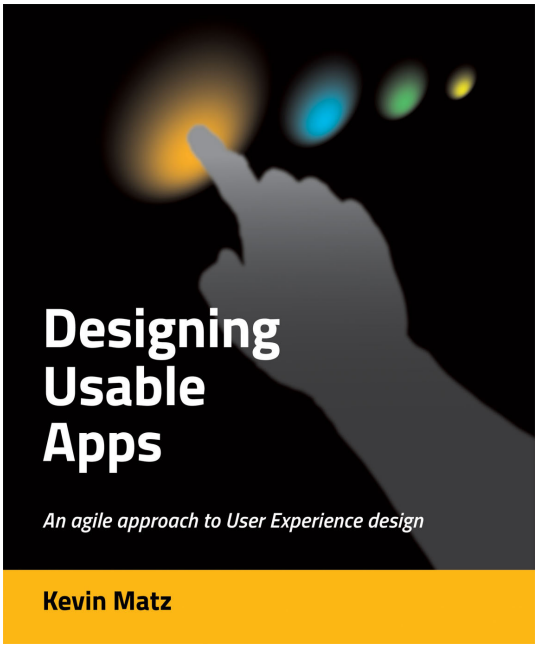
This is a free sample excerpt from the book:

Designing Usable Apps

An agile approach to User Experience design



Author: **Kevin Matz**
264 pages (softcover edition)
Print edition ISBN: 978-0-9869109-0-6
E-book edition ISBN: 978-0-9869109-1-3

www.designingusableapps.com

Available in print and e-book formats at Amazon.com and other booksellers

WinchelseaPress

6

Understanding product requirements

In order to effectively design a software product, you need to understand its requirements. This chapter explains what you information you need to find out about your users and their tasks, shows you how to discover that information, and offers suggestions on recording and communicating your findings.

Requirements

Requirements are statements of the things that your product must achieve for it to be considered successful. If you are building a customized solution for a client, requirements express the wants and needs of your client. If you are building a product for sale to a wider market, requirements express the aggregate wants and needs of potential customers that will be necessary for the product to be able to sell enough copies to be economically successful.

Requirements for software systems are typically classified into two types:

- **Functional requirements** are the features your product will offer — the functions and actions it will support. For example, some functional requirements for a word processor might be that it must support styling text with bold and italic type, it must

allow documents to be printed, and it must allow images to be embedded in documents.

- **Non-functional requirements** are quality constraints that are general or "cross-cutting" in nature. Performance, security, stability, reliability, capacity, and scalability are examples. For instance, an e-commerce website might be required to serve 10,000 users concurrently and to serve pages with a response time of 2.0 seconds or less.

The **scope** of your product and project is defined by the set of requirements that need to be implemented. Without careful management, additional wishes and demands will continually be added to the project's scope, and this type of *scope creep* can threaten your ability to deliver on schedule.

Requirements should state the needs of users and customers, without specifying a particular solution for meeting those needs. In other words, requirements should state *what* the product should do, but not *how* it should do it. Defining exactly *how* the product will meet the requirements is the goal of product design.

In a perfectly disciplined, sequential process, requirements would be gathered and documented, and then subsequently, in a separate analysis and design step, a design would be generated for a solution that will meet the requirements. In many projects, though, the activities intentionally take place together in a blended, parallel form; the requirements often aren't recorded and managed formally, and the team jumps straight into product design.

The idealized, sequential process also isn't fully realistic, because it's virtually impossible to get the requirements right the first time, and requirements often tend to change throughout the life of a software project. Introducing iterative and ongoing requirements analysis is recommended so that requirements change can be more easily managed, and so that the team's understanding of the needs of customers and users can be continually improved.

It should be possible to verify whether the product meets the stated requirements. Some high-level requirements, such as "The product shall be easy to learn and use", can be of such a subjective nature that is not objectively possible to test or prove that the product fulfils the requirement. For such requirements, it is important to state **acceptance criteria** that are stated in concrete, measurable terms. So for "The product shall be easy to learn and use", a measurable acceptance criterion might be "95 percent of users will be able to successfully process a standard passport application after the two-day training session".

Some requirements are "definitional"; they result from, and form part of, your definition of what the purpose and market positioning of your product is. But most requirements exist to ensure that the needs and interests of users and other *stakeholders* are satisfied by your product. Let's now take a closer look at who stakeholders are.

Stakeholders

Stakeholders are all of the different people, groups, and organizations that are affected by the success or failure of the project.

Your product will be operated by users, and your focus should be on designing the product to meet the wants and needs of the users. But in reality, you also have to satisfy other stakeholders, both within and outside your organization. These stakeholders will often contribute requirements and impose constraints, which may conflict with the requirements that are more directly relevant to the needs of the users.

Designing your product can often become a balancing act of trade-offs and compromises that are economic or political in nature. For example, you may wish to create a product with an enormous feature set, but the project manager and project sponsor will not permit this, as it would cause the project to exceed the deadline and budget constraints. Or the company president may insist that the product be decorated with bright neon pink and green branding, and because the president holds the power, you have little influence.

Understanding who the relevant stakeholders are in your project is a good first step to eliciting and managing requirements. Examples of stakeholders that you might encounter are:

- The **project sponsor**, usually a senior executive or an entrepreneur who initiates a project and makes available the funding and resources. The sponsor will tend to work to protect the project from competing forces and agendas in the organization.
- The **project manager** usually reports to the project sponsor and is responsible for planning the project and seeing it through to successful completion.
- Team members in the project.
- Other managers, departments, and specialists that you must interact with in your organization (e.g., marketing department, legal department).
- Government bodies and regulatory authorities that create legislation and rules that must be complied with.

- The **client**, for a custom development project, or **customers**, for products sold to a market.
- **End users**, the people who will be actually operating your product.
- **Secondary users**, discussed below.

Users vs. buyers/customers

For most consumer products, the user has purchased the product for their own personal use, and so the user is the customer.

But for many products, the end user and the buyer or customer are not always the same. End users use the product in a hands-on manner on a day-to-day basis. The customer is the person who makes the final decision to purchase the product.

For enterprise software such as enterprise resource planning (ERP) systems, high-level company executives will make the decision to purchase the system, but possibly may never operate the software themselves. Similarly, young children may influence purchasing decisions for educational computer games, but it is the parents or other adults who will actually buy the product. Products purchased as gifts are another example of where the customer is not the end user.

So while user experience design and usability testing must focus on the needs of the end users, if your users and customers are separate groups, your product must also simultaneously appeal to the people who will actually buy it.

Secondary users

Some products are used by staff members of an organization who use the product to carry out work in order to serve the organization's customers. The staff members are the end users or **primary users** of the product, and the customers whom the end users serve are called **secondary users**.

Imagine the case of financial aid advisors at a college who use a software system to process student loan applications. Students speak with the advisors but do not use the software directly. The advisors are the primary or end users of the system, while the students are the secondary users.

While secondary users do not use a system directly, they are impacted by it, and a system that leads to poor service delivery to the secondary users cannot be considered

successful.

Eliciting requirements from stakeholders

Once you know who your stakeholders are, you can then work towards understanding what they expect and need from the project and the product.

Interviewing stakeholders is the most common and most effective technique. Interviews may be **structured**, following a pre-prepared list of questions, or may be more **free-flowing**, allowing the interviewer to adjust the flow of the conversation to delve deeper and obtain further information when interesting or unexpected issues and opinions are expressed.

Questionnaire surveys offer the chance to collect responses from a large group, but restrict the feedback to a set list of questions with limited opportunities to probe into individuals' particular concerns and ideas. Surveys can be useful for validating your assumptions, however; you can determine what percentage of the stakeholders say they desire or would use various features.

For the end users who will actually use the product, the technique of **user observation** — watching users as they go about performing their tasks, and asking questions about *why* they do the things the way they do — is extremely valuable. During an observation session, the user may be performing a manual task that will be automated by your product, or the user may be operating a prototype of your product, an older version of your product, or a competing product.

In user observation sessions, you may consider using the **think-aloud protocol** technique, in which you ask the users to vocalize their thoughts and explain what they is doing at each step. While this can provide valuable insights, it can disrupt the flow of work, as many users find it difficult and unnatural to do, and users can feel embarrassed if they make a mistake while being so closely monitored.

Analytics, the collection and analysis of data such as **usage logs** and **usage statistics,** can be provide useful insights into how existing products are being used, including what features are being used most frequently, and where users are running into problems. This can help identify and prioritize requirements.

Requirements elicitation can also be aided with **research and analysis of source material** such as relevant books, academic literature, white papers, market research reports, and so on. If your project involves creating a new version of an existing product, inspecting past project documentation such as specifications, test results, and reports can provide a large source of material; this is sometimes called **documentation archaeology**.

You do need to be careful not to rely too heavily on past documents and repeat mistakes if the past projects were not deemed to be successful.

For the purposes of designing usable interfaces, of course, the most important stakeholders to consider are the end users of the product. Let's now examine how to understand your users' characteristics.

Understanding your users

You will want to find out characteristics of your end users (who we'll just call *users* from here on) and the work they will do with your product.

While every user is a unique individual, users will share common characteristics, and these commonalities are often most pronounced for role-related subgroupings.

User segments and roles

Many products can be used by different groups or categories of users. These groups, or **user segments**, have different goals and reasons for using the product, and in some cases the groups have distinct demographics.

Websites and software use **roles** to restrict access to functionality to different user segments. For instance, a web-based discussion forum will assign most users a "contributor" role, which permits posting and replying to messages, but some trusted users will be assigned the "moderator" role, enabling them to additionally edit and delete messages and ban problematic contributors from participating.

Other products may be general-purpose tools that can be used for various purposes, or may offer a wealth of features that users may use in various combinations. These products don't have explicit roles. For example, word processors may be used to write letters, business documents, essays, reports, novels, technical books, journalism articles, diary entries, webpages, and so on, and the needs of users for each case could be different. You'd also expect a word processor to be used by a very diverse user community — students, professionals, and home users, with varying ranges of educational attainment, and in some cases, with various physical impairments. User segments could be created based on these different groups, or based on the different uses of the product.

Understanding what user segments and roles are relevant for your product is an important early step, as the requirements of users in each group will drive the design of your product.

User characteristics

Once you have made an initial list of user segments or roles for your product, your next step is to understand the general characteristics of users in each group.

The following is a list of some of the characteristics you might want to know about each user segment. Not all characteristics are relevant for all types of product — some may only be appropriate for software used at a workplace, for instance.

- Age
- Gender
- Educational background
- Language and culture
- Computing skills
- Physical abilities and disabilities
- Domain-related knowledge and skills (e.g., accounting knowledge for an accounting application)
- Job experience and competence
- Place in the organizational hierarchy
- Attitudes, motivation, and morale
- Persistence, patience, confidence, problem-solving ability, curiosity, ability to deal with change, etc.
- Frustrations and problems relating to the user's tasks or activities
- General sources of stress or anxiety (e.g., deadlines, performance targets, workplace competition)

Additionally, you may also give some though to the context in which your users will use your product:

- The physical environment (e.g., home, office, factory, vehicle, oil rig, on-the-go in an urban environment, etc.)
- The social environment (position within organization, relationship to other groups, political and interpersonal factors, degree of freedom, influence in decision-making, etc.)

Since every individual is unique, there is a risk of creating generalized stereotypes that doesn't accurately describe many users in a user segment. So for some characteristics, you might describe an approximate range and an average. For instance, bank tellers at a particular financial institution might happen to range in age from 20 to 50 years of age, with the average age being 28.

For each user segment or role, you may want to write up the characteristics as a brief **profile**, which can then be reviewed and discussed with your project team. A profile for a user segment can be presented simply as a list with relevant characteristics described in point form. You might use a matrix to compare user segments side-by-side. Alternatively, you can represent user segments by means of *personas*.

Personas

Personas are a modelling technique introduced and popularized by Alan Cooper.

A **persona** or **user persona** is a brief textual description of a fictional character who is representative of a stereotypical user in a user segment. The persona describes some invented personal details about the character, explains in general terms what they do with the product, discusses the context or environment in which they use the product, and mentions some of the problems, frustrations, and concerns that they might face.

You should invent a fictional name and a descriptive title for each persona; for example, Carol Jones, an insurance adjuster, or Bob Davis, a middle-aged man wishing to book a vacation for his family. To make the persona more tangible and memorable, it's common to write a paragraph or so with a narrative of the person's background or history, and you might even include a photograph, perhaps selected from a stock photo repository.

For example, a sample persona for a new word processing product might be as follows:

> **Persona: Alice Smith, aspiring first-time novelist**
>
> *Alice is 32 years old, divorced, and has a two-year-old daughter. She has a bachelor's degree in English Literature from a state university, and currently works full-time as a marketing assistant for a major pharmaceutical firm. Alice sees herself as a very artistic person, and dislikes the fact that her corporate job offers no real opportunities to express herself creatively. She is particularly passionate about reading historical fiction, and so she now wishes to pursue her long-time dream of writing her own novel in this genre.*

Because of her job and child-care responsibilities, Alice tries to get her writing done in small blocks either early in the morning, or late in the evening after she has put her daughter to bed. On weekends, she enjoys writing while sitting in her favorite coffee shop. She is frustrated that it takes her a lot of time to get warmed up and get into the "flow" of writing, and because she only has short windows of time to work on her book, she doesn't feel that she is making very good progress.

Alice is currently using Microsoft Word on her MacBook laptop, but she is finding it difficult to organize her notes, plan out her plot structure, and generate an outline. She has character and plot notes and multiple chapters spread across multiple Word documents, as well as hand-written notes in a notebook and on various loose scraps of paper, and switching between these is becoming a hassle. She feels she could be more productive if she could just get a better handle on organizing all of her project materials.

She would like to have her novel published by a big-name publisher, but she knows that this is difficult for a first-time fiction author. She is considering self-publishing, but has not had an opportunity to research what is involved, and she is particularly worried about the costs involved. She would also like to make her novel available on e-book readers like the Kindle, but she is unsure of how to convert her manuscript to an e-book format and get it listed on major booksellers' sites like Amazon.com.

The advantage of personas is that they help your team develop a shared understanding of the requirements of each user segment, and they encourage empathy with the end users of your product. By bringing to life someone from each user segment, with a human name and face, designers can design specifically with that particular user in mind. When designing a task flow or a screen, you might ask, "How would Alice want this information presented?" or "How would Alice react in this situation?"

Personas can also help focus usability tests and evaluations. For preliminary evaluations of a prototype, tests could be carried out by somebody pretending to act in the role of the persona, or at least with that persona in mind. (Of course, it is preferable to involve real users from the user segments.)

You can also use the personas to prioritize the functional requirements. For instance, one release could concentrate on the core features needed by the Alice persona, and then the next release could concentrate on another persona.

Personas are not without criticism; many people find the invention of fictional names and personal details to be highly gimmicky. Trying to represent a diverse user segment

with a single persona runs the risk that the persona doesn't accurately represent a large percentage of the members of the user segment. And if you try to write up personas without ever actually speaking to real live users, these completely hypothetical personas may have no relation to reality.

From user characteristics to requirements

User characteristics are not requirements, but characteristics can serve as a source of requirements.

For example, if your user community is distributed around the world, and your users prefer to, or must, use their native languages rather than English, then you can establish a requirement that the product be available in various localized forms for different countries and languages. Or if your user community includes people with mild visual impairment (they need to wear glasses), you will likely want to have a requirement for the ability to adjust the font size.

Understanding the work and tasks

Software exists to get things done, and the things that users want to get done we will call **work**.

Work is a suitable term for the activities that users carry out with productivity applications, where the goal is to produce a deliverable of value, such as an essay, a financial statement, or a poster. It is also suitable when referring to business systems that users operate to provide services to customers or to manage and administer the organization.

Of course, there is a vast category of entertainment and communications software where the activities, like watching cat videos, shooting alien invaders, or flirting, revolve around recreation and socializing rather than producing something of commercial, intellectual, or artistic value. For the purposes of this book, however, we'll still refer to the activity that users do with the software as "work".

Work exists in a particular **domain** and **context**. Banking software, for example, obviously deals with the domain of banking, where concepts like accounts, transactions, loans, interest, and fees are relevant. And banking software is used in the context of a specific organization — in this case, a particular bank. So the work that users (bank tellers, loan officers, and so on in this case) do with banking software thus involves users working with the domain objects in the context of serving customers to further the organizational aims of the bank.

The process of designing your software product, then, usually revolves largely around understanding the domain and the work, choosing what parts of the work your product will support, and then specifically designing how your product will help the user carry out that work.

Users' work often consists of multiple **tasks**. In some applications, tasks can be highly structured, with a defined sequence of actions. For example, registering a new member into a medical insurance plan would have a very formally defined workflow. In other applications, tasks are more fluid, and users may have many things on the go at once. For example, while retouching a photo, a user may use various drawing tools and switch between them frequently while working towards the vague goal of making the photograph look more attractive.

Different user segments will typically engage in different kinds of work, so some tasks will only be relevant for certain user roles. There may be many tasks shared between roles, and sometimes tasks may be used by different user segments to achieve different ends.

When analyzing and designing the work and tasks relevant for your product, some questions to keep in mind include:

- How do users do the task currently? Do they use a competing product in the same category? Do they use a different type of product or some combination of products? Do they currently lack a technology-based solution?
- What are users' biggest complaints about the way they currently do the task? What are the biggest problems for them (or for their organization, in a work context)? What are the biggest time sinks, the major causes of errors, the sources of frustration?
- Are there any laws or regulations that dictate how the task is to be done (e.g., safety rules, the tax code, Generally Accepted Accounting Principles, etc.)?

The work and tasks will give rise to a large number of functional requirements for what features the product must offer and how the features should work. (In Chapter 14, we'll revisit the topics of understanding and modelling work and take a closer look at designing and documenting task flows and interactions.)

Because work is intricately linked with the application's problem domain, a critical part of understanding and modelling the work and tasks involves understanding and modelling the domain and the data your application will deal with. We'll examine this in Chapter 7.

Documenting requirements

On the basis of the information you've gathered about your users, the domain, and the work and tasks, you'll want to begin recording requirements for your product.

For each requirement, you will typically want to record the following information, at the minimum:

- A unique requirement ID number or code, for ease of reference
- A "headline" or summary description of the requirement
- A more detailed description, if necessary
- Acceptance criteria, i.e., some specific test or measurement that can be done to verify that the requirement has been fulfilled
- The role(s) for which the requirement is applicable
- Dependencies or relationships between this requirement and other requirements

You'll also usually want to have some way of indicating the prioritization, and you'll typically need some form of categorization scheme. You may or may not wish to record project management information, such as an estimate of the work involved, target milestones or deadlines, and so on.

For purposes of traceability, you will likely also want to indicate who created the requirement, and the date the requirement was created and/or approved (and ideally, you'd want to be able to track the version history of changes to the requirement, if possible).

It is recommended that you create a template so that all of your requirements include the same information items. An example of a requirement recorded using a template is shown in Figure 6-1.

Requirement ID	65
Summary description	The system shall allow a customer's annual statement to be generated for the current year or any past year on record.
Detailed description	The statement shall include all transactions during the chosen calendar year, and shall include the final balance at the end of the year (or the current balance if the current year has been selected). The generation of the report can be triggered by the customer via a function in the web portal. It can also be generated by a customer service representative or manager.
Acceptance criteria	The generated report contains the appropriate and correct data and content. The report's appearance adheres to the corporate visual design and branding standards. The report is generated in one of the acceptable file formats defined in the project handbook.
Requirement type	Functional
Category	Reporting - Customer Accounts
Priority	Medium
Implementation estimate	5 person-days
Depends on	15, 24, 25, 56, 59
See also	80
Created by	Alice Smith
Created on	09.09.2013
Approval status	Pending review

Figure 6-1

Some teams like to write requirements on paper cards. This is an excellent way to get started when brainstorming and discussing requirements, but of course this tends to become unmanageable in the long term for non-trivial projects. Requirements are often recorded in a spreadsheet, or in a word processing document with a template form that can be duplicated and filled out. There are also specialized requirements management tools available, and many companies and teams have been known to create their own requirements database.

The book *Mastering the Requirements Process* (Robertson and Robertson, 2012) recommends that all requirements pass through a so-called **quality gateway** process before being accepted as being part of the scope of the project. To pass the quality gateway, each requirement is inspected with a basic checklist that tests for completeness, and an authority (such as the project manager, product manager, or a review committee) must formally approve the requirement. This review process should also involve checking for conflicts and contradictions amongst the requirements, so that such issues can be resolved sooner rather than later.

To document in detail how it is intended that the user will interact with the product, and how the product will behave, design techniques like *use cases* are often used in addition to requirements. We'll explore this further in Chapter 14.

User stories

A list of potentially thousands of requirements of the form "The system shall..." makes it hard for readers to get an overall understanding of what the product is intended to do. In the agile community, informal **user stories** are often preferred over formal requirements templates.

User stories are intended to be concise and informal, and serve primarily as a basis for discussion, rather than attempting to capture all possible details related to the requirement.

User stories take the form of a simple statement that states a *goal* of a user, and then gives the *rationale* for why the user would want to achieve that goal.

A user story is a brief summary statement that usually follows the pattern:

> *"As a <role>, I want to <description of function> so that <rationale>."*

This pattern manages to capture "who" (the role), "what" (the function), and "why" (the rationale, i.e., the benefit or justification).

You could omit the rationale clause in cases where it is self-evident, but it is often useful to understand the reason why some piece of functionality should be included in the system, so don't make it a habit to omit it.

Here are some examples of user stories:

- *"As a mobile phone subscriber, I want to be able to block calls from a specific telephone number so that I do not have to be interrupted by repeated nuisance phone calls from telemarketers."*
- *"As a system administrator, I want to be able to create new user accounts so that new hires can access the system."*
- *"As a manager, I want to be able to view the timesheet records of my direct reports so that I can verify that staff are working on relevant tasks and billing appropriately."*

User stories can capture functionality at various levels of detail and abstraction. A high-level user story that encapsulates a very large amount of functionality is called an **epic**, and can be broken down into more detailed user stories.

We should note that non-functional requirements can also be captured as user stories. For instance:

- *"As a user, I want the system's webpages to load in under two seconds so that I can maintain a steady work rhythm without having to stop and wait."*

User stories are intended to be short; the general guideline is that a user story should be able to be written on an index card. But user stories may be accompanied by a more detailed description, sometimes called a "conversation", when more details are required.

User stories should eventually be accompanied by acceptance criteria, such as a test procedure that can be performed to verify that the user story has been implemented as described.

User stories and project management

When used for project planning purposes, user stories should also include an estimates of the amount of work involved for implementation. While the estimate can be stated in person-days, the usual recommendation is to use **story points**, which are an arbitrary measure of relative complexity, effort, and risk. You define your own story points scale by choosing a relatively trivial user story to count as the base unit, worth one story point. If you then rate another user story as being worth 20 story points, then it indicates that the complexity, effort, and risk are such that, on the average, it will take approximately 20 times as long as the base user story to complete.

In the Scrum methodology, user stories are placed on a list called the **product backlog**. The product backlog can include user stories that are wished for, but which may never actually get implemented. At the beginning of each iteration, or **sprint**, the priorities of all of the user stories are reevaluated, and a set of user stories are selected on the basis of

priority and estimates to form the **sprint backlog**, representing the work scheduled to be done in the sprint.

Writing requirements for usability

As discussed briefly earlier, we often find ourselves writing usability requirements such as "The product shall be easy to use" or "The product shall be easy to learn". The problem with these requirements is that they are *qualitative, subjective* statements that express general goals, and it is impossible to objectively prove that the product fulfils these requirements. Different evaluators will have different subjective opinions on whether the product actually is easy to use and learn.

Acceptance criteria for such requirements should be stated in terms of **usability metrics**, which are *quantitative* performance measurements of some usability-related factor. The process of finding quantitative, measurable proxy indicators for a qualitative concept or phenomenon is called **operationalization**.

For example, we can take the requirement "The product shall be easy to use" and operationalize it with acceptance criteria such as:

- *"An average trained user shall be able to process a standard application form containing no special cases in under five minutes."*
- *"An average user shall not have to access the online help system more than twice per hour under normal operating conditions."*
- *"An average user shall report no more than five annoyances per hour of system operation."*

Usability metrics

Tyldesley (1988), who summarizes material from Whiteside *et al.* (1998), suggests the following factors that could used in formulating usability metrics:

1. Time to complete a task
2. Percentage of task completed
3. Percentage of task completed per unit time (speed metric)
4. Ratio of successes to failures

5. Time spent on errors
6. Percentage or number of errors
7. Percentage or number of competitors that do some particular aspect better than the current product
8. Number of commands used
9. Frequency of help or documentation use
10. Time spent using help or documentation
11. Percentage of favorable to unfavorable user commands
12. Number of repetitions of failed commands
13. Number of runs of successes and of failures
14. Number of times the interface misleads the user
15. Number of good and bad factors recalled by users
16. Number of available commands not invoked
17. Number of regressive behaviors
18. Number of users preferring your product
19. Number of times users need to work around a problem
20. Number of times the user is disrupted from a work task
21. Number of times the user loses control of the system
22. Number of times the user expresses frustration or satisfaction

You can also ask evaluators to judge usability, learnability, satisfaction, or other subjective factors using a rating scale. Statistical derivations such as average values can then serve as a form of measurement. There are reliability issues such as bias that can threaten the validity of the results, so you do need to be careful when drawing conclusions from such data.