Cover Page

Title of submission: Guidelines are a tool: building a design knowledge management system for programmers

Category of submission: Design Practice Study

Name and full contact address (surface, fax, email) of the individual responsible for submitting and receiving inquiries about the submission: Mike Kuniavsky, 707 NW 19th Ave, Portland, OR 97209, +1 415.235.3468, mikek@orangecone.com

Guidelines are a tool: building a design knowledge management system for programmers

Mike Kuniavsky
Consultant
707 NW 19th, #302
Portland, OR 97209 USA
mikek@orangecone.com

Srinivas Raghavan
Staff Human Factors Engineer
Qualcomm
Wireless Business Solutions
Building L
5775 Morehouse Dr.
San Diego, CA 92121 USA

Abstract

This case study describes the creation of an internal design knowledge management tool for web developers as a means to encourage user-centered development practices. With a goal to shift a software development culture from waterfall-style to user-centered practices, the repository of knowledge and code is created as an incentive for programmers to create interfaces in a user-centered and consistent way.

Several experimental techniques are used in development of the tool. The process treats software developers as a user group and approaches the creation of design guidelines as if they were a product. In addition, the use of agile software development techniques, as driven by interaction and interface design, coupled with off-the-shelf blog software as a extensible, lightweight content management system makes this an experiment on multiple levels.

Results about the success of the experiment are still pending, but the authors are optimistic.

Keywords
Design Management, Interaction Design, Agile Design, Process Innovation, Program Management, User-Centered Design, Web Design

The Problem: waterfall development leading to design inconsistency

Qualcomm Wireless Business Solutions' (QWBS) web application software development team has been using a traditional waterfall process [10] from its inception several years ago. Initially, there were no user interface designers assigned to the team, and UI designs were inconsistent and often not to the desired level of quality. As designers were introduced, they were tasked to design the application user interface based on detailed product requirements documents that had been approved by a stakeholder team. Subsequently, the stakeholder team had to sign off the user interface design before the development team began working on their low-level software design document. The QA team then created test plans to validate compliance with the UI specification and product requirements documents. There was little or no user validation of the requirements or the design and no emphasis on consistency across designs.

Sporadically, as resources allowed, the design team conducted usability tests and user surveys to gather feedback on the current product and future designs. The entire process was highly document-centric and the UI design specification was managed under a change control process. The motivation behind this rigid process was largely to minimize iteration and maximize consensus from internal stakeholders. Involving the users was a desirable but not a required task of the design team.

As the single design team began to service the needs of multiple applications and members of the team were dedicated to specific applications, design resources became constrained. To create the interfaces required for all of the projects, programmers would continue to have to do UI design, as they had done before the design team was formed. The guidelines project started as an effort by the UI designers to maintain consistency and reduce the time required by developers to produce applications from UI design specifications. Initially, the goal was to minimize iteration, maximize specifications, and maintain the formal sign-off process. The original design guidelines was a CSS style sheet developed by a designer and used by developers whenever an element of the style sheet was referenced in the UI design specification document. As more developers worked with the design team, the Human Interface Guidelines (or HIG, as it came to be known) contents grew from a style sheet to include buttons and other graphic elements. Subsequently we added commonly used visual elements to display data such as tables, lists, detailed data presentation, maps, etc.

As the HIG project grew so did people's expectation. Was it "guidelines" or a mandate? How do you evaluate the "guidelines"? Who should own it? Who should review it? … As it grew we changed our thinking about the guidelines less as supports for the current process and more as tools for introducing iteration into the design process. This brought us to the point where we started on the new HIG project in August 2004.

The immediate goal was to support programmers doing detailed UI design and Quality Assurance teams evaluating completed products. The HIG was to distill and document standard UI elements and practices, to simplify specifications and to encourage standards that support the reuse of code and designs. A secondary goal was to produce a product that was flexible, immediately useful and an example of user-centered development.

Research

Early on, we recognized that our biggest challenge was the reluctance of developers to enthusiastically adopt design standards. There are many design documents that describe design "rules" and best practices. In our experience, however, in practice such guidelines are used infrequently, regardless of the quality of the recommendations. Writing yet another standards document in light of how programmers used (or didn't use) others seemed inefficient, at best.

We decided to treat the guidelines as a product and to design them in a user-centered way, with developers as our user market.

Guidelines are a product: competitive analysis

Our first goal was to understand why most guidelines seem to fail. Resource limitations did not allow for extensive research, so we decided to review the existing guidelines and analyze similar guidelines to understand what had been done, and we interviewed some developers to understand why they had failed.

We examined the current HIG. We determined that the document had some information architecture issues that made access difficult, but that it contained reasonable advice that, if followed, would produce decent interfaces most of the time.

Our goal then became to understand how to deliver this information so that developers would want to follow the guidelines. To do this, we compared the HIG to a selection of popular interface guidelines across a selection of UI-design disciplines. We examined guidelines from Apple [1], PalmSource [2], Sun [3], KDE [4], usability.gov [5] and remotecontrol.pbs.org [6].

Our survey showed a number of issues:

- *Guidelines are typically presented as rules.* Developers are intimately familiar with the fact that every development situation requires tradeoffs. When best practices are presented as immutable rules, it creates internal contradictions. Usability.gov [5] states "Establish a high-to-low level of importance for each category and carry out this approach throughout the entire Web site."
- Guidelines are often not written in developer-friendly language. Designers or technical writers write most guidelines from their own perspective, rather than developers'. For example, the Java guidelines [3] recommend that when designing icons, "Keep the drawing style symbolic, as opposed to photo-realistic."
- Guideline collections are long. It's not unusual for guidelines documents to run into the hundreds of pages. That may be required to document all of a system's interface functionality and it may be valuable to look up a detail or read about broad design philosophy, but it's daunting for a developer looking for a solution to a medium-sized problem..

- There's little incentive to comply. The documents themselves offer little incentive for people to read them and comply with their recommendations. There is generally some text in the preamble about why compliance is a good idea, but the rest of the document speaks only of painful necessities.

Developers are users: interviews

We next went to the source, turned our user research techniques inward [17], and conducted interviews with experienced developers. These took the form of phone calls, questions to mailing lists and a lunch. The interviews, although unstructured, showed a consistency in terms of identifying developers' frustrations with guidelines.

Our interviews reinforced our impression of a systemic problem with UI design guidelines. In general, developers didn't really use a lot of UI guidelines. They regularly refer to documentation to get information about how to do something technically, but not how it should be designed to work for the end user. Once it works, it's considered done.

On a mailing list [7], a programmer listed his preferences for UI development, which summarized the points that a number of other developers had made:

1. *Give the problem to someone else*
2. *Give me a way to auto generate the UI*
3. *Give me example code so I can cut and paste*
4. *You design it in a way I can understand and I will implement it* [15]

Underlying this list is the idea that creating a good UI is not his primary job (which it isn't), and he doesn't want to have to worry about it, but if he does, then he wants it to be as little additional work as possible.

Conclusion: Guidelines-as-rules are extra work, tools are better

"All developers building applications for Mac OS X should read and become familiar with the contents of this document."
—Apple Human Interface Guidelines [1]

"Ignore the guidelines, and you invite user frustration and confusion."
—PalmOS Guidelines [2]

Guidelines which are rules tend to be underused. There was nothing inherently wrong with the guidelines the various documents presented, or even how they were written. But rules need to be followed at all times. Rules are extra work. Software development is always behind schedule, and additional work is going to be resisted.

In our own experience, when developers are *forced* to use them, such guidelines do not produce particularly good solutions and tend to be followed literally and superficially, at best.

Furthermore, guidelines creators do not treat developers as users. This creates a several systemic problems in the guidelines documents:

- They fail to address the day-to-day needs of software developers
- They grow obsolete, since there's little incentive to update them by the people who are their audience

- There's little incentive for compliance
- They are treated as reference documents, rather than being integrated into the development practices and workflow that would encourage their adoption
- They are written as textbooks for interface designers

In other words, they're not tools that are designed for the usability of their target audience of developers. The quote that starts this section is an example of an unreasonable demand that a guidelines document places on its developer audience. Developers are busy people and memorizing the details of Apple's 300-page document is unlikely.

Based on this review, we came up with some "guidelines for guidelines," which are available in Appendix A.

A final observation is that toolkit-based guidelines appear to work better than lists of rules. Apple has a lot of success getting developers to create consistent interfaces by backing up their rules with a toolbox of interface elements and development tools ([8] and Figure 1). PBS does this, also (see [6]). These make it easier to create decent Macintosh interface designs than to create bad ones.

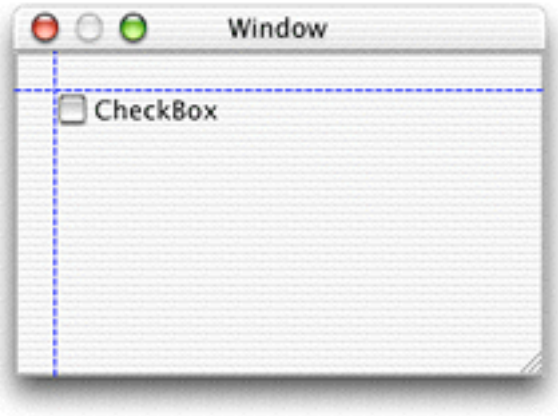


Figure 1. Apple's Interface Builder's "Snap to Aqua Guidelines" feature [8], showing correct placement of a checkbox relative to edges of the window.

This insight led us to think of the guidelines that we were building as a kind of tool, and to focus our efforts on creating something that felt more like a tool than a document, more like an assistant than a headmaster.

Solution: a design knowledge management tool for developers

Our goal was to make development more consistent by creating incentives, rather than constraints; to make following guidelines easier than not following them.

Based on our research conclusions, we decided that the governing principle was to create a *design knowledge management tool for developers*. We decided it would have the following qualities:

- *A knowledge repository.* It's the central place for knowledge about UI design within the constraints of QWBS's users' needs and how to solve them.

- *A tool.* To make it a tool, it needs to support the developers' workflow, rather than creating additional work.
- *Highly functional.* The tool needs to focus on core tasks, specifics and immediate solutions, not abstract principles.
- *Flexible.* We didn't know what would work for developers and didn't want to be too attached to our vision of the perfect solution. This was our first shot at solving this problem, so the tool needed to be easily tuned to what would actually work.

We wanted developers to use the tool immediately and modify it as needed. To do this, we decided it would:

- Provide boilerplate HTML, CSS, ASP and JSP code
- Provide details, examples and context to clarify interaction design documents and ideas
- Be QWBS-specific and document local UI design decisions and practices, rather than replicating content that existed elsewhere
- Allow users to add to and expand the knowledge contained in the HIG by themselves
- Make printer-friendly versions of all pages

Information Architecture

As a first pass, we wanted to focus on reorganizing existing content and creating ways of enhancing its value, rather than creating new content.

The original HIG had been organized into two main sections, Templates and Topics. We thought that "Topics" was too generic. We went through the existing content and grouped it into clusters that we felt fit developers' needs, then named the clusters (see Figure 2). The four primary sections are:

- *Components.* We defined these as groups of basic UI elements that together make up a single unit of functionality from the perspective of the user. For example, drop down boxes, buttons and form fields would not be components in themselves. The search box—a group of drop-downs, form fields and a button—is a component because it defines a single task from the user's perspective.
- *Templates.* Carrying over an idea from the current HIG, these are examples of UI designs using Components, with the components called out and linked to the extended descriptions.
- *Practices.* These are QWBS-specific ways of handling functionality that cross a number of templates or components. Error handling, validation and the Qualcomm-specific meaning of different colors are examples of practices.
- *Design principles.* These are general principles of good user interface design, and are not Qualcomm-specific. For example, "the use of color in UI design" would be an appropriate design principle, coupled with a link to the Qualcomm-specific color practice mentioned above.

In addition, each document type was divided into its own content sections (see Figure 3). For example, Templates pages had sections for:

- *Associated Files.* Links to code that's either used in the template or produces the template.
- *Related Templates.* Links to alternate or important supplementary templates.
- *See Also.* Links to relevant Practices and Design Principles.
- *In Use.* Links to examples of this template, or something similar, working in a QWBS product.

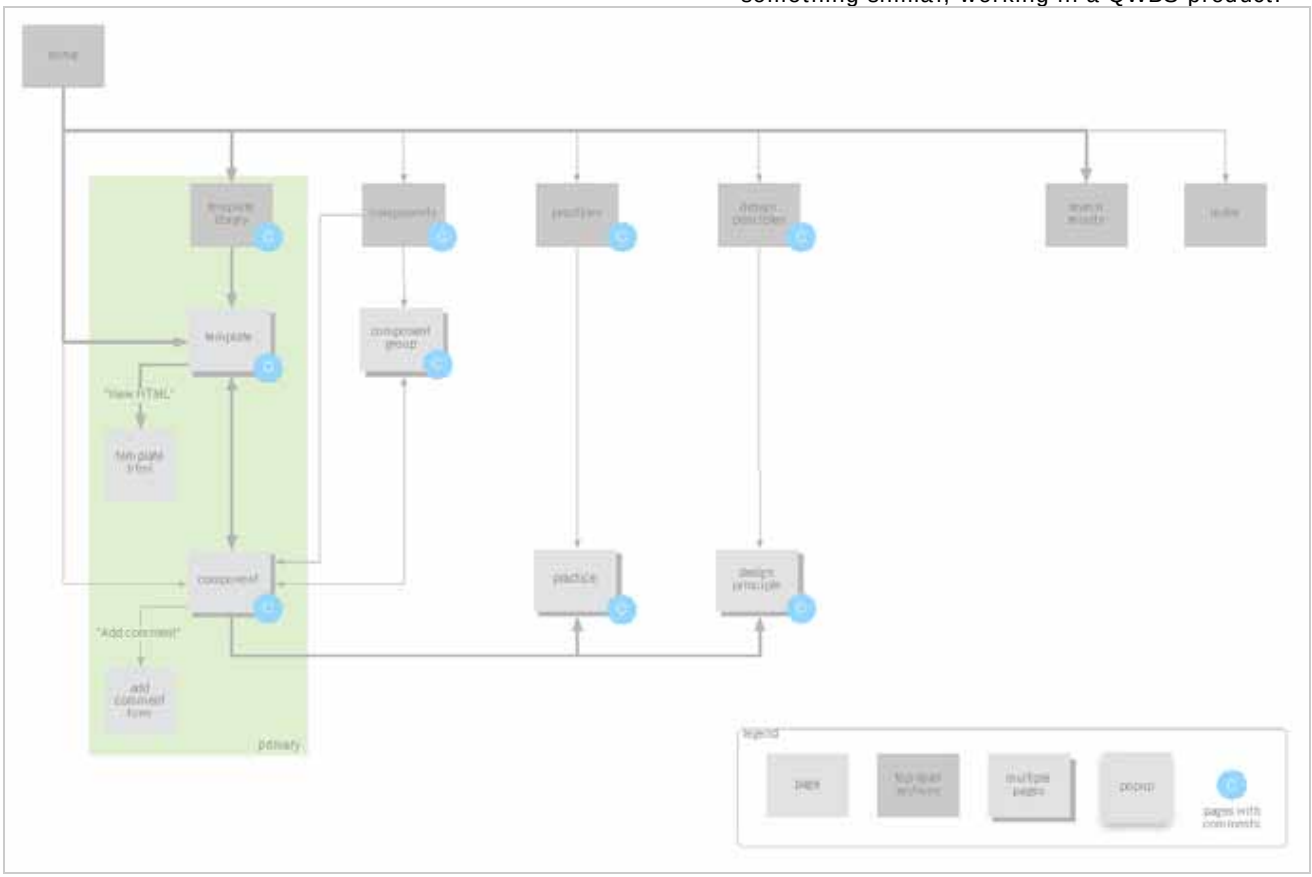


Figure 2. QWBS HIG Information architecture. Diagram by Nadav Savio of Giant Ant Design

Interaction Design

The site was supposed to be a familiar, and easy, web experience. To facilitate this, we used several common design conventions:

- Left-hand navigation [18], with main navigation on the left, in order of decreasing importance, set off with a different background color, and a top bar with a title and search field.
- One level of hierarchy that appears indented in subsections, with the current section highlighted and non-clickable. [20]
- "Home" always visible on the upper left,
- The most important information is above the fold to the right. [21]
- Comments at the bottom of every page, except the front door.
- Printable versions available from the upper right corner.

Figure 3. HIG template wireframe. Diagram by Nadav Savio or Giant Ant Design.

Process: Agile design

A philosophical cornerstone to the development of this project was the use of *agile development* methods [9]. *Agile* is the name of label describing a family of software project management and development practices that attempt to address problems with design-first/build-later software development, (such as the waterfall model) described earlier. Agile development does not require research and design to be completed or a detailed paper trail and explicit signoffs before development starts. Instead, agile methods focus on extensive communication and rapid iteration, knowing the goal, but continuously collecting information adjusting to it, rather than planning the entire process in advance

There are a number of agile methodologies: Extreme Programming [11], Scrum [12], the Crystal Methods [13], etc. We chose to use a set of practices adapted from Extreme Programming (XP). The practices we used can be grouped into three broad categories:

- Rapid iteration
- Focus on user goals
- Continuous communication

Rapid iteration: one-week cycles and minimal problem solving

We worked in one-week cycles. We made many small changes and integrated them continuously into the product. This allowed us to always have a product that was always functional, even if not all of the features ware hooked up or worked as designed.

We also designed quickly, creating much of the design in collaborative design sessions around a white board. Based on these, we created mockups and walked developers through them.

Starting a couple of weeks into code development, there was always a functional version of the software that was accessible to everyone involved, with an ever-increasing set of functionality. One of the first pieces of functionality we built, for example, was the ability to enter content into the system. The entry screen was basic and users had to use a markup language, rather than fill-out forms, but people were able to start entering content and providing feedback almost immediately.

Embracing another agile practice, we focused on doing the minimum necessary to satisfy functionality goals. Rather than trying to solve the general case, we solved for the immediate problem. For example, although we intended to have different permission levels (for people creating core content versus those using it primarily as a reference), at first we only had one account and trusted that everyone would respect each other as they used it. They did. We made other account levels eventually, but this saved time, put the product into users' hands earlier and allowed us to see what permission structure really made sense. We also kept this in mind when creating our information architecture, which was designed for the content we knew we were going use, rather than an arbitrarily large set. If we had much more content than we had bargained for, then that would be a sign of success and we could address the issues then.

Focus on user goals: stories and prioritization

We interpreted every decision in terms of its effect on user goals using two practices:

- All functionality was defined from the user's perspective
- Weekly prioritization

All functionality was described in terms of user stories. XP defines user stories like this:

> *Each User Story is written on a Story Card, and represents a chunk of functionality that is coherent in some way to the customer.* [14]

XP recommends using paper index cards, one per story, but our story cards were lines in a spreadsheet that we passed back and forth. Figure 4 is example from the week of November 15, 2004, about a month into programming.

QWBS HIG User Stories

Next story 73

Story #	Name	Description/Notes	Points
7	~~User adds note~~		0
8	~~Editor deletes note~~		0
13	~~Editor gets email update of added note~~		0
9	~~Editor adds content to existing component~~		1
49	~~Editor adds component with HTML description~~		2
6	~~User views template~~		1
19	~~User views a list of all templates~~	Index of all template-type entries	2
11	~~Editor adds Practice~~		1
45	~~User reads practice~~		1
		10/18-10/24 Total	**8**
17	~~Editor adds template~~		1
5	~~User selects template from homepage~~	Links initially, then thumbnails	2
54	~~User views homepage with links to all templates and all components~~	An intermediate step between not having any links on the homepage and having a selection of some chosen templates and components	2
		10/24 - 11/1 Total	**5**
12	~~Editor adds Design Principle~~		1
72	~~Editor adds only without bugs that time found the week of 11/1-11/8~~		3
		11/1-11/8 Total	**4**
2	User selects components from list of components on template page	Is there just a static list of components or is it somehow automatically generated?	3
58	User follows link to related component from template	Same as #2	0
59	Editor references component with basename	Same as #2; this is the technique to achieve to #2.	0
55	User views site on internal Qualcomm server		2
10	User looks at associated file	Link or Popup	1
29	User follows "in use" link to a site that uses component		1
30	User follows "in use" link to a site that uses template		0

Figure 4. QWBS HIG User Stories.

In the spreadsheet, "name" is the user story. We used two broad user classifications—User and Editor—to describe functionality. Whenever any functionality was discussed, we always tried to describe it from the user's perspective, and every idea for functionality was captured as a story. This helped bring perspective to solutions for which there was little need: if there was no good way to justify it in terms of a reasonable-sounding user story, then we hadn't defined the functionality enough. For example, one suggestion was to implement a markup language for defining different classes of content presentation in a broadly flexible

way. It sounded interesting, so we wrote two user stories for it: one for the broader idea and one for markup that described our immediate needs. The immediate story got prioritized high early on and implemented. We've never missed the broader functionality.

Admittedly, the process occasionally produced funny-sounding entries, but it was overall an excellent practice to focus on appropriate functionality and to make sure that no ideas were lost.

The second practice, prioritization, was more straightforward. We discussed user stories every week. Based on what had been accomplished, what problems had been encountered, and what new ideas had been described, we would decide what had to be done the following week. In the spreadsheet above, weeks are delineated with green horizontal bars. This did not create the typical infrastructure-first flow of features for the HIG programmer because it emphasized the user-experience over traditional programming practices, but it allowed us to focus on the most important end-user features first. For example, we started by working on the display of the key pages before the backend database that would generator those pages. This can be compared to painting the kitchen before the roof is installed, but we're not building a house. Early emphasis on core user functionality gave us something we could immediately show the end-users to verify that we were on the right track.

Continuous Communication: IM and velocity

Traditional XP practice requires everyone on a team to be in the same room and that all programming (and, by extension, design) be done in pairs. Our team was distributed all over the world, so this was impossible, but excellent communication was still critical.

We communicated with instant messaging and weekly conference calls. We always had instant messaging running, included each other in our buddy lists, and messaged frequently throughout the day. These chats allowed us to make small corrections and clarifications, to exchange and collect functionality ideas and questions, to reprioritize in between the weekly calls, and to make major judgment calls on the fly in several cases.

Another XP communication and project management practice that proved useful was the measurement of velocity. Velocity is a planning tool that measures how much gets done in a typical week to help a group of people understand their own workflow. Velocity is not an idealized performance metric such as a man-hour or the number of lines of code. It's calculated for every development team and for every project that team does.

We gave every user story a point rating from 1 to 5 that represented how much effort we estimated it would take to implement it. 1 meant very little effort and 5 meant a lot of effort (occasionally there was a 0 when the story was covered by other functionality, though we still listed the story, since it was part of the user experience). These values are documented in Column D of the spreadsheet above. Then we let development happen.

At the end of each week, we added up the points for the user stories completed that week. After a while, we got better at estimating points and how many points represented a reasonable amount of work for a typical

week. This greatly helped prioritization, since everyone was able to see whether a story was doable based on what else was on the schedule for that week. There was rarely a question of trying to squeeze things in: we averaged about 6 points per week, and if something took 3 points and we were already at 5 points, it wasn't going to happen until the week after, unless some other story was removed.

One way of helping maintain velocity was to avoid reinventing the wheel. This led us to base the HIG on Movable Type blog and lightweight content management software [19]. It provided enough functionality and flexibility that we could use its comment and search facility immediately and rapidly build custom content templates. We also used several open source Perl libraries.

Results

This is a work in progress and an experiment. Although the original focus of the HIG was to minimize iteration in the software development process (i.e., between the Requirements, Design, Development and QA teams), it became part of a broader effort to *break down* the waterfall method of interface design and support a broader effort to encourage user-centered development. As this is really a deep cultural shift, we tried to set realistic expectations.

Current impact is minimal, since teams started using the product only in the spring of 2005. We anticipated that adoption would be slow and use of this tool will have to be seeded to teams that were willing and able to try it. Some eager development teams are unable to use the tool because the guidelines content are too different from the applications they're currently developing. Implementing the design guidelines on one or more projects is our next step and we are endeavoring to make the guidelines worthwhile for more development teams to adopt.

We are in the process of conducting usability studies on the current design to determine how well the interface works. We're getting positive feedback from developers, but the most positive sign that there's core value to the tool-based approach is that another development team—not the one we had initially chosen to use the HIG—are enthusiastically using our blog software and templates to create a technical documentation reference tool.

Lessons learned

Cultural change is hard. It's even harder than we had anticipated. We learned many lessons from this project. Here are some:

1. Document maintenance needs to be built into the process. From the beginning, we knew that it would be an issue, but we didn't know how much. To manage it, we have created a dedicated "HIG manager" position and assigned a resource dedicated to this effort.
2. Constraints can be useful. There are always resource constraints in any project. In this case, treating this effort as an experiment and self-imposing time and functionality constraints proved valuable. The constraints forced us to prioritize higher the features that created the most user experience value, and made it easier to justify the scope of the project to management.
3. Agile development philosophy, applied to user experience development, helps scale expectations

with all stakeholders and to focus on the most important features early in the process. However, Extreme Programming does little to define the user experience, so we had to continuously improvise based on the goals of the process, rather always just following the XP practices literally.

4. Our biggest challenge was keeping our minds open about the purpose of the HIG and reminding ourselves that it's not a solution. It's a tool that supports a set of development practices. We regularly reminded ourselves that there were many parts to that process.

In general, we're satisfied. The process was straightforward, effective and repeatable. The creation of content documenting the current design standard was also a good place to start and could also easily be easily repeated with other designs. However, having a more defined evaluation plan from the beginning could have been valuable. The process allowed us to create an infrastructure and practice in implementation of the tool. The core issue of ensuring that programmers design interfaces in a user-centered way was not solved, but the tool encourages developers to design consistently, which we consider a step in the right direction.

Next steps

The HIG is primarily a new method of accessing status quo UI designs, which are in need of revision. Now that the tool is done, the content needs to be upgraded before being applied to any new products. After we complete a revision of the content and create a HIG 2.0 based on the visual design of a new product currently under development, we will roll it out to select development teams. We will iterate on the design of the HIG based on their feedback and on planned formal evaluation of both the 1.0 and 2.0 versions. Subsequent iterations will include revisions of key interaction design components and further revisions of the HIG.

References

[1] Apple Computer, *Human Interface Guidelines*, http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html

[2] Ostrem, Jean, *Palm OS User Interface Guidelines*, PalmSource Inc., 2003

[3] Sun Microsystems, *Java Human Interface reference*, http://java.sun.com/developer/techDocs/hi/, accessed June 30, 2005

[4] KDE Usability Project, *KDE User Interface Guidelines*, http://usability.kde.org/hig/current/, accessed July 1, 2005

[5] National Cancer Institute, *Research-Based Web Design & Usability Guidelines*, http://usability.gov/guidelines/, accessed July 1, 2005

[6] PBS and Adaptive Path, *Best Practices for PBS Member Stations*, http://www.pbs.org/remotecontrol/bestpractices/

[7] Agile Usability (mailing list) http://groups.yahoo.com/group/agile-usability/, accessed July 1, 2005

[8] Apple Commputer, *Interface Builder*, http://developer.apple.com/tools/interfacebuilder.html

[9] Wikipedia, "Waterfall Model," *Wikipedia: the free encyclopedia,* http://en.wikipedia.org/wiki/Agile_software_development, accessed June 27, 2005

[10] Wikipedia, "Waterfall Model," *Wikipedia: the free encyclopedia,*

http://en.wikipedia.org/wiki/Waterfall_model, accessed June 27, 2005

[11] Beck, Kent and Andres, Cynthia. *Extreme Programming Explained: Embrace Change* (2nd Edition), Addison-Wesley, ISBN 0-321-27865-8, 2004

[12] Schwaber, Ken and Beedle, Mike, *Agile Software Development with SCRUM*, Prentice Hall, ISBN 0-321-27865-8, 2001

[13] Cockburn, Alistair, *Crystal Clear: A Human-Powered Methodology for Small Teams*, Addison-Wesley, ISBN 0-201-69947-8, 2004

[14] Cunningham, Ward, "User Story," *Portland Pattern Repository*, http://c2.com/cgi/wiki?UserStory, accessed June 30, 2005

[15] Borlin, Phil (paraphrasing O'Byrne, Brian), *Agile Usability* (mailing list), http://groups.yahoo.com/group/agile-usability/message/339, August 10, 2004; accessed July 1, 2005

[16] W3C, "Cascading Style Sheets, Level 2 Revision 1," http://www.w3.org/TR/CSS21/, accessed July 1, 2005

[17] Kuniavsky, Mike, "Reverse the Polarity!" talk for New Paradigms in Using Computers conference, IBM Almaden Research Center, unpublished (available from author), 2003

[18] Nielsen, Jakob, *Designing Web Usability: The Practice of Simplicity*, Indianapolis: New Riders, 2000

[19] Six Apart, *Movable Type* (software), http://www.sixapart.com/movabletype, accessed July 1, 2005

[20] Czerwinski, M., Larson, K., and Robbins, D. (1998). "Designing for navigating personal Web information: Retrieval cues," *Proceedings of the Human Factors and Ergonomics Society 42nd Annual Meeting*, 458-462.

[21] Nielsen, Jakob, *Homepage Usability: 50 Websites Deconstructed*, Indianapolis: New Riders, 2001

Appendix A: Guidelines for Guidelines

Based on our review, we came up with some "guidelines for guidelines". In our view, in addition to all of the other best practices for internal documents, UI guidelines need to be:

1. *Designed for developers.* Programmers are the users of guidelines. The structure of a tool for them should be built around their needs.
2. *Focused around tasks, rather than design elements.* The practice of writing software is different from that of designing interfaces, and should be reflected in how documentation for it is structured.
3. *Specific, not principles.* Programming is an applied art, and specifics address developers' needs better than theories. Examples that resemble the current situation make it easier to understand the theory and make applying the guideline easier.
4. *Prioritized.* Design and development is a web of choices, and explicit prioritization helps make some of those choices. Not all guidelines have equal impact.
5. *Succinct.* Extra words won't get read and supplementary diagrams will not get examined when the reader is in a hurry, and the reader is always in a hurry.

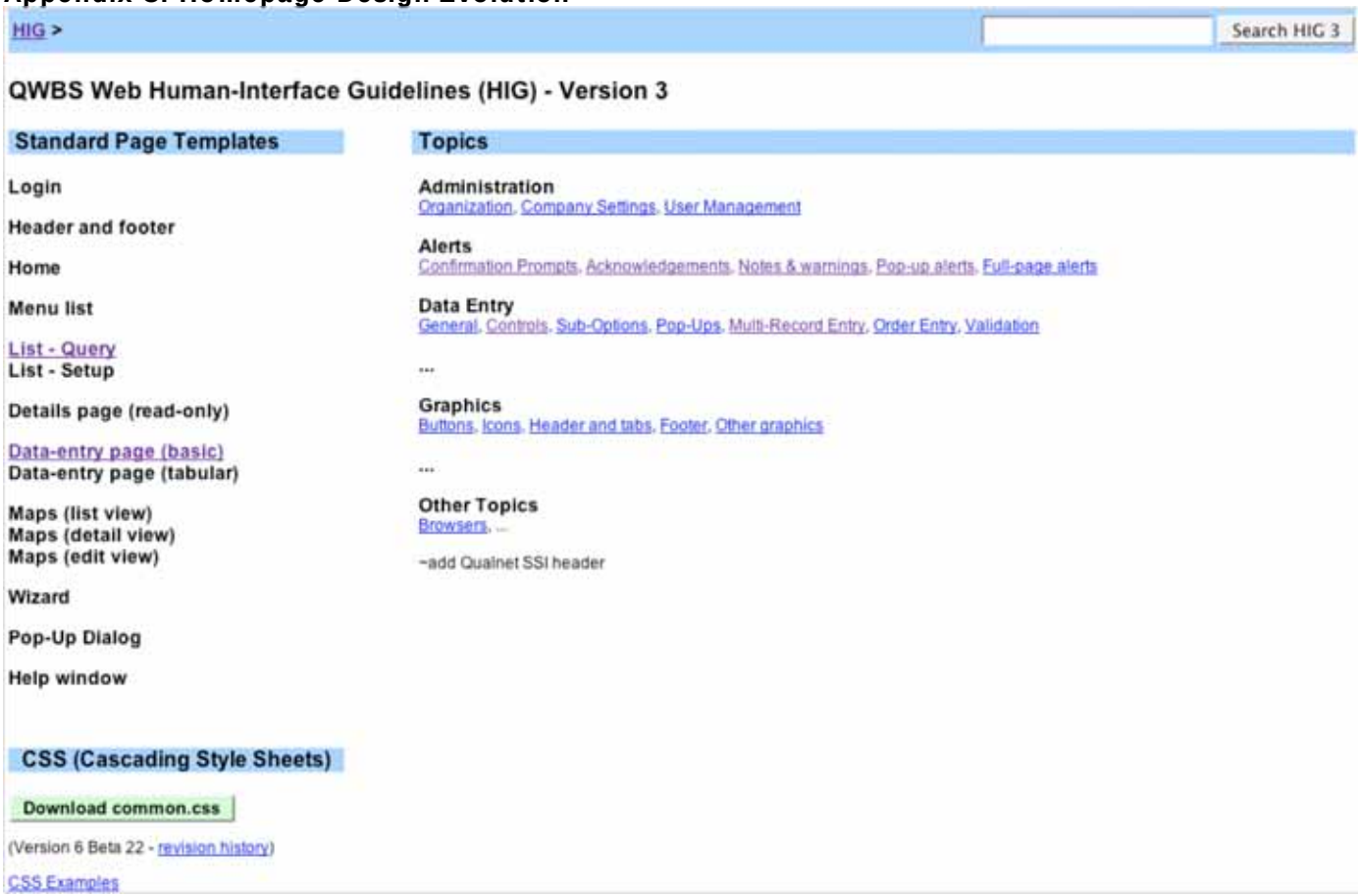
Appendix B: Development team

Our core team consisted of a visual designer, an interaction designer/project manager, a programmer, a content specialist and a project lead. The first three roles were filled by contractors (Kuniavsky was the interaction designer/project manager), the latter two by Qualcomm employees (Raghavan was the project lead). Nadav Savio of Giant Ant Design as the visual designer and production specialist, Tim Appnel of

Appnel Internet Solutions was the core programmer and Dave O'Brien was Qualcomm's content specialist, who created most of the content for the site.

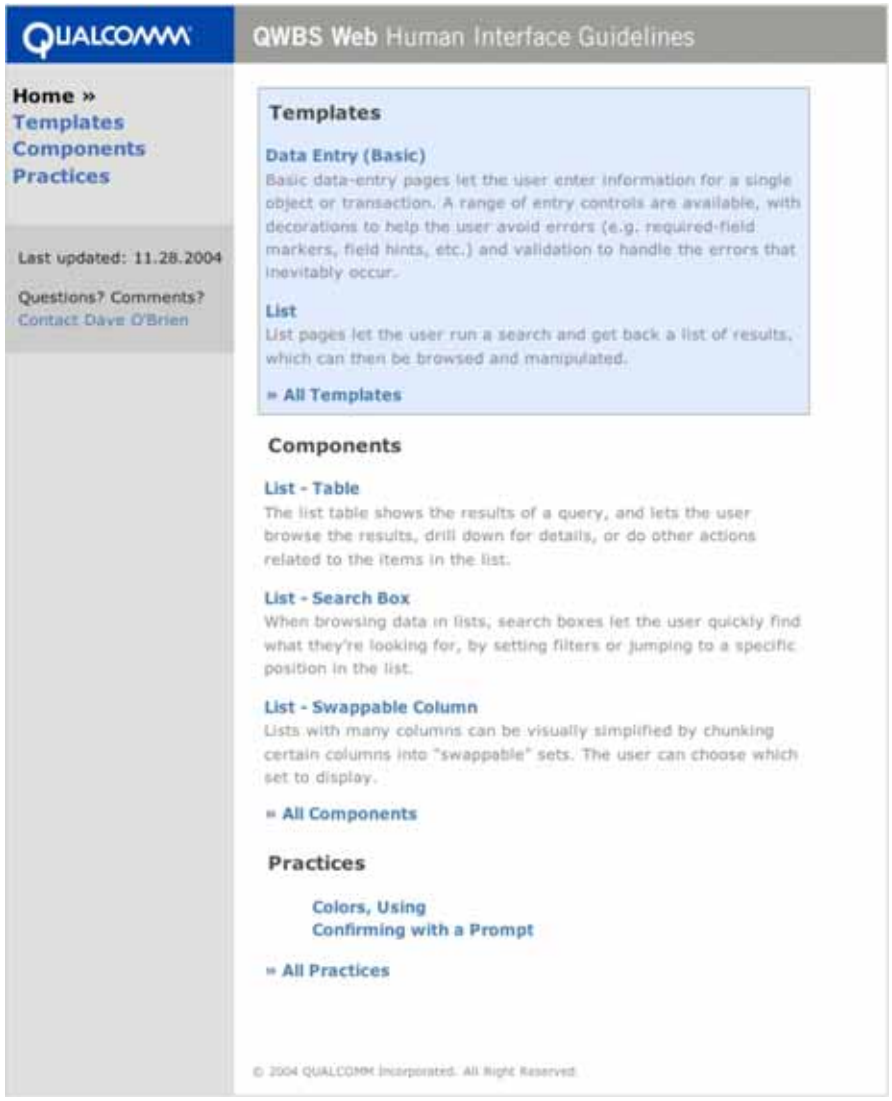
Contact info:
Nadav Savio, <nadav@giantant.com>
Tim Appnel, <tim@appnel.com>

Appendix C: Homepage Design Evolution



Original. The last version of the guidelines before the start of this project.

Wireframe. The wireframe front door, designed by Nadav Savio of Giant Ant Design.



Front door: In development. This is essentially how the tool looks now.

Acknowledgements

Tim Appnel, Nadav Savio, Dave O'Brien, Neeharika Gupta, Patricia Luke, David Volpi, Craig Lauer, Joan Waltman