I am usually leery of, and hence have made a point of, avoiding talks and articles that explicitly state the number of bullets in them in their titles.

I'm not sure why. Maybe because it smells like a cheap journalistic gimmick. Or an Executive Summary.

And yet, mine is the the third "Hail of Bullets" talk in a row in this track.

It follows "Ten Patterns of Database Architecture Refactoring" and "Four Strategies for Recovering the Ability to Design when Surrounded by Messy Legacy". (Really, what other kind is there?).

At least I'm packing more bullets than the other two combined, though, one might argue, theirs were of higher caliber. I'll let you be the judge.

One had better come adequately well armed if one is to follow Mike Feathers, and Eric Evans and Pramodkumar Sadalage.

I can’t remember how many bullets Feathers used?

So, I’m asking myself, am I feeling lucky?

Am I?

Another thing I should establish up front is that these bullets are not necessarily numbered in terms of relative importance, even though I present them as a David Lettermen style countdown (without the drumrolls, alas).

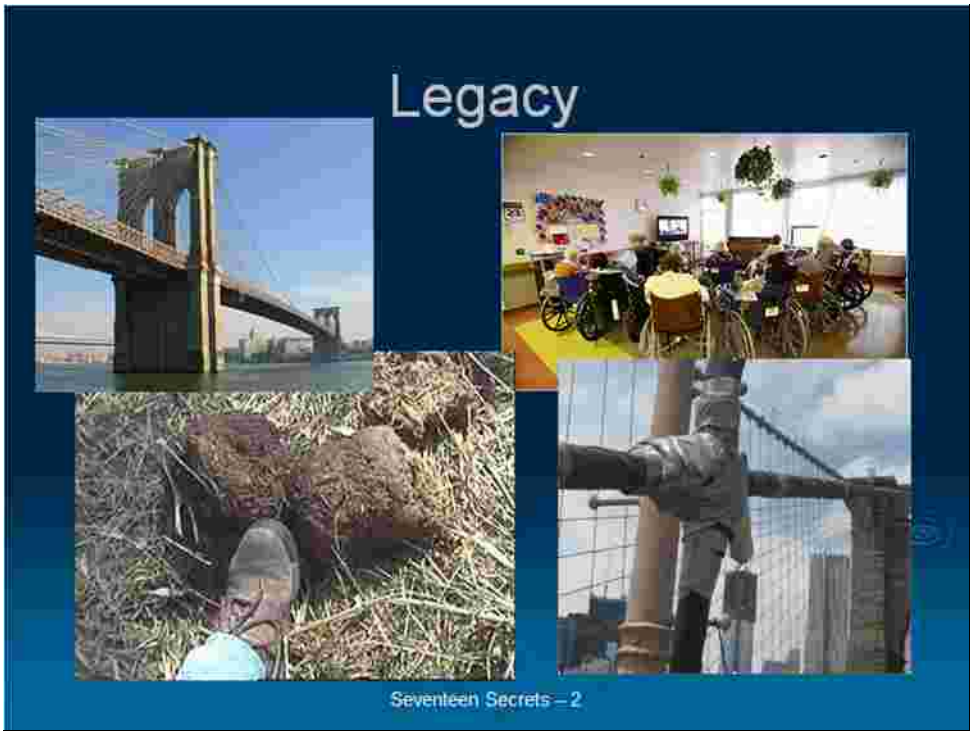
Oh, the last one is #1 in my book. Some I regard as nearly as important appear early on…

And … they are not really “Secrets” either. That’s there to keep things in the spirit of the Tabloid Titles that inspired this one.

Think of this talk as a tip-of-the-hat to the Great Makeover Masters, and to the variety of broad strategies that they have employed to cope with the Horror of Legacy. They are not new and they are not mine. In that sense, they have much in common with patterns. Indeed, some of them have been previously published as such. The other pattern-esque thing the keep in mind about these “Secrets” is that they, like patterns, are Solutions to a Problem IN A CONTEXT, at least in the sense that they all work, in certain / the appropriate / contexts, they all fail in others.

This talk is a litany of the the people and ideas that have, in hindsight, changed/ made a difference in the way with think about Legacy over the last generation.

[I dubbed them “Makeover Masters” in part because this talk is being delivered in New York City, the fashion capital of the USA… The first version of this talk was entitled “Seventeen Secrets of the Great Mudbusters”. Mike Hill talked me out of that title because he was uncomfortable with it. You’ll have to ask him why.]



A B-Tree Grows in Brooklyn: QCon New York 2012 is being held pretty much at the foot of the Brooklyn Bridge, on the Brooklyn side.

The Brooklyn Bridge was one of the 19th Century's greatest engineering achievements. A true 19th Century legacy in the traditional sense of the word.

It was completed in 1883. It's last major overhaul / makeover was competed in 1954 (Gulp, the year I was born).

As a result, these days it's … well, a bit of a fixer-upper… I was surprised to learn. I was genuinely surprised, and, I'll confess, more than mildly amused yesterday when we walked The Bridge to Manhattan, only to discover huge swaths of the suspension cables … wrapped in … and held together by … Duct Tape! …and realized, that … It's not just us. Maybe software isn't that different from other mature engineering artifacts after all…

==========

So: you show up for work one morning, and The Boss says: "Everyone! Wonderful News!. You are all starting work today on an enormous Legacy Codebase!". And, cries of joy erupt, as the assembled multitude contemplates

life a finished, polished, complete, debugged, lavishly, copiously documented, utterly perfected Body of Finished Work. Ah your worries are over. The heavy lifting has already been done. You'll be able to put up your feet, and wallow in the leisurely satisfaction of grafting the the occasional feature or flourish onto this Temple of Elegance…

Balderdash. We all know the reaction would more likely be one of dejection, horror, and despair.

So, I ask you, Legacy: Say the first word / first thing that comes to your mind when you think of Legacy Code:

Old. Big and Old. Ugly. Cobol. Code not under test (a wonderfully provocative idea from Mike Feathers). Code not written by me. Code written by me more than n (3-6) weeks ago… About a month ago… Code touched by anyone else… Any code written before say 5/1/2012. Or only shitty legacy… Code I can't read. Code written by That Guy on the team that I wish they'd fire.

==========

15 or 16 years ago (depending on how I reckon it) Joe Yoder and I put together an erstwhile pattern collection entitled "Big Ball of Mud". We were both members of Ralph Johnson's Software Architecture group back then, and we were reviewing some manuscripts from Frank Buschmann et al. that described "High Level" Software Architectural Patterns", such as Layered Architecture, Blackboard, etc, that would eventually become the backbone of Volume I of the POSA (Patterns of Software Architecture) Series.

It was a heady time for patterns, perhaps the Zenith of the post-GoF patterns boom (bubble?, nah). Still during our discussions, I recalled a notion I had a couple years earlier, while ponder something Grady Booch said, and listening to Desmond Decker's 007-Shantytown, that if we were to entertain the conceit that Software Architecture was like Real Building Architecture, then the kinds of programs I was seeing were quite unlike High Road structures like Empire State Building or the Pompidou Museum (or name your Pritzker Prize winning structure here). They more closely resembled Shantytowns instead. Squalid, haphazardly organized, built by the Jacks-of-All-Trades who lived in them, with the materials at hand…

Joe and I went on to posit that when it came to architecture, that the defacto

standard, most widely deployed architecture out there wasn't any of the high-minded styles being bandied about in those days, but was instead the "Big Ball of Mud". The claim was intended to have something of an "Emperor's New Cloths" *lese majeste* about it. I, for one, had naively expected at first that this would be perceived as a somewhat iconoclastic posture.

But, in hindsight, the thing I still find remarkable is that in the 15 years that have transpired, no one has ever disputed our premise. The reaction instead, a droll, yep, that's just how it is all right.

==========

The picture on the bottom left of the slide depicts (anyone recognize it?) several Big Balls of … Elephant Dung: Is this how Mud scales? Soft, runny and fetid and, well, disgusting in the middle, crunchy, but arguably adequately encapsulated on the outside…

The term "Big Ball of Mud" was suggested during the meetings alluded to above by Brian Marick, who'd observed that it was a phrase used in the Lisp community to describe the kinds of foul, tanged, rat's nest codebases we were describing. I've always suspected (but have yet been able to verify) that the word "mud" was a polite substitute /stand-in for one of several possible more overtly scatological terms.

Now, not all Mud is Legacy. And not all Legacy is Mud. But that the two terms are treated by many as virtually synonymous as often as they are speaks volumes…

==========

The picture in the upper right corner of this slide illustrates a Daily Scrum at a Legacy COBOL shop.

We've been celebrating of late, or more often, ignoring, the centennials of one of the 20th Century's, and America's, most celebrated engineering achievements. This work began in 1904, and was completed in 1914.

But, did you know that The Canal was, in effect, a Legacy Project? Before became the site one of the 20th Century's greatest engineering achievements, it was the site of one of the 19th Century's most tragic engineering failures.

[For reasons I can't claim to completely understand, I've observed that some American audiences take, at times, a slightly unseemly jingoist pleasure in hearing that they'd succeeded where the French had failed.]

The United States completed a used canal.

The first attempt to build a canal across the Isthmus of Panama was undertaken by the French, in 1881.

It was led by Ferdinand de Lesseps, who had led, a few years earlier, the successful effort to build a canal more than twice as long as the proposed

Panama canal across the Isthmus of Suez.

The basic strategy then, was to take what had succeeded spectacularly in Egypt, in fact, and repeat it in Panama (or northern Colombia, to be precise).

But, to cut to the chase, to what had worked in Egypt failed in Panama. Failed terribly. Failed Horribly.

Among the reasons for this debacle:

#1) Impossible Requirements: The French undertook to build a canal at sea level, just as they had at Suez. But a sea-level canal in Panama is beyond the reach of current technology, for all practical purposes, let alone that that was available to the French. During the heyday of the "Peaceful Atom" in the 50's, there was talk of using the Bomb as a steam shovel in places like Panama, which have upon subsequent, more sober reflection, been shelved. For the French effort, this alone was a certain showstopper.

#2) The French were excavating at such a steep angle that the canal filled back up with Mud nearly as fast as the French could dig. The flat, dry terrain of Suez had posed far fewer such challenges. Indeed, even today, the step sides of the Galliard Cut, as well as other stretches of the US canal require constant dredging to keep eroding mud from closing them down. (There is surely a lesson about software maintenance that might be drawn by analogy here…)

#3) The effort was top heavy with managers with little or no engineering training or hands on-experience with canal building (a personnel deployment predilection all too common in our industry, no?)

#3) Financial mismanagement and political corruption were rampant (thus is it always?)

#3) Steam shovels had been invented, but were still primitive. Railroad construction techniques were better established, but the French canal team was not well versed in them. So much of the digging was left to men with primitive hand tools…

#4) The science underlying the disease vectors for malaria and yellow fever was yet to have been established (by French scientists, no less. Disease took a horrible toll. Turnover was high as men fled the appalling conditions in the festering jungle swamps.

The French effort was abandoned in 1889 amidst scandal and recriminations. Reputations and careers were ruined, the triumph of Suez had been eclipsed. And disease and accidents had killed over 22,000 men. Now that's a "Death March"…

Casualties were highest among the European managerial caste. Local laborers as well as workers brought from Africa had acquired more resistance to the illnesses than the Europeans.

This all must have come as an awful shock to de Lesseps. What had work in one context has failed in another.

==========

An excellent treatment of the building of the Panama Canal can be found in historian David McCullough's The Path Between the Seas: http://www.amazon.com/The-Path-Between-Seas-1870-1914/dp/0743262131/ref=pd_sim_b_13

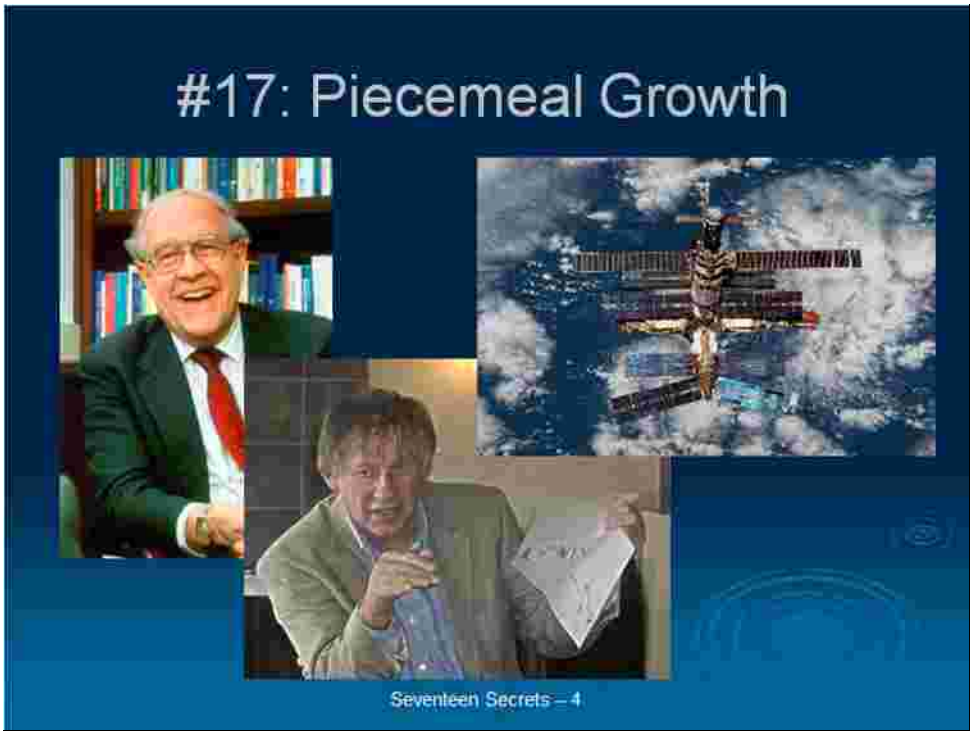
The sketch given here was drawn in part from the Wikipedia article at: http://en.wikipedia.org/wiki/Panama_Canal.

http://upload.wikimedia.org/wikipedia/commons/4/41/Gill-Lesseps.jpg

http://en.wikipedia.org/wiki/Ferdinand_de_Lesseps

http://upload.wikimedia.org/wikipedia/commons/thumb/4/45/YellowFeverVirus.jpg/506px-YellowFeverVirus.jpg

http://en.wikipedia.org/wiki/Yellow_fever



#17: Piecemeal Growth

==========

The man on the left is Fred Brooks, the Dean of the American Software Engineering community. How many of you have read Brook's The Mythical Man Month? Let's see a show of hands? Any of you who did not raise your hands should immediately pick up their laptops or mobile devices and acquire and read a copy immediately. It remains the finest book on software development ever published.

The anniversary edition also contains his truly classic essay on Accident and Essence in Software Engineering.

In it, Brooks observed: Some years ago Harlan Mills proposed that any software system should be grown by incremental development.[11] That is, the system should first be made to run, even though it does nothing useful except call the proper set of dummy subprograms. Then, bit by bit it is fleshed out, with the subprograms in turn being developed into actions or calls to empty stubs in the level below.

--Brooks, Frederick P. (1995-08-02). The Mythical Man-Month: Essays on

Software Engineering, Anniversary Edition (2nd Edition) (Kindle Locations 2256-2259). Pearson Education (USA). Kindle Edition.

One always has, at every stage in the process, a working system. I find that teams can grow much more complex entities in four months than they can build.

--Brooks, Frederick P. (1995-08-02). The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition) (Kindle Locations 2264-2266). Pearson Education (USA). Kindle Edition.

Big programs that work invariably started as small programs that worked.

In other words, find a pulse. Keep it. Start small. Or Never start…

Christopher Alexander is the Rogue Berkeley Architecture Professor from whom the Software Patterns community borrowed the very idea of Patterns. The goal he and his students had was to catalog the things that builders in communities all over the world had been doing "right", but never written down. Things that gave their work a sense of wholeness, of life, the Quality without a Name. They found that places with this quality had only resulted from a process of Piecemeal Growth.

They contrasted it with what we'd now call "Big Design Up-Front" this way:

Large-lump development is based on the idea of ***replacement****. Piecemeal Growth is based on the idea of* ***repair****. … Large-lump development is based on the fallacy that it is possible to build perfect buildings. Piecemeal growth is based on the healthier and more realistic view that mistakes are inevitable. … Unless money is available for repairing these mistakes, every building, once built, is condemned to be, to some extent unworkable. … Piecemeal growth is based on the assumption that adaptation between buildings and their users is necessarily a slow and continuous business which cannot, under any circumstances, be achieve in a single leap.*

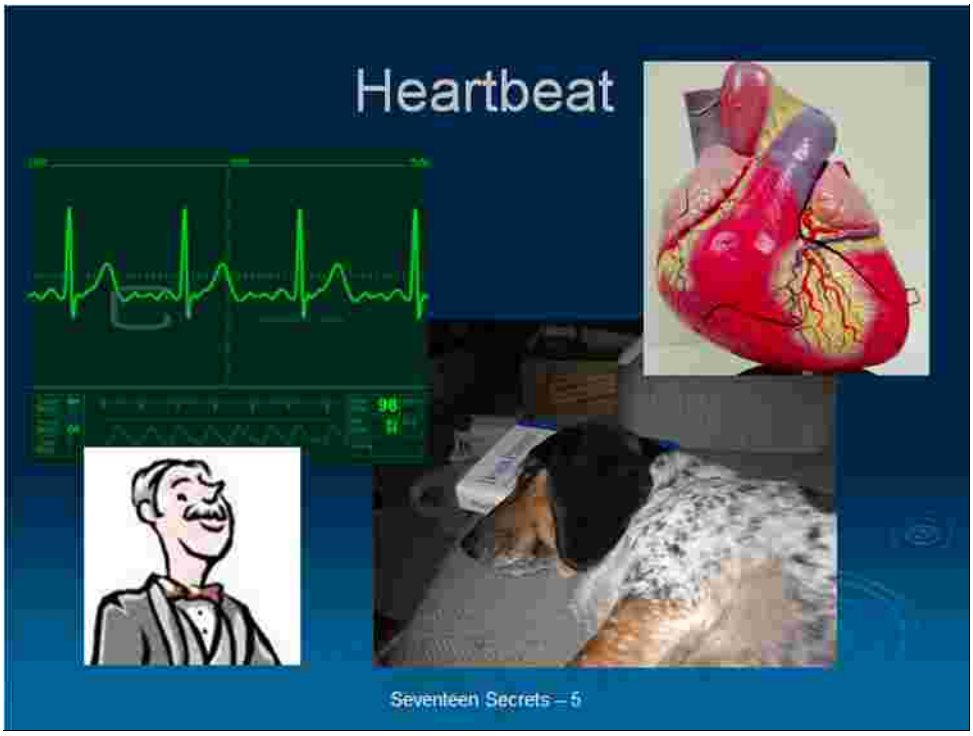
==========

The image in the upper right hand corner of this slide shows the Mir Space Station, constructed by the Soviets, an finished with Russian and American contributions, incrementally, over a (tumultuous) period of ten years. The

American contribution was one that might scarcely have been expected at work was begun on the station during the second height of the Cold War.

- Core *1986*
- Kvant 1 *1987*
- Kvant 2 *1989*
- Kristall *1990*
- Spekter *1995*
- Docking *1995*
- Priroda *1996*

[Yes, some of the material here was “re-sampled” from the Original Big Ball of Mud Paper”. Such is in keeping with changing ideas about the nature of originality in the 21st Century.]



Iterative / Incrementalism has gone over the last twenty years from being a rallying cry of a handful of radical waterfall insurgents to orthodox best practice.

It's a cornerstone, or pillar, or what-have-you, of every one of the Agile-ist Sects.

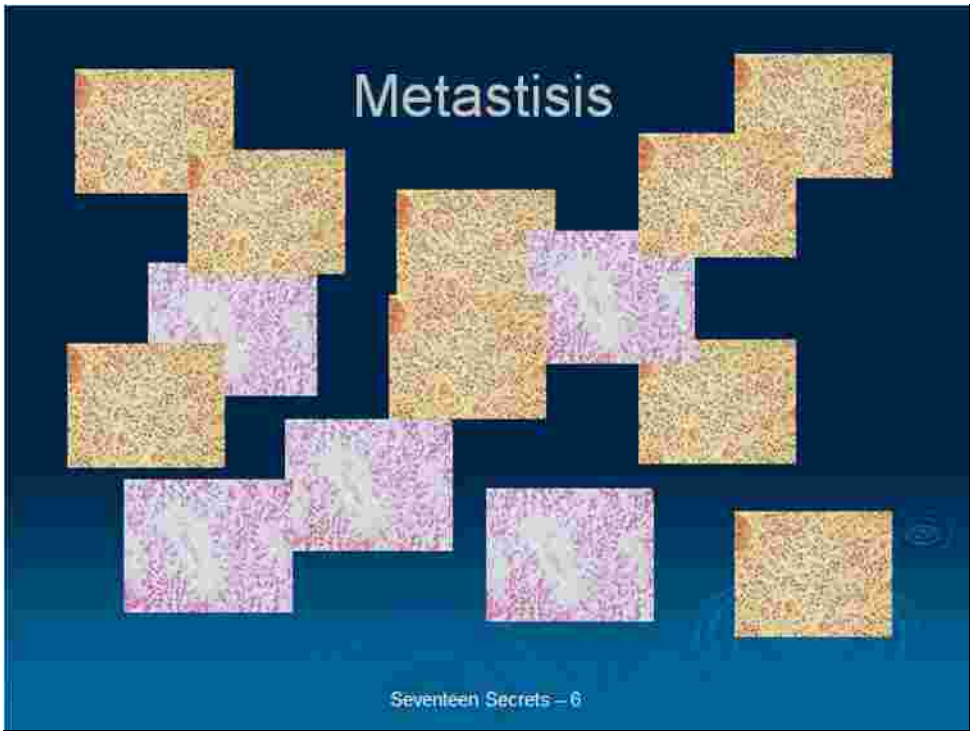
Get it working. And keep it working. Find a Pulse. Never let go.

If **you** go without a pulse for 5 minutes or so … brain death ensues…

Can the same be said to be true of a code base? It's a makeable case…

Finding and keeping a heartbeat is the underlying objective of automated testing frameworks and Continuous Integration Systems.

My favorite toast: L'Chaim. To Life.



Edward **Abbey**, environmental activist and writer. "Growth for growth's sake is the **ideology** of the **cancer cell**. **...**

Be warned: Uncontrolled growth, unconstrained duplication, can quickly turn malignant. Orderly, well-cultivated code is the result of keeping such impulses in check.

[There is a wonderful tale of the upside and downside of Copy and Paste to be told, but not here]

#16: Habitability

==========

Does anyone recognize the luminaries on this slide? The woman on the left is Adele Goldberg, who led the effort that led to Smalltalk-80 at Xerox PARC.

If you guessed the guitarist on the right was Jerry Garcia, you are … well, closer than you might think. He's Lisper/hacker/entrepreneur/poet Richard P. Gabriel, who has, as it turns out played some of Garcia's material in his own bands.

I cut my object-oriented teeth on Smalltalk-80 back during the '80s. I can only say that spelunking around the Smalltalk-80 image was the best apprenticeship in object-oriented programming, design, even patterns that a boy (or girl) could ever have had. You worked among the objects out of which the system was built. And you could roam around the system and browse through them all. You gazed at artifacts with an elegance and simplicity that has inspired their successors, but is yet to be equaled by any of them.

Key to the cultivation of this codebase was the ethos that the programmers

building it lived in / with / among the code they were cultivating, while they were building it. This was inevitably iterative and incremental, of course, by its very nature. And it gave the codebase itself a quality that Dick Gabriel calls "Habitability"

Habitability is the characteristic of source code that enables programmers, coders, bug fixers, and people coming to the code later in its life to understand its construction and intentions and to change it comfortably and confidently.

Habitability makes a place livable, like home.

It's related to a concept Alexander calls "organic order". The kind of order achieved when there is a perfect balance between the needs of the parts and the needs of the whole… [This sentence may have come from the Habitability Wikipedia page]

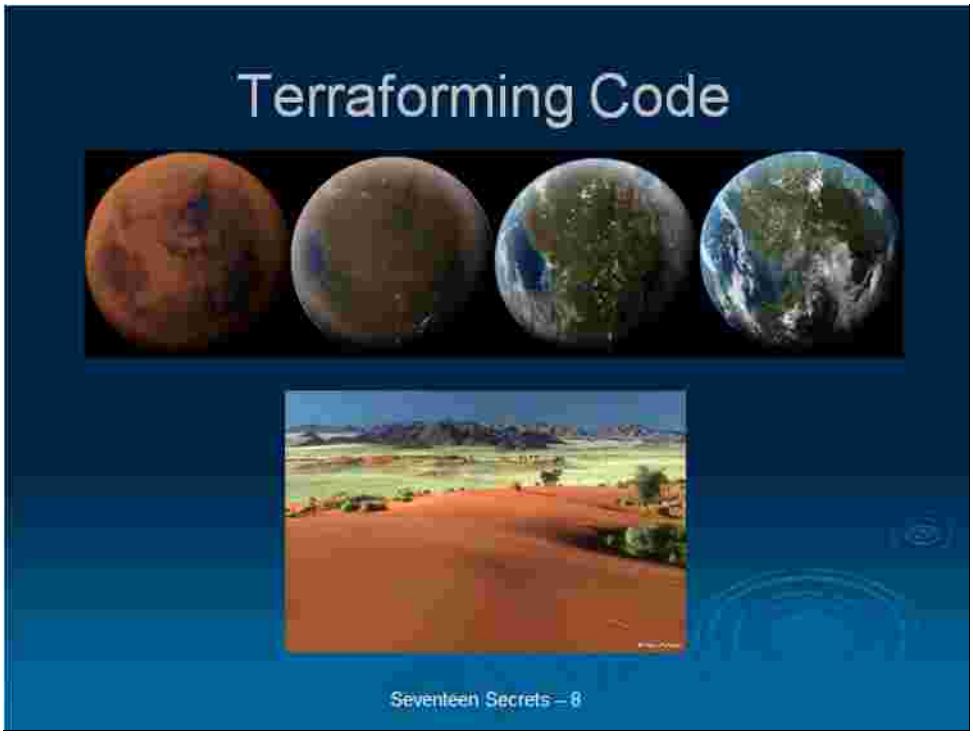
[Gabriel coined the term "Habitability", in relation to codebases, at least, in his book: Patterns of Software: http://dreamsongs.net/Files/PatternsOfSoftware.pdf. The Bad News is that this book has gone out of print. The Good News is that the copyright reverted to the author, and the PDF is available as a free download from the author's personal website.]

The Codebase is the Construction Site at which your arrive for work every morning. Oh wait, we are supposed to be trying to, as Brook suggested, deprecate the building and construction notion in favor of organic growth. Let's call it the Banzai Garden you've come to tend. The Cotton Field you've come to till. The Vineyard you've come to toil in. You get the idea.

Call it what you may, I think any serious software developer recognizes immediately / instinctively that there is a Sense of Place in the codebases they inhabit. That place can be a fetid, featureless, foreboding, bewildering, trek less quagmire, or a thriving, congenial, vital, familiar landscape, alive and buzzing with activity. In either case, (I'm convinced that) Navigating a codebase marshals many of the same metal facilities / capabilities that navigating a real space does. Our left-hemisphere verbal and analytical skills are buttressed and complemented by our right-hemisphere's navigational and visual pattern recognition abilities. When these skills work in concert they

enable us to easily locate those locations in our habitat where a job needs to be done, and quickly get to work.

As a result, both the literary and structural virtues of a codebase contribute to its Habitability, or lack thereof.



The rationale for terraforming, converting a utterly hostile, uninhabitable environment into somewhere human beings can flourish, is similar to that of for making code bases habitable…

Dick Gabriel calls a code base under control a Habitable Codebase. It's not perfect, but its somewhere where human beings can live, work, and, eventually flourish.

In the most extreme cases, bringing a codebase under control may be tantamount to terraforming.

I think the thing I like the most about this perspective is that it recognizes that habitability is a quality that needs to be evaluated / considered from the perspective of the people who live and work there…

A codebase that is hostile to its inhabitants is one that is devoid of recognizable, comprehensible landmarks and discernable structure, bewildering to traverse alien, foreign, and resistant to change.

Any modifications are cumbersome, painful, expensive … and slow.

The big problem is that Internal Quality, Habitability, Health, Cleanliness, call it what you will, is visible only to those who are actually will to look at, actual set foot in, the codebase.

All to frequently, even “bottom-level” managers are reluctant to set foot there, managing by cartoons, metrics, and meetings instead. And their keepers will most like go to their graves w/o deigning to visit the enlisted men in The Trenches.

“I have mud in places no man should” – Mike Rowe

Dirty Jobs is a Discovery Channel show hosted by the Mike Rowe. Here’s how their website describes it:

***“DIRTY JOBS** profiles the unsung American laborers who make their living in the most unthinkable — yet vital — ways. Our brave host and apprentice Mike Rowe introduces you to a hardworking group of men and women who overcome fear, danger and sometimes stench and overall ickiness to accomplish their daily tasks.”*

--http://dsc.discovery.com/tv/dirty-jobs/

Episodes typically depict Rowe stepping into, for a day, the shoe, or boots of folks who vacuum out porta-potties, clean out pigsties, scrape the barnacles off of barges, you get the idea…

I’ve long thought that if we could just find a way to visualize, to vividly depict, the sheer, fetid, foul, disgusting horror of trudging through a Legacy Codebase, day after day, that it would make a superb Dirty Jobs episode.

We'd need a CGI budget of … I'm not sure how much. Shows like this spend a bundle on CGI for less revealing graphics… Hmmm.

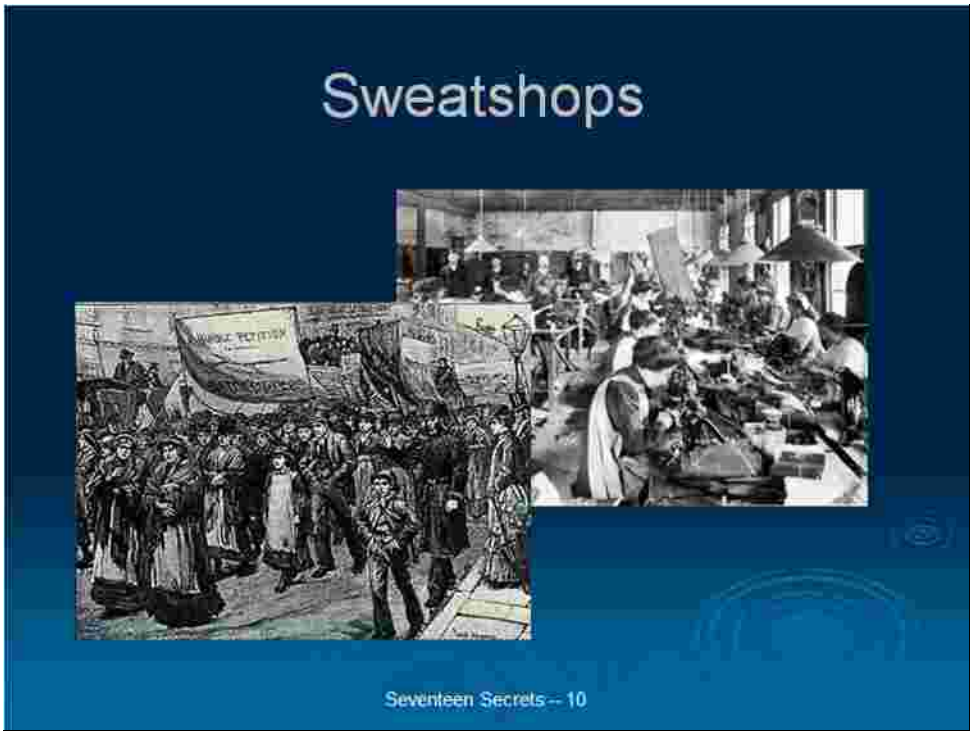
Still, I can just hear Mike now, opening the hatch leading to the code … Good God Man, that''s Foul! That's really revolting! You're not going to tell me someone has got to go in there! The stock formula on Dirty Jobs would be for Mike's guide, presumably a grizzled Legacy veteran, to reply, with a grin: "That's Right Mike, You Do!"

==========

Maybe we as a community have just been in denial. In denial that Mud is the #1 most frequently deployed architecture. In denial that working a legacy code base is intrinsically, inherently, a Dirty Job. And moreover, since most programming these days involves "pre-existing" code, we can extend this claim to … well … programming. Maybe Programming is just a Dirty Job.

Now workplaces like the ones that make it to Dirty Jobs often elicit a certain Male Bravado among those who work there. "Whadaya mean you can figure out where to change that! Weenie!". "Afraid to get a few bits under your fingernails, Mr. MBA!". One has to wonder if this is part of what has given software careers the lack of appeal they seem to have for those not burdened with a Y-chromosome…

Is it the case that programming is, for most of us, an inherently dirty job? Is it "prissy" to think otherwise? Buck up, clean freaks…



Given that you buy that the codebase is … A Place… A Workplace at that… Then, it follows that for many programmers, working conditions can only be described as *disgraceful*.

Over the last fifteen years, much of the world has banished tobacco from public places like offices, restaurants, even saloons. As gratifying as the 2/3s or more of the populations who don't use tobacco may have found this, in the US at least, the pretext for these bans was not the comfort of others per se, but the finding that second-hand smoke was a health hazard to the people who work in it.

It's amusing to speculate as to whether one could make a case for internal code quality / a ban on mud/ on the basis that wallowing in the mud is/ the appalling conditions found in many legacy codebases are / a threat to sanity, blood pressure, morale, and well-being of those who toil there.

What with the decline of organized labor, outsourcing, and the focus on the bottom line, this is most likely too much to hope for…

Like I said … It's amusing to speculate… [Were is Upton Sinclair when you need him.]

#15: Reconnaissance

Object-Oriented Reengineering Patterns, by Serge Demeyer, Stephane Ducasse, and Oscar Nierstrasz, is one of my favorite books on coping with legacy code. http://scg.unibe.ch/download/oorp/

The Bad News: this overlooked gem went out of print a few years ago. The Good News: the rights reverted to the authors, who've posted the PDF for the entire book at the URL shown, where it can be downloaded for nothing. It's written in pattern form, and chocked full of good advice. One could easily and fruitfully devote an entire presentation to these ideas. By all means, download it now, if you haven't already.

The patterns have titles like Read All the Code in One Hour / Chat with the Maintainers / Speak to the Round Table.

I'm going to focus here on / showcase / a pattern called First Contact. There, they proposed we "Read All the Code in an Hour"

It happens that about a decade ago Ralph Johnson's Software Architecture Group (of which I was an adjunct member at the time) reviewed some early drafts of this material. When we read "Read All the Code in an Hour", I remarked that I was reminded of an old Woody Allen quite. He said he'd just taken one of those "speed reading" courses, and that he'd read "War and Peace" In an hour. It was about Russia.

They used this in the book. Of course, in any sizable system, the authors are under no delusions you'll carefully read all the code. They are proposing that you merely do some initial, broad, cursory reconnaissance, to get a rough lay-of-the-land.

==========

Mike Hill (the forlorn-looking soul in the upper-right hand corner of this slide), aka @GeePawHill like talk about something he calls the "Legacy Stance". It's a air of defiance in the face of the bewildering reality that not only have you no detailed knowledge of what a system is like at the onset, but that you never will have such a command… ….ever… And will prevail regardless… …nonetheless…

==========

Have you had the experience of traveling alone to a city where you don't speak the language, or even read the local alphabet? It's a humbling experience to be stripped of the ability to utilize all the navigational skills that rely on language, and have to rely on sight, sound, and even scent. It's like being back in kindergarten. Your initial forays are timid, perhaps exploring a major artery, or streets nearby your "base". But, as time goes on, you become more confident, recognizing parks, turns, stores, restaurants, signs, colors, trees… Your mind fills in the details, and each expedition becomes successively more bold. After a few days, you look back and find that initial overwhelmed confusion has given way to a sense of familiarity, even comfort…

Learning a legacy codebase … a foreign place indeed, has this same kind of character…

==========

But back to the travails of the French. In 1803, US President Thomas Jefferson agreed to pay a cash-starved Napoleon Bonaparte $9,375,000 for the Louisiana Territory.

Congress subsequently approved an additional $2,500 to send a expedition, to be led by Meriwether Lewis and William Clark, to this territory despite mild resistance from the Federalists, who saw no point in spending money in the West. $969 was earmarked for gifts to the Indians. [And you thought earmarks were a contemporary invention.]

On 14 May 1804, the Corp of Discovery embarked from Camp DuBois, in the Illinois Territory and headed west. Lewis and Clark were skilled navigators, and the latitudes and longitudes of St. Louis and the mouth of the Columbia River were well established, hence the expedition could be quite confident in the overall distance that needed to be covered. They estimated that it would take them somewhere between 139 and 167 days to reach their destination. Four to six months, more or less.

The Corp of Discovery as some of you may know, missed these deadlines. They finally reach the mouth of the Columbia on 24 November 1805, after a journey of 559 days or approximately 18 months. Their estimate was off by more than a factor of three. [5/14/1804 11/24/1805 = 559 days Their guess 139-167 559/139 = 4.02, 559/167 = 3.35]

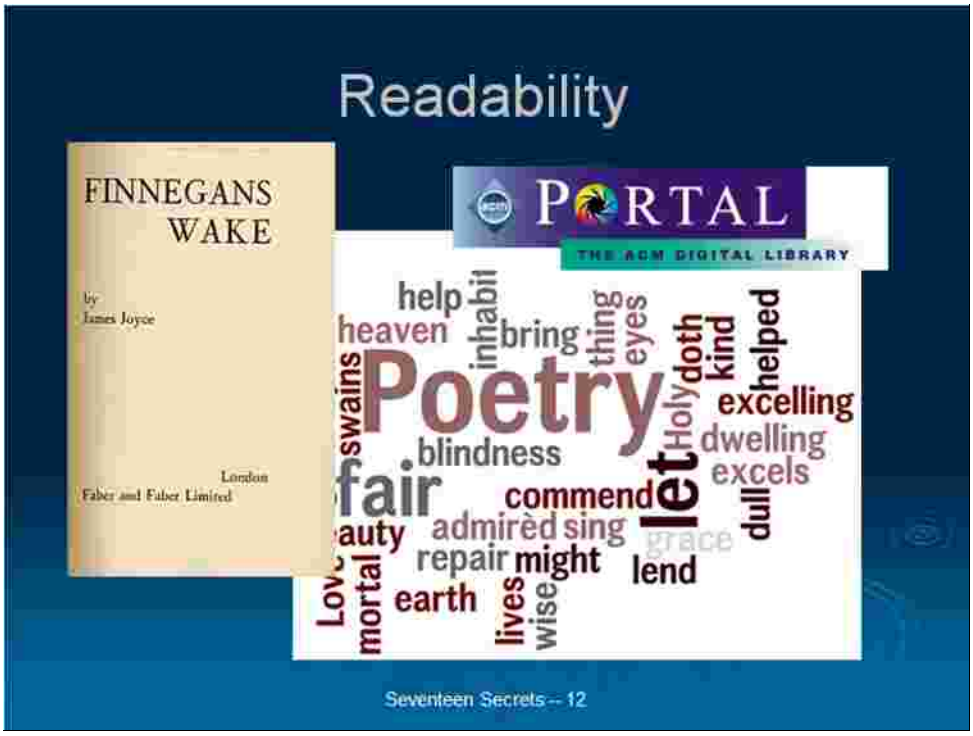
So they were far from on time. Under budget? Their initial estimate: $2,500. The actual cost: $38,722. [Cost Overrun: $38,722/$2,500 = 15.09]

We're they alive today, promising carreers in Software Project Management might await them.

I might add that their revised estimate of the time for the return trip was spot on. I helps to have done one already. Hindsight is better than Foresight. Paddling with the current probably didn't'hurt either.

==========

We need a Homestead Act for code. The government should give you 640kloc of code if you and your family agree to move in and cultivate it for 5 years…



If I may be permitted a brief digression, let's talk a little more about what it more about what we mean when we say code is "readable".

So:

Q: What do poems, 'blog posts, and programs have in common?

A: People seem to enjoy writing 'em more than they enjoy reading 'em.

[Should I add mathematical proofs?]

I used to say that if I needed to bury a chunk of prose where I could be confident no one would ever see it, I'd either stick it in [my | someone in the audience's] blog, or submit it to [the Communications of the ACM | the ACM Digital Library | POPL].

Is a page of code easier or harder to read than a page of prose?

That would depend on the prose, you say?

Fair enough? Dr. Seuss? Hemingway? Joyce?

What's a fair analogy? Textbooks? Math textbooks? Poetry?

Let's be frank with ourselves. The software industry has a dirty little secret, and that is: For all practical purposes, code, most code (not all code, but most) is effectively unreadable. Okay, there are still some wiggle words in there. Coward, you are thinking. Okay fair enough. Let me come out and say it: Code is Unreadable. For any reasonable, traditional, accepted meaning of the word "readable"…

Don't get me wrong, there is a difference between practically unreadable, and utterly incompressible. Bear with me.

So, nobody reads it. Once again, as a first approximation. There are actually two reasons for this. One is, it its, as we have finally, belatedly conceded, um, not really readable.

The other is … there is so damned much of it out there.

So, when you are (pretending to) read code, how much is too much?

Oh, 100 lines? Sure, we can all plow through that, right? Unless its PERL, right? Let's say we still printed code. How quaint, you say. Work with me here.

Let's pick an easy number to work with, like, say, 50 lines/page. A let's say, once again, for the sake of argument, that it a page is 11.5" tall.

So, 100 lines? 2 pages…

1000 lines? That's 20 pages. 19'. You're probably hanging in there (though we lose the Haskell guys somewhere about here.) 20 pages is probably a good upper bound for the length of a typical Computer Science technical paper. And we all know how easy those are to read. "Trivial" is the term academics customarily use to describe the difficulty of such tasks.

Who thinks 20 pages of code is harder to read than a 20 page Computer Science paper? Depends on the code, and the conference, right?

How ‘bout 5000 lines. That’s the wheelhouse for the Functional Program guys. That’s 100 pages.

At 40,000 or 50,000 lines, you are starting to get up to around the amount of code a lone wolf 1*10**0 team size [I have mentioned that that is my favorite team size, right?] can realistically ride herd over. I maintained a framework about this size for about ten years. It might have been five times that size had it not been a framework, but I digress.

50,000 lines is about a thousand pages. This is about the size of one very large novel, or two or three shorter ones There about 40,000 verses in the Bible, for instance.

But, I think, a few outliers aside (and yes, Richard Stallman, I am talking about you).

For convenience, let’s call 1MLoC a Stallman. A good weekend for Stallman, but a formidable helping for your typical mortal. Let’s posit, then, that the typical hacker can comfortably command 40 or 50 millistallmans, uncomfortably, what? 100-250 mRMS?

Beyond that we enter the realm of collaboration, and ignorance, and of Mike Hill’s Legacy Stance.

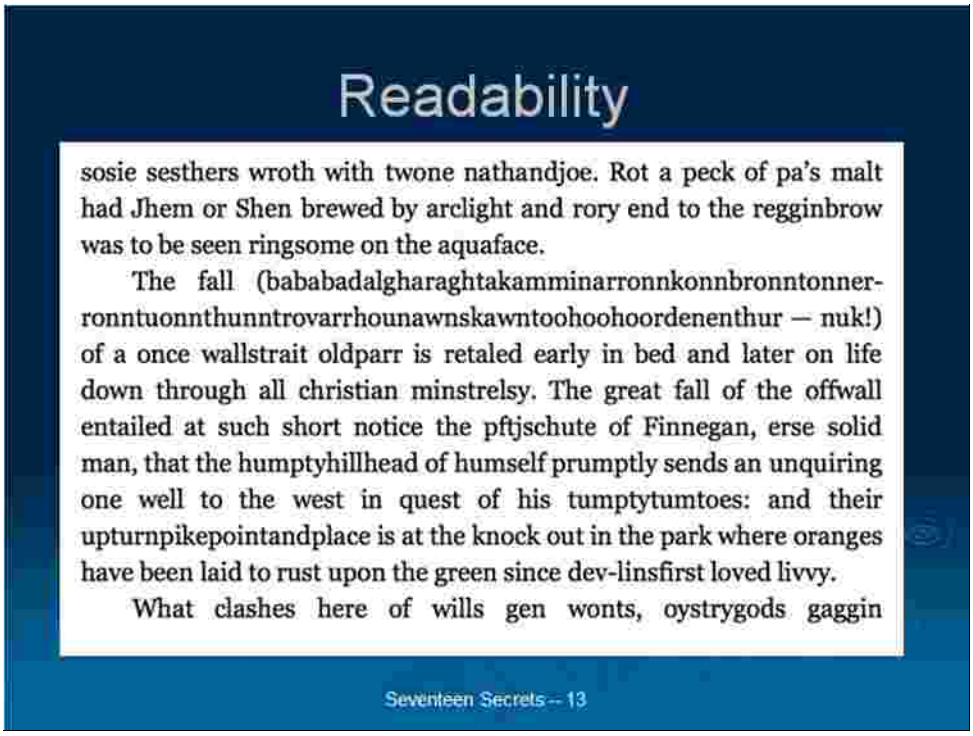
==========

The table below shows the sizes for listings, at 50/lines per page, on 11.5” paper, for listing of codebases of lengths from 100 to 1G LOC. The right had column gives metric units [for the Francophiles in the audience].

LOC (xRMS)	Feet	Inches	Miles	Pages	Kilometers
----	-----	---------	------	--------	----------
100 uRMS	1.92	23.00	0.00	2.00	0.00

1 mRMS		230.00		20.00
	19.17		0.00	0.01
5 mRMS		1,150.00		100.00
	95.83		0.02	0.03
10 mRMS		2,300.00		200.00
	191.67		0.04	0.06
20 mRMS		4,600.00		400.00
	383.33		0.07	0.12
40 mRMS		9,200.00		800.00
	766.67		0.15	0.23
50 mRMS		11,500.00		1,000.00
	958.33		0.18	0.29
100 mRMS		23,000.00		2,000.00
	1,916.67		0.36	0.58
200 mRMS		46,000.00		4,000.00
	3,833.33		0.73	1.17
500 mRMS		115,000.00		10,000.00
	9,583.33		1.82	2.92
1 RMS		230,000.00		20,000.00
	19,166.67		3.63	5.84
2 RMS		460,000.00		40,000.00
	38,333.33		7.26	11.68
5 RMS		1,150,000.00		100,000.00
	95,833.33		18.15	29.21
10 RMS		2,300,000.00		200,000.00
	191,666.67		36.30	58.42
20 RMS		4,600,000.00		400,000.00
	383,333.33		72.60	116.84
50 RMS		11,500,000.00		1,000,000.00
	958,333.33		181.50	292.10
100 RMS		23,000,000.00		2,000,000.00
	1,916,666.67		363.01	584.20
300 RMS		69,000,000.00		6,000,000.00
	5,750,000.00		1,089.02	1,752.60
900 RMS		207,000,000.00		18,000,000.00
	17,250,000.00		3,267.05	5,257.80

1 kRMS		230,000,000.00		20,000,000.00
	19,166,666.67		3,630.05	5,842.00



James Joyce's Finnegan's Wake is regarded by literary scholars (to say nothing of any of the rest of us who have ever tackled it) as one of the most difficult works in "English" ever written.

So, I ask you: Is code easier or harder to read than this?

This comparison is Apples and Oranges, of course.

==========

For the record, the (more or less randomly chosen) passage shown on the slide is:

I mean, are you kidding? I've never read it. Not even tried.

Frankly I don't' read much fuction, but when I'm in the mood for fiction, my taste tends more towards Programmer's best estimates of the time it will take them to do nearly anything… ….well, at least the first time…

Anyway, here it is:

sosie sesthers wroth with twone nathandjoe. Rot a peck of pa's malt had Jhem or Shen brewed by arclight and rory end to the regginbrow was to be seen ringsome on the aquaface. The fall (bababadalgharaghtakamminarronnkonnbronntonner-ronntuonnthunntrovarrhounawnskawntoohoohoordenenthur — nuk!) of a once wallstrait oldparr is retaled early in bed and later on life down through all christian minstrelsy. The great fall of the offwall entailed at such short notice the pftjschute of Finnegan, erse solid man, that the humptyhillhead of humself prumptly sends an unquiring one well to the west in quest of his tumptytumtoes: and their upturnpikepointandplace is at the knock out in the park where oranges have been laid to rust upon the green since dev-linsfirst loved livvy. What clashes here of wills gen wonts, oystrygods gaggin

--Joyce, James, 1882-1941 (2011-11-24T02:02:19.333140+00:00). Finnegans Wake (Kindle Locations 31-38). The University of Adelaide Library. Kindle Edition.

It might be entertaining to contrast this with a page from a "serious" author generally regarded as being easier to read, say Hemmingway, or Dr. Seuss, or both…

How many of you read, I mean really sight-read, music? Let me see those hands.

Okay, what is this? I've cropped the title and author information…

My point: We've been in denial. Code is effectively unreadable. Not utterly incomprehensible, but, for practical purposes, often effectively so.

Any of us choke on a generous enough helping. And then we put our game faces on … and well, put up a brave front…

But, I've digressed… Next, #14…

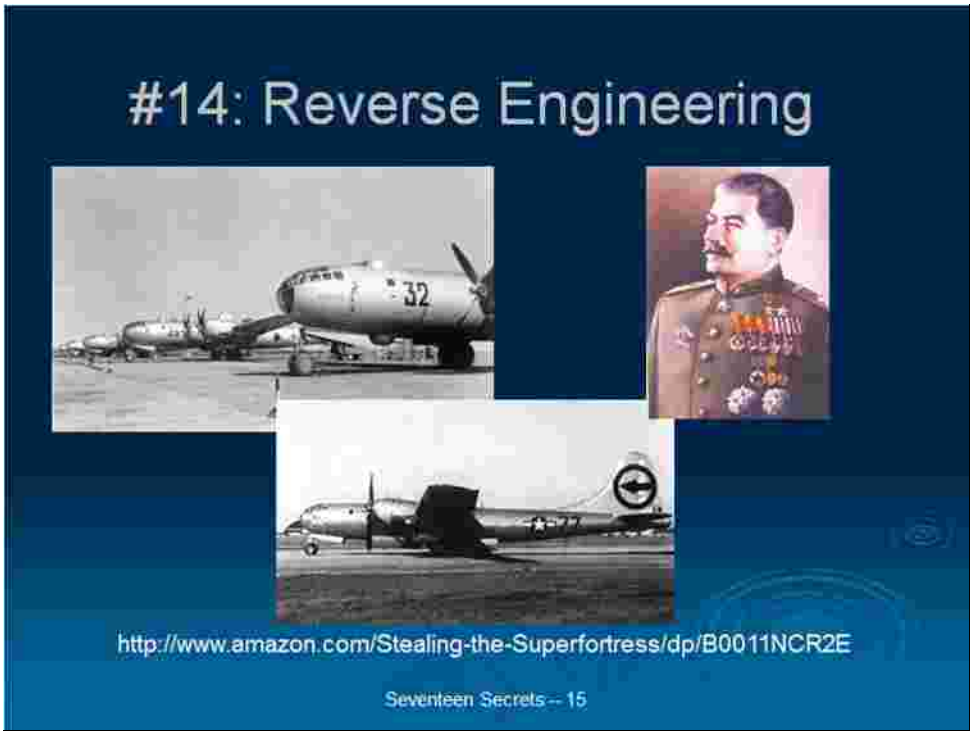
==========

Ludwig von Beethoven's best known etude, Fuer Elise. Many a piano student's first hurdle…

http://www.virtualsheetmusic.com/images/first_pages/BIG/Beethoven/ElizaFirst_BIG.gif

http://www.youtube.com/watch?v=o0VwTw1eZ1k

<object style="height: 390px; width: 640px"><param name="movie" value="http://www.youtube.com/v/o0VwTw1eZ1k?version=3&feature=player_detailpage"><param name="allowFullScreen" value="true"><param name="allowScriptAccess" value="always"><embed src="http://www.youtube.com/v/o0VwTw1eZ1k?version=3&feature=player_detailpage" type="application/x-shockwave-flash" allowfullscreen="true" allowScriptAccess="always" width="640" height="360"></object>



No one in this room (most likely) is old enough to directly recall that towards the end of the World War II, the United States was at war with Japan in the Pacific, but the Soviet Union, led by the fellow in the upper right hand corner of the slide, Josef Stalin, was, official neutral, up until the very very end.

The aircraft at the center of the slide is a Boeing B-29 Superfortress. Two such aircraft, the Enola Gay, and Boxcar delivered the first atomic bombs, Little Boy and Fat Man, that … hastened the end of the conflict.

Shifting geopolitical tensions being what they were, Stalin was, at the War's end, keenly interested in acquiring both atomic weapons, and the means to deliver them.

Now, Western experts estimated that it would take the Soviets something like ten years to "catch up" to the point where they could produce an aircraft with the level of technology of the B-29. So, they were dumbstruck three years later when, at an airshow in Moscow, one, two, three, then four aircraft (the last configured for passengers) flew past the reviewing stands.

How could this have been? Well, it turns out this was far from a clean room job.

Before the end of the war, several American B-29's had made emergency landings in (neutral) Vladivostok, in the Soviet Far East, after missions over Japan. Stalin dutifully repatriated the crews (eventually), but … held on to the planes.

After the war, Stalin ordered legendary aircraft designer Andrei Tupolev, on penalty of death (let's just say Stalin was something of an Old-School Manager) to clone the B-29. Not to use it as inspiration, not to merely examine it, but to clone it, bolt-by-bolt, inch-by-inch. Which meant, among other things, that because the Soviet Union was a Metric nation (ah, the French enter our tale again), that English tooling needed to be developed, or … ahem … acquired.

The result: three years later, the Tu-4 took to the air. They'd successfully copied all the parts from the "worked" examples they had, except for the tires, which they were able to pick up in Mexico as War Surplus Salvage…

The moral: Sometimes, not all the time, or event often, working backward from a worked example … as distasteful as it may seem, … works…

[We'll come back to this when we look at "Gentrification" and "Demolition. But don't give that away to a live audience.]

==========

http://www.amazon.com/Stealing-the-Superfortress/dp/B0011NCR2E

http://en.wikipedia.org/wiki/Tupolev

#13: Metaphor [I'm thinking of renaming it "Storytelling"]

The antithesis of Mud is code that tells an Effective Story about what it Does. How it Works. What it is

For. It answers the question: What is Going on Here?

Code that doesn't do so thwarts / obstructs us with every step / at every turn.

Have you ever wanted to grab your code by the lapels, (if it only had lapels), shake it, and scream: Speak to me damn it!

The people on this slide are people who recognize the importance of Storytelling. Eric Evans, Don Knuth, Kent Beck, and Hans Christian Andersen, all but the last … when it comes to code.

Internationally acclaimed Code Cosmetologist Martin Fowler has memorably remarked (and I'm paraphrasing from memory here): Any damned fool can write a program a computer can understand … writing programs people can

understand, that's harder…

[or something like that. I should look it up. I'm sure Martin put it more eloquently; he is after all, a Pretty Good Writer, for an Englishman, anyhow…]

[I suppose his mugshot belongs here too… Sigh. Worry not, Folwer fans shall have their fill of 'im soon enough.]

Eric Evans, the Dean of the Domain-Driven Design community, has been an early an vocal proponent of the cultivation of what he calls an Ubiquitous Language drawn, in part from the inherent, indigenous entities in the domain, with a dollop of metaphor.

…and Kent Beck is of course, ubiquitous as well, in this tale at least, the rogue methodologist who made System Metaphor one of the Pillars of XP…

The point here is that code that is about things that humans can recognize as relevant to understanding is easier to keep up and understand that code that isn't…

==========

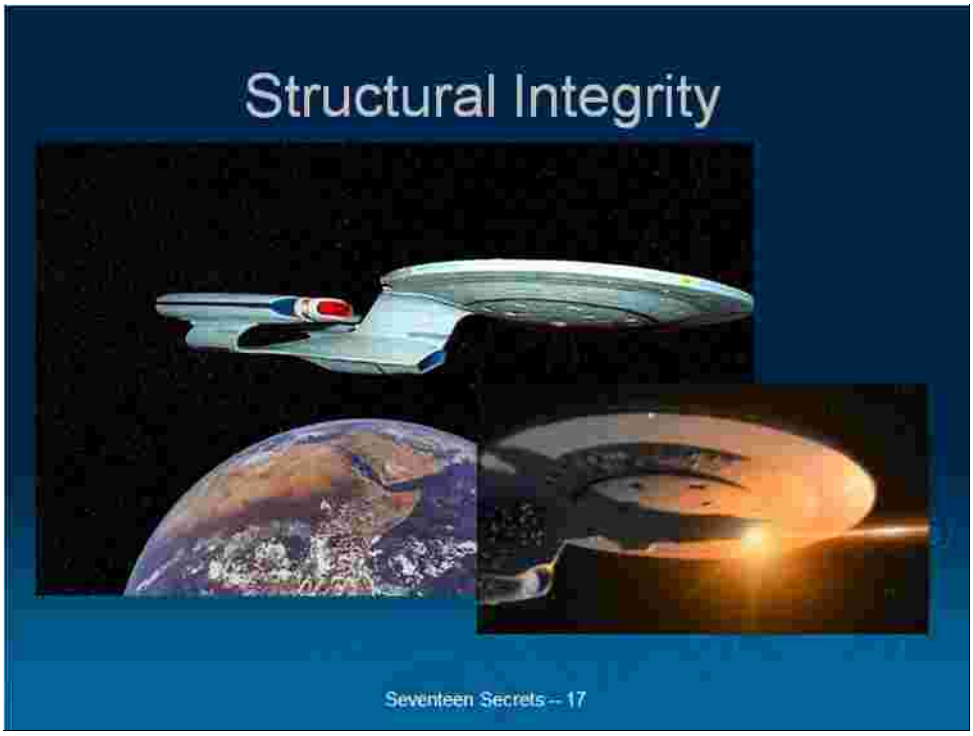
Here's a You Tube of the notorious Seinfeld "Master of my Domain" episode: http://www.youtube.com/watch?v=oi68hPMinAI

[I have thus far had the good taste not to use this to introduce Eric in combat…]

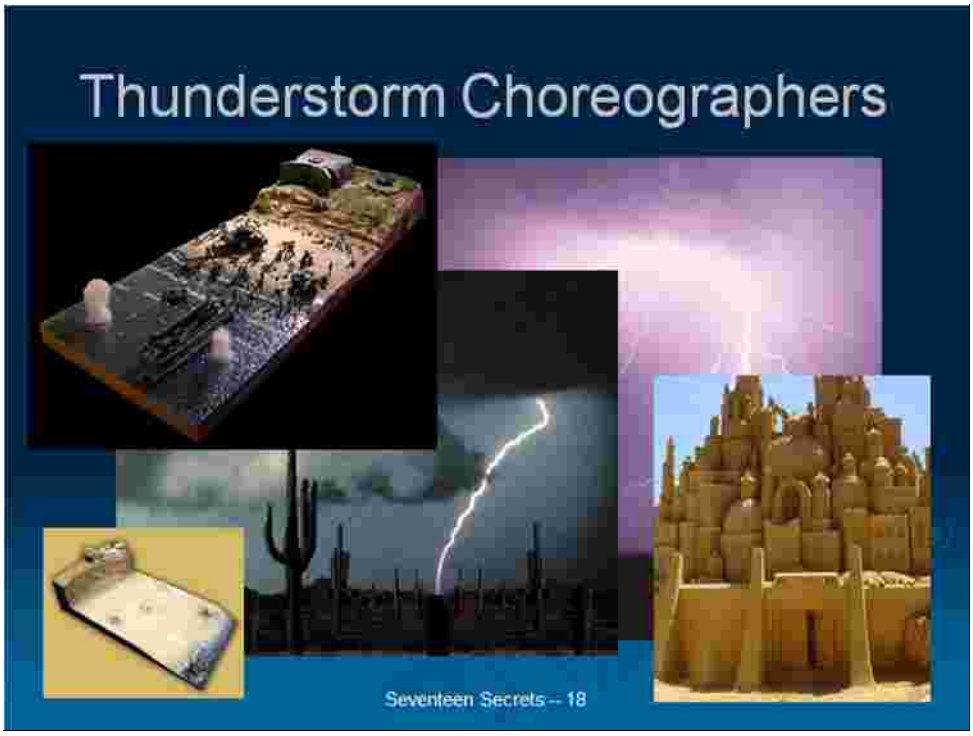
==========

When it comes to refactoring, Extract Method, Rename, … are among the most formidable weapons in the War on Mud.

I'm fond of going into client code blind, as a pair, or at a projector, and using these two to take a first cut at a code makeover on some Long Method. People are accustomed to code cleanups taking a long time, but it is amazing how quickly you can make palpable progress, when the wind is blowing in the right direction, with tools that support these refactoring…



I find myself using this phrase. A fair bit. Strange, eh.



What is it we really do? This is what we really do. This is the only part of what we are doing that's real.

We're meticulously, delicately, choreographing a miniature thunderstorm rolling through a delicately sculpted Lilliputian labyrinth across a silicon plain.

Every bolt, ever strike, pregnant, laden with significance, with meaning, A bonsai diorama in silicon and metal… A stage, and a dance. Every gesture scripted, every tiny, lingering bolt of lightning ripe, laden with Meaning… …to its masters, to us. Us!

A real thunderstorm isn't abound anything, but this one is. That little storm is real, the rest is in the realm of semiotics, storytelling and metaphor…

And Metaphor Matters.

…to you, Dear Reader.

We communicate that meaning through meetings, and documentation, and interaction during the building process, and design process, and in the artifacts themselves, in the variable names, and method names, In the narrative that the code itself conveys… it's part of the artifact, and is part of the institutional memory and culture of the people who build, cultivate, tend and care for.

#12: Patterns

And of course, we own a huge debt to these guys…

…these guys being the infamous Gang of Four, the authors of the 1994 megahit Design Patterns: Elements of Reusable Object-Oriented Software, shown, in the upper-left hand corner, from left to right: Ralph Johnson, Erich Gamma, Richard Help, and John Vlissides.

They are shown here as well eating their words at the 10th anniversary of the publication of their scurrilous screed, in 2004. It was, alas, John's swansong.

It fell to me, alas, to initiate the prosecution of these brigands for Crimes against computer science in 1999, and, we were able, thanks to the skilled and capable work of Lead Prosecutor Kent Beck (second from the left, who, yes, keeps showing up in this talk, for some reason), to secure their convictions on all charges.

The Design Patterns movement, of course, emphasized cataloging old ideas that other people had but had never written down, rather than, as was the custom in the academy, new ideas that you had had that no one else had written down yet. It was really quite a radical idea in its day.

Design Patterns, over the last 15+ years, has sold something like half-a-million copies, which means something like 1 in 7 programmers alive owns a copy…

==========

My favorite pattern for dealing with legacy wasn't even in the GoF Book. It's (Kent Beck's again) Composed Method. Extracting Methods as Composed Methods can be a quick route to additional clarity in a legacy codebase.

Façades can be helpful in cordoning off blight. Facades and Mediators can be useful when employing Eric Evan's Ubiquitous Language and Anti-Corruption Layer patterns…

The GoF notion of Factory has proven extremely influential, and when used as directed, can make systems more flexible and easier to configure.

==========

http://www.laputan.org/patterns/gang-of-four.html

http://www.laputan.org/patterns/trial.html

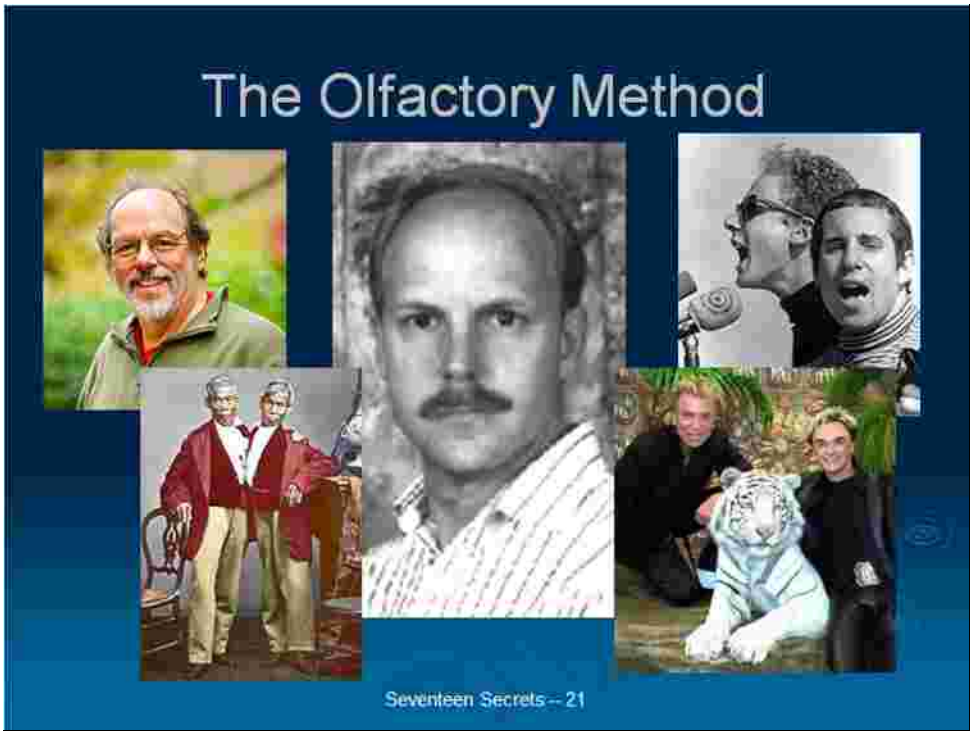
#11: Code Smells

I've made a habit of finding a way to fit at least one of my two favorite slides into all my talks. So here goes;

This is Laika, our Beagle-Bassett, named for the first dog in space.

To get away with this, I need a pretext, albeit flimsy, and, for this purpose, let me mention that the canine olfactory system is something like 10,000 times more acute than ours is…

[Sputnik II, launched on 3 November 1957, was the first to carry a living passenger of any kind into space. Alas, it was not the first to return one, heat-shield technology was not yet available. Sputnik II was a suicide mission from the start. Laika was Earth's first outer space martyr to "science".]



…what Kent Beck so vividly refers to as a “Code Smell”.

Kent Beck is one of the more remarkable figures the OO community has produced. I met him about twenty years ago when he managed to help get me an under-the-table copy of the Apple Smalltalk interpreter and image for the Apple Lisa. His seminal contributions are too numerous to mention… …but as intellectual paternity goes… …where do we begin?

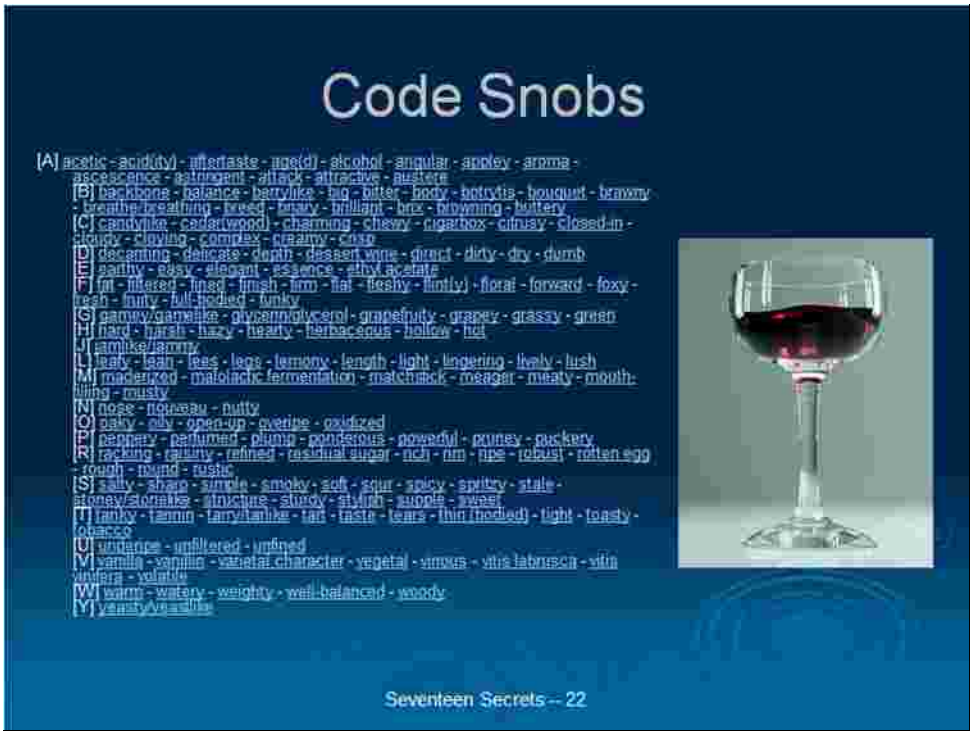
His brainchildren are numerous: father of the patterns movement, Smalltalk evangelist, with Ward, one of the original pair of pair programmers, SUnit, the first viable unit testing framework [in Smalltalk, no less], the Vince Lombardi of Extreme Programming…

Ward ‘n’ Kent became the best known duo in software, joined at the hip, doing to unruly code what Siegfried and Roy did with unruly felines.

However, it is sad that real genius exhibits / manifests itself in the gaps, at the borders, between disciplines.

It is for this reason that I think that Kent will be best remembered as the Founder of the field of Computational Scatology.

The notion of “code smell” is vivid, compelling, immediate… …ingenious. Beck veritably channels Marcel Proust.

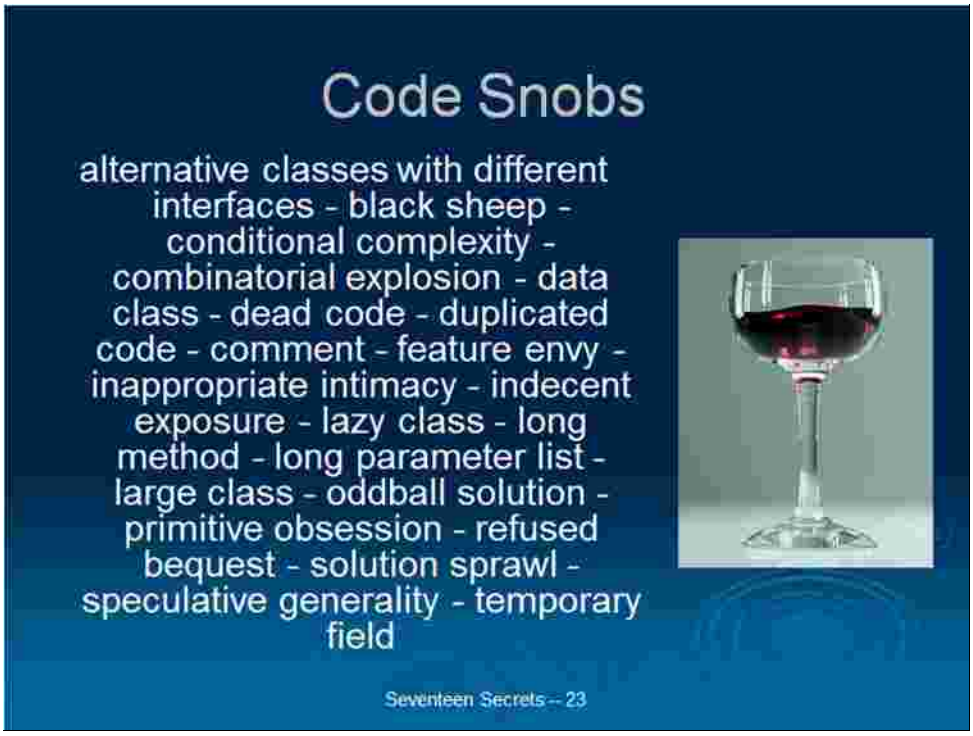


I'm from East-Central Illinois, which, to not put too fine a point on it, is not regarded as either wine country, or a hotbed of sophistication in general, for that matter.

We can pick up a glass of wine and mutter "not bad", which is high praise in our part of the country, or some utterance of disapproval, such as "this sucks", typically followed with an inquiry along the lines of "What beers do you have on tap?"…

So one can only imagine our chagrin when we visit places like Northern California, and are confronted with connoisseurs with elaborate oenological vocabularies the likes of which you see on this slide.

We cringe with envy when our betters at some tasting utter phrases like "Ah, this has a brawny, acetic bouquet redolent of well-balanced oakey funk, with a charming varietal pruney finish awash in tannin and vegetal lingering body", or some such.



And, those of us who are Software Professional might find ourselves thinking: “Damn, I wish we had a vocabulary like that for talking about Code”.

Well, thanks to Kent Beck, and a pair of particularly erudite Code Connoisseurs who came after him, Martin Folwer, and Joshua Kerievsky, we do.

We can Lord phrases over a Code Review like: This code is redolent of Primitive Obsession, exhibits an untoward weakness for Inappropriate Intimacy and Indecent Exposure, is engorged with Duplicated Code, and Solution Sprawl to boot, much to the amazement, awe, and occasional irritation of our peers…

Being able to talk about what’s wrong with code is an important and underappreciated step on the road to making it better, hence: Code Smells weighs in at #11.

#10: Sunlight

it's the best disinfectant around. The open source community likes to claim that with enough of these, any bug is trivial. Has this helped with code quality? The jury is out, but there is some evidence that it has.

One of mud's most effective enemies is sunshine. Subjecting convoluted code to scrutiny can set the stage for its refactoring, repair, and rehabilitation. Code reviews are one mechanism one can use to expose code to daylight.

Another is the Extreme Programming practice of pair programming [Beck 2000]. A pure pair programming approach requires that every line of code written be added to the system with two programmers present. One types, or "drives", while the other "rides shotgun" and looks on. In contrast to traditional solitary software production practices, pair programming subjects code to immediate scrutiny, and provides a means by which knowledge about the system is rapidly disseminated.

Indeed, reviews and pair programming provide programmers with something their work would not otherwise have: an audience. Pair-practices add an element of performance to programming. An immediate audience of one's peers provides immediate incentives to programmers to keep their code clear and comprehensible, as well as functional.

There is considerable anecdotal evidence that open-source projects produce code with a

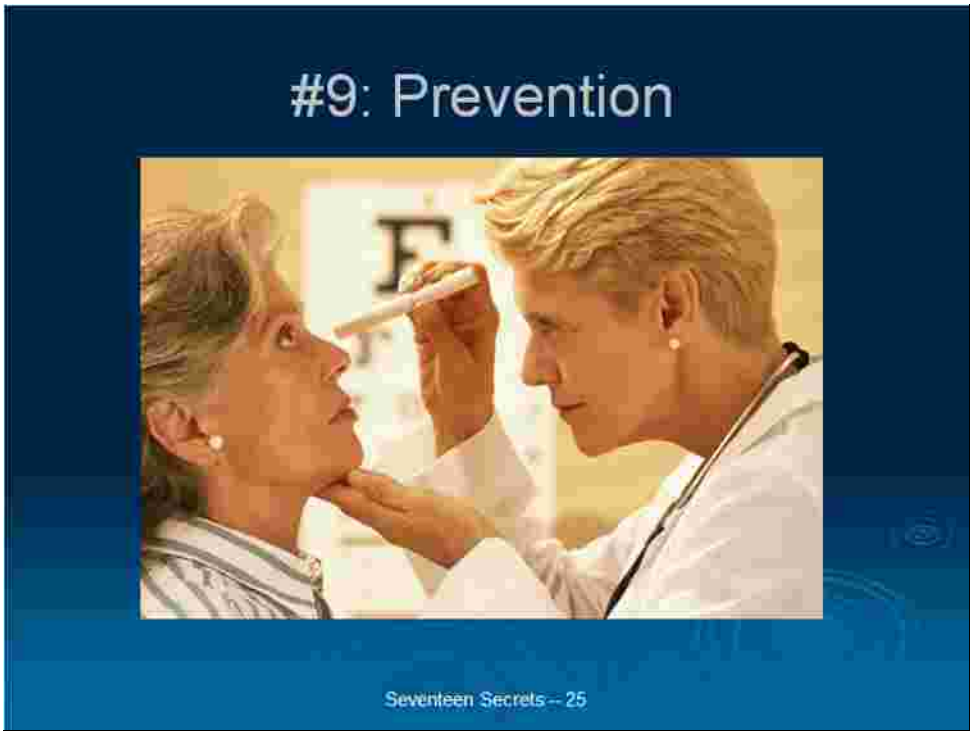
higher level of “internal quality” (readability, maintainability, health, etc.), that do closed shops, especially “enterprise” shops. Empirical comparisons are still hard to come by, alas…

Eric S. Raymond famously said, given enough eyeballs, all bugs are trivial. [Of course, given enough fingers, all programs are brittle, but I digress.]

[Sunlight turns Mud to what? Dust? A desert instead? A dust bowl… The dust bowl is a good metaphor. But for what.]

==========

http://www.catb.org/esr/writings/homesteading/



#9: Prevention

Legacy Assessments too often carry the same kinds of ominous, foreboding trappings as does an Oncology Consult. For, all too often The News … is Not Good.

I've made a habit over the last few years of asking decorated veterans of the Legacy Trenches questions like: "What's the sickest codebase you've ever seen get better?", and "Have you ever seen a Ball of Mud recover?". And, it is my grim duty to report, the News has not been good there either..

In fact, one hears the same advice from many Specialists in the area, from Software Oncologists and Gerontologists alike: Please! Don't let your code Go to Hell in the first place, because once it does, the prognosis is … rather bleak. As a rule, we just don't see systems that have succumbed / fallen prey to squalor, mud, and entropy … get better…

…but a number of these same Specialists have seen Healthy systems stay Healthy…

I am afraid I have some Bad News: The War on Bugs is Lost. In the generation since Nancy Reagan, consort to Ronald I, implored us to just say no to bugs, America's bug problem is worse than ever.

Bugs are like prime numbers. By now, we're pretty much resigned to the fact that we'll never find last one, but after the first billion or so they start to thin out a bit.

A determined, decades long "Just Say No to Bugs" campaign didn't work. They're still in our schools, our code. A handful of unrepentant formalists / zealots still believe they can be vanquished altogether. This premise has yet to be demonstrated beyond a few tiny, pilot programs…

Bug traffickers reap fortunes: we call them software companies. They hire lawyers, these bug cartels, I'd name them, but they'd cut me down in a hail of bullet … points.

The war on bugs has cost billions.

Bug tracking is a growth industry. Insects outnumbers us? Are bugs far behind? It's not a problem of enforcement. It's not something we can make disappear via, say, stronger type checking.

The supply of bugs is larger than ever, and attempts at domestic crackdowns have just driven bug production overseas.

The old story sounded something like this: You'd start with the Gateway bugs, first, you neglect to set a few error returns; you let a couple of exceptions get by unchecked, the next thing you know, its on to the NP-Hard stuff; a dependency injection overdose, some even risk deadlock.

New bugs are emerging, some 100s of times more potent than the bugs of twenty years ago, some nearly impossible to detect.

Concurrent and distributed programs […].

The War on Bugs has diverted attention from what works: TESTING, PREVENTION, and TREATMENT. [We're back to Beck's/Cockburn's medical metaphor.]

We're way beyond our formal logistic train. Treatment is more than checking your code into rehab, its about communicating with others that share your plight.

==========

[Okay, so this is a slide I'm still trying to figure out how to have some fun with. The point here is, for some, Quality is only about user experience, and outright malfunctions. Health and Internal Quality matter too. Quality is not just about winning an unwinnable War on Bugs, its about … Habitability, and Draining the Swamp… Something like that… One could bring Panama back into the story here…]

==========

That's Blake's "The Ghost of a Flea" in the lower right…

#8: Craft

The fellow wagging his finger on the left is well-known, unrepentant Francophile and Code Hygienist Bob Martin, Mr. Clean (with Hair), a prime mover in the North American manifestation of the Software Craftsmanship Movement.

The Crafts Movement has arisen over the last decade in response to the chronic, dismal condition of all too many contemporary codebases, and the lack of attention and concern being paid to the issue.

In one sense, of course, all the bullets on our list in this this talk / all the bullets in our chamber / are about Workmanship / Craft / Hygiene / Internal Quality / Well-written Code. Attention to, and concern for writing code the "right" way underlies the very premise for this romp… ….it's No Secret… …far from it… Concern over Quality is nothing new, to be sure.

Nonetheless, a shout out to this impulse, and to this movement, as well as to the issue, seemed in order.

And yet, it feels glib to say The Answer is: Just do it Right / Don't Write Crap. Because the question of, if this *is* the answer, why well-crafted code so relatively rare?

The morass of mud most software professionals find themselves mired in surely must be more than mass malpractice on the parts of … damned near the entire software industry.

[This slide was originally labeled "Workmanship", but I decided I liked the better known, and, moreover, more gender-neutral term "Craft" better (as a consequence of some combination of reflection and feedback, to be honest… I thought about "Hygiene" too… ….and will some more…]

In the long twilight struggle between the forces of Kraft and Kruft, I'm afraid Kruft is winning.

We need to better understand why this is. Merely bemoaning the fact that this is so is tantamount to capitulation or at very least, is a separatist, utopian agenda.

We live just north of Illinois Amish country, where Amish craftsmanship is often thought of as synonymous with Quality.

Is this always what we NEED? Why is pressboard popular? Because it is cheap.

Yet we all admire handcrafted hardwood. One reason, the quality is VISIBLE.

The Amish, of course, eschew modern tools and technology, and yet are quite shrewd about selected niches where quality matters, and where it commands a premium.

What would be the equivalent, quality assembly language/Fortran and VI? Craftsmanship is, commendably enough a high wage agenda.

The Amish are retro-artisinal, as Brian Marick would say…

Face it, a lot of people want hardwood quality at particle board / pressboard prices.

And a lot of others simply don't care. They have no idea what healthy code looks like, or of the appalling conditions in which all too many programmers must toil.

All too often, these are the folks calling the shots and paying the bills…

http://www.yodersamishfurniture.com/

Yoder's Amish Furniture, Greenwood, NE

http://upload.wikimedia.org/wikipedia/commons/thumb/a/a9/New_Harmony_by_F._Bate_(View_of_a_Community,_as_proposed_by_Robert_Owen)_printed_1838.jpg/400px-New_Harmony_by_F._Bate_(View_of_a_Community,_as_proposed_by_Robert_Owen)_printed_1838.jpg

Is Craftsmanship a quaint 20th Century affectation…

Even back then, programmers who took the time to "clean up" their code might be acused of "lily-gilding". Hell, I was.

[Guilded? A good pun….]

==========

Many years ago, in a time when televisions employed primitive vacuum tube technology, and Zenith employed Real Craftsmen to build TVs in Skokie, Illinois, Zenith famously had as their slogan: The quality goes in before the name goes on.

Handcrafted Quality, they trumpeted.

I'm old enough to remember when computers will still build out of circuit boards, some of which were soldered, with loving care, by hand.

We used to design, lay out, and produce our own circuit boards at the shop I

worked at at UIUC during the eighties.

But, the advent of integrated circuits and robotic circuit boards assembly eventually overwhelmed the old craftsmen.

Will once admired aspects of what we now think of craftsmanship soon head down that same road?

Henry Petroski has observed that attention to ornament and filigree is emblematic of / characteristic of / typical during the early stages of the evolution of new technologies. [As was variety in the market place, but that's another tale]. As the technology matured, and as hand-crafting gave way to assembly lines and automation, such flourishes begin to disappear.

Here you see the livers of a junior waterfall developer and a 51 year old scrum master [Joe Yoder's liver | someone in the audience].

Can you tell which is which?

Who was it exactly who said that code was supposed to be pretty? Who even sees it?

Why should anyone care about "internals", other than, perhaps, the people directly involved in working with them?

The simple fact is that it's the developers themselves, and maybe their first-line supervisors who ever look at the code.

Everyone else cares about … well, external quality sure… Is the GUI nice, does it work, is it buggy.

The case for internal quality / health / has to be made on the basis of more

than “programmers don’t like to work in that slop”…

And, that’s usually not the case that is made. Rather, the most common claim made for “clean” is that well organized, clear, comprehensible, code is easier to work on, less buggy, easier to maintain, and easier to extend. You can move quickly. You are not stuck in the Mud.

=========

If ways of better visualizing, better measuring, the horror show that many codebases have become could only improve, the case for healthy code might be easier to sell.

=========

Over the last decade or so, tobacco has been banished from workplaces in much of the world. Here in The States, at least, the pretext for this was not that the majority of the population found second-hand smoke unpleasant / odious, but that it was a demonstrable health hazard.

If only mud could be banished this way. Working in the kinds of squalid conditions many hackers must endure is arguably a threat to their well-being, even their sanity. Were such standards uniformly imposed and enforced, somehow, the people who “do it right” would not find themselves in the position of having to compete with the corner-cutters who don’t.

Using modern refactoring tools, developers can now not do what they should be doing one hundred times faster than they didn't do it before. And with greater accurately…

--**Donald Bradley Roberts**

==========

I had the privilege of being part of Ralph Johnson's Software Architecture Group at the University of Illinois at Urbana-Champaign during the time when he and his students did their seminal work on Refactoring.

Of course, programmers have been "cleaning up" their code since … the beginning… …and the practice of continuous, sustained code cultivation was common in the Smalltalk Community.

But is wasn't until Bill Opdyke's research began to surface in the late-'80s that a more refined notion of "refactoring" as a series of "behavior-preserving" code transformations entered academic parlance.

This research not only codified the kinds of transformations that might be made to existing code to improve its structure and clarity, but demonstrated

that under certain conditions, such transformations, these transformations could be performed semi-automatically.

[Opdyke is the smiling fellow standing next to the conference cruise ship captain in the upper right of this slide.]

Two of Johnson's students, Don Roberts and John Brant, went on, during the mid-'90's to construct the first practical Refactoring Tool, the Smalltalk Refactoring Browser.

[Brant is the tall fellow on the left of the "line-up" shot at the bottom of the slide. The others are Yours Truly, Roberts, Johnson, and Joe Yoder.]

This tool fell into the hands of Smalltalkers like Kent Beck and Martin Fowler [center-right], who's book "Refactoring" put refactoring on the map / made it a household word. In hacker households anyway.

Fowler, who, as I think I have already mentioned, is a pretty good writer, for an Englishman, carefully cataloged the refactorings he'd learned from the Refactoring Browser, and others, of course, and described step-by-step recipes for performing these refactorings in Java, which was then all-the-rage. This roadmap, in turn, help Erich Gamma [lower left] and his cohorts to incorporate real refactoring support into Eclipse.

The rest is history, as they say…

The ability to very quickly make complex improvements to existing code, with the guarantee that they will be performed correctly, in most cases, has been transformative.

The gentleman in the upper left hand corner of this slide is Malcolm Gladwell. In his prize-winning book “The Tipping Point” (the book that made the idea of “the tipping point” a household “word” / er, words), he told the story about how New York City, under Rudy Gulliani [shown to the left and right of the graffiti on the bottom of the slide] embarked on a campaign to no longer tolerate petty manifestations of urban blight like graffiti and broken windows. Over time, this lack of tolerance led to a reduction, and then, a sea change, in the prevalence of petty crime, pan-handling, squeegee-guys, etc.

Fix the little problems, one step at a time, and the big problems start to get better too…

This approach can work with code too.

The fellow in the upper-right hand corner is Joshua Kerievsky, whose Jolt-Cola Award winning book “Refactoring to Patterns” meticulously documented how to take blighted code, and clean it up.

#6: Quarantine

A few years back, I was teaching a design patterns course at a client whose name you would all recognize…

And the subject turn to a a large (13MLoC / that's how many Stallmans? Just seeing if you were paying attention…) legacy C++ codebase they were entrusted with keeping alive.

This code was at the core of one of their principal product lines. There is a good chance you use / come into contact with / it nearly every day.

This code had gotten *so bad* that every time anyone touched it, something else would break. Attempts to repair problems and fix bugs were adding more bugs than they eliminated.

It was horrible stuff, said those who'd been in there. Vile, fetid, festering, malodorous sludge. And yet, it was a cash cow. It's making them a bundle. Still. And it *works*. Millions

So, do you know what they did?

They effectively *quarantined* it. They froze it. That is, no is allowed to touch it, under penalty of termination, for any reason, without getting extra-special permission from the very highest ranks in this product's division. New features, if any, were added as wrappers or sprouts

outside the Monster, dealing with it only through its public APIs.

I’m not saying this is an ideal approach under all circumstances, but, that given some, it works.

#5: Demolition

During the ‘60s, a number of municipalities built the kinds of “Cookie-Cutter” multiple use stadia shown here. The idea was that these circular facilities could be configured for baseball, football, even concerts, and save money by sharing expenses. They’d save space too.

But everywhere they were built, these concrete cookie-cutters proved widely unpopular. They were lousy places to see a baseball game. They were lousy places to see a football game too. Ditto for concerts. Nearly everyone was far away from everything.

Over the last couple of decades, the economics of the sports have changed, and, one by one, these white elephants were replaced by single use stadia, and these multipurpose failures were blown up…

Sometimes, you just have to get rid of the old one and replace it. If you don’t build a better replacement, your competition will.

[The next few slides shamelessly pander to what experience has taught me will usually be a largely male audience by incorporating lots of pictures of stuff being blown-up.]

The air forces (the Luftwaffe and the US Army Air Corps (USAAC)) of World War II, in going about the courses of their businesses, provided some tragic opportunities / maybe urgent necessities is a better way to put it, to rebuild substantial swaths of cites like Coventry, Dresden, and Hiroshima from the ground up.

Sometimes, circumstances allow for no choice but to start over from scratch.

==========

The bottom shot is from the DOE website at:
http://www.cfo.doe.gov/me70/manhattan/images/HiroshimaRuinsLarge.jpg

These shots show Coventry, Dresden, Hiroshima today.

Certainly, much of the charm and history these cities once had was lost forever.

But, the opportunity to rebuild from the ground up presents many of the same opportunities for modernization as does a “green fields” start…

…They did it by making the **single worst strategic mistake** that any software company can make:

They decided to rewrite the code from scratch.

--Joel Spolsky

Why does Joel Spolsky call rewriting the code from scratch "The one thing you should never do"?

I think we all realize where the temptation is here. You are mired in some God-Awful tub of goo, you are bewildered, demoralized, disgusted, and want nothing more than …. to just blow it up.

You find yourself thinking: just let us write it over ourselves. We can do way better than these clowns did. And we'll understand why things are the way they are too. C'mon…

The problem is, you are probably overestimating yourselves, and underestimating "those clowns". That code is littered with stuff that is there for A Reason. And you have no way of reconstructing those intricate

compromises, other than through painstaking line by line analysis that may nonetheless never reveal underlying intent.

Remember, the old one Works. It has a pulse. Getting it on the air wasn't easy….

Ironically, while Spolsky's advice is good advice in general, his poster child example was Mozilla, which, in hindsight, is one of the few large code bases one can point to that actually has Gotten Better over the last few years…

#4: Gentrification

Given the perils of starting over, can we instead gradually improve the system, a part at a time, a step at a time.

The paradox: It's harder to get it back… ….once you've lost it.

One might argue that an engineering organization with the competence to pull off an overhaul wouldn't have gotten themselves into that kind of mess in the first place…

Still, look at it this way. If you want to make a system better, live in the running system, and fix it Incrementally, rather than replacing it entirely.

You can build each element using whatever modern methodology and toolset you wish, and graft it into the existing framework.

Transplant one organ at a time. You always (or almost always) have a pulse.

Renewal / Rehabilitation

You can usually find a way to live there while the work is going on.

Incremental Rehabilitation can be approached as incremental, fine grain demolition, or as a pure refactoring task.

Rehab is not for everyone, but if you need to make a codebase that is making you money more nimble, it's probably the most promising strategy available.

#3: Neoteny

Neoteny is idea from Developmental Biology whereby, in order to progress, an organism reverts to an earlier stage in its development.

For instance, the modern domestic canine is thought to have developed from wolves in part by arresting its development at a juvenile stage. Our dogs are effectively wolf pups who never grow up.

The broader idea is one I got from Arthur Koestler's "The Act of Creation". In order to progress, we revert to an earlier, simpler stage in a system evolution or development. Draw Back to Leap, he dubbed this tactic / stragegy.

A software example of this is Squeak and Pharo Smalltalk. These didn't develop from the mature PARC/Cincom codebases, but instead began with an earlier codebase that harkens back to the Xerox images of the mid-'80s. By starting with this less complex foundation, these systems were able to more easily set out and innovate in different directions than might have been possible had they begun with the more mature trunk.

Sometimes you can't get there from here, and you need to backtrack, an start anew.

==========

[Undifferentiated Stem Cells could be used to embellish this tale.]

==========

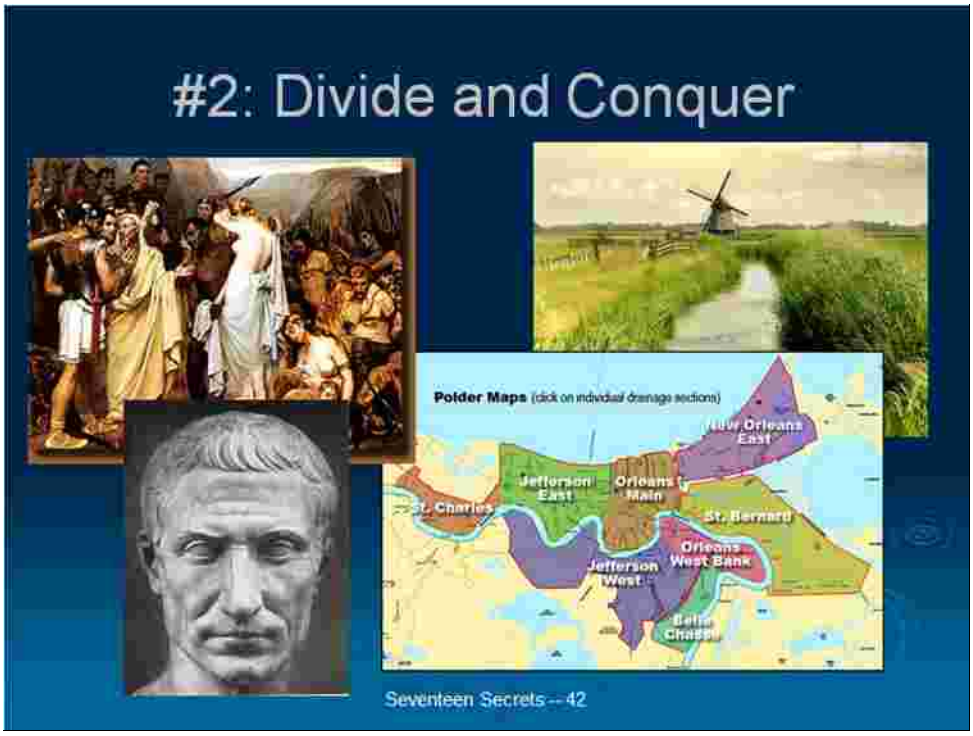
This idea is also consonant with the idea of Simulated Annealing and Annealing searches. These work by using "heat" to undo tertiary, and even secondary settled constraints, to allow the system to re-settle primary constraints in the context of a new fitness landscape.

[Hence, the furnace/Samurai Sword images.]

It's interesting to ask whether the kind of literary and structural features that we associate with well-honed / well-crafted code can be an obstacle to its evolution.

If structure emerges late, if refactoring for readability comes later, then effort to repurpose code may be impeded by artifice cultivated to make the code friendlier to humans.

It's a perverse thought, but doesn't it make sense to strip the human-readable veneer, muddy it back up if you will, to de-evolve it, to permit the code to re-evolve in a new direction?



#2: Divide and Conquer

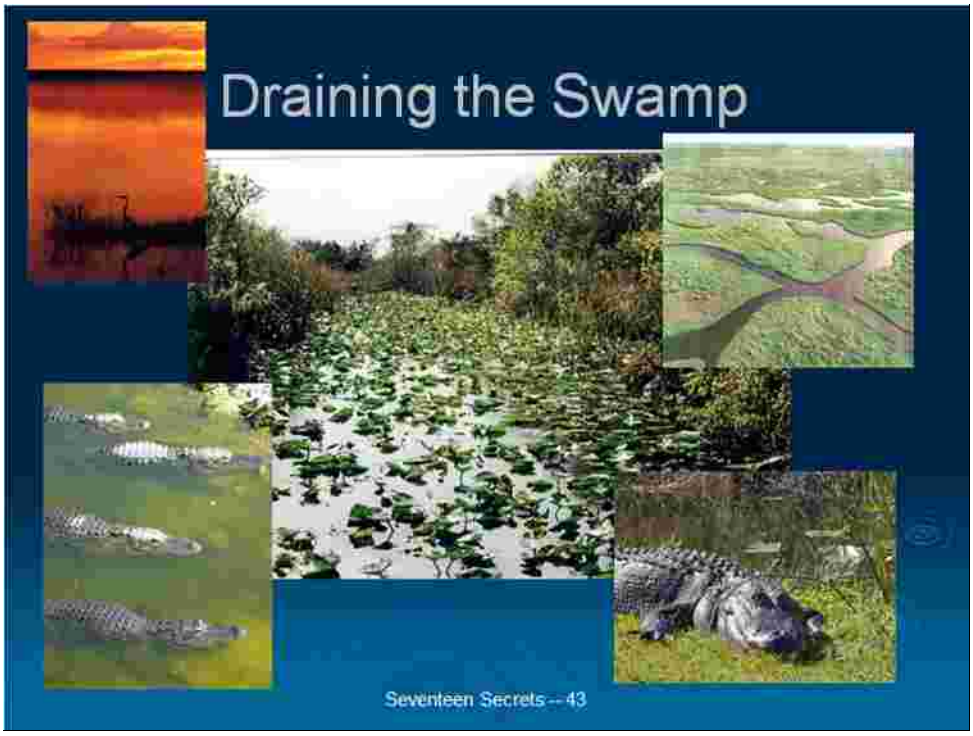
A fundamental strategy for tackling nearly any complicated task is to carve it up into smaller, simpler tasks.

When the Dutch want to drain a polder, they first build a dike to keep the sea out, and then, they subdivide it into Polders, and drain and develop them a piece at a time.

This scheme can be effective with the undifferentiated Kudzu entangled legacy quagmires that we must confront too.

Cordon it off, and develop one section at a time…

[Julius Caesar and “Victory of Caesar over the Belgians”…]



How do we Navigate a quagmire?

The Alligator Dilemma: [which I've seen referred to as "Foote's Law"] When you're up to your elbows in alligators, it's hard to remember that your goal was to drain the swamp.

Is this Space Navigable? Why yes. Where do you want to go?

#1: Testing

Could anyone NOT guess what #1 is going to be? [This question might better be asked at the end of the rap for the previous slide…]

There is a story I like to tell about machine problems from hell… …so tell it here:

Okay, here it is.

A few years back, I found myself back at the University of Illinois working on a NASA project to refactor a large legacy Fortran Framework for doing Type IA Supernovae Simulations.

I was watching some of the young graduate students I was working with work on an assignment for a graduate Operating Systems class.

I'd taken that same class there during the '70s, and I was struck by how little

had changed. Sure, they used Java now, instead of Pascal, but the assignment itself was familiar, a virtual memory page table simulation. And I chucked to myself, recalling my own experience with the “weed-out” programming assignments, where one was docked a letter-grade for each day one’s work was turned in late. They all seemed rather blasé about this chore, so I counseled them, affecting my best avuncular demeanor, that I knew just what was going to happen. They were going to underestimate how much time it would actually take them to program this conceptually simple simulation, and as the deadline approached, they’d be pulling all-nighters to get their code to work and turn it in on time.

I chuckled to myself as I said this, recalling my own all-night coding benders at the old Computer Science building across the street, and staggering home after no sleep to find that a roommate’s cat had use the bed I was longing to collapse into as a litter box… But I digress…

I was sure these guys were about to go through the same kind of ordeal…

…but…

A couple of things had changed. For one thing, there were no all-nighters on campus, everyone worked on their own rigs at home, but the Big Difference was this:

These days, these machine problems are graded using a set of test cases, which are provided along with the assignment, that, should they all pass, indicate that your solution was correct.

…and, as I watch these students casually code their own solutions, I’d swear that these tests took a good day off the time it used to take to finish this kind of task.

Nothing I have seen over the last thirty-five years, other than perhaps objects themselves, has had the impact on the development process that micro/unit testing has had.

==========

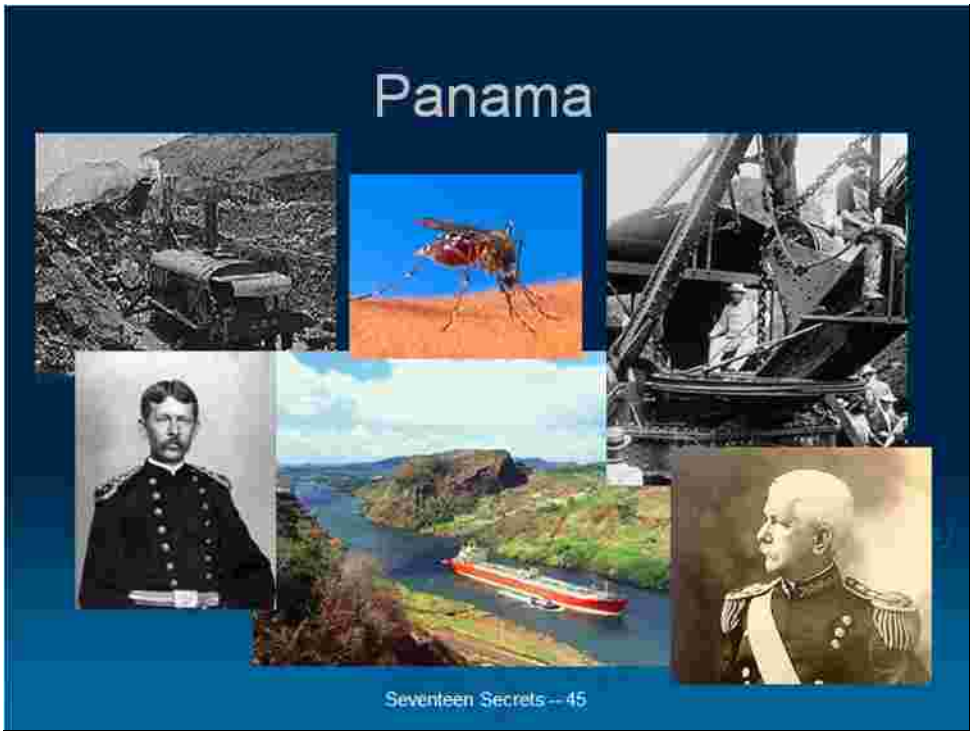
A vintage vacuum tube tester, vacuum tubes, Kent Beck [Yeah, again… I

know], “King of Legacy” Michael Feathers”, Nat Pryce, a commercial testing product, Steve Freeman…

==========

Beck Image: http://astah.net/friends-of-astah/friends

Feathers Image: http://www.flickr.com/photos/fraserspeirs/3394782283/sizes/o/in/photostream/



[A Man, a Plan, a Canal: Panama That's Theodore Roosevelt manning the steam shovel in the upper-right hand picture]

The United States succeeded in Panama, where the French had failed, for a number of reasons. The lock-and-dam design, the availability of more modern tools…

…and…

an unprecedented focus on dare we call it, the creation of a **habitable workplace**.

Medical men like Walter Reed, William Gorgas, Cuban Dr. Carlos Finley knew that the yellow fever-causing mosquito *Aedes aegypti* needed to be dealt with if disease was not to ravage this workforce as it had during the French effort.

Their weapons: a focus on Sanitation. They dispensed, it is said, a ton of quinine a year.

Before they sent men to work in those fester jungles, they set out to literally Drain the Swamp. Pools of stagnant water were eliminated or treated with oil.

Gorgas divided the mire into districts. Buildings and houses were searched, and pools of stagnant water were eliminated or treated with oil.

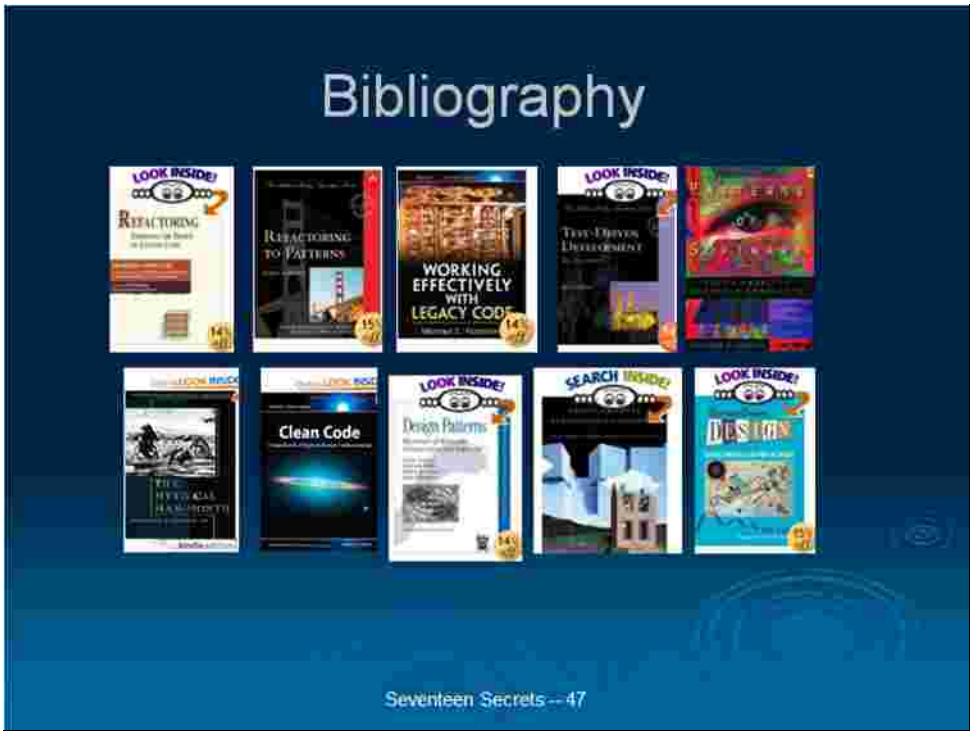
The analogy is too tempting to ignore*: By getting rid of the bugs, by taming the jungle, the way was cleared for the engineers and workers to succeed.*

And they did.

[This is from the Wikipedia article:
http://en.wikipedia.org/wiki/Health_measures_during_the_construction_of_the_Panama_Canal]

The Bullets.

[In hindsight, there should have been more material about Tools and Metric, among other things…]



The Books mentioned in this talk…

Thanks you to Amr Elssamadisy, Joshua Kerievesky, and YOU…

http://www.elssamadisy.com/Recongize_And_Respond/Welcome.html