Will They Try Harder?
IBM Is Now Number 2

It's the end of an era!

IBM was passed as "the world's biggest computer company" in late 2006! It had held that position for over 40 years, since the advent of the Series 360 mainframe computer.

Who replaced them? Hewlett-Packard. Why? Because

- IBM sold its PC business (to China's Lenovo)
- IBM quit the printer and storage business.

What's IBM's reaction to this? IBM CEO Sam Palmisano says "We don't measure ourselves by trying to be the biggest. We'd like to be the best earnings and cash generation entity in our industry."

Yea, right!

Information Source –

"HP Set to Take the Big From Big Blue," The Australian, July 18, 2006; John Sterlicchi

THE SOFTWARE PRACTITIONER

NOVEMBER – DECEMBER 2006 | The newsletter by and for software professionals. | VOLUME 16, NO. 6

The Future of IT
Employment Increasing, Huge Change, High Morale

The future of IT looks quite bright, according to the results of a survey of IT CIOs [Alter 2006] published in CIO Insight.

Regarding employment:

- 47% of respondees saw the number of full-time IT employees increasing over the past year, and only 20% saw it decreasing
- 45% expect an increase in the number of IT employees over the next year, and 12% see a decrease
- 73% see new employees needed because of "new applications and infrastructure," and 69% see them needed because of general corporate growth
- 36% will be hiring project managers, 33% programmers/system developers, and 30% help desk support
- 55% will be looking for people who have superior business (as opposed to technical) skills and knowledge, and 79% see those people as more likely to be promoted
- 82% of IT employees are in-house staff, 9% are contractors, 6% are domestic outsourced, and only 3% are foreign outsourced

Regarding the IT organization:

- 57% see IT as going through more change than they have ever seen before
- 52% believe it is viewed as strategic, and 48% believe it is seen as a staff function
- at 49% of companies, the CIO reports to the CEO (at 22%, it's the CFO; at 21%, it's the COO)
- 72% believe "most of our IT professionals understand my company's business strategy"
- 65% say that most ideas for using new technology come from the business users, not the IT staff
- 79% disagree with the statement "our IT department's morale is so low that it impedes our company"

Reference:

Alter 2006 – "IT's Future is Brighter Than You Think," CIO Insight, Aug., 2006; Allen Alter

IT Job Growth Solid

There's good news in the IT jobs department, at least if you are hoping to show that an IT education leads to plenty of lucrative jobs…

- Growth rates for IT jobs are "solidly exceeding the outsourcing rate"
- IT employment is now 17% higher than it was in the dot-com boom year of 1999

The source for this and other related data is a study/literature search by Gettysburg College Department of Computer Science.

Terminology Matters
CIO a Better Job Title Than VP/IS or Director/IS

The "C" in "CIO" stands for Cash. At least, that's one conclusion of the annual 2006 Salary and Careers Survey conducted by CIO Decisions magazine (and published in their June, 2006 issue).

The reason for that conclusion is the salary data that shows that CIOs earn far more than their VP/Director-titled colleagues – 66% of CIOs earn more than $150,000 a year in salary and bonuses, compared with 56% of VPs and only16% of Directors.

Regarding raises and bonuses, the same picture holds – nearly a third of CIOs received a raise of 6% or more in 2006, compared with only a fifth of IT Directors. And regarding bonuses, nearly half of the CIOs who received a bonus got more than $30,000, as opposed to only a third of VPs and 10% of Directors.

These positions are adding to their longevity, according to the survey. Whereas in 2000 the average CIO had a tenure of only 18 months, last year Forester estimated that average tenure was 3.6 years, and Gartner estimated it as being longer than 4 years. And more and more often CIOs report directly to the corporate CEO – almost half, says the survey, as opposed to only 26% of VPs and only 16% of IT Directors.

However, the news from the CIO front is not all good. As many as 58% of all IT executives had been laid off or fired from their position at one point in their careers (the figures were 58% for those aged 60-69, 40% for those 50-59, 28% for those 40-49, and 23% for those 30-39).

Regarding job switching, most (42%) CIOs found their current job via networking (whereas only 21% were promoted into it, and 18% were recruited).

There were 457 subscriber respondents to the survey, with an average respondent age of 46 and an experience level of over 20 years in IT.

Also in This Issue

"I'm sorry, we can't hire you. Your references check out, and your background check was stellar… but nothing came up when I GOOGLED you!"

"Young man, I won't have you out at all hours doing Google-knows-what!"

THE SOFTWARE PRACTITIONER

ISSN = 1083 - 6861

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

ART EDITOR:
P. Edward Presson

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 earned, 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 _____

_____

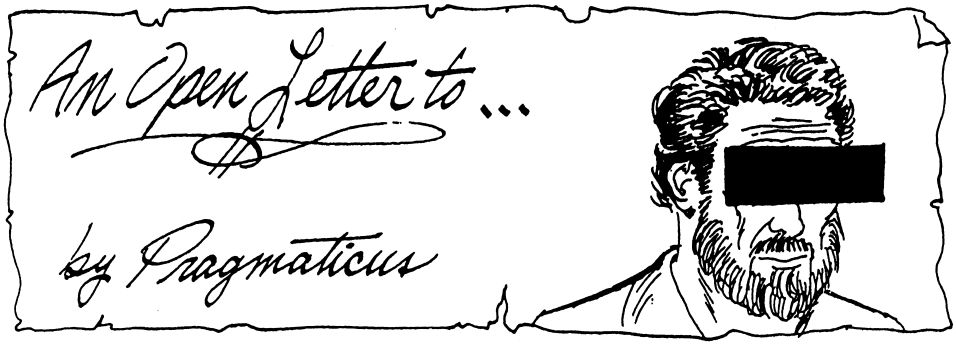
_____

E-mail _____

Shareware Subscriptions!!!

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

_____



An Open Letter to the Whole Software Engineering World

By Pragmaticus

Hey, guys. There's something we do really badly in our field. Like, we really almost never do it at all.

And that's postmortem reviews. Or even premortem reviews. We hardly ever take a look at how our software project is going, give it a thorough critique, and figure out what lessons we can learn from that.

The result of that is we don't learn from our mistakes (perhaps worse yet, we also don't learn much from our successes!) Lessons learned from practice are barely visible in our field. "Best of practice" studies usually end up telling us to do what the textbooks say or what theory says, and that's no good, because all too many textbook/theory ideas have never been evaluated in the cauldron of practice.

And then our friends the consultants and academics criticize us for not improving how we go about our business. Well, duh – they're right! It's hard to improve when you don't study what you did or what you're doing.

Now, of course, you're going to say that in our software project "death march" world, there's never time to work in a good post/pre mortem. And you're right, of course. But I don't think any of us want that death march world to prevail. There are times when we simply have to shout "stop the death march, I want to get off, there's something more important to do." Try it … all together, now!

I could go on and belabor this point for several more paragraphs. But I'm not going to do that. Instead, I'm going to suggest that you read the article "Postmortems and Checkups" by Linda Rising elsewhere in this issue of SP. And that you take what it says to heart.

To the Editor:

Regarding the "OO Inheritance - Most Cited, Least Used" article in the July SP, I agree with your conclusion: "there is a huge discrepancy between what writers on the subject of OO see as important, and what is actually used in practice". Indeed, it seems that practitioners prefer composition, not inheritance. See http://www.artima.com/lejava/articles/design-principles4.html, " Design Principles from Design Patterns - A Conversation with Erich Gamma ."

In my opinion, design patterns are a layer more important than academic OO principles. For example, a VB programmer (who converted to OO, using C#) I was going to replace astonished me when I saw a declaration of a connection string (the same value) in every class he wrote! To my question, he answered something like "but every class must have its own responsibilities, not depend on other classes"! Perhaps he wanted to avoid static variables, but he would have used a Singleton / Factory pattern to get a connection to the database, therefore improving maintenance issues.

Roberto Mannai, Italy

Quote of the month:

"Excellent people with average processes will normally outperform average people with excellent processes."

- Ed Yourdon, in "Focus on Ed Yourdon ... a CAI State of the Practice Interview," Computer Aid, Inc., Sept. 2006

Postmortems and Checkups

Linda Rising
www.lindarising.org
risingl@acm.org

If you had known at the beginning what you knew at the end, how much would you have saved?

Many software development organizations use postmortems at the end of a project as a way of improving their processes. This can be effective but it doesn't really help the project that has just ended. Teams should not only pause at the end of a project to capture lessons learned but there should also be checkups before the project ends. In much the same way that preventive care helps us live longer, healthier lives, projects can pause for a quick reading and take corrective action while there is still time to apply the lessons learned.

Checkups are especially appropriate if your team is following an iterative development process. At the end of each iteration, take time to reflect. The benefits are easier to identify because you're more likely to remember things that happened a few weeks ago and a checkup takes a lot less time than a full-blown postmortem.

But, wait, let's back up. I made one of those blanket assumptions at the beginning of the first paragraph. Just in case your organization needs a reminder about the benefits of postmortems, let's have a little review.

First of all—the word "postmortem." It calls up a vision of a group of concerned analysts gathered around a cold, dead body. Most of the projects I've been on produced something worthwhile. The team should celebrate success as well as mourn mistakes. A good postmortem, or as I would prefer to call it, and will from now on in this article, a good ***retrospective*** provides an opportunity to see what went well as well as those oh-so-easy-to-identify-things-that-didn't-go-well that we don't want to repeat.

Conducting retrospectives is one of my professional interests. As an independent consultant, I help teams by facilitating retrospectives and supporting the follow-on activity of building on the knowledge captured in the process to make future projects better. There are others who are in the retrospective business. We met recently in the first ever retrospectives facilitators' conference in a little town on the Oregon coast. We met to create an identity and to begin to share information about how we conduct our businesses and how we can improve.

Let me tell you a story about a retrospective, one of the most valuable I've ever facilitated. You can judge for yourself whether this kind of experience would help your organization. Names have been changed to protect the company and the team.

The team lead called me in early January. His voice was full of despair, "Linda, our project has just been cancelled. These guys are really upset. Some have already given their notice. I can't blame them. They worked over Christmas to get last minute fixes done and now management says it's over. I want to make sure that this kind of thing never happens again. I don't know whether there are patterns here but we need to learn from this." Of course I took on this messy assignment. Failed projects happen. But when they do, team members take that failure personally. They carry that guilt and anger with them to the next project—even if the next project is in another company. If these issues are not addressed and continue to build in subsequent projects—burnout is inevitable.

Fred, my good friend and colleague, agreed to help me with what promised to be an especially challenging retrospective. This was a large team that had been together for over a year. I would need someone with me to facilitate this session.

My mentor, Norm Kerth, has written an excellent book [Kerth01] on this important part of the development cycle. He advises taking teams off-site for three days for an end-of-project retrospective. I usually don't have the luxury of either the off-site location or the extensive amount of time. Most managers are lucky to find a couple of hours and the budget to buy lunch. In our story, I had two hours to help 50 people through a retrospective.

Since the group was so large, we decided to use "quiet storming," a technique that allows each person on the team to write his feelings about any part of the project on a large index card and then post the card on the wall. We had categories across the top of the wall: Requirements, Design, Coding, Testing, Customer Interaction, and so on. Sometimes a card won't fit the existing categories and a new category is added. As cards are written, they are brought to the wall and posted. Writers peruse the posted cards and as they do, ideas for more cards are generated. Anyone can write anything and all cards are anonymous.

As the wall began to fill up, people were spending more and more time, reading the cards. At one point, nearly everyone was standing at the wall. It was eerie. It reminded me of the Vietnam War Memorial. People were re-living the history of the project, dredging up bits and pieces, flashes of insight, painful moments, happy moments. They stood in small clusters, whispering, pointing at cards, touching the cards. I let them have time for this.

When it was clear that no more cards were being added, I asked them to look at the cards in each category and see if they could detect any trends. I asked them to look at the bigger picture, look beyond the details on the cards and see if we could capture: (1) What went well. We need to document those good things—and there are always good things—that we're always forgetting. (2) What should be done differently? Let's not make the same mistakes again. (3) What do we still not understand? What puzzles us?

As we became more analytical, people began to think ahead and move beyond the painful recollections of this project. They found that they could easily identify good things. They could see that there had been successes—things they could be proud of. They were able to talk about things that hadn't worked well and what they would like to correct before the next project.

We were able to identify some potential patterns—from the good things and determine some action items—from the things that should be corrected on the next project. Follow-on meetings were planned and responsibilities assigned for the patterns and the action items.

Toward the end of this exercise, someone in the back of the room said, "What good is all this anyway? We know nothing will come of it. We're just going to go out there and do this all over again. We've already started to make some of the same mistakes on the next project and we've only been on it a few days!" To my very great surprise, the team lead jumped up and said, "I promise all of you that what we have captured today will be presented to every other manager in this company. This will not just go into some web page where no one will ever read it. I'll take it all the way to the executive council." I was impressed—I think the other team members were, too, because no one said anything for nearly a minute. They were all taking this in. Maybe this company really could learn from its mistakes. Maybe things really would get better on the next project. Maybe there really was hope.

I guess that's the point of the whole story. Without the retrospective, the team would have never found any hope. Each team member would have carried his burden onto the next project and the next, maybe never completely recovering. Good companies understand that what people produce reflects the things people bring to it.

One of my favorite stories about this phenomenon is from a book by Ruth Sawyer [Sawyer76].

He was Bavarian, little, very old. He had come to measure our davenport for re-covering. With painstaking care he got out of his coat and into his apron of blue-and-white-striped ticking, adjusted his pincushion, his shears, hung his tape measure about his neck. He got only as far as measuring the front; and there he sat, on his heels, while he told me about his experience as an under-master in the palace of King Ludwig. He talked about a special event held every year when everyone who worked for the king took part in an opera. Those who could sing were in the chorus. Those who played instruments made up the orchestra. A conductor from Dresden was brought to direct. The soloists came from the big cities. The great Wagner came. For a week the celebration was held; then everyone went back to work.

What the Bavarian upholsterer said at the last I have always remembered: "All the goodness, the lift of the heart that we got from playing those operas, we would put back into our work—in the draperies and tapestries we hung, in the cabinets we made. Nothing was lost. That is how it should be when you have experienced something great and beautiful. And so, my lady, something of those operas will go into your sofa."

There it is. Software developers or creators of any product put into their work the joy they find in their lives. When the joy has been taken away by hopeless schedules, 60-hour-weeks, missed ball games or piano recitals, it becomes harder and harder to retain that optimism and heart we software junkies are so proud of. One of the best ways of getting that back is a retrospective. Let the team have a moment to reflect on the past project. Let the team learn. And, even better, do this often, while there is still time to make a difference in the current project.

Our story has a happy ending. The team lead really did spread the word about lessons learned from the project. A report really was presented to the executive council. Several good patterns were also written and I began to talk about them in other retrospectives.

For more information on retrospectives, please buy and read Norm's book and look for a new book out in the summer of 2006 by Esther Derby and Diana Larsen [Derby06]

References

[Derby06] Derby, E. and D. Larsen, *Agile Retrospectives: Making Good Teams Great*, Pragmatic Bookshelf, 2006.

[Kerth01] Kerth, N., *Project Retrospectives: A Handbook for Team Reviews*, Dorset House, 2001.

[Sawyer76] Sawyer, R., *The Way of the Storyteller*, Penguin Books, 1976.

Standish CHAOS
Reporting On An Interview with the Study's Creator

Robert L. Glass

You must have noticed that we at SP have a fascination with studies of software project failure rates. We've published so many stories on failure data (just the most recent ones are cited below) that you may be getting fed up with the whole thing.

You probably have also noticed that we've been all over Standish for its well-known "CHAOS" studies that show software projects having huge failure rates. We don't believe their data and their conclusions, and have been so brash as to say so on numerous occasions, even challenging them in print to justify their findings (these are also listed below).

So perhaps we should give Standish a bit of equal time. They have, after all, believed for over a decade that they are providing a service to the software community by reporting software project success/failure data. What do they say to negate our doubts and refute our challenges?

At this point, ideally we would print a response from Standish answering to all those public challenges we've made over the last few years. Sad to say, however, there is still no such response. So what follows is the next best thing.

Deborah Hartmann has written an online story reporting on an interview with Jim Johnson, who is the "creator" of those Standish CHAOS studies and stories, and has been nice enough to share that story with us at SP (as well as obtaining permission for us to publish excerpts here with appropriate credit). That story was published on the online site InfoQ (www.infoq.com). Here is some of the most interesting content of that interview:

- Johnson believes the CHAOS report results are "representative of application development in general"
- he sees them as unbiased ("I don't think there's any bias in there")
- prior to the Glass/Pragmaticus challenges, the Standish findings have "not really" ever been challenged before
- Johnson is called a "seeker of the truth" by a colleague
- "we pay people to fill out the survey" … "we pay them for their time" (presumably this further removes any hint of bias in the responses)
- the survey respondees are broadly-based,

but there are "no vendors, suppliers, or consultants. So Microsoft isn't in our sample."
- "we're not asking for 'failure' projects."
- "it doesn't take a genius to know our methodology, it's always been public" (their original study, he says, resulted from a broad-based mailing, but more recent studies have invited people who meet certain criteria (none having to do with failure, he says) to participate on their website).

The above interview content is largely supportive of the CHAOS studies. But there are some odd questions raised by the interview, as well:

- Johnson makes the point that the Standish studies are about success as well as failure, and that out of the CHAOS studies they glean success factors they can report to their clients. But he also notes that Standish also seeks "instructive failures" as case studies, and says that he has personally written a book called My Life Is Failure.
- Johnson says that the original study resulted from an IBM class Standish conducted in Belgium, where one subject discussed was tracking sales of middleware, wherein the data "just didn't track right," leading to the Standish interest in project failure. My problem with this story is that I don't recall middleware being a hot topic in the early 1990s.
- perhaps their "methodology has always been public," as stated above, but noted researchers have repeatedly failed in their quest to get access to the data Standish collects. Standish says that is because they want to protect client privacy, and because the data is proprietary, but they also say that the data has been scrubbed of client identification.
- the InfoQ interview is largely objective. But at the outset they say that Standish is a "globally respected" source of independent primary research and analysis, and also that they are "a pioneer of modern research techniques;" such complementary statements are not the mark of objective research.

Perhaps the most interesting content of the InfoQ interview is this chart of project failure and overrun rates, gathered long-term over all of the CHAOS studies:

Project result	1994	1996	1998	2000	2002	2004
% Succeeded	16	27	26	28	34	29
% Failed	31	40	28	23	15	18
% Challenged	53	33	46	49	51	53
Average cost overrun %	180	142	69	45	43	56
Average time overrun %	164	131	79	63	82	84

It is particularly interesting to note that by far the worst year in the CHAOS studies was the first year, 1994 (the one most researchers cite), and the best year was 2002 (most researchers seem unaware of the CHAOS studies since 1994, and therefore rarely cite the data improvement over time. But note also that, since 2002, things have gotten somewhat worse).

The software field owes a debt to InfoQ for obtaining these direct responses from Standish on their CHAOS studies.

References:

Software Practitioner stories on failure data:

- July 2006: "How Large are Software Cost Overruns? A Review of the 1994 CHAOS Report," by Magne Jorgensen and Kjetil Molokken-Ostvold
- July, 2006: "More Data on Software Project Failure"
- May, 2006: "Failure Happens, But Increasingly Infrequently"
- May, 2006: "Software Projects: 40% a Total Success, Only 55% Failures"
- Nov., 2005: "Cutter Consortium: Some New Data in the Project Failure Statistics Wars"
- Mar., 2005: "Standish 'CHAOS:' Failure Up, Success Down, in 2004"

Stories/columns that challenge Standish' findings:

- Aug, 2006 Communications of the ACM Practical Programmer column by Robert L. Glass, "The Standish Report: Can We Really Believe it Describes a Software Crisis?"
- Nov., 2005 SP: "An Open Letter to Standish (Version 2.0)" by Pragmaticus
- May, 2005 SP: "An Open Letter to Standish About Its 'CHAOS' Reports" by Pragmaticus
- May, 2005 IEEE Software Loyal Opposition column by Robert L. Glass, "IT Failure Rates – 70% or 10-15%?"

Nov., 2003 IEEE Software, Guest Editor's Introduction (by Robert L. Glass) to a special issue on the state of software's practice

Feedback from a Poll of SP Readers On Workspace Preferences

by Bruce Gaarder

Recently I posed a question to the editor of the Software Practitioner about workspace preferences, and he decided to try something new, asking you readers to respond with your own reactions. What follows is a short version of my original question, and a summary and sampling of some of the most interesting responses (there were 13 responses in all).

The question:

I am looking for opinions and experiences regarding workspaces for software workers. The company for which I work is beginning to move to a "pod" or cubicle of 4 or 8 workstations with a pair of workers sharing a table-top island across an aisle from another pair. I would be interested in any literature or opinions bearing on this issue of workspace arrangement.

A summary of the responses:

Too much noise, loss of privacy (5 responses)

The reason for such a change is always cost, in spite of management saying other things (3)

Provide small enclosed "cockpits" (work carrels) for deep thinking tasks (2)

Several other items with one response apiece

Web sites cited:

http://www.research.ibm.com/journal/sj/171/ibmsj1701C.pdf
"IBM's Santa Teresa Laboratory -- Architectural Design for Program Development"

http://www.joelonsoftware.com/articles/FieldGuidetoDevelopers.html
"A Field Guide to Developers"

http://www.joelonsoftware.com/articles/fog0000000050.html
"Whaddya Mean, You Can't Find Programmers?"

http://www.joelonsoftware.com/articles/BionicOffice.html
"Bionic Office"

http://www.azstarnet.com/sn/printDS/117661
"Distractions are Costly"

http://www.kmworld.com/Articles/PrintArticle.aspx?ArticleID=14543
"The High Cost of Interruptions"

http://www.ecommercetimes.com/story/45606.html
"Teleworking - Is It Right For Your Enterprise?"

http://www.zazamedia.de/pages/BASEX%20-%2005-06.html
"London Interruption" [13-02-06] entry

http://gd.tuwien.ac.at/softeng/lambs-archive/archive/cubicle
News group exchange

http://www.12simplesecrets.com/management.htm
"The 12 Simple Secrets"

http://www.news.cornell.edu/releases/Jan01/noisy.offices.ssl.html
"Even low-level office noise can increase health risks and lower task
motivation for workers, Cornell researchers find"

http://www.findarticles.com/p/articles/mi_m3495/is_n3_v43/ai_20853287
"A humanistic approach to space - offices designs"

http://money.cnn.com/2006/03/09/magazines/fortune/cubicle_howiwork_fortune/index.htm?cnn=yes
"Cubicles - the Great Mistake"

A Sampling of Responses:

Organizations are definitely moving toward

the open, collaborative spaces - since I see a lot of "agile" teams, this is part of that approach to development. But, these same organizations also realize the importance of a quiet space, not only for working, but for a change of pace or to make a private phone call. So, they have both.

Here's one example. A large project in Texas had several large rooms where 4-6 people worked in a collaborative setting. Each large room had two little "closets" where an individual could have a quiet space as needed. The small closets didn't belong to anyone, they reminded me of the study carrels we used to have in the library. They were sound proof and allowed a place to think and work quietly.

The large project also had a small kitchen with a few tables where the team had breaks and lunch. It looked like a very nice working environment! I see this in so many places, I might almost call it a trend.

Our building has so-called "cockpits" (enclosed 1-person work carrels) where you can work alone undisturbed or make a private call. One obvious problem with this: you have to walk to get to them. Perhaps not surprisingly they are often empty.

I do believe that open cubes increase communication, but only if the people in the pod are working on the same project -- and I mean the same *specific* project, not projects that are only loosely related. So I think increased communication can be a definite benefit.

No question that collaboration and knowledge transfer are important, at some points in the software creation process, but they won't be improved with cubicles. The impediments to collaboration and knowledge transfer among programmers are cultural, not physical.

The real reason for cubicles is cost. It's vastly cheaper to have a big room and movable partitions. Pods and open space may be great for some professions, such as advertising, but for programmers, in my view all those "benefits" are just a cover story for lower cost.

The proposed design has some positives, but having less space for whiteboards sounds like a real negative to me. I work in a building full of cubicles. Almost every cubicle has a small whiteboard, about 2'x3'. We use them often when having informal discussions involving two or three (or sometimes more, even though it's a squeeze) people. I think they help communication a lot.

What have I found that enhances software development productivity?

1. Long periods without distraction; no phone calls, emails, internet browsing. For me a maximum of 2 hours.
2. At the end of each development shift (e.g. up to 2 hours) a good break from development to work on small/short tasks to relax the mind. Make some phone calls, respond to e-mails etc.
3. Views are good. From my desk I have a view of the outside world into something serene i.e. not distracting. I have found it greatly reduces eye and back strain if, during moments of consideration I can look away from my screen, into the distance, but not be distracted by what I see.
4. Collaboration is best done during these breaks from development. When a vexing problem is taking too much time then end a development session. Spend some time on non-development tasks to relax the mind then take the problem to associates, news groups, on-line resources etc., for a solution before commencing the next development session.

More Data on Software Failure: 67% Successful!

Yet another source of data on software project failure will soon be published. In [Sauer, Gemino and Reich 2007], the authors report on a study of project experiences in the U.K. and find that:

- 67% of projects are delivered close to or exceeding expectations for budget, schedule, and scope.
- 23% of projects are budget or schedule challenged
- only 9% of projects are abandoned.

These figures contrast drastically with those reported by Standish, which repeatedly (in all of its every-two-year studies) shows projects challenged or failing at the over 50% level.

The research approach used by the authors for this study was to ask 421 experienced UK project managers to discuss their most recent completed project.

There was, however, a peculiar aspect to the findings of this study. Although it is commonly assumed that large software projects fail more often than small ones, in this study "Star projects" (those where project performance was superior) were "distinctly larger" than the less effective but still successful "Good projects." The authors speculated that "organizations assign their very best project managers to the projects with the largest budgets."

The authors also found that projects with managerial changes were less likely to be successful. It was not clear whether the management change was a cause or a result of poor project performance, however.

In a private communication, the third author of the paper notes that data collected for a subsequent study in the US shows the same patterns as their UK data, thus reinforcing the disagreement with Standish data. This same author suggests "we [researchers and practitioners] need to stop using Standish, start collecting our own data, and put [Standish] behind us."

Reference:

Sauer, Gemino, and Reich 2007 – "IT Project Performance: The Impact of Size and Volatility," accepted for publication in Communications of the ACM; Chris Sauer, Andrew Gemino, and Blaize Horner Reich

"A Tortured History of Bad Software"

We at the Software Practitioner engage in biased reporting when it comes to software project failure! You have probably noticed that we tend to paint a rosy picture of the success/failure rate for software, and emphasize research findings that support that viewpoint.

So in the interests of fair play, in this article we will do the opposite. This is a report on a very visible piece of writing that appeared in IEEE Spectrum several months ago [Charette 2005]. In that article, Robert Charette, a consultant who specializes in risk management activities for software projects, presented an alarmist – and alarming – view of where software has been headed, and where it's going in the future.

For example, Charette said such damning things as:

- "few IT projects … truly succeed."
- "the problem only gets worse as IT grows ubiquitous"
- "over the last five years, … project failures have likely cost the U.S. economy … as much as $75 billion"
- all of this represents "a tortured history of bad software"

Charette also speaks of a "Software Hall of Shame," identifying a dozen or so projects that lost huge amounts of money for their companies, and liberally sprinkling his article with case studies of failed projects. He also notes that "IT spending is now one of the largest company expenses outside of employee costs."

The data Charette cites to make his point is not quite as convincing as his rhetoric, however. He suggests that "5-15% [of software projects] will be abandoned …," and notes later in the article that "I am convinced that the failure rate is 15-20% for projects that have budgets of $10 million or more." This certainly supports his notion that "such failures occur far more often than they should," but those percentages – being far from the failure rates that Standish cites in its Chaos studies - seem mild compared to the extreme verbiage that represents the main thrust of the article.

Still, what Charette says here is typical of the bad press software projects are getting in a great deal of the literature – be that literature academic, practitioner, or popular. And, once again, we present it here in the interests of journalistic fair play.

Reference:

Charette 2005 – "Why Software Fails," IEEE Spectrum, Sept., 2005; Robert N. Charette

Dagstuhl Revisited
Trends and Status of Empirical Software Engineering Research

Schloss Dagstuhl, Germany - 14 years ago a conference was held here at Dagstuhl castle on the subject of empirical research in software engineering. During the last week in June, 2006, the first repeat of that conference was held, with over 60 attendees from all parts of the software engineering world gathering together to revisit that topic.

The purpose of the conference was to update the status of empirical research into software engineering, 14 years later, and as part of that updating two interesting presentations were made:

Marv Zelkowitz of the University of Maryland presented his own update of an earlier study on the prevalence of computing research that evaluates its findings. (Numerous studies through the years have shown that the lack of validating research may well be one of the most serious problems in computing research).

Walter Tichy of the University of Karlsruhe presented his initial study of the characteristics of empirical research in software engineering.

What made Zelkowitz' study particularly interesting was that, for the first time since his studies began in 1985, there was an improvement in the quantity of validating research. The number of studies that did not support their findings with validating research dropped from 27% in 1985 to 11% in 2005, Zelkowitz found, and the percentage of those papers that used one or more validation methods rose from 29% to 66%. Zelkowitz concluded that "clearly the situation is improving," and speculated that the accessibility of open source software may be one reason this data is improving. But he also noted that his study assessed the quantity, but not the quality, of those validations.

Whereas Zelkowitz' study was longitudinal, covering several snapshots over a 20-year span, Tichy's findings were a single-point summary of empirical research, circa 2006 (his study examined the full 10.5 year spectrum of publications in the Journal of Empirical Software Engineering). He particularly focused on the quality of those studies, via such classifications as research method used and the background of the study subjects (e.g., professional vs. student). The most common topic areas, Tichy found, were metrics and software process. But perhaps more interesting were Tichy's views on "what is missing?" from such empirical research. Tichy noted few studies of such formerly important topics as programming languages, design notations, and formal methods. He also suggested several Grand Challenges for the field, including

- which software methods work, and why?
- including context in studies (overcoming the "it depends" issue)
- how to handle outsourcing / global development

Together, the two presentations gave a nice view of both the status and trends in empirical software engineering research.

The 30th NASA/IEEE Software Engineering Workshop 2006

by Christopher Ackermann and
Mikael Lindvall
Fraunhofer Center for Experimental
Software Engineering Maryland

Introduction

For the 30th anniversary of the NASA/IEEE Software Engineering Workshop (SEW-30), NASA technical staff, contractors, academics and industrial practitioners interested in the advancement of software engineering gathered at the Loyola College in Columbia, Maryland. Conference Chair Mike Hinchey coordinated the event, which provided a forum sharing experience as well as discussing new and emerging technologies.

Researchers and practitioners from different countries participated in the workshop and provided insight into the research work that is being conducted in the different research centers, universities and companies. Many good speakers presented their findings, approaches and techniques addressing various aspects of software development and maintenance. Interesting talks on topics from requirements modeling to formal methods and their application were presented. In this summary, we can not entirely cover all of the talks but mention a few that caught our attention.

Keynote speaker Victor Basili from the University of Maryland and the Fraunhofer Center for Experimental Software Engineering Maryland opened the workshop with a presentation about the role of empirical studies in Software Engineering. He pointed out that human-based studies are crucial to analyze and synthesize products, processes and the relationships between them. The results can then be used for evolution, which requires the willingness to change the way we think. Vic used examples from some of his recent projects to show how the use of empirical approaches and their results can be beneficial.

Together with Barry Boehm, he leads the Center for Empirically Based Software Engineering (CeBASE). The goal of CEBASE is to use empirical results to help researchers evaluate software technologies and to help software developers choose among them.

He was also one of the leaders of NASA's High Dependability Computing Project (HDCP), which aimed to increase the ability of NASA to engineer highly dependable software systems via the development, evolution, and empirical evaluation of new technologies. As part of HDCP, he developed the Unified Model of Dependability (UMD), which is a requirements engineering framework for eliciting and modeling dependability requirements. Also part of HDCP, he led the development of a software testbed, which is a set of artifacts and the infrastructure needed for running experiments with the goal to evaluate technology. The testbed is based on a safety critical air traffic control software component called TSAFE and seeded with faults of various kinds that can be use to determine which ones can be detected by a particular technology. One such technology was Tevfik Bultan's design for verification approach that is based on model checking. The result of that work earned a best paper award at the Automated Software Engineering conference.

The goal of the High Performance Computing Systems (HPCS) project is to improve buyers' ability to select the best high end computer, not only based on computation speed, but also based upon productivity. The productivity pertains to the Time To Solution (TTL), which takes into account not only the execution time but also the time to develop the software.

Victor's vision is an empirical research engine for software engineering that facilitates experimentation and evaluation of technologies and that could be used to build models supporting the decision making process. He stressed that we need empirical studies to test and evolve technologies for their appropriateness in context.

Other speakers also reported about achievements in the empirical domain. Giuseppe Lami from the Institute of Information Sciences and Technologies in Italy presented the results of an empirical study that investigated the relationship between quality of requirements and quality of the final product. The hypothesis that expressiveness defects in requirements result in an increased number of test failures was supported by the results of this study, which was based on data from an industrial software project.

The empirical study conducted by Christopher Ackermann from the Fraunhofer Center for Experimental Software Engineering in Maryland aimed to gain understanding of how change requests affect software. In order to determine change impact, he analyzed data from historical change request and their implementations. His results suggest that the implementation effort of many of the change requests was underestimated because the programmers were not aware of all the software aspects that were affected. His idea is to help programmers understand the severity of the change by illustrating the change impact in a set of selected diagrams that would most clearly reveal the impact on the different software aspects.

Other interesting papers that we just want to

mention briefly were the validation technique for Design Patterns developed by Benjamin Tyler from Ohio State University et al. and the concept of how to engineer survivable security biometric systems established by Roy Sterrit et al from Northern Ireland.

Formal Methods

In this year's workshop, a trend toward formal methods was clearly noticeable. Many of the papers incorporated at least some formal methods aspects. One reason for the unusually high numbers of papers on formal methods is probably the fact that SEW-30 was collocated with three other workshops with close connections to formal methods as part of the Systems and Software week, see http://www.systemsandsoftwareweek.org.

The industry, however, does not seem to adopt these concepts in a broad manner. Although, some speakers reported successful applications of formal methods, many others have not developed sufficient trust in those techniques and their return of investment.

Manfred Broy from the Technical University of Munich in Germany made clear, however, that formal techniques are needed especially in the automotive domain. He talked about the key properties of automotive systems, such as real-time behavior and what challenges arise from these requirements. While working with automotive systems, he realized that people tend to forget about the specific attributes of automotive applications, such as, what he calls, feature interactions. Feature interaction refers to the interactions of different devices in a car, which requires appropriate interfaces. The communication between these devices in the safety critical automotive domain must be highly reliable. Formal methods could be an appropriate technique to ensure this level of reliability.

Development of automotive systems necessitates a requirements engineering approach that takes into account the concurrent and distributed behavior of that type of software and that provides quality insurance having in mind that they must be dependable during their entire long life time.

He said that the industry asks for formal methods that enable them to verify automotive systems. However, existing modeling languages are not sufficient to model the automation of these systems and there is need for new modeling techniques. A modeling technique that meets the requirements of the automotive domain is currently under development by Manfred.

The need and the opportunity for solving problems with formal methods was also recognized by a number of other researchers of which we will mention a few.

A first step towards higher acceptance of formal methods could be the work of Yves Ledru et al. They developed an approach to transform a formal specification to a graphical notation that can be used to illustrate specifications. Yves is a professor at the University Joseph Fourier in France. His work attempts to close the gap between formal methods and stakeholders, who often have difficulties understanding the abstract notation of formal description languages. He particularly focuses on the B method, which is a formal approach that covers the entire software development life cycle. His technique produces UML diagrams from existing B descriptions, which can be used to illustrate the specification to stakeholders.

Frantisek Plasil is a professor at the Charles University in Prague. He presented a model checking approach that evaluates the behavior of software and its conformance to a high-level specification defined using behavior protocols. The main concept is to combine the Java PathFinder model checker developed by NASA Ames Research Center and the Protocol Checker, which was developed at the Charles University. The Java PathFinder is used to parse the source code. The parsing results are sent to the Protocol Checker, which evaluates the conformance of that data to the specified behavior. The results show that the model checking approach can be applied in a reasonable time. It does, however, require modifications to the Java PathFinder, which lead to some considerable drawbacks on the performance side.

The popularity of the Java PathFinder among researchers motivated Luigi Rigo from the University of Milan in Italy to transform the tool into a plug-in that can be integrated into the Eclipse developing environment. This is a crucial step in making such tools more attractive for software developers by integrating them with common development environments and allowing developers to easily apply them with little effort.

Model checking is also an essential part of the testing technique presented by Rick Kuhn from the National Institute of Standards and Technology (NIST) in Maryland. Rick et al. describe an approach for automated specification-based testing that integrates combinatorial testing with model checking. Algorithms that emerged from recent advances in Software Engineering are used to efficiently generate covering arrays. Model checking was then applied in order to generate useful combinatorial tests. A valuable outcome of this research is the description of the most efficient way to use Model Checking for this purpose. The results of his research suggest that the approach can be used to effectively test software systems. Rick also pointed out that due to the large number of test cases that has to be generated even for a relative small software system, the approach would probably not be practicable for large systems.

A successful application of formal methods was presented by Jinli Zu from the Nokia Research Center in Finland. Jinli described how the Nokia Research Center investigated using formal methods to solve reliability issues. This might serve as an example for situations in which companies reach for formal verification in order to meet certain goals.

Their need was to incorporate the evaluation of behavioral properties, such as performance and reliability, into the early phases of system design. Although behavioral properties are essential to the quality of software, they have not been part of the system design at their facilities and with conventional architecture analysis methods they found that it was not possible to evaluate the behavioral properties.

To address this issue, the research center considered the use of formal methods and found that Colored Petri Nets could help to achieve their goal of incorporating behavioral evaluation into the design phase. They used the Colored Petri Nets to create software architecture-level behavior models and applied the technique in a case study to a large telecommunication system. The results suggest that the technique can indeed be used to quickly evaluate new architectural solutions at an early design stage with respect to reliability and performance.

The researchers of the Nokia Research Center have not only shown that techniques based on formal methods can be successfully applied in practice but their work also indicates that there is a wide range of issues where formal approaches can help to solve problems.

Verified Software Grand Challenge

Keynote speaker Jim Woodstock from the University of York, UK, took the role of formal methods to yet another level. He stated that methods for formal verification of correctness could be used to give warranties for software. This is something that nowadays seems impossible considering the high number of bugs and the lack of reliability in current software products.

He claimed that formal methods are practical to use and are also the cheapest way to ensure the functional correctness of software. For formal methods to be successful, Jim added, it is crucial that developers and software engineers are willing to change their habits and adopt the new techniques.

Jim forecasted that in 5 years, the research community will have developed a foundation for work through development of mature tools and standards based on formal methods. In 20 years he says, "we want to have well developed theory and a powerful suite of tools."

Conclusion

The workshop was a successful gathering of people from industry, academia and research, where many new ideas were presented and research opportunities were pointed out. The speakers of the workshop openly discussed not only their successes but also some drawbacks of the approaches they reported on so that others can learn from them and use the knowledge for their own work. Much of the workshop was also about the future of software engineering, the needs and the opportunities. For example, David Atkinson from the Jet Propulsion Laboratory/Caltech presented NASA's road map for the following years and pointed out challenges of software used in space shuttles, robots and on the ground. It seems like interest in formal methods is growing. However, the currently available formal techniques and tools have not yet been able to fully convince industry because they are still too difficult to use compared to the perceived value. However, an event like the software engineering workshop gives the opportunity to align research with the needs of practitioners in order to make formal methods more useful in practice.

"IT Responsiveness" Fastest-Growing CEO Priority

Acapulco, Mexico – The top priorities of CEOs, according to an IDC survey reported on by Eric A. Prothero, a keynote speaker at AMCIS (Americas Conference on Information Systems) here August 5, are:

1. Customer care
2. Product innovation
3. Sales Productivity
4. IT responsiveness
5. Business monitoring

The interesting thing about those findings, from an IT point of view, is that "customer care" is relatively flat over time, but the concern for "IT responsiveness" is booming. 72% of CEOs identified IT as the key to corporate innovation. And, in what Prothero called a "scary" statistic, nearly all CEOs and CFOs said that their enterprise should adopt an "aggressive" stance toward IT, while only half of the CIOs - the ones responsible for IT – agreed. (Prothero speculated that it was because CIOs were already too busy).

Prothero, VP for Latin America for IDC, chose as the title of his keynote "Changes in the Use and Consumption of IT Around the World." He accompanied his talk with lots of gee-whiz statistics:

- worldwide IT spending is engaged in "huge growth," reaching over one trillion dollars (1000 billions) in 2005
- the rise in IT spending, after a fall over the dot-com bust, is now steeper than ever
- 23% of business investment world wide is now for IT
- less than 50% of the devices connected to networks are PCs

Prothero also provided some interesting demographic data:

- 51% of the population in emerging nations is under 20 years old (that figure is 21% in developed countries)
- the "huge" college population in China+India+Russia+Mexico+Brazil is more than 25% higher than US+Europe+Japan
- Estonia is the leading country in electronic penetration, with (for example) 108% of the people having cell phones

Prothero ended his talk with a look toward IT's future. He sees lots of growth in a field called "crowd sourcing," where communities form around growing certain kinds of content (he talked about several examples, including a "seekers/solvers" community that either posts problems to be solved, or works to find solutions to posted problems, and a photographic community that posts photos for others to download, where the posters get a royalty).

But Prothero identified three specific trends for the future:

1. A transition to "dynamic IT," where systems are thoroughly integrated (this is a vendor-driven trend, he said, with such vendor-chosen names as "on demand," "web services," and "service-oriented architecture."
2. The "long tail" phenomenon (the name is taken from a book title, and refers to the typical usage curve where there is a spike of high usage organizations followed by a long tail of lesser users). The trend is about serving huge numbers of lesser users. Prothero sees such companies as Google and eBay successfully pursuing the long tail.
3. A shift in the innovation balance from large companies to small, independent sources (Prothero derived this trend from the book Open Innovation). The trend is toward building what Prothero called "innovation communities."

Prothero's talk, late in the second day of the AMCIS conference, was lightly attended. Apparently most conference attendees chose to pursue Acapulco's fine beaches instead!

Speaker Advocates His Own Version Of "Out With the Old," "In With the New"

Brisbane, Australia - Australian Dave Thomas, a researcher at Object Mentor, speaking at a Queensland University of Technology industry seminar here August 29, found lots of fault with existing software approaches, and proposed a very different approach, in his talk "Radical Thoughts on the Future of Programming, 2010-2020."

Thomas spared almost none of today's technologies from criticism:

- "open source allows everyone to have their own copy of the application" (he went on to say that he learned why that is a mistake, and learned the value of closed source, when he kept making homegrown changes to a system until it crashed, and he couldn't recover a working version)
- Java and C++ are creating a "legacy mess"
- "the bug rates for Java libraries don't go down", OMG and UML are for "losers"

But he saved his most scathing criticism for object-orientation:

- "objects are too hard for normal people"
- "object-think is artificial for many computations"
- "objects are too cumbersome and slow for current machines to execute them efficiently"

He also lit into some of the social infrastructure surrounding the programming field:

- "certification is for the incompetent"
- "CS graduates are getting dumber and dumber"
- "many CS algorithms are wrong" (they ignore cache, for example)

What DOES Thomas believe in? Model-driven development, specifically when it's application-domain focused. He sees models being developed in what he calls VHLLs (Very High Level Languages) focused on domain-oriented programming (DOP). VHLLs, Thomas says, must have simple syntax, clear (although complex) semantics, and "support mapping domain abstractions." He went on to identify several current such DOP-focused languages, while noting that few of them are very well known. He had a special role for the Agile approaches in this regard, noting that the "Ultimate Pair" programming should consist of a domain expert and an expert developer.

Thomas, who performs research toward the goals he identified in his talk, identified these DOP challenges/opportunities for VHLLs:

- language design – they must be expressive, readable, writeable, and strong on specifics, providing uniform access to data
- fusion – they must support multiple domains/paradigms
- implementations – they must provide for direct execution, and hardware software co-design of solutions.

Blink: The Power of Thinking Without Thinking

by Malcolm Gladwell
Published by Little, Brown, and Co., 2005

Review by Robert L. Glass

Which is better, a solution arrived at by deep thought, or one arrived at in the blink of an eye? This book takes the position that those blink solutions may be just as good.

What's more, it's convincing at it. Via anecdotes and research results, Blink spins a neat web of belief.

By the time I'd read only a few pages, I found myself believing in the power of what the author variously calls "fast and frugal" decision-making, using the "adaptive unconscious," "rapid cognition," "snap judgment," and "thin slicing." Personally, I think what the author is talking about here could be called "intuitive thinking," but for some reason he never uses that term.

This book is in the genre of "one trick pony" books (books that develop a single narrow theme and then elaborate on it for a couple of hundred pages). That's not necessarily bad – Gerry Weinberg's eventual best-seller The Psychology of Computer Programming was labeled a one trick pony book by the publishing houses that made the mistake of not publishing it!

The author says he has set three tasks for himself in writing the book – (1) to convince his readers that quick decisions can be every bit as good as those made cautiously (I'd give him an A+ for accomplishing that); (2) to teach us when to trust this thinking process (I think the author neglected this task, and I'd give him only a C); and (3) to convince the reader that quick decision-making can be educated and controlled (if examples were sufficient to accomplish this task, then the author did just fine. But given that he doesn't say much about how to "educate and control" the process, I'd give him only a B here).

The author isn't entirely a blink fanatic. He does, for example, say that "spontaneity isn't random," noting that effective spontaneity depends on appropriate preparation.

Is this a book that's worth reading? Yes. Treat it as a lightweight, enjoyable read, with some mildly profound implications. And then consider how to take that one trick pony to heart!

The Insider's Guide to Outsourcing Risks and Rewards

by Johann Rost
Published by Auerbach, 2006

Review by Robert L. Glass

There's a lot published on the subject of computing outsourcing. A lot of it is hype – hand-wringing scenarios about how good or bad it's going to be. A lot of it is academic – material written by people who have never done any outsourcing, but who have read enough theory on the subject that they sound – at first hearing – like experts.

This book is different. First of all, it's objective. The author tells it like it is, not like some people wish it were. Secondly, it's "been-there, done-that." The author's material has a fresh, believable tone, derived from his real outsourcing experiences.

The book is also different in another way. Much outsourcing talk, of course, is about the Asian countries, like India and China. There's some of that in this book. But an interesting fraction of this book's content is about eastern European outsourcing. The author is a German software practitioner serving as a visiting professor of information systems in Romania, and there's a lot of that background in what he writes.

What's good about the book? Its anecdotes – the author illustrates many of his points with stories about real outsourcing experiences. Its content – there's material on myths vs. realities, on particularly frequent challenges … and even sections on such far-out topics as "industrial espionage" and "business continuity in case of war."

What's bad about it? The net effect of some of the anecdotes is chilling, since they report on incidents most prospective outsourcers would rather not ever have to think about. The index, which is so short that almost none of the book's key topics are reported there.

But on balance, this is a very good book. If you're considering any form of outsourcing, especially offshore outsourcing, it's hard to imagine tackling it without having read this book first.

The Seasprite Puzzle

Robert L. Glass

Here's a puzzle for you. I'm going to describe an embedded software project, and your part in solving the puzzle is to decide whether the project's problems are due to software or something else.

The application is the Royal Australian Navy Seasprite helicopter project. The Seasprite is/was a 1960s era aircraft, and the software portion of the project had to do with modernizing the capabilities of that ancient chopper. The contract to do the work was let around 1996, and the sole customer for the upgrade work was the Royal Australian Navy (the Navy calls it a "one-off" development). The cost of 11 updated Seasprites was a (fixed-price) $660M, and the chopper's builder, Kaman Aerospace International, a U.S. firm, was to do the work. Initial deliveries were to have happened in late 2000, with all work completed by mid-2001.

Litton Systems was subcontracted to do the software portion of the upgrade, but it ran into trouble fairly early and the work was transferred to CSC Australia and Northrop Grumman. As is typical of such Australian projects, $350M of the software spend was to be performed by Australian companies.

That was then, this is now. What's been happening lately?

At this point, only 9 of the choppers has been "provisionally" accepted, and none of them have been unconditionally accepted. According to Australian Defence sources, a stability problem has developed when flying the chopper under extreme instrument-flying conditions. The problem is severe enough that the aircraft has been grounded from operational flying. The government is considering "all options," and "legal action" has been mentioned as one of them. It looks likely that Defence is going to give up on ever flying the aircraft operationally.

Defence people are saying things like "in retrospect, it looks like the specs exceeded what was practical" and "it was probably a mistake to pursue ambitious Australian requirements rather than buying off the shelf." But they are also calling this a "software failure."

So there's the puzzle. Does this sound like a "software failure" to you? Or some kind of interrelated complex embedded system problem that simply was too big to be tackled by any discipline, software or otherwise?

Information Source – "Nelson Faces a Systems Failure," the Australian, May 16, 2006; Patrick Walters

