L08 Usability

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Harvinder S Jabbal
CSIS, Work Integrated Learning Programs

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SEZG651/ SSZG653 Software Architectures Module 2-CS 02

Chapter Outline

innovate achieve lead

What is Usability?

Usability General Scenario

Tactics for Usability

A Design Checklist for Usability

Summary

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

What is Usability?

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.

Over the years, a focus on usability has shown itself to be one of the cheapest and easiest ways to improve a system's quality (or, more precisely, the user's *perception* of quality).

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

What is Usability?

Usability comprises the following areas:

- Learning system features.
- Using a system efficiently.
- Minimizing the impact of errors.
- Adapting the system to user needs.
- Increasing confidence and satisfaction.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Usability General Scenario

Portion of Scenario	Possible Values
Source	End user, possibly in a specialized role
Stimulus	End user tries to use a system efficiently, learn to use the system, minimize the impact of errors, adapt the system, or configure the system
Environment	Runtime or configuration time
Artifacts	System or the specific portion of the system with which the user is interacting.
Response	The system should either provide the user with the features needed or anticipate the user's needs.
Response Measure	One or more of the following: task time, number of errors, number of tasks accomplished, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, or amount of time or data lost when an error occurs.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Sample Concrete Usability Scenario

The user downloads a new application and is using it productively after two minutes of experimentation.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Goal of Usability Tactics

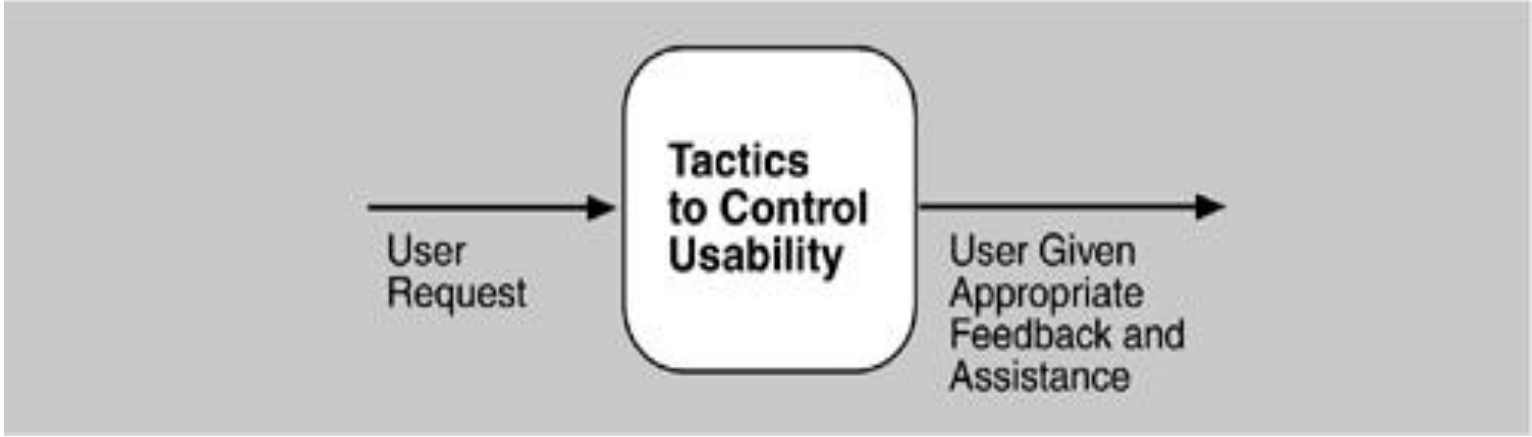
Researchers in human-computer interaction have used the terms "user initiative," "system initiative," and "mixed initiative" to describe which of the human-computer pair takes the initiative in performing certain actions and how the interaction proceeds.

Usability scenarios can combine initiatives from both perspectives.

We use this distinction between user and system initiative to discuss the tactics that the architect uses to achieve the various scenarios.

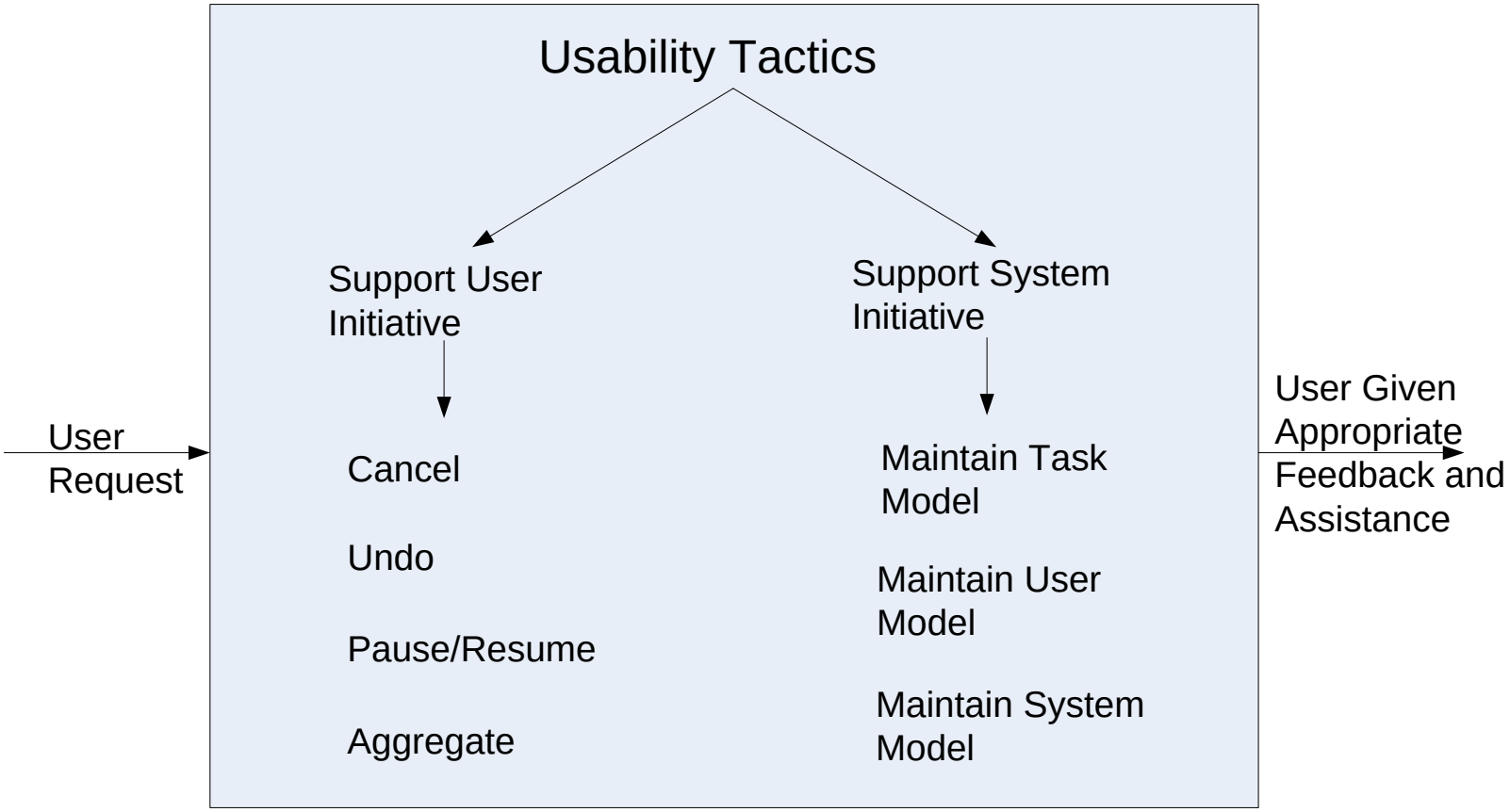
© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Goal of Usability Tactics



© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Usability Tactics



© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Support User Initiative

Cancel: the system must listen for the cancel request; the command being canceled must be terminated; resources used must be freed; and collaborating components must be informed.

Pause/Resume: temporarily free resources so that they may be re-allocated to other tasks.

Undo: maintain a sufficient amount of information about system state so that an earlier state may be restored, at the user's request.

Aggregate: ability to aggregate lower-level objects into a group, so that a user operation may be applied to the group, freeing the user from the drudgery.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Support System Initiative

Maintain Task Model: determines context so the system can have some idea of what the user is attempting and provide assistance.

Maintain User Model: explicitly represents the user's knowledge of the system, the user's behavior in terms of expected response time, etc.

Maintain System Model: system maintains an explicit model of itself. This is used to determine expected system behavior so that appropriate feedback can be given to the user.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Design Checklist for Usability

Allocation of Responsibilities	**Ensure that additional system responsibilities have been allocated, as needed, to assist the user in** • learning how to use the system • efficiently achieving the task at hand • adapting and configuring the system • recovering from user and system errors

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Design Checklist for Usability

Coordination Model	Determine whether the properties of system elements' coordination—timeliness, currency, completeness, correctness, consistency—affect how a user learns to use the system, achieves goals or completes tasks, adapts and configures the system, recovers from user and system errors, increases confidence and satisfaction. For example, can the system respond to mouse events and give semantic feedback in real time? Can long-running events be canceled in a reasonable amount of time?

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Design Checklist for Usability

Data Model	Determine the major data abstractions that are involved with user-perceivable behavior. Ensure these major data abstractions, their operations, and their properties have been designed to assist the user in achieving the task at hand, adapting and configuring the system, recovering from user and system errors, learning how to use the system, and increasing satisfaction and user confidence For example, the data abstractions should be designed to support undo and cancel operations: the transaction granularity should not be so great that canceling or undoing an operation takes an excessively long time.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Design Checklist for Usability

Mapping Among Architectural Elements	Determine what mapping among architectural elements is visible to the end user (for example, the extent to which the end user is aware of which services are local and which are remote). For those that are visible, determine how this affects the ways in which, or the ease with which the user will learn how to use the system, achieve the task at hand, adapt and configure the system, recover from user and system errors, and increase confidence and satisfaction.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Design Checklist for Usability

Resource Management	Determine how the user can adapt and configure the system's use of resources. Ensure that resource limitations under all user-controlled configurations will not make users less likely to achieve their tasks. For example, attempt to avoid configurations that would result in excessively long response times. Ensure that the level of resources will not affect the users' ability to learn how to use the system, or decrease their level of confidence and satisfaction with the system.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Design Checklist for Usability

Binding Time	Determine which binding time decisions should be under user control and ensure that users can make decisions that aid in usability. For example, if the user can choose, at run-time, the system's configuration, or its communication protocols, or its functionality via plug-ins, you need to ensure that such choices do not adversely affect the user's ability to learn system features, use the system efficiently, minimize the impact of errors, further adapt and configure the system, or increase confidence and satisfaction.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Design Checklist for Usability

Choice of Technology	Ensure the chosen technologies help to achieve the usability scenarios that apply to your system. For example, do these technologies aid in the creation of on-line help, training materials, and user feedback collection. How usable are any of your chosen technologies? Ensure the chosen technologies do not adversely affect the usability of the system (in terms of learning system features, using the system efficiently, minimizing the impact of errors, or adapting/configuring the system, increase confidence and satisfaction).

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Summary

Architectural support for usability involves both allowing the user to take the initiative in circumstances such as cancelling a long running command, undoing a completed command, and aggregating data and commands.

To predict user or system response, the system must keep a model of the user, the system, and the task.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Other Quality Attributes

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Chapter Outline

Other Important Quality Attributes

Other Categories of Quality Attributes

Software Quality Attributes and System Quality Attributes

Using Standard Lists of Quality Attributes

Dealing with “X-ability”

Summary

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Other Important Quality Attributes

Variability: is a special form of modifiability. It refers to the ability of a system and its supporting artifacts to support the production of a set of variants that differ from each other in a preplanned fashion.

Portability: is also a special form of modifiability. Portability refers to the ease with which software that built to run on one platform can be changed to run on a different platform.

Development Distributability: is the quality of designing the software to support distributed software development.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Other Important Quality Attributes

Scalability: Horizontal scalability (scaling out) refers to adding more resources to logical units such as adding another server to a cluster. Vertical scalability (scaling up) refers to adding more resources to a physical unit such as adding more memory to a computer.

Deployability: is concerned with how an executable arrives at a host platform and how it is invoked.

Mobility: deals with the problems of movement and affordances of a platform (e.g. size, type of display, type of input devices, availability and volume of bandwidth, and battery life).

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Other Important Quality Attributes

Monitorability: deals with the ability of the operations staff to monitor the system while it is executing.

Safety: Software safety is about the software's ability to avoid entering states that cause or lead to damage, injury, or loss of life, and to recover and limit the damage when it does enter into bad states. The architectural concerns with safety are almost identical with those for availability (i.e. preventing, detecting, and recovering from failures).

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Other Categories of Quality Attributes

Conceptual Integrity: refers to consistency in the design of the architecture. It contributes to the understandability of the architecture. Conceptual integrity demands that the same thing is done in the same way through the architecture.

Marketability: Some systems are marketed by their architectures, and these architectures sometimes carry a meaning all their own, independent of what other quality attributes they bring to the system (e.g. service-oriented or cloud-based).

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Other Categories of Quality Attributes

Quality in Use: qualities that pertain to the use of the system by various stakeholders. For example

- Effectiveness: a measure whether the system is correct
- Efficiency: the effort and time required to develop a system
- Freedom from risk: degree to which a product or system affects economic status, human life, health, or the environment

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

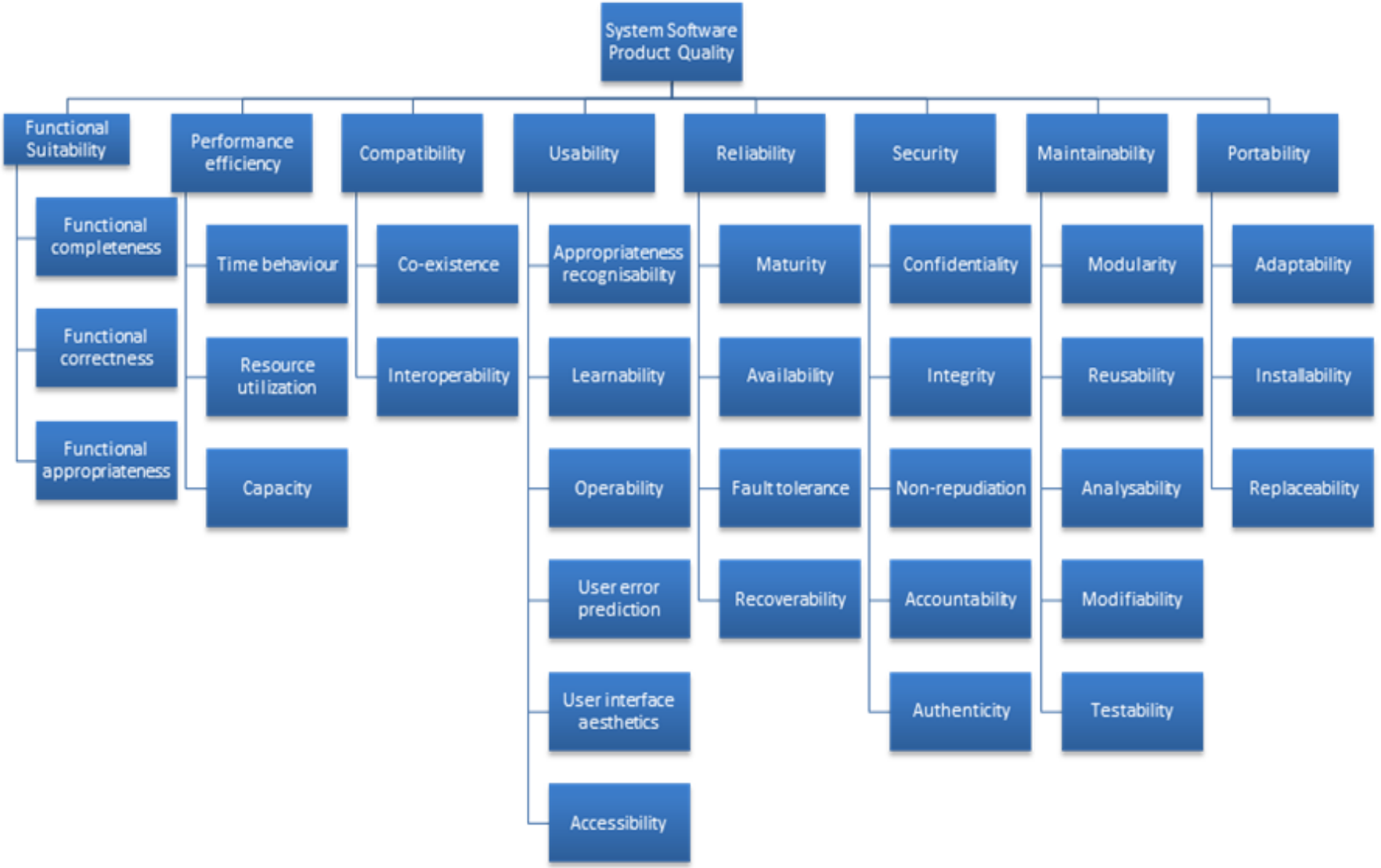
Software Quality Attributes and System Quality Attributes

Physical systems, such as aircraft or automobiles or kitchen appliances, that rely on software embedded within are designed to meet a whole other litany of quality attributes: weight, size, electric consumption, power output, pollution output, weather resistance, battery life, and on and on.

The software architecture can have a substantial effect on the system's quality attributes.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Standard Lists of Quality Attributes



Kazman, distributed under Creative Commons Attribution License

Standard Lists of Quality Attributes

Advantages:

- Can be helpful checklists to assist requirements gatherers in making sure that no important needs were overlooked.
- Can serve as the basis for creating your own checklist that contains the quality attributes of concern in your domain, your industry, your organization, your products, …

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Standard Lists of Quality Attributes

Disadvantages:

- No list will ever be complete.
- Lists often generate more controversy than understanding.
- Lists often purport to be *taxonomies*. But what is a denial-of-service attack?
- They force architects to pay attention to every quality attribute on the list, even if only to finally decide that the particular quality attribute is irrelevant to their system.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Dealing with “X-ability”

Suppose you must deal with a quality attribute for which there is no compact body of knowledge, e.g. green computing.

What do you do?

1. Model the quality attribute
2. Assemble a set of tactics for the quality attribute
3. Construct design checklists

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Summary

There are many other quality attributes than the seven that we cover in detail.

Taxonomies of attributes may offer some help, but their disadvantages often outweigh their advantages.

You may need to design or analyze a system for a “new” quality attribute. While this may be challenging, it is doable.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License