

Lecture Week 4

Semester 2 2023



UNIVERSITY OF
CANBERRA

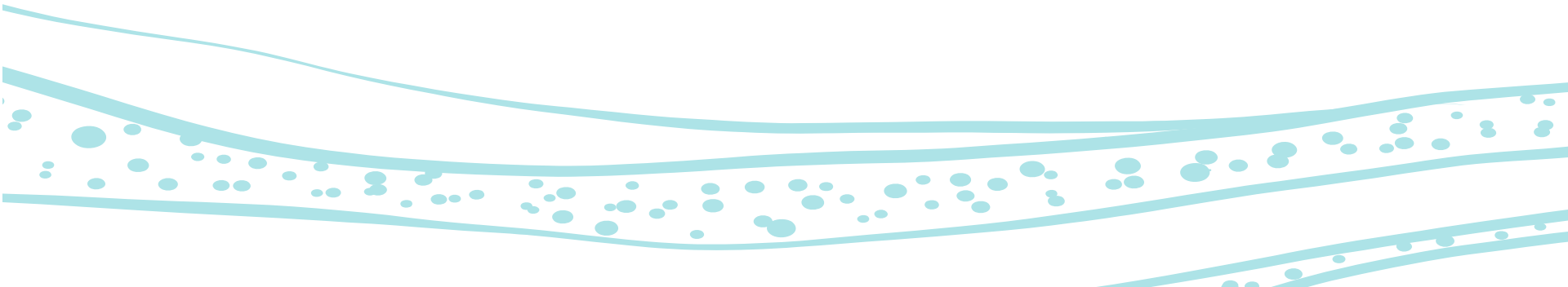
SOFTWARE SYSTEMS ARCHITECTURE UG/G (11491/8746)

Introduction

Richa Awasthy
richa.awasthy@canberra.edu.au

ACKNOWLEDGEMENT OF COUNTRY

The University of Canberra acknowledges the Ngunnawal peoples as the traditional custodians of the land upon which the University's main campus sits. I pay my respect to all Elders past, present and emerging.



Thank you for ISEQ

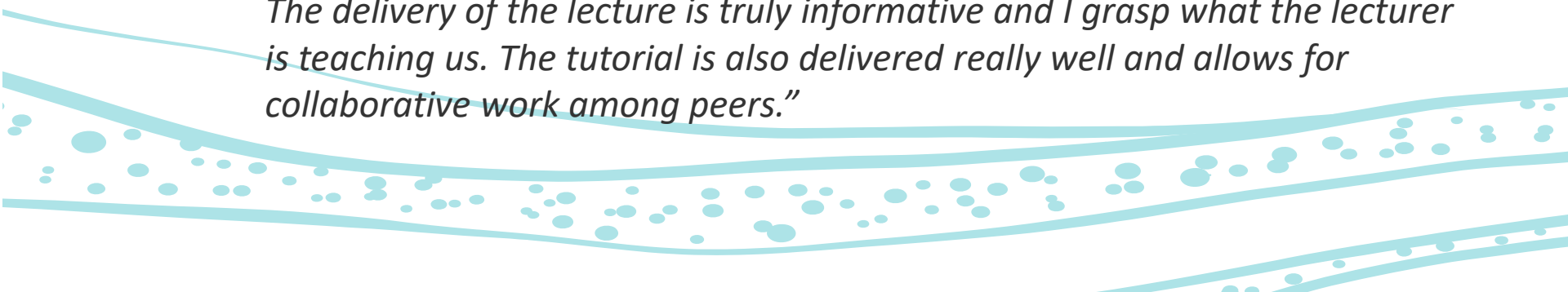
74 participants: Most of them satisfied with the unit delivery

Positive comments:

"Straight forward, nothing negative or criticise on"

"The class so far has been good, the lectures cover everything in depth and everything we need for the tutorials."

"My learning experience with this unit has been really great! The delivery of the lecture is truly informative and I grasp what the lecturer is teaching us. The tutorial is also delivered really well and allows for collaborative work among peers."



Thank you for ISEQ

Suggestions/improvement:

- Break during the lecture
- Ambiguity in quiz questions
- Alerts/notifications during the lecture
- Extent of Examples
- information about the assignments is underemphasized.
- Not sure how current content relates to software systems.
- Having all of the assessment on canvas sooner would have been better
- Forming groups was forced
- It seems all we're doing is writing a report.
- Tutor is rushing
- Tutorials have not had a formal method to submit but popped up

Agenda:

1. **Availability**

- Scenarios
- Tactics

2. **Modifiability**

- Scenarios
- Tactics

3. **Usability**

- Scenarios
- Tactics

1. Availability



Scenarios

What is Availability?

Availability refers to a property of software that it is there and ready to carry out its tasks when required.

- This is a broad perspective and encompasses what is normally called reliability
- Availability builds on reliability by adding the notion of recovery (repair)
- Fundamentally, availability is about minimizing service outage time by mitigating faults

Availability General Scenario

Source	Internal/external: people, hardware, software, physical infrastructure, physical environment.
Stimulus	Fault: omission, crash, incorrect timing, incorrect response.
Artifact	System's processors, communication channels, persistent storage, processes.
Environment	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation.
Response	
Response Measure	

General Scenario - Response

- Prevent fault from becoming a failure
- Detect the fault:
 - Log the fault
 - Notify appropriate entities (people or systems)
- Recover from the fault
 - Disable source of events causing the fault
 - Be temporarily unavailable while repair is being effected
 - Fix or mask the fault/failure or contain the damage it causes
 - Operate in a degraded mode while repair is being effected



General Scenario - Response Measure

- Time or time interval when the system must be available
- Availability percentage (e.g. 99.999%)
- Time to detect the fault
- Time to repair the fault
- Time or time interval in which system can be in degraded mode
- Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing



System Availability Requirements

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hr, 36 min	3 days, 15.6 hr
99.9%	2 hr, 10 min	8 hr, 45 min.
99.99%	12 min, 58 sec	52 min, 34 sec
99.999%	1 min, 18 sec	5 min, 15 sec
99.9999%	8 sec	32 sec

Sample Concrete Availability Scenario

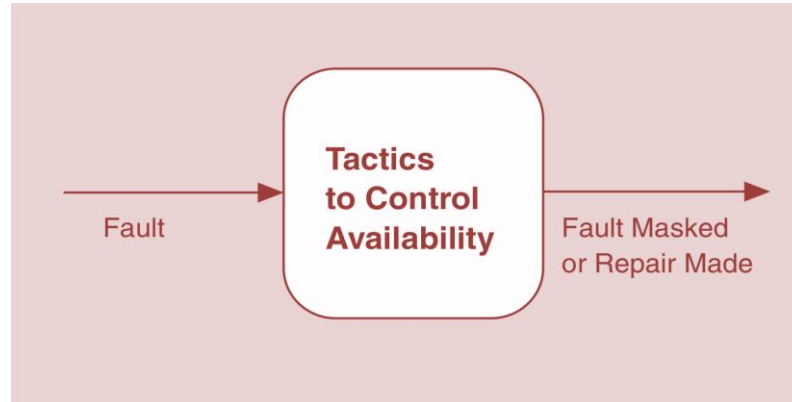
- The heartbeat monitor determines that a server (from a cluster of servers) is nonresponsive during normal operations. The system informs the operator and continues to operate with no downtime.

Source	External: heartbeat monitor.
Stimulus	Fault: a server is nonresponsive.
Artifact	System's processes (from the nonresponsive server).
Environment	Normal operations.
Response	System informs the operator and continues to operate.
Response Measure	No downtime.

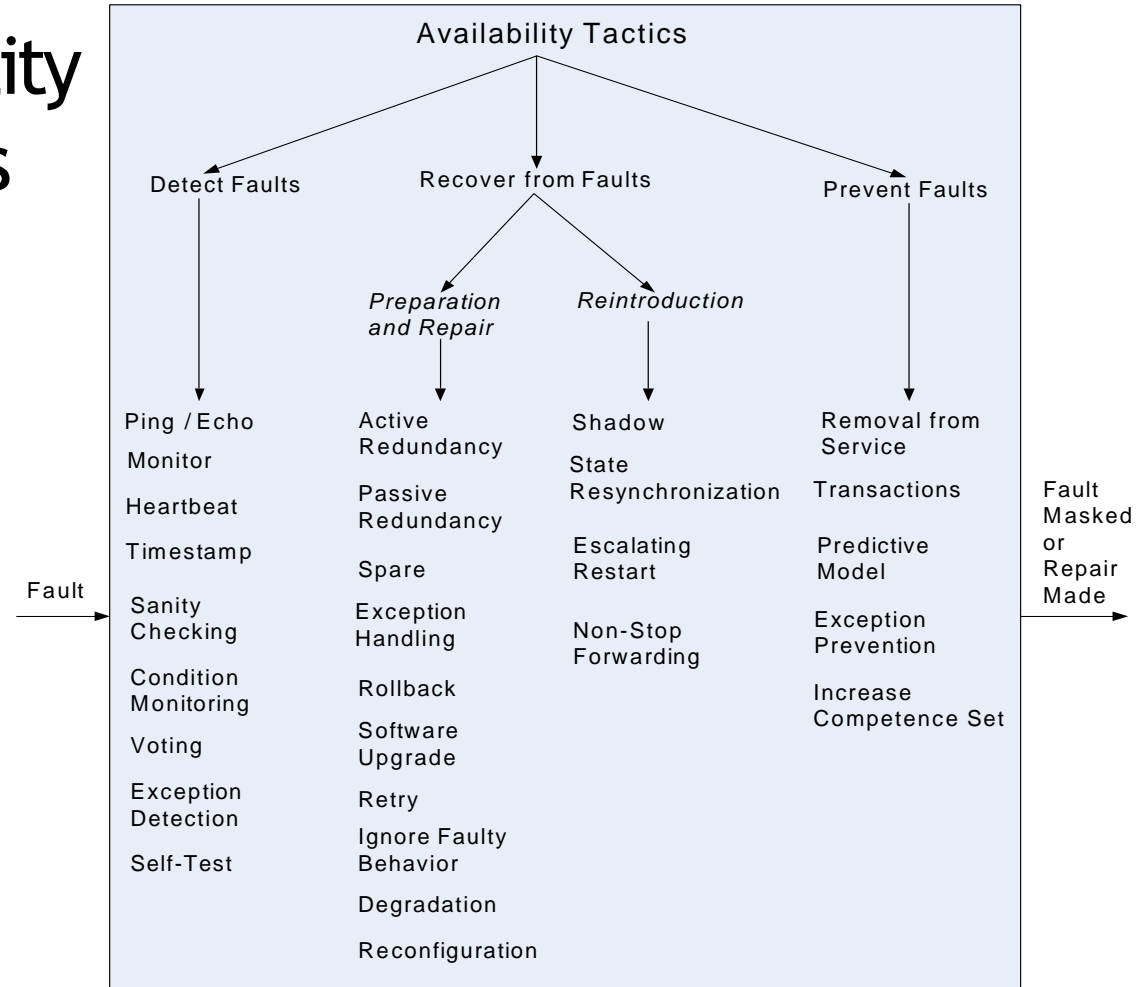
Tactics

Goal of Availability Tactics

- A failure occurs when the system fails to deliver a service which is consistent with its specification
- A fault (or combination of faults) has the potential to cause a failure
- Availability tactics enable a system to endure faults so that services remain compliant with their specifications
- The tactics keep faults from becoming failures or at least bound the effects of the fault and make repair possible



Availability Tactics



Detect Faults

Ping / Echo

- Asynchronous request/response message pair exchanged between nodes
- Used to determine reachability and the round-trip delay through the associated network path

Monitor

- Component used to monitor the state of parts of the system
- It can detect failure or congestion in the network or other shared resources
 - E.g., from a denial-of-service attack

Heartbeat

- Periodic message exchange between a system monitor and a process being monitored

Timestamp

- Detect incorrect sequences of events, primarily in distributed message-passing systems

Sanity Checking

- Checks the validity or reasonableness of a component's operations or outputs
- Usually based on a knowledge of the internal design, the state of the system, or the nature of the information

Detect Faults

Condition Monitoring

- Checking conditions in a process or device, or by validating assumptions made during the design

Voting

- Check that replicated components are producing the same results
- Comes in various flavours:
 - Replication
 - Functional redundancy
 - Analytic redundancy

Exception Detection

- Detection of a system condition that alters the normal flow of execution
- E.g. system exception, parameter fence, parameter typing, timeout

Self-test

- Procedure for a component to test itself for correct operation

Recover from Faults - Preparation & Repair

Hot Spare	Warm Spare	Cold Spare	Exception Handling	Rollback
<ul style="list-style-type: none">• All nodes in a group receive & process identical inputs in parallel• Spare maintain synchronous state with active node• A.k.a. <i>Active Redundancy</i>	<ul style="list-style-type: none">• Only the active members of a group process input traffic• One of their duties is to provide the redundant spares with periodic state updates• A.k.a. <i>Passive Red.</i>	<ul style="list-style-type: none">• Redundant spares remain out of service until a fail-over occurs• Power-on-reset procedure is initiated prior to it being placed in service• A.k.a. <i>Spare</i>	<ul style="list-style-type: none">• Dealing with exceptions by reporting it or handling it, potentially masking the fault by correcting the cause of the exception and retrying	<ul style="list-style-type: none">• Revert to a previous known good state, referred to as the “rollback line”

Recover from Faults - Preparation & Repair

Software Upgrade

- In-service upgrades to executable code images in a non-service-affecting manner

Retry

- Where a failure is transient retrying the operation may lead to success

Ignore Faulty Behaviour

- Ignoring messages sent from a source when it is determined that those messages are spurious

Degradation

- Maintains the most critical system functions in the presence of component failures, dropping less critical functions

Reconfiguration

- Reassigning responsibilities to the resources left functioning, while maintaining as much functionality as possible.

Recover from Faults - Reintroduction

Shadow

- Operating a previously failed or in-service upgraded component in a “shadow mode” for a predefined time prior to reverting the component back to an active role

State Resynchronization

- Partner to active redundancy and passive redundancy where state information is sent from active to standby components

Escalating Restart

- Recover from faults by varying the granularity of the components restarted and minimizing the level of service affected

Non-stop Forwarding

- Functionality is split into supervisory and data
- If a supervisor fails, a router continues forwarding packets along known routes while protocol information is recovered and validated

Prevent Faults

Removal From Service

- Temporarily placing a system component in an out-of-service state for the purpose of mitigating potential system failures

Transactions

- Bundling state updates so that asynchronous messages exchanged between distributed components are atomic, consistent, isolated, and durable

Predictive Model

- Monitor the state of a process to ensure that the system is operating within nominal parameters
- Take corrective action when conditions detect likely future faults

Exception Prevention

- Preventing system exceptions from occurring by masking a fault, or preventing it via smart pointers, abstract data types, wrappers

Increase Competence Set

- Designing a component to handle more cases - faults - as part of its normal operation

2. Modifiability



What is Modifiability?

Modifiability is about change and our interest in it is in the cost and risk of making changes.

- To plan for modifiability, an architect needs to consider various questions:
 - What can change?
 - What is the likelihood of the change?
 - When is the change made and who makes it?
 - *What is the cost of the change?*

Scenarios

Modifiability General Scenario

Source	End user, developer, system administrator.
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology.
Artifact	Code, data, interfaces, components, resources, configurations,...
Environment	Runtime, compile time, build time, initiation time, design time.
Response	
Response Measure	

General Scenario - Response

- One or more of the following:
 - Make modification
 - Test modification
 - Deploy modification

General Scenario - Response Measure

- Cost in terms of:
 - Number, size, complexity of affected artifacts
 - Effort
 - Calendar time
 - Money (direct outlay or opportunity cost)
 - Extent to which this modification affects other functions or quality attributes
 - New defects introduced

Sample Concrete Modifiability Scenario

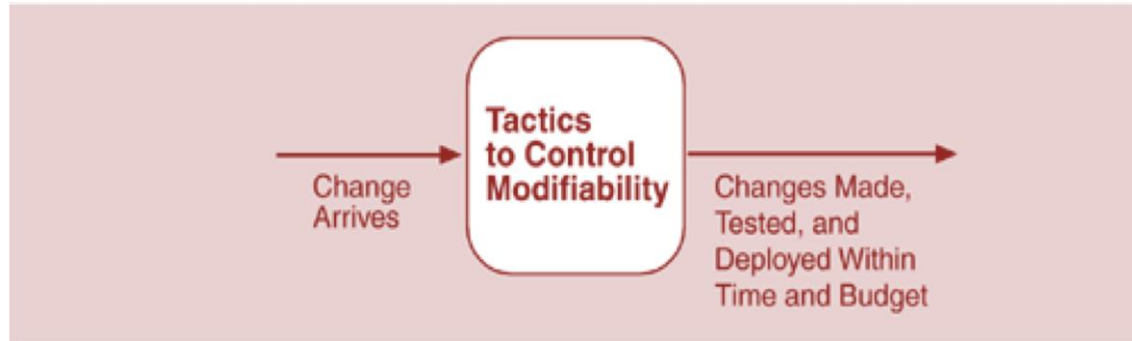
- The developer wishes to change the user interface by modifying the code at design time. The modifications are made with no side effects within three hours.

Source	Developer.
Stimulus	Change the user interface.
Artifact	Code.
Environment	Design time.
Response	Modifications are made with no side effects.
Response Measure	Modifications are made within 3 hours.

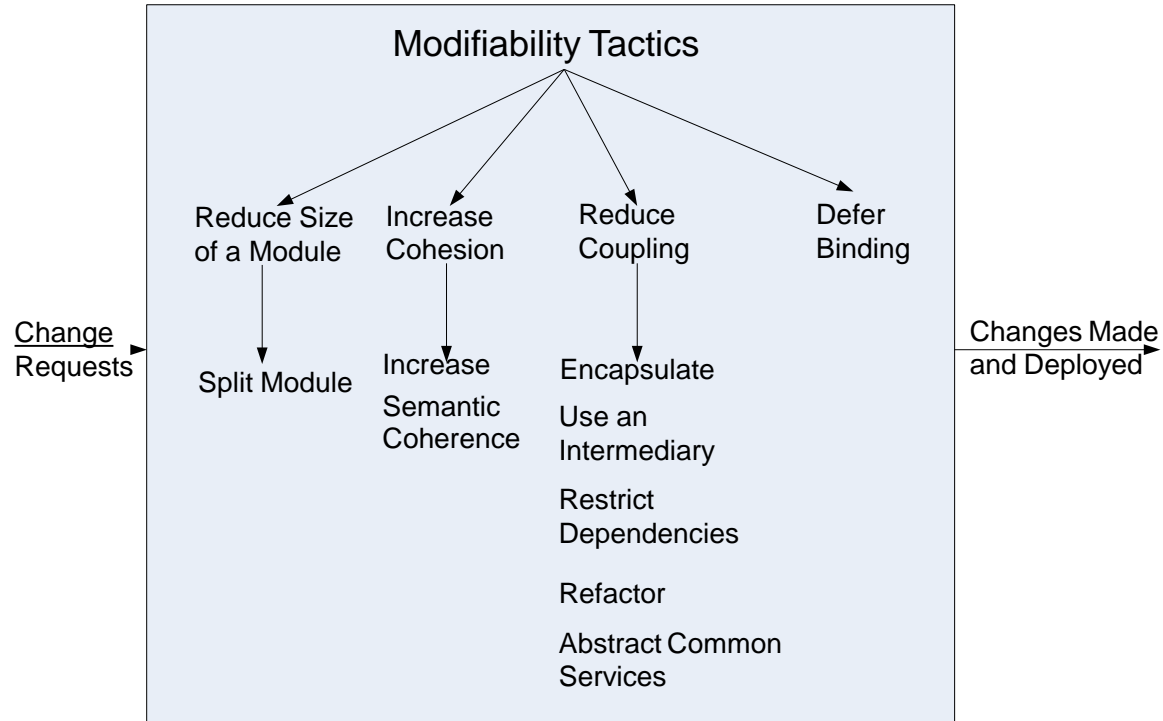
Tactics

Goal of Modifiability Tactics

Tactics to control modifiability have as their goal controlling the complexity of making changes, as well as the time and cost to make changes.



Modifiability Tactics



Reduce Size of a Module

- Split Module
 - If the module being modified includes a great deal of capability, the modification costs will likely be high
 - Refining the module into several smaller modules should reduce the average cost of future changes

Increase Cohesion

- Increase Semantic Coherence
 - If the responsibilities A and B in a module do not serve the same purpose, they should be placed in different modules
 - This may involve creating a new module or it may involve moving a responsibility to another existing module

Reduce Coupling

Encapsulate

- Introduces an explicit interface to a module
- This interface includes an API and its associated responsibilities

Use an Intermediary

- Given a dependency between responsibility A & B (for example, carrying out A first requires carrying out B), the dependency can be broken by using an intermediary

Restrict Dependencies

- Restricts the modules which a given module interacts with or depends on

Refactor

- Undertaken when two modules are affected by the same change because they are (at least partially) duplicates of each other

Abstract Common Services

- When two modules provide not-quite-the-same but similar services, it may be cost-effective to implement the services just once in a more general form

3. Usability

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
- 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

Scenarios

Usability General Scenario

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.
Artifact	System or the specific portion of the system with which the user is interacting.
Environment	Runtime or configuration time.
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.

Sample Concrete Usability Scenario

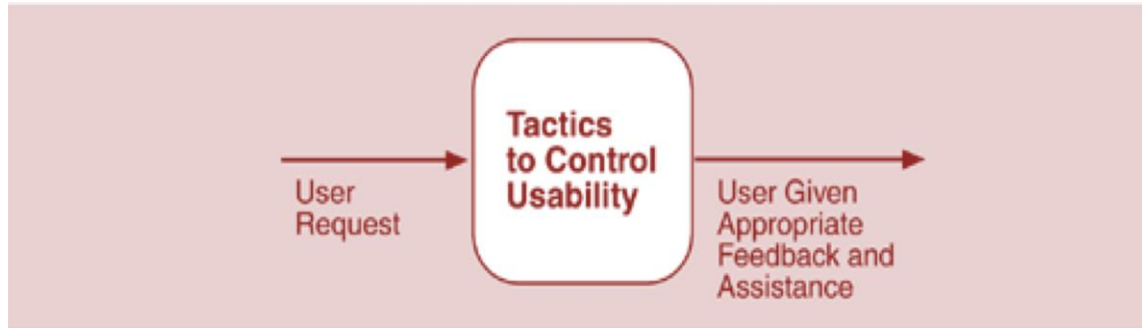
- The user downloads a new application and is using it productively after two minutes of experimentation.

Source	User.
Stimulus	Downloads a new application.
Artifact	System.
Environment	Runtime.
Response	User uses application productively.
Response Measure	User uses application within two minutes of experimentation.

Tactics

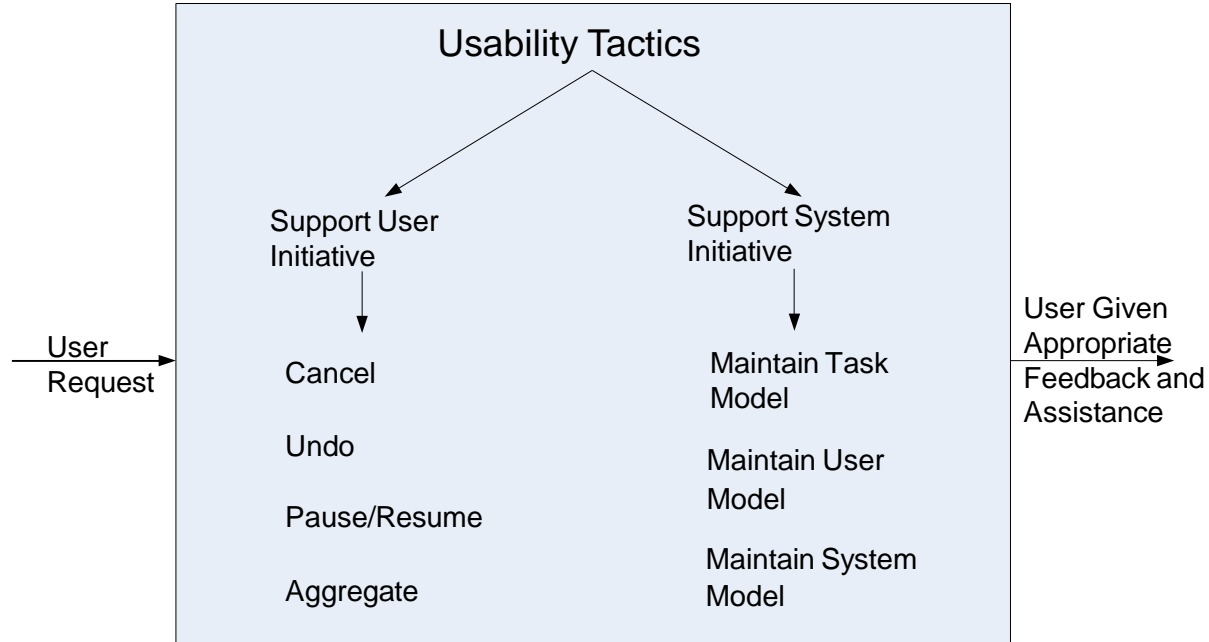
Goal of Usability Tactics

- 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



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

Usability Tactics



Support User Initiative

Cancel

- The system must listen for the cancel request
- The command being canceled must be terminated
- Resources used must be freed
- 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 monotonous work.

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

References

- Len, Bass, Clements Paul, and Kazman Rick. (2013) "Software architecture in practice." Boston, Massachusetts Addison. 3rd Edition.
 - CHAPTER 4 - Understanding QualityAttributes
 - CHAPTER 5 -Availability
 - CHAPTER 7 -Modifiability
 - CHAPTER 11 -Usability

