A Manifesto for Collaborative Tools

Eugene Eric Kim <eekim@blueoxen.org>

March 29, 2004

This article also appears in the May 2004 issue of Dr. Dobb's Journal. Many thanks to Jon Erickson for giving us permission to post this here.

This essay is a manifesto about software for collaboration -- why the world's future depends on it, why the current crop of tools isn't good enough, and what programmers can and must do about it.

It is only proper that such a manifesto begin with the story of Doug Engelbart. In the 1960s, Engelbart and his laboratory at the Stanford Research Institute (SRI) invented the fundamental building blocks found in all of today's collaborative tools -- everything from the data structures (hypertext) and user interfaces (windowing systems), to applications (groupware) and physical interfaces (the mouse).

It's an impressive list, and not surprisingly, Engelbart has received a number of prestigious awards in recognition of these inventions, including the 1997 ACM Turing Award and the 2000 National Medal of Technology. What is surprising is that, as much as we have celebrated Engelbart's work, most of us have missed the point.

Engelbart's work was driven by some deceptively simple observations, which he described in his 1962 paper, "Augmenting Human Intellect: A Conceptual Framework." His thesis was this: Society's problems are scaling at unprecedented rates, so solutions need to scale also. Our very survival depends on our ability to work together more effectively, to get collectively smarter. Computers -- when used properly -- can help us do this.

Today, we celebrate Engelbart's accomplishments, but we forget his motivation. Computers should help us become smarter and work together better, and in many ways, they have. But instead of progressing, tool builders these days are moving in circles, treading water rather than swimming forward.

Good Enough Is Not Enough

One reason for this stagnation is that we seem to think we've reached the limits of what software can do for us and what we can do with software. Nothing could be further from the truth. Our software tools -- particularly in the collaboration space -- are nowhere close to fulfilling their potential.

Consider a basic collaborative task: document-sharing. A number of applications (both commercial and open source) claim to solve the document-sharing problem, and yet, the predominant method for sharing files is to email them back and forth. This is the computational equivalent of sneakernet. If the tools that purport to solve this problem are good, why aren't we using them?

We see similar problems in other basic areas. I can walk into any meeting anywhere in the world with a piece of paper in hand, and I can be sure that people will be able to read it, mark it up, pass it around, and file it away. I can't say the same for electronic documents. I can't annotate a Web page or use the same filing system for both my email and my Word documents, at least not in a way that is guaranteed to be interoperable with applications on my own machine and on others. Why not?

Solutions to these individual problems exist. Unfortunately, most of them are not very compelling, usually because they are not designed with people's needs in mind. On the other hand, some of them are very good. Unfortunately, being good is not enough. The reality is that our needs are broad and varied. We use a number of different tools to do our work, and we always will. In order to make a real impact in the collaborative space, tools must not only be good, they must be interoperable.

Improving collaborative tools, then, boils down to this: We must be people-centric when designing and building applications, and we must work with other developers to make our tools more interoperable.

People Come First

The natural starting place for building people-centric applications is the user interface. The opportunities to improve here are endless, and the increasing number of publications in this area show that people are paying greater attention.

The problem with usability is not a lack of good ideas; it's that most of these ideas never make it into real applications. There are many reasons for this, from organizational shortsightedness to the vagaries of the marketplace. As frustrating and as uncontrollable as these factors may be, the onus for changing the situation is on both the researchers who develop these ideas and the programmers who implement them. Open source software offers an excellent and underutilized avenue for disseminating innovations in user interface. Researchers should be writing plugins for widely-used open source applications, such as the Mozilla Web browser, instead of developing prototypes from scratch. Open source developers should be scouring academic publications for ideas, rather than simply duplicating the user interfaces in commercial products.

Being people-centric isn't just about user interface, however. It's about attitude -- how we think about our applications in general. The wrong attitude can steer people away from some very useful technology.

The Semantic Web is a perfect example of this. Fundamentally, the Semantic Web is about standards for knowledge representation, which is an important and worthwhile effort. However, the grand vision underlying the project is not people-centric enough. Evangelists suggest that by making human knowledge machine-readable, computers can help us in smarter ways. This makes sense, but it doesn't explain how that knowledge is captured and converted in the first place. We have a hard enough time expressing knowledge so that other people can understand. Is it realistic to expect massive numbers of people to express knowledge that computers will be able to unambiguously understand?

While the futuristic applications promised by the Semantic Web have failed to crop up, some of the resulting work is quietly making an impact in different ways. The Resource Description Framework (RDF) data model has proven to be an extremely powerful tool for representing information, and several applications -- including the RDF Site Summary (RSS) syndication format, the Mozilla Web browser, and the Chandler personal information management tool -- are starting to use it. More importantly, RDF is slowly changing the way many of us think about data. That influence is often reflected in the architecture of our tools, even if the RDF data model itself is not used. The danger is that the overarching vision of the Semantic Web is detracting from these smaller successes, things that are working right now.

A Shared Conceptual Framework

The second step towards improving our collaborative tools is to make them more interoperable. Some opportunities for standardization are obvious -- instant messenging protocols, for example -- and, in many of these cases, these efforts are already underway. There are, however, less obvious but potentially more significant opportunities for standardization. In order to identify these, we need to have a shared conceptual framework for thinking about collaborative tools.

There is much precedence for this in software. Today, we take things like filesystems and memory management for granted, but there was a time when operating systems did not exist. Now, we expect every computer to have an operating system, and while there are differences, the basic metaphors between different systems are the same.

Similarly, although there are several different relational databases, all of them rely on the same underlying conceptual model -- relational algebra -- and all of them understand SQL. The creation of this shared conceptual model for databases was one of the most important developments in the history of information systems.

A shared conceptual framework for collaborative software would provide a common vocabulary for thinking about and discussing these tools, and would also reveal opportunities for standardization. In order to create this framework, we need to identify the commonalities between different collaborative applications. This can be easier said than done, but we can use Engelbart's ideas as a starting point.

Backlinks: An Example

Consider backlinks, a concept in Engelbart's original hypertext system. Backlinks are links from other documents pointing to the document in question. For example, if someone were to create a link from their web site to this article, that link would be one of this article's backlinks.

Are backlinks a common feature in collaborative systems? Two applications immediately come to mind. The first is Google, the web search engine that determines the relevance of documents based on the number of backlinks. The theory is that the number of links to a document indicates its relative importance. In order to determine what those backlinks are, Google spiders the Web and has essentially constructed the world's largest backlink database.

The second is the WikiWikiWeb, or Wiki for short. Wikis are Web sites that anyone can edit via a browser. Because of their simplicity, Wikis are rapidly becoming popular as collaborative hypertext authoring systems. One of the best known Wiki sites is Wikipedia, an open encyclopedia that allows anyone on the Internet to contribute and that is more frequently accessed than the venerable Encyclopedia Britannica Online. There are a number of different Wiki implementations in a variety of languages, and most of them support backlinks.

A feature found in only two applications may not seem important enough to merit inclusion in a grand unifying framework, even if those applications are important and popular. However, if we examine other tools more closely, features that look very similar to backlinks seem to crop up over and over again.

Weblogs (also known as "blogs") are online journals consisting of multiple entries sorted chronologically. Blogs are commonly used to comment on other content, such as news items and other blogs. In practice, bloggers converse with each other by repeatedly linking and commenting on each others' entries. Naturally, bloggers want to know what sites link to their entries. In order to enable this, Ben Trott (creator of the popular MovableType blogging tool) invented TrackBack, an open specification for automatically informing bloggers of links to their entries. TrackBacks are backlinks.

Backlinks even appear in email, a collaborative tool to be sure, but not something typically thought of in hypertext terms. When you respond to an email, most applications record the ID of the email to which you are responding. That ID is equivalent to a link. If you want to see all of the responses to an email (to construct a thread view, for example), you essentially want to see the backlinks to that email.

Each of these backlink implementations were independently invented for the same reason -- the desire to see who is responding to one's work. As a common feature for facilitating this behavioral pattern, backlinks are also a potential candidate for standardization. A TrackBack-like system could be used for an entire web site, not just a blog. The same backlink engine that drives certain Wikis could conceivably be used by other applications as well.

Standardizing backlink databases could ultimately create more open forum systems. Many publication web sites incorporate forum software that lets readers comment on articles. However, URLs to interesting articles are often circulated in other forums. It would be far more valuable and interesting if publication Web sites could display links to commentary in all of the forums in which an article is discussed.

My point is not to evangelize backlinks (although I think they are a valuable concept) or to list all of the potential applications that could arise from standardizing backlinks (I'm certain there are many more). My point is that our collaborative tools are more similar than we may think. Developing a shared conceptual framework will help reveal those commonalities, which in turn will create opportunities for making our tools more interoperable, and hence more useful.

Shared Language for Modeling Documents

A shared conceptual framework will naturally consist of multiple components, no single one of which will necessarily take priority over another. Improving the interoperability of any of these components -- even if only in small, imperfect ways at first -- will improve our tools significantly. As a result, we do not have to agree on priorities upfront. We can collaborate in parallel on the components that interest us individually, and make progress on many different fronts simultaneously.

That said, one area of focus that would have a particularly significant impact would be to standardize the way we view, express, and manipulate structured data, especially documents. In other words, do for documents what relational algebra and SQL did for databases.

All collaborative tools generate data, usually in the form of a structured document. At the most basic level, all tools allow us to do the same things with this data -- create it, view it, and sometimes edit it. Today, it's clear that other basic tasks would be desirable in all of our applications as well.

Creating links, for example, would be a natural and desirable feature in all collaborative tools. Several developers have already recognized this and support linking in some form within their tools. However, linking is most valuable if it is implemented in an interoperable way among different applications. Your PIM application may already allow you to link a person's name in your calendar to that person's contact information within the same application. However, it should also allow you to link to that person's contact information in another application, to the email with directions to the meeting place, to the document containing the agenda, to the instant messenging transcript where you arranged the meeting, and so forth, all independent of the tools or file formats being used.

Another basic feature that would be useful in all tools is fine-grained linking. A single document often contains many nuggets of useful knowledge. It seems unfortunate to limit linking to the entire document rather than to the portions that are actually relevant. More importantly, the level of granularity should be configurable for each document type. It makes sense to think of an article in terms of paragraphs, sentences, and words. However, these granular constructs do not apply to software source code, whereas classes, methods, variables, and functions do.

The knee-jerk reaction to fulfilling these requirements interoperably is to convert all data formats into XML. XML, after all, is flexible enough to represent all sorts of information, and there are already standards for granularly addressing documents expressed in XML. However, this approach is both unnecessary and misguided. The syntax used to express a document is ultimately irrelevant. What's needed is a standard way to express and manipulate the fundamental constructs of a document, regardless of the syntax.

The lesson from XML -- as well as its predecessors, SGML and HyTime -- is that we can express all kinds of data as graphs. If there were a standard language for expressing graph-like data models (such as RDF) and standard APIs for manipulating these data models, the actual syntax of the data would be rendered mostly irrelevant. It would enable us to do things such as create a link from a word processing document to a function in source code without having to translate either document into some intermediate format.

Roadmap for the Future

All of the conceptual and technical ideas I've proposed in this essay share one thing in common: They won't make a difference unless tool developers work on them together. Creating a shared conceptual framework is a truly collaborative problem. It will not be solved by a single person in an ivory tower and forced upon the rest of the community. It will require constructive, passionate dialog, open minds, and much experimentation. It will require respect for other people's work and ideas. Most importantly, it will require a shared desire to make the world a better place by improving the way we work together.

With this in mind, these are the steps for improving collaborative tools:

* Be people-centric. This applies both to how we design our tools, and how we market them.

* Be willing to collaborate. We all belong to a community of like-minded tool developers, whether or not we are aware of it. Working together will both strengthen this community and improve our tools.

* Create shared language. Our tools share more similarities than we may think. Conversing with our fellow tool builders will help reveal those similarities; creating a shared language will make those similarities apparent to all. As a shared language evolves, a shared conceptual framework for collaborative tools will emerge, revealing opportunities for improving the interoperability of our tools.

* Keep improving. Improvement is an ongoing process. Introducing new efficiencies will change the way we collaborate, which in turn will create new opportunities to improve our tools.

Finally, never forget Doug Engelbart's fundamental tenet: Computers should help us become smarter and work together better. Remembering this will keep us on the right track.

Acknowledgements

Many thanks to Danny Ayers, Chris Dent, Doug Engelbart, Johannes Ernst, Richard Gabriel, H. Jessica Kim, and Justin Lin for reviewing earlier drafts of this article.

References

There are several papers that share the philosophy expressed in this manifesto and, in some cases, directly inspired it. The most important is Doug Engelbart's 1962 paper, "Augmenting Human Intellect: A Conceptual Framework." Chris Dent's paper, "The Computer As Tool: From Interaction to Augmentation," (December 7, 2001) builds on the ideas expressed by Engelbart and others, and provides an excellent framework for thinking about tools and augmentation.

Paul Prescod makes a powerful case for the importance of addressability in, "Addressing the Enterprise: Why the Web Needs Groves" (July 1999). Lee Iverson offers similar ideas and a potential solution in, "NODAL: A Network-Oriented Document Abstraction Language," (May 1, 2001) where he identifies relational databases as a precedent for what needs to happen with documents. Gregory Rawlins has incorporated a number of these ideas into his KnownSpace Symphony project, and has written several good papers describing the motivation and potential uses and ramifications of these ideas. My paper, "Interoperability Between Collaborative Knowledge Applications" (August 6, 2002) is a more technical synopsis of how a standard language for expressing documents as graphs can make our applications more interoperable.

I offer more commentary about the Semantic Web in my essay, "Do We Need the Semantic Web" (August 2, 2003).

Richard Gabriel's series of essays -- which falls under the moniker, "Worse Is Better," -- builds a strong case for designing evolvable, loosely-coupled systems.

Finally, Ken Jordan, Jan Hauser, and Steven Foster's white paper, "The Augmented Social Network" (May 15, 2003), presents a similar and similarly inspired vision for building collaborative tools, focusing largely on issues surrounding digital identity.