THE SOFTWARE PRACTITIONER

JANUARY – FEBRUARY 2012 The newsletter by and for software professionals. VOLUME 22, NO. 1

No Software Crisis
"What's the Matter With You Software People?"

The headline said "All Late Projects are the Same." But the message in the article [DeMarco 2011] was much more profound than that – it wasn't explicitly stated but it was really "There's no such thing as a software crisis – there is only a software estimation crisis."

Of course, there's a reason that's the conclusion I came to after reading the article referenced here. I've been saying that for years, and it was nice to read that someone else – someone who has earned a great deal of respect over the years - was saying the same thing.

In case you're having doubts, here's what [DeMarco 2011] actually said – "In pouring over nearly a billion dollars worth of software litigation, I came across no failures due to poor quality, slow response time, or unworkable human interface. All the failures were about lateness."

The article began by exploring the question "What's the matter with you software people?" so often asked by non-software people in an organization. And, according to the cited article, "the answer was surprisingly simple – "we're occasionally late."

Now I can hear the uproar starting already. From gurus, who make their money by offering up solutions to the "software crisis." From academics, who think they know better than us practitioners about what we're doing wrong. By everyday software folk, who have been brainwashed by the cries of crisis over the years. "Splutter splutter," the criers cry out, "there is TOO a software crisis, and I know because I write about it all the time."

The article concludes by noting that "What's really wrong with us software folk is that we're continually beating ourselves up for something that's somebody else's fault." And whose fault is it? According to the cited article, it's "all projects that finish late have this one thing in common – they started late." So the problem, this author says, involves

- those who didn't have the guts to kick off the project on time
- those who thought starting on time would shed light on the fact that the budget was too small
- those who weren't really convinced the project was needed until its window of opportunity started closing.

And who were those people? Business decision-makers. Not software people. For once, we in software get to say "We have met the enemy, and he is NOT us!"

Reference –

DeMarco 2011 - "All Late Projects are the Same," from the "Sounding Board" column, IEEE Software, Nov. 2011; Tom DeMarco

The Problem is The Problem

Extracted from: Michael Jackson's book *Software Requirements & Specifications*; Addison-Wesley, 1995; used with permission

Focusing on Problems

An inability to discuss problems explicitly has been one of the most glaring deficiencies of software practice and theory. Again and again writers on development methods claim to offer an analysis of a problem when in fact they offer only an outline of a solution, leaving the problem unexplored and unexplained. Their readers must work out their own answers to the question: If this is the solution, what was the problem? Ralph Johnson, a leading advocate of Design Patterns in object-oriented development, has this to say:

"We have a tendency to focus on the solution, in large part because it is easier to notice a pattern in the systems that we build than it is to see the pattern in the problems we are solving that lead to the patterns in our solutions to them."

That's unfortunately true. And this tendency to focus on the solution has been harmful both to individual development projects and to the evolution of methods. Many projects have failed because their requirements were inadequately explored and described. The requirements are located in the application domain, where the problem is; but in most developments all the serious documentation and discussion focuses on the machine which is offered as a solution to the problem. At best there will be a careful description of the coastline where the machine meets the application domain. But the hinterland of the application domain is too often left unexplored and unmapped.

Methods and Problem Frames

Failure to focus on problems has harmed many projects. But it has caused even more harm to the evolution of development METHOD. Because we don't talk about problems we don't analyse them or classify them. So we slip into the childish belief that there can be universal development methods, suitable for solving all development problems. We expect methods to be panaceas — medicines that cure all diseases. This can not be. It's a good rule of thumb that the value of a method is inversely proportional to its generality. A method for solving all problems can give you very little help with any particular problem.

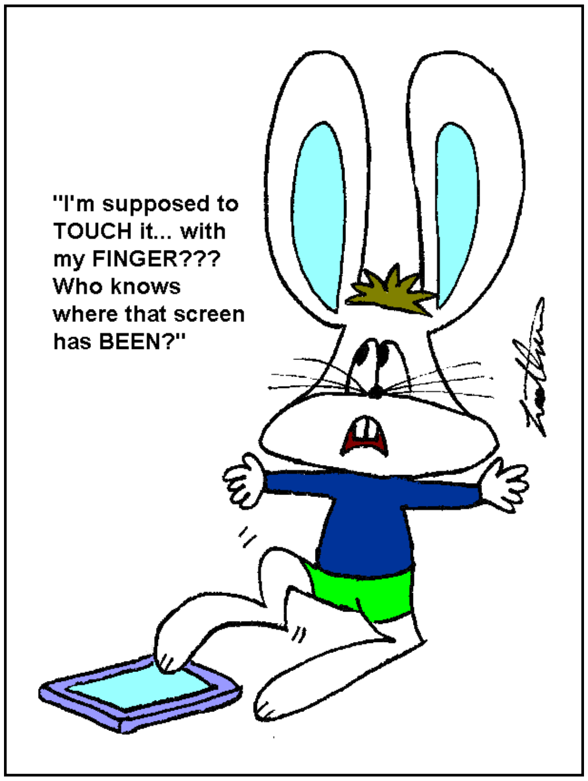
But have you ever read a book on a development method that said: "This method is only good for problems of class X"? Probably not. It's not surprising, because without an established discipline of analysing and classifying problems there can't be a usable vocabulary of problem classes X, and Y, and Z.

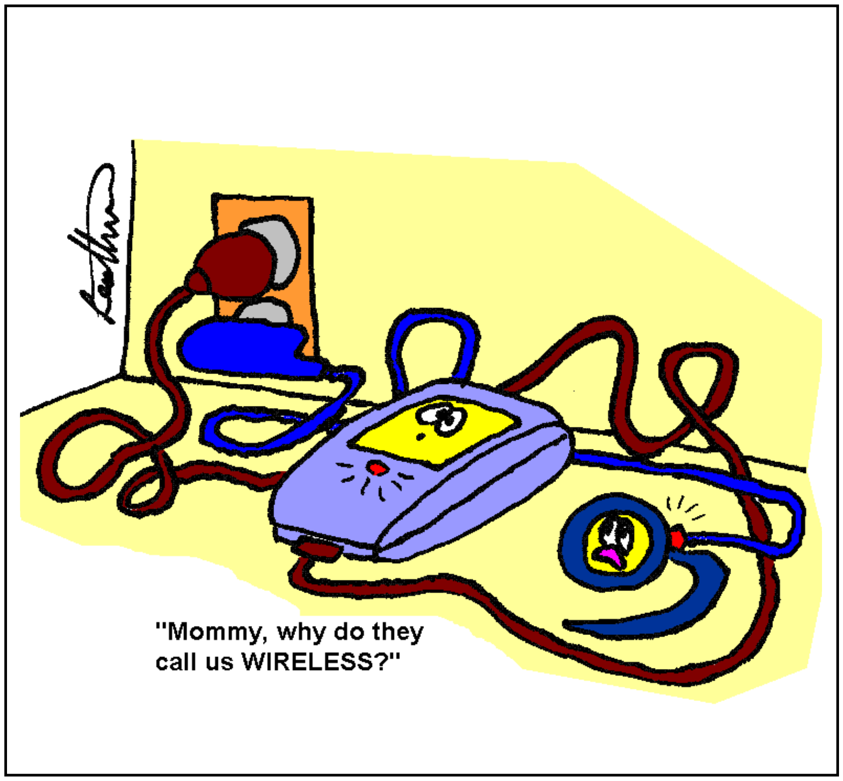
Classifying problems and relating them to methods is a central theme of this book. The crucial idea here is the idea of a problem frame, derived from the work of Polya. A problem frame defines a problem class, by providing a ready-made structure of principal parts into which all problems of the class must fit. Whether a particular problem fits a particular frame depends on the structure and characteristics of the application domain, and the structure and characteristics of the requirement.

A good method addresses only those problems that fit into a particular problem frame. It exploits the properties of the frame and its principal parts to give systematic and sharply focused help in reaching a solution. This means that the power of a method depends on the quality and precision of the frame. There's a principle here:

continued on page 5

Also in This Issue


"I'm supposed to TOUCH it... with my FINGER??? Who knows where that screen has BEEN?"



THE SOFTWARE PRACTITIONER

ISSN = 1083 - 6861
Distribution by Ebsco Publishing has been authorized

PUBLISHER:
Computing Trends,
18 View Street,
Paddington QLD 4064, Australia
61-7-33-11-12-13
email: rlglass@acm.org

EDITOR:
Robert L. Glass

ASSOCIATE EDITOR:
David N. Glass
Bill Medland

ART EDITOR:
P. Edward Presson

EDITORIAL ADVISORY BOARD:

David D. Lang
Consultant (simulation)

Larry Welke
President,
Children Without Limits
(data processing)

Steven C. McConnell
Construx Software Builders
(micros)

Donald J. Reifer
President, Reifer Consultants, Inc.
(management/large projects)

Unless otherwise stated, articles in The Software Practitioner are written by Robert L. Glass, cartoons are created by John Leatherman.

CALL BOARD

The Software Practitioner, a newsletter by and for software professionals, needs your help. This call board is our way of telling you what help we need:

Call for Papers: We especially like lessons learned, approaches tried, experiments conducted, surveys analyzed, unusual applications, controversy, humor. If it's something you'd like to read, we'd probably like to publish it. We pay for accepted articles in either subscription time (two years per published article) or advertising space (1/2 page per article), your choice.

Call for Subscribers: We need you. We hope you need us! Subscribe now, and make sure you get every issue of the Software Practitioner. The cost is REALLY low - $39/year, $29 for renewals, and $99 for institutions.

Call for Advertisers: Our readers are the people who make recommendations to the decision makers. People who want reality, not hype. If you'd like to reach that audience, we'd love to talk to you. Our rates? $99/page, $54/half-page, $29 quarter-page.

Call for Reviewers: This is a reviewed publication. If you'd like to review for us, tell us the topic area for which you're qualified.

RESPONSE FORM

Please…

☐ Enter my subscription
 ☐ Individual, $39/yr.
 ☐ Institutional, $99/yr.

☐ Send me information for advertisers

☐ Consider the enclosed article for publication in SP

☐ Consider me as a reviewer on (topics)

Name ____________________

Address ____________________

____

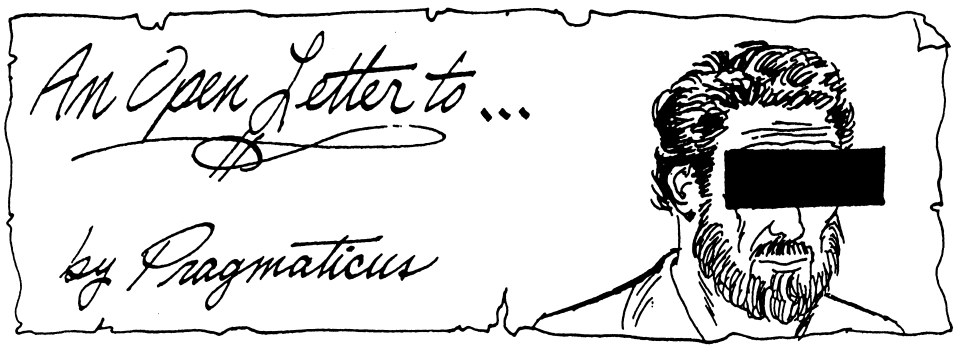
____

E-mail ____________________

Shareware Subscriptions!!!

If you are an SP subscriber, get a friend to subscribe and receive a $5 rebate. Just have them put your name below, and send in their check with this form.

____



An Open Letter to Poul-Henning Kamp, who wrote “The Software Industry IS the Problem”

Poul, you must live in some other universe that I have never come in contact with. Or, at least, that’s the reaction I get when I read your Practice section column in the Nov., 2011 issue of Communications of the ACM.

Now, I do understand that your column is really an appeal for software liability laws, wherein software producers would need to take responsibility for the flaws contained in their products. I get that, and at some level I might even support it.

But it’s the way you go about reaching and expressing that appeal that deeply troubles me. Here are the things you say that I find other-worldish:

1. You make much of an old Ken Thompson quote, “You can’t trust code that you did not write yourself.” I disagree with that one on so many levels that I’m not sure where to start. For one thing, I trust other people’s code all the time. I trust my banking system software. I trust the word processor on which I’m writing this column. I trust Google to do an astonishing job of gathering facts that I had no way of gathering whatsoever just a couple of decades ago. I trust all the other pieces of software that make our world, circa 2012, workable and comfortable. I’m sorry, Ken (and Poul), that you don’t trust software written by others, because if you really mean that you’re missing out on much of the richness of life in our era.

But let me go further here. In fact, I am no more likely to trust code that I have written than code written by someone else. As a human being, I am capable of writing bad software. I have been known to produce software that, on occasion, fails. That troubles me a bit, but the creation of successful software is one of the most complex tasks humanity has ever undertaken, and I suspect that I will never overcome my failings in this regard. The people I might least trust, in fact, are those arrogant enough to think their code will never fail. Ken and Poul, do you really believe that your code is better than all of the code written by others?

2. You say that this Ken Thompson quote is “a fundamental law of nature” which “follows directly from pure mathematics.” Poul, I see none of this as any kind of fundamental law, and I see no relationship whatsoever between this blabbering and the field of pure mathematics. If you understand what I’m saying here – and how can you not? – can you begin to see why I think you’re from another universe in this article?

3. You refer to universities as “the only places where computers were exposed to hostile users who were allowed to compile their own programs.” Huh? I have worked in lots of industrial settings, and in each of those places we programmers compiled our own code all the time. I suppose one could argue about whether we were hostile or not, but then I have to admit I’m not at all sure what point you are trying to make here, and I therefore fail to see the relevance of the word “hostile” in this context.

4. Near your conclusion, you say “the only people with source-code access are the software houses and the criminals.” Once again, huh? Double-huh?? Do you truly believe that only software houses write software? Once again, I spent my entire career working in industrial settings (with an occasional university setting thrown in) where my time was always spent working with source code. I am beginning to wonder, at this point in your column, if you really understand the software industrial world at all.

Because of that, I looked at your bio, which CACM always includes at the end of an article. And I see that you have “programmed computers for 26 years” and have worked in both “open source and commercial products.” One of three things occurs to me here, Poul. Either (a) your career has been spent in sheltered and strange environments, (b) you are exaggerating for effect here, or (c) you really understand much less than you ought to about the software field.

Now, about that software liability law proposal of yours. You say it would have three clauses:

0. If damage is caused intentionally, it should be handled under criminal laws. This proposal is only about unintentional harm.

1. If your software is delivered with “complete and buildable source code and a license that allows disabling any functionality or code,” then your liability is limited to a refund. As best as I can figure in this murky proposal of yours, this clause is meant to protect those who create or modify open source software. Why would you want to protect them? Shouldn’t they be as liable as the next programmer for bad code they create?

2. In any other case, you are liable for whatever damage your software causes when used normally. OK, I suppose, but this could be considered Draconian. If I write software for an Airbus 380 that crashes because of a bug I have inadvertently included, do I have to pay for the whole airplane?

Still, I see your proposal clauses as an interesting jumping-off place for a discussion of the topic of software liability laws. I do have a feeling that your understanding of the field is so peculiar that I doubt if you know enough to make such a proposal sensibly, but still...

And then you conclude with this: You quote objectors as saying this law will mean the end of computing as we know it.” And then you present your response – “Yes, please! That was exactly the idea.” And at that point, my own personal reaction was “Why did I bother to read this drivel?”

To the Editor:

On the topic of whingeing, in particular Pragmaticus’ column in the November issue on the subject, here’s some additional information that you might find interesting.

It has been the conventional ‘wisdom’ of Australians for generations, at least since I was a boy, that it is the English who are the whingers. The derisive term ‘whinging pom’ was often bandied about to describe arrivals from the old country. An interesting thing seemed to be going on here. Many English immigrants seemed to regard Australia as culturally inferior, our society being based on convict culture. So when they arrived, they would promote themselves to the next higher social class. Working class became middle class, and middle class became insufferably bourgeois. But they were insecure in their new social status, and often floundered, making social gaffes that they hoped we would be too ignorant to notice. But as we know, insecurity often makes people more nervously vocal, so it was common to hear them voicing what they thought was wrong with Australia, hence the term whinging pom. Australians liked to think of themselves as stoic non-complainers. Oh, how things have changed in recent times.

Below is a link to a little Kindle ebook I wrote on Australian slang. I think you’ll find it both interesting and amusing.

http://www.amazon.com/Australian-Slang-Vernacular-English-ebook/dp/B00579XJ9I

– David Tuffley

From the Editor:

Re the British and convicts:

From both a (Brit) friend of mine and from a guide at Port Arthur national park in Tasmania, I hear that the British see the Americans as also having a convict heritage. I think that allows them to see the Americans as inferior, also. (And perhaps it also explains how those ruffians were able to beat them in a war of independence!).

Now American history says no such thing! We speak of indentured servants and people seeking religious freedom from a variety of European countries, but not British convicts.

So it is interesting to see what unbiased history says. It turns out there were British convicts brought to America, but in nowhere near the numbers that settled Australia, and if you look at the numbers those convicts were a very small percentage of the American settlers. But based on what I've seen, there's no way of ever overturning the Brit belief that America, like Australia, is basically of convict origin.

Touching the Void

Linda Rising

linda@lindarising.org www.lindarising.org

A recent PBS special told the story of an adventure that took place in 1985. British mountaineers Joe Simpson and Simon Yates set out to climb the imposing snow-covered, 21,000-foot Siula Grande in the Peruvian Andes, one of the most formidable peaks in the world. Their goal was to conquer the notorious west face of the mountain, a nearly vertical ascent that had been attempted without success by other professional climbers. Simpson and Yates were confident they would succeed. They were young (in their twenties), in tip-top shape, and experienced.

Their story has been captured in Simpson's book *Touching the Void*, and in the PBS special (and DVD) by the same name. Obviously, there were no cameras on the original climb, so actors recreated the events while the real Simpson and Yates narrate. The result is surprisingly dramatic. It is obvious that the climbers survive, but with the tension of the ordeal you somehow forget that the climbers are telling the story.

The climbing approach they used is called Alpine style, which for us non-climbers, means they didn't carry a lot of stuff and there isn't a series of base camps with emergency supplies. The strategy is to move fast and not make any mistakes. You can see the connection with software, can't you?

The first part of the climb, the ascent, was successful (read "they were lucky"). And then, on the descent, luck ran out. Simpson fell and broke several bones in his right leg. His lower leg pushed through his knee joint. It's horrible to imagine such an injury in any situation but at that altitude and in those remote conditions, it was effectively a death sentence.

With essentially no food or water, the climbers had to get down—fast. They had made an agreement that if one of them became injured, the other would leave him behind to save himself, but Yates was determined to get his friend home, even though a storm was moving in. Each climber carried 150 feet of rope, so Yates tied these pieces together with the idea of lowering Simpson down the mountain in increments of 300 feet. The only complication was that Yates had to stop after the first 150 feet of each repetition and signal for Simpson to give him enough slack so that he could get the knot past his harness.

As you can imagine, rolling down the mountain was agony for Simpson. In the film he cries as he bounces over the rough terrain, but Yates has to ignore his partner's suffering. Both their lives were at stake.

Things were progressing unexpectedly well when after one slide down the mountain, Simpson did not respond. Yates was stuck. He had no idea why Simpson was not giving him the slack he needed, so Yates positioned himself against the mountain and waited in a blinding storm. It took all his strength to hold onto the rope, and he became increasingly aware that eventually his muscles would fail and both men would plummet down the incline.

What Yates couldn't know was that Simpson had fallen over the edge of a crevasse and was hanging in midair. Simpson's fingers were frostbitten and he was unable to climb back up the rope and unable to communicate with Yates.

They remained like this for an hour, with Yates' strength slowly ebbing away and Simpson's weight on the rope slowly pulling him toward the edge of the cliff. Yates realized he was facing an impossible dilemma: if he continued to hang on to the rope until his strength gave out, Simpson's weight would pull them both down the mountain. On the other hand, if he let his friend go it would surely mean Simpson's death, but possibly saving his own life. It also occurred to Yates at this point that Simpson might, in fact, already have died. That would explain the lack of response and the dead weight on the end of the rope.

Logic would suggest that it's better for only one man to die rather than both—especially if the one is already past hope. But for a climber, the biggest taboo is to cut the rope that binds him to his partner. In fact, this tactic is anathema.

Yates is completely candid when discussing the ethical dilemma of this decision for which he was nearly hounded out of mountaineering and the reason Simpson wrote the book about the event—in defense of Yates' actions. In the film, 17 years after the fact, Yates calmly states that when he realized that he had a Swiss Army knife in his pack (and that he knew that sooner or later he would be dragged off the cliff by the weight of his partner, whom he had no way of helping) he decided 'fairly quickly' to cut the rope.

This is the moment I call the first decision point. We'll return to this later in the article.

After Yates cut himself free, he must still make his way down the mountain burdened with the immense guilt over what he believes is the death of his friend.

When Yates cut the rope, he had no way of knowing that not only was Simpson still alive, but that he survived the fall when Yates cut the rope and landed inside the crevasse. Simpson's frostbitten fingers and the trauma of his injury have sapped his strength. He was unable to climb the rope to get out of the crevasse. He could only hang there, suspended, waiting to die.

As it got dark and he got tired of calling for help that he really didn't expect to arrive, he reached his own decision point and also did the unthinkable. He cut himself loose. His decision was, as Yates', incredibly difficult but very logical. Hanging there he was sure to die, and not a pleasant death. If he cut the rope, he would fall further into the crevasse. One of two things would happen: either he would further injure himself and die sooner or, perhaps, he would discover a way out at the bottom.

This is the moment I call the second decision point.

Since you know that both climbers tell their stories in the documentary, you know that Simpson does find a way out at the bottom of the crevasse. Even with his injury and no food or water, he manages to find his way back down the mountain. Luckily, Yates was still in the area, taking time to recover from his own ordeal and not wanting to give up completely on his climbing partner.

Touching the Void is a gripping and horrifying film. In 2004 it won *Outstanding British Film of the Year* at the BAFTAs and *Best Film of the Year* from the Evening Standard British Film Awards. It featured prominently in both the Telluride and Toronto Film Festivals and has surpassed *Bowling for Columbine* as the highest-grossing documentary in the UK. I hope you'll find it at your nearest video rental store or on-line. Be aware that I haven't given you enough information to spoil your enjoyment of this production.

Now, let's see what we can learn from this, even though we may not be professional climbers! Since I'm a firm believer in retrospectives—at an organizational and a personal level, I examined this experience, especially the two critical decision points, and asked what could be learned from this mesmerizing drama.

I believe that most viewers will be impressed with the determination and the incredible will to survive demonstrated by the two young men, especially Simpson. When we hear stories like this—how people conquer incredible odds—we are hopeful and uplifted by the display of courage. But let's focus on the two major decision points. (1) When Yates cut the rope and later (2) when Simpson cut the rope.

At decision point one, when Yates cut the rope that connected him to his injured partner, it was momentous for several reasons. This ac-

"Touching the Void is a gripping and horrifying film. In 2004, it won Outstanding British Film of the Year at the BAFTAs and Best Film of the Year from the Evening Standard British Film awards."

tion was unthinkable for a climber. Yes, you could argue that logically it seemed the right thing to do. Yates could only hold the rope so long before he would be pulled over the cliff. By cutting the rope, he reasoned that he was increasing his own chances of survival, while probably sending Simpson to his death. Simpson, recall, was injured, and was only making progress down the mountain because Yates was lowering him on the rope. Recall, too, that Yates wasn't sure that Simpson was still alive. Lack of response could have indicated that Simpson has suffered a fatal injury in the rapid descent. Cutting the rope meant increasing the chances that at least one would survive. Better one climber survives than no climbers survive. Seems simple.

However, the intriguing thing is, that not only was this the logical thing to do, it was the **only** thing that would, as it turned out, enable both men to survive. It was the **only** way out. It was not on the list of accepted best practices for climbers. In fact, some still say today that Yates violated the rules that all climbers follow. Violating the rules in this case enabled the survival of both men.

Luckily Yates was able to violate the climber's taboo and as he himself describes it "fairly quickly" (after about an hour) cut the rope.

Now the second decision point. I don't know about you, but as I was agonizing with Simpson as he considered the hopelessness of his situation, I was urging him on, "Cut the rope! Cut the rope!" Easy for me, sitting in my living room. But imagine hanging inside a crevasse with a broken leg, calling out for help that doesn't arrive, that couldn't possibly arrive. Simpson cried. He cursed. He called himself a host of names and railed against his fate. It's interesting that Simpson, raised as a Catholic, never prayed to God for deliverance. Even in this most dire of circumstances—perhaps that's a true measure of how he really was, literally, at the end of his rope. But cutting the rope would not mean the release of a burden, as it did for Yates, not necessarily an increase in the odds for survival. He was going to surely plummet further into the crevasse. He was surely going to suffer more and probably increase the extent of his already severely broken right leg. He was only able to do this because he was out of options. This still isn't an easy situation.

I recall the story of Aron Ralston, a 27-year-old climber who escaped danger by cutting off his arm. After being pinned by a nearly 1,000-pound boulder for five days in a remote slot canyon in eastern Utah, he cut off his pinned arm with a pocketknife, rappelled down a rock wall and hiked until a search helicopter found him. Ralston decided to sacrifice his arm to save his life only after his water ran out and he realized that he would not survive unless he took drastic action. He used his pocketknife to amputate his right arm below the elbow and then applied a tourniquet and administered first aid from a kit he had in his backpack. He then rigged anchors and fixed a rope to rappel nearly 75 feet to the bottom of the canyon and hiked until a search helicopter rescued him.

Yes, it's clear how necessary these actions were, but what courage it took and what a difficult decision it was to make—even though it was the only way out of a life-threatening situation.

It brings us up short to even draw a remote comparison with the decisions we face. Yes, we work in an incredibly dynamic, fast-paced industry. Just when you think you've got it all figured out: how to organize your team or connect with your customers, something unexpected and uncomfortable leaps out at you.

The advice seems to be the same. Let's hear from Richard Pascale in his book Surfing *the Edge of Chaos: The Laws of Nature and the New Laws of Business* (Crown Business, 2000), where he and coauthors Mark Millemann and Linda Gioja describe a system of thinking and managing to help companies adapt in this perilous environment.

Let's be clear. The idea of 'living systems' isn't a metaphor for how human institutions operate. It's the way they really are.

Rapid rates of change, an explosion of new insights from the life sciences, and the insufficiency of the old-machine model to explain how business today really works have created a critical mass for a revolution in management thinking.

Pascale's approach is the result of studying current organizations and developing a set of laws that say that the first rule of life is also the first rule of business: Adapt or die!

The first new law of life that leaders need to recognize is that equilibrium equals death. We may enjoy a period of time when following the established rules really works. It may give us a dominant position. It may result in economic success. But it also makes us increasingly vulnerable when the world changes. When things change those rules we followed to success in the past become our own worst enemy.

The question we have to ask ourselves is, "Does the formula I have been following still work? Am I becoming a victim of my own success? Has the world changed so that my winning strategy is no longer the way to survive?" Just as in climbing, survival favors heightened adrenaline levels, wariness, and experimentation—a willingness to violate the rules.

How do we pull back from the edge? First, members of the organization must become innovators, champions of new ideas. This is a perfect spot to give a shameless plug for my book: *Fearless Change: patterns for introducing new ideas*! People should be encouraged, not just allowed, but encouraged to look for the next big idea. In other words, to cut the rope!

Leaders can encourage this behavior by saying that if an error occurs, it's everybody's fault. If everyone is encouraged to come up with the next big idea, if something goes wrong, then no one starts finger pointing and blaming. People in an organization must see themselves as entrepreneurs who are responsible for identifying and satisfying customer requests. They must feel free to "cut the rope," if they see a way out of a dilemma.

One way to see how this works is to consider a bad example from Pascale's book. In January 1999, when a blizzard closed Detroit Metropolitan Airport and canceled outbound flights, snowplows kept the runways open, and a number of inbound planes were able to land throughout the evening. Most carriers were able to bring their planes to the gates to off-load their passengers but not Northwest Airlines.

Northwest's ground staff seemed paralyzed by indecision, held hostage by rigid policies and practices. Nearly 4,000 passengers were virtually imprisoned on 30 Northwest flights for as long as eight hours without food, water, or working toilets. Fights broke out. Passengers threatened to open emergency-exit doors. Pilots yelled at ground staff over the radio to tow the planes to the gates before they lost control of the situation.

Northwest's inflexibility in adhering to rules and procedures for passenger safety caused them to overlook many possible solutions to the problem. In other words, no one felt free to "cut the ropes." They could have towed the planes close to the gates and let passengers off on the tarmac. They could have let them off on the runways and bused them to the terminal. They could have brought service vehicles out to the planes. It could have been a different scenario if leaders had told the staff, "We have a huge disruption on our hands. Be innovative and imaginative, and demonstrate to each other and to our customers that we can come through when it counts." Instead, Northwest lacked the capacity to violate the rules, cut the rope, and survive.

We can learn a lesson from the climbers and hikers who survived: always carry that Swiss Army knife in your pack and be prepared to break the rules when faced with a potentially "life-threatening" dilemma. Your inventiveness might help you survive!

The Problem is

continued from page 1

The Principle of Close-Fitting Frames

The purpose of a problem frame is to give you a good grip on the problem. So the frame must define and constrain the problem very closely indeed. You can't work effectively on a problem that's wobbling around in a sloppy, loose-fitting frame.

Problem Complexity

The largest description structures in software development are the structures that arise from problem complexity. A complex problem is a problem that can not be completely accommodated by any available frame and completely solved by any available method. You need more than one view of the same problem. That's not just taking more than one view of the machine — for instance, data, function, and state transition, or class hierarchy, object aggregation, and object interaction. It's more like recognizing that the same problem is simultaneously a control system problem, an information system problem, and a message switching problem. It's a multi-frame problem.

The task then is to separate out the different problem frames in their parallel structure. Identify the principal parts of each frame; and identify the overlapping parts and aspects of the application domain and requirement that they cover. This is the essence of problem decomposition. When you separate a problem by problem frames, you're separating it into simpler sub-problems that you know how to solve. Then you have a real chance of mastering the complexity that is found in every realistic software development.

Julian Assange – the Unauthorized Autobiography

By Julian Assange and
an unidentified ghost writer

Published by Canongate, 2011
Review by Robert L. Glass

This is a review of the book named above. But it is also a review of the story behind the book, which is in many ways at least as interesting as the book!

Julian Assange, of course, is the head of WikiLeaks, an organization that can be described variously as a whistleblowing company or a leaker of information, and can be seen by the reader as either a heroic company doing heroic deeds or as a subversive company tattling other people's secrets.

The reaction you get from reading this book will in all likelihood be an elaboration of whatever bias you had when you began reading it. The book was published quite recently, to the extent that this reviewer has seen only one previous review of the book. That reviewer produced a very factual review, but then, in a final paragraph, described Assange as something of a hero, casting doubts on the objectivity of the review itself! To announce my own position up front, I question the work that Assange has done, and you will have to read this review taking that bias into account.

Now, for the story behind the book. When Assange was arrested in England because of rape charges brought against him, he went to court and fought extradition to Sweden where he would be tried for those crimes. During the time that his extradition hearing was being scheduled, he first spent time in a prison, then in house arrest in a mansion owned by one of his supporters, and he vowed to work on this autobiography during that time.

Apparently he and the ghost writer he had chosen made great progress toward this autobiography, as the world waited for the book with baited breath. But, as the time for the extradition hearing approached (as of this writing, Assange was indeed extradited to Sweden to stand trial), Assange became less and less enamored with the book that was being written, and in the end asked for the contract to produce the book to be terminated. But there was a problem – Assange had already spent the advance the publisher had given him, and since he was unable to return it, the publisher pressed ahead with publication. (The publisher notes all of this in a preliminary section of the book). (Oddly, Assange still owns the copyright for the book).

All of this background story, of course, whets the reader's appetite as they proceed with reading the book. What did Assange find to dislike about his book so much that he tried to prevent it being published? We will deal with that issue in the context of the review of the book that follows. But in the only explanation Assange has offered, he said he found the book "too personal." Given that Assange is best known for advocating openness and transparency in all things, this is a curious position for him to take. But, as it turns out, the book does contain a discussion of where Assange places his emphasis on openness, and that explanation does allow for understanding that the book was "too personal."

The book is, as you would expect (since it is an autobiography) written in the first person, and the nature of its content leaves little doubt that it is transcribed directly from tapes of Assange's dictation. It opens with a lengthy discussion of what it is like to be imprisoned (the early phase of his being held over for the extradition hearing), in which Assange likens himself to previous "heroic" prisoners such as Oscar Wilde and Daniel Ellsberg, and in which he apparently tries to create sympathy for himself as having to endure prison conditions. (Interestingly, he spends little comparable book space on his later mansion house arrest!) (Of course, this analysis reflects my anti-Assange bias). During this early chapter, he also characterizes his company, WikiLeaks, as being "engaged in communications warfare," a curious description since in most of the rest of the book he lauds WikiLeaks for its peaceful intent.

Following the "prison" chapter, he retreats, autobiographically, to his youth. It was an unusual youth. His parents were roaming hippies, and he attended 30 different Australian schools during those early days. Recalling those days, he says "non-conformity is the only passion worth being ruled by," says his little family was "very much like fugitives" (one former stepfather hunted them down with apparent malicious intent), and said "I was the kind of child who was shopping for things to take a stand against." In other contexts, he characterized himself as "I was just bred to hate the system," "as arrogant as we were insubordinate," a "slacker teenager," a "misfit," and an "anarchist."

Another thing that stands out in reading these early portions of the book is that Assange rarely seemed to accept responsibility for his actions. He describes one episode where he "hit a girl over the head with a hammer," and offers up no apology. When he is jailed and then convicted for 31 hacking offenses later (his sentence, in the end, did not require jail time) he once again offers no apology and seems to think that he and by implication all hackers are not bad guys, just curious youth fighting against an evil establishment. (He seems not to realize that some hackers have indeed been evil). Elsewhere, he said that he never "did anything illegal," yet he describes with apparent pride a "real estate agency" he ran to facilitate squatters in vacant properties, and talks about the time when a traffic light outside his apartment annoyed him so he hacked into its controller and froze it in a "green light" position.

(Could it be these sections that Assange saw as "too personal"?

Moving forward in time, Assange describes his discovery of the computer as a "positive space in a negative world," and went on to characterize himself as "a person who came along in time to do a piece of work ... [that} made a difference in the world." It was during this time that he began to see himself as a "fighter for freedom" and as a "journalist."

In the book, Assange does a lot of name-calling about people whose actions he disagrees with or doesn't like. The prosecutor in his hacking case was a "lunatic," the journalists who frustrated him in his later WikiLeaks days were "moral pygmies," "weakest and most self-protective men," "hid behind the coat-tails of the school's bad boys," "cowards," and – finally "lily-livered gits."

(Could this be the material that Assange saw as "too personal"?)

There are some oddities about the book. Early on, Assange obliquely refers to the rape charges against him without any previous discussion of what has to be seen as a major topic in his life. (It is almost as if he wants to play down their importance). Later, he says that, in the 1980s when he first became enamored of the computer, "they came with no software" (that was simply not true). And even later, he discusses a time when "in the early 1990s, the US government tried to argue that a floppy disk containing code must be considered a munition" (this is untrue and furthermore paranoid!)

There are two interesting explanations that Assange offers in the book to issues of major importance:

- Regarding transparency and openness, "it was never my position that privacy was bad: rather the opposite. What I oppose ... is the use of secrecy by institutions to protect them selves against the truth of the evil they have done." (In this way, he dispenses with the charge that he himself uses secrecy to cloak his own actions, while decrying it in others).
- He spends a great deal of space talking about the circumstances that led to the two rape charges against him, the charges that are resulting in his being extradited to Sweden. Oddly, nowhere does he mention the issue of condoms (the women in question say the sex was consensual, but he refused to wear a condom when they asked him to, and proceeded to have sex with them anyway) (His version of events mentions their concern about sexually transmitted diseases, but nowhere mentions the condom issue).

And then, even more oddly, Assange attacks Sweden for its attitude on a number of things only somewhat related to his rape charges. And one other thing that makes no sense. It is fairly well-known in the popular literature that the WikiLeaks computers are housed in a former military underground establishment near Stockholm in Sweden. But nowhere in the book does Assange mention this fairly strong connection with Sweden.

But, returning to the book. Assange discusses his college years at the University of Melbourne (Assange was born in Australia and spent his early years there), where he enjoyed his studies of math and quantum mechanics. He also came to "hate religion," and in the book spends a considerable rant on the Scientology religion. He went on to found WikiLeaks in 2006, calling it "the first intelligence agency of the people." He emphasizes throughout the book that WikiLeaks is impartial, and mentions its coverage of leaks of information in Kenya, Malaysia, Somalia, Switzerland, and other places throughout the world to reinforce that point (some people see him as picking on the US).

He does a kind of "head 'em off at the pass" defence against Daniel Domscheit-Berg (who writes a scathing analysis of Assange and WikiLeaks in his own book reviewed elsewhere on these pages), calling him "a curious asset," "ambitious," and "reckless."

In the end, the book abruptly stops. It is apparent that the ghost writer never had an opportunity to finish the book, tie up its loose ends. The book concludes with an "Afterword" section containing a number of more recent factoids about Assange and WikiLeaks, loosely tied together. And then it provides, in 70-odd pages of appendix, examples of some of the leaked material that are what WikiLeaks is all about.

So, what did this anti-Assange reviewer conclude from all of that? That Assange's motives are, for the most part, honorable and in fact exemplary. That the material he releases is a mixture of irrelevancies and important facts. That he is quick to form judgemental opinions, in areas where most of us would continue to subscribe to "the wisdom of doubt.' (He takes positions, for example, related to political disagreements in Malaysia, Somalia, and Kenya, where most of us would still be trying to figure out which side was really telling the truth). That at best, he is an "investigative journalist" par excellence. And at worst he is an opinionated troublemaker who knows how to milk the world's attention to his own benefit.

In the end, I suppose, we will have another opportunity to know who the true Julian Assange is. The rape trial should expose enough facts about him that we will all know a great deal more (about him and his adversaries) than we do now. Stay tuned!

Inside WikiLeaks

By Danrel Domscheit-Berg
with Tina Klopp

Published by Crown Publishers, 2011
Review by Robert L. Glass

This is an important book on a lot of different levels. It gives an alternative insight into the work of Julian Assange, the founder of WikiLeaks, whose unauthorized autobiography is reviewed elsewhere on these pages. (The subtitle of the book says it is about "my time with Julian Assange at the world's most dangerous website."). It gives a lot of philosophical thought to what a WikiLeaks-like activity should consist of (in the end, Domscheit-Berg disagreed with Assange, was fired by him, and opened his own similar company OpenLeaks).

But there is another level of importance to this book – as a source of new information about Assange and WikiLeaks. For example, the book first broke the news that Assange "boasted about how many children he had fathered in many parts of the world," a fact which – if true – may have important bearing on the Swedish charges of rape against Assange based on his refusal to wear a condom during otherwise consensual sex.

But even more important than that is the claim, late in the book, that when Domscheit-Berg left WikiLeaks he took with him the chief WikiLeaks system architect and the sole knowledge about how the WikiLeaks software system worked, which – again if true – means that **WikiLeaks, at least at the time of writing this book, was basically non-functional!** That particular factlet has not been seen anywhere else in the popular or professional press, to the best of my knowledge, perhaps because the average journalist reporting on the matter did not understand the significance of what was being said there!

But let's get on with reviewing this book.

First of all, what is WikiLeaks? Domscheit-Berg calls it "a secretive organization whose motto is transparency." He goes on to note that, although the company creates a smokescreen to appear to be huge, it began as "two loud-mouthed young men working with an antiquated server" (computer) and a collection of fictional colleagues. (Even at the peak, Domscheit-Berg says, there were at most 20 technical employees in the company).

(All of that raises an interesting question. The popular press has said, repeatedly, that the primary WikiLeaks computer is housed in an underground bunker near Stockholm. Neither the Assange autobiography nor the Domscheit-Berg book support that statement. One wonders, after reading these books, if that underground bunker is a bit of urban mythology!) The Domscheit-Berg book also notes, as perhaps part of its characterization of WikiLeaks as "the world's most dangerous website," that only Assange and Domscheit-Berg subjected incoming leaks to authenticity analysis, a serious personnel weakness of the company as the number of such leaks increased dramatically.

How does Domscheit-Berg feel about Assange? At various places in the book, he calls him "extreme," "imaginative," "energetic," "brilliant," "paranoid," "power-hungry," "megalomaniac," " "zany," "dangerous," "uncompromising," "one of the greatest hackers in the world," "he had a free and easy relationship with the truth," and "rarely was anything his fault." You may have noticed a certain ambivalence in those descriptions. Perhaps more to the point, Domscheit-Berg also said "Julian Assange was my best friend," we "spent the best years of our lives together," and "I'd do it all over again."

There is a good chronology to the book, starting with an excellent timeline of WikiLeaks activities. According to Domscheit-Berg, their first leak had to do with Swiss bank accounts, which are well known worldwide for being havens of tax avoidance for the rich. They released some information leaked from an insider source that named names of those avoiding taxes. The Swiss bank in question filed suit against them and, according to Domscheit-Berg, because of clever defensive preparations they won the case and furthermore they have never been sued since!

Both this book and the autobiography spend quite a bit of space on a curious relationship WikiLeaks developed with the country of Iceland. The WikiLeaks principles spent considerable time trying to encourage Iceland to pass laws that would make it "a safe haven for freedom of the press," a quest which – as far as can be seen in both books – failed to bear fruit. And, in fact, it turned out worse than that. The two of them, with several teammates, spent their time in Iceland in such tight housing quarters that they got on each other's nerves, which apparently directly led to the personal breakup of the two of them. (The breakup was exacerbated when Assange, according to Domscheit-Berg, wanted WikiLeaks to cover his costs in the Swedish rape case, and Domscheit-Berg refused).

Some have wondered if Assange has concentrated on the US as a subject for his leaks. In the autobiography, Assange says "no." But Domscheit-Berg is a bit more ambivalent – he notes that the only language Assange knows is English, which eliminates many other countries from his scrutiny; he saw the US as the "biggest possible adversary" (which could gain the fledgling company the greatest visibility); and both Assange and Domscheit-Berg believed the Iraq and Afghan wars were wrong.

The book contains a fascinating miscellany of other facts. It notes, for example, that one government deliberately revised Wikipedia entries to make themselves look better; it also notes that Assange supported "constructing the truth by creating facts" (a damning indictment!).

How does Domscheit-Berg feel about the rape charges against Assange at this point? He is against anything that would result in extraditing Assange to the US. But at the same time, he believes that Assange should face the rape charges in Sweden at once.

And how does he feel about what OpenLeaks would become that WikiLeaks was not? He sees his new company as providing only a "technical infrastructure for whistleblowers,"and, for example, he thinks that the whistleblowers themselves should make the decision about how their information should be distributed (those decisions at WikiLeaks were generally made by Assange).

Ironically, rape cases in Sweden are not to be covered by the press, but the facts of the Assange case were leaked by one of the Swedish tabloids. A leaker brought down by a leak!

No matter how you feel about Julian Assange and WikiLeaks, the two books reviewed here are essential reading to anyone interested in drilling down to the facts underlying the story. This book is well-written, believable, and seems – to this reviewer, at least – to be free for the most part from the political agenda that is hard to avoid in the Assange book.

Software Engineering Meets ... (Lots of things!)

The October issue of Computer, the IEEE computer society magazine, is a special issue on "Software Engineering Meets..." where the three dots are replaced by several different topics.

Theory and Practice – Uncommon Bedfellows?

One of those topics is the relationship between theory and practice. In [Broy 2011] the author asks the question "Can Practitioners Neglect Theory and Theoreticians Neglect Practice?" and the answer it provides is an unsurprising "no." Since the article appears in a journal biased toward research and theory, some of its content is predictable:

- "Software engineering theories have made enormous progress in the past 40 years. In particular, those related to specification and verification, architecture and design, testing, software project management, and software economics. However, we are still far from comprehensive and established theoretical foundations."

But some of its content provides some excellent insights into why software theory is so seldom useful in practice:

- "Much more work on theory is needed to master the enormous future challenges related to software evolution..."
- "But most pure branches of theoretical informatics study different kinds of process algebras completely independent of any questions related to software engineering practice."
- "As long as theoreticians do not take into account issues from practice, their work will not be very helpful from a pragmatic viewpoint and will not find an immediate road into practice ... software engineering lives in a dirty and imperfect world, and many compromises must be accepted in practical software engineering."
- "Bringing proper theory into the unsystematic world of practice is a difficult endeavor."

Reference:

Broy 2011 – "Can Practitioners Neglect Theory and Theoreticians Neglect Practice?" IEEE Computer, Oct., 2011; Manfred Broy

Open Source – Can SE Learn from It? Can It Learn from SE?

The special issue has another "three dots" study on open source (OSS) vs. traditional software engineering, doing an excellent job of comparing and contrasting the two. Unbiased and objective articles on this subject are, unfortunately, all too rare.

The article does present some of the traditional (and questionable) assertions about the benefits of OSS:

- "OSS, beyond its obvious cost advantages, is of very high quality. Contributors to OSS projects are in the top 5% of developers worldwide..."

... but quickly counters those claims with things like:

- "it is virtually impossible to predict the usability, stability, and reliability of these products ... the uneven quality is not helped by a lack of documentation ..."

From then on, the article examines several important characteristic of software projects and concludes;

Quality – OSS "results are quite average in the software industry."

Community feedback – "there is very little [OSS] feedback on design issues"

And eventually notes that "OSS is best suited to horizontal domains in which there is widespread agreement on the design architecture and the general composition of the software requirements is fairly well known and unproblematic." And goes on to contrast that with "On the other hand, in vertical domains where requirements and design issues are a function of specific domain knowledge that can only be acquired over time – the case with many business environments – there are not likely to be as many OSS offerings."

Its conclusion? "While the notion of OSS as a silver bullet might be an inaccurate stereotype, OSS projects clearly exhibit many of the fundamental tenets of software engineering."

Reference -

Fitzgerald 2011 – "Open Source Software: Lessons from and for Software Engineering," IEEE Computer, Oct., 2011; Brian Fitzgerald

Parnas On Software As Engineering

Another of the "three dots" article is by David Lorge Parnas, one of the best of the software engineering technologists, one who clearly understands the links between theory and practice in the software field. In [Parnas 2011], he decries – as many do – the lack of a traditional engineering content in the field of software engineering, and then concludes with a succinct and marvellous summary of "my position:"

- "If we are still writing papers arguing that we need some theory or mathematics to be a profession, rather than arguing about specific areas of theory that a software engineer should know, we have not established a profession."
- "If we find that individual application areas are discovering common problems and solving them with their own terminology and specialized techniques, we have not yet established a profession."
- "If we do not consistently distinguish between engineering problems, management problems, technology problems, and business-plan issues, we have not yet established a profession."
- "If we are, as many others have noted, a field that is dominated by fads with clever acronyms that cause a flurry of interest and then fade away, we have not yet established a profession."
- "If we continue to treat software engineering as a 'grab-bag' research area rather than as a regulated profession, we have lost sight of the original goal."

Parnas identifies himself as "president of Middle Road Software," a title apparently new to this long-time software engineering spokesperson.

Reference:

Parnas 2011 – "Software Engineering – Missing In Action: A Personal Perspective," IEEE Computer, Oct., 2011; David Lorge Parnas

CSDP Exam Contents

That same issue of Computer lists the content of the Certified Software Development Professional credential exam, in terms of its examination categories:

Software requirements – 11%
Software design – 11%
Software construction – 9%
Software testing – 11%
Software maintenance – 5%
Software configuration management – 5%
Software engineering management – 8%
Software engineering process – 7%
Software engineering methods – 4%
Software quality – 7%
Software engineering professional practice – 5%
Software engineering economics – 5%
Mathematical foundations – 3%
Engineering foundations – 4%

And a Note on Estimation...

And finally, it contains this tidbit - "On average, the effort required to complete a [software] project deviates 30% from the initial projection."

Cyber Threats Forecast

What might be the cyber threats of the future? According to Georgia Tech's Cyber Security Summit, held recently, they are:

- Search poisoning – Attackers optimize malicious links among search results, so that users are more likely to click on their URL because it ranks highly on a search engine.
- Mobile web-based attacks - Attacks aimed specifically against mobile web browsers.
- Stolen cyber data used for marketing – Private user information stolen and sold to legitimate business channels for marketing purposes.

Big Brother, but Under Your Control

Are you concerned about whether you're spending your computer time productively? If you're not, perhaps your boss is?!

There's a new software application called RescueTime (RescueTime.com) that keeps track of how you use your computer and, after a few days, shows you a summary of what you've been doing. It lists the websites you visit and the software packages/emails/documents you open or create. It also allows you to block certain websites and issue yourself reminders if you're spending too much time in activities you think are counterproductive.

In short, it's a personally tailorable Big Brother.

Perhaps the Shortage of IT Positions is Over!

Signs of the Times –

Hiring advertisements in the October 2011 issue of IEEE Computer –

Yahoo – nearly three pages of densely-worded, specific position listings!

Apple – two pages of less densely worded listings for 8 kinds of positions

Standford (?!) University Seeks Faculty Applciants

Life's embarrassing moments department...

A "faculty opening" position ad in the Oct., 2011 IEEE Computer magazine says that "Standford University" is looking for candidates for its Dept. of Computer Science faculty. Since later on the ad mentions "Stanford University," we must assume that "Standford" is a (terribly embarrassing!) typo!

Eurology – It's Not What You Think!

According to a cartoon in the newspaper The Australian, the scientific study of the economic problems of the Eurozone countries is known as "Eurology."

As the New York Times Sees It

The Future of Computing

The New York Times, in its Science Times section, recently (Dec. 6, 2011) discussed "The Future of Computing" from a global viewpoint. What topics were covered?

- the evolution of Silicon Valley
- China's aims for high-tech primacy
- the promise of quantum computing
- the outlook for self-driving cars
- prospects for research taking root in Africa
- innovating in an open-source society
- the potential for computer scientists to help cure cancer

Jobs vs. Richey

Robert L. Glass

I've had a couple of died-in-the-wool software folk tell me that they decry the major fuss made over the death of Steve Jobs vs. the minor fuss made over the death of Dennis Richey.

That thought hadn't occurred to me. I understand that Jobs was a quirky person, but at the same time his effect on the computing (and entertainment) world was probably more vital than Richey's. Whole companies died or lived via his efforts. Richey's achievements are significant, especially to those living in the Unix/C world, but would you really class him ahead of Jobs?

I'll print the best of the responses to this issue.

Six Impossible

continued from page 10

robustly address the topic of dishonesty on software projects and by software practitioners.

The Dark Side of Software Engineering: Evil on Computing Projects,[2] by Johann Rost and Robert L. Glass, is a survey of the bad things people do on software projects, plus some other software-related evils like hacking and computer scams. The range of subjects means the book is a bit of a hodgepodge, but it's a fascinating read. There's some numerical analysis I found less than compelling, mainly because the samples are too small for the numbers to have real statistical significance. This doesn't detract from the book overall, which has lots of good stories, qualitative analysis and suggested remedies from which we can learn important lessons. Among other benefits, you can learn to spot patterns of certain kinds of nefarious behaviour you may not previously have noticed on projects.

A nice counterpart to *Dark Side* is *The Clean Coder: A Code of Conduct for Professional Programmers*,[3] by Robert C. Martin (Uncle Bob, as he's known in the Agile world.) I doubt anyone will be surprised to hear that Bob Martin comes out strongly against lying on software projects. You may be surprised at how he defines lying, including that it's a lie to say you'll try to meet a date when you already know you cannot achieve it. That advice alone is worth the price of the book. Martin also emphasizes the importance of learning to say "no"—an essential skill for people who want to tell the truth (one that Jerry Weinberg has been promoting and teaching for a long time).

Dark Side and *Clean Coder* are important for software practitioners of all specialities, including testers. The subject of professional ethics is vital for us all. We need to learn to recognize and stamp out lies and other ethical lapses on our projects—not just in other people, but in ourselves. These books will help. It's a bonus that they are also good reads: engagingly written and full of good stories we can relate to.

I don't know whether people on software projects are any more prone to dishonesty than the culture at large. I do know that deception in varying degrees is common on software projects. It's a dirty little secret we don't much explore, although it's an open secret in the business. I'm glad to see that other people are writing about it.

One of my articles on tester and consultant ethics[4] prompted a reader to protest that people can take grave risks telling the truth on software projects, and in a tough economy truth may be a luxury some can't afford. I believe that an expedient lie is the luxury we can't afford. Not for our professional reputations, not for our projects, not for our self respect. I think most testers would agree.

Lying hurts software projects. How many projects have you been on where "everyone knew" the schedule was fiction? Everyone except management, that is, the managers having conveniently forgotten they'd set the project up for failure at the beginning. And perhaps those managers had confidently told senior executives the fake schedule was all certain and wonderful—and now they're running out of budget and running scared. So they put pressure on their teams.

Apart from the cynicism engendered by living a lie, software people do shoddy work under pressure. Designing, coding and testing are all difficult work that requires a clear head. In my experience, the projects where people lie the most produce the worst software.

Lying hurts people too. Every time I present at a conference on "When a Tester is Asked to Lie",[5] one or two people take me aside and say, "This is so timely. It's happening for me right now and I don't know what to do." Others tell me it has already happened to them, and it's a nightmare they don't ever want to repeat. A test manager told me he'd been fired because "we don't think you're comfortable lying to the customer." ("Too right I'm not!", he said to me.)

Yes, it can be risky to tell the truth when others are lying. It can also be unexpectedly rewarding. I have more than once seen an unhappy project benefit from the act of a single tester or programmer bravely stepping forward and saying, "I'm way behind. I'm not going to make the schedule, and I'd like to explain why." Sometimes that can be all that's needed to enable others to speak openly. Though the ensuing discussion might be painful, it could lead to a realistic replanning exercise that puts a project on an achievable path to recovery.

So why don't we just stop lying? We don't have to practice believing ANY impossible things before breakfast. We don't have to convince other people to believe them.

I've loved reading *Alice* most of my life, but I've never taken the White Queen to be a role model. Have you?

What lies have you heard on software projects? Add to my list

I've written mostly in this article about schedule and status lies, but my list of thirty-plus includes many other types.

I've put it on my blog at http://quality-intelligence.blogspot.com/ so you can see the whole list and add your comments. Can you add to the list? Do you have experiences of project lies (or truth-tellings) you'd like to share?

Notes

[1] Lewis Carroll, *Through the Looking-Glass* (Kindle edition), Chapter V.

[2] Johann Rost and Robert L. Glass, *The Dark Side of Software Engineering: Evil on Computing Projects* (IEEE Computer Society, John Wiley and Sons, 2011).

[3] Robert C. Martin, *The Clean Coder: A Code of Conduct for Professional Programmers* (Prentice Hall, 2011).

[4] I've published 4 other articles dealing with the subject of ethics, all on Stickyminds.com. Copies are also on the Publications page of my website, www.quality-intelligence.com. Search for the titles:

Sophie's Choice (September 1, 2007)

Deception and Self-Deception in Software Testing (June 1, 2009)

Negative Positive (February 8, 2010)

No Compromise (June 21, 2010)

[5] Besides the conference presentation "What Price the Truth: When a Tester is Asked to Lie", which deals with a specific type of project lying, I also lead an experiential workshop on the broader topic of "Deception and Self-Deception in Software Testing".

Obfuscation: Code or Article?

Robert L. Glass

There's something I truly don't understand about this news article [Jackson 2011].

The subject of the story is a contest for writing "obfuscated code." So far, so good. I understand "obfuscation," I understand "code," and in fact the whole thing sounds like a cute and fun idea.

But then things start to get murky. The article mentions that obfuscated code must be "mind-bogglingly difficult to understand." They are "not necessarily looking for badly-written code," and in fact go on to say that winners in the past "have actually been offered jobs (presumably to write non-obfuscated code)." But the reason I truly don't understand what they're looking for is that the article says "The code must be maintainable and adaptable," yet it later goes on to say "the judges are looking for code written in such a way that another programmer would have extreme difficulty figuring out what the program actually does."

The article tries to illustrate its topic by talking about past winners, which included:

- a 2001 program that "rearranged a line of input and then reorganized itself so that it completed the same task using a different algorithm the next time it ran."
- a 1987 one line program that looks like it can't possibly compile, but does.

The contest rules specify that the code must be compilable via a standard compiler, and must come with documentation that clearly states what the program is supposed to do.

So there you are – obfuscated code must be "maintainable" but "another programmer would have extreme difficulty figuring out what it does." That doesn't fit my notion of what constitutes maintainable code! Is there anyone out there who has participated in this contest or can explain why this description makes sense? If not, I suspect this is all a January Fool's joke, and the real contest is to write obfuscated computing-related articles - like this one!

Reference:

Jackson 2011 – "Obfuscated code contest returns," networkworld.com/news, Nov. 22, 2011; Joab Jackson

Six Impossible Things Before Breakfast

Fiona Charles

Originally published in Tea-time with Testers, June, 2011. Slightly revised for The Software Practitioner.

Alice laughed. "There's no use trying," she said: "one CAN'T believe impossible things."

"I daresay you haven't had much practice," said the Queen. "When I was your age, I always did it for half-an-hour a day. Why, sometimes I've believed as many as six impossible things before breakfast..." [1]

"Aha!" I thought, "Then you must have worked on software projects!"

Waiting for a bus a couple of years ago, I began listing software project lies. The exercise kept me nicely occupied until the bus arrived and through most of the ensuing ride. I ended up with an astonishing thirty examples of lies.

In case you think that sounds like an improbable number, I'll begin by outlining the parameters I set. First and foremost, I had personally to have heard the lie told one or more times. Each lie or category of lies had to be material to a software project, though it could have been told to make a sale before a project began or to describe a project after its end. Each lie had to be relatively common in the industry—or at least not rare. A lie had also to be significant to a project: to have influenced perceptions and/or decisions. And finally, I excluded malicious lies intended to subvert or sabotage an individual on a project, though I have both heard and (on one unhappy project) been the object of some of these kinds of lies.

Reviewing the list as I prepared to write this article, I immediately added a couple more, then roughly grouped the list under the headings:

- lies told to make a sale or get funding for a project
- management lies
- lies about programming
- lies about testing
- lies told by customers
- miscellaneous lies (such as deliberately downgrading bug severity to make a release)

The (depressing) reality is that in a career spanning three decades, I cannot recall a single project where there was not at least one significant and material lie. I thought I remembered consulting on one very nice-minded and squeaky-clean project, but then I recalled the programme manager telling a PM, "We're pushing the date of your project out, but it's vital that you do not tell your teams. Everyone needs to believe in the original date so they don't slack off."

Does the number of lies on my list horrify you? Am I exaggerating? Or could it be that we have all heard so many lies so often on software projects that we've become desensitized? I mentioned the "don't tell the team" lie to a notably honest programmer friend, and he said, "Well... we hear that one so often it almost doesn't qualify as a lie."

"Right", I said. "And how is it different from those other oldies but goodies:

'We're running 3 weeks behind, but it won't impact the end date.'

and 'We just need to work a couple of weekends to sort out all the quality problems.'"

(A project manager I once worked with used to say, "Show me where on your project plan it says 'A miracle happens here'!")

Perhaps you'll say team lies like this aren't deliberate lies—that the techies or the testers are just being over-optimistic, persuading themselves that the commitments they're making aren't patently impossible. We've all done it, haven't we? And yes, we probably have. But self-deception is still deception. A lie to one's self is no less a lie.

How far really are all those "we're going to make it" team lies from the infamous bait-and-switch scam some consulting firms routinely practice, or the deliberate underbids put forward to make a sale? ("We'll more than make it up in change requests," the sales people say.)

Lying of various kinds is quite common on software projects. Past the sale, the lies often continue with management arbitrarily slashing estimates and imposing upfront commitments to deliver fixed scope with fixed staffing within unachievable timeframes and budgets.

On projects that start out with fraudulent commitments, lies are propagated in the hotbed of fear. Managers demand certainty from people who are often barely in a position to give better than rough estimates, plus or minus fifty percent. Programmers and testers make desperate commitments they know they can't really achieve, and then, week after week, over-optimistically report their progress and status (lie). People lie—or avoid telling the truth, which is pretty much the same thing—to deflect blame and to get management off their backs, hoping to put off well-founded suspicions that they aren't going to deliver the impossible, and anxious to get on with productive work.

How many so-called "troubled" or "failed" projects would actually have gone much over time and over budget if they hadn't started out committed to fiction?

We may think lies like these are symptomatic of big waterfall projects, and indeed that is often true. But Agile projects are not immune to deception. A quick scan of the Agile blogosphere reveals plenty of discussion about impossible project or sprint commitments made by stakeholders outside the project teams or even by the teams themselves.

The rapid feedback built into an Agile process leaves much less room for practitioners to hide impossible estimates or falsify status. But teams calling themselves Agile have been known to play games with the concept of "done", redefining it to meet the actual state of the work when they haven't achieved their goals. Human nature is what it is, on Agile or waterfall projects.

It's a healthy sign, therefore, that two recent books by well-known and respected authors

(continued on page 9)