PHASE

"My perspective is that the bulk of our industry is organized around the demonstrable myth that we know what we want at the start, and how to get it, and therefore build our process assuming that we will take an optimal, direct path to get there. Nonsense. The process must reflect that we don't know and acknowledge that the sooner we make errors and detect and fix them, the less (not more) the cost."

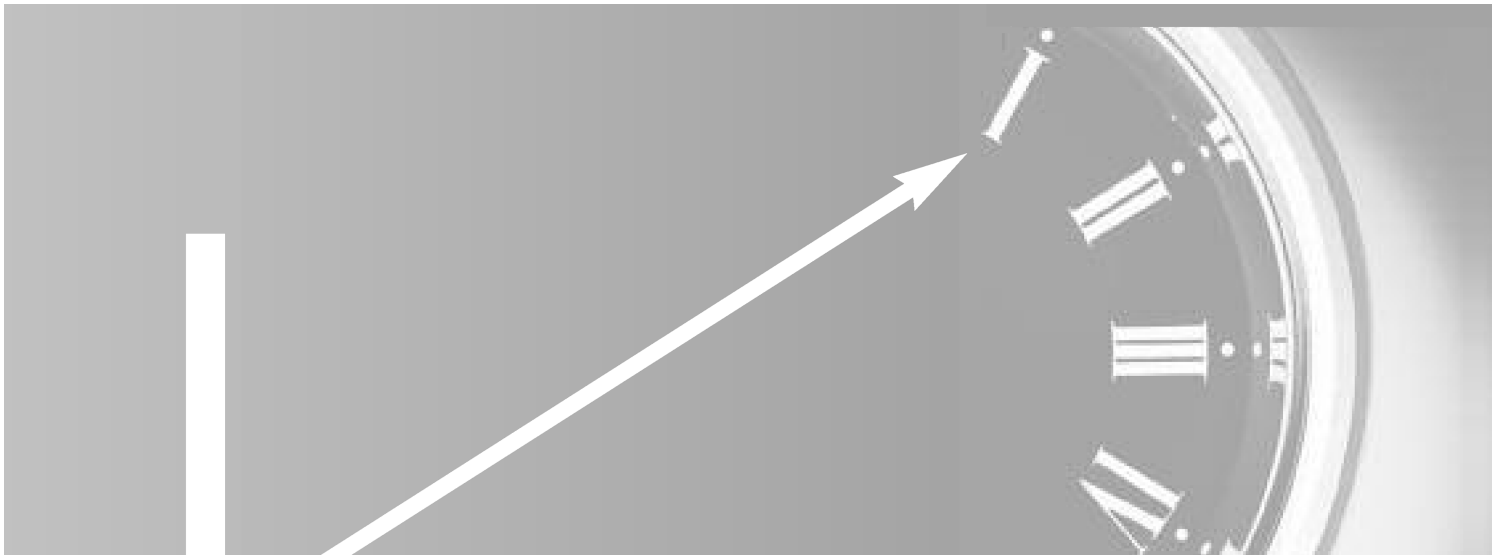
Bill Buxton

Phase I describes the planning aspects of prototyping.

Chapter 3: Verify prototype requirements

Chapter 4: Develop task flows and scenarios

Chapter 5: Define prototype components and content



PLAN YOUR PROTOTYPE

By recognizing, adopting, and adhering to a prototyping plan you can understand and identify the requirements and assumptions of prototypes, develop task flows and scenarios to set the context, and then decide on the mix of content and fidelity relative to that context.

Because prototyping is a key design activity for transforming assumptions and requirements into a software design solution, Phase I begins with validation of those assumptions and requirements. This verification is an essential step in the prototyping process and is used to inform the overall prototyping objective through focusing on specific requirements.

Assumptions and requirements, however, are still not enough to create an effective prototype. You need more context; that is, knowledge of how the requirements and assumptions are handled and interpreted in the real world. The two proven methods for obtaining this context are to first create a **task flow** and then write a **scenario** that fulfills the task flow. If the task flow and accompanying scenario establish how the prototype will work, content and fidelity of that content determine what will be prototyped and the appropriate level of detail. A prototype can contain many different kinds of content, including information, visuals, and navigation structure. Any mix of content can be shown in a continuum from low fidelity to high fidelity. The higher the fidelity, the more representational of the intended finished product it will be.

Although we discuss products from Adobe, Apple, and Microsoft we also have had success in using various open source equivalents, such as OpenOffice products. The placement of commands often changes, but surprisingly similar results can be achieved. Although we discuss the commercial products because of their ubiquity, we do not advocate them as the only prototyping tools.

CHAPTER

The Effective Prototyping Process

3

VERIFY PROTOTYPE ASSUMPTIONS AND REQUIREMENTS

This chapter focuses on transforming prototyping assumptions and requirements into a basis for a prototype design. Before you can effectively prototype, the first important activity is reviewing the balance of assumptions and requirements, including business/marketing, functional, technical, and usability ones. This review will help you determine whether you need more requirements before proceeding to build a prototype. It is also the point where you and your team can be clear about the assumptions (if any) being used as a basis for a software concept. Prototypes allow these assumptions and requirements to be validated, invalidated, modified, or substituted early in the software process thus helping you avoid the risk and costs of making major modifications to your design after discovering that the design concept and usability are faulty during the software development stage.

Prototyping can track the software's assumptions and requirements because of the continuous interplay of testing assumptions against documented requirements until all the assumptions are satisfied and requirements are validated. The nature of prototype iteration makes this interplay necessary. In one prototype iteration some requirements are validated while new questions arise to be addressed in the next iteration. As often as we have seen only one prototype produced in an entire software project, we have never seen that prototype provide all the answers. Indeed, the first prototype, as the first visualization of the software, invariably raises more questions about the requirements than it resolves. Likewise we have also seen a prototype received poorly, and see whole scale direction changes when only one or two of the key concepts needed changing. By charting both requirements and assumptions you give your next iteration a basis for keeping what is good (what was validated) and discarding what was bad (what was invalidated).

The remainder of this chapter assumes you have the basic knowledge of software requirements. It is outside the scope of a book on prototyping to go into the details of requirements gathering; there are plenty of books on this subject already. Refer to the sidebar on requirements gathering if you would like

a brief review. The sidebar below is only a brief introduction to requirements; if you are interested in more in-depth coverage of the topic, we suggest Courage and Baxter [2004], Kuniavsky [2003], Holtzblatt [2005]. If you are also interested in using personas to help drive requirements or to help drive prototyping definition, we would recommend the most complete reference for practical use of personas, Pruitt and Adlin [2006].

PROTOTYPING REQUIREMENTS ARE NOT SOFTWARE REQUIREMENTS

The basic prototyping strategy advocated in effective prototyping is the practice of basing your prototype design on a mixture of requirements and assumptions. You start with some requirement (even if it is only a vision) and prototype for this requirement based on assumptions. Some assumptions will be validated, others modified, and some others not evaluated. The results are documented, leading to firmer requirements for the next round of prototyping. (See sidebar for an example.) This iterative approach eventually leads to many firm requirements and few, if any, assumptions. The rest of this chapter assumes that your requirements come from some legitimate source in the software-making process, and in our example we also ask you to believe that when a requirement is referred to as established, it has been validated through user research, marketing research, technical investigation, or other legitimate means for establishing software requirements.

In software creation, requirements arise from a number of different sources, such as businesses, market places, end users, customers, and technical opportunities. From the perspective of design and prototyping, requirements can be broken into four main categories: business/marketing, functional, technical, and usability.

Business/marketing requirements–define the needs of business or the marketplace. They are generally derived from any combination of the following: market field research, market analysis, competitive analysis, domain expertise, sales force intelligence, and focus groups. The initial product vision is often embodied in these requirements. A typical source for business and marketing requirements is a standard document, usually called a marketing requirements document (MRD), business requirements document (BRD), or a product requirements document (PRD).

Functional requirements–define the functionality necessary to support the business or marketing requirements. Similar to business/marketing requirements, functional requirements are generally derived from any combination of the following: field research (best in conjunction with user research), market analysis, competitive analysis, domain expertise, sales force intelligence, and focus groups. Usually, functional requirements are investigated and defined in parallel with business/marketing requirements. Functional requirements are also often identified as a result of user research and usability testing. Because product sales influence these requirements, they often include more functions (a.k.a. features)

than those needed to satisfy the business requirements. For example, a business requirement may require form fill functionality. However, if the software is deployed internationally, it may require multiple language support in which form filling in the Roman alphabet can be functionally different from form filling in other languages, such as Asian languages based on ideograms. Functional requirements are usually found in a formal document, often called a functional requirements document or other functional specification document. Functional specifications are typically reflected in a prototype.

Technical requirements–define the technology needed to implement the required functionality. Functionality can often be compromised by the immaturity, prohibitive costs, or unavailability of a required technology. Technical requirements are generally derived from any combination of the following: technology research, technical analysis, competitive analysis, technology expertise, sales force intelligence, and other similar means. Technical requirements are sometimes articulated in a formal specification, such as a technical definition document, but they can also be found in company-specific technology architecture documents or platform-specific guidelines, such as the Windows User Experience Guidelines or the Macintosh OSX Human Interface Guidelines.

Usability requirements–define the user experience and usability requirements needed for user adoption of the software. They are generally derived from any combination of the following: user research, task analysis, competitive analysis, domain expertise, sales force intelligence, customer support intelligence, design and prototyping, usability testing, and other related means. Usability requirements, in conjunction with the other requirements described above, are transformed into user interface specifications, which are often embodied in high-fidelity interactive prototypes.

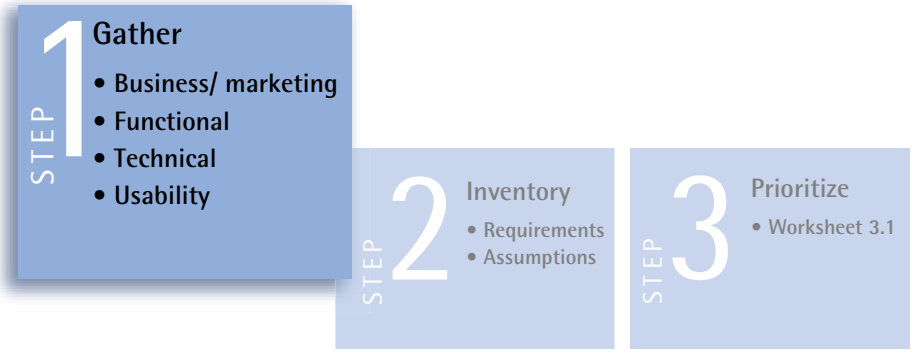
When prototyping, not all (or any) of the above requirements may be available or in a form that is helpful for prototyping. Therefore, it is important to share the prototype with key requirements stakeholders to ensure complete coverage. For business requirements, it would be best to seek out product management and marketing as these stakeholders. For functional requirements, a domain specialist or anyone who has conducted user or market research can provide input. For technical requirements, either a lead architect or software engineer can provide critical input. Usability requirements can come from user research, task analysis, prior similar experience, and usability validation of your design. However, despite coming from individual experts, all these requirements still need to be validated holistically because software is never merely a sum of its requirements. Requirements may have unwanted side effects that, before creation of the final product, only a prototype can expose.

TRANSFORMATION OF ASSUMPTIONS TO REQUIREMENTS

To illustrate a typical course for converting assumptions into requirements through prototyping, we have outlined how requirements are first elicited in the form of assumptions. These assumptions are then iteratively tested in a prototype. A prototype is completed in rounds of iterations until all the high priority assumptions are validated into requirements. Per available time and resources, this is the process of validating prototype directions to the extent allowed by a given prototyping method that enables an iteration to proceed to the next step. The entire process is a repeatable series of three steps.

To begin these three steps for transforming assumptions into requirements, you need to have the requirements necessary for proceeding with development of the prototyping schema.

STEP 1: Gather Requirements



(see sidebar on prior pages for some potential sources)

The prototype should not undertake an entire requirements-gathering process, which needs to be quite thorough; instead, it is assumed here that the software requirements-gathering process is occurring in parallel and is incomplete. So the gathering done here is just the current state of the requirements, as they are known at this moment. Depending on your stage in the requirements-gathering process, the list can be short and vague in the beginning to quite long and detailed at the end. After these requirements are gathered, enter them into Worksheet 3.1, which will look something like this.

WORKSHEET 3.1: Requirements

Project Name:
Project Date:
Product Name:
Current Phase: Design
Prototype Name:

Requirement Name (Examples)	Type	Priority	Validated? Y/N	Results	Requested Changes

This worksheet lists requirements for time management software. The requirements are currently coming from two sources: a MRD and a presentation of a wireframe sketch to a group of stakeholders.

INFORMAL PROCESS TIP

There are formal processes for software product development in almost every software company. These processes guide you through the development process and help you determine at what stage (usually extremely early) you can change requirements and when they should be frozen (usually very early). By taking the inventory of requirements suggested in Step 1, you immediately discover how far you have progressed in the software development process. Are all the requirements known and worked out? Have they been validated? If so, you should be in the later stages of the process.

If the list remains short and vague, regardless of where the company believes they are in their own development roadmap, they are still in the early stages of the software creation process. The more vague and general the requirements, the more they are open to interpretation and completion with non-validated assumptions. Too often in the software process, design and creation are crammed into the later stages when there is little time available for prototyping and validation, thus leading to high-risk software development. As a general rule, the more you base software on assumptions, the higher the risk. Conversely, the greater degree that software is based on validated assumptions, the lower the exposure to risk.

STEP 2: Inventorize the Requirements

WORKSHEET 3.1: Requirements [Example]

Project Name: Time Out
Project Date: Dec 2007
Product Name: Time Out Time Management For All
Current Phase: Design
Prototype Name: T55

Requirement Name (Examples)	Type	Priority	Validated? Y/N	Results	Requested Changes
User must be able to enter time worked by week. MRD- Use Case 1.2	**Functional**		**Y 31 DEC 2005**	**1207minutes.doc**	**Add ability to change project code for previous entries**
User wants to optionally enter project information	**Usability**		**N**		
User wants excel interface	**Usability**		**N**		
Time summary reports for managers	**Business**		**N**		
Project reports for project managers	**Business**		**N**		
Time entry reports for employees own data	**Business**		**N**		

List the state of each requirement. Make note of each requirement's origin (preferably linked to the related document). You can usually judge by its source whether a requirement is validated or just an assertion. If the requirement is validated, note when it was validated. As contradictory or complementary requirements arise later, it may become necessary to challenge old requirements that have fallen back into assumptions. You may, at any point, reclassify a requirement as an assumption if you believe that it has not been adequately validated or has been invalidated in light of newly introduced developments.

The sample worksheet lists a requirement already validated via an MRD for a time-management application and includes a link to the wireframe and the meeting minutes of the wireframe presentation. The validation also uncovered a new assumption: that users may also need to enter project information. It is listed as a direction because the assumption came from a wireframe presentation, so it is a little firmer than just a mere assumption: We know there is a need to enter project information but have no idea yet as to how. If the analysis reveals that other assumptions are hidden in a requirement,

they can be fleshed out in this worksheet. For example, the requirement states time entered by week. This requirement may have additional dimensions: Maybe time needs to be entered by a configurable time period? Or maybe just by week or month? The worksheet helps to list explicit new assumptions as well as the firm requirements. Finally, the worksheet shows an assumption that users want to use an Excel-like interface because the program they are currently using has a similar interface.

STEP 3: Prioritize Requirements and Assumptions

The resulting worksheet from Step 2 is shown below. The last step in transforming assumptions into requirements is their prioritization. The priority refers to the priority for inclusion in the prototype, not priority for implementation in the software. Given that the first prototype is a storyboard, some requirements that are very important for the software are not the main concern of the prototype.

WORKSHEET 3.1: Requirements [Example]

Project Name: Time Out
Project Date: Dec 2007
Product Name: Time Out Time Management For All
Current Phase: Design
Prototype Name: T55

Requirement Name (Examples)	Type	Priority	Validated? Y/N	Results	Requested Changes
User must be able to enter time worked by week. MRD- Use Case 1.2	Functional	High	Y 31 DEC 2005	1207minutes.doc	Add ability to change project code for previous entries
User wants to optionally enter project information	Usability	High	Planned		
User wants excel interface	Usability	Medium	N		
Time summary reports for managers	Business	High	Planned		
Project reports for project managers	Business	Medium	N		
Time entry reports for employees own data	Business	Medium	Planned		

REQUIREMENTS AND THE BIG PICTURE

Figure 3.1 shows how prototyping requirements all fit together holistically in the software creation process. We demonstrate how assumptions are essential to the prototyping process by tracing the diagram. Only when all major assumptions are satisfied should prototyping end.

ITERATION 1: FROM IDEA TO FIRST VISUALIZATION

In the first prototype iteration, the product (or the function or new addition to the existing product) is just an idea in a product manager's or business analyst's mind. Working interactively with a designer or by sketching the idea out themselves, some of the assumptions can be visualized. This visualization provides a vague idea of the business value (i.e., is the idea worth pursuing?).

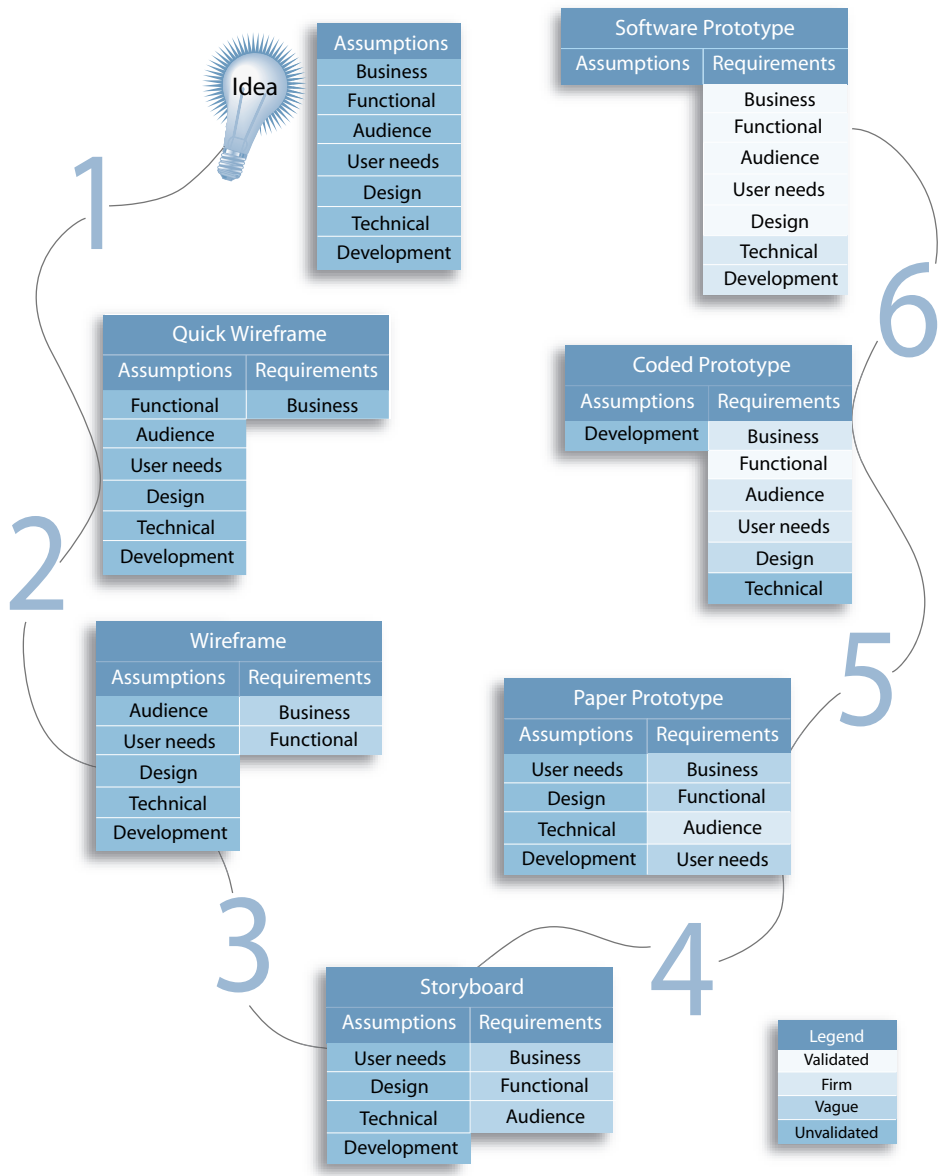


FIGURE 3.1 *Step-by-step conversion of assumptions to validated requirements. Note: The levels of gray show how refinement continues throughout the process; the darker the background, the less (if any) validation used.*

So even a quickly developed prototype can validate assumptions by providing visualizations to realize an idea. The visualization itself can communicate the value of proceeding to the next step in the process: working out the idea even further and validating its companion requirements. Also, through visualization some competing assumptions can make an idea less desirable, thereby invalidating the business requirement.

The travel and expenses reports example shown in Figure 3.3 seems like a plausible idea to the stakeholders. Starting as just an idea for travel reporting, the

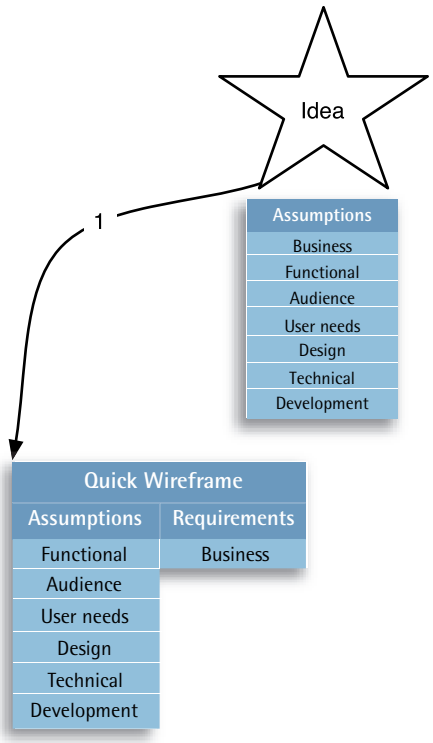


FIGURE 3.2 *First iteration from idea to a quick prototype transforms business assumptions to firm requirements.*

visualization provided the product manager with the idea of a new assumption: adding normal expenses in addition to travel expenses. The visualization of this new assumption has not uncovered any undesirable effects, so the project proceeds to the next step: working out the wireframe depicted in Figure 3.3 to make other assumptions clearer.

FIGURE 3.3 *A quick wireframe for a travel and expense reporting software.*

ITERATION 2: FROM QUICK WIREFRAME TO WIREFRAME

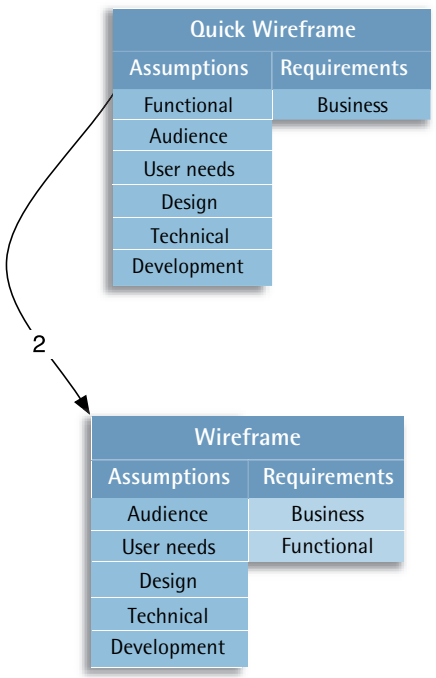


FIGURE 3.4 *Second round of iterations verifies more requirements and makes other assumptions clearer.*

The second round of iterations is performed similar to the first prototype iteration. Assumptions and requirements are prototyped to either validate or invalidate them. Through this second round, the requirements worksheet becomes more complete and the direction of the product becomes clearer.

With the creation of the more refined wireframe shown in Figure 3.5, not only is the business case a little clearer, but the functions needed to support this business case also become clearer and better reflected in the buttons and navigation visible in the wireframe.

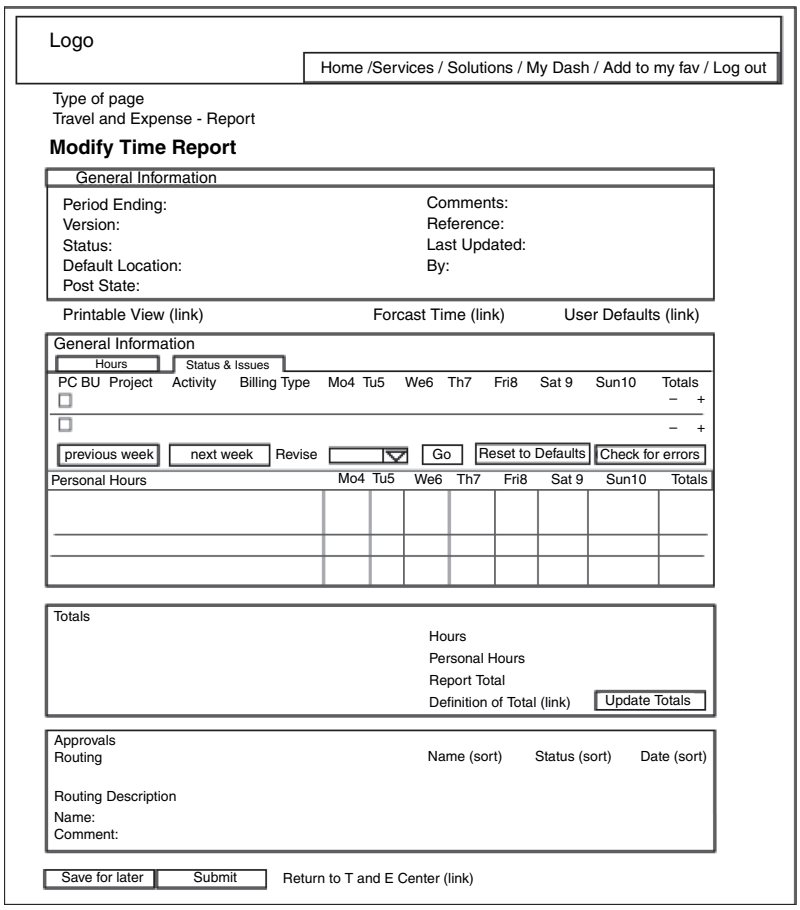


FIGURE 3.5 *A wire-frame showing more functional assumptions of the product.*

ITERATION 3: FROM WIREFRAME TO STORYBOARD

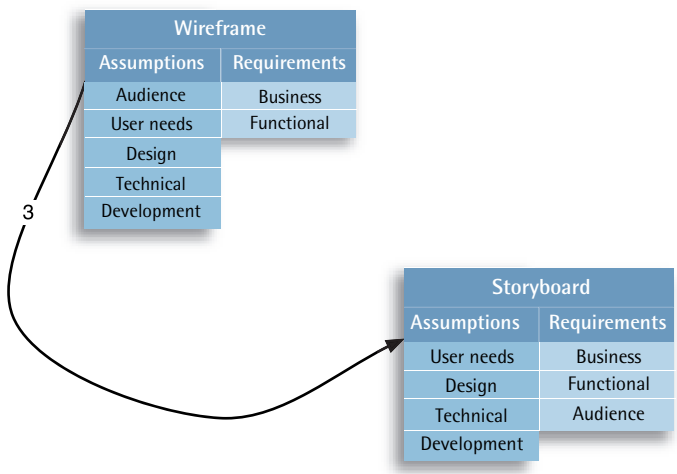


FIGURE 3.6 *End-user focus in requirements becomes visible as a result of a storyboard prototype.*

By using the recommended Worksheet 3.1, we see a definite increase in end-user–focused requirements when we get to the storyboard phase, which attempts to tell the story of the software in context. During this phase, new requirements will surface and other user requirements can be validated. For example in the storyboard picture shown in Figure 3.7, the end user does indeed need to track his project budget, validating not just optionally entering project data in the time report but also the need to have a project time report for the end user.

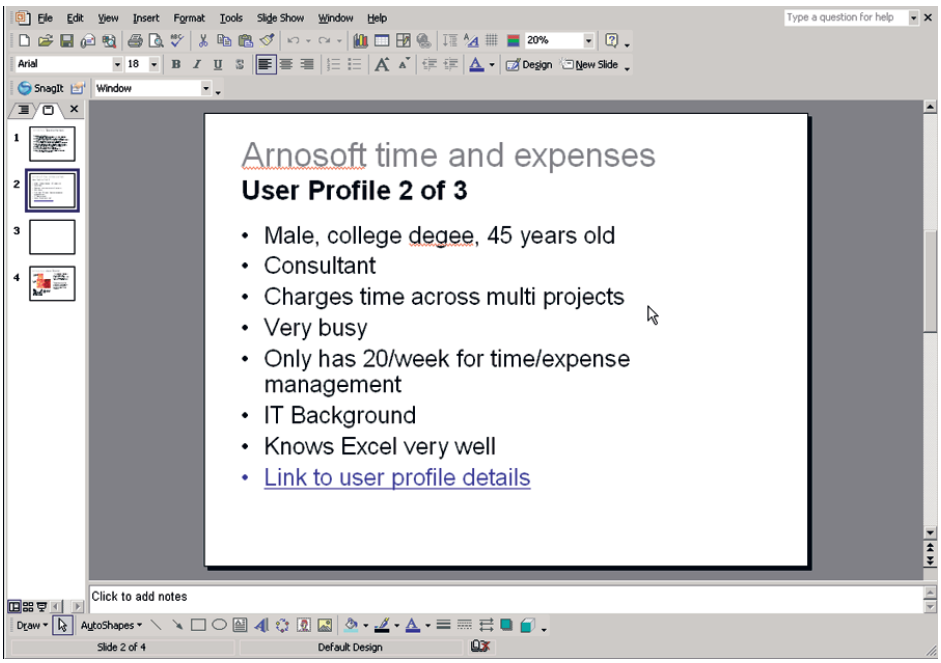


FIGURE 3.7 *Storyboard showing validation of the need to enter project data for an archetypal end user.*

ITERATION 4: FROM STORYBOARD TO PAPER PROTOTYPE

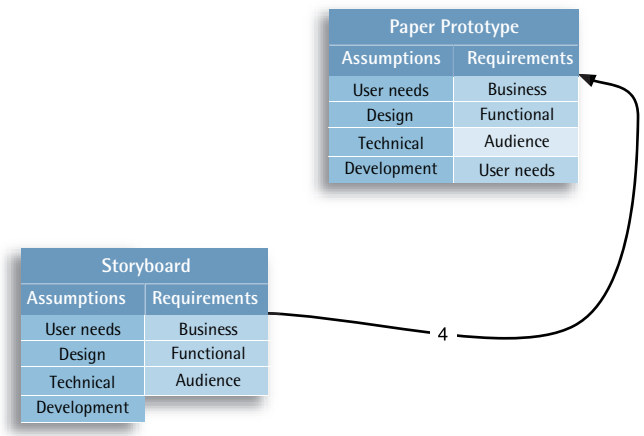


FIGURE 3.8 *Given the more visually explicit and interactive nature of a paper prototype, requirements start being validated in rapid tempo.*

As the prototypes progress and the audience shifts from internal to external stakeholders, requirements start to be validated quickly, especially with direct user contact. Designer "requirements" about the audience suddenly become much clearer, and user needs becomes much more tangible. In the paper prototype shown in Figure 3.9, the assumption is that an Excel-like interface failed miserably in usability testing. It invalidated an assumption of the end-user needs and replaced it with a new model that fits better but will still need verification (not pictured).

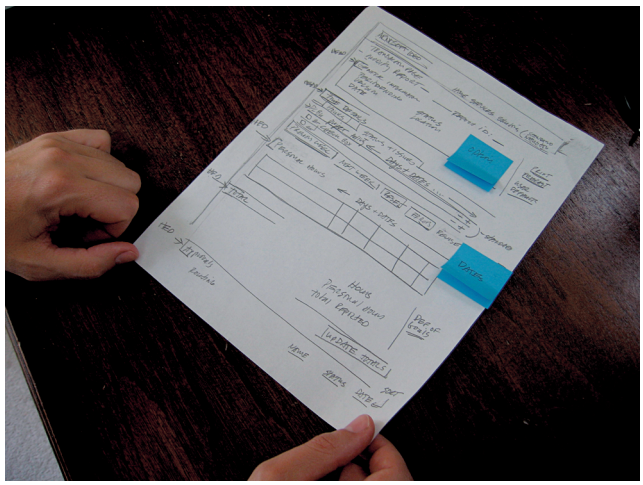


FIGURE 3.9 *An interactive paper prototype.*

ITERATION 5: FROM PAPER PROTOTYPE TO CODED PROTOTYPE

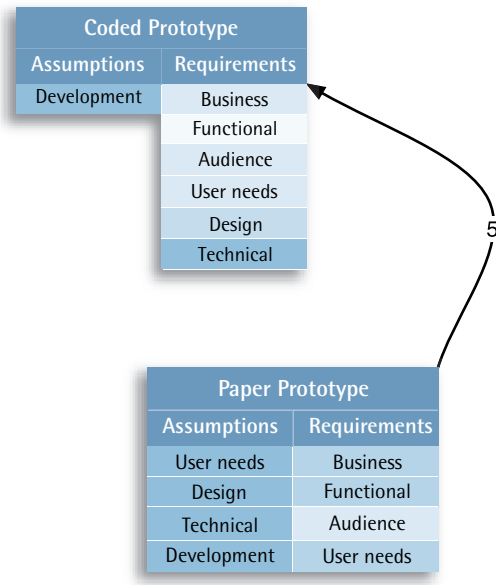
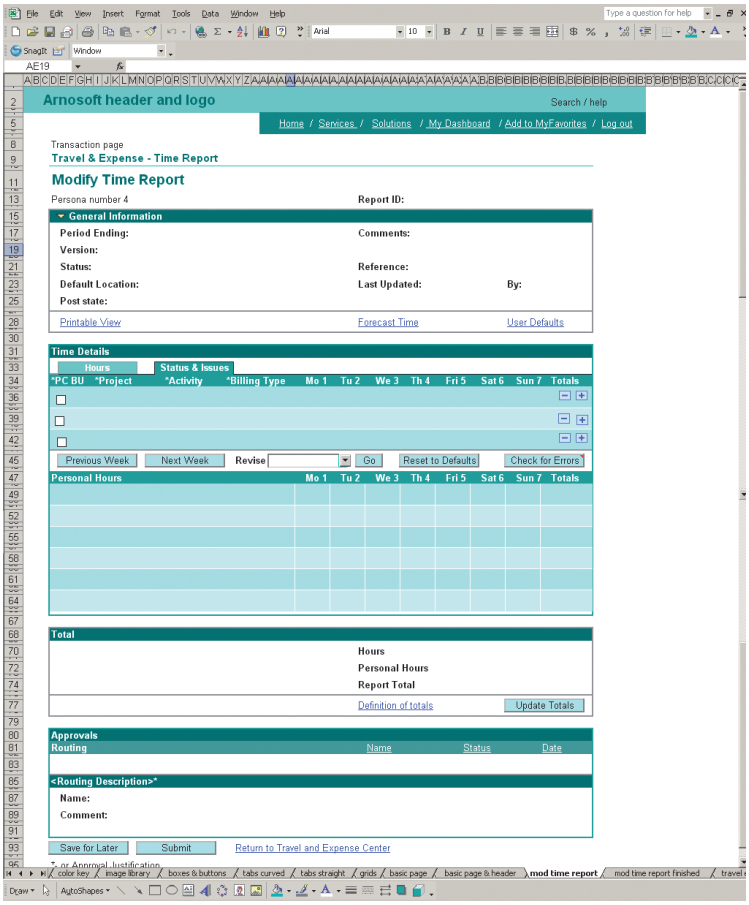


FIGURE 3.10 *Late high-fidelity prototypes come closer to resembling a software product as well as the requirements.*

In the last stages of prototyping, many open design and technical questions can be answered. The required functionality, the audience, and the business case are already firm and no longer the source of focus. Now, a high-fidelity prototype is used to firm up the remaining requirements and design details.

FIGURE 3.11 *Late prototypes resemble the real software as the requirements become firmer, and more advanced prototype development can take place with greater confidence.*



ITERATION 6: FROM CODED PROTOTYPE TO SOFTWARE REQUIREMENTS

In the last step, specifying the requirements from a late high-fidelity user-facing prototype (here in the form of a coded prototype) enables us to finally say we have validated all the software requirements. The worksheet that was the basis for evaluating the prototype requirements could now almost double for a table of contents or central reference point for the software requirements. So the journey from the interplay of assumptions and requirements is now complete; prototyping has been the primary aid in validating assumptions and transforming them into requirements. Although, it is important to note that prototyping has been an aid, not the sole source of requirements validation, such as focus groups, usability testing, market research, competitive analysis, etc.

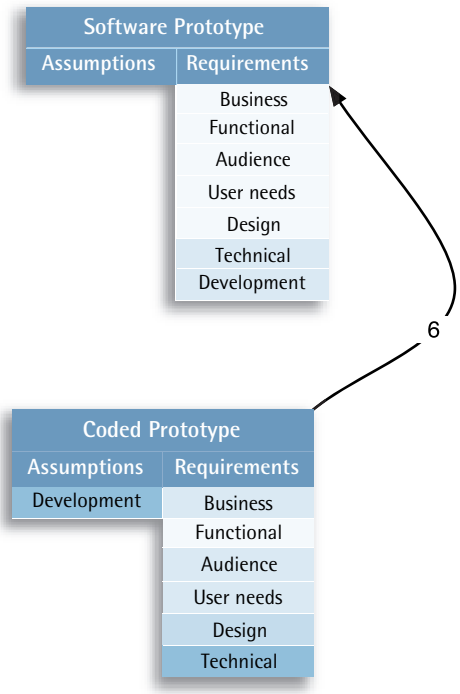


FIGURE 3.12 *Only at the end of the prototyping process do the assumptions finally give way to concrete data to base the software creation and development.*

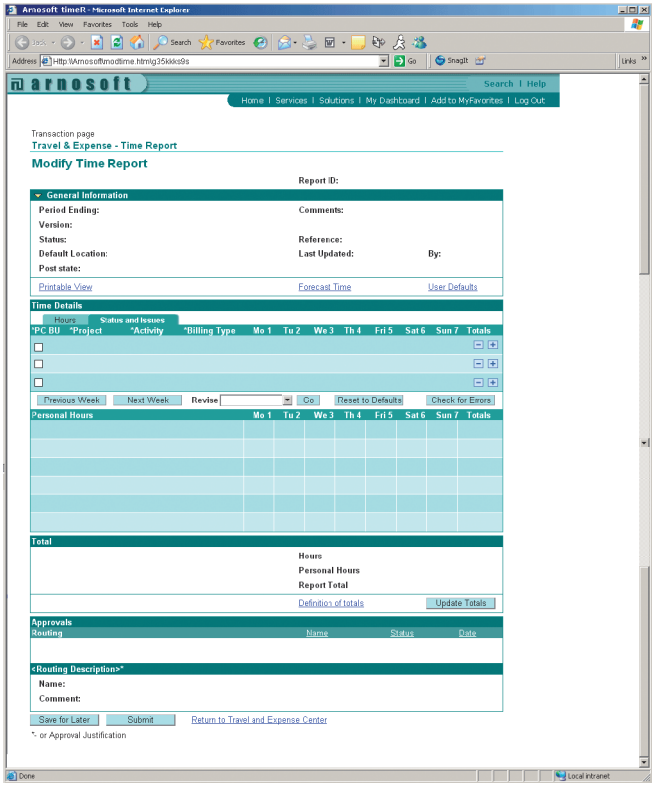


FIGURE 3.13 *The final end product for time entry at the end of the project.*

SUMMARY

We reviewed requirements setting for prototyping as the first step toward collecting prototype content. We have seen that prototyping requirements try to come as close as necessary to the actual business, functional, technical and usability requirements. However, a prototype also has the flexibility to be based on assumptions. In fact, prototyping can be used to play with assumptions while being gradually turned into concrete validated requirements. For this validation, a worksheet template supports the three-step process:

Following this validation process and using the worksheet template, you can be assured that your prototype will address exactly the right assumptions and requirements your team judges to be important. The worksheet, with the prioritization of requirements and assumptions, also helps others understand what they should and should not be looking for when reviewing your prototype.

ARNOSOFT GOES THROUGH THE REQUIREMENTS

"We should do a wireframe!" insisted Art.
"No, we should do a paper prototype, so we can conduct usability testing with it," argued Ina.
"No, we should do a proof of concept that will help us code the product; this will help my team and that is what we need in this time crunch situation, full stop," demanded Dirk Spine.
"I think all Reed wants is some proof of concept. Maybe Ina can do something in Flash for it?" asked Alfredo.
"Why should we just give Reed what he wants? If we can find some way the prototype can help the team, we should do that instead," countered Dirk.
Maybe we can find a win–win situation here. Let's start with the first step in the effective prototyping process and try to establish the requirements of the prototype as well as any assumptions.

The team starts with trying to decide the prototype requirements. They notice immediately a huge gulf between the requirements they need and the ones they have in hand. All they really have are the brainstormed business requirements. Because Reed participated in the brainstorming, the business and functional requirements are considered validated. Nevertheless, the team decided to use the step-by-step method described above by first gathering requirements as well as listing any assumptions.

Many of the requirements for the ceramic ware site are being extrapolated from the existing gardening equipment site. In the words of the CEO, Valmar Vista, "think of it as indoor gardening."

The team also decided to prioritize the business, functional, and technical requirements because a planned user research project is meant to address many of the usability requirements. After a brainstorming meeting the following worksheet was developed. After following the steps of the requirements process, a storyboard prototype was chosen because it could be quickly created and "clicked through." The results of the storyboard exercise are also reflected in the spreadsheet.

REFERENCES

Apple Computer. Inside Macintosh: Macintosh Human Interface Guidelines. Boston: Addison-Wesley, 1996.

Bill Buxton. Software design. Proceedings of the Second International Conference on Usage-Centered Design, Portsmouth, NH, October 26–29, 2003, pp. 1–15.

Bill Buxton, R. Sniderman. Iteration in the design of the Human-Computer Interface. Proceedings of the 13th Annual Meeting, Human Factors Association of Canada, 1980, pp. 72–81.

Catherine Courage, Kathy Baxter. Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques. The Morgan Kaufmann Series in Interactive Technologies. Amsterdam: Elsevier/Morgan Kaufmann, 2004.

William Cushman, Daniel Rosenberg. Human Factors in Product Design. Amsterdam: Elsevier, 1991.

Karen Holtzblatt. Rapid Contextual Design. San Francisco: Morgan-Kaufman, 2005.

Mike Kuniavsky. Observing the User Experience. San Francisco: Morgan-Kaufman, 2003.

Scott MacKenzie, R. William Soukoreff. Card, English, and Burr (1978)–25 years later. Extended Abstracts of the ACM Conference on Human Factors in Computing Systems–CHI 2003. New York: ACM Press, 2003, pp. 760–761.

John Pruitt, Tamara Adlin. The Persona Lifecycle. San Francisco: Morgan-Kaufman, 2006.

Bruce Tognazzini. First principles of interaction design. http://www.asktog.com/basics/firstPrinciples.html. Accessed June 17, 2005.