



**Presents**  
**An IT Metrics and Productivity Journal Special Edition**

**Focus on Tom Love, Ph.D. and CEO of ShouldersCorp**  
**A CAI State of the Practice Interview**  
**August, 2005**

**Biography of Tom Love**

Tom Love is the Co-Founder and CEO of ShouldersCorp. Prior to founding ShouldersCorp, he was CEO of WorldStreet Corporation, Managing Director of Morgan Stanley and Vice President of IBM Consulting Group (now Global Services). In 1983 he co-founded Stepstone Corporation, the first Object Oriented software products company (Objective-C; Software-ICs).

Tom has contributed significantly to the commercial evolution of the software industry. Before becoming an entrepreneur, Tom Love held technical and director-level positions with General Electric, ITT and Schlumberger.

Tom is the author or co-author of over 60 journal articles, book chapter and technical reports, as well as the author of *Object Lessons: Lessons Learned from Commercial Object-Oriented Application Development*, published by Cambridge University Press.

Tom Love has a Ph.D. in Cognitive Science from the University of Washington where he studied the characteristics of successful computer programmers. He received his BS in psychology and mathematics from the University of Alabama.

Tom's new book *Software Pilots* will be available in December, 2005.



**CAI: One of the most enjoyable things about your writing is your ability to string together colorful metaphors to help people better understand software best practices issues, an area that is typically written about in very dry terms. For instance, in a single article, *Extreme Software Engineering*, you make analogies to carpentry, aviation, fiction writing, motorcycle construction, the linkage of the British Isles and France by rail, and the building of the Empire State Building. In light of this, how might you make use of the "Manufacturing Revolution" as a metaphor for understanding where we are in IT today?**

**TOM LOVE:** As far as I can recall, leaders in the software engineering field, for more than three decades now, have been looking for ways to increase the productivity of

programmers and the predictability of software engineering projects. For example, Capers Jones and I were the first two members of a several hundred person group put together at ITT back in 1979 to examine why so many software projects fail. And this group effectively created the spin-offs known today as SEI, in Pennsylvania, and as MCC, in Texas.

What came out of these two organizations was a lot of discussion in the 1980s about Japanese software factories, about component or object oriented software development in the late 1980s, and about frameworks and patterns in the 1990s. Throughout all of this, much was written about what we can learn from manufacturing in order to improve software engineering.

Companies seem to be entering a new phase now which I call the "don't even try" phase of software engineering; more specifically: "We always fail so we shouldn't even think about custom software development." I should mention though that this is happening at the same time that my own company, ShouldersCorp, has just managed to complete 20 successive projects on schedule, on spec, and on budget. So in fact, it can be done. It can be done predictably and it can be done using statistical process control just like we do in manufacturing. But it can't be done by amateurs, anymore than F22 raptors can be flown by student pilots.

**CAI: If the manufacturing revolution allowed the average manufacturing company to increase productivity by 400%, how might you quantify the opportunities facing IT?**

**TOM LOVE:** Much larger than 400%. With manufacturing you are dealing with physical things. In the case of software, you don't deal with the same physical constraints. So I think the opportunity for creating 1000% increases in software productivity is clearly present. It's sometimes even realized, just not very often.

**CAI: Given all of the advances in methodologies and CMM tools we still have data pointing to the fact that IT software productivity has remained flat for the past 10 years. Do you have any thoughts about why that may be and why we've made such little progress?**

**TOM LOVE:** I don't agree with your premise. I think we have made major strides in productivity, simply through object oriented design and through the reuse of various frameworks across projects. In the 1980s, for example, the projects I ran would typically start off with a UNIX operating system and a set of functions associated with that and probably also a database management system. Everything else would need to be built from scratch. But when I start a project today I begin with enormous GUI libraries, application servers, web servers, workflow engines, screen generators, testing tools, persistence tools, etc. And so if you count all of the code that doesn't have to be

written, we are actually much more productive than we used to be. On the other hand, if what you are measuring is how long it takes to write a lot of code in itself, I think you will find that this metric has remained more or less constant over the years.

**CAI: For those IT organizations that have been successful in the effort to create systematic long term value and cost savings, what is it that they've been doing right?**

**TOM LOVE:** First of all, I think that the companies that are winning right now are the ones that innovate with a lot of knowledge about what's been tried before. If you think of Edison, he tried lots of different materials for his light bulb until he found one that worked, thousands if I remember correctly, and he didn't just take the same material and use it over and over again without keeping track of what he was doing. He kept very meticulous notes in order to figure out exactly where he should be going with his next round of improvements. Similarly, winning organizations must be able to learn from their history and from the history of the industry. And to do that, they must be able to maintain stable teams. So workforce stability is a critical success factor, and it will always be directly tied to continuity of knowledge and to innovation.

The second secret to being successful in the software business is to understand the science behind the engineering. Back in 1903, the Wright Brothers had to figure out how a propeller worked. Their key insight was that a propeller was a spinning ring whose list created thrust. Once they figured this out, all they had to do was engineer a couple of propellers. And because they understood the science behind what they were doing, the very first propeller they developed, one that was hand carved from a piece of spruce, turned out to be 80% efficient. Keep in mind that the most efficient propellers today, propellers designed using sophisticated computer simulation, are only 83% efficient. My point is that the Wright Brothers were successful because they understood the science behind the engineering. This holds especially true in the software business.

A final success factor involves understanding people. Few organizations that I run into seem to fully understand that two people with the same background and experience on their resumes could have a 26-1 difference in productivity, and that's actually based upon experimental studies from the 1970s. I actually think, and most people who are still doing scientific studies on this think, that the spread is presently around 100-1. That's on account of the effect of reusable software.

**CAI: What are the 5 things that you would do to improve processes and productivity at a CMM Level 1 organization within year one?**

**TOM LOVE:** After several decades of being in the consulting business, I can tell you that I see a lot more organizations at Level 1 than at Level 5. Consequently, most of

the recommendations I have made over the years have been with companies at the Level 1 stage of maturity.

My first recommendation would be to buy the best and build the rest. Don't write code that you can purchase. The best way to increase programmer productivity is to eliminate the job.

My second recommendation would be to get a real architect and to take their advice. If you've got a team of people that are building code randomly it is not going to be effective. Nevertheless, a remarkably large number of organizations actually seem to be operating like this.

My third recommendation is to have everyone read ShouldersCorp's four word development handbook- "*Make More Interesting Mistakes*"- and then to record their lessons learned. A lot of people have development handbooks that are sitting on their shelves that still have the plastic wrap on them. Part of the reason behind this is that most handbooks are written under the assumption that everybody on the project needs to know everything, and this is something which is conspicuously not true. Moreover, nobody ever has the time to read that much material anyway. In contrast, if a manager or developer could have access to a simple set of web pages that documents the lessons learned from other managers or developers on similar projects, that would actually be really interesting and relevant. People would actually take the time to read it.

My fourth recommendation is to use the smallest qualified team to get the job done. Two is optimal. It becomes less optimal as the team grows from there. Amazingly, very large projects have actually been completed using two person development teams, and in first iteration. There is a chapter in my book *Object Lessons* that's all about two person development projects.

My fifth and final recommendation is to manage using good estimation tools and real time project data. In my book *Software Pilots*, I refer to the latter as "project panels." If you don't have a good estimation tool, it's going to be hard to figure out how to build a project panel. Organizations need both.

**CAI: What would you realistically expect to get out of the effort you just outlined and how would you define that value in business terms?**

**TOM LOVE:** First of all, when I use the term "project panel," that effectively implies that you've got at least part of the organization operating at a higher CMM Level already. That's because you have figured out by this point how to quantify what you're doing, how to make corresponding improvements, and how to get the information in real time. It also means you've done enough training of project managers so that they know what they are looking at and what to do based upon what they see.

What kind of impact could this have on an organization? The potential impact is huge. What if you could start doing development projects in four months as opposed to two

or three years? What if you could do so many successive 100 day development projects that when the organization said it was going to be done in 100 days people started expecting it to be done in 100 days? This kind of thing can have amazing effects on an organization. In contrast, if people think you are giving one date while actually meaning another, they are much more likely to put a lot of pressure on you to continuously make changes. And that makes everybody's job harder. Schedules get missed and then everything starts to slip out of control.

**CAI: In your book *Object Lessons*, you derive two basic software axioms out of a story about an extravagantly expensive rehabilitation, in modern times, of a Swedish warship that sank to the bottom of Stockholm harbor in 1628; specifically, that the systems we build may last far longer than we ever imagine and that maintenance costs often exceed the original development costs, even for unsuccessful projects. Despite the fact that this is a well known truth within the world of software, we still see all of the best thinking and publishing in our field- about metrics, about estimation, and about processes- being done in the area of new development as opposed to maintenance. Why is that the case given that so much more of the money is being spent on maintenance? Do you agree that this is an area of opportunity?**

**TOM LOVE:** Here's a place where we can really learn from both the automobile and the helicopter industry. You can make a lot more money, for instance, selling a car or even a helicopter one part at a time than by selling it fully assembled and tested. Similarly, even small software systems can wind up costing a lot to maintain. That's because software lasts so long and is so hard to change.

I think that the fundamental way to reduce maintenance costs is to reduce the amount of code you have to maintain. The metric I've always used is that a single programmer can maintain half a box of printed code. That turns out to be about 50,000 lines of code. That means that one million lines of code will take twenty people to maintain, assuming that these twenty people are actually doing a good job. Of course, you have software coming out of Microsoft these days that consists of 65 million lines of code and you've got numerous commercial organizations with products amounting to 10 or 20 or 30 million lines.

Nevertheless, the sad fact is that most CIOs don't even know how many applications they are managing, yet alone the number of lines of code in them. I think any sensible maintenance improvement plan has to begin with creating an inventory that measures things. What you should measure is not only the bulk of the code that you have to maintain and support but also the technical variety. Once you have measured these things, you can then begin a bulk reduction and variety reduction campaign. This can be very effective.

I once took over an organization of 225 developers and actually discovered that they were using 32 separate procedural programming languages. My first question to them was why we couldn't get by with 16. It is incredibly expensive to try to maintain software written in 32 or even 16 programming languages. The problem is that once

people get freed up from their projects they lack the programming skills for moving on to the next project. This causes the complexity of running the organization to go up exponentially.

**CAI: From a process and productivity improvement perspective, how would you address the maintenance issue? What questions should we be asking ourselves in order to re-apply development best practices over to the world of maintenance? What specific metrics would you advise organizations to track if they were focusing primarily on maintenance as opposed to new development?**

**TOM LOVE:** I would focus on bulk reduction and variety reduction, just as I described in your last question. The nice thing about this is that it requires first that you count and second that you measure your progress. I would also take a look at procedural variety. Procedural variety is how many different approaches there are in the organization to doing development projects. This is another kind of technical variety that needs to be managed.

Nevertheless, I don't believe that everybody should be doing every project exactly the same way. I think that's unrealistic. You don't learn from that. I would advise people to go back to the "*Make More Interesting Mistakes*" model. In other words, it's OK to try something new as long as it has not been tried 13 times before and failed every time.

**CAI: In your new book, *Software Pilots*, you discuss the importance in aviation of multiple controls, multiple gauges, visual and auditory and kinesthetic feedback and you then compare this to the importance in the software world of real time management systems for collecting data, guiding processes, and providing visibility. To quote your book, "Software pilots need a simple instrument panel to control their projects. It's madness to attempt a flight without basic information and sufficient control mechanisms." Could you discuss with us what, in your opinion, a successful software pilot's control panel would look like and why?**

**TOM LOVE:** All project panels are not created equal. Nevertheless, I would still expect them to share some common features, just like panels in airplanes share common features. Airplanes, for instance, need fuel gauges. Likewise, software projects need "remaining budget" gauges. Airplanes need avionics to figure out where they are going and to determine if they are on the right course. Software projects have the same need, the only difference being that we have no agreed upon system of navigation for this.

In my experience of the past 25 years I've found that the unit of work represented by a class in an object oriented program provides an unusually good and reliable way of

measuring what needs to be done and where you are in your process. Consequently, at ShouldersCorp we construct three dimensional models of the systems we are developing using Styrofoam balls to reflect the overall structure of the classes. The balls represent the classes themselves and the diameters of the balls represent the complexity of the classes. All of this is labeled. We actually hang the whole thing from the ceiling where as many people as possible can see it and then we apply the sophisticated case tool of colored ribbon to keep track of the current development status of each one of the classes. In this manner, we can quickly determine whether a class has been designed, coded, tested, or documented since each phase is represented by a different color ribbon.

This may seem amazingly simple but it represents a major change. One of the things that happens is that everybody has to agree that we are building these specific classes and not some other set of classes. Additionally, when you come in on Monday morning and discover that the structure is tilted 30 degrees, it becomes apparent that something major has changed over the weekend.

We've used this method for every project we've done over the past 8 years and we wouldn't do without it. Interestingly, customers ask for things like this, so we also have a virtual reality version that you can look at it on display screens. But we still prefer the Styrofoam balls.

**CAI: So much has been made of the cost differentials between India and the U.S. Ultimately, and over the long run, do you believe that the trend towards India will lead the software best practices movement backwards or forwards?**

**TOM LOVE:** I've often been asked questions about offshore outsourcing. And I always apply the rule that if an organization can not manage development projects that occur in a single room they shouldn't even think about trying to do it from 10,000 miles away. On the other hand, if the job is very well defined, or if the process is extremely well defined, offshore outsourcing can work, and it can work very efficiently. But what I run into a lot is this naïve notion that by lowering the hourly cost we are going to reduce our overall cost and I find that to be simply delusional. I actually knew a CIO who told me that although he knew it was going to take longer and cost more to offshore the job he wanted to do it anyway to reduce the hourly costs. We should create our own version of the Darwin awards for such people.

I'm not at all opposed to offshore outsourcing. I'm actually on the board of directors of a company that does all of its development work in St. Petersburg, Russia. This has worked out spectacularly well because they've got a process in place that is sufficient to get the job done. But the key here is that you have to have that process in place. If you have an organization that doesn't have process and can't figure out how to build things or specify what they need or make decisions then it's just not going to work.

Also, your probability of success will go up dramatically if you have a real time project panel that is feeding you real time objective information about the project. In contrast, if you are waiting for a written monthly report to find out what is going on you are

going to have troubles. There are going to be a lot of misunderstandings about what has occurred in the past month and the time to correct any misconceptions will simply be too great. Just think of it in terms of aviation- what would happen if you turned the yoke of your airplane and the rudder responded in 30 days?

This is the reason why I prefer to do development projects with everybody sitting in the same room. If we are all in the same room, I can overhear two people in the corner just as they are about to head off in some misconceived direction that will waste us a week or two weeks or even a month's worth of effort and I can correct it in 15 minutes. However, with the work being done 10,000 miles away, you might not detect the mistake for several months.

Questions? Suggestions? Comments? Please contact the IT Metrics and Productivity Journal Editor at **[michael\\_milutis@compaid.com](mailto:michael_milutis@compaid.com)**