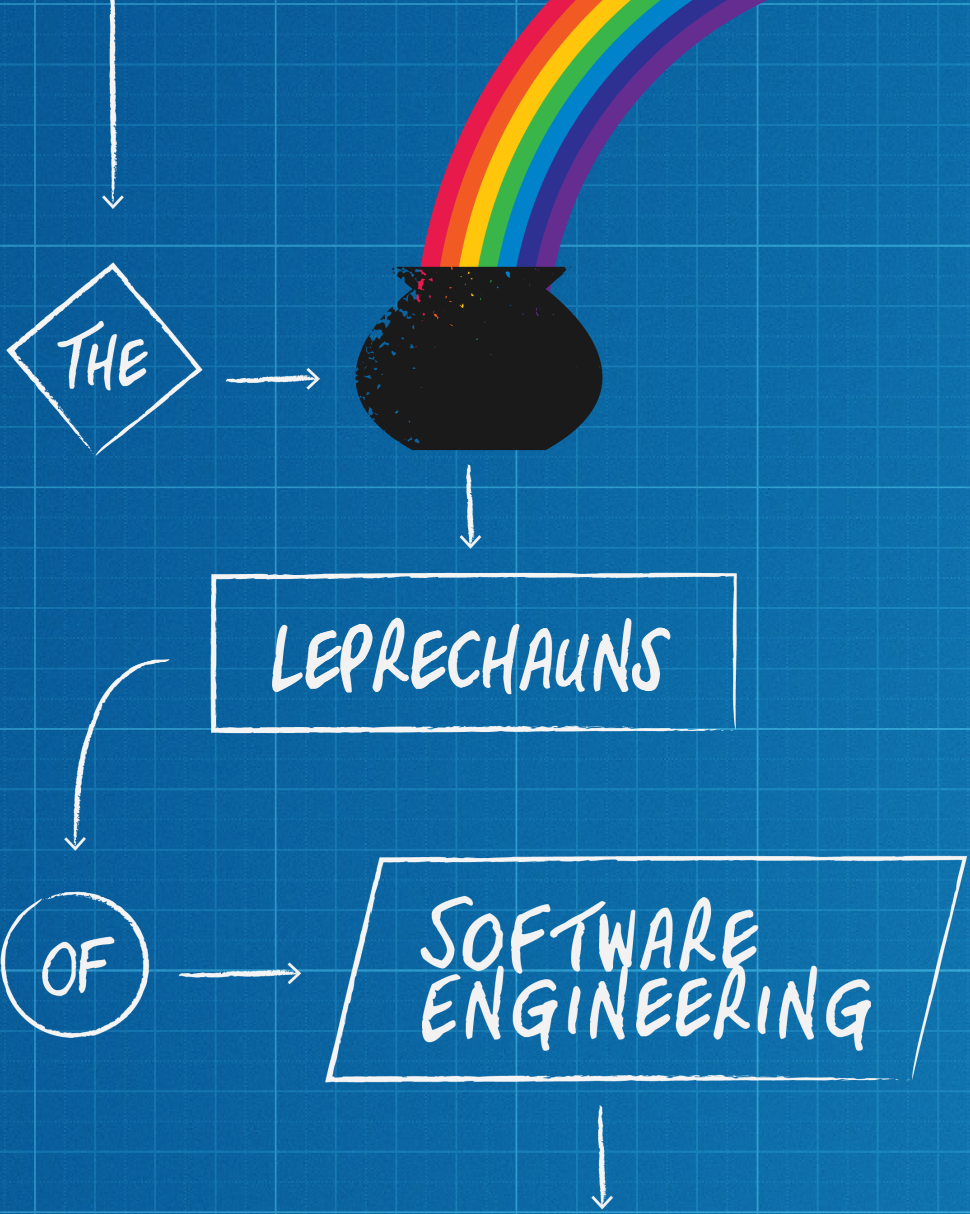
The Leprechauns of Software Engineering



HOW FOLKLORE TURNS INTO FACT AND WHAT TO DO ABOUT IT

Laurent Bossavit

The Leprechauns of Software Engineering

How folklore turns into fact and what to do about it

Laurent Bossavit

This book is for sale at http://leanpub.com/leprechauns

This version was published on 2017-06-27

Tweet This Book!

Please help Laurent Bossavit by spreading the word about this book on Twitter!

The suggested hashtag for this book is #leprechauns.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#leprechauns

Contents

Preface

This book is a work in progress.

I don't know when, if ever, it will be finished. This is the new normal for books, apparently. It has been the new normal for software for a long time now.

The sample contains two chapters in addition to this preface, and should give you an idea of the topic and my writing style.

I'm still writing further material. If you purchase the book, you are entitled to free updates with all the new material that goes into this book, at no extra charge.

I have no schedule to finish the book, nor a very clear idea of what it will be like when it's done. Probably not too different from what it is now, but assume that any of the book is up for revision.

I'm writing this book because I *must*, not as marketing material or anything else. These are topics that I find myself obsessed with, and I find that the only way I get them out of my system is to write about them. I write about them in various places, and you can probably get your hand on nearly everything I've written for free.

Turning these writings into a book only means that I'm making more of an effort to weave a coherent story out of what learnings I've been able to glean.

By purchasing the book - at any price and in any state - you are supporting me in this effort to understand better what we know about software engineering and why we think we know it.

I appreciate your feedback in any form: cold hard cash, a pat on the back, a piece of gentle or harsh criticism. You're even welcome to tell me what an idiot I am (and I may ignore you).

Thank you.

Chapter 1: Software Engineering’s telephone game

The software profession has a problem, widely recognized but which nobody seems willing to do much about. You can think of this problem as a variant of the well known “telephone game”, where some trivial rumor is repeated from one person to the next until it has become distorted beyond recognition and blown up out of all proportion.

Unfortunately, the objects of this telephone game are generally considered cornerstone truths of the discipline, to the point that their acceptance now hinders further progress.

It is not that these claims are outlandish in themselves; they started as somewhat reasonable hypotheses. The problem is that they have become entrenched as “fact” supposedly supported by “research”, and attained this elevated status in spite of being merely anecdotal.

How we got there

One of the ways that anecdote persists is by dressing itself up in the garments of proper scholarship. Suppose you come across the following claim for the first time:

> Early results were often criticized, but decades of research have now accumulated in support of the incontrovertible fact that bugs are caused by bug-producing leprechauns who live in Northern Ireland fairy rings. (Broom 1968, Falk 1972, Palton-Spall 1981, Falk & Grimberg 1988, Demetrios 1995, Haviland 2001)

Let's assume that this explanation immediately appeals to you: it makes sense of so many of the things you've seen in software engineering! The proliferation of bugs in the face of huge efforts to eradicate them; their capricious-seeming nature - why, that is very leprechaun-like!

Of course, you, my reader, may be the kind of hard-headed skeptic who absolutely and definitely dismisses the idea that fairies and leprechauns exist at all. If so, please allow that there exists the kind of person who would be persuaded by a leprechaun-based explanation; but who, while an open-minded person, nevertheless thinks that it is important that explanations be adequately backed by evidence.

Surely you agree that this claim would be convincing to someone like that, since it cites so many respected authors, and papers published in peer-reviewed journals.

As it happens, there are many ways this citation style can be misleading, even without outright fabrication or evil intent:

- the papers are not really empirical research
- the papers support weaker versions of the claim
- the papers don't support the claim directly, but only cite research that does

- the more recent papers are not original research, but only cite older ones
- the papers are in fact books or book-length, and you'll be looking for a needle in a haystack
- the papers are obscure, hard to find, out of print or paywalled, and thus hard to verify
- the papers are selected only on one "side" of an ongoing controversy

Surface plausibility

When we look closely at some of the "ground truths" of software engineering - the "software crisis", the 10x variability in performance, the cone of uncertainty, even the famous "cost of change curve" - in many cases we find each of these issues pop up, often in combination (so that for instance newer opinion pieces citing very old papers are passed off as "recent research").

Because the claims have some surface plausibility, and because many people use them to support something they sincerely believe in - for instance the Agile styles of planning or estimation - one often voices criticism of the claims at the risk of being unpopular. People like their leprechauns.

In fact, you're likely to encounter complete blindness to your skepticism. "Come on," people will say, "are you really trying to say that leprechauns live in, what, Africa? Antarctica?" The leprechaun-belief is so well entrenched that your opposition is taken as support for some other silly claim - your interlocutors aren't even able to recognize that you question the very terms upon which the research is grounded.

For instance, when I argued against the "well-known fact" of 10x variations in software developers' productivity, the objection I often met was "do you really believe that all developers have the same productivity?" Very few people can even imagine not believing in "productivity" as a valid construct.

Leprechaun spotting

Leprechauns come in many forms, which I'll call tacit, folklore and formal. We need to deal with these various forms differently.

Tacit

Some Leprechaun claims have become so pervasive in software engineering discourse that they don't even appear *as* claims any more.

For instance, people who are trying to hire "rockstar" or "ninja" programmers are probably influenced by a tacit belief in the supposedly large variations in programmer productivity, even if they don't explicitly say that they are looking for a "10x productivity programmer". There is a hidden inference at work: "there exist programmers who are ten times as productive as the average, *therefore* it is a profitable investment for me to go to great expense to find one of these".

Another example might be someone who defends Agile testing techniques, such as Test-Driven Development (TDD), because "they reveal defects early". There is a hidden inference too, which relies on the "well-known fact" that software defects are more costly to fix the later they are detected - and *therefore* TDD lowers costs by catching defects early. Unfortunately, this claim on the cost of fixing defects is at best problematic, as we'll see later on.

Folklore

In many cases, the claims are only secondary. They are reproduced in an article, a blog post or a Powerpoint presentation, often by someone who hasn't read - in fact hasn't even looked at - any of the original references.

Here the inference is explicit: there is a point being made, and the claim is offered in support of the point. It can even be the same point as when the claim is tacit, such as the importance of hiring rockstar programmers or the great value of TDD.

Quite frequently, the Leprechaun claim is only ancillary to the main argument: the author has other reasons for believing in the conclusion they are presenting, and the claim is mostly there as a bit of window-dressing.

Formal

Lastly, there is the case of the primary author: someone who did the bibliographical footwork in the first place, should have known better, and is causing a leprechaun-belief to spread.

Whether we like it or not, software practitioners pay scant attention to academic writing about software development. Rather, most of the insights we take for granted come from authors who have a knack as *popularizers*. They play more or less the same role as popular science journalists with respect to the general public.

Science journalism is a fine and important thing, but it has a well-known failure mode: sensationalism, where the lure of an attention-grabbing headline causes writers to toss caution to the wind and radically misrepresent a claim.

The examples I've examined (the cone of uncertainty, the 10x variability, the cost of change curve, etc.) strongly suggest that we should raise our expectations of rigor in software engineering writing, especially writing that popularizes research results.

What you can do

This book is intended as a handbook of skeptical thinking and reading, with worked-out examples.

What I want you to take away from reading the book is a set of reflexes that you will call on whenever you come across a strong opinion about software development, whatever "camp" or "community" or "school" that opinion comes from.

It will probably be easiest to apply these reflexes against what I've called the "folklore" and "formal" version of Leprechaun claims: when you come across them in an article, blog or book, and the claim is spelled out explicitly.

It isn't necessarily the best of ideas to always call out such claims, especially if you are overly antagonistic about it; you may end up being seen as a "troll" - someone more interested in winning arguments than in the truth of things. However, these false claims *will* keep spreading unless somehow kept in check. I cannot any longer accept that it's better to keep quiet and not rock the boat.

The best approach is probably to keep track of where the *best* and *most even-handed* treatments of these various claims reside, and to respectfully point people to them. I hope that this book serves as one such source - but I'm under no illusion that I can deal with even a substantial fraction of all bogus claims within the space of a single book.

The hardest step

The real challenge will be to apply these reflexes *to your own beliefs*.

An inspiring example

Graham Lee is the author of "Test-Driven iOS Development" (Addison-Wesley, 2012). Page 4 of his book includes a table which reproduces a claim about the "cost of defects", which we'll be examining in detail in a later chapter.

In september 2012, after reading an early draft of *Leprechauns*, Graham published a retraction in the following terms: "I made a mistake. [...] I perpetuated what seems (now, since I analyse it) to be a big myth in software engineering. I uncritically quoted some work without checking its authority, and now find it lacking."

Graham not only took seriously my warning about the "cost of defects" claim. He actually went looking for the actual evidence and made his call on that basis, rather than taking my word for it. That's the kind of behaviour I'd like to see more of.

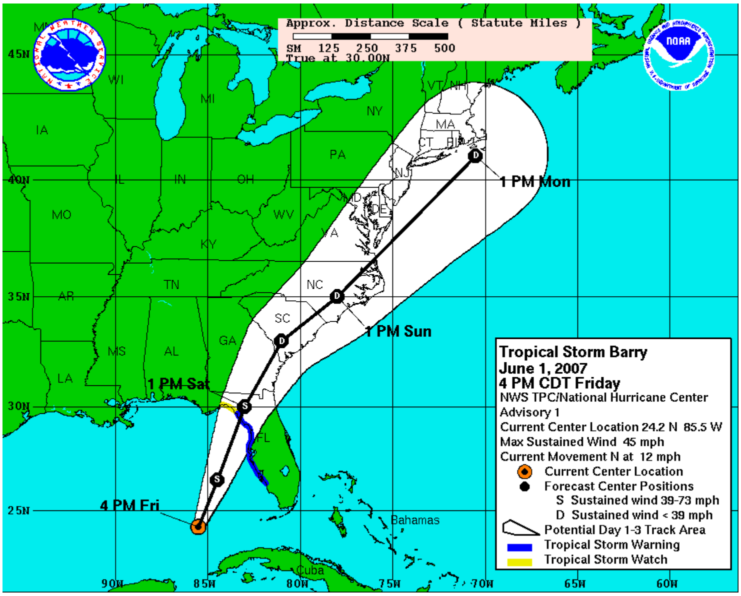
I hold out little hope that people can, in general, convince *others* to let go of specific pet notions. Speaking out against belief X may not do much for those who currently hold belief X strongly enough that they are writing or blogging about it, although there will hopefully be some happy exceptions like Graham.

However, I do believe that if we manage to raise our overall level of "epistemic hygiene", we can prevent Leprechauns from spreading in the first place. Like its real-world counterpart, epistemic hygiene can be vastly improved by the use of specific techniques that aren't hard to learn, like washing hands.

That's what's coming next. Onwards!

Chapter 2: The Cone of Uncertainty

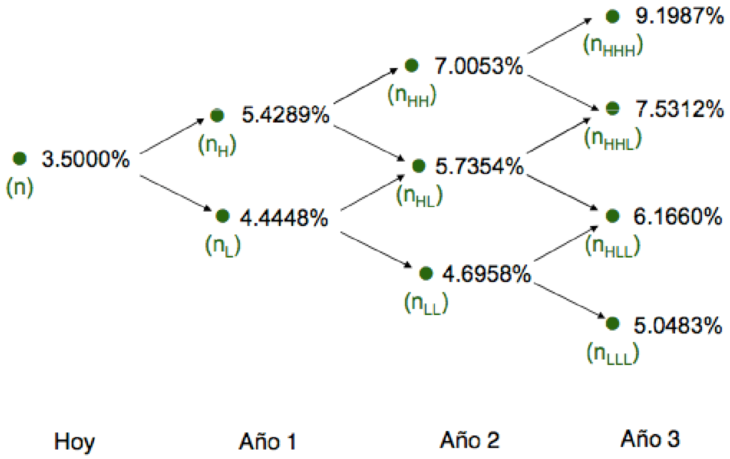
You may have come across something called the “cone of uncertainty”.


Approx. Distance Scale (Statute Miles)
SM 125 250 375 500
True at 30.00N
1 PM Mon
1 PM Sun
1 PM Sat
4 PM Fri
Tropical Storm Barry
June 1, 2007
4 PM CDT Friday
NWS TPC/National Hurricane Center
Advisory 1
Current Center Location 24.2 N 85.5 W
Max Sustained Wind 45 mph
Current Movement N at 12 mph
Current Center Location
Forecast Center Positions
S Sustained wind 39-73 mph
D Sustained wind < 39 mph
Potential Day 1-3 Track Area
Tropical Storm Warning
Tropical Storm Watch
Bahamas
Cuba
45N
40N
35N
30N
25N
90W
85W
80W
75W
70W
65W
60W

Cone of uncertainty, meteorological version

There are several graphics out there called that. The one with the widest recognition is probably the one used to model the future paths of hurricanes.

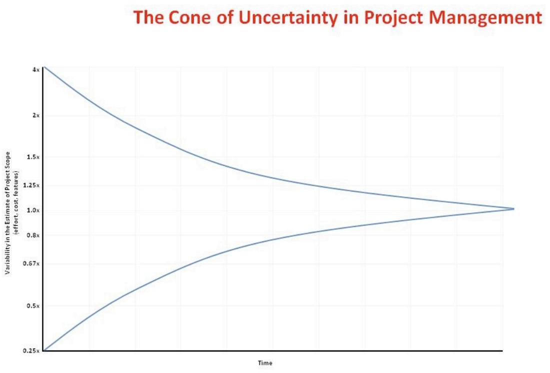
A more obscure one is the “binomial tree” used in the theory of financial options.



Cone of uncertainty, financial version

In both cases this is diagrammed from a present-time perspective, with the cone widening toward the future. We have some certainty about what is going to happen a few seconds from now - in all likelihood it will be whatever is happening now, more or less. And the further into the future we try to peek, the murkier it becomes.

What we’re going to discuss - and what has come to be accepted as one of the “well known truths” of software engineering, more specifically of software project management, is the following “inverted” cone, popularized by Steve McConnell after a diagram originally from Barry Boehm.



Cone of uncertainty, project management version

This is diagrammed from a future-time perspective, and shows uncertainty as a symmetrically widening range as we move further toward the present, which (given the usual convention for diagrams with a time axis) is on the left of the figure.

How to feel foolish in front of a class

The diagram became well-known when it was published in Steve McConnell's "Rapid Software Development", in 1996. McConnell cites a 1995 Boehm article as the source, but the diagram can in fact be found as early as 1981 in Boehm's famous book "Software Engineering Economics".

McConnell's book is where I first came across the Cone, and it struck me as entirely plausible. I started using it to illustrate the perils of project planning. I distinctly remember one particular occasion when I was instructing a group of software engineers on the topic of “agile planning”, and I started drawing a picture of the cone of uncertainty.

And I stopped dead in my tracks.

Because I’d just realized I hadn’t the foggiest idea what I was talking about, or how I’d know if it made sense. I was just parroting something I’d read somewhere, and for once trying to explain it wasn’t helping me understand it better, it was just making me confused. And all I wanted to say anyway was “don’t trust estimates made at the beginning of a project”.

Making sense of the picture

What is odd, to any experienced software engineer, is that the Cone diagram is *symmetrical*, meaning that it is equally possible for an early estimate to be an over-estimate or an under-estimate. This does not really square with widespread experience of software projects, which are much more often late than they are early.

If you think a little about it, it can get quite puzzling what the diagram is supposed to *mean*, what each of its data points represents. A narrative interpretation goes like this: “very early in the project, if you try to estimate how long it will take, you’re likely to end up anywhere within a factor of 4 of what the project will eventually end up costing”. So a 1-year project can be estimated as a 3-month project early on, or as a 4-year project. Even after writing a first cut of requirements, a 1-year project can be estimated as a 6-month project or as a 2-year project.

It's not clear that this latter case is at all common: a project that has reached this phase will in general take *at least* as long as has been planned for it, an instance of Parkinson's Law. The Cone suggests that the distribution of project completion times follows a well-behaved Gaussian. What the Cone also suggests is that the "traditional" project management activities help: "By the time you get a detailed requirements document the range of uncertainty narrows considerably." And the Cone suggests that uncertainty inevitably narrows as a project nears its projected release date.

Widespread experience contradicts this. Many projects and tasks remain in the "90% done" state for a very long time. So if you wanted a diagram that truly represented how awful overall project estimation can be, you would need something that represented the idea of a project that was supposed to be delivered next year, for 15 years in a row. (Yes, I'm talking about Duke Nukem Forever.)

Getting to the facts

Boehm's book is strongly associated with "waterfall" style project management, so for a long while I resisted getting the book; I'd verified earlier by looking at a borrowed copy that the diagram was indeed there, but I wasn't really interested in digging further.

What I seemed to remember from that brief skim was that the diagram arose from research Boehm had done at TRW while building his large quantitative database which forms the basis for the COCOMO cost-modeling framework, and which is the book's main topic.

I assumed that the diagram was the "envelope" of a cluster of data points obtained by comparing project estimates made at various times with actuals: some of these points would fall within the Cone, but the ones farthest from the original axis would draw the shape of the Cone if you "connected" the dots.

After seeing the Cone turn up in blog post after blog post, for a number of years, I finally broke down and ordered my own copy. When it arrived I eagerly turned to p.311, where the diagram is discussed.

And found a footnote that I missed the first time around:

> These ranges have been determined **subjectively**, and are intended to represent 80% confidence limits, that is 'within a factor of four on either side, 80% of the time'.

Emphasis mine: the word "subjectively" jumped out at me. This puzzled me, as I'd always thought that the Cone was drawn from empirical data. But no. It's strictly Boehm's gut feel - at least that's what it's presented as in the 1981 book.

The telephone game in action

And then I chanced across this bit from a bibliography on estimation from the website of Construx (Steve McConnell's company):

> Laranjeira, Luiz. 'Software Size Estimation of Object-Oriented Systems,' IEEE Transactions on Software

> Engineering, May 1990. This paper provided a theoretical research foundation for the empirical observation of the Cone of Uncertainty.

Wait a minute. *What* empirical observation?

Curious, I downloaded and read the 1990 paper. Its first three words are “the software crisis”. (For a software engineering leprechaun-doubter, that’s a very inauspicious start; the “crisis” being itself a software engineering myth of epic proportion - possibly *the* founding myth of software engineering. We’ll come back to that in a later chapter.)

The fun part is this bit, on page 5 of the paper:

> Boehm studied the uncertainty in software project cost estimates as a function of the life cycle phase of a product. The graphic in Fig. 2 shows the result of this study, which was empirically validated (3, Section 21.1)

The reference in parentheses is to the 1981 book - in fact precisely to the section I’d just read moments before. Laranjeira, too, takes Boehm’s “subjective” results to be empirical! (And “validated”, even.)

Laranjeira then proceeds to do something that I found quite amazing: he interprets Boehm’s curve mathematically - as a symmetrical exponential decay curve - and, given this interpretation plus some highly dubious assumptions about object-oriented programming, works out a table of how much up-front OO design one needs to do before narrowing down the “cone” to a desired level of certainty about the schedule. Of course this is all castles in the air: no evidence as foundation.

Even funnier is this bit from McConnell's 1996 book "Rapid Software Development":

> Research by Luiz Laranjeira suggests that the accuracy of the software estimate depends on the level of refinement of the software's definition (Laranjeira 1990)

This doesn't come right out and call Laranjeira's paper "empirical", but it is strongly implied if you don't know the details. But that paper "suggests" nothing of the kind; it quite straightforwardly *assumes* it, and then goes on to attempt to derive something novel and interesting from it. (At least a couple later papers that I've come across tear Laranjeira's apart for "gross" mathematical errors, so it's not even clear that the attempt is at all successful.)

So, to recap: Boehm in 1981 is merely stating an opinion - but he draws a graph to illustrate it. At least three people - McConnell, Laranjeira and myself - fall into the trap of taking Boehm's graph as empirically derived. And someone who came across McConnell's later description of Laranjeira's "research" should be forgiven for assuming it refers to empirical research, i.e. with actual data backing it.

But it's leprechauns all the way down.

Controversy

In 2006 my friend and former colleague on the Agile Alliance board, Todd Little, published empirical data in IEEE Software that contradicted the Cone of Uncertainty. (Such data can be hard to come by, if only because it's hard to know what "officially" counts as an estimate for the purposes of measuring accuracy Todd used the project manager's estimates, included in project status reports).

Todd's article immediately generated a controversy, which is precisely what we should expect if the Cone of Uncertainty belongs to the "folklore" category - it is a belief that is hard to let go of precisely because it has little empirical or conceptual backing. It has *perceptual* appeal, insofar as it supports a message that "estimation is hard", but it also has very, very misleading aspects.

Apparently as a result of the controversy, and in a further departure from the original concept from Boehm, McConnell insisted strongly that the Cone "represented a best case" and that in fact, in addition to the Cone one should envision a Cloud of Uncertainty, shrouding estimates until the very end of the project. Metaphorically one "pushes" on the Cloud to get at something closer to the Cone.

By then though, that model has lost all connection with empirical data: it has become purely suggestive. It has no predictive power and is basically useless, except for the one very narrow purpose: providing an air of authority to "win" arguments against naive software managers. (The kind who insist on their teams committing to an estimate up front and being held accountable for the estimate even though too little is known.) But we should not be interested, at all, in winning arguments. We should be interested in what's true and in what works.

The "Cone" isn't really a good pictorial representation of the underlying concept that we want to get at (which is really a probability distribution). It has drifted loose from what little empirical moorings it had thirty years ago.

What to make of all this?

First, that there is a "telephone game" flavor to this whole thing that is reminiscent of patterns we'll see again, such as the claimed 10x variation between software developers' productivity. One technical term for it is "information cascade", where people take as true information that they should be suspicious of, not because they have any good reason to believe it but because they have seen others appear to believe it. This is, for obvious reasons, not conducive to good science.

Second, the distinction between empirical and conceptual science may not be clear enough in software engineering. Mostly that domain has consisted of the latter: conceptual work. There is a recent trend toward demanding a lot more empirical science, but I suspect this is something of a knee-jerk reaction to the vices of old, and may end up doing more harm than good: the problem is that software engineering seems bent on appropriating methods from medicine to cloak itself in an aura of legitimacy, rather than working out for itself methods that will reliably find insight.

Third, I wish my colleagues would stop quoting the "Cone of Uncertainty" as if it were something meaningful. It's not. It's just a picture which says no more than "the future is uncertain", which we already know; but saying it with a picture conveys misleading connotations of authority and precision.

If you have things to say about software estimation, think them through for yourself, then say them in your own words. Don't rely on borrowed authority.

Key points

Before quoting an "authority" or "well-known fact" in our field, be sure to apply basic critical thinking. One formulation I like is by James Bach: "Huh? Really? So?[1]"

That is, *Huh?*, do I really understand the claim? Can I rephrase it in my own words? *Really*, is it in fact true? Can I locate the evidence behind the claim? And finally, *So?* or *So what?*, does anything important depend on the claim being true or not?

Finally, remember that if "a picture is worth a thousand words", a meaningless picture wastes the equivalent of two pages or ten minutes of speech!

[1] http://how-do-i-test.blogspot.fr/2011/08/huh-really-so.html

Chapter 7: Who’s afraid of the Big Bad Waterfall?

Let’s take a break from the numbers game for a chapter or two, and examine some qualitative rather than quantitative claims. They’re fun too! And we’ll get back to “harder” topics quite soon. We are, however, still looking at how strong opinions can form around a topic, quite independently of any evidence that exists on the topic.

As software professionals, we should be interested in knowing at least the basics of our own history, for just the same reasons that as citizens we are expected to know about our national history and about world history: so that we will be able to make informed decisions and know who to trust, who to listen to; so that we are not deceived by lies. Untrue histories generally have an agenda - “someone trying to sell you something”, as the saying goes.

Quite a bit of the current debate on software engineering relies on opinions regarding the “creation myth” of the discipline: the so-called waterfall model of sequential software development, also known as the SDLC (Software Development Life Cycle).

Unfortunately, most of these opinions are wildly inaccurate.

The standard story

An article[2] by Robert Martin provides (along with some other interpretations that I'll come back to) what is now the nearly universal explanation of how conceptions of the SDLC became pervasive in the discourse of software engineering:

> In 1970 a software engineer named Dr. Winston W. Royce wrote a seminal paper entitled Managing the Development of Large Software Systems. This paper described the software process that Royce felt was appropriate for large-scale systems. As a designer for the Aerospace industry, he was uniquely qualified. [...] Royce's paper was an instant hit. It was cited in many other papers, including several very important process documents in the early '70s. One of the most influential of these was DOD2167, the document that described the software development process for the American Department of Defense. Royce was acclaimed, and became known as the father of the DOD process.

You can find further confirmation of the "seminal" character of Royce's paper on Wikipedia:

> The first formal description of the waterfall model is often cited as a 1970 article by Winston W. Royce, though Royce did not use the term "waterfall" in this article.

[2] http://cleancoder.posterous.com/what-killed-waterfall-could-kill-agile

For many, the standard story is the whole story; over the ensuing decades, even though many variants on the "waterfall" life cycle were proposed that all have their uses in one context or another, the waterfall still remains one of the major foundations of software engineering. It's the model to learn as a basis for learning other variants, and as such is taught quite seriously at many universities. It is a constant fallback of enterprise software development efforts, a norm against which other models are judged.

In any case, the following are widely, in fact almost universally, agreed upon:

- Dr. Winston Royce "invented" the waterfall model in 1970
- The nascent software engineering community welcomed the break from "artisanal" practice of the past
- The model was instantly enthusiastically adopted as a sequential, non-overlapping phases model
- Having become an industry norm, the model was taken up by the US DoD
- Variants of the model were developed later and used where appropriate

Alternate endings

There are at least two "modern" endings to the mythical story, told by different people according to whether they agree with the tenets of the Agile movement; for the Agilists,

- Tragically, this was all a misunderstanding, based on careless reading of Royce

- Royce was actually advocating an "iterative and incremental" approach in his paper (!)

Whereas for people who disagree with Agilists,

- "waterfall" is recent coinage and has been used only as a straw-man term
- formal lifecycles are not actually as inflexible and risky as "waterfall" is made out to be
- Royce's paper wasn't actually advocating a rigid sequential approach

The article by Robert Martin cited above is representative of the first group, which I'm tempted to characterize as "agile revisionists"; for instance he writes:

> [Royce] began the paper by setting up a straw-man process to knock down. He described this naïve process as "grandiose". He depicted it with a simple diagram on an early page of his paper. Then the paper methodically tears this "grandiose" process apart. […] Apparently the authors of DOD2167 did not actually read Royce's paper; because they adopted the "grandiose", naïve process that Royce's paper had derided. To his great chagrin, Dr. Winston W. Royce became known as the father of the waterfall.

Larman and Basili, in their "Brief History" of iterative and incremental development, offer support this interpretation with a quote by Walker Royce - the son of Winston Royce: "He was always a proponent of iterative, incremental, evolutionary development."

The second group is well represented by the following excerpts from a 2003 Web essay, titled "There's no such thing as the Waterfall Approach (and there never was)":

> I don't recall when I first saw that term, but I'm sure it was in a pejorative sense. I'm unaware of any article or paper that ever put forth the "waterfall methodology" as a positive or recommended approach to system development. In fact, "waterfall" is a straw-man term, coined and used by those trying to promote some new methodology by contrasting it with a silly alleged traditional approach that no competent methodology expert ever favored. [...] Phase disciplines, when practiced with sensible judgment and flexibility, are a good thing with no offsetting downside. They are essential to the success of large projects. Bogus attacks on the non-existent "waterfall approach" can obscure a new methodology's failure to support long-established sensible practice.

Just the facts

In fact, *both* modern interpretations are demonstrably wrong. Not only that - but *all* the elements of the standard myth turn out to be false or at least substantially at odds with the historical record.

First, what was Royce actually saying in that 1970 paper? Many who echo the “agile revisionist” quote a part of that paper where he says that the unmodified sequential approach “is risky and invites failure”.

However, as we all know, with selective quotation we can make anyone say anything we want. The full sentence was “I believe in this concept, but the implementation described above is risky and invites failure.” In other words, Royce is cautioning against simplistic interpretations, but not condemning the basic idea; a few paragraph further Royce adds this, which for some reason is much more rarely quoted:

> I believe the illustrated approach to be fundamentally sound. The remainder of this discussion presents five additional features that must be added to this basic approach to eliminate most of the development risks.

From a close reading of Royce’s paper, “the illustrated approach” refers to Figure 3; that is, the picture showing a “cascading” series of activities, but allowing that iteration occurs between succeeding phases (the analysis phase may undergo rework on the basis of errors uncovered in the design phase, for instance; or the design may undergo rework as a result of errors in the coding phase). The “risky and invites failure” comment can be inferred, from its placement in the text, to refer to Figure 2 - which showed the same cascade of activities but no feedback loops at all.

Regarding the “five additional features”, again many people give in to the temptation to mention only one, that supports their reinterpretation of Royce: the recommendation to “Do It Twice”, i.e. flush out technical risk by building a prototype; and this is only #3 of the five. For completeness, the remaining four recommended features are:

- add what we would now call an "architecture phase" at the start of the process (#1)
- err on the side of too much documentation (#2)
- make sure to "plan, control and monitor" the testing process (#4)
- have the customer formally involved to sign off on various deliverables (#5)

Finally, Royce's "iterative" recommendations stop short of allowing at any point that the first two "requirements" phases of the cycle can be included within an iterative loop: the "Do It Twice" recommendation is confined to the design and implementation stages.

No paper is an island

Anyone with an interest in so-called "development processes" should read the Royce paper carefully at least a couple times in their careers, but the really interesting part comes when we remember that in any discipline with an academic tradition, papers don't exist in isolation.

Fortunately, with the help of today's Web serious bibliographic research is within everyone's reach. For instance, we can use Google Scholar to understand the real history of Royce's paper within the larger context of the software engineering discipline.

Scholar gives us in particular a list of the *citations* of Royce's paper, which can be narrowed by date.

Did Royce's 1970 paper "invent" or "formally define for the first time" the concept of the software development life cycle, or the notion of a succession of stages?

The answer is no: the first papers cited that mention (and draw a diagram of) a sequential or stagewise model of software development go back at least to 1956. The identification of Royce's paper as the origin of the waterfall is largely arbitrary. Royce's paper *was* however the origin of the most common *picture* of the waterfall - with its instantly recognizable downward cascade of boxes (and the loops showing iteration between phases, which some later authors omit). But as I'll explain presently, that was probably not Royce's fault at all - though it wasn't due to careless reading either.

Was Royce's paper "an instant hit"? The answer is no.

Taking a step back, let's look at a graphical representation of how often Royce's paper is cited in the software engineering literature:

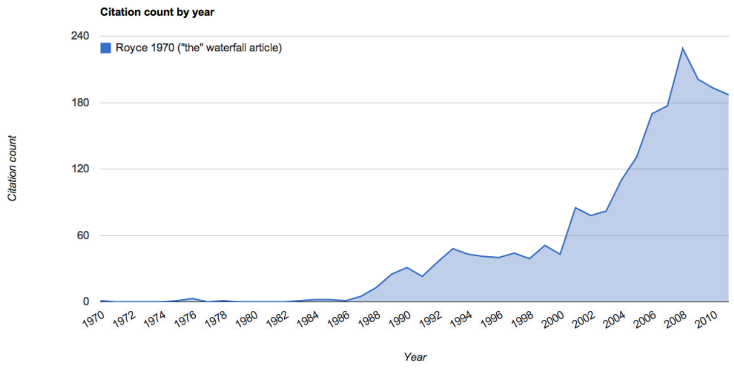


Figure 1

We can see that the 1970 paper in fact remained almost totally unknown until 1987.

What the chart doesn't show is a peculiar property of these early citations: just about every single one of them is by an author at TRW, a US defense contractor who employed several of the authors involved in the early years of the software engineering discipline, including Barry Boehm, known for an exceptional number of contributions to the field.

It turns out that Boehm (who cannot be accused of having read Royce "carelessly") used Royce's well-known diagram (Figure 3, the one with feedback loop between successive phases that Royce characterized as "fundamentally sound") in a 1976 paper modestly titled "Software Engineering".

In that paper, Boehm didn't give credit to Royce for the picture (though he cites an unrelated paper of Royce's). Rather, that diagram was used, with only the briefest of explanation, to provide a structure for the rest of the paper, which examined phase by phase the recent advances in software engineering. Several early authors in fact refer to the diagram as "Boehm's waterfall", rather than "Royce's waterfall".

Here is a quote from a paper by two of Boehm's colleagues at TRW the same year:

> The evolution of approaches for the development of software systems has generally paralleled the evolution of ideas for the implementation of code. Over the last ten years more structure and discipline have been adopted, and practicioners have concluded that a top-down approach is superior to the bottom-up approach of the past. The Military Standard set MIL-STD 490/483 recognized this newer approach [...] The same top-down approach to a series of requirements statements is explained, without the specialized military jargon, in an excellent paper by Royce; he introduced the concept of the "waterfall" of development activities.

Rather than support the idea that Royce's paper drew instant support and influenced military practice in software development, this quote suggests quite the opposite: the defense contractor TRW (who also had contacts within the US Defense departments responsible at the time for defining these standards) seems to have seized on Royce's paper as a good introduction and justification of existing practice.

Late bloomer

Using Google Scholar to reconstruct the history of Royce's paper, we can finally better understand how it ended up being known as "the origin of waterfall".

Notice that there is a sudden surge of publications citing Royce's paper in 1987: this is due to the paper having been republished in that year's international software engineering conference, at the initiative of none other than Barry Boehm in his "Software Process Management: Lessons Learned from History[3]". Royce's paper was republished alongside two others that Boehm deemed of particular interest from a historical standpoint (one of the others was the 1956 paper which already defined a stagewise model).

(Another sudden surge of publications can be seen around 2001: the cause is harder to identify, because by then the myth is well established and the overall number of papers published each year that cite Royce is quite significant; but it is a safe bet that this renewed interest is due to the growing popularity of Agile at that time.)

There was a very good reason to call attention to the waterfall model at that time: Boehm had just introduced the (iterative) Spiral model of software development which would become one of his most significant publications. Boehm wrote in 1987

> Royce's paper already incorporates prototyping as an essential step compatible with the waterfall model.

[3]http://dl.acm.org/citation.cfm?id=41798

Birth of a myth

What I find particularly striking in this quote is the “compatible with”. Boehm seems to forget that if he takes Royce as the originator of waterfall then this prototyping step isn’t *compatible with* waterfall - it is * part of* waterfall. So, in effect, this quotation is kind of a smoking gun - it is the rhetorical moment where waterfall is being separated into two halves, one “stupid” waterfall and one “enlightened” reading. This later enlightened reading isn’t given the name waterfall, perhaps because to do so would diminish the import of Boehm’s own “Spiral” contribution.

In this sense, the interpretation of the waterfall as a “straw man” is not entirely false. But it isn’t accurate, either, to say that waterfall was *always* a straw man - for the first two decades, nearly, it was discussed generally quite favorably - if only within a relatively small circle of authors.

The story that the written record seems to tell is that the “Royce invented Waterfall” was a convenient myth. Convenient because people could satisfy the requirement of garnishing their papers with a citation, and because it provided a (relatively protean) straw man of “older, more primitive” processes that the more modern “iterative” processes could be contrasted with. And a myth whose career began seventeen years after original publication, breaking a long spell of obscurity but also starting down the road to infamy.

I find this story, the true story of waterfall, much more interesting and enlightening than its caricatures.

Key points

Ideas have histories. They don't come into the world complete and of one piece. An idea's history is often more interesting and complex than we suspect.

Knowing (and researching) the history of your field is an important asset in critical thinking. It will protect you from common misconceptions.

Indexes like Google and Google Scholar provide great opportunities to reconstruct the history of particular ideas and concepts, providing everyone with basic tools of bibliometrics ("a set of methods to quantitatively analyze scientific and technological literature" as defined by Wikipedia).