Name: **ZOHAIB HASSAN SOOMRO**

RollNo#: **19SW42**

Subject:



**Usability Tactics:**

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of support the system provides to the user. Two types of tactics support usability, each intended for two categories of "users." The first category, runtime, includes those that support the user during system execution. The second category is based on the iterative nature of user interface design and supports the interface developer at design time.

1. **Run-Time Tactics**:-

It is the management of applications at run-time. It includes:

**Maintain a Model of the Task:**

* It is to manage work patterns and their use by categories. Clearly manage each type of usage. Reduce duplication and have standards for users to understand and use, for example, a banking transaction system that has a model of use separated by location and device, separated by usage of branch staff and customers, separated by device into Operate via ATM, mobile phone or tablet, PC or notebook.

**Maintain a Model of the User:**

* It is to manage work and usage patterns by user type, such as screens for employees, separate screens for customers, displaying and working according to individual user rights that may be different. This technique can be very useful if users have a variety of characteristics or behaviours. Therefore, it is best to start by analysing and classifying user attributes or behaviours.

**Maintain a Model of the System:**

* It is the management of work patterns and usage by system such as

Processing bank transactions that receive requests from ATMs should take a short time. Because ATMs are physical, if there are many people in line, it will make people wait longer. This is different from using internet banking and mobile banking.

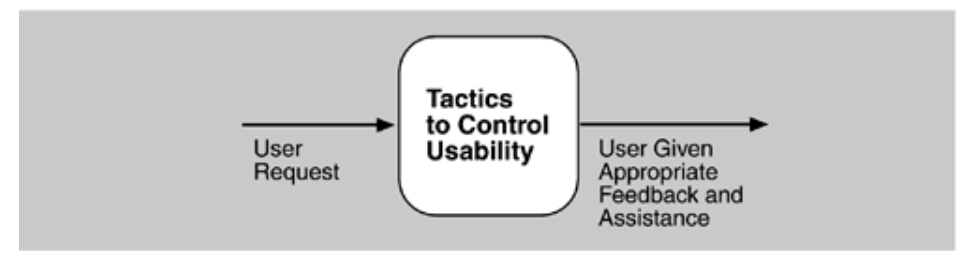
**Example:** The 'Ratchadamnoen' room on pantip.com is a room where politics is mainly discussed. Therefore, there are stricter controls for language, content, and comments than any other room on the web etc.

1. **Design-Time Tactics**:-

It is the management of applications at design-time, It includes:

**Separate User Interface:**

* It is the separation of the display or interface between the user and the functional part of the system, for example, the presentation of the application is separated from the application to allow for more independent management of the display, which may use different patterns. It helps a lot such as
  + Model-View-Controller (MVC)
  + Presentation-Abstraction-Control (PAC)
  + View Helper
  + Dispatcher View
  + etc.



**RUNTIME TACTICS**

Once a system is executing, usability is enhanced by giving the user feedback as to what the system is doing and by providing the user with the ability to issue usability-based commands . For example, *cancel, undo, aggregate*, and *show multiple views* support the user in either error correction or more efficient operations.

Researchers in human-computer interaction have used the terms "user intiative," "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. The usability scenarios we enumerated in [Chapter 4](https://people.ece.ubc.ca/matei/EECE417/BASS/ch04.html#ch04), Understanding Quality Attributes, combine initiatives from both perspectives. For example, when canceling a command the user issues a cancel-"user initiative"-and the system responds. During the cancel, however, the system may put up a progress indicator-"system initiative." Thus, cancel demonstrates "mixed initiative." We use this distinction between user and system initiative to discuss the tactics that the architect uses to achieve the various scenarios.

When the user takes the initiative, the architect designs a response as if for any other piece of functionality. The architect must enumerate the responsibilities of the system to respond to the user command. To use the cancel example again: When the user issues a cancel command, the system must be listening for it (thus, there is the responsibility to have a constant listener that is not blocked by the actions of whatever is being canceled); the command to cancel must be killed; any resources being used by the canceled command must be freed; and components that are collaborating with the canceled command must be informed so that they can also take appropriate action.

When the system takes the initiative, it must rely on some information-a model-about the user, the task being undertaken by the user, or the system state itself. Each model requires various types of input to accomplish its initiative. The system initiative tactics are those that identify the models the system uses to predict either its own behavior or the user's intention. Encapsulating this information will enable an architect to more easily tailor and modify those models. Tailoring and modification can be either dynamically based on past user behavior or offline during development.

* *Maintain a model of the task.* In this case, the model maintained is that of the task. The task model is used to determine context so the system can have some idea of what the user is attempting and provide various kinds of assistance. For example, knowing that sentences usually start with capital letters would allow an application to correct a lower-case letter in that position.
* *Maintain a model of the user.* In this case, the model maintained is of the user. It determines the user's knowledge of the system, the user's behavior in terms of expected response time, and other aspects specific to a user or a class of users. For example, maintaining a user model allows the system to pace scrolling so that pages do not fly past faster than they can be read.
* *Maintain a model of the system.* In this case, the model maintained is that of the system. It determines the expected system behavior so that appropriate feedback can be given to the user. The system model predicts items such as the time needed to complete current activity.

#### DESIGN-TIME TACTICS

User interfaces are typically revised frequently during the testing process. That is, the usability engineer will give the developers revisions to the current user interface design and the developers will implement them. This leads to a tactic that is a refinement of the modifiability tactic of semantic coherence:

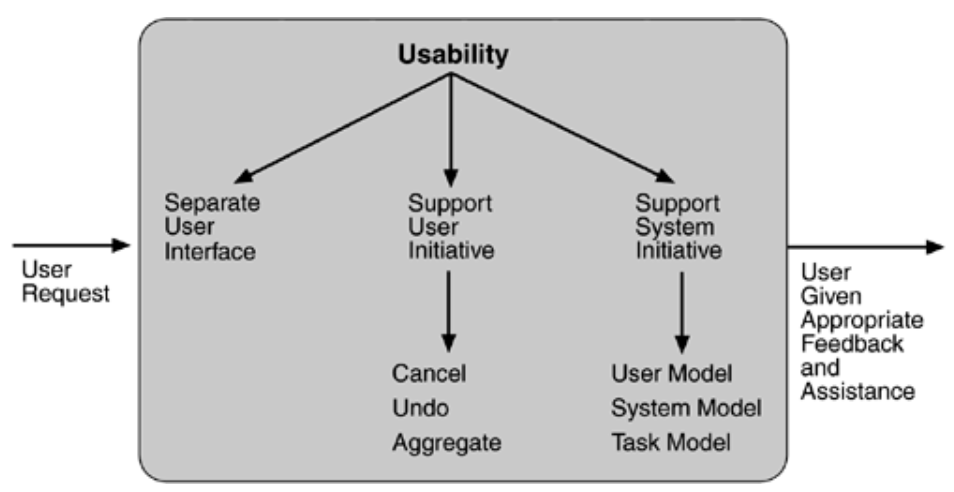
* *Separate the user interface from the rest of the application.* Localizing expected changes is the rationale for semantic coherence. Since the user interface is expected to change frequently both during the development and after deployment, maintaining the user interface code separately will localize changes to it. The software architecture patterns developed to implement this tactic and to support the modification of the user interface are:

- Model-View-Controller

- Presentation-Abstraction-Control

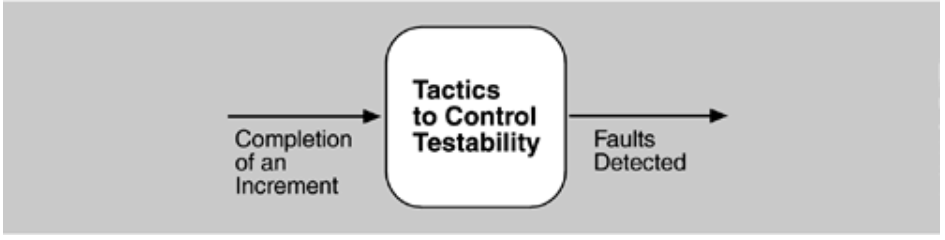
- Seeheim

- Arch/Slinky



**Testability Tactics:**

The goal of tactics for testability is to allow for easier testing when an increment of software development is completed. Architectural techniques for enhancing the software testability have not received as much attention as more mature fields such as modifiability, performance, and availability, but, since testing consumes such a high percentage of system development cost, anything the architect can do to reduce this cost will yield a significant benefit.



The goal of a testing regimen is to discover faults. This requires that input be provided to the software being tested and that the output be captured.

Executing the test procedures requires some software to provide input to the software being tested and to capture the output. This is called a test harness. A question we do not consider here is the design and generation of the test harness. In some systems, this takes substantial time and expense.

We discuss two categories of tactics for testing: providing input and capturing output, and internal monitoring.

INPUT/OUTPUT

There are three tactics for managing input and output for testing.

* *Record/playback.* Record/playback refers to both capturing information crossing an interface and using it as input into the test harness. The information crossing an interface during normal operation is saved in some repository and represents output from one component and input to another. Recording this information allows test input for one of the components to be generated and test output for later comparison to be saved.
* *Separate interface from implementation.* Separating the interface from the implementation allows substitution of implementations for various testing purposes. Stubbing implementations allows the remainder of the system to be tested in the absence of the component being stubbed. Substituting a specialized component allows the component being replaced to act as a test harness for the remainder of the system.
* *Specialize access routes/interfaces.* Having specialized testing interfaces allows the capturing or specification of variable values for a component through a test harness as well as independently from its normal execution. For example, metadata might be made available through a specialized interface that a test harness would use to drive its activities. Specialized access routes and interfaces should be kept separate from the access routes and interfaces for required functionality. Having a hierarchy of test interfaces in the architecture means that test cases can be applied at any level in the architecture and that the testing functionality is in place to observe the response.

INTERNAL MONITORING

A component can implement tactics based on internal state to support the testing process.

* *Built-in monitors.* The component can maintain state, performance load, capacity, security, or other information accessible through an interface. This interface can be a permanent interface of the component or it can be introduced temporarily via an instrumentation technique such as aspect-oriented programming or preprocessor macros. A common technique is to record events when monitoring states have been activated. Monitoring states can actually increase the testing effort since tests may have to be repeated with the monitoring turned off. Increased visibility into the activities of the component usually more than outweigh the cost of the additional testing.

