Microsoft Memo

To: Chris Peters; Richard Fade; Jon De Vaan; Pete Higgins; Nathan Myhrvold

From: Steven Sinofsky

Subject: Web-Centric Productivity Documents

Date: August 1996

This memo describes in a brainstorming fashion what a web-centric productivity tool might look like. This idea comes from numerous discussions with many people, so many of the ideas in this are not mine but have been collected here. The goal in describing this sort of application is to think about what a "new" DAD product would look like and what some of the assumptions that might go into building such a beast. This is a follow on to the memo about evolving Office (SteveSi).

The basic premise of this memo is that the Internet is so fundamentally different that we must re-examine what we mean by "Office" in light of the changes brought about in infrastructure and workstyle by the Internet. This does not mean that Office is obsolete or should be abandoned, rather this memo attempts to describe what an application that starts off with the assumption that Internet is primary, Internet protocols are the defaults, and that we have learned a great deal about what workers need in productivity tools.

Who is the customer?

The biggest threat to the Office business is if we lose our ability to sell upgrades to large corporations. We have a number of clever ideas for turning this business into an annuity stream, but there is some chance people will just balk and we will become a one time purchase for each new computer. One computer, one copy of Office for the life of the computer becomes the business model. This needs to be addressed in Office as part of the upgrade/TCO work that is being thought about for Office9.

However, the real threat to Office is the broadening of users of PCs inside of corporations. These users are now less part of the knowledge/information workforce and part of the broader workforce. The IT excitement over of "browsers" and "applets" is that these workers can be given access to computing in an easier to use and lower cost means. Office is overkill for these sorts of workers. There is also some general sentiment that Office has become overkill for even traditional customers. There is certainly a lot of truth to the fact that putting Access, VBE, Pivot Tables, etc. on every single desktop is conceivably overkill. There is also a great deal of benefit achieved from the standardization on these tools, as we well know, but this does not make the requests from current customers for a trimmed down version of Office moot.

Email in a sense represents this threat most clearly. It is now more important for users to communicate quickly and easily electronically than it is to prepare lengthy documents and distribute them in print. There are no fewer proposals, budgets, or plans being created, but in an exponential sense the number of "document snippets" being created is exceeding these traditional documents. As the providers of productivity tools to workers, we must recognize this and apply our skills towards solving the problem of creating these new knowledge worker documents.

The observation one could make regarding the Web and its impact on corporations is that many more people will be creating small documents and distributing them via the Web. What might have been distributed as a copied document will now default to web distribution. These people are starting to use email, though unsatisfactorily. Email allows someone to easily and quickly create and distribute a short document to a large audience. Still they are unable to use email for standard corporate communications. WordMail will start to address this concern, but nonetheless there remains the view that Word provides too much "baggage" to solve this problem as well as customers are demanding.

The typical document created in a corporation is well known. Today, it looks mostly like a Word document—streams of text, an embedding or two, and that's about it. However, the Web has forced a new way of thinking about communication on corporations. This goes beyond links and pictures towards a new level of integration of graphics, list management, collaboration, and online presentation.

People have always wanted the "universal container" or the single document type that meets their needs. We have traditionally approached this by viewing their "needs" as a superset of the functionality provided by all of Office. My opinion is that this is where we need to take a step back and re-evaluate this assumption. We need to provide an application tuned towards a new paradigm where a document more closely resembled a collaborative web page that contains the frequently used elements of our current offerings. Although one could hear "Works" as the answer to this question, this isn't the case since Works is nothing more than Office with fewer features.

Taking these observations, a big leap of faith, and a speech from Chris Peters about making bets on the future, the remainder of this memo looks at what this new application should look like. The name *Stretch* has been chosen for this application.

To be totally clear, this application does not replace Office in version one nor is it designed to be a superset of Office—today. The goal is to build a framework so that a we can have a great version three in 3-5 more years. A secondary goal is to force platforms to think about certain problems and issues they might otherwise ignore if ISVs such as DAD were not pushing their vision of a platform.

An imperative of embarking on creating this application is that the majority of DAD remain focused on Office9 and solving many of these same customer issues within the successful framework of Office 97. It is clear to me that much coopetition will take place between the Office9 team and the team building this new application. This is healthy and should be encouraged.

What Do We Build?

The biggest area where this application would differ from Office lies in the initial design, architecture, and user-model decisions that get made. In particular the key distinction rests with the fact that the world is a different place today and anyone building an application would do different things. There are several key areas which this memo will focus as major architectural differences. Each of these areas shows how the current Office framework is essentially the opposite of what one would do (or has done) for Office 97.

The items below are just some major concepts that I think we should explore. It is hardly definitive or complete and many might turn out to be wrong, but the general desire is to look at the assumptions that go into Office today and see which ones are no longer correct. Clearly Office9 should revisit this same list and change as many as possible. The only argument being made here is one that says, if you make this assumption from the very beginning you can have a better result. This is no unlike the difference between porting a DOS application to Windows and writing a new application for the Windows API.

Some of the important assumptions that are worth considering include:

- Stand-alone applications dominate
- Categories consisting of spreadsheet, word processor, presentation graphics, database
- Document type defined by paper based presentation format (accounting paper, 8x11 pages, slides)
- Disk-based file formats
- Binary file formats for efficiency, with ASCII formats for transportability
- Code to manipulate the document resides on the desktop only in the applications we ship
- Networking limited to file/print sharing
- CPU cycles are a limited resource

- 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
- Macros were derived from the user-interface of the editor
- Finding documents is based on storage location conventions
- Naming conventions and directories are the standard way to group files
- Most information is stored locally
- Document structure manipulated and created by the user
- User-Preferences are stored on the user's personal machine
- Running application setup is the way to get bits onto the personal machine

The above list is not complete, but it gives you an idea of the sorts of things we should "turn on end" to see what falls out. The following are some of the key ways in which a new application might leverage web services.

Storage and Management of Documents: The current applications are all file and UNC based. This application would store documents on a web server, with server side code intervening in the storage and management of documents. The facilities provided by the server daemon (an ISAPI application) include: content indexing, keeping track of the users personal documents, publishing documents for a workgroup, publishing documents for everyone to see, logging the history of a document, etc. The best way to think of this is that when you create a document, the default action is to place it on the web server, rather than the local drive. You would store the document in a web-centric version of \My Documents, if it is private, or in a workgroup/public location as defined by your administrator. Users would be able to log on to any machine and through the web browser see their documents, and workgroup documents just as if they were at their own machine. This is done via server side code that understands the notion of where Stretch documents are stored. Of course there is a special case for when you want to save documents to a floppy, but this is the exception not the rule as it is today.

Project Workspaces: We have long struggled with the idea of creating a workspace of related information. The Binder provides this to some degree, but is really designed for printed documents (it does not even support links today). The fundamental problem we have continually faced with Binder was the lack of a rich storage model. The availability of a web server, which can virtualize a store by using server side code, combined with native hyperlinks provides us with a unique opportunity to implement a true workspace. For example, it will be easy to query a server for all documents with a certain category (or edited by a department) and create a virtual page (dynamically generated) representing a project, which can include any number of document types including email, public folders (via the IMC), etc. The use of templates and standard presentation techniques will make this a routine way to construct ad hoc projects. These projects just become locations on the web; they can be favorite places, you can mail URLs to these projects around, and you can easily add documents to the URL (by adding a link to the home page). It is only because we would write server side code that creates this notion of a tracking project that this would be possible. This notion is very distinct from the Nashville world where one merely provides richer views of the same old physical directory we have been trying to work around for years.

Personalization: A key aspect of server computation and browsers today is the ability to deliver personalized pages and content. One can think of this as a glorified per-user registry, only it has the unique advantage that it is dynamic and can be parameterized. For example, server side code can keep track of queries that I have

recently issues, or it can store those queries for later use. Then from any machine I would be able to gain access to my personalized information. This is also closely related to the notion of workspaces, where my personal workspace is one use of storing personal information on the server. Since there is a separation between the client and server code, the personal information is available to the browser without running the application code. Today our applications default to personalizing things on the client side with no separation between the editing application and the personalized data. Using this notional of personalization, one could also use this to store any sort of state information that one would normally include in places like normal.dot, stevesi.acl, etc.

User-Interface: One of the common misconceptions about web-based applications is that they need to "suffer" with least common denominator user interface. This is true if you wish to be able to access your server application from a TV or Linux machine, but if you think of the interface to a productivity application as a set of IE3 pages implemented with controls designed by us to work together in the tightest, most integrated fashion then you can see how we can build this application to our standards. We would be giving up the "run everywhere" idea of web applications, but what we gain is very customizable application that can separate the description of the user interface from the code that implements it. For example, creating a blank document is really just going to the Blank Document favorite place, which fetches a page that loads a bunch of controls (made up of real live Office97 user-interface objects like command bars) along with code behind those controls that dispatches commands to client side editing code. Of course there are still dialogs, toolbars, etc., but these are all glued together with VBS and calls to a rich automation model on the client and server side. Again, this is a fundamental assumption where we would change our perspective from today where we assume the user-interface is hard coded in C within the application to one where the user interface is described by pages that contain our own controls which can be arbitrarily integrated. At the recent retreat, these pages were termed Doclets, as they have a close relation to Applets.

Collaboration and Annotation: Perhaps the biggest paradigm shift taking place is one where we are moving from a world where sharing documents is the exceptional case to one where sharing documents is the rule. As the workgroup task force from Office97 showed, there is a clear need for some simple collaborative techniques to be added to our documents (and have been in Office97) which include multi-user documents, comments/annotations, and change tracking. In the web-centric view of documents all documents have these features and these features are on by default. They are part of the normal mode of working. If I am reading a document, I should be able to attach an annotation as I browse the document on the web. This should not require all the power of editing and there should even be mechanisms where I can do this if I am running Netscape on Linux, for example. Today our code for managing versions of documents and reconciliation is on the client, which makes it hard for other tools (management systems) to locate revisions or provide support for batch operations.

Deployment: Much thought has gone into the issues associated with deploying today's Office and this situation will improve dramatically in Office9. This new application would of course leverage the code download scenarios for "installing" software on the client machine (caching is probably a better word). Again, the assumption today is that you always run setup and there are some aspects tied to a machine. In the application being described, it is entirely possible to buy a new computer, go to you personal URL and start working as if you had never installed any software manually. As Internet Terminals (really just PCs you can rent at airports and hotels) become available this facility will be crucial. Of course we can and will do much of this with Office, but this application will be designed from the beginning for this functionality.

Type of Document: The most crucial difference between today's Office and the future knowledge worker's document creation needs is the type of document to be created. Our split of document types today is confusing to most users, and gets worse as we add tools like Publisher or PictureIt. There is absolutely a need for a tool as deep and broad as Excel, perhaps for the majority of workers in some corporations. If you are writing a book, then you absolutely need the power of Word. However, one look at the vast majority of even today's business communication and you have to agree that there is some subset of functionality we can pull out and provide a much more integrated editing functionality. There are reasons why we keep putting the same features in all of our applications or that WordPerfect has built a mini-spreadsheet within their word processor. Along with this subset of functionality there are two other distinctions regarding the documents of

tomorrow. First, they will not be printed by default. In fact, as with email today there will be little need to print most documents so we can do without the baggage of printing (both in code and user-interface). Second, documents on the web are "different" than any of our current categories. They have some 2D layout like Publisher, they have some streaming text like Word, and they have many of the online presentation capabilities of PowerPoint. It is very difficult to find a web page today that you could even create a facsimile of in Office97 without really stretching your ability to use our products. So the document creation tool of this application would combine many of these aspects, making tradeoffs where features are lost no doubt, but the ability to seamlessly switch between the best of all of our tools tuned for knowledge worker communication is key.

IntelliSense: One area where we can so clearly leverage our strength is in the application of our "do what I mean" design philosophy. I include this not because there is anything special we can do in this area that we could not do in Office, but because so many of the other people creating tools for the Internet are still designing tools for programmers or people that want to understand the tool, not people who have other jobs.

File Format: An area where our assumptions of the past are very hard to cope with is file formats. Today we have binary file formats where one needs the full application to even read the contents of the file (except for some third parties viewers than can yank the text out). HTML is not a magic document format, but it shows the power of having a very low level least common denominator that any piece of code can manipulate. In other words, UNIX made a good choice a long time ago. I am not confused about the fact that there is nearly a 1:1 correspondence with the file format and features in an application. However, the benefit of having an HTML derived format is that you can perform a lot of operations on the file without our code (we have solved indexing, but annotations is the one that is really important, and writing viewers becomes trivial). I don't think there is a world where we can have our own extended HTML file format and anyone with Navigator Gold could edit the document (imagine trying to edit a table in an editor that doesn't support tables—you would just munge the document beyond recognition), but it will be possible to have multiple versions of our own applications be clever about dealing with the file. We should extend HTML arbitrarily to support whatever features we need (via the OBJECT tag or just our own tags) with the only rule that we should think hard about how these tags would be viewed by IE3 or Navigator 3.

Mail: One of the toughest challenges about building a new productivity tool is deciding on the role of email in this application. One view would be to assume email becomes the predominant interface to all of your work and everything you do, and email is another application on the desktop. This is probably true. The real challenge is to decide what document format you mail around. Well the marketplace and our platforms group have decided that this is HTML. So this application can be thought of as a custom mail note, not as another mail client. In other words, any time you are editing a document you can just show the message header and send it via SMTP to anyone you'd like. This is not unlike our File Send command, except it just uses native Internet protocols. Since the document is a viewable variant of HTML any basic Eudora client can at least view the message, but it will also be a MIME attachment that can leverage a viewer. My inbox will still be managed by whoever builds the best mail client, but over time my inbox will be filled with documents created by our new tool, rather than the basic HTML editor, just as a few years ago inboxes got filled with Word attachments, which recently have become dominated by URLs.

News: It is clear that HTML based NNTP will be a key information sharing protocol moving forward (despite the fact that 20 years later it still isn't proving useful to anyone). Just as this application is a mail note, it is also a posting.

Replication: Perhaps the hardest technical problem to solve is how laptops work. The real problem that needs to be solved is getting at the server functionality and server documents when you are on a laptop. Rather than assuming ubiquitous wireless connectivity, this application would need to support seamless local replication (like Milktruck, or Exchange if you didn't have to intervene manually).

Annuity: A key aspect of this application is that it is sold as a combination of a client and a server. Each user is licensed and since there is abundant server-side functionality it is easy to track use. Since the application is really implemented as a series of web pages on the server there are ample opportunities for seamlessly integrating new content, as well as managing QFEs, etc.

Programmability: One of the key decisions we made in Office97 was to bundle the development environment with the applications. One key decision we would make for this application is not to bundle the development environment. This application would still have a rich object model, in fact this would be a primary design goal and leveraging Office97 (code and APIs) is key. However, programming this application will be done outside the application. Instead of writing a macro against a document, one is much more likely to write server-side code that manipulate the document or write a custom page that replaces the standard interface with a more task-oriented interface. I realize this sounds pretty much like components everywhere for zero cost, but separating out the development environment along with using the forms to build both the application and custom solutions is an imperative.

