Plaintiff's Exhibit
6709
Comes V. Microsoft

From: Steven Sinofsky
Sent: Tuesday, February 01, 2000 7:53 PM
To: Bob Muglia; Paul Maritz; Jim Allchin
Cc: Steve Ballmer
Subject: Evolution

header htm

At the TLT meeting we were discussing the idea of why sort of assumptions we would make differently about applications. Here is a short note on this from a while back. I think it would be good for us to have as a goal, a list of "what sort of things we would assume differently" to tell developers about NGWS.
See the sections on Dinosaurs and the Comet, and skip the first section.

To:
Chris Peters; Richard Fade; Jon De Vaan; Pete Higgins; Nathan Myhrvold
From:
Steven Sinofsky
Subject:
On the Evolution of Office
Date:
October 1995

We have spent some time over the past few months considering how DAD will invest longer term, even longer than our current 12-24 methodology. This memo proposes an approach, and a new product we should build, with the explicit goal of making a bet that will pay off in 3+ years, but in the short term is distinct from our current investments.
This memo puts down thoughts based on numerous conversations that we have all had over the past few months, so many of the ideas or thoughts written are not entirely those of the author. The goal is to provide an impetus to begin thought on this issue and how to proceed, not to totally solve this problem. This memo should be shared among this small group of people.

You must not fight too often with one enemy, or you will teach him all your art of war.
Napoleon Bonaparte (1769-1821)

12/24/48?

The 12/24 methodology has allowed us to invest in the major code-bases of the Office product while still managing to remain competitive in the marketplace. Some aspects of this 12/24 are going very well:
· Office 95 is perceived by most to be a major upgrade, one that stands on its own even with the leverage that came from Windows 95. This came from working on highly leveraged and visible features, and investing very little under the covers.
Working on Office 95 was exciting for much of the team-the small product atmosphere and the lack of external dependencies was fairly exciting.
· By avoiding major reworking of the code or architectural changes, we avoided changing the file format of Word and Excel. The benefit of this has been immeasurable. Feedback from marketing and corporate accounts is crystal clear on how important a "feature" this is to the release.
· DAD proved that it can release all the major products at the same time, something which had not been previously accomplished.
Office 96 is a major major release. The depth of the feature-set is greater than in any previous release of Office and clearly represents the work of the 80% of the resources that have been on the project for the past 18 months.
· Investing in sharing code and architecture, although not a done deal until June, has become a first thought not an afterthought. The amount of leveraged code and architecture in Office 96 is very significant-perhaps several megabytes by the time we ship.
Office 96 made clear architectural advances in core areas of the products: memory management, localization, string management, user-interface, OLE, and drawing.
We should not understate the advancements made by the product teams in this area. However, there are many issues that we still must find a way to deal with adequately in order to safeguard our four plus billion dollar business. In particular we must look ahead at the market and customers and ask ourselves if we are prepared to respond to their needs and if we can do so in a timely manner
One way of looking at this is to evolve the 12-24 paradigm to one that places a longer bet, so called 12-24-48. Chris has

written down some possible scenarios for how we can build upon our products and process a 48 month stage. My conclusion in thinking through the 48 month cycle is that the idea that somehow this bet must be connected to our current products is itself a false basis upon which to build a development strategy.
By betting all or a portion of a team on building 48 month developments into the current product, we are doomed to failure. No group is smart enough about our industry to know what bets to make now in our products today in order to have them pay off in four years, all in the same product. One way to think about this is to ask what features were worried about four years ago in Excel and compare that to what ended up in the product. Although there are some things that have been perennially on the list of adds, and then cuts, the marketplace clearly did not miss them (though perhaps we regret not having done them for development efficiency reasons). We can continue to explore how to just "extend" our 24 month cycle to a 48 month analogue, but it would be hard to convince me that we would find a process by which we can work on meaningful features in parallel with our current products.
What is needed, though, is a redefinition of the 48 month aspect. Instead of thinking of it in parallel to our current process, we should consider the 48 month time frame to be an independent bet on something that we think (strongly believe) will pay off handsomely in the four year time frame. In other words, while we should continue to bet largely on the code, process, and architectural aspects of 12-24, we must look hard at the current state of the marketplace and products and spend some of our efforts on a completely different effort. As Pete likes to remind us, we must be sure that the "generals are not fighting the last war."
This differs distinctly from the previous attempts at longer term bets in DAD (i e., Pyramid), and is in fact closer to the bet on Windows NT.
What DAD needs to do is build a product that could become the future version of Office in four or more years, but has intermediate deliverables that will in many ways be perceived to be "inferior" to Office, a subset of features of Office, overlapping with Office, and in general similar but different.

Code Is Like Dinosaurs *

Given the success of DAD, questioning the path we are taking is always a perilous path. One should be rightly cynical of anyone that claims to have a better way to do things, since clearly there is no proof to support such a bold statement. There is no question that the code in Office is the best in the world at doing what it does, but like all organic things it has a finite lifetime. This finality is not based on poor engineering, but is based primarily on the assumptions made at the very start of the product. We can look at some of these assumptions in a moment, but first let's consider an analogy.
Going back to primordial times consider dinosaurs walking the earth. They were finely tuned beings for their time. The earth was covered with methane, shrubs, and was in general a pretty hostile place. Dinosaurs evolved in a manner consistent with their environment and thrived.
Of course we all know just as dinosaurs were thriving something drastic happened, an ice age or a comet or alien beings. In any event, some external force greatly changed the environment around the dinosaurs and they found themselves unable to adapt. Now this was not an instant change, even in geologic times. Nevertheless one day the earth was a different place and the assumptions built into the dinosaur were no longer valid. Out of this change came mammals-animals that were quite different from their dinosaur ancestors-warm blooded, mobile, adaptable, fur-covered, and small. Even though the goal is the same, survival of the fittest, the environment created a number of different assumptions that led to radically different beings walking the earth. The rest is history.
What the heck does this have to do with Office? If one looks back to the start of Office, perhaps just the first Windows applications, there were many assumptions that were made (though these are not universal by any means, even across Office):

- Stand-alone applications dominate
- Categories consisting of spreadsheet, word processor, presentation graphics, database
- Testing software was an afterthought, or a small portion of development at best
- Teams were started with 2-3 programmers, but we reached a limit at about 40
- The product architecture was really the work of "one guy"
- Sharing code is hard
- Disk-based file formats
- Networking limited to file/print sharing
- CPU bound applications are the norm
- Virtual memory not available
- Operating system services are slow
- Users can run setup on their own
- Documents are primarily printed
- Images in documents are primarily adornments
- Macros were run in process and for a single application
- Most information is stored locally
- Document structure manipulated and created by the user

This list is hardly complete and meant to be illustrative of the sort of things that are pervasive through our designs, code, and user-interface. The claim is not that these are bad or wrong, or even that it is not possible to change any one assumption. Rather it is the collection of these assumptions that have led to the current products
Our products have evolved over time by questioning these assumptions and modifying them one or two at a time, but that still leaves the core body that is built upon these assumptions. In some sense, we have been attempting to glue fur on the

dinosaur. † I would agree that it is easy to take any one of these assumptions and change course taking this into account. But as we know the critical aspect of software, the aspect that gives it an organic characteristic, is the very starting place of the product. Even today, the fundamental architecture of every one of our products remains essentially as it was in the first release. Even with the re-architecture that takes place in every release, even more so in Office 96, our ability to add new code to that existing infrastructure far exceeds our ability or desire to revisit fundamental assumptions and adjust our course significantly.

We are the products of editing, rather than of authorship.

George Wald (b. 1906), U S. biochemist.

"So what?" you are probably asking. All would be fine if the evolution of software continued as it has been for the past 3 years, but as we all know things are fundamentally changing and the business around us is changing.

The Comet

The dinosaurs had their comet. The personal computer has the Internet. As Bill has written and as we all know by now, the Internet more important than the GUI in terms of the development of the software business. The Internet has become the next assumption we must make, just as when we assumed the user had a 10MB hard drive, then 4MB of RAM, then protect mode, then recently Windows 95, we must now build products that not only assume the connectivity of the Internet and functionality of its protocols, but we must leverage this new infrastructure.

What is different about the Internet, however, is that many of the original assumptions built into our product now fly in the face of the evolutionary trends. I am confident we could address each one of them as they come up, but the time is right to build a new set of assumptions and build a new product that has those assumptions imbued in each one of those seminal lines of code.

What sort of assumptions would one make if one was starting an application today?

- Drawing and graphics are the norm, not an exception
- Virtual memory replaces disk based file formats
- Multi-stream documents are the norm, along with progressive rendering
- Interoperating with Internet protocols is a requirement
- Programmability should start from higher abstractions than the user-interface
- Documents will be viewed on-line
- Documents will contain more active, user-encoded behavior
- Applications need to be easier to setup and install
- Knowing the structure of a document is of paramount importance
- File formats need to be tagged for upward compatibility

As with the previous assumptions, these are all clearly issues that can be addressed in our current applications, but in the end we will address these assumptions as new and additive, rather than replacing old assumptions we made, assumptions which would still permeate the existing applications.

Evolving-Placing a Bet

What I propose we do is fairly simple in concept, but hard in practice. We should do is build a new product, conceptually a product we could think of as "WebWorks" and build it. This is our 48 month bet-if it goes right, in 4 years we will be selling Web works the way that Microsoft sells Windows NT versus Windows 95 today.

First the obvious. Works clearly hasn't taken over for Office sales, so why would it start? I'm using to the Works moniker because clearly the combined functionality of Office will be greater than that of this new product, in the traditional areas. But it will be the simplicity of integration and seamless interoperability with the Internet that will distinguish WebWorks. Specifically what we are not setting out to do is build Word+Excel+PowerPoint+Access all in a new fancy integrated way. Rather we are building a new application that will borrow concepts and code from each of these, but with a goal of producing the "documents of tomorrow by the users of tomorrow for the readers of tomorrow."

So what is it we should build? Looking at the documents we think people will create in the next 3-5 years, we will start there. Documents will be rich in graphics, links, animation, content from other sources. They will be viewed on line and published, rather than printed and distributed. They will look like a combination of Word, PowerPoint, and Publisher. The documents will have the graphics handling and layout capabilities of PowerPoint. But they will also have the text handling of Word. Documents will have the structure of Publisher documents. In addition, we know that the functionality of spreadsheets are necessary to the average Office user.

Is this product just an HTML editor? Well in some sense it would be. Clearly we want to have a tool that is part of the Internet. Some aspects of HTML solve problems we have with our products today: intrinsic links, ASCII based tagged file format, multiple streams, ease of extensibility, etc.

Well this still isn't concrete enough for you is it? Designing the product or creating a vision statement in this memo probably is not the right thing to do, but I'd be glad to talk about what I think it should be Let's say we started with our text editing architecture, but we would focus on delivering a the needed tagged file format integrated with standard Internet browsing. To that we would add drawing, probably from Escher. Links would be a pervasive concept and not one added as an afterthought. We would build in programmability, including events. The fundamental user model needs to be a new kind of document Not a sheet of paper, not a VGA screen, and not a piece of accounting paper. AndrewK describes this as a piece of graph paper, where the user just does what they want where they want to, and the application just "does the right thing."

We also need to take a look at the assumptions above and understand better the assumptions we are making up front-the axioms of the product. For example, we should certainly build this product with "testability" or performance measurement up front design issues, much like how Windows NT changed their approach from Windows.
The base philosophy of this product is no different than any other DAD product We must build this assuming that this is a tool for the average person who is creating and analyzing information for their "real" job That is, the Office user of today and the WebWorks user of tomorrow have jobs that are not in the publishing or media production business-they are office workers, sales persons, authors, accountants, teachers, students, small business owners, managers, and CEOs.

How Do We Evolve?

I'll make a big leap and think that you agree that building a new product is a worthwhile bet. If you are not convinced, I will steal a line from Nathan and ask, "isn't this worth just spending a couple of percent of our effort, just in case?"
The goal we are striving for is to build a framework for the real future version of Office, but with an intermediate deliverable. This deliverable in some ways is the product of creating a "super" version of Office DLL, one that is really the foundation for a new generation of application. We should also plan on stealing a lot of code from our existing code bases. In that sense we have a number of groups clearly working on "technologies" rather than features and there is no reason at all to think we need to re-write these, for example, natural language, charting, pivot tables, drawing.
Clearly the idea of building a new product is exciting. It is even more exciting if you get to start from scratch with a clean slate On the other hand, Microsoft does not have a good track record for delivering on these visions. Perhaps only Excel , Windows Word, and Windows NT are the only examples of such products, products that evolved from a clean slate, but based on the work of existing product groups.
We must continue on 12-24 for the rest of this century. Given the issues corporate accounts could potentially raise, we might consider modifying this to return to an 18 month single release cycle. I think that is the right thing to do, but this is a separate decision. The most important work in DAD for the next 5 years will be done by the team that is working on the next 3 releases of Office, Word, Excel, and PowerPoint. Anyone who thinks otherwise thinks that the climate around the dinosaurs changed overnight, literally.
But in order to begin this project we should gather a team, not al all-star team, just a team of people equivalent to the Office 95 team in size from across the DAD product groups. This should be a logical team, in the sense that creating a new business unit would only accentuate the separation and elite nature of this product, which is not what should be done. The initial vision and leadership should come from within our existing management infrastructure. I know this sounds like a recipe for failure, but I am siding with both caution and a desire for maintaining contact with the core DAD business, at least until the product takes shape.
We should start with the program managers working on designs and prototypes through the next spring. Since we really can use code from our existing applications, much of the up front work really revolves around designing a new user-model and we should invest heavily in this at first. As Office 96 development finishes we should begin to form the development team. As coding begins we would create the testing group as well. The initial goal would be to have an architecture, design, and development plan within 9 months of the first assignments to the problem. This product would be run just like any other DAD product, after the initial "open-ended" kick-off of 6-9 months of pre-search It is at that point, and only at that point, that we should consider forming an official product unit to build this.

Where Do We Go From Here?

Thus memo was written as a starting point for discussions. There are many discussion to be had and decisions to be made. Below I list a few areas that we should consider as points that we need to discuss, in a sense these are some of the "open issues" that come to mind. Again this is hardly a complete list.

Process

- Who do we pick for this product? How many people?
- When do we start program management? Development? Testing?
- When do we create a product unit?
- How do we make sure Office remains the cool place to work?
- What is the time frame on deliverables (protospec, specification, schedule)?
- How do you get something up and running soon?
- How do ideas move freely and easily between Office and WebWorks? Are there specific goals for sharing code, information?
- Do we encourage "inefficiency" in sharing code between Office and WebWorks?
- If we want to design for testability, how do we involve testing in creating a new "framework?"

Technology

- What is the page model really? Many have tried but can't get past the prototype?
- How do we avoid oopaholism?
- What code do you start with? Many would say Quill
- Some critical components are from the "dinosaur" age-especially OLE, ODBC, etc.

- Do we attempt to have converters to/from existing documents? Is it a "requirement"?
- Does this product incorporate e-mail?
- How do we avoid being very politically incorrect, regarding Forms3, Nashville, etc.
- What assumptions to we make about hardware?
- Is there a Mac version?
- How do we implement programmability?

---

* There are probably better analogies than dinosaurs, especially when trying to communicate this issue broadly, but this is the one that came to me. I have some ideas though...

† If this sounds like something EdF would say, you are correct, though he used it in quite a different context.