68255-4_10). ISBN 978-3-540-68255-4.
7. Conrad Weisert, Waterfall methodology: there's no such thing! (http://www.idinews.com/waterfall.html)
8. Bell, Thomas E., and T. A. Thayer. Software requirements: Are they really a problem? (http://pdf.aminer.org/000/361/405/software_requirements_are_they_really_a_problem.pdf) *Proceedings of the 2nd international conference on Software engineering.* IEEE Computer Society Press, 1976.
9. "Military Standard Defense System Software Development" (http://www.product-lifecycle-management.com/download/DOD-STD-2167A.pdf) (PDF).
10. McConnell, Steve (1996). *Rapid Development: Taming Wild Software Schedules* (https://archive.org/details/rapiddevelopment00mcco). Microsoft Press. ISBN 1-55615-900-5.
11. "Waterfall Software Development Model" (http://www.oxagile.com/company/blog/the-waterfall-model/). 5 February 2014. Retrieved 11 August 2014.
12. Arcisphere technologies (2012). "Tutorial: The Software Development Life Cycle (SDLC)" (http://softwarelifecyclepros.com/wp-content/uploads/2012/05/Tutorial-Software-Development-LifeCycle-SDLC.pdf) (PDF). Retrieved 2012-11-13.
13. Hughey, Douglas (2009). "Comparing Traditional Systems Analysis and Design with Agile Methodologies" (http://www.umsl.edu/~hugheyd/is6840/waterfall.html). University of Missouri – St. Louis. Retrieved 11 August 2014.
14. Parnas, David L.; Clements, Paul C. (1986). "A rational design process: How and why to fake it" (https://www.cs.tufts.edu/~nr/cs257/archive/david-parnas/fake-it.pdf) (PDF). *IEEE Transactions on Software Engineering* (2): 251–257. doi:10.1109/TSE.1986.6312940 (https://doi.org/10.1109%2FTSE.1986.6312940). S2CID 5838439 (https://api.semanticscholar.org/CorpusID:5838439). Retrieved 2011-03-21.
15. McConnell, Steve (2004). *Code Complete, 2nd edition* (https://archive.org/details/codecompleteprac00mcco). Microsoft Press. ISBN 1-55615-484-4.
16. Ensmenger, Nathan (2010). *The Computer Boys Take Over* (https://archive.org/details/computerboystake00ensm). p. 42 (https://archive.org/details/computerboystake00ensm/page/n50). ISBN 978-0-262-05093-7.
17. Larman, Craig; Basili, Victir (2003). "Iterative and Incremental Development: A Brief History" (http://doi.ieeecomputersociety.org/10.1109/MC.2003.1204375). *IEEE Computer* (June ed.). **36** (6): 47–56. doi:10.1109/MC.2003.1204375 (https://doi.org/10.1109%2FMC.2003.1204375). S2CID 9240477 (https://api.semanticscholar.org/CorpusID:9240477).

18. A Waterfall Systems Development Methodology … Seriously? (http://get.syr.edu/news_alt.aspx?recid=401) by David Dischave. 2012. Archived (https://web.archive.org/web/20140702113551/http://get.syr.edu/news_alt.aspx?recid=401) July 2, 2014, at the Wayback Machine
19. "Methodology:design methods" (http://myprojects.kostigoff.net/methodology/development_models/development_models.htm).

External links

- Understanding the pros and cons of the Waterfall Model of software development (https://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/)
- Project lifecycle models: how they differ and when to use them (http://www.business-esolutions.com/islm.htm)
- Going Over the Waterfall with the RUP (http://www-128.ibm.com/developerworks/rational/library/4626.html) by Philippe Kruchten
- CSC and IBM Rational join to deliver C-RUP and support rapid business change (http://www.ibm.com/developerworks/rational/library/3012.html)
- c2:WaterFall