Waterfall model

From Wikipedia, the free encyclopedia

The **waterfall model** is a sequential software development model (a process for the creation of software) in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing (validation), integration, and maintenance. The origin of the term "waterfall" is often cited to be an article published in 1970 by W. W. Royce. Royce himself advocated an iterative approach to software development and did not even use the term "waterfall."

Software development process

Activities and steps

Requirements | Architecture | Implementation | Testing | Deployment

Models

Agile | Cleanroom | Iterative | RAD | RUP | Spiral | **Waterfall** | XP

Supporting disciplines

Configuration management | Documentation | Project management | User experience design

Contents

History of the waterfall model

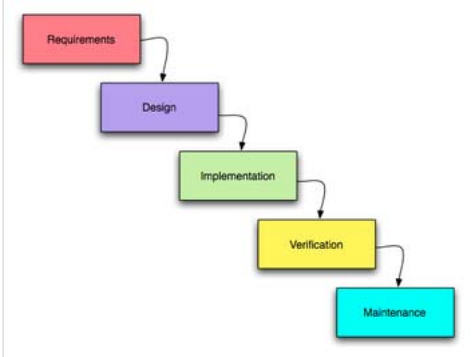
In 1970 Royce proposed what is presently referred to as the waterfall model as an initial concept, a model which he argued was flawed (Royce 1970). His paper then explored how the initial model could be developed into an iterative model, with feedback from each phase influencing subsequent phases, similar to many methods used widely and highly regarded by many today. It is only the initial model that received notice; his own criticism of this initial model has been largely ignored. The "waterfall model" quickly came to refer not to Royce's final, iterative design, but rather to his purely sequentially ordered model. *This article will use this popular meaning of the phrase waterfall model*. For an iterative model similar to Royce's final vision, see the spiral model.

Despite Royce's intentions for the waterfall model to be modified into an iterative model, use of the "waterfall model" as a purely sequential process is still popular, and, for some, the phrase "waterfall model" has since come to refer to any approach to software creation which is seen as inflexible and non-iterative. Those who use the phrase waterfall model pejoratively for non-iterative models that they dislike usually see the waterfall model itself as naive and unsuitable for a "real world" process.

Usage of the waterfall model

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

1. Requirements specification
2. Design
3. Construction (aka: implementation or coding)
4. Integration
5. Testing and debugging (aka: verification)
6. Installation
7. Maintenance



The unmodified "waterfall model". Progress flows from the top to the bottom, like a waterfall.

To follow the waterfall model, one proceeds from one phase to the next in a purely sequential manner. For example, one first completes "requirements specification" — they set in stone the requirements of the software. (Example requirements for Wikipedia may be "Wikipedia allows anonymous editing of articles; Wikipedia enables people to search for information", although real requirements specifications will be much more complex and detailed.) When the requirements are fully completed, one proceeds to design. The software in question is designed and a "blueprint" is drawn for implementers (coders) to follow — this design should be a plan for implementing the requirements given. When the design is fully completed, an implementation of that design is made by coders. Towards the later stages of this implementation phase, disparate software components produced by different teams are integrated. (For example, one team may have been working on the "web page" component of Wikipedia and another team may have been working on the "server" component of Wikipedia. These components must be integrated together to produce the whole system.) After the implementation and integration phases are complete, the software product is tested and debugged; any faults introduced in earlier phases are removed here. Then the software product is installed, and later maintained to introduce new functionality and remove bugs.

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is completed and perfected. Phases of development in the waterfall model are thus discrete, and there is no jumping back and forth or overlap between them.

However, there are various modified waterfall models (including Royce's final model) that may include slight or major variations upon this process.

Arguments for the waterfall model

Time spent early on in software production can lead to greater economy later on in the software lifecycle; that is, it has been shown many times that a bug found in the early stages of the production lifecycle (such as requirements specification or design) is more economical (cheaper in terms of money, effort and time) to fix than the same bug found later on in the process. ([McConnell 1996], p. 72, estimates that "a requirements defect that is left undetected until construction or maintenance will cost 50 to 200 times as much to fix as it would have cost to fix at requirements time.") This should be obvious to some people; if a program design is impossible to implement, it is easier to fix the design at the design stage than to realize months down the track when program components are being integrated that all the work done so far has to be scrapped because of a broken design.

This is the central idea behind Big Design Up Front
(BDUF) and the waterfall model - time spent early on making sure that requirements and design are absolutely correct is very useful in economic terms (it will save you much time and effort later). Thus, the thinking of those who follow the waterfall process goes, one should make sure that each phase is 100% complete and absolutely correct before proceeding to the next phase of program creation. Program requirements should be set in stone before design is started (otherwise work put into a design based on "incorrect" requirements is wasted); the programs design should be perfect before people begin work on implementing the design (otherwise they are implementing the "wrong" design and their work is wasted), etcetera.

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. More "agile" methodologies can de-emphasize documentation in favour of producing working code - documentation however can be useful as a "partial deliverable" should a project not run far enough to produce any substantial amounts of source code (allowing the project to be resumed at a later date). An argument against agile development methods, and thus partly in favour of the waterfall model, is that in agile methods project knowledge is stored mentally by team members. Should team members leave, this knowledge is lost, and substantial loss of project knowledge may be difficult for a project to recover from. Should a fully working design document be present (as is the intent of Big Design Up Front and the waterfall model) new team members or even entirely new teams should *theoretically* be able to bring themselves "up to speed" by reading the documents themselves. With that said, agile methods do attempt to compensate for this. For example, extreme programming (XP) advises that project team members should be "rotated" through sections of work in order to familiarize all members with all sections of the project (allowing individual members to leave without carrying important knowledge with them).

As well as the above, some prefer the waterfall model for its simple and arguably more disciplined approach. Rather than what the waterfall adherent sees as "chaos" the waterfall model provides a structured approach; the model itself progresses linearly through discrete, easily understandable and explainable "phases" and is thus easy to understand; it also provides easily markable "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.

It is argued that the waterfall model and Big Design Up Front in general can be suited to software projects which are stable (especially those projects with unchanging requirements, such as with "shrink wrap" software) and where it is possible and likely that designers will be able to fully predict problem areas of the system and produce a *correct*
design before implementation is started. The waterfall model also requires that implementers follow the well made, complete design accurately, ensuring that the integration of the system proceeds smoothly.

The waterfall model is widely used, including by such large software development houses as those employed by the US Department of Defense and NASA (see "the waterfall model (http://www1.jsc.nasa.gov/bu2/PCEHHTML/pceh90.htm) ") and upon many large government projects (see "the standard waterfall model (http://web.archive.org/web/20040403211247/http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm) " on the Internet Archive). Those who use such methods do not always formally distinguish between the "pure" waterfall model and the various modified waterfall models, so it can be difficult to discern exactly which models are being used to what extent.

Steve McConnell
sees the two big advantages of the pure waterfall model as producing a "highly reliable system" and one with a "large growth envelope", but rates it as poor on all other fronts. On the other hand, he views any of several modified waterfall models (described below) as preserving these advantages while also rating as "fair to excellent" on "work[ing] with poorly understood requirements" or "poorly understood architecture" and "provid[ing] management with progress visibility", and rating as "fair" on "manag[ing] risks", being able to "be constrained to a predefined schedule", "allow[ing] for midcourse corrections", and "provid[ing] customer with progress visibility". The only criterion on which he rates a modified waterfall as poor is that it requires sophistication from management and developers. (*Rapid Development*, 156)

Criticism of the waterfall model

The waterfall model however is argued by many to be a bad idea in practice, mainly because of their belief that it is impossible to get one phase of a software product's lifecycle "perfected" before moving on to the next phases and learning from them (or, at least, the belief that this is impossible for any non-trivial program). For example, clients may not be aware of exactly what requirements they want before they see a working prototype and can comment upon it; they may change their requirements constantly, and program designers and implementers may have little control over this. If clients change their requirements after a design is finished, that design must be modified to accommodate the new requirements, invalidating quite a good deal of effort if overly large amounts of time have been invested into "Big Design Up Front". (Thus, methods opposed to the naive waterfall model--such as those used in Agile software development--advocate less reliance on a fixed, static requirements document or design document). Designers may not (or, more likely, cannot) be aware of future implementation difficulties when writing a design for an unimplemented software product. That is, it may become clear in the implementation phase that a particular area of program functionality is extraordinarily difficult to implement. If this is the case, it is better to revise the design than to persist in using a design that was made based on faulty predictions and that does not account for the newly discovered problem areas.

Steve McConnell in *Code Complete* (a book which criticizes the widespread use of the waterfall model) refers to design as a "wicked problem" - a problem whose requirements and limitations cannot be entirely known before completion. The implication is that it is impossible to get one phase of software development "perfected" before time is spent in "reconnaissance" working out exactly where and what the big problems are.

To quote from David Parnas' "a rational design process and how to fake it (PDF) (http://www.ece.utexas.edu/~perry/education/360F/fakeit.pdf) ":

> “Many of the [system's] details only become known to us as we progress in the [system's] implementation. Some of the things that we learn invalidate our design and we must backtrack.”

The idea behind the waterfall model may be "measure twice; cut once", and those opposed to the waterfall model argue that this idea tends to fall apart when the

problem being measured is constantly changing due to requirement modifications and new realizations about the problem itself. The idea behind those who object to the waterfall model may be "time spent in reconnaissance is seldom wasted".

In summary, the criticisms of a non-iterative development approach (such as the waterfall model) are as follows:

- Many software projects must be open to change due to external factors; the majority of software is written as part of a contract with a client, and clients are notorious for changing their stated requirements. Thus the software project must be adaptable, and spending considerable effort in design and implementation based on the idea that requirements will never change is neither adaptable nor realistic in these cases.
- Unless those who specify requirements and those who design the software system in question are highly competent, it is difficult to know exactly what is needed in each phase of the software process before some time is spent in the phase "following" it. That is, feedback from following phases is needed to complete "preceding" phases satisfactorily. For example, the design phase may need feedback from the implementation phase to identify problem design areas. The counter-argument for the waterfall model is that experienced designers may have worked on similar systems before, and so may be able to accurately predict problem areas without time spent prototyping and implementing.
- Constant testing from the design, implementation and verification phases is required to validate the phases preceding them. Constant "prototype design" work is needed to ensure that requirements are non-contradictory and possible to fulfill; constant implementation is needed to find problem areas and inform the design process; constant integration and verification of the implemented code is necessary to ensure that implementation remains on track. The counter-argument for the waterfall model here is that constant implementation and testing to validate the design and requirements is only needed if the introduction of bugs is likely to be a problem. Users of the waterfall model may argue that if designers (et cetera) follow a disciplined process and do not make mistakes that there is no need for constant work in subsequent phases to validate the preceding phases.
- Frequent incremental builds (following the "release early, release often" philosophy) are often needed to build confidence for a software production team and their client.
- It is difficult to estimate time and cost for each phase of the development process without doing some "recon" work in that phase, unless those estimating time and cost are highly experienced with the type of software product in question.
- The waterfall model brings no formal means of exercising management control over a project and planning control and risk management are not covered within the model itself.
- Only a certain number of team members will be qualified for each phase; thus to have "code monkeys" who are only useful for implementation work do nothing while designers "perfect" the design is a waste of resources. A counter-argument to this is that "multiskilled" software engineers should be hired over "specialized" staff.

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. Many different models are covered by Steve McConnell in the "lifecycle planning" chapter of his book *Rapid Development: Taming Wild Software Schedules*.

While all software development models will bear at least some similarity to the waterfall model, as all software development models will incorporate at least some phases similar to those used within the waterfall model, this section will deal with those closest to the waterfall model. For models which apply further differences to the waterfall model, or for radically different models seek general information on the software development process.

Royce's final model

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—now the basis of participatory design and of User Centred Design, a central tenet of Extreme Programming.

The "sashimi" model

The sashimi model (so called because it features overlapping phases, like the overlapping fish of Japanese sashimi) was originated by Peter DeGrace. It is sometimes simply referred to as the "waterfall model with overlapping phases" or "the waterfall model with feedback". Since phases in the sashimi model overlap, information of problem spots can be acted upon during phases of the waterfall model that would typically "precede" others in the pure waterfall model. For example, since the design and implementation phases will overlap in the sashimi model, implementation problems may be discovered during the "design and implementation" phase of the development process. This helps alleviate many of the problems associated with the Big Design Up Front philosophy of the waterfall model.

Other alternative models

The "waterfall model with subprojects" and the "waterfall model with risk reduction" are two other modified versions of the waterfall model. In some ways the spiral model is comparable to the waterfall model; a "waterfall model with iteration."

See also

- Agile software development
- Big Design Up Front
- Chaos model
- Iterative and incremental development
- Rapid application development
- Software development process
- Spiral model
- System Development Methodology, a type of waterfall model
- V-model

References

This article was originally based on material from the Free On-line Dictionary of Computing, which is licensed under the GFDL.

- McConnell, Steve (2006). *Software Estimation: Demystifying the Black Art*. Microsoft Press. ISBN 0-7356-0535-1.
- McConnell, Steve (2004). *Code Complete, 2nd edition*. Microsoft Press. ISBN 1-55615-484-4.
- McConnell, Steve (1996). *Rapid Development: Taming Wild Software Schedules*. Microsoft Press. ISBN 1-55615-900-5.
- Parnas, David, A rational design process and how to fake it (PDF) (http://www.ece.utexas.edu/~perry/education/360F/fakeit.pdf) An influential paper which criticises the idea that software production can occur in perfectly discrete phases.
- Royce, Winston (1970), "Managing the Development of Large Software Systems" (http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf) , *Proceedings of IEEE WESCON* 26(August): 1-9.
- Joel Spolsky on Big Design Up Front (http://www.joelonsoftware.com/articles/AardvarkSpec.html)
- Joel Spolsky - "daily builds are your friend" (http://www.joelonsoftware.com/articles/fog0000000023.html)
- "Why people still believe in the waterfall model" (http://tarmo.fi/blog/2005/09/09/dont-draw-diagrams-of-wrong-practices-or-why-people-still-believe-in-the-waterfall-model/)
- The standard waterfall model for systems development (http://web.archive.org/web/20050310133243/http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm) NASA webpage, archived on Internet Archive March 10, 2005.
- Parametric Cost Estimating Handbook (http://www1.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm) , NASA webpage based on the waterfall model, archived on Internet Archive March 8,2005.

External links

- The pros and cons of the Waterfall Model of software development (http://articles.techrepublic.com.com/5100-3513_11-6118423.html?part=rss&tag=feed&subj=tr#)
- Further criticism of Joels arguments for BDUF and of BDUF in general (http://haacked.com/archive/2005/08/18/9536.aspx)
- (Technical) discussion of the waterfall model on c2.com (http://c2.com/cgi/wiki?WaterFall)
- "Waterfall model considered harmful" (http://www.it-director.com/technology/productivity/content.php?cid=7865)
- Project Lifecycles: waterfall, rapid application development, and all that (http://www.lux-seattle.com/about/whitepapers/waterfall.asp) (A whitepaper on the waterfall model from the Lux group (a software "design, development and documentation" company))
- The waterfall model at whatis.com (http://searchvb.techtarget.com/sDefinition/0,,sid8_gci519580,00.html)
- "The demise of the waterfall model is imminent" and other urban myths (http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=110)
- Project lifecycle models: how they differ and when to use them (http://www.business-esolutions.com/islm.htm)
- Conrad Weisert, Waterfall methodology: there's no such thing! (http://www.idinews.com/waterfall.html)
- A spoof international Waterfall conference (http://www.waterfall2006.com)
- Software Processes (also the Waterfall Model) (http://www.the-software-experts.de/e_dta-sw-process.htm)

Categories: Software development process | Articles with unsourced statements since June 2007 | All articles with unsourced statements

---