Introduction

The design of a user interface (UI) appears to be relatively simple at a high level—everyone has an opinion about what is right and wrong in any given UI. Just about anyone is able to design parts of any UI, though few are able to handle the entire job and various aspects of design through deployment. Even fewer people are able to prototype or implement a software UI, much less deploy it to users on schedule and within constraints.

Rule of Thumb: Anybody can design a UI, anybody can design a client/server application, anybody can design a web-based application, and anybody can design a database—just look at the evidence in the sea of software in the world today. It’s when a design must meet challenging constraints for competitiveness, UI, usability, consistency, cost, resources, skills, and schedule that the need for special expertise becomes evident.

Numerous opinions about a UI don’t work when:

- The design of software must meet stringent criteria for the UI, usability, consistency, and integration
CHAPTER 1  INTRODUCTION

- Implementation must occur within resource and schedule constraints placed on a design and implementation team with a certain set of skills
- Strict accountability for results is tracked

The process of designing a UI is highly complex, nonlinear, nondeterministic, and nonorthogonal. Complexity is normal in software and even more so in the UI because of the large number of factors and unknowns. Design is nonlinear because there is not necessarily a fixed, orderly, and straight-line path from start to finish. The process of design is nondeterministic because there is no equation to produce the same results given the same inputs, and it would be almost impossible to produce the same results even under duress. UIs are nonorthogonal in the sense that a decision in any one facet of a design can influence other facets in not always pleasantly surprising ways.

Statistical independence is not joy that is available in creating UIs and systems—it’s more of a puzzle.

**Rule of Thumb**: UI design and development is surprisingly complex and easy to underestimate relative to planning, requirements, and deployment.

In order to simplify a discussion of such a broad topic and be effective at the same time, a “best practices” approach for UIs and usability is provided for people involved in software planning, design, implementation, testing, and deployment. The focus is on identifying the right things that must be done correctly in order to achieve success. These right things are the basics that are independent of the current industry buzzwords used to name them.

This chapter covers:
- A project—keeping things real
- A challenge
- Major causes of project failure (or success)
- An approach to processes
- An approach to solutions
- Best practices

The basic layout of the text is to present chapters in a somewhat linear and design task oriented flow. However, each chapter is somewhat independent of preceding ones for those with a need to reference information in a different order.

A project introduced in this chapter is used throughout the text. It is the basis for exercises designed so a reader can experience a comprehensive set of real-world design and implementation situations and tasks.
The opinions of the author, readers, UI and usability engineering experts, or others are important to a product’s success, but users and/or usability testing ultimately determine whether a solution meets requirements or whether one solution is better than another. In fact, a large number of alternative solutions may satisfy the likely requirements for the project, but what determines the best design possible?

A Project—Keeping Things Real

The market research department has convinced senior management of the viability of a potential new product. As a result, and in spite of very tight budget and headcount constraints, management has approved the immediate assignment of a small team to pursue initial validation of project concepts and likely development costs.

The Application. The project is to deliver a software application supporting scheduling and attendance of events at major conferences like Guide, Share, and Computer Human Interaction (CHI). Conference events include tutorials, workshops, presentations, and after-hours sessions. Likely users of the software are conference attendees, as well as conference personnel who define and maintain events for the system.

The Platforms. The conference scheduling application is intended to operate on three hardware and operating system platforms:

- Laptop computers running Microsoft Windows
- Network computers running Windows/NT or UNIX
- Interconnected personal digital assistants (PDAs)

Web-based and kiosk support may be important for a later release of the software, though this is not known with certainty at the time of project kick-off. Market Research has an action item to return with an answer to this question and others within three months.

Tools. The implementation languages are not known. No development languages or tools are standardized for project development at this facility. However, Java, C++, Visual Basic (VB), and HTML are leading candidates for all or part of the software. Several other languages and tools are in use within the development organization, but no other development tools have been discussed or finalized for use on the project.
CHAPTER 1  INTRODUCTION

The Team. The project startup team consists of an experienced and aggressive software development manager and two software developers. One of the developers is very experienced in client-server software infrastructure, and the other developer is experienced in GUI-based client software applications. Both are experienced leading software teams.

The members of the team are relatively new to the parent development organization and have not worked together before. None of the team members have developed software for PDAs, Java, object-oriented software, Web-based technology, or UNIX.

The initial market research personnel have been reassigned to other locations and jobs, but their management is finding personnel to complete remaining work items. The Senior Management Team of the project startup team has agreed to provide marketing and other support if the project proves promising. No other staffing is available for more than minor consultation.

The Schedule. Market research predicted potential first year sales of over 10 million units of the software worldwide if delivered within 15 months at a certain price. Competition is anticipated and is expected to erode the sales prediction significantly with any delay beyond 18 months. Potential units are well over 50 million if extensions to support scheduling of educational, sporting, and other events are considered.

Senior management has requested a weekly 15-minute status meeting with the leader of the project team. Senior management has requested a preliminary assessment, project schedule, and likely resource estimates within 30 days from today. The project leader has agreed to the requests made by senior management.

Any questions?

A Challenge ____________________________________________

The project is not too unreal as a description of how development of shrink-wrapped applications begins. For software developers working on internal applications, the scenario is not too far off the mark with minor modifications. For example, what if the development project is intended to provide a solution for scheduling internal education first, and then the solution is to be extended, packaged, and marketed externally at a later time?

There are many challenges in today’s software development environment. There is severe pressure to reduce costs, shorten schedules, achieve more
predictable plans, deliver higher quality, provide easier to learn and use software, learn and use new technologies and tools, and to achieve better results relative to competition and other factors.

**A Very Broad Topic.** A very large portion of user satisfaction with a product or its usability is typically attributed to its UI. This is a very large and unsupported burden. A broader view of usability is required. In addition, a broader view of user interface and how it is designed and implemented is required.

Before addressing ease of use as a topic, a broader and common understanding of factors that drive user satisfaction is required. Several studies have demonstrated that user satisfaction is a function of a small number of factors. An equation to represent user satisfaction as a function of these factors is:

\[
\text{User Satisfaction} = \text{Function of (FEATURES, USER INTERFACE, RESPONSE TIME, RELIABILITY, Installability, Information, Maintainability, and Other Factors)}
\]

The capitalized factors are the most significant in the equation. Certainly, it is easy to find examples where a product is not usable if:

- Key functional features are missing
- The user interface is missing key features, has poor appearance and behavior, and places high knowledge and interaction demands on users
- Response time and throughput are slow
- The system does not stay up and/or causes work or data to be lost

The factors in the equation can be refined further—for example, UI has components related to appearance, behavior, user knowledge demands, and user interaction demands. Other factors can certainly be added to the equation.

The bulk of a user’s focus is getting a job done quickly, easily, and reliably with tools that automate, augment, and facilitate the task. Ease of install/deinstall is important because it sets the initial and final tone of a user’s experience with a product. Ease of software upgrade is as important as other interactive features of a product. Information (training, tutorials, help, and performance support) is important during initial or later learning but is not necessarily task relevant unless used during an infrequent or complex task.
CHAPTER 1  
INTRODUCTION

There are many other factors, such as consistency, integration, and cost, that affect user satisfaction depending upon role, environment, task, and situation.

**Rule of Thumb:** All factors in the user satisfaction equation and their relative importance must be kept in mind during product planning, requirements analysis, design, implementation, test, and deployment.

The focus of the book is on usability in general and the UI in particular. Only high-level guidance and considerations are made for performance and reliability; the importance of these factors cannot be minimized.

**Risky Business.** Software development is a high-risk endeavor in even the best of cases. There are many examples of software development horror stories—where projects fail due to cancellation, significant schedule slips, rejection by users, or significant cost overrun. There are many examples in the media of the horrors visited upon users and purchasers of software that is deficient in the major areas that contribute to user satisfaction.

One of the major goals of the book is to address the risk areas relative to usability and UI, that is, where things can go wrong. Pragmatic approaches and specific guidance that are immediately applicable to design and usability efforts are provided.

**Timely.** Although there is much that is not known, many things are known about users and UI technology. There are also many things known about how to achieve good development plans, designs, software, tools, evaluation, and iteration. In particular, accepting an effective and rapid iterative process within constraints is a best practice approach to achieving desired results.

The guidance provided applies to achieving results quickly by requirements personnel, designers, trainers, writers, developers, testers, usability engineering personnel, team and project leaders, and managers—everyone involved in software development must do a broader and more effective job.

**Effective.** Just as developers are under pressure to reduce cycle time, UI and usability engineering personnel are under pressure to produce better results in a more effective manner. This pressure means that

- Product personnel do the right things right
- Usability engineering cycles are reduced without compromising results
Causes of Software Project Failure or Success

An exploration of sources such as Brooks, McConnell, and Keil reveals common themes as to why projects fail or succeed. Failure takes the form of major cost overruns, schedule delays, project cancellation, and user rejection. Success takes the form of a product that meets requirements, is delivered roughly on schedule and within constraints, and has a long and positive relationship with its users. These themes are explored briefly here. The root causes of failure and success are explored throughout the book.

User Involvement. Understanding users, their environment, tasks, constraints, pressures, and goals is essential to delivery of a product that meets needs. Methods are available for effectively involving users in requirements, design, construction, evaluation, and throughout deployment and usage.

Rule of Thumb: A user or business representative may not be able to define needs in a way that can be translated easily into requirements or a usable design. However, a user can certainly be adamant about what is liked or disliked about your product when it is made available for testing or use.

Requirements. Business and functional needs are the most common and easiest requirements to gather. However, requirements for UI features, usability, consistency, and integration are overlooked very often. Many times, UI features are defined implicitly if not unconsciously in expectations. With up to 50 percent of software code bulk related to the UI, more effort is required to understand the requirements for product features that are visible to users. In addition, requirements for usability, consistency, and integration must be defined explicitly and measurably. Compliance is then measurable and testable in product plans and evaluation efforts.

Rule of Thumb: Once defined, requirements must be controlled and managed. Project teams must beware of requirements and feature creep by users and the project team. Feature creep kills!
Planning. Developers are thought to be schedule optimistic, but a more likely cause of schedule optimism is being work unaware. This is basically a skills and experience issue relative to current UI technology. There is a large volume of detail, expected behavior, and redundancy associated with many UI styles and features.

Developers responsible for a UI may not be aware of the volume of very fine detail until after the second or third time around. Unfortunately, by that time the more experienced developers have moved on to other tasks, and a new set of developers is working on UI software and encountering the same old UI speed bumps—a costly and vicious cycle.

Planning must provide sufficient time for skill building, use of UI and usability quality methods, and tracking and reporting results. In addition, plans for UI and usability tasks may need to be provided at a lower level of detail than is provided for other areas of a project.

Skills. There is a large amount of learning required for new software development efforts. There are relatively new operating systems, new languages, new application architecture styles, new user interface styles, and lots of new technologies and tools. Each new thing requires time to learn just the basics, more time to internalize and make the basics operational, and then even more time to learn how to exploit. Exploitation means being able to use tools and techniques quickly and to best effect for users. The more things that must be learned concurrently, the higher the risk to a project.

Old Rule of Thumb: There are three major causes of project failure:

• A team that has not worked together before
• An operating system that the team has not used before
• New hardware that the team has not used before

The presence of any two of the three causes leads to a major project failure.

New Rule of Thumb: There are six major causes of project failure:

• A team that has not worked together before
• An operating system that the team has not used before
• New hardware that the team has not used before
• A new UI style that the team has not used before (e.g., nongraphical user interface for a PDA)
• A new design paradigm that the team has not used before (e.g., OO)
• Other new technologies that the team has not used before (e.g., multimedia)

The presence of any two of these causes leads to a major project failure.

Project failure is defined as project cancellation, major project delays, significant cost overruns, inability to deliver major project features, inability to achieve major product criteria, or product rejection by the user community. Major project failures are not fun.

**Design Practice.** It is always better to design prior to implementing and even better to formulate architecture as input to design. Just as there are good practices in software development, there are good practices in UI and usability. Involving users, multidisciplinary teams, design reviews, prototyping, evaluating, iterating, and evolving rapidly are all good development practices if requirements are to be met. If these techniques were used more often by experienced and inexperienced software development teams, many more products would succeed.

**Rule of Thumb:** Follow good design practice earlier rather than later.

**Risk Management.** Forewarned is forearmed. Knowing that software and UIs are high-risk areas, a product team can be proactive in managing the course of events. More anticipation of problems and solutions is required. Given the vast experience of most software development groups, there must be more learning about awareness of typical problems and solutions.

Plan templates help encode and reuse organizational knowledge. Project teams can proactively address the things that typically go wrong—and badly. A template for a risk list of common UI and usability problems is potentially a common and reusable component for project planning; e.g., “too many steps” and “desktop unfriendly screen size.”

The same things seem to go wrong repeatedly. The software “deja vu all over again” phenomenon is explored further as the project is experienced. A major task of software UI and usability is avoiding the tar pits and pitfalls of UI and usability problems.

**Rule of Thumb:** Projects have a better than random chance of succeeding with the right planning, execution, and tracking.
An Approach to Processes

Just having a great process documented does not guarantee a great product. Painting by the numbers does not create great art. Many great artists developed drafts (models or prototypes) of their great works and iterated on their methods and approach until a great work of art was delivered. Understanding needs and success criteria, having a design vision, delivery of the vision, measuring delivery against criteria, iterating until the right design is instantiated—these are very important.

A necessary and sufficient process is described in this book. Since this is not a cookbook, techniques to allow variations and deviations from a basic theme are provided as well. Techniques to judge sufficiency of execution are described.

An Approach to Solutions

Just as there are lots of ways to skin a cat, there are lots of ways to skin a UI. Better solutions are achieved when options are considered. However, the search for alternatives is likely to be constrained by time, cost, skills, movement of technology, availability of resources, competition, or all of the above—and more. At some point, a single solution must be chosen for delivery to users.

Rule of Thumb: Beware of the search for the best design possible. Invariably, any design can be improved upon, but a design that meets all project requirements is what customers really want.

Techniques to generate alternatives, including generation of alternatives to problems discovered during development are described. Techniques to judge sufficiency of alternatives and selection criteria are also described.

Best Practices

An experience-based approach is employed for selection of described techniques. The techniques have been proven to work on multiple projects that include shrink wrapped and internal application software. Use of the techniques does not automatically guarantee success—the wrong things can be done with the right practices. Not using the techniques does not automati-
cally guarantee failure. There are a small number of projects that are lucky or that are right for the wrong reasons.

**Rule of Thumb:** Using the right techniques the right way improves the likelihood of success in a significant manner.

**The Remainder of the Journey**

The journey undertaken in the remainder of this book is one based upon the project and the School of Hard Knocks. The reader is to experience

- Planning
- Requirements
- Analysis of users and tasks
- Design of UI, help, training, and graphics
- Instantiating a design in a specification, prototype, style guide, and product
- Evaluating a proposed product
- Iterating to fix what’s broken and not break what’s right
- Deployment

**Rule of Thumb:** A software developer should be a sadomasochist and really enjoy this type of work.

**Back to the Project**

The project lead has come into your office unexpectedly. He gives you a brief project overview. He also tells you about his commitment to senior management. He tells you how he is organizing the project and whom he has selected as the lead for the non-UI work.

He says that he has spoken with your manager and tells you that he would like to appoint you as the technical lead for the UI work, and expresses his confidence in your skills and abilities. You are currently finalizing some documentation on your previous project and are unassigned to any other project. In spite of the many concerns that you have about schedule and scope, you realize that your choices are somewhat limited in the current environment in the development organization.
CHAPTER 1  INTRODUCTION

The project lead asks you to put together your thoughts about the approach, schedule, and resource needs for the project’s UI. He would like to meet with you in 30 minutes. The team lead for the non-UI work will be there as well.

Any questions?

References