

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

©2012 Laurent Bossavit



This is a Leanpub book which is for sale at <http://leanpub.com>.

Leanpub helps you connect with readers and sell your ebook, while you're writing it and after it's done.

Contents

Preface	1
Chapter 1: Software Engineering’s Telephone Game	3
How we got there	3
Surface plausibility	5
Who’s responsible?	6
Chapter 2: The Cone of Uncertainty	7
How to feel foolish in front of a class	10
Making sense of the picture	11
Getting to the facts	12
The telephone game in action	14
Controversy	16
What to make of all this?	17

Preface

This book is a work in progress.

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

If you purchase the full ebook, you'll get at least (as of this writing) fourteen full chapters plus two appendices, totaling about 25,000 words. Some of the chapters are writing of mine that has appeared elsewhere but that I'm reworking into an ebook with an overarching theme.

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 in mind yet for the newer material, nor do I know how much of it there will be: at least 15 chapters, 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 an ebook 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 anything 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, Havi-land 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.

Well, 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.

So, 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.

Who's responsible?

In many cases, the claims are only secondary - reproduced in a blog post or a Powerpoint presentation, by someone who hasn't read - in fact hasn't even looked at - any of the original references. It's all but impossible to argue such people out of a belief in leprechauns, and I suggest you don't even try.

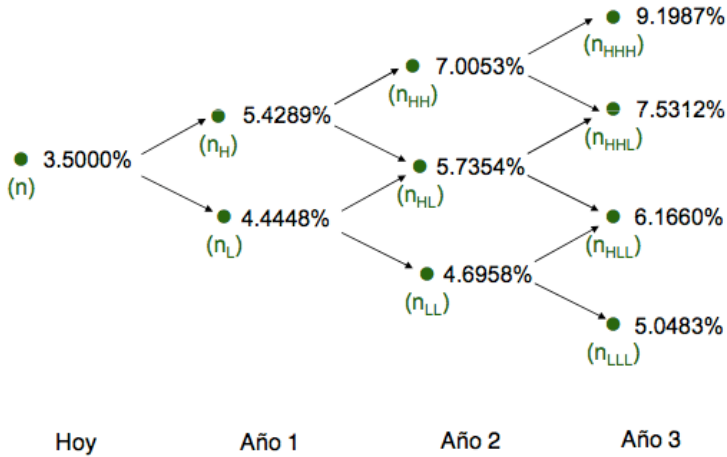
Our real issue, I argue, is with the primary author: the one who did the bibliographical footwork in the first place, and is causing the leprechaun-belief to spread. The real problem is our expectations for software engineering writing, especially writing that popularizes research results.

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, especially in the use of citations, and especially when it comes to citations that reference empirical research.

Chapter 2: The Cone of Uncertainty

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

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.

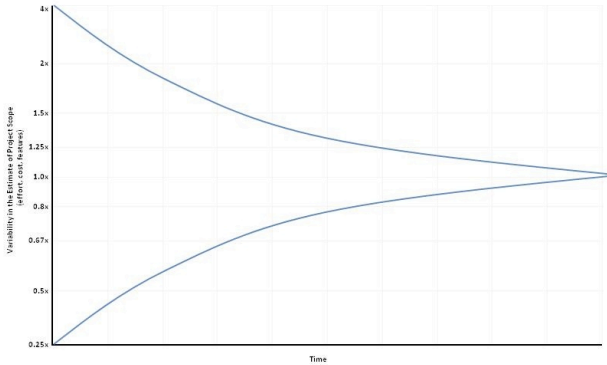


Cone of uncertainty, financial version

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.

The Cone of Uncertainty in Project Management



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 whatever 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.