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

# Focus on Dr. Norman Fenton
# A CAI State of the Practice Interview
# July, 2006

## Biography of Dr. Norman Fenton

Norman Fenton is Professor of Computing at Queen Mary (London University) and is also Chief Executive Officer of Agena, a company that specializes in risk management for critical systems. Between 1989 and March 2000 he was Professor of Computing Science at the Centre for Software Reliability, City University. Norman is a Chartered Engineer (Member of the IEE and Fellow of the BCS) and a Chartered Mathematician (Fellow of the IMA). His previous academic posts were at University College Dublin, Oxford University and South Bank University where he was Director of the Centre for Systems and Software Engineering. He has been project manager and principal researcher in many major collaborative projects in the areas of: software metrics; formal methods; empirical software engineering; software standards, and safety critical systems. His recent research projects, however, have focused on the use of Bayesian Nets (BNs) for risk assessment. Norman's books and publications on software metrics and formal methods are widely known in the software engineering community. Our interview between Michael Milutis, Executive Director of the IT Metrics and Productivity Institute, and Dr. Norman Fenton took place in March of 2006.

◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆

**CAI: Could you tell us a little bit about yourself, the path your career has taken and what you are working on today?**

**FENTON:** I am CEO of Agena, a company that has built risk assessment software since

Computer Aid, Inc.
1390 Ridgeview Drive
Allentown, PA 18104
(610) 530-5000

1998. I am also Professor of Computing Science at Queen Mary (University of London). Before coming to Queen Mary in 2000, I was a professor in the Centre for Software Reliability at City University.

For many years my research revolved around the evaluation of critical software. By "critical software" I mean the kind of software found in aircraft control and nuclear protection systems. My work with critical software was focused specifically on how to measure the reliability of such systems. Although critical software obviously has very high reliability requirements, the state of the art at the time for reliability measurement using testing data alone was not enough to adequately meet these needs. Consequently, we had to combine other process factors with the testing data, factors such as the quality of the development process, the quality experience of staff, defect data, etc.

Trying to overcome the problem of reliability measurement in critical software systems is what led me to my work of the past five years: research and development on causal modeling using Bayesian nets. This type of modeling enables us to get more accurate predictions. For example, when you try to predict software reliability based on testing data alone, you often get predictions that are too conservative. However, if you take process, quality, and experience into account, your predictions will become demonstrably more accurate. Causal modeling enables you to combine different factors and to measure the uncertainty inherent in these factors.

I cannot stress enough how important I think this is for the whole of software metrics. For many years I was involved in a lot of traditional IT metrics initiatives. I did much work, for instance, with the telecom companies, where there was a great deal of attention paid to defect metrics. We were continually discovering that the defect numbers in testing were never indicative of the defects likely to be seen in operation. Nevertheless, the accepted metrics practice at that point in time was to view defect counts in testing almost as a surrogate for operational quality. In other words, it was popularly believed at the time that the modules which were most defect prone in testing would be the ones most defect prone in operation. Consequently, if you wanted to focus on your maintenance efforts, you would first identify where you had the most defects in testing.

However, if you really looked at the data, you would have come to a different conclusion. In our case, we discovered that the modules which were the most defect prone in testing actually tended to have a very low operational defect rate. Was that counter-intuitive? No. It could in fact be explained by causal, experiential factors. Specifically, the modules with the most defects in testing were the modules that had received the most attention up front. Consequently, when it came to operation, they worked better.

Another example of this is a story that was related to me about a major IT company. During the set-up of their first company-wide software metrics program the company reviewed the initial metrics from their software projects to determine what they could learn. In so doing they discovered a small number of projects that had reported zero operational defects. At first, the metrics team assumed that these systems were the highest quality systems. This was in line with the reigning logic at the time that the defects found in operation were directly correlated to quality delivered, i.e. that less defects meant higher quality and vice versa. Upon further inspection, however, it turned out that these systems were in fact the worst quality systems. They were so bad that they had simply never gotten used after delivery, which is why no operational defects had ever been reported. Nevertheless, a very simple metric had indicated to the metrics team the exact opposite conclusion. The point to this story is that if the company had taken a simple experiential factor into account – in this case, operational usage - everything would have been crystal clear.

My current work at Agena is focused on this same issue – how to incorporate experiential factors into causal models. Agena builds systems that help companies better conduct reliability prediction and risk analysis. Because many different industries need risk assessment applications, our clients are not just software companies. We work with air traffic companies, railway companies, and electronics companies, as well. We have also packaged our offerings into a more general purpose risk modeling tool, which provides many different models to help our customers get started.

**CAI: How widespread is the usage of causal modeling in the software**

**industry?**

**FENTON:** It is relatively new, but very quickly becoming more visible. We have thousands of users of our tools. There are also many companies at the critical end of software development, such as nuclear power plants and aircraft control system creators, that are reliably using or evaluating the technology.

**CAI: How would you define Bayesian nets for people who are not familiar with this term?**

**FENTON:** A Bayesian net is just a graphical model combined with probabilities. The graphical model represents all uncertain variables as nodes, and these nodes are joined together when there is a causal or inferential relationship between two variables. Associated with each node is a probability table which expresses your beliefs about a variable's impact on any dependent nodes. When observations or actual data are entered, the model applies Bayesian analysis, which is the classic means of updating beliefs in probabilistic terms, given new information. When data are entered, the model will also update information about all of the variables you do not know about. At any time there will be known variables, for which you enter observations, and also uncertain variables. The model will tell you as rationally as possible what the true uncertainty is of any of these variables.

**CAI: You've written a book- *Software Metrics: A Rigorous and Practical Approach*. What was your motivation for writing this book? What did you set out to accomplish with it?**

**FENTON:** My simple objective with this book was to transform software metrics into as rigorous a measurement activity as the kind of measurement found in other engineering disciplines. That meant saying that it was possible to apply the Theory of Measurement to software metrics. The Theory of Measurement explains what it means to define a measure and also how to validate a measure in different contexts.

To give a simple example from the book, lines of code is the most commonly used and

easily evaluated software metric. Although it is certainly a valid measure of program length, it is hardly a valid measure of program complexity. Consequently, if you use lines of code as a surrogate metric for program complexity, you need to be aware of the pitfalls of doing so. The book explores these pitfalls.

The book also explains the importance of measurement in general while outlining some fairly practical measurement guidelines. Moreover, having established a theoretical basis in the first edition from 1991, the subsequent editions of 1996 and 1998 focus much more on empirical results and provide a great deal of actual benchmark data. In these later editions I wanted to help people understand what was known in our industry about productivity rates and defect rates and to try to be as rigorous as possible.

**CAI: Why is there such controversy in our industry over the question of lines of code versus function points? Which is the better approach to software measurement?**

**FENTON:** Lines of code is a very simple metric which is easy to obtain and can reveal many things. Even if you don't have a tool to measure it, you can easily run a macro or a two line program to obtain your lines of code. In that simple program you can learn some interesting things. For example, you can make a differentiation between comment lines and non-comment lines and thereby gain a very simple measure of how well your program is documented within the code. You might also distinguish between new and re-used code, and perhaps use these results as a normalizing metric for productivity and quality.

Unfortunately, people often try to use lines of code for inappropriate things. One example of this might be the use of person-months divided by lines of code to measure productivity. The rationale behind this particular calculation is that lines of code is a measure of output, and that it also incorporates within it, in some sense, utility and functionality. However, when you think about it, lines of code is not an accurate measure of these things.

For all of the areas in which you would normally use lines of code as a normalizing factor for accurate predictions, function points are a better metric. Moreover, function

points do not suffer from the many problems that plague lines of code, such as language dependence. If there is a downside to function points, it can be found in the fact that they are incredibly difficult to do properly. Nevertheless, a lot of progress is being made on this front.

I am a very big fan of function points. In fact, in the research projects I mentioned earlier, we actually used function points as a key sizing metric in our models. However, when we tried to put the models into practice, the organizations came back to us and said, "We can use most of this, but we don't use function points." It turned out that the lines of code in these models were being used purely as an indicator of function points. People were actually putting lines of code into the models instead of function points. The models would then produce a probability distribution for how many function points there were, given the lines of code. It was an unsatisfactory method, to say the least, because if you were only going to be using lines of code, you would have been better off taking the function points out entirely. In this case, the function points were only serving to confuse things and were creating a lot of unnecessary uncertainty.

It is a big issue, to be sure, and no matter what arguments are made in favor of function points, people will ultimately keep turning to lines of code because they are simpler.

**CAI: Could you quantify for us the benefits of having a solid software metrics program? What, for example, would be your characterization- in aggregate- of the resulting improvements, in terms of ROI, for IT organizations that have effectively integrated successful metrics programs into their software operations?**

**FENTON:** We recently did some work for a major bank whose metrics program was focused on a very simple objective.

The bank had hundreds of new software projects being proposed each year. Each of these projects was potentially risky. In fact, the bank had calculated that 30 percent of all new projects ended up losing them money in some way, either through indirect costs, defects or loss of customers. Consequently, their simple objective for this new

metrics program was to be able to identify risky projects as early as possible.

With the help of Agena, the bank managed to develop a model that made use of some very simple metrics, all available at the start of each project, and these metrics enabled them to identify project risk early.

The ROI in this case was potentially in the order of thousands of percent. That's because it cost the bank very little to set up this program. It only required a few days work for their risk experts and project managers to identify the various relevant risk factors. Moreover, much of the data needed at the start of these projects was already in the database somewhere, so there was not much additional effort required beyond that. In fact, the total effort in this case was less than a few dozen man days. What that means is that even if the bank is only able to preemptively identify a very small portion of the 30 percent in total project failures each year, they will save millions of dollars. Keep in mind that each of these projects costs the bank upwards of $500,000.

The whole idea of using a metrics based approach for early risk assessment in IT projects is actually fairly new; at least in the sense that causal models are being used. One of the reasons this was not done until recently is that the technology simply wasn't there.

In terms of more traditional metrics programs, a lot of our work has been focused on defect predictions. One of the groups we work with develops software for consumer electronics devices. Accurate defect prediction is critical to their decision making about when to release their software. They need to know when their software has a sufficiently low defect state so that it can be embedded into the components. And they have to deal with real product recall issues if there are too many defects.

Before we came in, they had already invested a lot of effort in a metrics program which was collecting very detailed defect data throughout the software development process. This was not a case in which the client would be getting a thousand percent return on investment from their metrics program. They were happy to see very small improvements in their predicted accuracy. That's because in their case, the savings were being generated in the form of reduced costs (related to less fixing of mistakes). Keep in mind that if you can get an increase of a couple of percent in the accuracy in

your defect prediction rates, you might get a 20 percent decrease in the costs of fixing the metric.

Before we came in, they were getting what they thought were reasonably accurate predictions on the order of 80 percent. After working with them for a few years and incorporating causal factors into their method, they reached 95 percent predicted accuracy in their domestic environment. I am not suggesting that any company getting started with a metrics program is going to get this kind of predictive accuracy. In this particular case, the company actually had a metrics program that had been in place for many years. Moreover, a lot of the work that we did for them was in their software production company in Bangalore, which is a CMM Level 5 rated organization.

**CAI: There are many different metrics out there and many different consumers of metrics. How do organizations best determine which metrics will be most meaningful, both to technical as well as executive management? What sort of questions should they be asking themselves when selecting and analyzing such metrics?**

**FENTON:** At Agena, our focus has been two-fold. First, we figure out how to identify and monitor risky products. Putting metrics goals and objectives into quantifiable risk terms will get you powerful buy-in from senior management. Second, we focus on quality metrics.

Keep in mind that when it comes to quality, it is not good enough to look only at technical objectives such as decreased delivery defects. As in the case of risk, quality metrics need to be built around objectives that senior management can understand, such as lower maintenance costs and higher return.

When selecting and analyzing metrics, questions should revolve around who, what, where, when and how. This is covered in my book. Such questions help to ensure that the metrics identified are appropriate for an organization and that they can be implemented and managed at a reasonable cost.

Once these questions have been asked, I think it makes sense to conduct an expected

return on investment analysis. This can be a simple spreadsheet, one that evaluates the new costs required for the data collection.

How you decide to analyze your metrics depends on your objectives. For example, someone with classic process and quality improvement objectives might want to focus their analyses on issues such as the defect fix rate. In order to monitor this properly, however, data would need to be collected - not just about defect numbers - but also about the effort and time needed to fix the defects.

**CAI: Would you expect an organization's overall approach to metrics to be significantly different for maintenance as opposed to new development? If so, are there any specific metrics that you feel are critical or essential to organizations that are more focused on maintenance?**

**FENTON:** Finding and fixing defects – and the corresponding costs associated with this – is the crucial issue for maintenance.

Interestingly enough, one of the most challenging problems in maintenance involves the measurement of operational usage. For example, if Microsoft wants to determine the maintenance costs associated with Microsoft Office, they know exactly how many users they have and can consequently capture operational usage easily. However, if you are dealing with a software system that was developed as a one-off for a single client, you will only be able to determine defects when the user is using the system. Going back to the example mentioned earlier, organizations can go for very long periods of time and never see any defects being reported. But this doesn't tell you anything about maintenance, since you cannot know anything definitive about your maintenance costs unless you also know if there was operational usage during that same period. Knowing how and when the system is being used is crucial if you want to capture meaningful maintenance metrics.

**CAI: It is frequently cited that 80% of IT spending is directed towards the running and maintenance of existing systems and infrastructure, as opposed**

**to new development. Despite this, 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?**

**FENTON:** Very few researchers or developers take an interest in software maintenance. The developers want to develop new systems and the researchers want to develop new methods. All one needs to do is to look at the software engineering literature; the proportion of books that is dedicated to maintenance is very small. In fact, if you pick up any software engineering textbook, you will be hard pressed to find a single chapter devoted to software maintenance. I would be willing to bet that of all the words written in books about software engineering, less than 1 percent is about maintenance. The same is true for software journals.

Why is this the case? I believe that the entire academic system mitigates the amount of research that is being conducted on maintenance. It is simply not considered to be "sexy." I think this is as true in the US as it is in Europe.

**CAI: Process and metrics are fairly inter-dependent. What do organizations first need to have in place on the process side of things before they can expect to make any progress on their metrics efforts? What role, if any, do tools place in all of this?**

**FENTON:** A lot of what is essential in a metrics programs is already in place at most organizations. I will give you the best example of this. Every software development organization needs to report defects. To accomplish this, there are many simple technical options out there. One solution is Bugzilla, which is not just free but also easy to adapt. In fact, there is no excuse for any software development organization not to use a tool like this.

At a basic level, organizations should have an effort tracking tool and some kind of electronic planning tool. Additionally, their code development environment should be able to turn out product metrics, code metrics, and design metrics. Most of the

development environments these days have features such as this, and the output can be used as the basis for a lot of metrics.

If you want to do risk based analysis with the causal model, you need to use a tool such as AgenaRisk. However, for almost all other basic metrics, organizations already have all the tools they need in place.

Questions? Suggestions? Comments? Please contact the IT Metrics and Productivity Journal Editor at **michael_milutis@compaid.com**

Computer Aid, Inc.
1390 Ridgeview Drive
Allentown, PA 18104
(610) 530-5000