Carnegie Mellon University
Software Engineering Institute

Software Architecture Principles and Practices Student Workbook

Rick Kazman

2020

SEI Training

http://www.sei.cmu.edu

Carnegie Mellon University
Software Engineering Institute

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-0138

Table of Contents

Exercise 1: Developing Concrete Scenarios

A Quality Attribute Scenario is a short description of how a system is required to respond to some stimulus. In this exercise, you are asked to characterize a quality attribute requirement by crafting a concrete scenario.

STEP 1. The System

Select a system with which you have familiarity. Perhaps a system you have developed, are currently working on, or have an interest. Then choose a quality attribute to explore. The table below provides a quick guide to information about Quality Attributes in the textbook.

Quality Attribute	Location in Text	Table in Text
Availability	p. 86	Table 5.3
Interoperability	p. 108	Table 6.2
Modifiability	p. 120	Table 7.1
Performance	p. 134	Table 8.1
Security	p. 150	Table 9.1
Testability	p. 163	Table 10.1
Usability	p. 176	Table 11.1

Table 1. Quality attribute references in the textbook, *Software Architecture in Practice, 3rd Edition*

STEP 2. The Scenario

Develop a concrete scenario for the chosen system. A Quality Attribute Scenario consists of six parts. Use the following table to craft your scenario, identifying its six parts.

Scenario Part	Definition	Your Scenario
Stimulus Source	*an entity that generates a stimulus*	
Stimulus	*a condition that affects the system*	
Artifact(s)	*the part of the system that was stimulated by the stimulus*	
Environment	*the condition under which the stimulus occurred*	
Response	*the activity that results because of the stimulus*	
Response Measure	*the measure by which the system's response will be evaluated*	

Compose a sentence that incorporates all parts of the scenario.

_____

_____

Exercise 2: Patterns and Quality Attributes

Think about the following quality attributes and how they are likely to be affected by the use of the Layers pattern in designing an architecture.

In this exercise, indicate your responses in the following table.

Quality Attribute	+/-/0	Quality Attribute	+/-/0
Buildability		Usability	
Modifiability		Subsetability	
Reusability		Dependability	
Portability		Interoperability	
Reliability		Availability	
Security		Safety	
Testability		Performance	
Other? _____		Other? _____	

For each quality attribute indicate whether you think layering positively affects it (+), negatively affects it (-), or whether the effect is neutral or mixed (0). Provide your reasoning for each decision.

Compare your responses to those of the instructor in the Appendix.

Exercise 3: Designing with Patterns

In this exercise, your task is to propose a design for the Messaging Infrastructure layer of the new Brokerage Information System (BIS) for the BizCo company. To accomplish this, you must select a pattern and instantiate it, modifying the architecture to meet the quality attribute requirements for the company.

STEP 1. Understand the problem.

The Organization: BizCo

BizCo is a fictious, mid-sized company which provides brokering appraisal services for the commercial and residential real estate industry in the United States. BizCo is headquartered in Pittsburgh, PA, and has branch offices throughout the United States.

BizCo's overarching organizational goal is to become the leader in the industry. They have identified a number of supporting business goals.

BizCo Business Goals

To establish BizCo as the industry leader in brokering appraisal for commercial and residential properties. We must:

- Increase market share of brokerage services
- Dramatically decrease response time to changing market conditions
- Have direct, secure access to brokerage information 24/7
- Have reporting that aggregates, collates, and communicates timely information clearly and as needed

BizCo Business Processes

A real estate appraisal is needed to secure a mortgage loan when purchasing a property. A real estate appraisal is performed by licensed appraisers who work as independent contractors. As a real estate appraisal broker, BizCo connects the lender with an independent appraiser.

The typical workflow at a BizCo branch office begins with a Service Agreement between BizCo and lenders, and between Bizco and appraisers. The Service Agreement Management business process establishes and records the service agreements.

The Appraisal Request Management business process records appraisal requests and routes residential and commercial requests differently pairing commercial and residential appraisals with commercial and residential appraisers and routes the requests accordingly.

Once the property has been appraised, the Appraisal Fulfilment process records the appraisal results, and notifies the Billing process to bill the lender and pay the appraiser according to their service agreement.

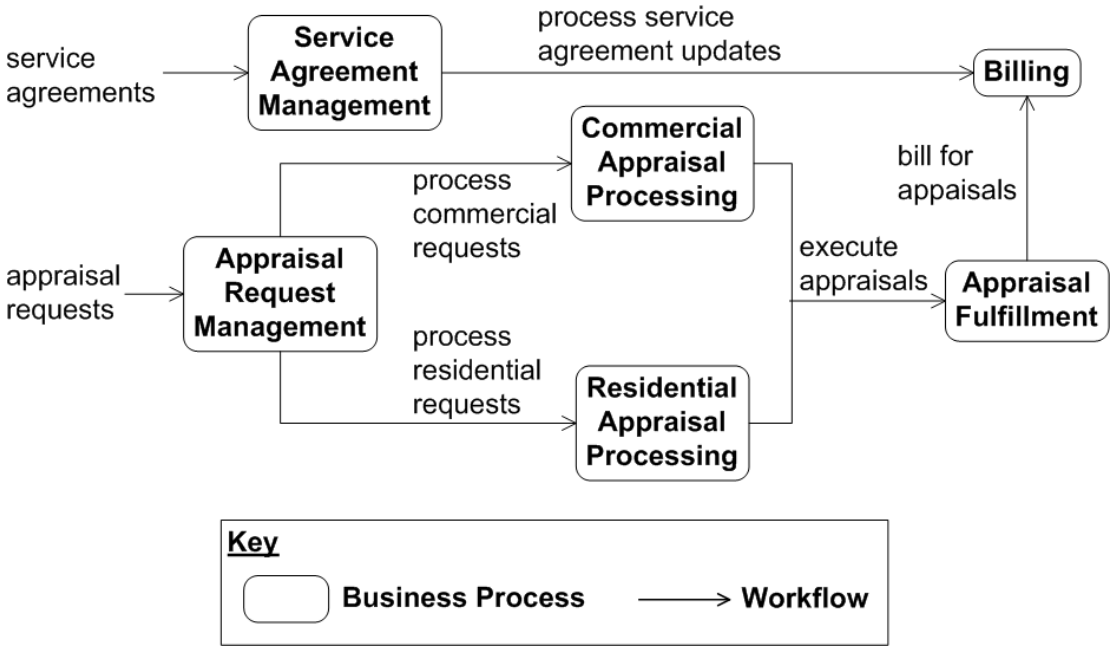

Figure 1 BizCo Business Process Model

BizCo Information Systems

Three standalone systems operate at each BizCo branch. Please note that the systems are not integrated within the branch and do not integrate externally.

Table 2 - BizCo Existing Information Systems per Branch

BizCo Branch Systems	Description
Commercial Property System (CPS)	Services requests for data about appraisal requests submitted, assigned, and fulfilled for commercial properties including request details, lender/appraiser assignments, and status.
Residential Property System (RPS)	Services requests for data about appraisal requests submitted, assigned, and fulfilled for residential properties including request details, lender/appraiser assignments, and status.
Brokerage Billing System (BBS)	Services requests for data about branch office, lender, and appraiser business addresses and contacts, contract terms and conditions, and accounts receivable.

Table 3. X indicates system support for a business process

Business Process	CPS	RPS	BBS
Appraisal Request Management	**X**	**X**	
Commercial Appraisal Processing	**X**		
Residential Appraisal Processing		**X**	
Appraisal Fulfillment	**X**	**X**	
Billing			**X**
Service Agreement Management			**X**

BizCo Information Reporting

CPS, RPS, and BBS at each branch generate reports independently

- Summarize brokerage activities
- Biweekly
- Hard-copy
- Sent to main office in Pittsburgh via courier

BizCo Brokerage Information System (BIS) – the Proposed System

BizCo needs to gather more extensive information from the branch offices' existing systems in a timely manner. To this end, your organization plans to develop the new **Brokerage Information System (BIS).**

The BIS must allow the main office to display information on:

- Branch office, lender, and appraiser business address and contacts
- Contract terms and conditions for lenders and appraisers
- Appraisal requests submitted, assigned, and fulfilled
- Brokerage activities across and for individual branch offices
- Brokerage activities across and for individual lenders, appraisers, and brokers
- Account receivables aggregated among all offices

BizCo BIS Software Architecture

The BizCo systems architect has designed a Layered Architecture, consisting of four layers, as a module structure for the new system, encapsulating the three existing systems, CPS, RPS, and BBS.

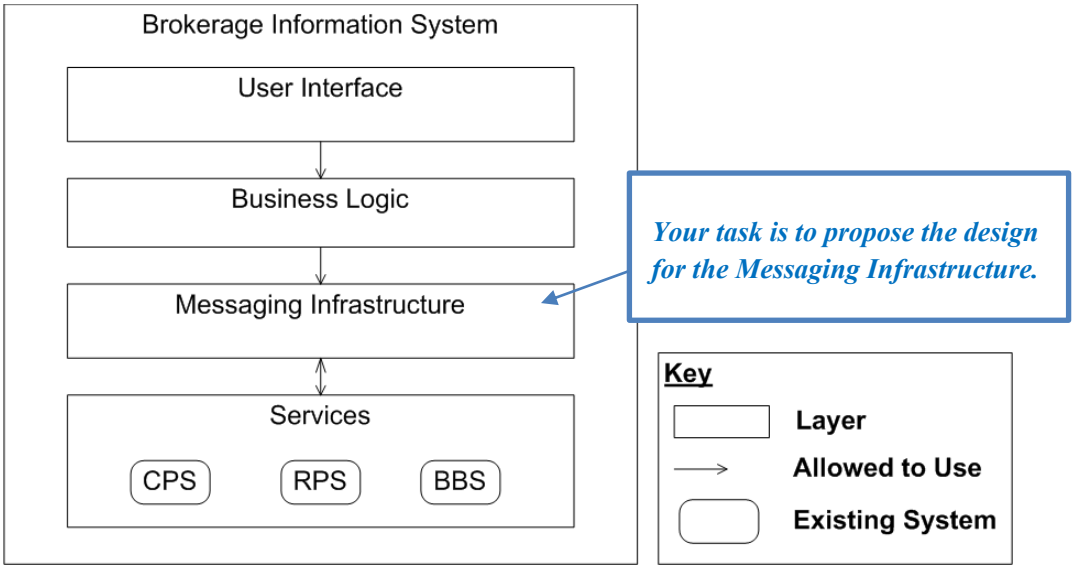

Figure 2 BIS Architecture

Table 4 BIS Architecture Layers

Layer	Description
User Interface	Allows users to select, customize, and display brokerage data and activity reports.
Business Logic	Interacts with remote services to collect data and collates the data into brokerage reports.
Message Infrastructure ★ *Your design task.*	Provides access to remote services.
Services	A collection of services that provide brokerage data. CPS, RPS, and BBS will provide these services.

STEP 2: Identify the quality attribute requirements.

Using the following table, write down what you think are the most important quality attributes for the Messaging Infrastructure layer and prioritize their relative importance on a scale from 1 to n, where 1 is most important quality attribute requirement for the messaging infrastructure and n is the least important.

Hint: Review the business goals!

Relative Importance	Quality Attribute

Compare your prioritized list with those in the Appendix.

STEP 3: Design the Messaging Infrastructure Layer

Now you can begin designing the Messaging Infrastructure layer using one of three patterns: Messaging, Publisher-Subscriber, or SOA. In this step you must first consider the patterns, weighing the pros and cons of each, and the tradeoffs from a *quality attribute perspective*.

Read about and consider the Messaging, Publisher-Subscriber, and SOA patterns.

The Messaging Pattern

The Messaging Pattern	
Context	Some distributed systems are composed of services that were developed independently. To form a coherent system, however, these services must interact reliably, but without incurring overly tight dependencies on one another.
Problem	Integrating independently developed services, each having its own business logic and value, into a coherent application requires reliable collaboration between services. However, since services are developed independently, they are generally unaware of each other's specific functional interfaces. Furthermore, each service may participate in multiple integration contexts, so using them in a specific context should not preclude their use in other contexts.
Solution	Connect the services via a message bus that allows them to transfer data messages asynchronously. Encode the messages (request data and data types) so that senders and receivers can communicate reliably without having to know all the data type information statically.

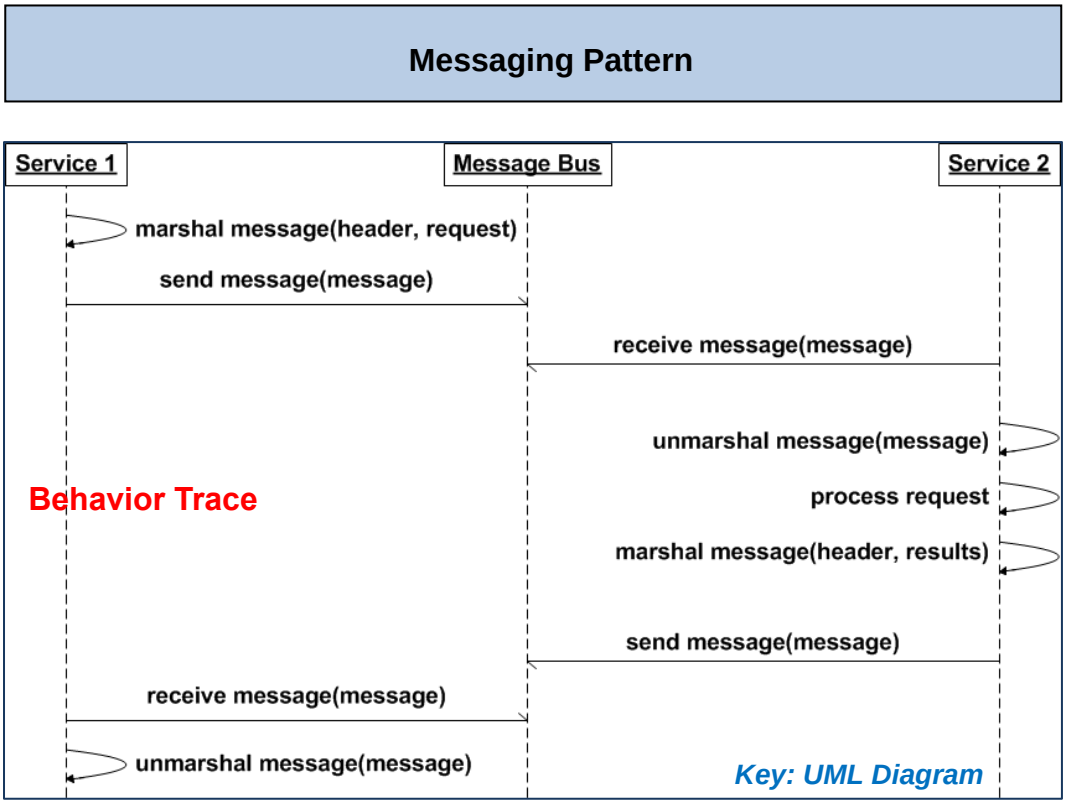

Figure 3 Messaging Pattern, Sequence Diagram of Services Interactions

Messaging Pattern Benefits	Messaging Pattern Liabilities
Services can interact without having to deal with networking and service location concerns.	Lack of statically typed interfaces makes it hard to validate system behavior prior to runtime.
Asynchronous messaging allows services to handle multiple requests simultaneously without blocking.	Service requests are encapsulated within self-describing messages that require extra time and space for message processing.
Allows services to participate in multiple application integration and usage contexts.	

The Publisher-Subscriber Pattern

Publisher-Subscriber Pattern	
Context	Components in some distributed applications are loosely coupled and operate largely independently. If such applications need to propagate information to some or all of their components, a notification mechanism is needed to inform the components about state changes or events that affect or coordinate their own computation.
Problem	The notification mechanism should not couple application components too tightly, or they will lose their independence. Components want to know only that another component is in a specific state, not which specific component is involved. Components that disseminate events often do not care which other components want to receive the information. Components should not depend on how other components can be reached or on their specific location in the system.
Solution	Define a change propagation infrastructure that allows publishers in a distributed application to disseminate events that may be of interest to others. Notify subscribers interested in those events whenever such information is published.

Publisher-Subscriber Pattern

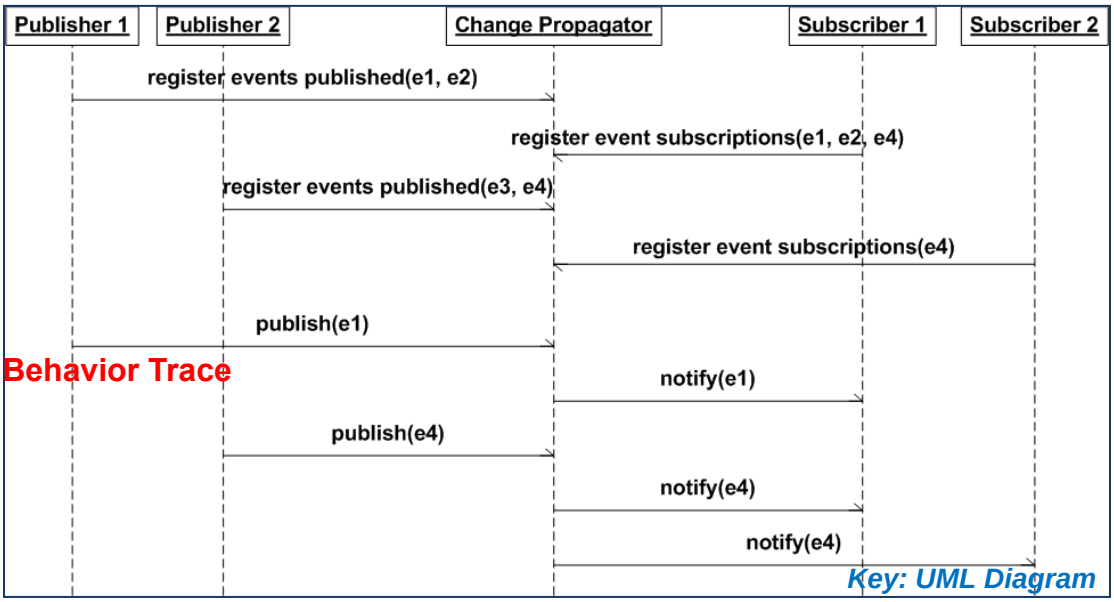

Figure 4 Publisher-Subscriber Pattern, sequence diagram of services interactions

Publisher-Subscriber Pattern Benefits	Publisher-Subscriber Pattern Liabilities
Publishers can asynchronously transmit events to Subscribers without blocking.	Publishing can cause unnecessary overhead if Subscribers are interested in only a specific type of event.
Asynchronous communication decouples Publishers from Subscribers, allowing them to be active and available at different times.	Filtering events to decrease event publishing and notification overhead can result in other costs (e.g., decrease in throughput, unnecessary notifications, breakdown of anonymous communication model).
Publishers and Subscribers are unaware of each other's location and identity.	

Dynamic Routing Pattern

Dynamic Routing Pattern	
Context	It is often necessary to build complex business processes by wiring together a set of relatively simple services in a dynamic way.
Problem	Routing messages through a distributed system based on filtering rules is inefficient because messages are sent to every destination's filter and router for inspection and rules resolution, whether or not the message could be processed.
Solution	Define a message router that includes both filtering rules and knowledge about the processing destination paths so that messages are delivered only to the processing endpoints that can act on them. Unlike filters, message routers do not modify the message content and are concerned only with message destination.

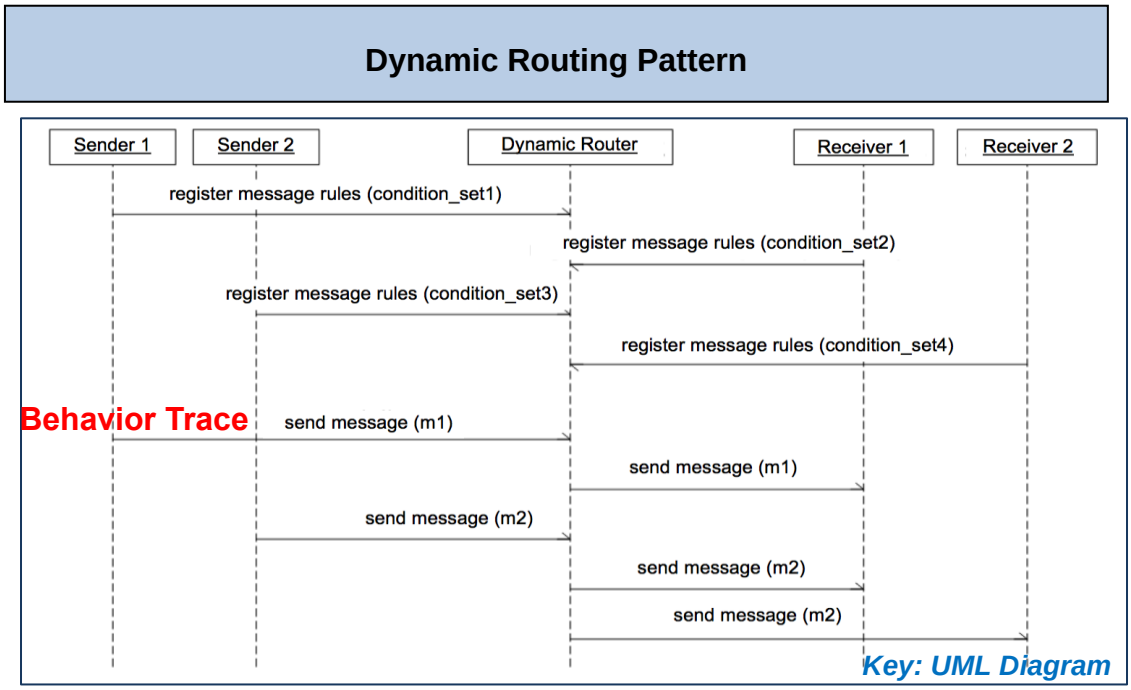

Figure 5 Dynamic Routing Pattern, Sequence diagram for services interactions

Dynamic Routing Pattern Benefits	Dynamic Routing Pattern Liabilities
Services do not need to deal with networking concerns or know each other's locations since all requests are handled through the message router.	Performance overhead.
Communications can be optimized as services dynamically become available or unavailable.	Single point of failure.
Efficient, predictive routing.	Potentially complex, unintuitive behavior when rules conflict.

STEP 4: Select and instantiate a pattern.

Select the pattern that you think will best meet the messaging infrastructure requirements. Instantiate the pattern by describing the roles of its constituent participants, their responsibilities and relationships, and the ways in which they collaborate.

Pattern	_____
Overview	
Elements	
Relations	
Constraints	
Weaknesses	
Other Assumptions	

STEP: 5 Identify the tradeoffs using this pattern.

Tradeoffs

STEP 6: Sketch the proposed design for the Messaging Infrastructure.

STEP: 7 Document your assumptions.

Assumptions

Compare your results to a recommended solution, created by the course author, in the Appendix of this document.

Exercise 4: Applying Tactics

Based on your design decision for the BIzCo BIS, what quality would you like to improve?

_____

What tactics would you choose to improve your design of the messaging infrastructure for the system?

_____

_____

_____

_____

_____

Quality Attribute	Text Reference[i]
Availability	p. 87-95
Interoperability	p. 110-112
Modifiability	p. 121-125
Performance	p. 135-141
Security	p. 150-154
Testability	p. 164-168
Usability	p. 177-181

[1] Software Architecture in Practice, 3rd Edition; Bass, L; Clements, P.; Kazman, R; Addison-Wesley Professional, 2012.

Consider the tradeoffs. What are the issues associated with the selection of those tactics?

_____

_____

_____

_____

_____

_____

_____

_____

Exercise 5: Documenting Software Architecture

Based on your architecture sketch(es) for the BizCo BIS from the previous exercise, use the principles of software architecture documentation to create multiple views of your architecture structure.

STEP 1: For each view, define

- Elements
- Relations
- Properties

Be sure to augment each view with an explanation of the documentation organization and of the system as a whole.

Remember to include a notation key for each diagram! An informal notation is acceptable provided that there is a notation key.

STEP 2: Select your most important scenario

Choose your most important scenario. Examine the document views that allow you to precisely reason about that scenario; and determine what information you would like to know if you were to create the system to satisfy that scenario.

Selected Scenario:

Exercise 6: Evaluation and Architecture

In this exercise your task is to analyze a scenario, as part of the Architecture Tradeoff Analysis Method (ATAM)© Phase 2.

STEP 1: Consider this scenario and complete the ATAM Scenario Analysis.

A BizCo server at a branch location stops responding. The failure is detected and the system is restored to normal operation within 90 seconds.

Scenario Analysis Template, Part 1

ATAM: Scenario Analysis		
Scenario		
Business Goal(s)		
Attribute		
Attribute Concern		
Scenario Refinement	Stimulus	
	Stimulus Source	
	Environ-ment	
	Artifact	
	Response	
	Response Measure	

STEP 2: Examine the architectural decisions and reasoning, and expand the ATAM Scenario Analysis.

Scenario Analysis Template, Part 2

Architectural Decisions and Reasoning	
Risks Potentially problematic architectural decisions.	1. 2. 3. n...
Sensitivities Architectural parameters that significantly affects achievement of quality attributes.	1. 2. 3. n...
Tradeoffs A tradeoff is a sensitivity point that affects multiple responses and affects them in opposite directions.	1. 2. 3. n...
Non-Risks Good architectural decisions that are frequently implicit in the architecture	1. 2. 3. n...
Other Issues	1. 2. 3. n...

Appendix: Expert Solutions and Commentary

Exercise 2: Patterns and Quality Attributes

Quality Attribute	+/-/0	Quality Attribute	+/-/0
Buildability	+	Usability	+
Modifiability	+	Subsetability	0
Reusability	+	Dependability	0
Portability	+	Interoperability	0
Reliability	0	Availability	0
Security	0	Safety	0
Testability	+	Performance	-
Other? _____		Other? _____	

Buildability is enhanced by layering, assuming that the layers are properly implemented, as you can define interfaces at a higher level of the layering that can be used right away, even if those interfaces have not been completely fleshed out.

Usability may be enhanced by layering if the user interface is encapsulated into one layer (or a small number of layers). Thus, changes to the UI, which are often frequent, are isolated from the remainder of the system.

Modifiability is the quality that is most often associated with layering. If the layers are properly designed and implemented, coupling should be low across layers and cohesion high within layers, and these properties support modifiability.

While it may seem like Subsetability would be enhanced by layering, it is often the case that subsets of a system's functionality cross layers, in which case the benefits of layering may be limited.

Reusability is clearly supported by the layers pattern. The principles of low coupling and high cohesion that dominate proper layering provide exactly the right conditions for layers (or modules within layers) to be easily reused in different contexts.

Dependability, reliability, and availability are all run-time attributes related to how long a piece of software can be expected to run fault- or failure-free, and how quickly it can be recovered in the event of a failure. Hence layering has no direct contribution to such concerns. It has, at best, an indirect relationship, in that properly layered software is easier to modify and hence adding or changing the fault-detection and recovery logic might be simpler.

Portability is clearly supported by the effective use of layering. Lower layers can hide platform, operating system, or other implementation details from other layers. Upper layers should, in this case, only depend on the abstractions exposed by the lower layers and not on any knowledge of the specifics of their implementation.

Interoperability is primarily about the amount of coupling—syntactic, semantic, temporal, etc.—between distinct systems. While layering may affect the coupling within a system, the coupling between systems is not directly addressed in this pattern.

Security, like usability, is not directly affected by layering, although it is indirectly affected. If security controls are packaged within a layer, or are abstracted by a layer, then it makes it easy to modify them without affecting other parts of the system, and it makes it easy to use an external component (for example, a framework) for security functionality.

Safety is primarily a run-time attribute, like dependability or availability. As such the use of layering has little direct consequence on the achievement of this system quality.

Testability is enhanced by limiting the state space of a program. This property is, in turn, enhanced by lowering the coupling in a system because, all other things being equal, restricted coupling reduces the size of a system's state space. Thus layering may improve testability as—properly implemented—it leads towards reduced coupling among the system's modules.

Layering adds layers of indirection—interfaces—between a system's modules and these interfaces come with a cost: they add a performance penalty. Thus it is generally accepted that there is a tradeoff between modifiability and performance imposed by the use of layering.

Exercise 3: Designing with Patterns

Quality Attributes Requirements

Expert Advice: This exercise illustrates the importance of learning what is important to your stakeholders to determine the one, true set of priorities of the qualities of the system.

Relative Importance	Quality Attribute
1	Interoperability
2	Availability
3	Security
4	Modifiability
5	Testability
6	Performance

Here is an example list of quality attributes for the BizCo system, along with their priorities. You may, however, have come up with a slightly different list. You might want to consider why this is the case. Why did we not arrive at exactly the same list?

The answer is that we all interpret what we hear through the lens of our own assumptions and experience. This is exactly why it is important to include stakeholders in any such prioritization discussions—because you may be biased or not completely informed and any architectural decisions that you make based on these erroneous assumptions may have enormous downstream consequences. To ameliorate this risk, hold a Quality Attribute Workshop or do a Utility Tree exercise.

Selected Pattern and Instantiation, Tradeoffs and Assumptions

Expert Advice:

We chose the Messaging pattern as the basis for the Messaging Infrastructure layer. We felt that, while all of the patterns could be used, Messaging was the simplest and there was no requirement pushing us to a more complex pattern such as Publisher-Subscriber or Dynamic Routing. The application requirements, as we understand them, are reasonably simple and, while they will doubtless grow as the system evolves, we have not heard any requirement for the kinds of flexibility that Publisher-Subscriber or Dynamic Routing provide. While there is some performance overhead in the use of this pattern—a tradeoff with performance—this is well within the acceptable bounds of the system, since we have not heard any requirement for high performance—the system is primarily producing reports or responding to relatively simple requests for information, and so the small performance penalty for messaging appears to be a reasonable tradeoff.

In the sketches below, we provide two views of the architecture: a module view, showing how we instantiate the Messaging Infrastructure layer, and an allocation view, showing how we intend to instantiate and deploy the system. For the module view we have provided our own notation, describe in the accompanying key. For the allocation view we have used UML as our notation.

In the module view we have added a “Message” class with methods that elements in the Business Logic and Services layers can call to send and receive messages. Note that we have added an “ack” parameter to the sendMessage class. When set to true (the default is false) the message bus will explicitly acknowledge receipt of the message.

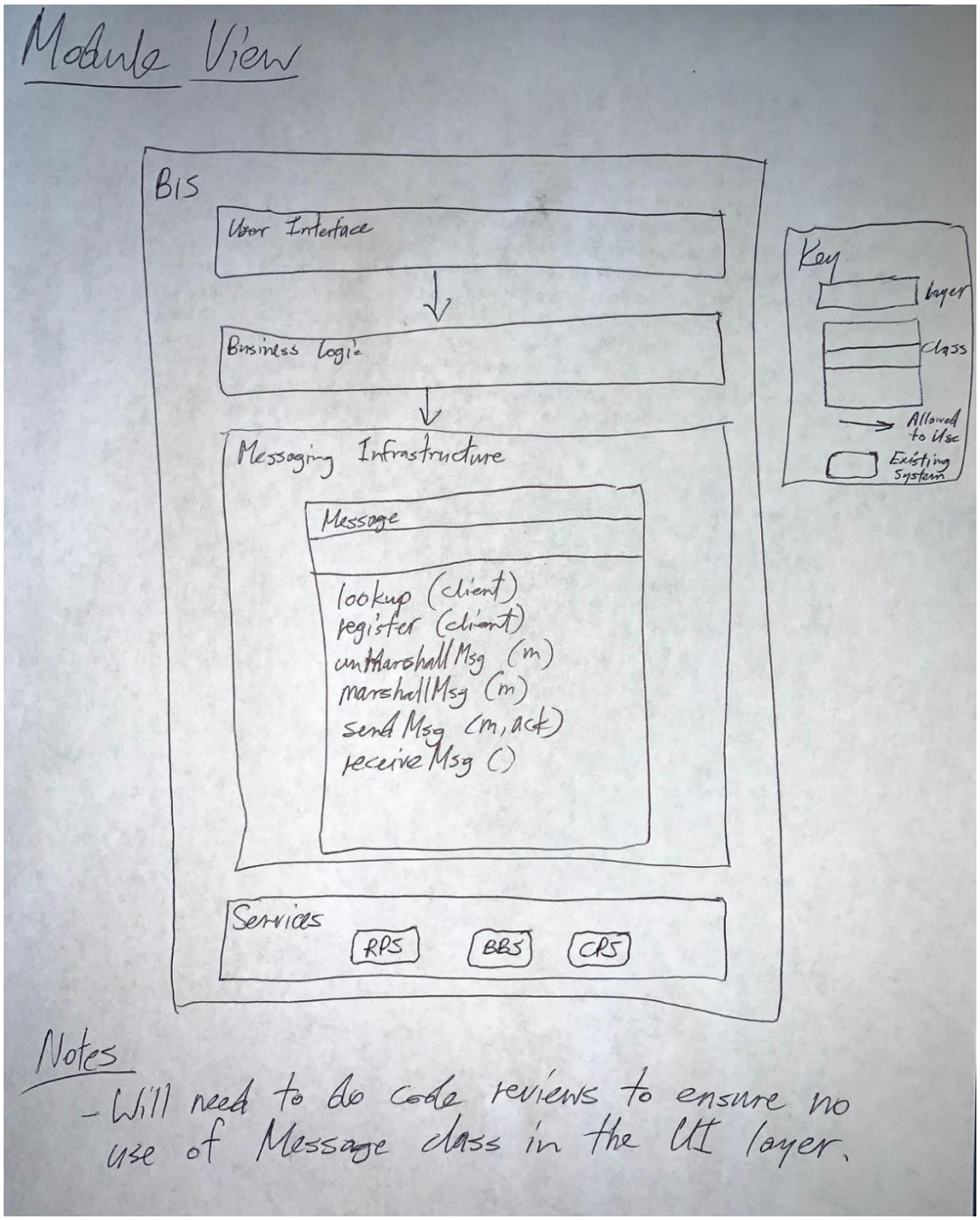

Figure 6 - Module View – how to instantiate the Messaging Infrastructure Layer

In the allocation view we have shown all of the services as peers. That is to say there is no difference in the software deployed for head office or for any branch office. The roles and responsibility of each peer will be determined by the authorizations of the user who is logged in. While there must be at least one MessageBus component, t is expected that there will be more, for fault tolerance. For this reason, a LoadBalancer component has also been added, to manage and balance the traffic on each Message-Bus, and to monitor the liveness of each. The LoadBalancer will also ensure that the register messages go to all MessageBus instances so that they all share the same state, in terms of what peers exist in the system. It is expected that the LoadBalancer will periodically ping each of the MessageBuses to determine their liveness. It is assumed that the LoadBalancer and MessageBus can be purchased as off-the-shelf components, requiring configuration but little or no coding. Finally, it is future work to determine how to detect failures of clients.

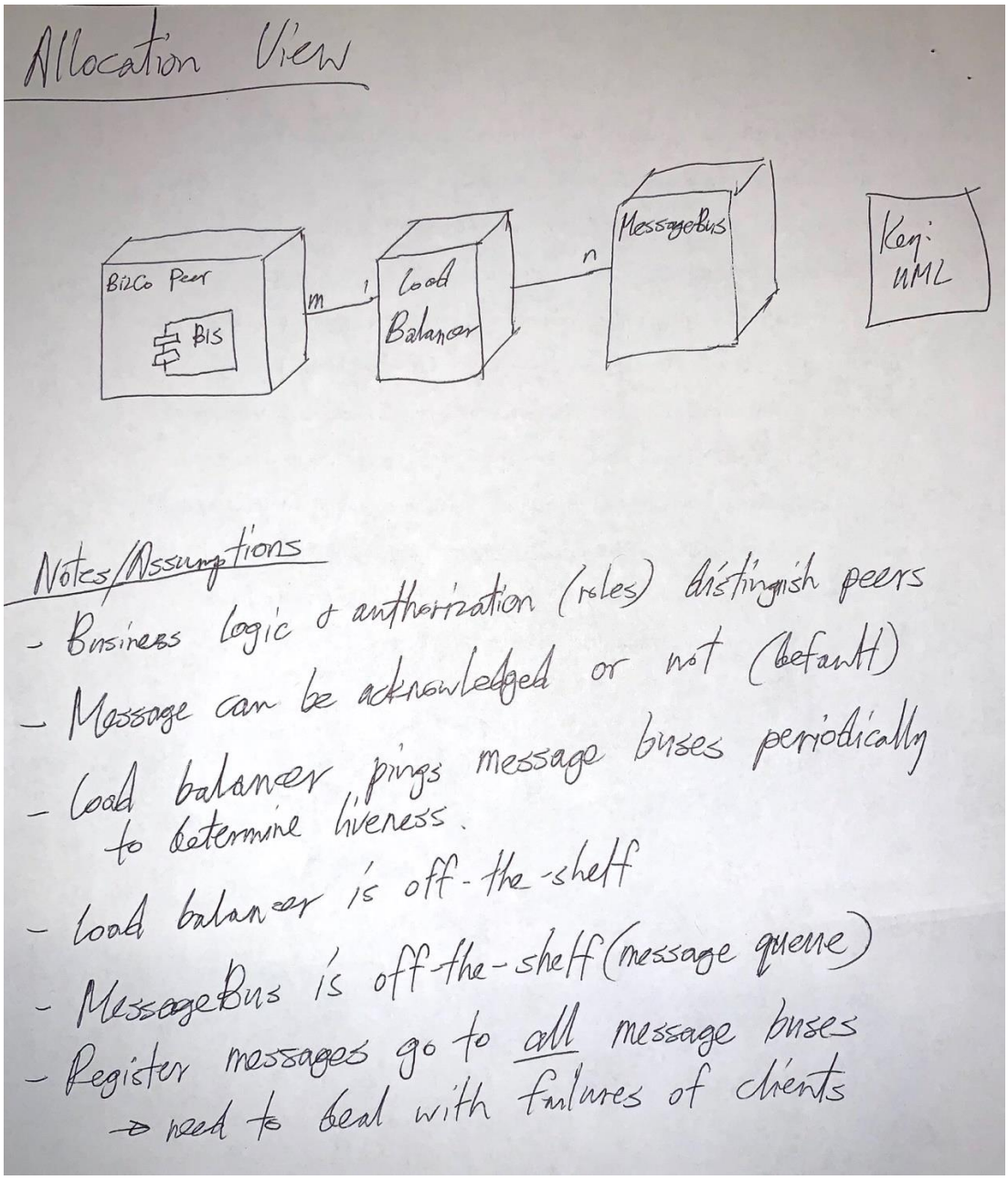

Figure 7 Allocation View-instantiate and deploy the system

Exercise 4: Applying Tactics

Currently there is no mention of security in the existing system services—RPS, BBS, and CPS. We would like to add a number of security tactics to the architecture since this system contains substantial amounts of private and business-critical information. Security was our third-most important quality attribute in the ranking that we made above.

We will employ the *Detect Message Delay* and *Verify Message Integrity* tactics to detect attacks. We will *Identify Actors, Authenticate Actors*, and *Authorize Actors* to resist attacks. In addition, by keeping and encapsulating our legacy services and not keeping all of our information in a single repository, we are employing the *Limit Exposure* tactic. Finally, we will employ the Maintain Audit Trail tactic, keeping audit trails both on individual BIS system and on the Message Bus, so that we can trace the actions of attackers, as a means of recovering from attacks.

Exercise 5: Documenting Software Architecture

See sketches above. In any architectural documentation, it is advisable to document the rationale for any design decision. For more information on this, see the book: H. Cervantes, R. Kazman*, Designing Software Architectures: A Practical Approach*, Addison-Wesley, 2016.

Exercise 6: ATAM Scenario Analysis

ATAM: Scenario Analysis		
Scenario	A BizCo server at a branch location stops responding. The failure is detected and the system is restored to normal operation within 90 seconds.	
Business Goal(s)	24/7 Operation	
Attribute	Availability	
Attribute Concern	Mean time to repair	
Scenario Refinement	Stimulus	Lack of response
	Stimulus Source	BizCo server at a branch location
	Environment	Normal operations
	Artifact	BizCo server
	Response	The failure is detected and the system is restored to normal operation
	Response Measure	90 seconds

Architectural Decisions and Reasoning	
Risks Potentially problematic architectural decisions.	1. Ping/echo protocol not specified 2. Decisions about the number and "temperature" of spares have not been prototyped 3. n...
Sensitivities Architectural parameters that significantly affects achievement of quality attributes.	1. Recovery time is sensitive to "temperature" and number of spares 2. 3. n...
Tradeoffs A tradeoff is a sensitivity point that affects multiple responses and affects them in opposite directions.	1. Recovery time versus cost 2. Ping frequency versus performance for normal operations 3. n...
Non-Risks Good architectural decisions that are frequently implicit in the architecture	1. Excellent prototyping and simulation of failure modes 2. Choice of off-the-shelf load balancers 3. n...
Other Issues	1. Training material needs to be created for operations staff. 2. 3. n...