

TABLE OF CONTENTS



Forward by Jakob Nielsen	3
Executive Summary	5
Background	10
Brief Description of Methodology, Participating Companies, Applications and Development Teams	11
Overview of Numerical Rankings	14
Detailed Description: Best Team Practices for Designing User Strategy, User Experience, and User Interface for Web-delivered Applications	
Team Dynamics	19
Core Team Roles	28
Typical Timeline	55
Determining Business Requirements	56
Gathering User Requirements	60
Designing User Strategy	72
Designing the User Experience	79
Designing the User Interface	95
Documentation	104
Tips on Outsourcing	110
Checklist and Best Practice Priorities	
Team Dynamics	114
Core Team Roles	115
Determining Business Requirements	115
Gathering User Requirements	117
Designing User Strategy	117
Designing the User Experience	118
Designing the User Interface	118
Documentation	119
Appendix	
Author Bios	121
Study Methodology	122

EXECUTIVE SUMMARY

The Tristream study team researched, evaluated, and defined the best practices of development teams that result in highly usable and highly productive user strategies, user experience, and user interfaces for web-delivered applications. Applications that were developed using a significant number of these best practices resulted in higher user satisfaction, ease-of-use, productivity, accuracy, efficiency, customer service satisfaction, reduced employee needs, reduced training costs, reduced help desk time, and bottom-line ROI.

Differentiating Practices

Our goal was to identify the practices used by teams that “stood out” from the rest. We did not try to catalogue all the practices in use by development teams—even when those practices are essential to development, such as many project management practices. Instead, we looked for practices that clearly resulted in superior application development and which were absent from, or poorly executed by other teams.

The best practices we identified range from team structure to team member skill sets, from user strategy development techniques to wire-framing, from internal documentation to establishing ongoing user groups, from prioritizing business requirements to working with offshore teams. In all cases, the best practices we identified, like cream, rose to the top.

“Core” Teams

In this, the first version of our best practices study, we concentrated on “core team” practices. Core teams vary in size from 5 to 15 people. The core team is hands on and works directly with a specific application, or portion of an application, and is responsible for the development of all the screens required, down to the last pixel. Frequently the “core team” hands off the programming to a larger group, often off-shored.

There are frequently multiple core teams working on the same application in large application development projects. In the next version of this report, we will address the best practices for managing and supporting multiple teams working on the same application. However, in this report we do not have sufficient information to address the coordination of multiple teams other than with an observation here and an observation there.

However, we will say without reservation that it is the core team that really insures great application development. All the coordination in the world cannot offset poor work in the core team—and vice versa, a great core team can even rise above some of the problems created by poor coordination.

So whether your company has an IT department under a hundred people, or in the thousands of people, you will find the best practices identified and described in this report to be very applicable to your core teams.

Off-shoring

Many of the best practice descriptions address specific ways to work well with off-shored elements of the development process, particularly the programming team. Off-shoring appears to be here to stay and the best teams have adapted to the opportunities and problems presented by an offshore team.

Challenges and solutions center on communication and cultural understanding, which are sometimes intermixed. Offshore programming teams in India, for example, are not comfortable with the directness of the typical

American conversational style, and will often take a passive role in the process as a result. The best teams make an effort to draw out their counterparts' good solutions, often sending team members to India on a regular basis.

On the other hand, offshore teams need to send key Team Leaders to their client facilities on a regular or long-term basis. Only then can the offshore team gain a real feel for the users and business needs around which the application is being developed.

Thorough communication is essential. Offshore teams generally need a more specific design specification than an internal programming team sharing the same facilities. This report addresses many best practices that alleviate many of the problems associated with off-shoring, including a special section, "Tips on Outsourcing."

New Applications and New Releases

Most of the best practices that have been identified in this report best apply to the creation of new applications and to the process of developing major new releases of existing applications. We focused on how the best teams applied their methods to applications where there were as few pre-existing limitations as possible. A clean-slate start, or a major new release (new version) of an application affords the maximum opportunity to implement greater usability and productivity.

However, the best practices we identified are also applicable to making incremental changes between major releases by recognizing pre-existing limitations and by a common-sense application of these best practices. The same principles apply, but typically a team isn't able to go as deeply into changing user strategy or the user experience as they might like to do when involved in incremental change. When making incremental changes, teams typically need to preserve the overall user experience conventions that are already in use throughout the rest of the application, otherwise the user will have an inconsistent and confusing experience.

Differentiating Between User Strategy, User Experience, and User Interface

The application development industry tends to treat user experience development (UE) and user interface development (UI) as the same process, sometimes using the acronym UE/UI. We have found that to be emblematic of many of the problems we routinely encounter—teams do not recognize or make distinctions between what should be separate phases or separate processes in the development process. As a result, teams often develop user strategy, user experience, and user interface elements out of sequence—or in the case of user strategy—not at all.

There is an old story (told with many variations) of a science professor filling up a glass jar in front of his students without any explanation. First he fills the jar with two or three large rocks. Then he adds as many medium-sized rocks as he can fit around the big rocks, then he fills up all the remaining space he can with fine rocks and sand, then finally he adds water right up to the very brim. Then he asks his students, "What did you learn from this experiment?"

Many students offer answers such as, "You can always get more in the jar," or more metaphorically, "You can always do more with your time."

Finally the professor, acknowledging the truth of what his students offered, says, "All that may be true, but the most important thing you learned was that you have to get the big rocks in the jar first."

Using this analogy, user strategy corresponds to the big rocks. User strategy typically addresses big picture considerations such as, where does this application fit in with the rest of the company's information systems? Is it part of a portal? Does it serve vendors and customers as well as internal employees? What is the basic "flow" of the application? Does it mirror internal divisions, such as sales, accounting, etc.? Or does it mirror workflow, such as initial call, proposal creation, order confirmation, fulfillment, follow-up? Is the application primarily for new users? Occasional users? Power users? Will the application have more than one "track," or have significant user-preference settings?

If these overarching user strategy decisions are not consciously made in the beginning of the development process, user strategy will be determined almost randomly by a series of other decisions—and it is rarely clear, usable, or productive as a result. As the old saying goes, "If you don't know where you're going, any road will get you there."

The best teams address user strategy first and make it a distinct step in the process. Your user strategy will probably need to be adjusted as the development process unfolds, and as new needs or limitations emerge, but the core strategy should be strong enough to survive such adjustments without losing clarity.

User experience development flows from user strategy and corresponds to the medium-sized rocks. Once overall user strategy has been decided upon, the best teams address themselves to understanding their users' task and workflow. The best applications make the users feel as if the designers knew exactly what they want to do next. In addition, the best applications recognize distinct user types and allow them to perform their tasks without any confusion. New users, or occasional users, will not have an agreeable user experience in an application designed for everyday power users, and vice versa.

There are no hard and fast "best user experience" conventions. If a team were to put all the best conventions together in one application, it would be a confusing mess. The "best user experience" is one that is clearly consistent and matched well with the specific needs of each user type.

Great user experience design is born of an intimate knowledge of business needs, the users' needs, and a robust, iterative wireframe design process. We did not find a single excellent, or even good, application that was not first designed visually. A user's experience is primarily visual. What he or she sees on the screen is the basis of his or her experience.

User interface design flows from user experience design and corresponds to the sand and water that go into the jar last. The user interface is made up of the visual elements seen on the screen—colors, shapes, banners, headers, modules, typography, icons, buttons, and controls. The user interface "clothes" the user experience in an easy-to-see, easy-on-the-eye ergonomically developed toolkit of interface elements.

The basic development flow—user strategy, to user experience, to user interface—insures that the most important decisions get made first and then cascade down to the smallest decisions.

Early in the Development Lifecycle

This study focused on the elements of the development lifecycle that had the most influence on creating highly usable and highly productive applications. We found, not surprisingly, that the initial development stages have the

most to do with creating great applications: determining business requirements, gathering user requirements, designing user strategy, then user experience, then the user interface elements and, throughout the process, good communication, and documentation.

We did not spend significant time studying acceptance-testing, IT implementation, QA, user-training, initial launch, and other very important steps to getting an application developed, because we found that these activities, falling late in the process as they do, had a diminishing impact on the final quality of the application.

The most important work—as far as usability and productivity gains are concerned—happens in the early stages of the development process. If you don't get things right early, it is very hard to do much about them later in the process.

Across All Types of Applications

The best practices we identified and describe in this report work for any web-delivered application. We included both publicly accessible Web sites, employee-only “intranet,” and business-to-business applications in our study. The best practices would, however, be applicable to any process involving a screen, whether a full-sized computer monitor, or a small, handheld screen.

The applications we studied are all on the critical path of each company's service delivery or internal operational processes or, where directly accessible to consumers, are central to the company's revenue generation. The majority of applications under study are internal and/or B2B applications and therefore our study results are strongly applicable to teams engaged in the development of these types of applications.

However, the development teams that were engaged in the development of B2C applications, and who employed these best practices, also engendered high user satisfaction and productivity.

Company investment in the applications was significant to extremely significant. The companies that participated in the study ranged in size from small (\$20 million annual revenues) to global (Nokia with nearly \$40 billion annual revenues). The applications under study ranged from small, single-purpose applications, to enterprise-wide applications with dozens of user types and thousands of users. Development team size ranged from 5 to 100 (depending on the size of the programming team). Development budgets ranged from hundreds of thousands, to tens of millions of dollars. Development time frames ranged from a few months to as many as five years.

Despite this wide range of company types, application types, sizes, and budgets, development team sizes and development time frames, very clear and consistent best practices emerged that cut across all these factors. Tristream found clear and universally effective best practices for gathering user input, gathering business requirements, developing user strategy, user experience, and the user interface, team dynamics and composition, and documentation that any development team can take advantage of regardless of team size or project scope.

Across All Types of Tools

Our evaluation does not address the strengths or weaknesses of any particular development processes (such as RUP, GPLC, XP), support applications (such as Visio, Rational Rose), or development tools (such as UML). The decisions which drive the selection or deployment of processes, support applications, and tools relate more to the

budgets, established culture, and preferences of the individual companies. By themselves, they do not insure, or work against, creating a great application.

The best practices Tristream identified cut across all processes, applications, and tools.

Best Practices Contrasted against “Normal” Practices

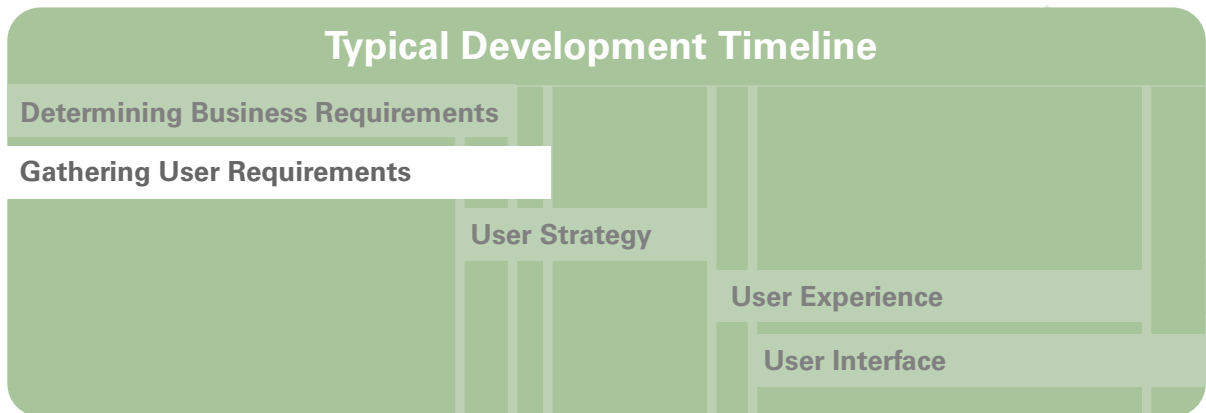
In the detailed descriptions of the best practices, you will find that we also include the “Normal Practices” that we encountered as well. We found that many teams considered themselves to be following the best practices we shared with them as the study concluded—when in fact we knew that they were not fully following the best practices. Perhaps they were short-cutting some of the key aspects of the best practices, or perhaps they had been doing it one way for so long that they could no longer truly see the fine points of what they were doing.

We therefore added descriptions of “Normal Practices” which will allow development teams to contrast themselves against actual best practices. The differences are sometimes subtle and therefore we tried to provide as many quotes and anecdotes as possible to illustrate both the best practices and the normal practices.

Bottom Line

Where the best practices identified were used, application usability and productivity were clearly superior; where they were omitted, application usability and productivity were clearly inferior. ▲

Gathering User Requirements



Any great (or even good) application design is based on the real needs of its actual users. The best teams engage in a methodical user requirements-gathering process to learn and immerse themselves in those needs.

1. The team sets clear objectives for gathering user requirements, including timelines and success metrics.

Best Practices

- a.** The team objectives for gathering user requirements are simple and clear, and they include specifics for how the team will include the information gathered into the design process, including the use cases.
 - This need not be complicated, just clear. Break your users down into groups and sub-groups, figure out how many of each you can get input from, and how you are going to get that input. Some teams put off user research simply because it seems like a “big deal.”
 - Where there is a need for extensive and thorough user research, setting objectives becomes highly important. User input on very specific aspects of the application under development, or on very specific task- and workflows, takes careful planning.
- b.** The objectives and success metrics are understood and agreed upon by the entire team before the user-gathering requirements process begins.
- c.** The team decides on a practical documentation system to facilitate the user requirements being included in all phases of the application design process.
 - Simple is best. Documentation that is performed by team members should include their own analysis. Passing on recordings, transcriptions, or raw survey data is time-consuming for the team (which generally means most other team members will not take the time) and does not give the team the benefits of the team members' impressions and insights gained while conducting the research. Separate the wheat from the chaff.

- d. The project timeline provides adequate time for the gathering, documentation, and prioritization process.
- Many teams complain that they do not have time for user research, when in fact they never even try to get it in their schedule. The best teams make the time by working it into their schedule at the very beginning of the project—even if all they can spare is the time to talk informally with a handful of users.

Normal Practices

- a. Teams may have an orientation toward the importance of user requirements input into the design process, but have no real plan for how to achieve it, other than doing “acceptance-testing” toward the end of the development process. Generally, once the development process has reached the stage of acceptance-testing, it's not feasible for serious user experience modifications to be made.
- There is an unstated confusion on many teams between user research and user-testing. User research, generally speaking, is a more informal process consisting of interviews, asking users to perform scenarios, exploring task- and workflow in workshops and doing a lot of watching and listening. This kind of user input allows teams to get a feel for the users and their most important concerns, needs, and problems. Without this, it is hard to begin designing even the first screen of the application. Once screens get designed, then user-testing becomes appropriate in order to refine and improve on the core experience design. But if your core experience design was not informed by prior user research, no amount of user-testing is going to help you overcome core user experience issues—short of starting over.
- b. Teams may do some initial user requirements-gathering, but have made no plan in terms of process or documentation to include the information into the application design.
- User needs, like business requirements, need to be distilled and prioritized. If not distilled into specific features and functions, the user needs remain as a generalized concept which can not get inserted methodically into the process. If user needs are not prioritized, then teams have to deal with the same problem as non-prioritized business requirements—everything is of equal importance and features which are, in reality, of lesser importance can end up dominating screen designs.
- c. The timeline doesn't allow information from the user requirements-gathering to actually influence use cases and/or interface designs.
- Gathering user requirements, gathering and determining business requirements, and clarifying infrastructure and programming parameters should all be done in parallel and come to conclusion before any of the design process begins. Once use cases are written and/or screens are being designed, it is very difficult to get significant and new features added and therefore new features are added only when they have a very high priority. Realistically speaking, given the schedule pressures of most projects, information not known before the development of wireframes and use cases will not make it into the application.