Waterfall - The Dark Age of Software Development Has Returned!

September 24, 2023

https://kallokain.blogspot.com/2023/09/waterfall-dark-age-of-software.html

I honestly beleave it iz better tew know nothing than two know what ain't so.
— Everybody's Friend, 1874, by Josh Billings

To my horror, more and more often the past few years, I have seen people advocating for using the Waterfall method of software development. I have also seen it in job descriptions, for example "we use a mix of Agile and Waterfall".

Why am I horrified? Because Waterfall was never intended to become a software development method. Waterfall is now, as it was from the beginning, a way to describe why many large software development projects fail.

There is no scenario where Waterfall is a good approach to software development!

To back up that statement, let's go back to the origin of Waterfall, a whitepaper by Dr. Winston Royce, *Managing the Development of Large Software Systems* from 1970.

The Origin of Waterfall

Note*: The origin of Waterfall goes back quite a bit further than 1970. In another, later article, Waterfall vs. Agile: Battle of the Dunces or a Race to the Bottom, I go back to the 1956 article by Herbert Benington that kicked it all off. Worth noting that Benington did not like Waterfall either.*

I have seen articles by Waterfall advocates referring to Dr. Winston Royce and his paper. In most cases it is obvious the article authors have never read it! The paper does not say what they think it does. It actually says the opposite! Here is an example, from an article advocating Waterfall:

It's a thorough, structured methodology and one that's been around for a long time, because it works.

— The Ultimate Guide…Waterfall Model, ProjectManager.com

Not true! As I will show in this article, Waterfall has been around for a long time because of lack of competence, unfortunate and downright weird misunderstandings, and a badly written US DoD standard.

Dr. Winston W. Royce at the Lockheed Software Technology Center introduced the concept in a paper published in 1970 on his experience developing software for satellites.

— waterfall model, TechTarget article by Ben Lutkevich

The above statement is technically true, but misleading. Royce did introduce the concept in his paper. What both articles I quoted left out, is what Royce wrote about the Waterfall approach. Here is an example:

…the implementation described above is risky and invites failure.

— Managing the Development of Large Software Systems, Dr. Winston Royce, 1970

Just a few lines later in Dr. Royce's paper:

The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

I'd like to draw your attention to the last sentence. It is worth repeating:

In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

Actually, Dr. Royce underestimated the cost overruns with waterfall projects. Even the successful ones cost a lot more than an agile project, if the agile project uses an agile method that supports both Cost of Change reduction, and small batch transfers, like Extreme Programming, one of the Crystal family of methods, or Lean Software Development.

Dr. Royce did believe in breaking a project down into a linear sequence of phases, but, and it is an important but, he understood very well that feedback, and thus making corrections iteratively, are necessary. He understood very well that Waterfall is risky, and that the risk of very large cost overruns is high.

Note that this was in 1970. Object oriented programming languages, and modern software design, had not been invented yet! There were no design patterns, no Domain-Driven Design, no Test-Driven Design, no automated testing, no refactoring, or refactoring patterns. Barry Boehm's groundbreaking work on software economics would not be published until eleven years later. It was ten years before the Lean revolution changed the manufacturing industry, so only a few people, who were either scientists or working at Toyota, understood things like the effects of batch transfer sizes and Work-

In-Process. AI assisted software development existed only in Science-Fiction stories, usually with horrible consequences.

There were PERT (1958) and GANTT (1910-1915) charts. Critical Path (1957) did also exist. That means an important idea, *parallelization of independent tasks*, did exist. However, Waterfall is strictly linear and sequential, so waterfall projects do not take advantage of this.

We will delve into that in more depth, but we are not done with Dr. Royce's paper yet. The waterfall approach described by Dr. Royce, and now advocated as a method, looked like this:

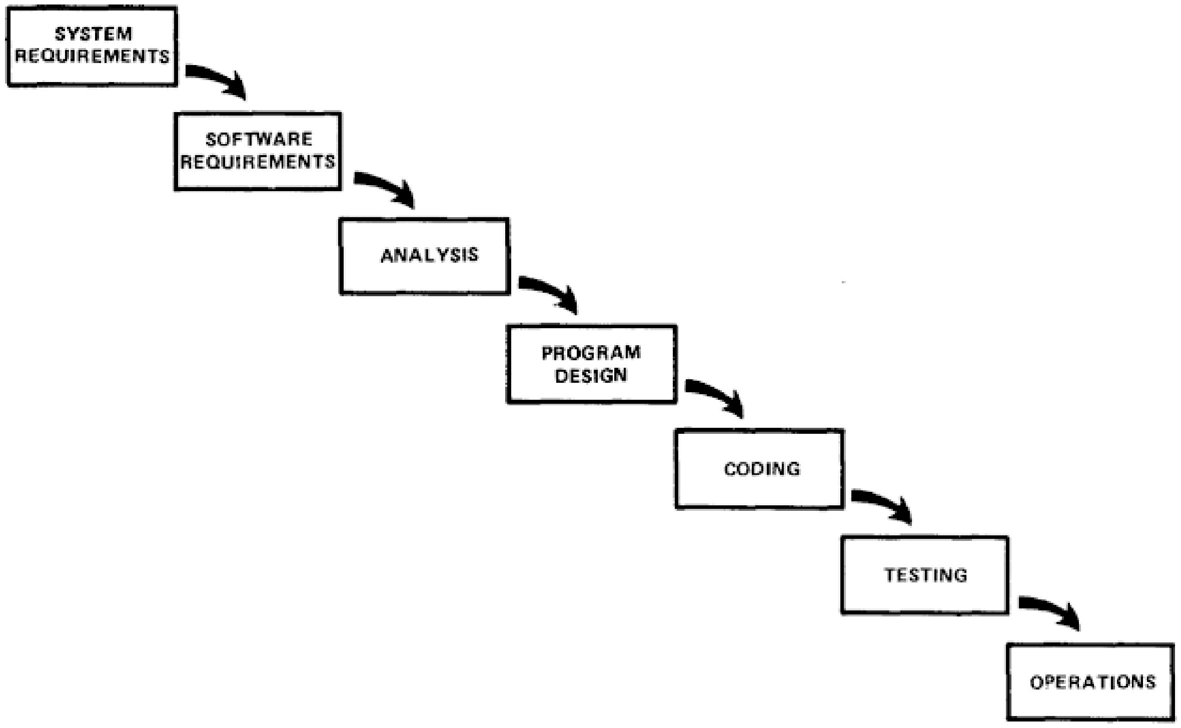


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Figure 2: The picture above shows what Dr. Royce warned about! It is a fundamentally unsound way of developing software that leads to increased lead times, increased risk, and vastly increased cost, even under the best possible conditions. The resulting increased stress makes Waterfall projects horrible working environments.

Dr. Royce's picture of Waterfall projects is on page two in his paper. In the following pages, he presents a succession of more and more sophisticated project models, all involving feedback loops and multiple iterations.

I will not show you all the intermediate versions. Do read Dr. Royce's paper if you want the nitty-gritty details. I will show Dr. Royce's final illustration:

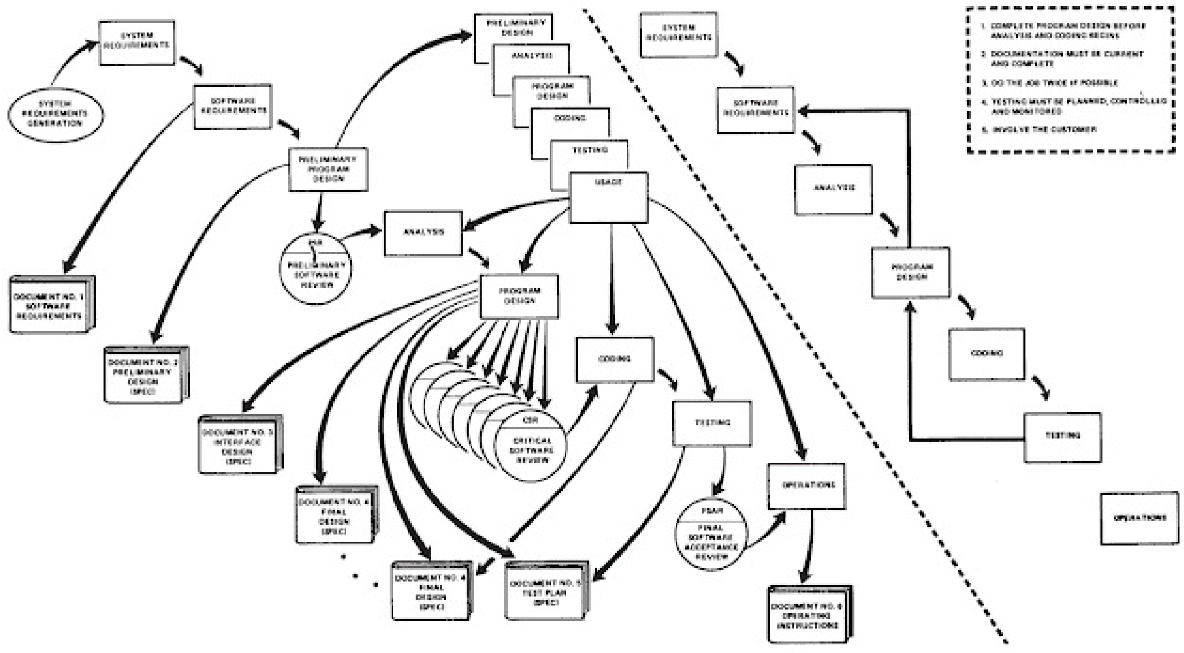

SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
PSR
PRELIMINARY SOFTWARE REVIEW
ANALYSIS
PROGRAM DESIGN
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
CSR
CRITICAL SOFTWARE REVIEW
DOCUMENT NO. 5 TEST PLAN (SPEC)
CODING
TESTING
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS
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

Figure 10. Summary

Figure 3: Dr. Royce's project model. This model has prototyping, and iterative development. It is not anywhere close to Waterfall development!

In Dr. Royce's model, development starts with a preliminary design phase. The purpose of this is to determine constraints, like storage capacity, timing, and other operational constraints. Then, a preliminary version, a prototype, is built.

Prototyping is not done in Waterfall projects. On the other hand, it is done in some agile methods. For example Dynamic Systems Development Method (DSDM), one of the original agile methods, uses rapid prototyping. Lean Software Development and the Crystal family of methods also advocate prototyping. Extreme Programming has a practice called Spike, which sometimes, but not always, entails building prototypes. Even SAFe advocates prototyping, as part of Design Thinking. (In practice, I have never seen a SAFe project that actually uses Design Thinking and prototyping, but it is there in the method description.)

Dr. Royce was also big on involving the customer:

For some reason what a software design is going to do is subject to wide interpretation even after previous agreement. It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery. To give the contractor free rein between requirement definition and operation is inviting trouble.

— Managing the Development of Large Software Systems, Dr. Winston Royce, 1970

The above quote is definitely not Waterfall, where all requirements are gathered up front. It does, however, fit with agile ideas of continuously involving customers.

Dr. Royce's software development model is not agile, but it is much closer to agile than any Waterfall project will ever be.

The Great Misunderstanding

Why on Earth do so many people believe Waterfall is a good model for developing software? I do not have a good answer, at least not one that is polite. However, the misunderstanding goes pretty far back.

Royce never used the word "waterfall" in his paper. One of the first uses, perhaps the very first (though this is uncertain) use of the word in a software development context, is from a 1976 paper on software requirements by T.E. Bell and T.A. Thayer:

The same top-down approach to a series of requirements statements is explained, without the specialized military jargon, in an excellent paper by Royce [5]; he introduced the concept of the "waterfall" of development activities. In this approach software is developed in the disciplined sequence of activities shown in Figure 1.

— Software Requirements: Are They Really a Problem?, T. E. Bell and T. A. Thayer, TRW Defense and Space Systems Group Redondo Beach, California, 1976

Note the reference to Figure 1 in the quote. Let’s look at Figure 1 from the paper:

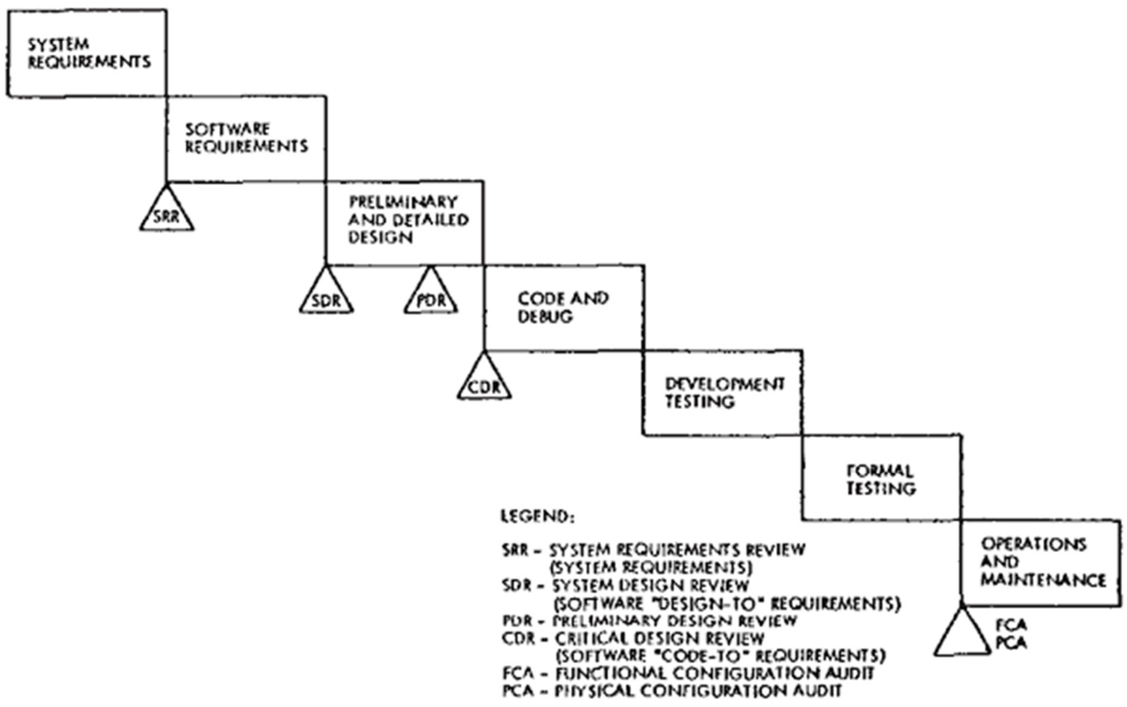

SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
SRR
PRELIMINARY AND DETAILED DESIGN
SDR
PDR
CODE AND DEBUG
CDR
DEVELOPMENT TESTING
FORMAL TESTING
OPERATIONS AND MAINTENANCE
FCA
PCA
LEGEND:
SRR - SYSTEM REQUIREMENTS REVIEW (SYSTEM REQUIREMENTS)
SDR - SYSTEM DESIGN REVIEW (SOFTWARE "DESIGN-TO" REQUIREMENTS)
PDR - PRELIMINARY DESIGN REVIEW
CDR - CRITICAL DESIGN REVIEW (SOFTWARE "CODE-TO" REQUIREMENTS)
FCA - FUNCTIONAL CONFIGURATION AUDIT
PCA - PHYSICAL CONFIGURATION AUDIT

Figure 1. Development Phases of the System Development Cycle

Figure 4: This is the figure in Bell and Thayer’s paper, that purportedly shows the project model Royce advocated. It does not! This is the model Royce warned about!

It’s the wrong diagram! Bell and Thayer copied the wrong diagram from Royce’s paper, and then claimed that this was the idea Royce advocated. As far as I can tell from the paper by Bell and Thayer, they did not even bother to read Royce’s paper. They just looked at the diagram on page two in Royce’s paper and went “Yep, that’s the way to do it!”

We now know that “Waterfall Method” is all a big misunderstanding, but why did something as bonkers as this spread? Enter the US military!

In 1988 the US Department of Defense released a standard for working with software contractors, DOD-STD-2167A. This standard may not have been the best move, at least not for US tax payers.

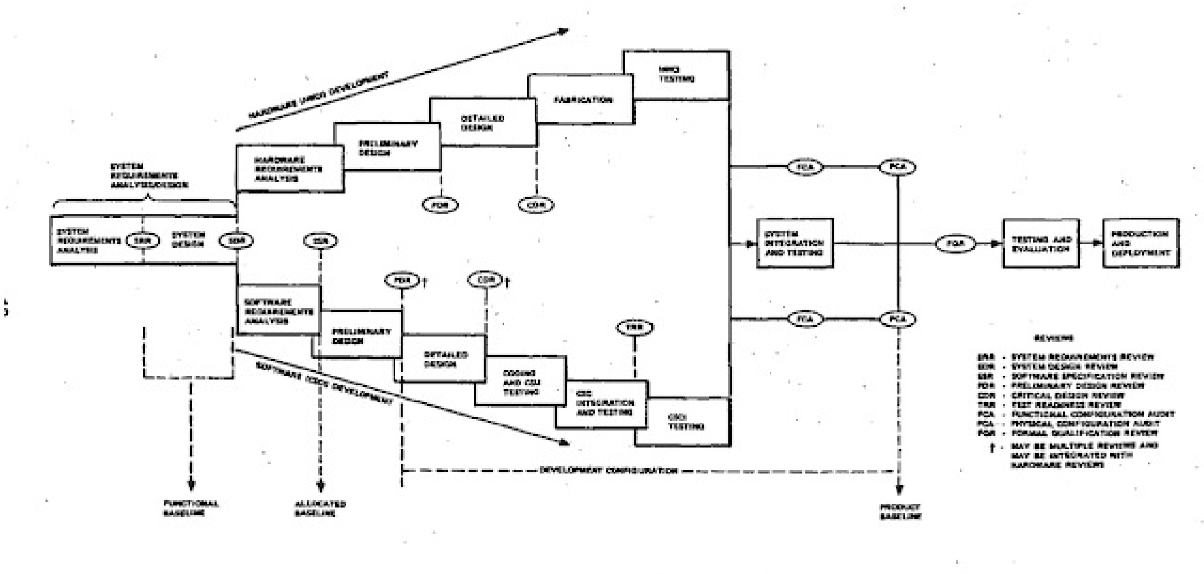
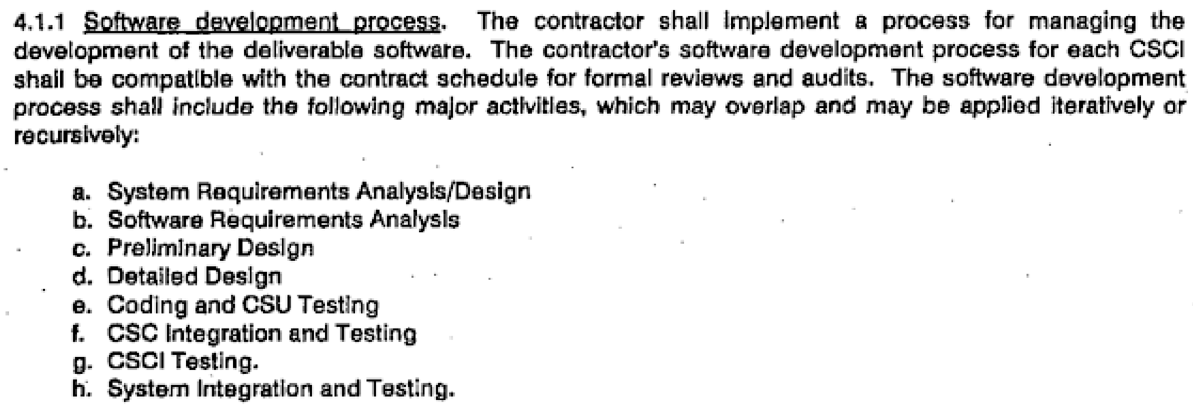

HARDWARE (UNDER DEVELOPMENT)
HWCI TESTING
FABRICATION
DETAILED DESIGN
PRELIMINARY DESIGN
HARDWARE REQUIREMENTS ANALYSIS
SYSTEM REQUIREMENTS ANALYSIS/DESIGN
SYSTEM REQUIREMENTS ANALYSIS
SRR
SYSTEM DESIGN
SDR
SSR
PDR
CDR
FCA
PCA
SYSTEM INTEGRATION AND TESTING
FQR
TESTING AND EVALUATION
PRODUCTION AND DEPLOYMENT
SOFTWARE REQUIREMENTS ANALYSIS
PRELIMINARY DESIGN
DETAILED DESIGN
CODING AND CSU TESTING
CSC INTEGRATION AND TESTING
CSCI TESTING
TRR
SOFTWARE (CSCI DEVELOPMENT)
DEVELOPMENT CONFIGURATION
FUNCTIONAL BASELINE
ALLOCATED BASELINE
PRODUCT BASELINE
REVIEWS
SRR - SYSTEM REQUIREMENTS REVIEW
SDR - SYSTEM DESIGN REVIEW
SSR - SOFTWARE SPECIFICATION REVIEW
PDR - PRELIMINARY DESIGN REVIEW
CDR - CRITICAL DESIGN REVIEW
TRR - TEST READINESS REVIEW
FCA - FUNCTIONAL CONFIGURATION AUDIT
PCA - PHYSICAL CONFIGURATION AUDIT
FQR - FORMAL QUALIFICATION REVIEW
† - MAY BE MULTIPLE REVIEWS AND MAY BE INTEGRATED WITH HARDWARE REVIEWS

FIGURE 1. An example of system development reviews and audits.

Figure 5: The figure above is from the DOD-STS-2167A document. It shows the basic processes for developing both hardware and software, and it is a Waterfall process.

To be fair, the US DoD did not explicitly forbid more sane approaches:



4.1.1 Software development process. The contractor shall implement a process for managing the development of the deliverable software. The contractor's software development process for each CSCI shall be compatible with the contract schedule for formal reviews and audits. The software development process shall include the following major activities, which may overlap and may be applied iteratively or recursively:

a. System Requirements Analysis/Design
b. Software Requirements Analysis
c. Preliminary Design
d. Detailed Design
e. Coding and CSU Testing
f. CSC Integration and Testing
g. CSCI Testing.
h. System Integration and Testing.

Figure 6: As the excerpt shows, the standard allowed iterative and recursive approaches to software development. It did not mandate them though.

It is worth noting that while DOD-STS-2167A does allow iterative and recursive processes, the mandatory formal reviews and audits makes practical implementation more or less impossible. The rules push development processes to be sequential.

Also, from the perspective of a contractor, if you can choose between an approach that require much less training, and which allows you to make a lot of money due to delays and cost overruns, and an approach that will require you to spend money on extra training for your people, and will simultaneously reduce project lead time and cost, so you will make much less money, which approach would you choose?

It's no surprise that the contractors went with getting the most money.

DOD-STS-2167A was criticized, of course, but that didn't help much. For a period of time, Waterfall was the way to do things. Eventually, in the 90's, agile methods emerged as a response to Waterfall, and various Waterfall-inspired approaches. (No, agile did not begin with the Agile Manifesto. The methods existed before that. Take a look at the original authors of the manifesto, and the methods they represented.)

Waterfall vs. Critical Path

So far, we have established that Dr. Winston Royce, unfairly credited with advocating Waterfall, actually advocated replacing Waterfall with an early form of iterative development. We have also established that the whole crackpot idea of Waterfall being a good approach for software development can be traced back to a monumental misunderstanding of the idea in a 1976 whitepaper, and a 1988 military standard of ill repute.

Let's look closer at *why* Waterfall is a bad idea. First of all, Waterfall projects have strictly sequential phases! One type of activity does not start until the previous type of activity ends. This is bad because it becomes difficult, and expensive, to fix mistakes in a phase, if the mistake is discovered in a later phase.

The reason it is so expensive, is that fixing that mistake made in an earlier phase, often triggers a cascade of massive changes. For example, if a mistake in requirements analysis is discovered while testing, this may trigger a redesign of the whole software.

The Waterfall advocates counter argument is that we will use Waterfall only in projects where the requirements are well understood up front. Unfortunately, that argument does not hold up.

Let's assume for the sake of argument, that we have a project where all requirements can be gathered up front. Furthermore, we will assume that all the work, in all the phases, is executed perfectly, with no mistakes. Waterfall still looses big!

Let's look at a sample Waterfall project from an article advocating Waterfall, and then rework it using only a project planning method from the 1960's, Critical Path.

First, let's look at the Waterfall plan from the article. I won't steal their picture of the plan. Instead, I've made a simplified sketch. I did add names to each role though, so I can refer to Anna, rather than "the wireframe builder", and Eve, rather than using the very unappealing term "content creator".

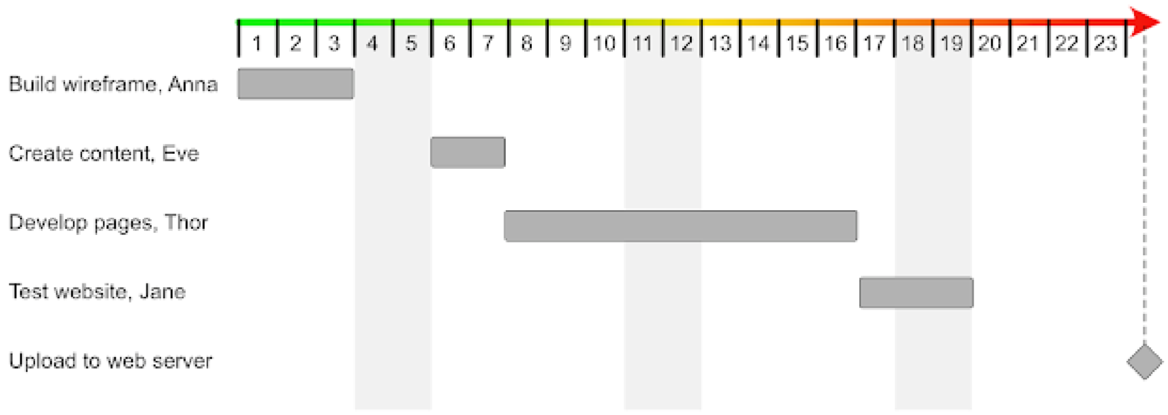


Figure 7: Waterfall project plan from the article What Is Waterfall Project Management?

The first thing to note is that there are some oddities in the original Waterfall plan in the article. For example, team members are expected to work two out of the three weekends. (Weekend days are grayed out.) For another, testing is supposed to be finished on day 19, but nothing is uploaded to the website until day 24.

I do not know if the above oddities are intentional, or planning mistakes. Let's fix this by keeping the estimated times for the activities, but giving everyone weekends off. We will also cut a few days from the schedule by uploading immediately when testing has finished, instead of five days later:

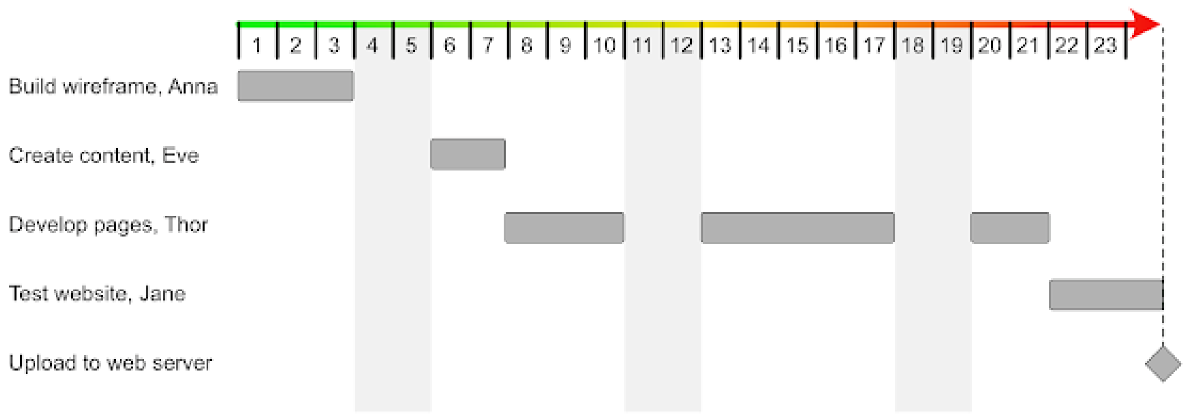


Figure 8: Corrected Waterfall project plan, where team members get the weekends off.

So, now we have a Waterfall plan that does not force team members to work weekends. We have also cut some unnecessary lag time. The net effect is that the project will take 24 calendar days, just as before. We will upload a few hours later on the 24th day though.

One drawback of the Waterfall approach is IOTTMCO (Immediately Obvious To The Most Casual Observer): Because no activity is allowed to begin until the previous activity has finished, the project takes longer than it has to.

For example, Eve, the content creator, is not really dependent on Anna, the wireframe builder (well, designer) to finish building the wireframe version of the pages in order to begin. Actually, Eve probably has ideas that would enable Anna to improve her design, if she got the input. That means Eve and Anna should work together, in parallel, preferably sitting next to each other in the same room.

What about Thor? Is he from Asgard? Yes, but he is also a kick ass web developer. What does he need to get started? He needs an overall idea of the web page design, and the structure of the web site. He most certainly does *not* need a complete set of wireframe models from Anna to get started. Let’s bring Thor in early, so he can join the initial design meeting with Anna and Eve.

What about Jane, the tester? She could start testing as soon as Thor has one web page up and running. It does not even have to be a complete page. She can test functionality as soon as Thor has built it. A tester does more than just checking functionality though. A tester also checks the structure of the web site, to see if the pages are easy to navigate, and that the flow is logical. That means Jane can have useful input even at the wireframe stage. So, bring her in early too. Jane can't finish testing until Thor is done building it, so we will keep her on the job as long as Thor, plus one extra day.

Let's see what the schedule looks like now:

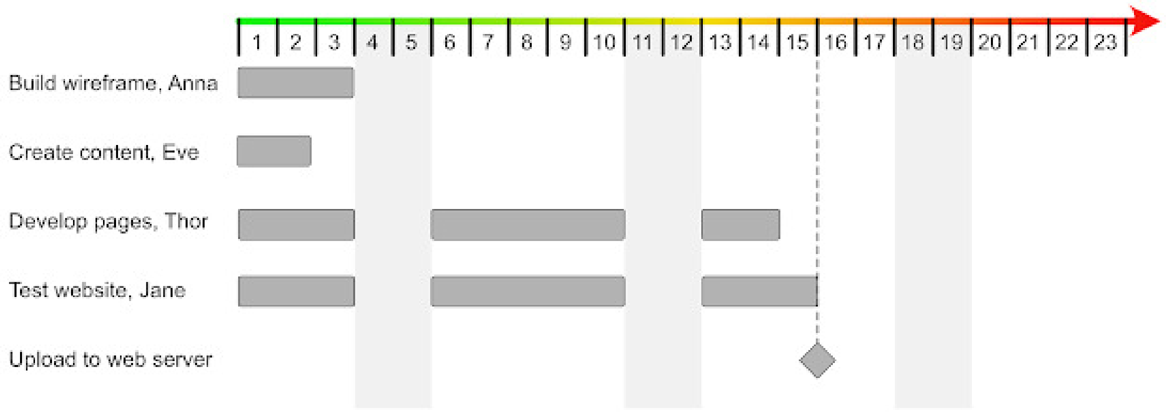


Figure 9: Critical Path Method plan with parallelization of activities. The Critical Path is the longest sequence of dependent activities, in this case 15 days.

Look at that! We now have a 15 day project instead of a 24 day project. We reduced the project lead time with 9 days. *That is a 37.5% (9/24=0.375=37.5%) reduction in lead time*!

To put it another way, if we use Critical Path as the base planning method, and switch to Waterfall, *Waterfall planning delayed the project by 60%* (9/15=0.6=60%).

Note that we did nothing Agile here. We have compared Waterfall planning with the Critical Path Method (CPM), a method that has been around since the late 1950's. The Critical Path is the longest stretch of dependent activities, in this case, Jane's activities. (I have omitted dependencies to each team member's individual work packages, to keep things simple.)

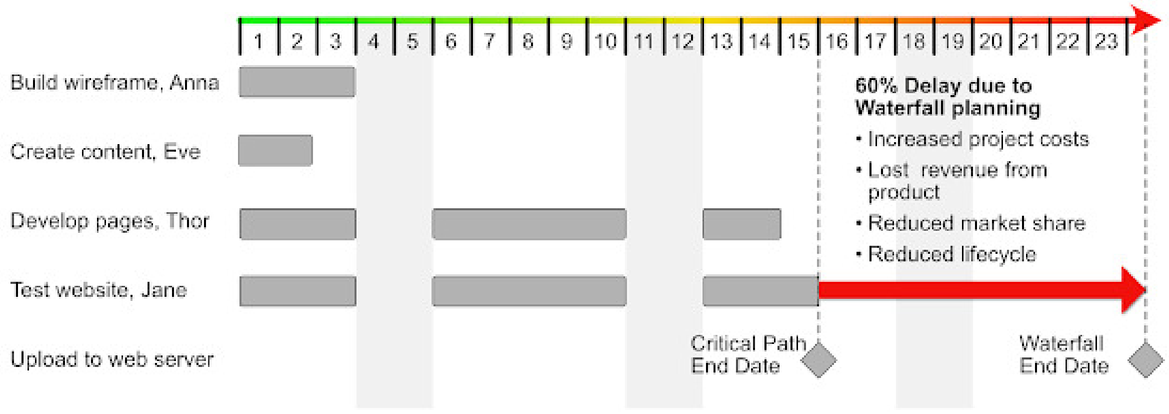


Figure 10: The difference between Critical Path and Waterfall planning leads to increased project costs, lost revenue, reduced market share (and more revenue loss), and shortened lifecycle (and even more revenue loss).

So, compared to Critical Path planning, the Waterfall approach increased delivery time by 60%, or 9 days. That means, the website cannot generate revenue for those 9 days. That may not sound like a lot, but for a large project, a 60% delay can be the difference between a successfully launched project and a market flop.

A delayed project means increased project cost, and loss of revenue due to the delayed launch. Quite often, a delayed launch means reduced market share, and a shorter lifecycle for the product. Add all that up, and it is a lot of money.

In a real, large, project, we can use *Profit & Loss statements* to forecast the *Cost of Delay* we get by using Waterfall instead of Critical Path. Delving into that would require an article of its own though, so I won't.

I should mention, while we are on the topic, that the Critical Path algorithm is quite a bit more complicated than I've shown here. I've simplified things to avoid getting lost in unimportant details. For a large project, there are alternatives to Critical Path, i.e. Critical Chain, that may give more optimal results.

Waterfall vs. Agile

Now that we have used Critical Path, a planning method from the late 1950's to soundly trounce Waterfall, can we trounce Waterfall even more?

It turns out that *sometimes* we can, but not necessarily always. Enter agile methods!

Assuming that we have a well run Critical Path project, agile methods will not enable us to reduce total project lead time, but it may enable us to deliver value sooner, which adds up to substantially increased total value.

A, usually unexamined, assumption in both Waterfall and the Critical Path Method, is that we do a single delivery at the end of the project. What if we can split the delivery up, so we can do multiple smaller deliveries?

In our web site example, what if we could deliver a minimal set of web pages the first week, after only three days of work? We could then, perhaps, make a second delivery at the end of the second week, and a third and final delivery at the end of the project. Let’s try it!

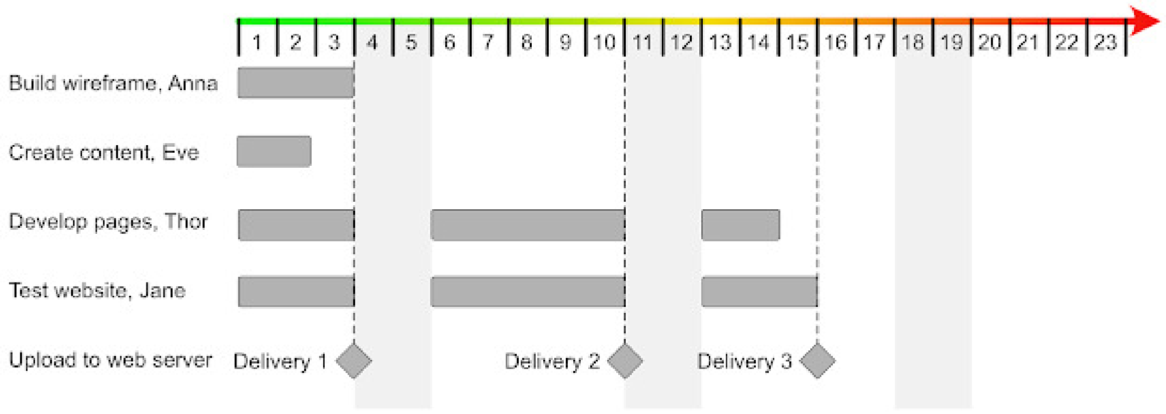


Figure 11: With agile methods, we do small, frequent deliveries, which enables us to get revenue from working software much earlier than with Critical Path or Waterfall planning. However, this does not work in all kinds of projects.

In this scenario we make a first, very minimal, delivery after 3 days, a second delivery after 10 days, and the third and final delivery after 15 days. That means we can start generating revenue after 3 days, We make revenue for 12 days more than with Critical Path, and for 21 days more than with Waterfall.

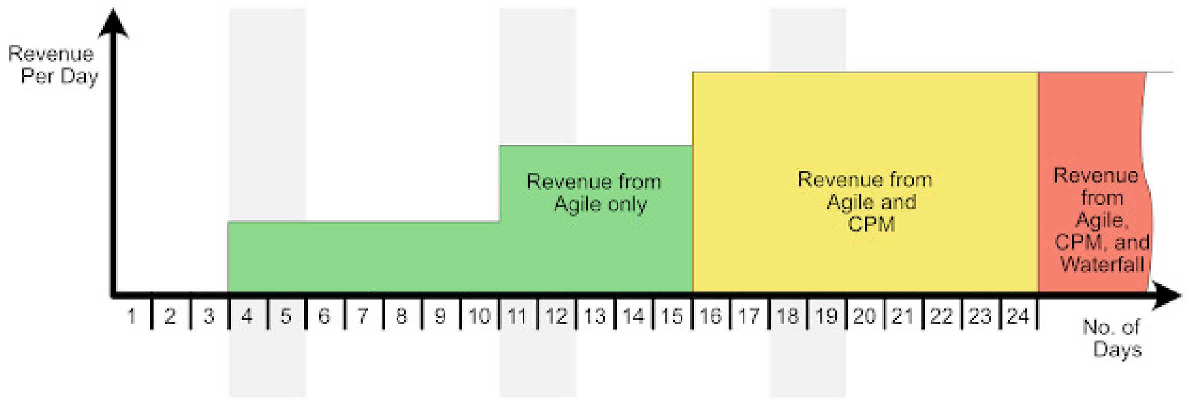


Figure 12: As we can see in the picture above, CPM is more profitable than Waterfall. Agile is more profitable than CPM, if we can make multiple deliveries. Even if we cannot make multiple deliveries, Agile would still beat Waterfall hands down.

From an economic perspective, agile methods win hands down *if* it is possible to slice the project into multiple deliveries. If that is not possible, agile and Critical Path look pretty even. In practice, Critical Path planning, and similar methods, do have some advantages over agile planning when it comes to reducing lead time in large projects. (The exception is the Crystal family of methods, which handles work package breakdown and parallelization better than other agile methods. Maybe on par with Critical Path.)

Neither agile, nor Critical Path, ever comes close to being as bad as Waterfall, under any circumstances.

Reviving Waterfall is a giant leap more than fifty years back in time. Waterfall leads to delays, and cost overruns, it reduces the quality of software, and makes developers incredibly stressed.

I for one, do not want the bad old times back!

We should look forward instead. Build on the knowledge of software development and software project management we have gathered over the past half century. Learn from both failures and successes, and develop better ways of working.