Up Close with Microsoft's Paul Maritz

CrossTalk had the opportunity to speak with **Paul Maritz** of Microsoft about commercial off-the-shelf products, open systems, quality, and Microsoft's software development culture. He is vice president of Microsoft's Developer Group, which includes platform technologies, development tools, and database products as well as providing support programs for the developer community. He is a member of the Microsoft Business Leadership Team, which shares responsibility with Microsoft CEO Bill Gates for the company's broad strategic and business planning. Maritz joined Microsoft in 1986 and has managed such software product groups as Networking, and Windows operating system and application units. He spent five years at Intel Corp. before joining Microsoft. He also worked for Burroughs Corp. and at the University of St. Andrews in Scotland. He is a graduate of the University of Cape Town and the University of Natal, South Africa where he studied computer science and mathematics.

CrossTalk: How does Microsoft determine whether to build or buy software?

Maritz: You have to decide first whether you want to do some unique things in the software. It is really a decision between what you want to achieve with the software; do you want to have a unique proposition for your customers? Secondly, if it is something you think is a standard piece you want to have inside your environment, are there such products available? Are they of sufficient quality? What are the business terms under which you can acquire them? All those things have to be thought through.

We do buy or license software in many cases. Over time, though, we tend to license them under scheme-ware by which we can put them through the same quality cycle, the same maintenance and support cycle. We are in the business ourselves of providing standard building blocks and our customers expect to see no difference between a piece of software that we built ourselves and one that we acquired. They do not want to be bothered by us saying 'we can support this one but we cannot support that one.' They want them to have the same level of quality.

We tend to look at situations where we have the opportunity to acquire software because in general it is a lot more efficient to acquire it because you do not have to do the development work yourself; however, you have to factor in:

- Do you want to have a unique value proposition?
- How are you going to support it?
- What quality level is it up to?
- The degree you want to integrate it into the rest of your environment.

In a lot of our cases you cannot buy an operating system with a full environment off the shelf, so we look to acquire elements of that and engineer it to certain quality levels and integrate it into the environment.

You cannot generalize too much in the decision criteria that we would go through as compared to a customer or user software. There is a different environment, although perhaps with some of the embedded systems or mission-critical systems in the Department of Defense (DoD) they would have to go through a similar set of criteria. You probably are going to have a differentiated competency or functionality vs. your enemy. It does not help you to have exactly the same software system as your enemy. You are going to want to make some changes to it, and you are going to want to customize it; you are going to want to engineer it to a certain level and support it over a period of time.

I think the first decision you have to make is what role is the software going to play in the environment? Then go from there.

CrossTalk: How does Microsoft evaluate, select, and integrate commercial off-the-shelf (COTS) software into its processes? Understand, this is a big issue in the military.

Maritz: We are in the business of trying to buy standardized building blocks that other people can use. While Microsoft will buy software for our products, we tend to buy them in the context of how we integrate them with the building blocks that we are going to supply. Another way you can look at it is how we use commercial software in the running of our company? We have made some very strong moves to get onto standardized software. Looking at our own internal financial and accounting software, if you go back five years, we had a hodgepodge of systems that we had taken and customized. We needed a good, efficient accounting system. We put ourselves through a very rigorous process of moving to the Systems, Applications, and Products in Data Processing (SAP) R/3 system; that was a three-year process, but it was a top-down decision. Today we are able to turn reports dramatically more quickly than in the past.

CrossTalk: That is very insightful. What you are saying is you do not create all of your own software.

Maritz: Absolutely. We want to focus our software development efforts on the things that will make us unique as a company and take effort off the things that will not make us unique. I think any entity has to think about those two things very carefully. The hard thing about getting in the areas that are not critical and [where] you do not have to be unique—it requires an explicit top-down decision. It really has to be a business decision; if you leave it to the technology people, there will always be 15 reasons why you cannot move.

CrossTalk: Do you have a percentage as to how much is COTS and how much is your own software that you use?

Maritz: Our software efforts fall into three categories:

Areas where we decide to build software products— operating systems, personal productivity tools, development tools. In those areas, we do buy the software; we tend to buy it and integrate it into our process. The goal is when it reaches the software it is indistinguishable from our other pieces of software.

Areas where we act as a consumer of our own product— wherever possible, we should be using the same products that we sell to our customers. We use Exchange as our messaging system; that was not always the case. There was a time when we had our own home-brewed, internal electronic mail system.

Areas where we decide not to build software products— there is a lot to be gained by leveraging the research and development that SAP does across a variety of industries rather than trying to have unique software for our environment.

The hard thing is you have to strike the right balance between being willing to change your internal ways of working and processing, and getting the software customized. If you take the off-the-shelf product and decide that you are not going to change the way you work, you run the risk of writing a layer of customized software that puts you back in the situation where you were before—having unique, expensive software that you have to maintain.

You have to be willing when you go to a COTS environment, to look at your business processes and be willing to re-engineer them. Most companies that are really successful realize the value of that and are willing to do business process and re-engineering in parallel to putting the standardized software in.

There has to be a balance between getting the software to do what you want it to do and working within the bounds of the software. Otherwise you can quickly evaporate any cost savings.

CrossTalk: Do you have any lessons learned from working with COTS products?

Maritz: There are really two key lessons:

[1] You want to make it a top-down decision. Typically you want to get all of your organization to use the same basic, underlying products; you do not have multiple relationships, multiple learning. Approach it as an important philosophical or business strategy decision and be prepared to put the willpower and effort to see it all the way through.

[2] Being willing to re-engineer your processes, if need be, to fit in the bounds of capabilities of the software you are buying.

CrossTalk: The DoD is developing a common operating environment to improve interoperability. Microsoft seemed to develop a common operating environment in Windows for commercial use. Do you have any advice to the DoD in its endeavor?

Maritz: Be pragmatic about things. It is unlikely that the world is ever going to move to one operating environment. There are always going to be three or four commonly used operating environments. The critical thing for the DoD is to be able to leverage the research and development that went into each of those environments. If you get too far ahead of industry, what happens is that in order to get products for a common operating environment, you end up being the one who has to fund the new development for it. You can get in the situation where the DoD has to pay for vendors to adapt their products to a common operating environment, which is not a good situation to be in either.

Having standards is very important. You do not want to have whole parts of your organization going in different directions without being able to reuse and leverage your investment.

The other thing is to recognize that most important standards in the industry have evolved in a de facto way out of existing products, rather than in a top-down way. The real challenge is to have a balance of bottom-up and top-down, recognizing what is working in the industry and being willing to adapt your framework standards to take advantage of those.

Do not get caught in the trap of doing standards for standards' sake. Otherwise you could end up funding it all, which is not what you want to do. What you want to do is leverage other people's investment.

CrossTalk: If Apple OS9 represents a closed operating system and Linux represents an open operating system, where is Windows NT on this continuum?

Maritz: Is Linux an open operating system? You have free access to the source but is there any official standard party that controls the interfaces to Linux? No. Is that an open process or not? I do not know.

We do not think of [Windows NT] either as a closed or open operating system. We think of it as basically as a system designed to solve computing problems for a wide variety of users.

CrossTalk: How do you maintain interoperability among third-party application developers?

"Great software design is being able to strike the right balance between the right granularity in terms of your design, and the practicality of really using your resources, CPU, and memory efficiently."

Maritz: We do this by maintaining a complete set of compatible Windows-based interfaces that we maintain on a generation-to-generation basis.

We are also very careful not to break compatibility when we release new versions of Windows. There are literally hundreds of man-years that go into preserving that application invested by third-party developers. We literally have to test thousands of applications to make sure that in each application, third-party investment by those who have written the applications, can be leveraged in going forward.

On the other hand, we also provide standard interfaces, such as the Internet set of interfaces (HTML, XML, etc.), that people have invested [in] that are in or around those interfaces.

We have a responsibility to the Windows developer community, which is a very large, very important community. And we have a responsibility to try and provide the other important standards that are in the industry.

CrossTalk: On your third-party developers, you do not set any kind of process standards or testing standards?

Maritz: We have to walk a fine line between not trying to tell other entities how they should run their businesses, and providing some guidance to customers as to what they can expect from an application. We do have a certification program where people can certify that the applications have a certain characteristic but we cannot be in the position of determining other companies' fate by blessing or not blessing their applications. We have to provide the tools that enable people to write good applications, but we cannot be the ones to be the arbiter of their business success.

CrossTalk: The SEI Capability Maturity Model (CMM®) has a strong influence on defense software development. Does Microsoft use the CMM? Why or why not?

Maritz: We do not use the CMM directly. We use common ideas from the CMM and again, this is an issue of looking at the different environments that we operate in. There are both similarities and differences when you are developing software basically as a component for some larger system. You are developing software that has to be sold as a product in millions of units. I think there is a common element, which is discipline and maturity. Those are, without any question, key characteristics. We use different mechanisms to assure discipline and maturity in our process, which do not fit into exactly the same terminology or sequence that CMM does. But the goal is the same.

We have different issues that come into play. Typically in a CMM model, it is understood that the requirements are well understood up front. We operate in an environment where you know there are certain basic requirements you understand up front, but you really do not know what is going to be required until you can get the product in use by users. People are going to use this product in many, many different ways. We have to have a different methodology that gets user feedback much earlier into our cycle.

I think one of the major differences between our process and a traditional Waterfall software process is that we use what we call a milestone methodology. Essentially you release the product in a series of releases; typically, anywhere from two to four releases occur. You try to build a base set of functionality and try to subset that functionality so you can get it useable by a certain critical mass of users as early as possible in the cycle. That is important for two reasons:

[1] You cannot anticipate how people might react to the product. It is very important how the end users are going to react. Are the dogs going to eat the dog food?

[2] No matter how much effort we put into writing software test suites—we literally have one developer writing test code for every developer writing code to begin with—you cannot predict all the ways the product is going to be used. The product is always going to be used in unexpected ways that are not covered by your test suites. It is not enough to have a product that passes your test suites. This is something we learned in the 1980s, together with IBM, when we were applying traditional software methodologies to developing commercial software. We got in situations where we would parcel our test suites and release the product and people would say, 'that product is terrible.' You have to strike a balance between different techniques for assuring quality. Quality comes from design discipline and formal software verification via test suites, and actual product usage. It is really the union of those things that we have found [is] what gives a good product.

We have a different way of trying to inject discipline and maturity into our process, but the goal is the same. It takes time. We found it takes awhile for a team to develop that can both formally meet the criteria and informally function well enough as a team to deliver on that.

CrossTalk: How does Microsoft assure software quality?

Maritz: We try [to] put a net together of well thought-through designs, and use design quality checks such as design walk-throughs, peer code reviews, etc.

We write an extensive amount of software that we use to test our own software. In addition to that, one of the things we have found over the years developing very large pieces of software is that you have to be constantly measuring yourself as to where you really are. Also, [we follow] this notion of exposing yourself early to end-user testing.

We typically look at a software project and we have found there are several questions you can ask a team. They will sure tell you whether they are out of control and have bad quality. How regularly is the whole team putting the software together? What we have found is that no matter how good a design is and how well you have decomposed the problem, nobody is ever perfect in their design and how all the pieces are going to splice together. If you have a team that is not mature, and not functioning well, it tends to leave that very late in the cycle. It is a real effort to put all the pieces together and integrate them on a regular basis.

One of the disciplines we use is insisting on regular integrations. If a team cannot integrate all its pieces, on a weekly basis, that is a warning signal. It means it really does not have the right contract between design elements in place. Because our software methodologies are good at describing structural aspects of software, we do not have good theoretical tools for predicting performance in terms of software design. Unless a team has written tests to measure its performance as being rigorous about measuring itself and performance, that can be a warning signal. In many cases, having too many layers and too many interfaces militates against performance.

Great software design is being able to strike the right balance between the right granularity in terms of your design, and the practicality of really using your resources, CPU, and memory efficiently. Forcing yourself to be honest about where you are in terms of performance is another key issue.

Another issue is your bug backlog. No team can recover if it lets its bug backlog, its performance backlog, or its design backlog get too far out of hand. What we try and do is structure our life cycle to say you really are going to release this product three or four times and you need to have an audience and a set of criteria of quality and performance associated with each of those releases.

You have got to be very disciplined about treating each of them as a release. When you hit a milestone, you have got to say 'we have formally met our quality criteria, we have formally met our performance criteria, and here is this set of people, outside of the team, that says 'Yes, this is usable, you are on the right track.'

One of the key things important for people to internalize is that quality is an absence of bugs, it is the right performance, and it is the right set of features. Anyone of those three things can lead a product to have a bad reputation.

CrossTalk: What standards are used by Microsoft to assure product quality? How are they created? How are they enforced?

Maritz: The key is to get people to be honest in articulating up front the criteria they are going to use. Realize that you also have to revisit your criteria at each milestone, because you will learn when you reach a milestone about the nature of the software you are building. It is OK to revise your criteria, but only when you get to a milestone. When you get to a milestone, there are two things you learn—if you met your original set of criteria, and if you should revise your criteria for the next milestone?

CrossTalk: But is that difficult to do? Change is always difficult, you have a process and sometimes you get so wedded to the process.

Maritz: Exactly. Often there is a lot of pride involved and people like to kind of fudge on the milestones. That tends to be pennywise, pound-foolish. You are much better off forcing issues out earlier in the cycle.

CrossTalk: What is the single most important aspect to assuring the quality of Microsoft software?

Maritz: It is hard to say the single most important aspect. We try to use a network of different techniques and ways of looking at the problem. You cannot simply say formal methods alone will do it for you. Formal methods will get you some of the way. However, informal methods tend to be very important because they ultimately tell about you whether it is the right product or not, but they can be very difficult and expensive to use.

It is basically a toolbox of techniques that we have learned over the years. The thing that is most emblematic of our process is this notion of milestones. You try and decompose the product —not just breaking it into subsystems, but equally important, breaking it into releases where each release has been chosen to have a certain level of functionality that you can get objective, formal testing on and objective user-testing on. Both are needed to prevent situations where you get products that go out of control and you find out too late in the cycle.

CrossTalk: How does Microsoft attract and retain good software developers and managers?

Maritz: With difficulty, like everyone else. You try and create a culture where those people believe they are valued. You try and create a culture where they believe they can do interesting and important work. You balance that with the fact that you cannot be doing interesting and important work all the time. There has to be a balance between inspiration and creativity, and plain old hard work and slugging through it.

We try to create an environment where software engineers know they are at the core of what we do. They are valued as individuals. If they put in hard work they will be rewarded by being treated well, being compensated well, and getting the opportunity to do interesting work. They can continue to grow as individuals.

Most people who do software development do it only partly as a way of earning a living. They do it because they have a passion for it and they get a lot of satisfaction out of doing it.

CrossTalk: It is something that the DoD struggles with, attracting and retaining that intellectual resource.

Maritz: Are you putting enough attention into structuring so that these people have an interesting career path and get to work on interesting problems? I think there should be no shortage of [interesting software problems] in the Department of Defense, balancing that with the fact that every job in life has a certain amount of plain old hard work associated with it.

The world is becoming a software world. There is no aspect of life that does not have software in it these days. Consequently, there is tremendous demand for these people. These people are a scarce resource and have to be compensated accordingly.

CrossTalk: Do hotshot software engineers make good software managers?

Maritz: Not necessarily, but some of them do. There are certain hotshot software engineers who have no interest in being a manager, will be bad managers, and you have to construct a career path so those people can advance and be valued by the organization without having to become a manager. In many cases that means teaming them up with somebody who is a good manager.

We have a distinguished engineer program, where somebody can rise up where essentially they are compensated as a senior executive of the company would be compensated.

You need both. You need great architects and great software managers. Finding those people who have the right characteristics —their level of maturity and their propensity to want to work with other people—and encouraging them is very important.

The best software managers are those who also have a good understanding of what their people are doing. They do not have to be expert, but they have to have a good appreciation for what people are doing. Software engineers do not have a lot of respect for their leadership unless they know their leadership has a basic understanding of what their issues are and have sympathy for the challenges that they face. [Knowing] if some critical decision has to be made that that decision will not be made randomly. The manager will have enough background to know whom to ask and get good advice.

The key is to get the individual to recognize which track he is on. It is, in many cases, convincing people who are on a technical track or management track to be comfortable with which track they are on, accept that, and realize that they do not have to be the other in order to get where they want to go.

CrossTalk: What qualities that are rare among software engineers do you look for in a good software manager?

Maritz: When you become a manager, you can never be the expert in all areas. You are somebody who has to show leadership by absorbing a lot of good ideas and advice from other people. Those people who tend to do that are more secure in themselves. They are able to take feedback. They are not afraid to appear to be dumb by asking questions and learning. They tend to be people who can get honest feedback. They are people who have a lot of self-confidence with a small 's' and a small 'c.' I'm not talking about braggadocio. And [they] also have a good appreciation for what it is the team is trying to do. You are leading technical people. You have to make it a point to know the problems that people are facing. You do not have to come up with the answers, but you have to know enough to realize when somebody is trying to put one over on you, when somebody is lying, or when people are in need of help.

CrossTalk: What does Microsoft do to encourage long-term employment and loyalty from productive software developers?

Maritz: It is a combination of three things:

[1] Making them feel very good about working in an environment that values and respects them—they feel they are *mission critical.* Nobody wants to work long-term for an organization where you think you are in a secondary or under-appreciated role.

[2] Making sure that they can work on things that engage them—every job has its element of tedium associated with it, but you have to make sure there are enough interesting and exciting things that people can feel engaged and challenged.

[3] Compensating them.◆