

Lecture Week 5

Semester 2 2023



UNIVERSITY OF
CANBERRA

SOFTWARE SYSTEMS ARCHITECTURE UG/G (11491/8746)

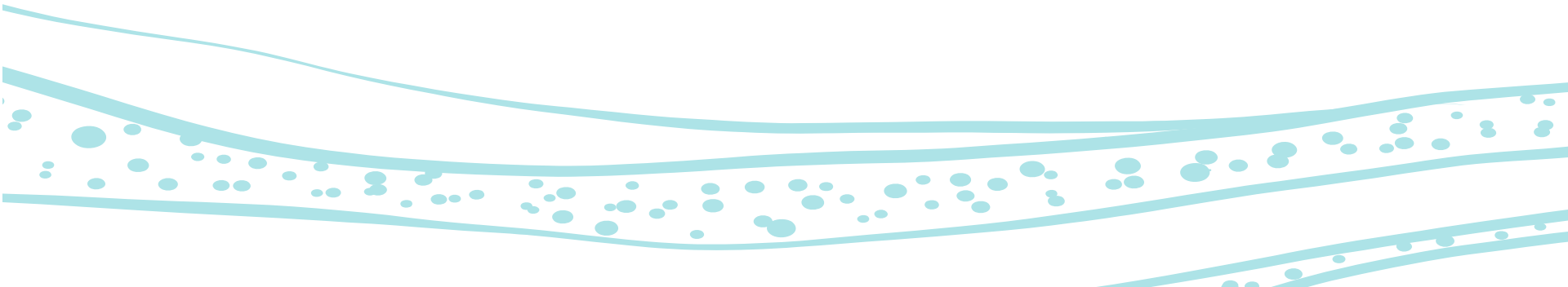
Quality Attribute Scenarios

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.

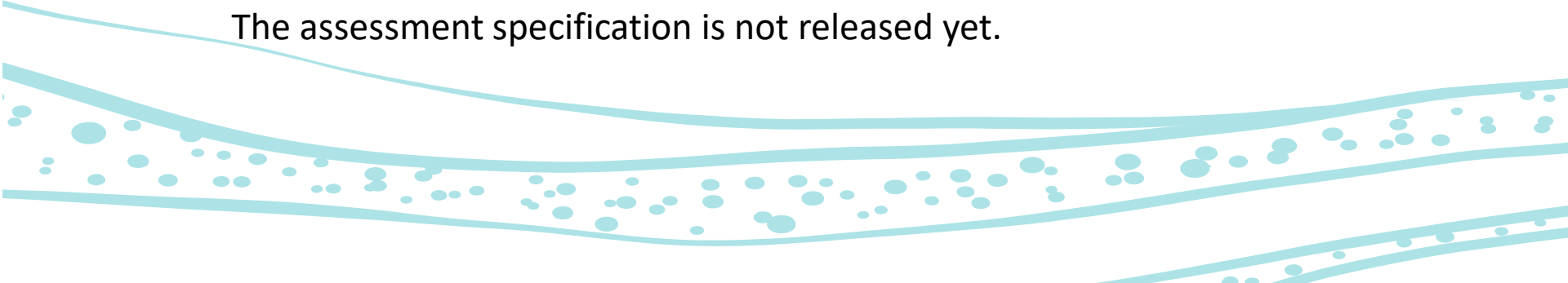


Announcements:

- The **mid-term assessment** is scheduled during Week 6 (next week).
- **You must attend the lab that you are enrolled in during Week 6.**
- Due to limited computer systems and space, no additional student can be allowed randomly. So, please make sure that you turn up to your lab.
- The mid-term assessment is a computer-based test during the lab. There will be multiple choice/small written response questions.
- It will be open book, meaning the access to Canvas material, and the recommended textbook will be allowed. No additional material is allowed.
- The test will be based on the content covering the understanding of the material from Week 1 to Week 5.
- The duration of the test is one hour.
- New lab -

Thu	10:30	11:30	Building 9 Level B Room 26 - PC Lab
-----	-------	-------	--

Announcements:

- Grades for Quiz 4 and Tutorial 2 have been released. Please let me/tutor know if there is any concern.
 - Weekly tutorials have graded 'learning by doing' group activities. So, please make sure that you attend the tutorial you are enrolled in for engagement and completion of activities.
 - **11491 Final assessment specifications** have been released and updated. The rubric and the assignment specifications are being reviewed by a reader.
 - Final assignment specification for **8746** will be as per the unit outline. The assessment specification is not released yet.
- 

WEEK 5: AGENDA

1. Interoperability

- Scenarios
- Tactics

2. Performance

- Scenarios
- Tactics

3. Testability

- Scenarios
- Tactics

1. INTEROPERABILITY

WHAT IS INTEROPERABILITY?

Interoperability is about the degree to which two or more systems can usefully exchange meaningful information.

SCENARIOS

Interoperability General Scenario

Source	A system.
Stimulus	A request to exchange information among system(s).
Artifact	The systems that wish to interoperate.
Environment	System(s) wishing to interoperate are discovered at runtime or known prior to runtime.
Response	
Response Measure	

General Scenario -Response

- The request is (appropriately) rejected and appropriate entities (people or systems) are notified
- The request is (appropriately) accepted and information is exchanged successfully
- The request is logged by one or more of the involved systems

GENERAL SCENARIO - RESPONSE MEASURE

One or more of the following:

- Percentage of information exchanges correctly processed
- Percentage of information exchanges correctly rejected

- At runtime, our vehicle information system sends our current location to the traffic monitoring system. The traffic monitoring system combines our location with other information, overlays this information on Google Maps, and broadcasts it. Our location information is correctly included with a probability of 99.9%.

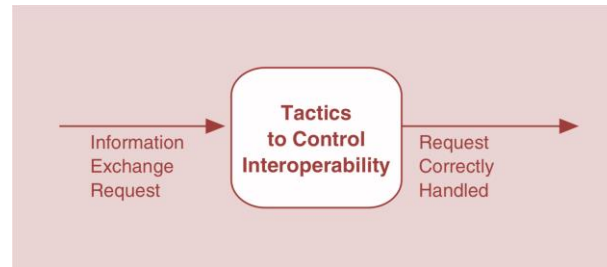
SAMPLE CONCRETE INTEROPERABILITY SCENARIO

Source	System: our vehicle information system.
Stimulus	Exchange information: sends our current location to the traffic monitoring system.
Artifact	Vehicle information system, traffic monitoring system and Google Maps.
Environment	System(s) wishing to interoperate are discovered at runtime.
Response	Traffic monitoring system combines our location with other information, overlays this information on Google Maps, and broadcasts it.
Response Measure	Our location information is correctly included with a probability of 99.9%.

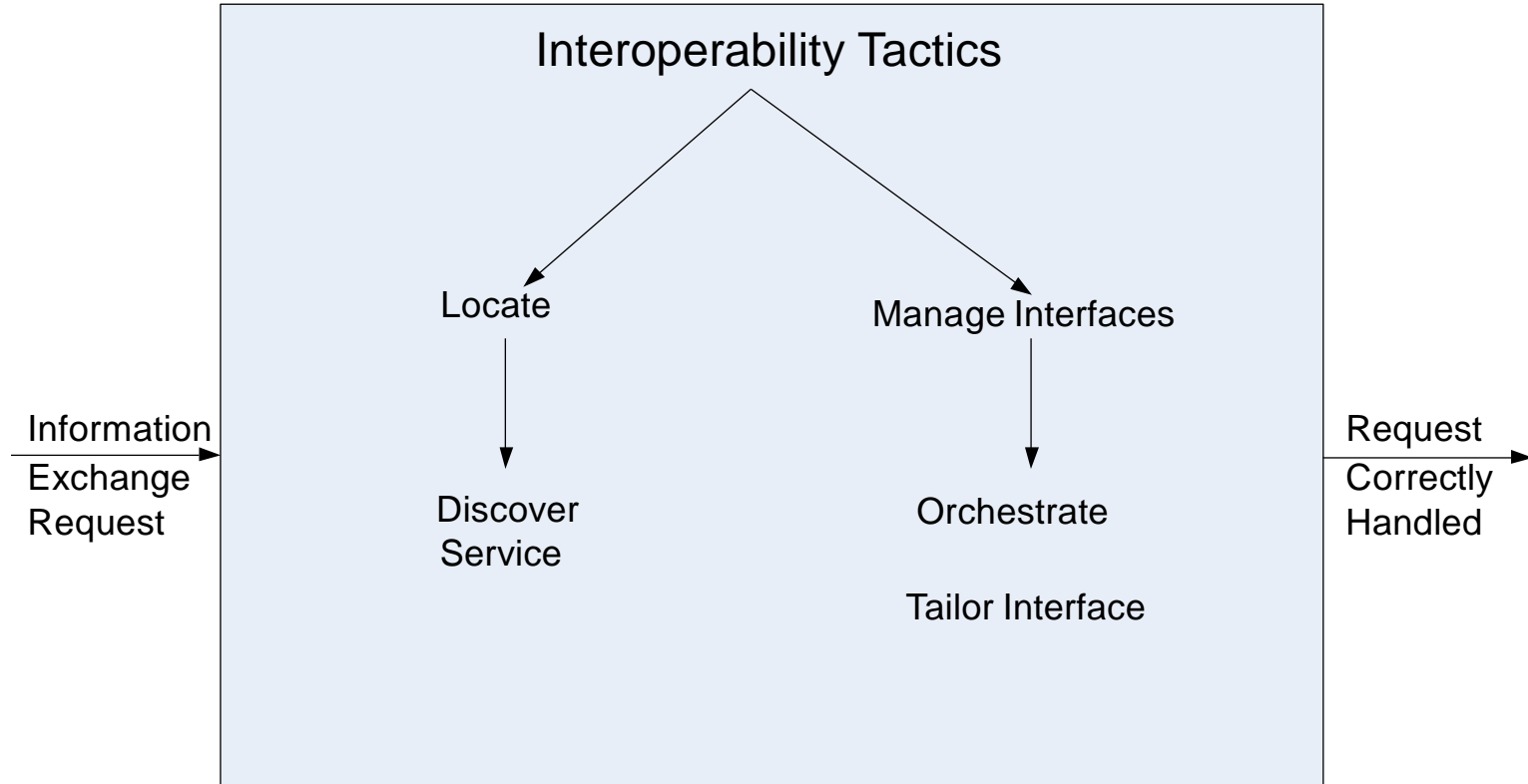
TACTICS

Goal of Interoperability Tactics

- For two or more systems to usefully exchange information they must know about each other; that is the purpose behind the locate tactics.
- Exchange information in a semantically meaningful fashion; that is the purpose behind the manage interfaces tactics.
- Two aspects of the exchange are: Provide services in the correct sequence, and Modify information produced by one actor to a form acceptable to the second actor



Interoperability Tactics



Locate

Discover service:

Locate a service through searching a known directory service

There may be multiple levels of indirection in this location process.

Manage Interfaces

Orchestrate

- Uses a control mechanism to coordinate, manage and sequence the invocation of services
- Orchestration is used when systems must interact in a complex fashion to accomplish a complex task

Tailor Interface

- Add or remove capabilities to an interface such as translation, buffering, or data-smoothing

2. Performance

What is Performance?

Performance is about time and the software system's ability to meet timing requirements.

When events occur - interrupts, messages, requests from user or other systems, or clock events marking the passage of time - the system, or some element of the system, must respond to them in time.

Characterizing the events that can occur (and when they can occur) and the system or element's time-based response to those events is essential for Performance.

SCENARIOS

Performance General Scenario

Source	Internal or external to the system.
Stimulus	Arrival of a periodic, sporadic, or stochastic event.
Artifact	System or one or more components in the system.
Environment	Operational mode: normal, emergency, peak load, overload.
Response	Process events, change level of service.
Response Measure	Latency, deadline, throughput, jitter, miss rate.

Sample Concrete Performance Scenario

Source	Users.
Stimulus	Initiate transactions.
Artifact	System.
Environment	Normal operations.
Response	Transactions are processed.
Response Measure	Average latency of two seconds.

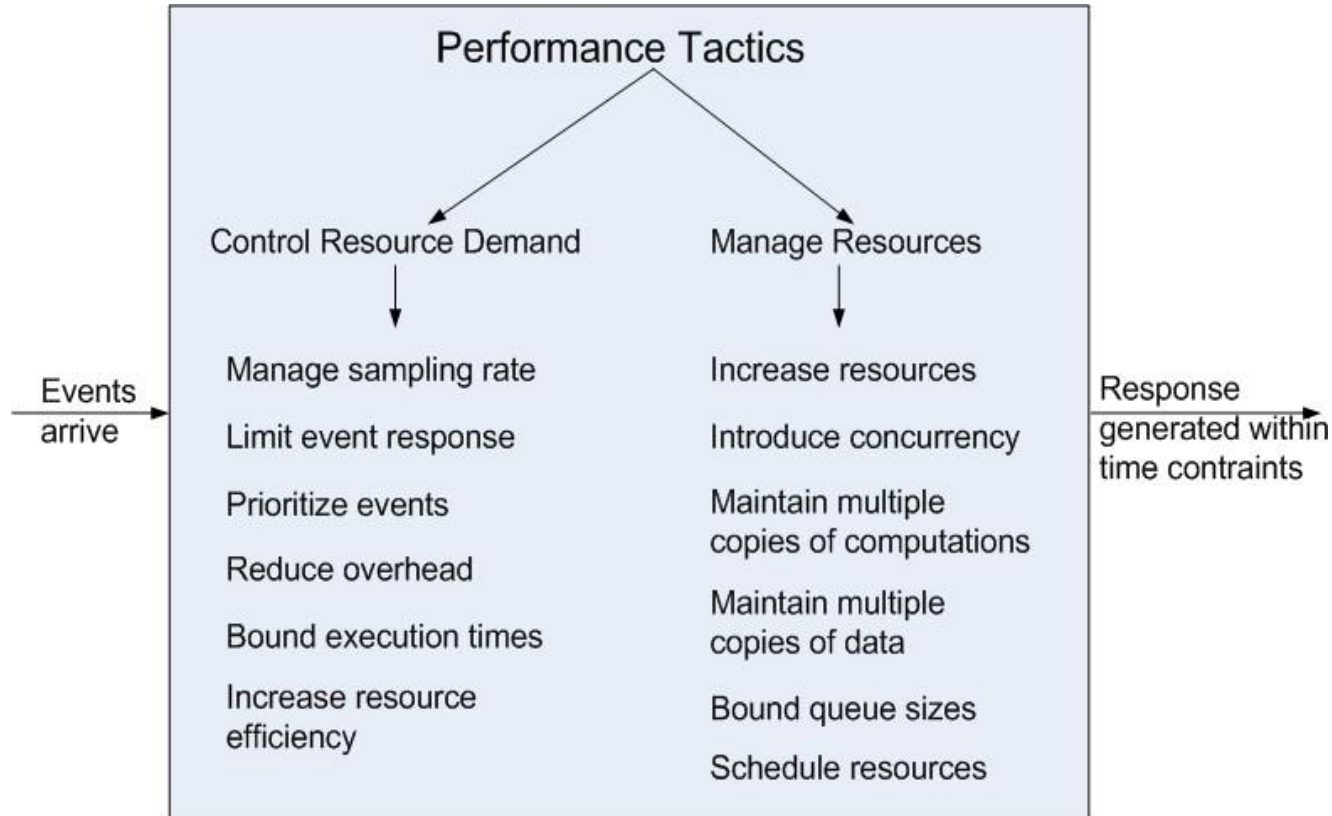
TACTICS

Goal of Performance Tactics

Tactics to control Performance have as their goal to generate a response to an event arriving at the system within some time- based constraint.



Performance Tactics



Control Resource Demand

Manage Sampling Rate

- If it is possible to reduce the sampling frequency at which a stream of data is captured, then demand can be reduced, typically with some loss of fidelity

Limit Event Response

- Process events only up to a set maximum rate, thereby ensuring more predictable processing when the events are actually processed

Prioritize Events

- If all events are not equally important, you can impose a priority scheme that ranks events according to how important it is to service them

Control Resource Demand

Reduce Overhead

- The use of intermediaries (important for modifiability) increases the resources consumed in processing an event stream; removing them improves latency

Bound Execution Times

- Place a limit on how much execution time is used to respond to an event

Increase Resource Efficiency

- Improving the algorithms used in critical areas will decrease latency

Manage Resources

Increase Resources

- Faster processors, additional processors, additional memory, and faster networks all have the potential for reducing latency

Increase Concurrency

- If requests can be processed in parallel, the blocked time can be reduced
- Concurrency can be introduced by processing different streams of events on different threads or by creating additional threads to process different sets of activities

Maintain Multiple Copies of Computations

- The purpose of replicas is to reduce the contention that would occur if all computations took place on a single server

Manage Resources

Maintain Multiple Copies of Data

- Keeping copies of data (possibly one a subset of the other) on storage with different access speeds

Bound Queue Sizes

- Control the maximum number of queued arrivals and consequently the resources used to process the arrivals

Schedule Resources

- When there is contention for a resource, the resource must be scheduled

3. Testability

What is Testability?

Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.

Specifically, testability refers to the probability, assuming that the software has at least one fault, that it will fail on its next test execution.

If a fault is present in a system, then we want it to fail during testing as quickly as possible

For a system to be properly testable, it must be possible to control each component's inputs (and possibly manipulate its internal state) and then to observe its outputs (and possibly its internal state)

SCENARIOS

Testability General Scenario

Source	Unit testers, integration testers, system testers, acceptance testers, end users, either running tests manually or using automated testing tools.
Stimulus	A set of tests are executed due to the completion of a coding increment such as a class, layer or service; the completed integration of a subsystem; the complete implementation of the system; or the delivery of the system to the customer.
Artifact	The portion of the system being tested.
Environment	Design time, development time, compile time, integration time, deployment time, runtime.
Response	One or more of the following: execute test suite and capture results; capture activity that resulted in the fault; control and monitor the state of the system.
Response Measure	One or more of the following: effort to find a fault or class of faults, effort to achieve a given percentage of state space coverage; probability of fault being revealed by the next test; time to perform tests; effort to detect faults; length of longest dependency chain in test; length of time to prepare test environment; reduction in risk exposure (size(loss) * probability(loss)).

Sample Concrete Testability Scenario

- The unit tester completes a code unit test in a component of the system during development and performs a test sequence whose results are captured and that gives 85% path coverage within 3 hours of testing.

Source	Unit tester.
Stimulus	Perform unit test on the code unit completed.
Artifact	Code unit in a component of the system.
Environment	Development time.
Response	Results are captured (capture activity that resulted in the fault).
Response Measure	85% path coverage in 3 hours of testing.

TACTICS

Goal of Testability Tactics

The goal of tactics for testability is to allow for easier testing when an increment of software development has completed.

Anything the architect can do to reduce the high cost of testing will yield a significant benefit.

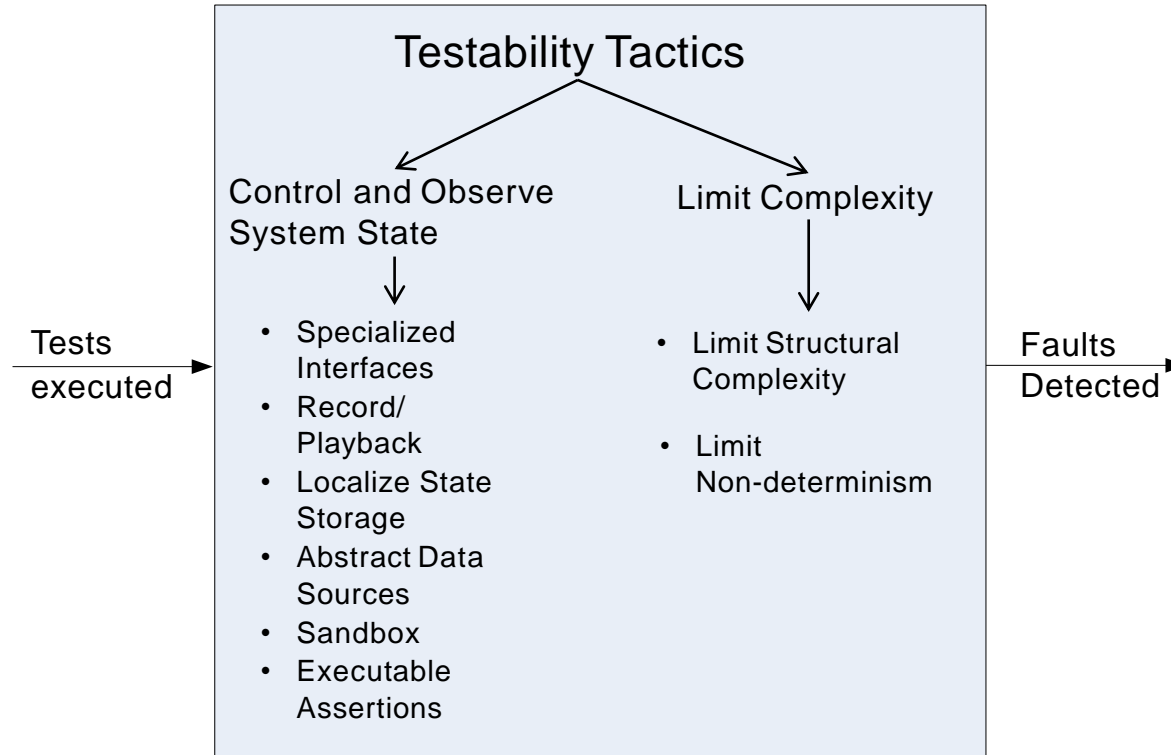
There are two categories of tactics for testability:

The first category deals with adding controllability and observability to the system.

The second deals with limiting complexity in the system's design.



Testability Tactics



Control and Observe SystemState

Specialized Interfaces

- To control or capture variable values for a component either through a test harness or through normal execution

Record/Playback

- Capturing information crossing an interface and using it as input for further testing

Localize State Storage

- To start a system, subsystem, or module in an arbitrary state for a test, it is most convenient if that state is stored in a single place

Control and Observe SystemState

Abstract Data Sources

- Abstracting the interfaces lets you substitute test data more easily

Sandbox

- Isolate the system from the “real world” to enable experimentation that is unconstrained by the worry about having to undo the consequences of the experiment

Executable Assertions

- Assertions are (usually) hand coded and placed at desired locations to indicate when and where a program is in a faulty state

Limit Complexity

Limit Structural Complexity

- Avoiding or resolving cyclic dependencies between components, isolating and encapsulating dependencies on the external environment, and reducing dependencies between components in general

Limit Non-determinism

- Finding all the sources of non-determinism, such as unconstrained parallelism, and weeding them out as far as possible

References:

Len, Bass, Clements Paul, and Kazman Rick. (2013) "Software architecture in practice."
Boston, Massachusetts Addison. 3rd Edition.

Chapter 4 - Understanding Quality Attributes

Chapter 6 - Interoperability

Chapter 8 - Performance

Chapter 10 - Testability

