Software Architecture DIT344

Truong Ho-Quang
truongh@chalmers.se
Software Engineering Division
Chalmers | GU

Schedule

Week		Date	Time	Lecture	
36	L1	Wed, 2 Sept	13:15 – 15:00	Introduction & Organization	Ho
37	L2	Wed, 9 Sept	13:15 – 15:00	Architecting Process & Views	Ho
37	S1	Thu, 10 Sept	10:15 – 12:00	<< Supervision/As	
38	L3	Wed, 16 Sept	13:15 - 15:00	Requirements & Quality Attributes	bara
38	S2	Thu, 17 Sept	10:15 – 12:00	<< Supervision/Assignment>>	TAs
38	L4	Fri, 18 Sept	13:15 – 15:00	Architectural Tactics & Roles and Responsibilities	Truong Ho
39	S3	Wed, 23 Sept	13:15 – 15:00	<< Supervision/Assignment>>	TAs
39	L5	Thu, 24 Sept	10:15 – 12:00	Functional Decomposition & Architectural Styles P1	Truong Ho
39	L6	Fri, 25 Sept	13:15 – 15:00	Architectural Styles P2	Truong Ho
40	S4	Wed, 30 Sept	13:15 – 15:00	<< Supervision/Assignment>>	TAs
40	L7	Thu, 1 Oct	10:15 – 12:00	Architectural Styles P3	Sam Jobara
40	L8	Fri, 2 Oct	13:00 – 15:00	Guest Lecture: Scaling DevOps – GitHub's Journey from 500+ to 1500+ People	Johannes Nicolai
41	S5	Wed, 7 Oct	13:15 – 15:00	<< Supervision/Assignment>>	TAs
41	L9	Thu, 8 Oct	10:15 – 12:00	Current Industrial SW Architecture Issues: Software Architectures of Blockchain with Case Study	Sam Jobara
42	L10	Wed, 14 Oct	13:15 – 15:00	Design Principles	Truong Ho
42	S6	Thu, 15 Oct	10:15 – 12:00	<< Supervision/Assignment>>	TAs
42	L11	Fri, 16 Oct	13:15 – 15:00	Guest Lecture: Architecture changes at Volvo Truck's Application System (TAS)	Anders Magnusson
43	L12	Wed, 21 Oct	13:15 – 15:00	Architecture Evaluation	Truong Ho
43	L13	Thu, 22 Oct	10:15 – 12:00	Reverse Engineering & Correspondence	Truong Ho
43		Fri, 23 Oct	13:00 – 15:00	To be determined (exam practice?)	Teachers
44	Exam	30 Oct	8:30 – 12:30		

We are HERE!

Assignment schedule

Week		Date	Lecture	Assignment 1 – Task 1 (A1T1)	Assignment 1 – Task 2 (A1T2)	Assignment 2 (A2)
36	L1	Wed, 2 Sept	Introduction & Organization			
37	L2	Wed, 9 Sept	Architecting Process & Views	A1T1 released		
37	S1	Thu, 10 Sept	<< Supervision/Assignment>>	Planing A1T1		
38	L3	Wed, 16 Sept	Requirements & Quality Attr.			
38	S2	Thu, 17 Sept	<< Supervision/Assignment>>	Work A1T1		
38	L4	Fri, 18 Sept	Tactics & Roles			
39	S3	Wed, 23 Sept	<< Supervision/Assignment>>	Work A1T1		
39	L5	Thu, 24 Sept	Decomposition & Style P1	Hand-in A1T1		
39	L6	Fri, 25 Sept	Architectural Styles P2		A1T2 released	
40	S4	Wed, 30 Sept	<< Supervision/Assignment>>	Feedback A1T1	Planing A1T2	
40	L7	Thu, 1 Oct	Architectural Styles P3			
40	L8	Fri, 2 Oct	Industrial lecture 1			
41	S5	Wed, 7 Oct	<< Supervision/Assignment>>		Work A1T2	A2 released
41	L9	Thu, 8 Oct	Industrial lecture 2			
42	L10	Wed, 14 Oct	Design Principles			
42	S6	Thu, 15 Oct	<< Supervision/Assignment>>		Work A1T2	Work A2
42	L11	Fri, 16 Oct	Industrial lecture 3		Hand-in A1T2	
43	L12	Wed, 21 Oct	Architecture Evaluation		Feedback A1T2	
43	L13	Thu, 22 Oct	Reverse Engineering			Hand-in A2
43		Fri, 23 Oct	Exam practice			Tue, 27 Oct: Feedback A2
44	Exam	30 Oct				

Assignment schedule

Week		Date	Lecture	Assignment 1 – Task 1 (A1T1)	Assignment 1 – Task 2 (A1T2)	Assignment 2 (A2)
36	L1	Wed, 2 Sept	Introduction & Organization			
37	L2	Wed, 9 Sept	Architecting Process & Views	A1T1 released		
37	S1	Thu, 10 Sept	<< Supervision/Assignment>>	Planing A1T1		
38	L3	Wed, 16 Sept	Requirements & Quality Attr.			
38	S2	Thu, 17 Sept	<< Supervision/Assignment>>	Work A1T1		
38	L4	Fri, 18 Sept	Tactics & Roles			
39	S3	Wed, 23 Sept	<< Supervision/Assignment>>	Work A1T1		
39	L5	Thu, 24 Sept	Decomposition & Style P1	Hand-in A1T1		
39	L6	Fri, 25 Sept	Architectural Styles P2		A1T2 released	
40	S4	Wed, 30 Sept	<< Supervision/Ass…nent>>	Feedback A1T1	Planing A1T2	
40	L7	Thu, 1 Oct	Architectural Style…			
40	L8	Fri, 2 Oct				
41	S5	Wed, 7 Oc…	…>		Work A1T2	A2 released
41	L9	Thu, 8 Oct				
42	L10	Wed, 14 C…				
42	S6	Thu, 15 Oc…	…>		Work A1T2	Work A2
42	L11	Fri, 16 Oct	Industrial lecture 3		Hand-in A1T2	
43	L12	Wed, 21 Oct	Architecture Evaluation		Feedback A1T2	
43	L13	Thu, 22 Oct	Reverse Engineering			Hand-in A2
43		Fri, 23 Oct	Exam practice			Tue, 27 Oct: Feedback A2
44	Exam	30 Oct				

We are HERE!

First hand-in in 6 days

Supervision sessions are mandatory

If you cannot attend a supervision session (with a valid reason):

- Inform your team members
- Inform your supervisors prior to the session
- Catch up with your team members!

Voluntary student representatives

Drop me an email by **Monday, Sept. 21** if you want to become a student representative of this course.

Recap of previous lectures

- L1: What, Why, How? SW Architecture
- L2: Architecting process, stakeholders, views
- L3: Requirements – Quality attributes

Part I: Architectural Tactics

Objectives

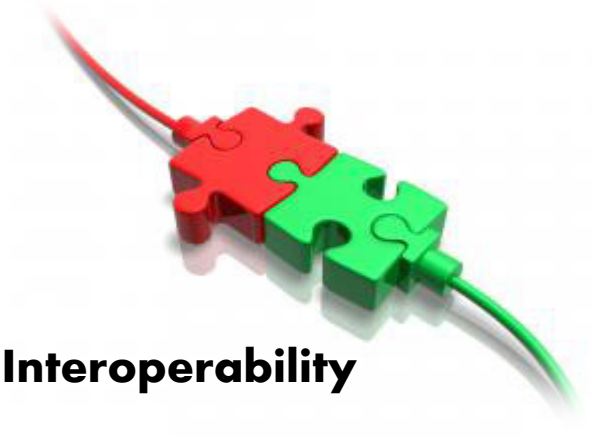
- In this lecture you will learn:
  - What is an **architectural tactic**
  - How to address quality requirements through tactics
  - A catalogue of tactics

Tactics

- A tactic is a design decision that influences the achievement of a quality attribute response
  - Different tactics for each quality attribute
  - The same tactic could be relevant to many quality attributes
  - No consideration of tradeoffs
  - Building blocks of archiectural styles

Example Quality Attributes

Availability



Interoperability

Performance

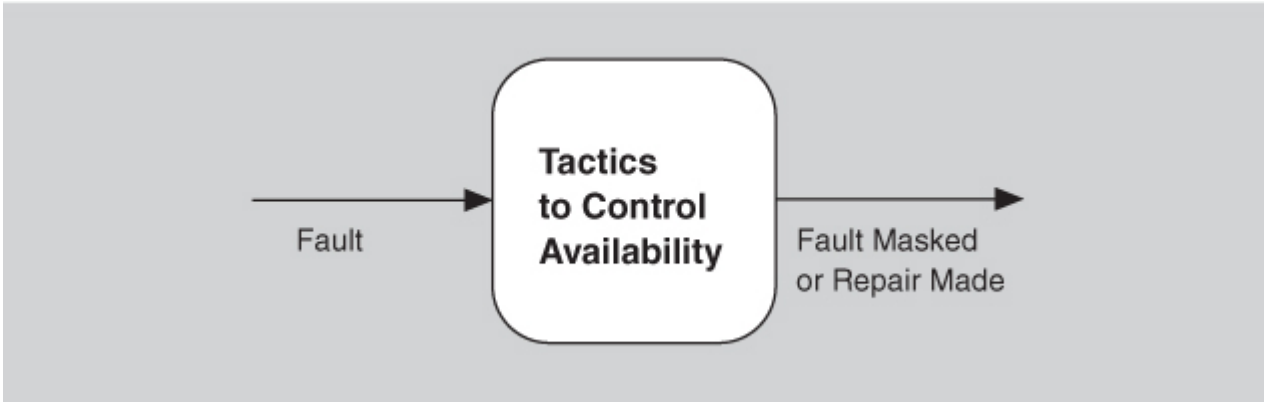
Modifiability

Security

Availability

- **Definition**:

  The ability of a system to mask or repair faults such as the cumulative service outage period does not exceed a required value over a specified time interval.



Availability Tactics

- Recall: A failure occurs when the system no longer delivers a service that is consistent with its specification;
- Possibly a fault or a combination of faults has/have the potential to cause a failure
- Recall: also that Recovery and Repair are also an important aspect of availability.
- The tactics we discuss in this lecture will keep faults from becoming failures or at least bound the effect of faults and make repair possible.

Approaches to maintaining availability

It may involve some type of:

- Redundancy
- Health monitoring to detect a failure
- Recovery when a fault is detected

- The monitoring and recovery can either be automatic or manual

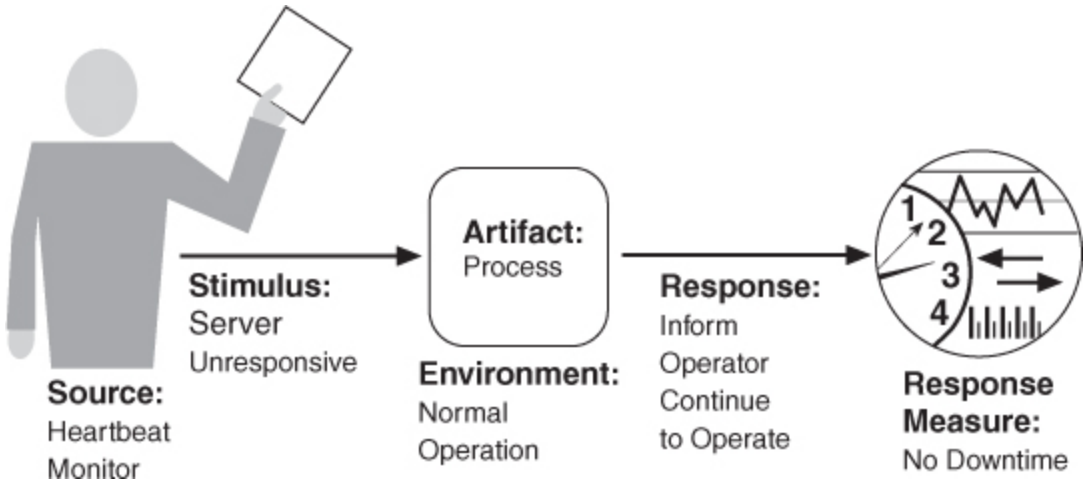
From Lecture 3

Availability General Scenario

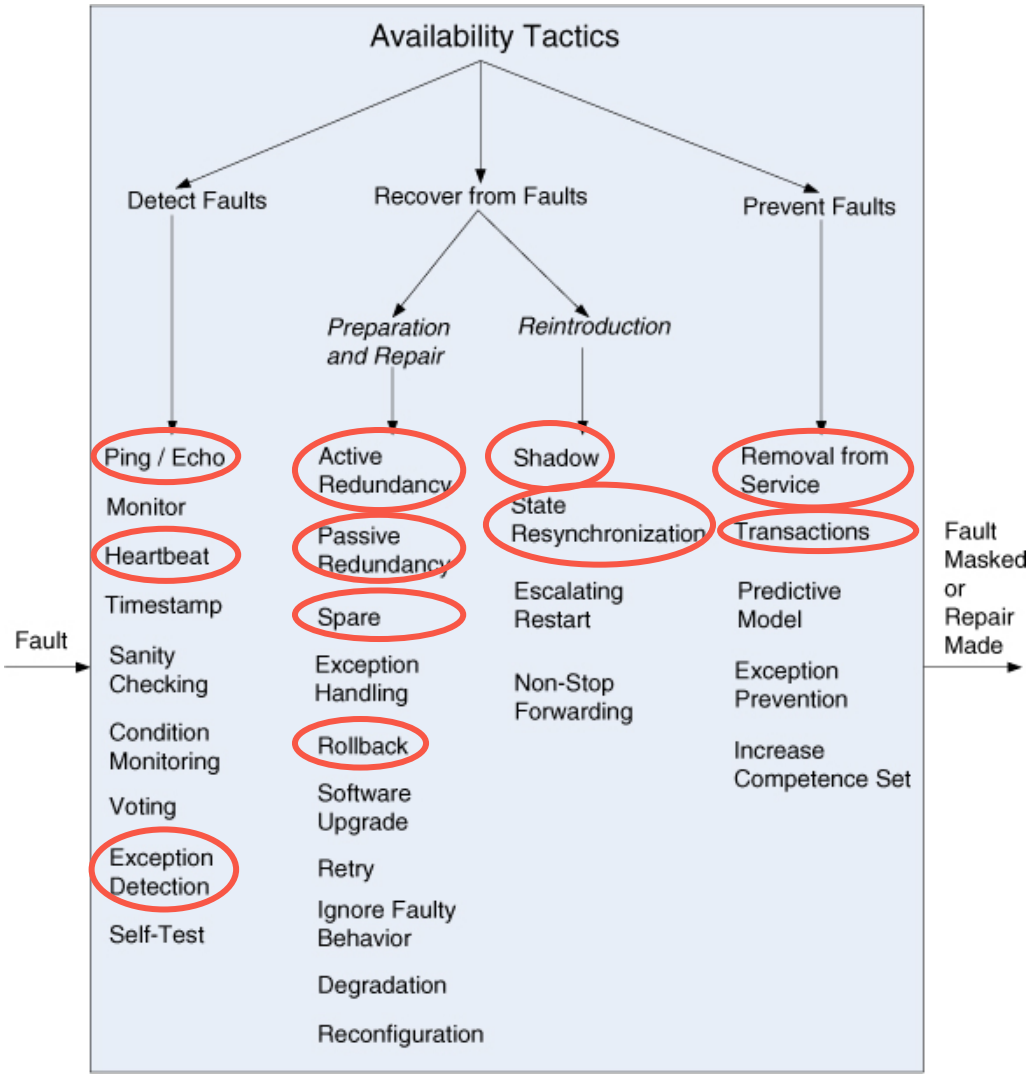
Portion of Scenario	Possible Values
Source	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	Fault: omission, crash, incorrect timing, incorrect response
Artifact	Processors, communication channels, persistent storage, processes
Environment	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
Response	Prevent the 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
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

From Lecture 3

Availability Concrete Scenario



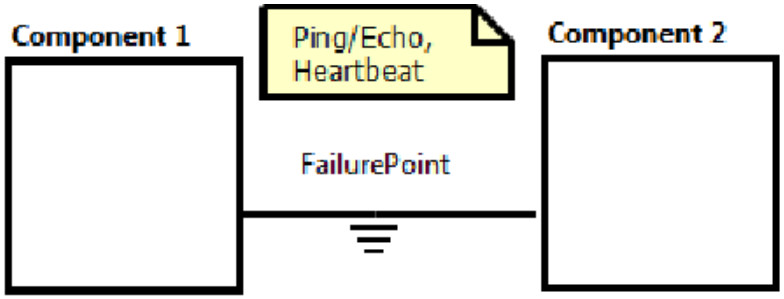
Availability Tactics



Availability tactic: Fault Detection

Ping/Echo

- One component issues a ping and expects to receive back an echo within a predefined time
- May be used with a single or group of components
- Clients use it to ensure that server object and communication path to server are operating within expected performance bounds
- Organized in hierarchy (higher level fault detector pings lower levels ones and vice versa)



(a) Ping/Echo and Heartbeat Modeling

Ping(response: 20ms, src: Component1, dest: Component2);
Heartbeat(pooling:5sec, src: Component1, dest: Component2)

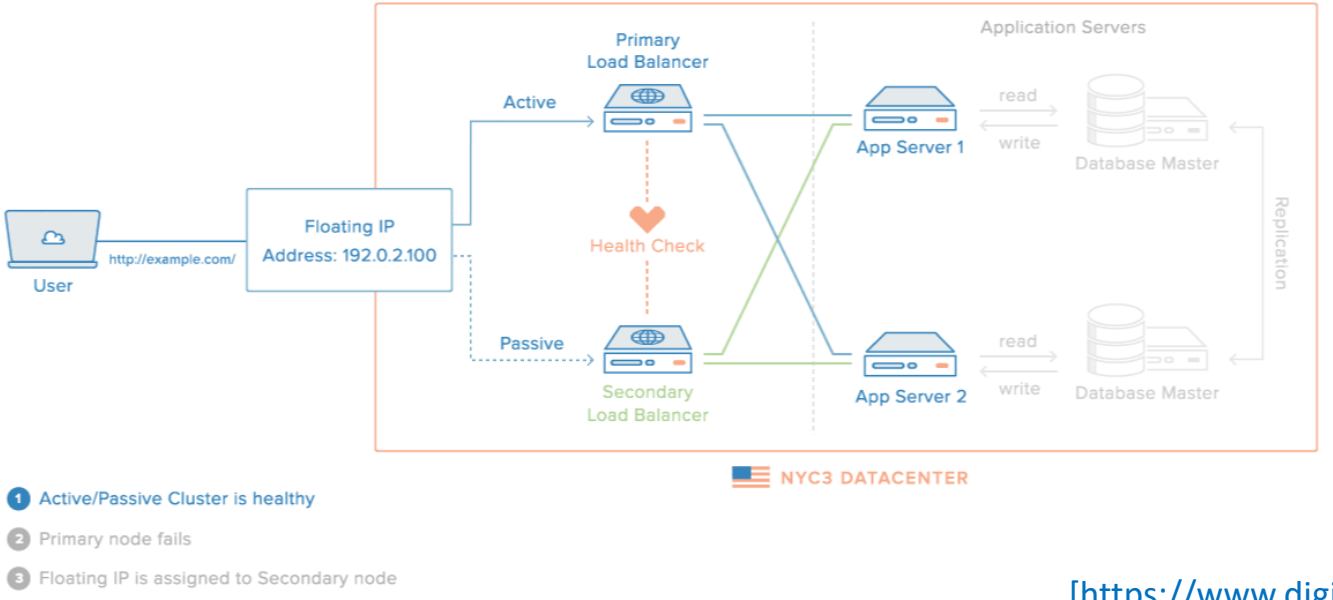
Availability tactic: Fault Detection

Heartbeat(dead man timer)

- One component emits a heartbeat message periodically and other component listens for it.
- If heartbeat fails, the originating component assumed to have failed and a fault correction component is notified
- A heartbeat can also carry data e.g. ATM

Exception

- Raised when one of the faults classes execute
- The exception handler typically executes in the same process that introduced exception

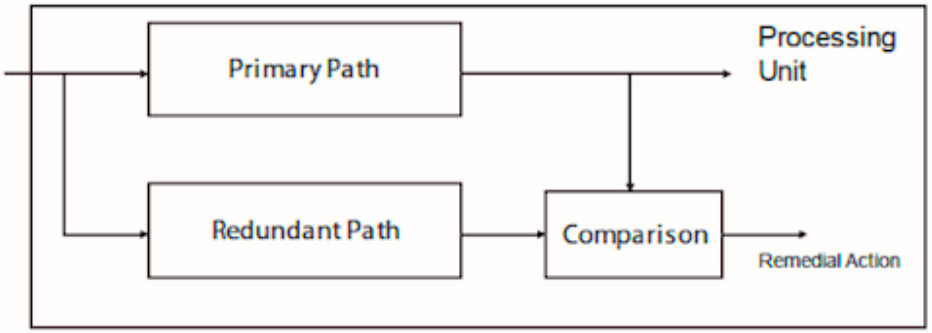


[https://www.digitalocean.com]

Availability tactic: Fault Recovery (Preparation & Repair)

Voting

- Processes running on redundant processors
- Each take equivalent input and a simple output value that is sent to the voter
- If the voter detects the deviate behavior from a single processor, it fails it
- The voting algorithm can be
  - Majority rules
  - Preferred component
  - Or some other
- This method is used to correct
  - Faulty operation of algorithm
  - Failure of processor



https://www.edn.com/redundancy-for-safety-compliant-automotive-other-devices/

ISO26262 Specs – Road vehicles – Functional safety

Availability tactic: Fault Recovery (Preparation & Repair)

Active redundancy (Hot restart)

- All redundant components respond to events in parallel

Passive redundancy (Warm restart/ dual redundancy/ triple redundancy)

- One component (the primary) responds to events and inform the other components (the standbys) of state updates they must make

Spare

- A standby spare computing platform is configured to replace many different failed components
- It must be rebooted to appropriate s/w configuration and have its state initialized when a failure occurs.
- Making a checkpoint of the system state to a persistent device periodically- resume to that persistent state afterwards

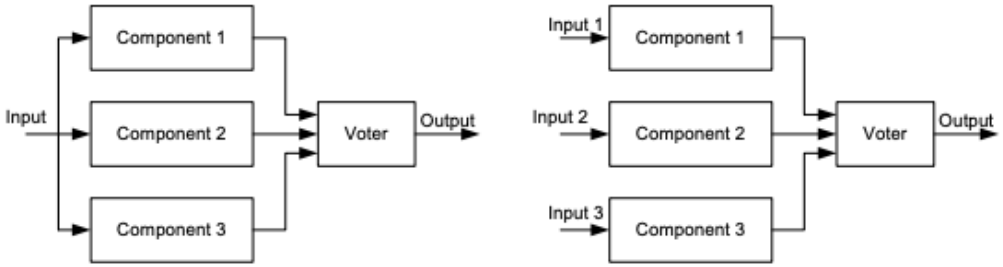


Figure 1. A TMR pattern and a variation

Wu, W. and Kelly, T., 2004, September. Safety tactics for software architecture design. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.* (pp. 368-375). IEEE.

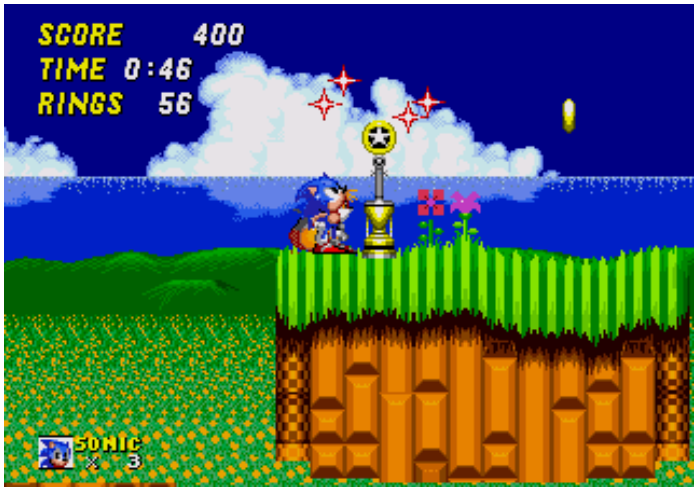
Availability tactic: Fault Recovery (Recovery-reintroduction)

Shadow operation

- A previously failed component may be run in “shadow-mode” for a short time.
- It mimics the behavior of working components before actual restore

State Synchronization

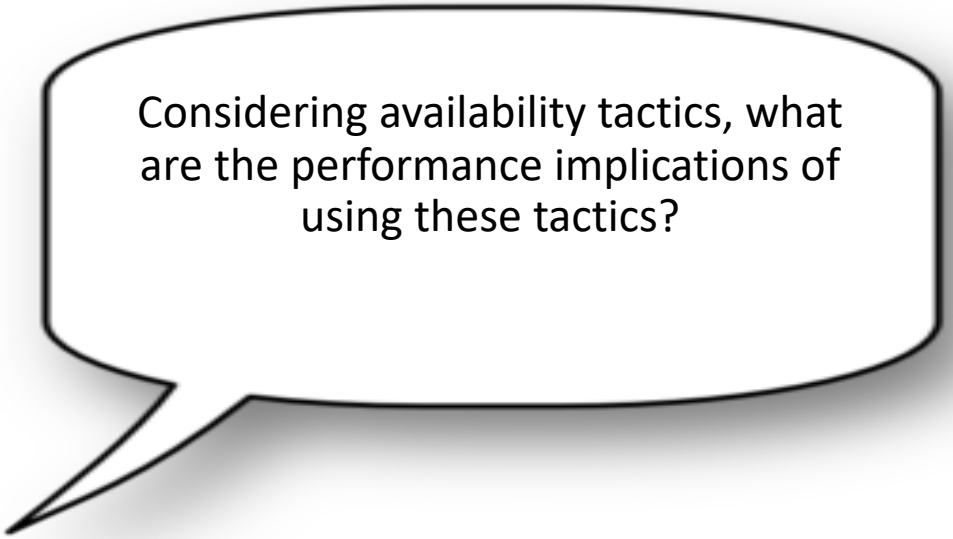
- Passive and active redundancy tactics require the component being restored to have its state upgraded before its return to service.
- State updating may depends on
  - Down time, size of update, number of messages to update
    - Single message to state-update is preferable
    - Incremental state upgrades with periods of service may lead to complicated s/w



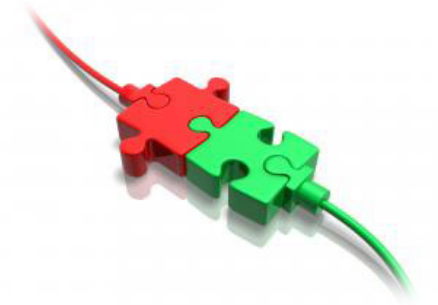
Checkpoint/ roll back

- It is a recording of consistent state created either periodically or in response to specific events.

Question

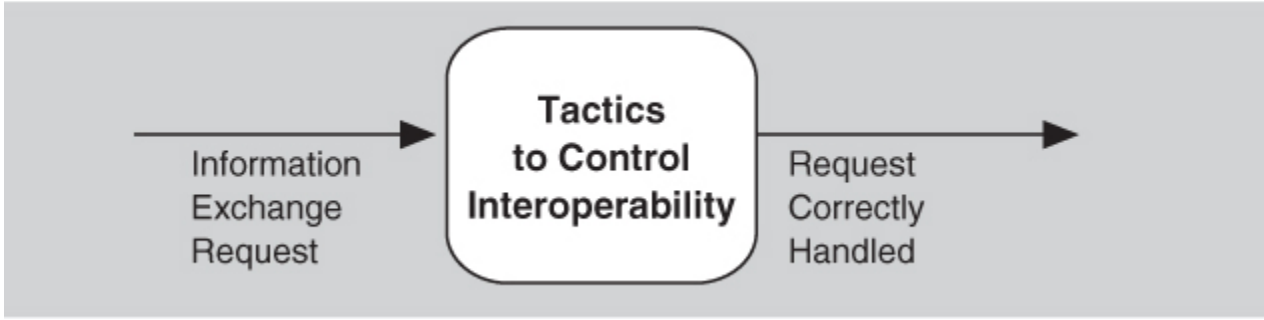

Considering availability tactics, what are the performance implications of using these tactics?

Interoperability



- Definition:

The degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context.



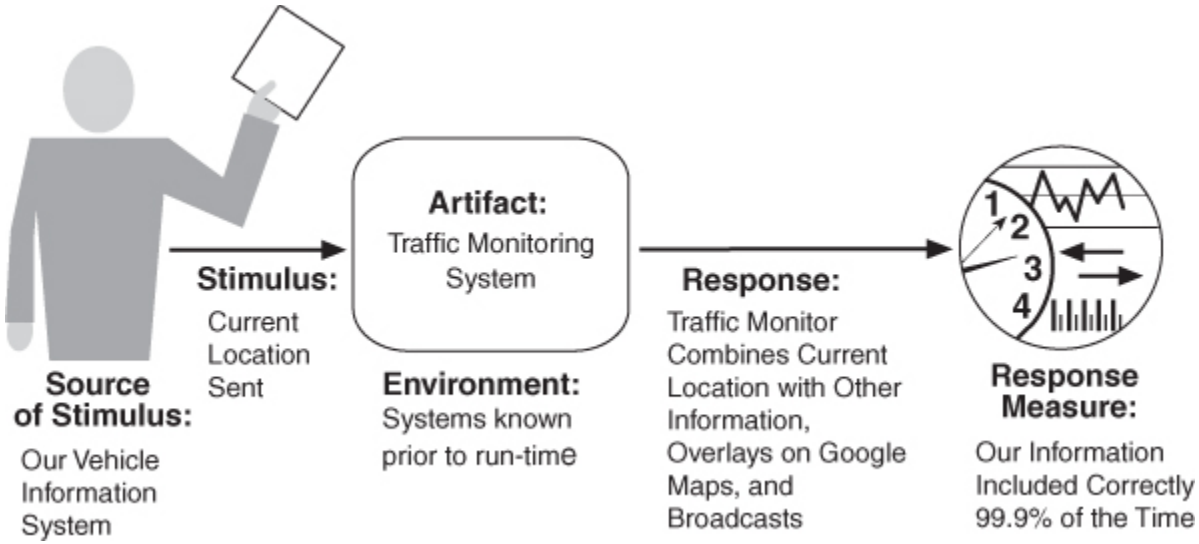
From Lecture 3

Interoperability General Scenario

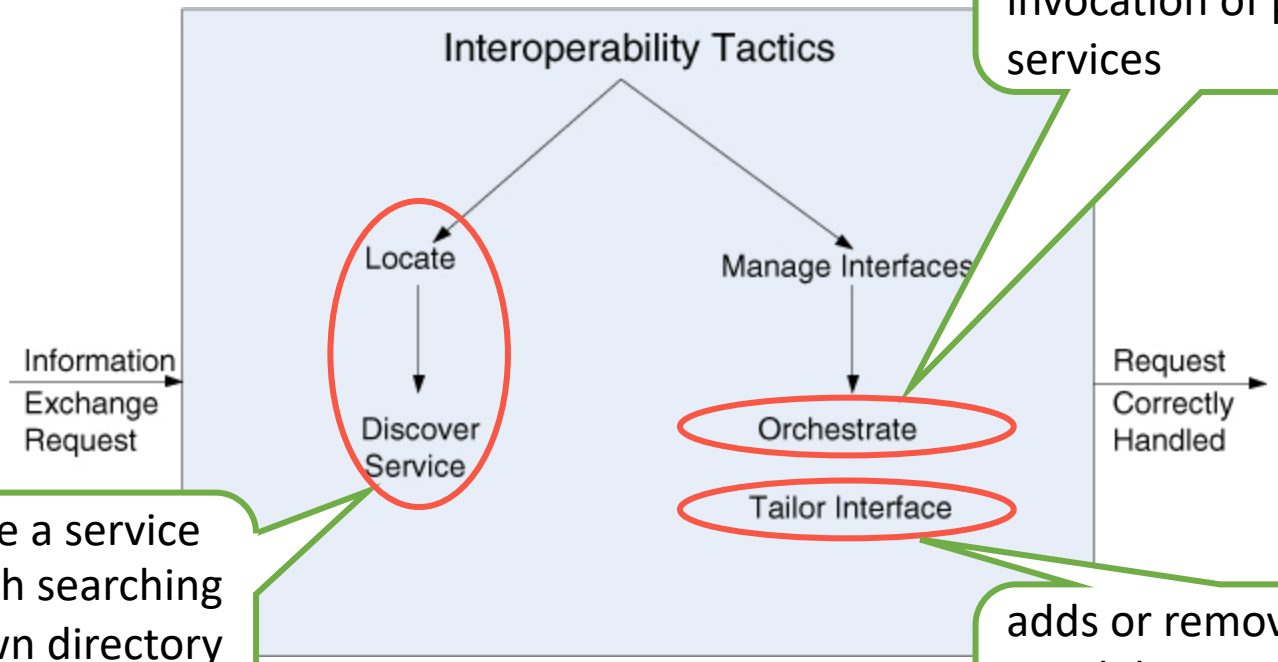
Portion of Scenario	Possible Values
Source	A system initiates a request to interoperate with another system.
Stimulus	A request to exchange information among system(s).
Artifact	The systems that wish to interoperate.
Environment	System(s) wishing to interoperate are discovered at runtime or known prior to runtime.
Response	One or more of the following: ▪ 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.
Response Measure	One or more of the following: ▪ Percentage of information exchanges correctly processed ▪ Percentage of information exchanges correctly rejected

From Lecture 3

Interoperability Concrete Scenario



Interoperability Tactics

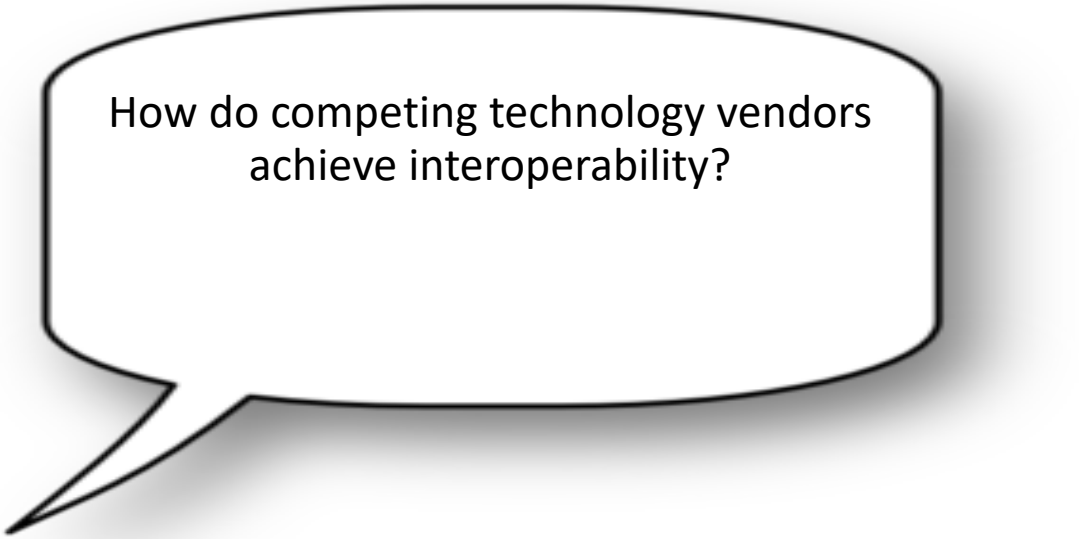


coordinate and manage and sequence the invocation of particular services

Locate a service through searching a known directory service.

adds or removes capabilities to an interface (e.g. to hide information from untrusted users)

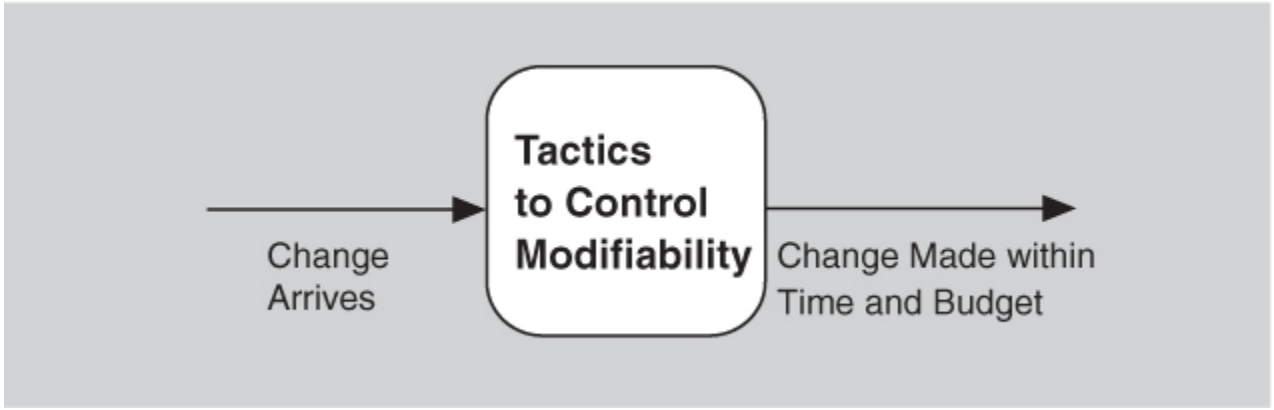
Question


How do competing technology vendors achieve interoperability?

Modifiability

- Definition:

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



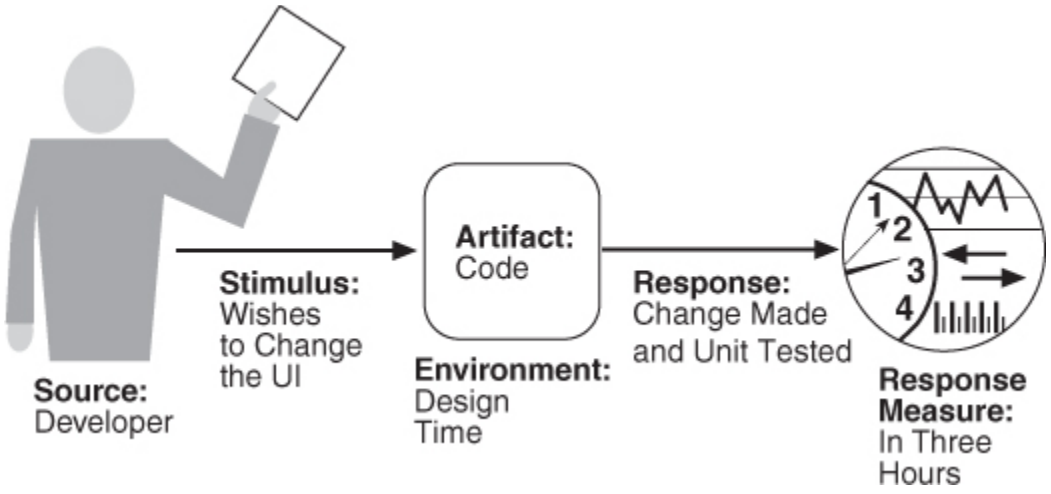
From Lecture 3

Modifiability General Scenario

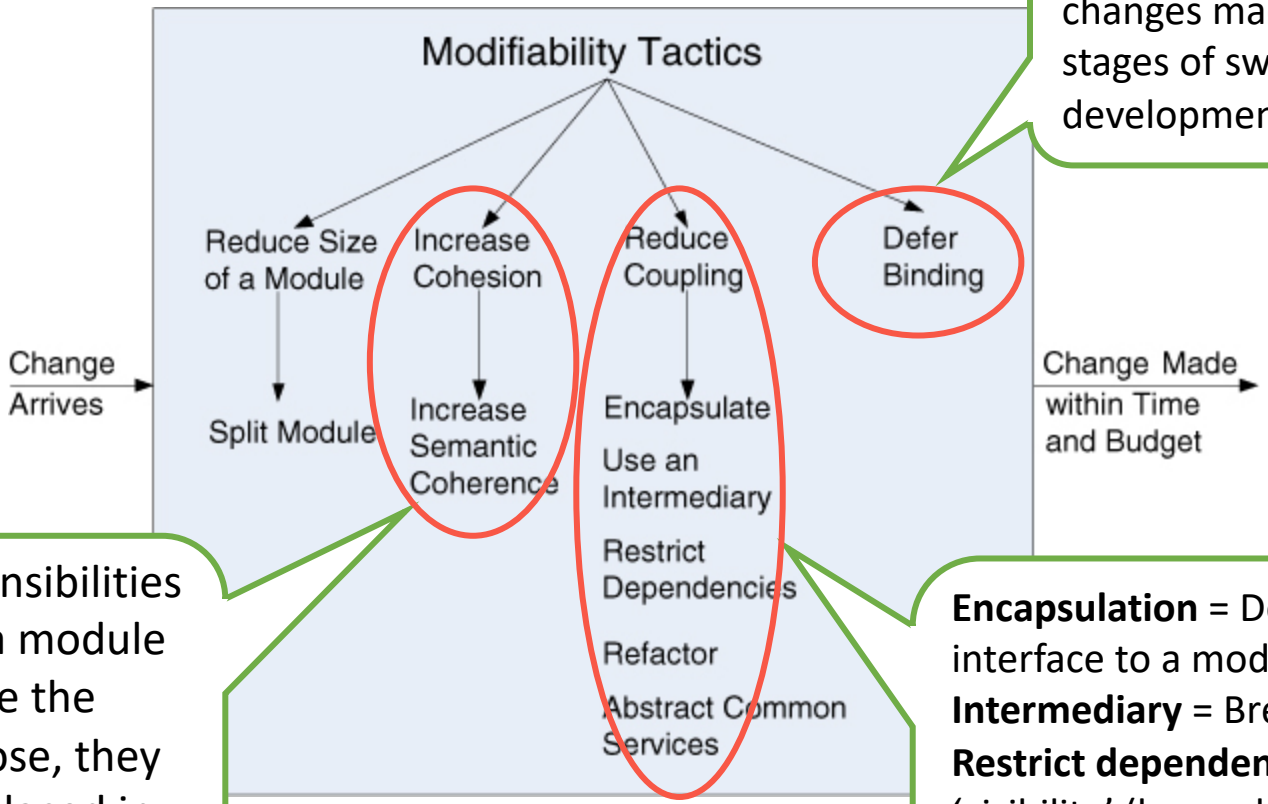
Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ▪ Make modification ▪ Test modification ▪ Deploy modification
Response Measure	Cost in terms of the following: ▪ 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

From Lecture 3

Modifiability Concrete Scenario



Modifiability Tactics

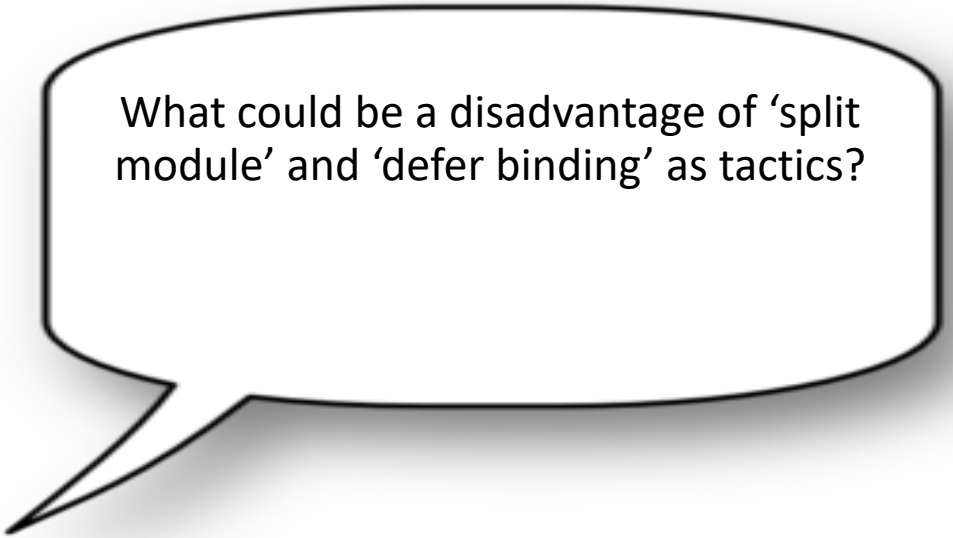


Flexibility of the system to changes made in different stages of sw development/deployment

If the responsibilities A and B in a module do not serve the same purpose, they should be placed in different modules

Encapsulation = Define an explicit interface to a module.
Intermediary = Break a dependency
Restrict dependencies = control 'visibility' (layered architecture)
Refactor = reducing duplicate code/responsibilities

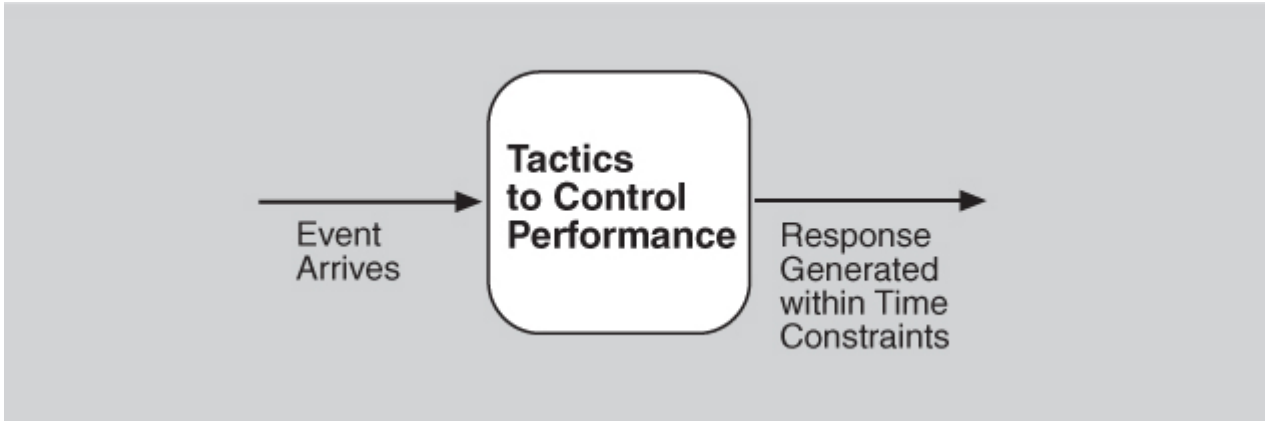
Question


What could be a disadvantage of ‘split module’ and ‘defer binding’ as tactics?

Performance

- Definition:

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



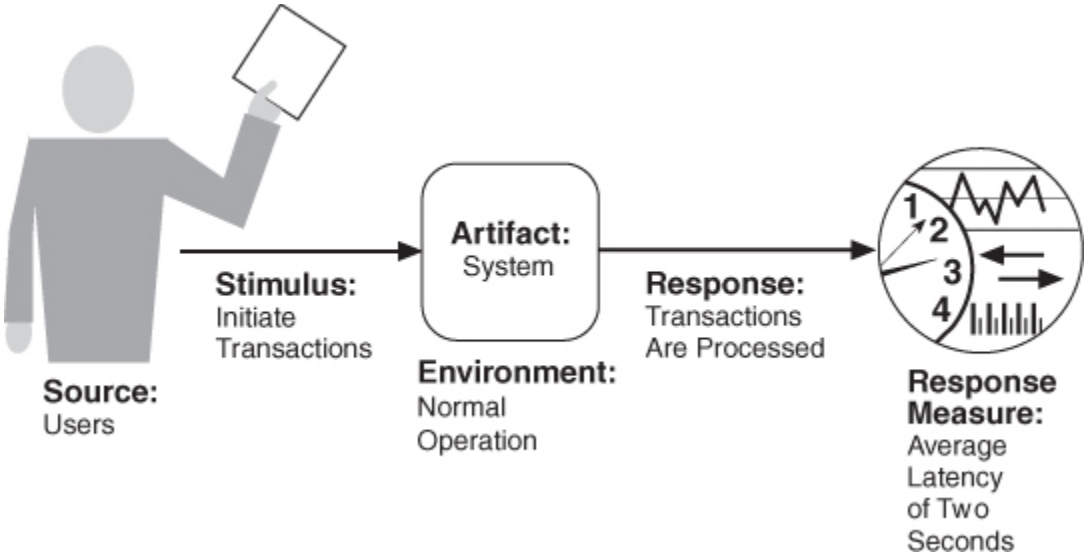
From Lecture 3

Performance General Scenario

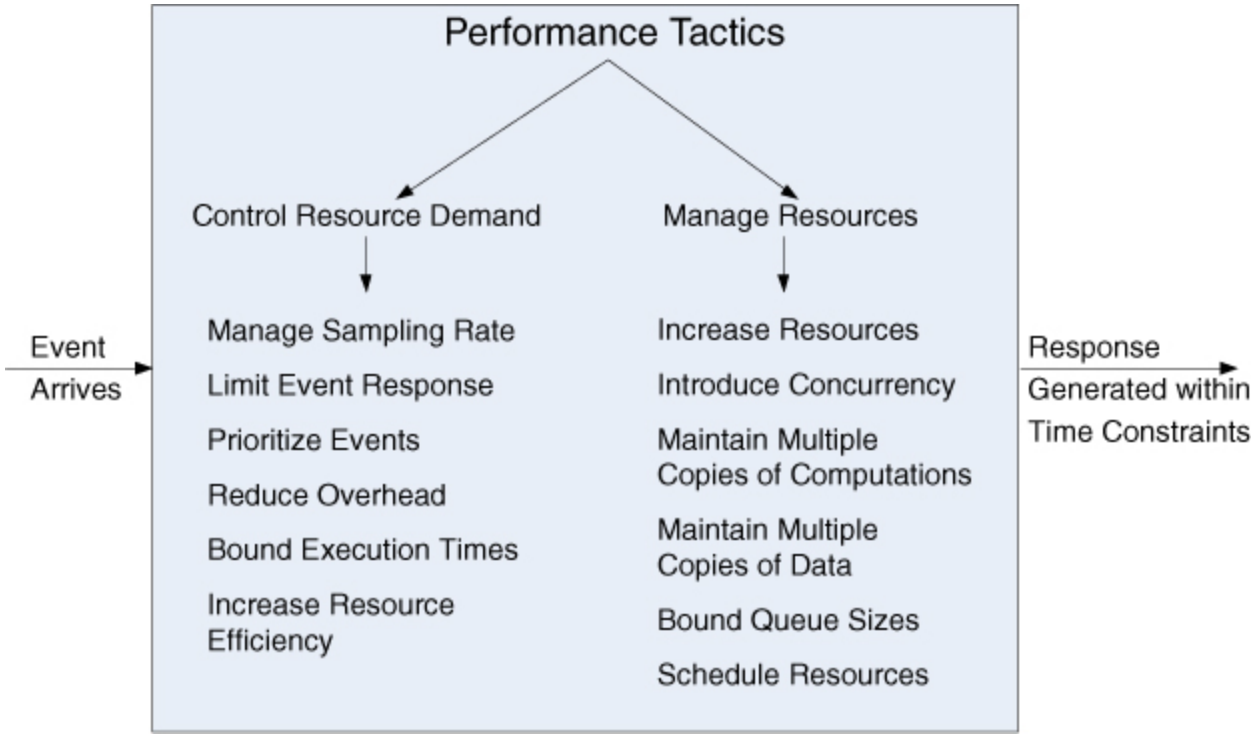
Portion of Scenario	Possible Values
Source	Internal or external to the system
Stimulus	Arrival of a periodic, sporadic, or stochastic event
Artifact	System or one or more components in the system
Environment	Operational mode: normal, emergency, peak load, overload
Response	Process events, change level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

From Lecture 3

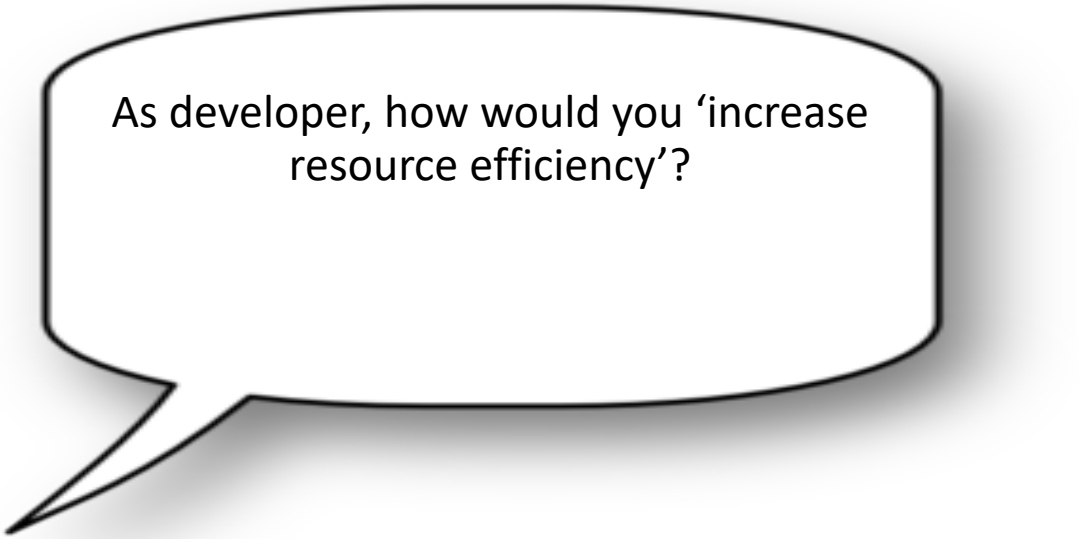
Performance Concrete Scenario



Performance Tactics



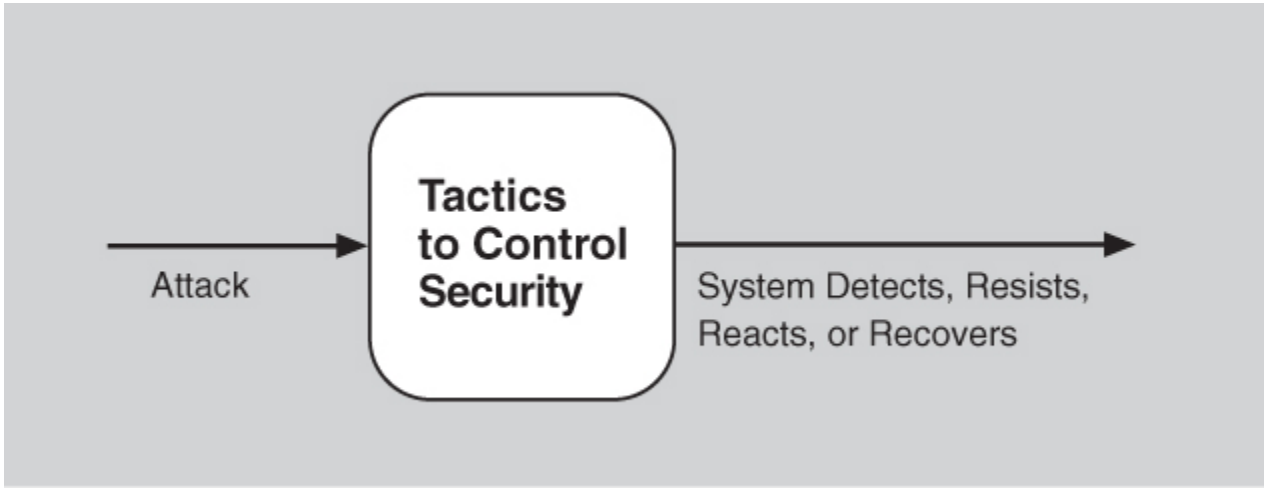
Question



Security

- Definition:

Security is a measure of the system's ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized.



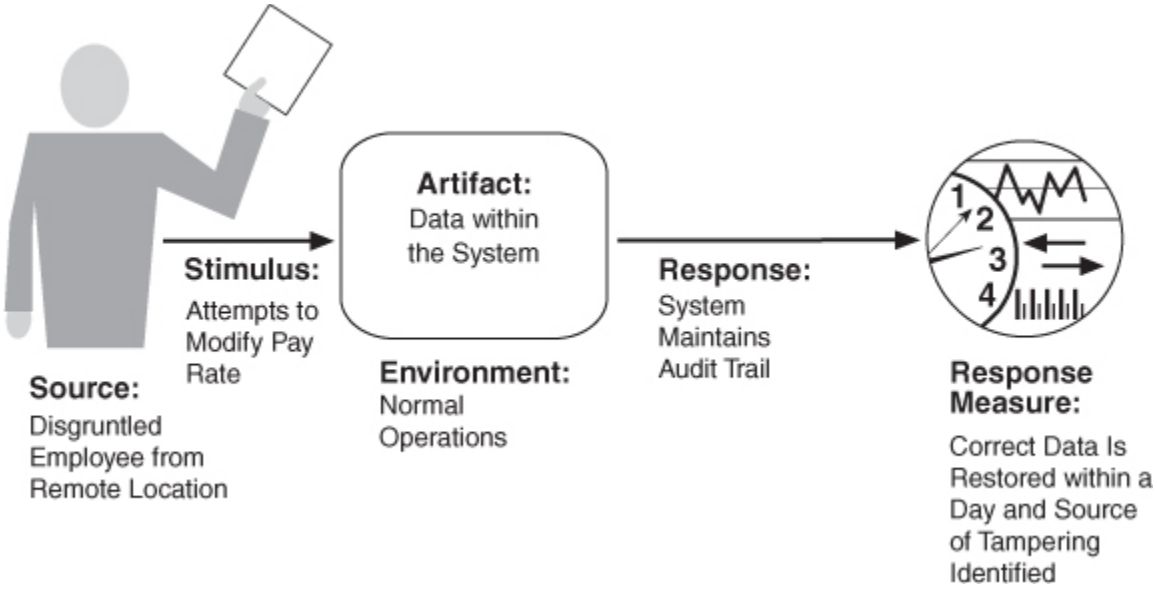
From Lecture 3

Security General Scenario

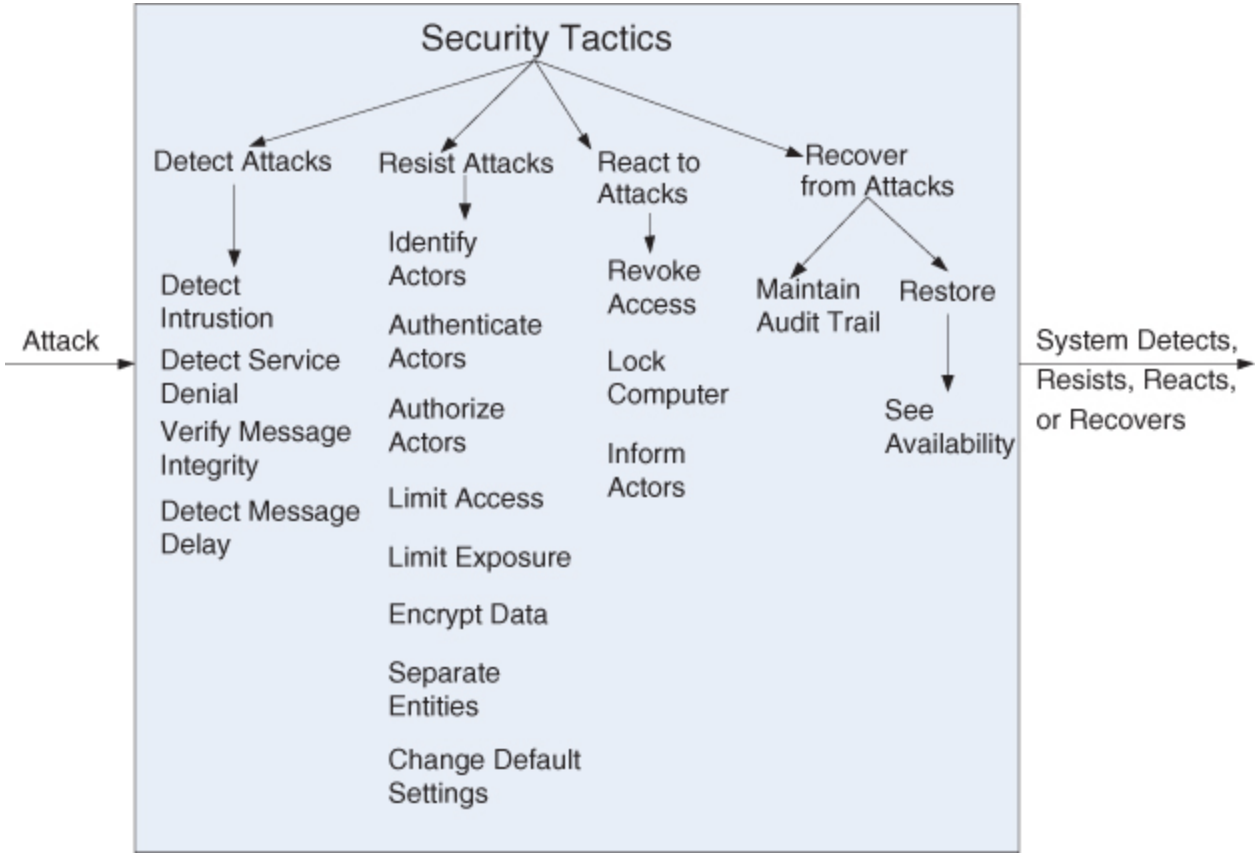
Portion of Scenario	Possible Values
Source	Human or another system which may have been previously identified (either correctly or incorrectly) or may be currently unknown. A human attacker may be from outside the organization or from inside the organization.
Stimulus	Unauthorized attempt is made to display data, change or delete data, access system services, change the system's behavior, or reduce availability.
Artifact	System services, data within the system, a component or resources of the system, data produced or consumed by the system
Environment	The system is either online or offline; either connected to or disconnected from a network; either behind a firewall or open to a network; fully operational, partially operational, or not operational.
Response	Transactions are carried out in a fashion such that ▪ Data or services are protected from unauthorized access. ▪ Data or services are not being manipulated without authorization. ▪ Parties to a transaction are identified with assurance. ▪ The parties to the transaction cannot repudiate their involvements. ▪ The data, resources, and system services will be available for legitimate use. The system tracks activities within it by ▪ Recording access or modification ▪ Recording attempts to access data, resources, or services ▪ Notifying appropriate entities (people or systems) when an apparent attack is occurring
Response Measure	One or more of the following: ▪ How much of a system is compromised when a particular component or data value is compromised ▪ How much time passed before an attack was detected ▪ How many attacks were resisted ▪ How long does it take to recover from a successful attack ▪ How much data is vulnerable to a particular attack

From Lecture 3

Security Concrete Scenario



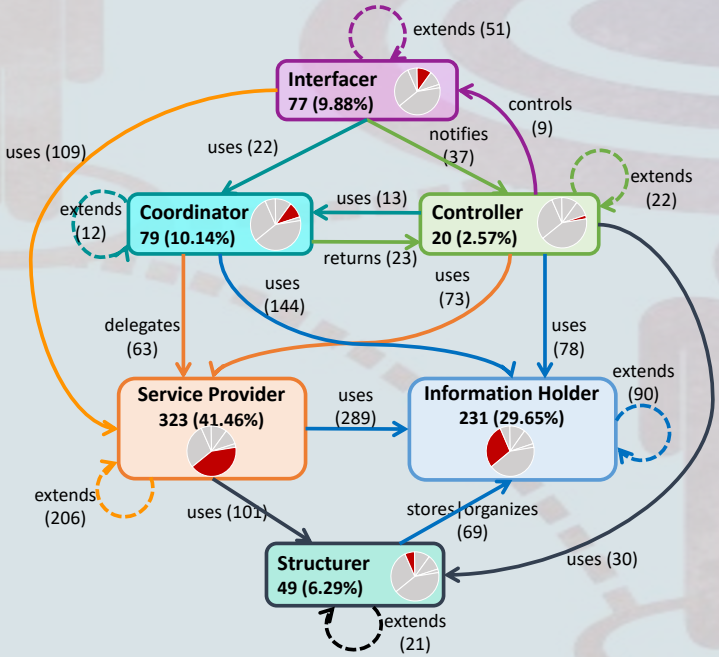
Security Tactics

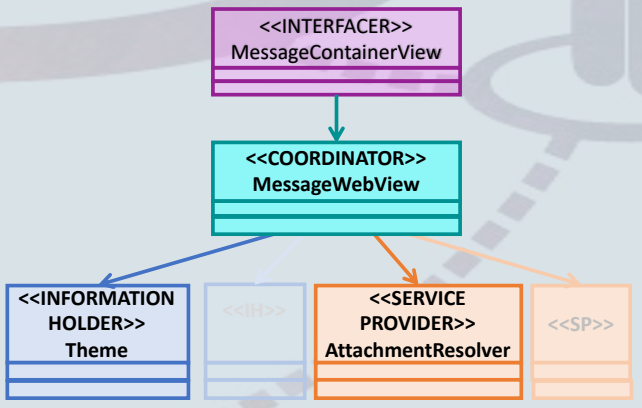


Summary Part I

- Definition of Architectural Tactic
- Well-known tactics for some of most common quality attributes
- Tactics are building blocks of architectural styles
- Tactics can get outdated!

Part II:
Roles & Responsibilities





Theme/Objective of this lecture

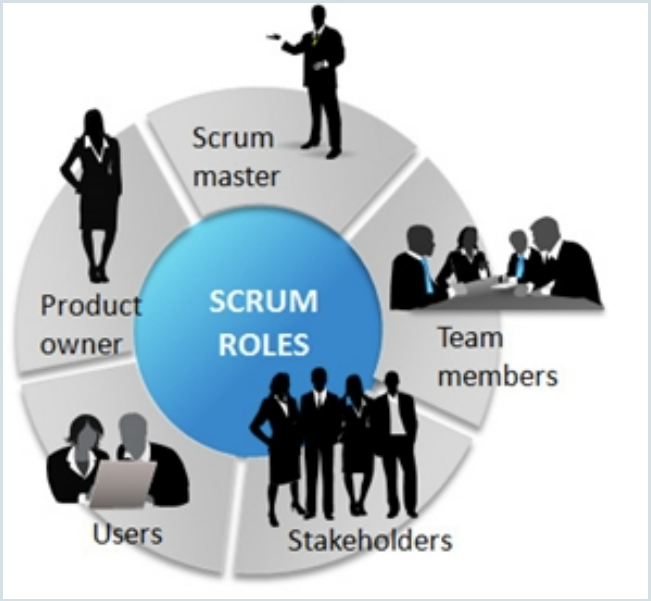
- Understand the importance of being aware of role when designing software.

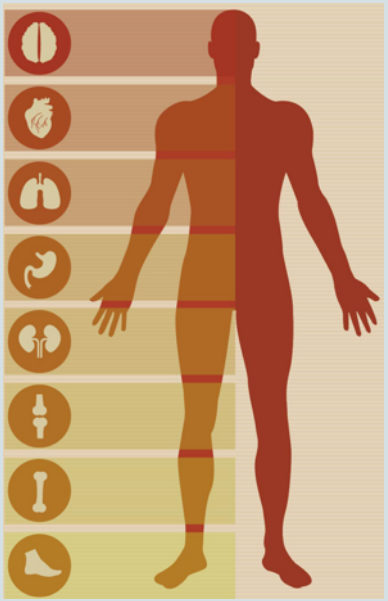
- Build vocabulary for characterizing role/responsibility
  - a set of six(6) common roles (role stereotypes)
  - collaborations between role stereotypes

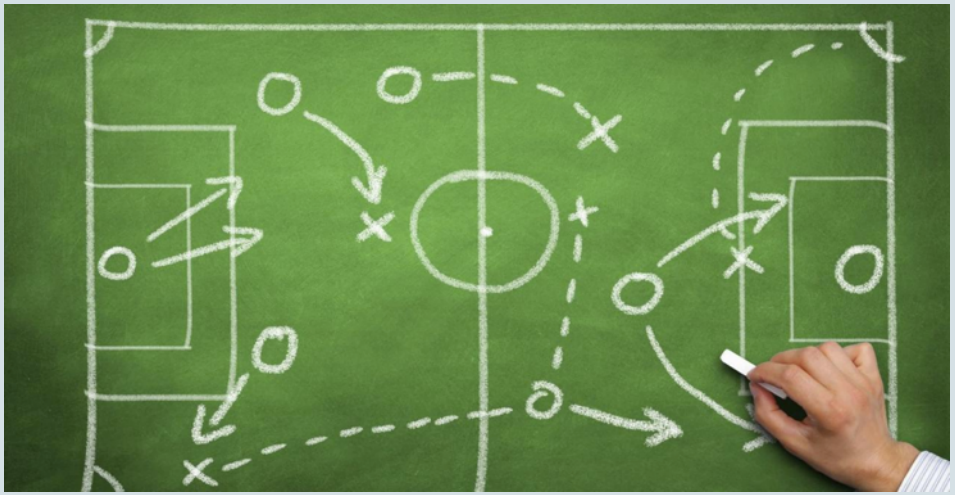
- Exploring impacts of role/stereotype in design quality metrics in two realistic cases

What is role & responsibility?

Where to find role/responsibility?







Why defining role is so important?

- To establish working scope
- To seek agreement
- To facilitate communication/collaboration when performing tasks
- Less waste

Role & Responsibility in Software Design

- Software is a set of components that
  - carry different roles
  - collaborate with different components
- Being aware of component's role when designing would help to:
  - achieve better distribution of responsibility
  - manage complexity/communication
  - avoid redundancy
  - increase mainteability

Role Stereotypes

- Definition
- Relationships between role stereotypes

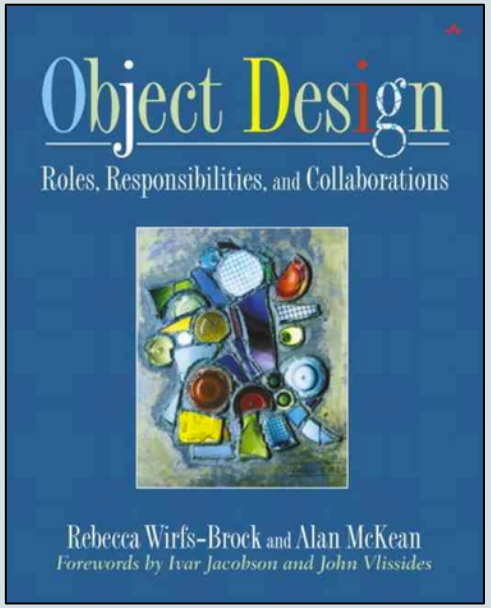
Stereotype

A conventional, formulaic, and oversimplified conception, opinion, or image

(www.thefreedictionary.com)

Role Stereotypes

- The concept "role stereotype" was introduced by Rebecca Wirfs-Brock.
- The concept indicates generic roles that an software object plays in the design.
- It is recommended that each object carries a single role/responsibility.



Object Design: Roles, Responsibilities and Collaborations, Rebecca Wirfs-Brock and Alan McKean, Addison-Wesley, 2003

- **Service providers** do things
- **Interfacers** translate requests and convert from one level of abstraction to another
- **Information holders** know things
- **Controllers** direct activities
- **Coordinators** delegate work
- **Structurers** manage object relations or organize large numbers of similar objects

Role stereotypes

Service Provider (SP)

- performs specific work
- offers services

'-er', '-or'; public static methods; might contains some logics to do specific tasks.

Information Holder (IH)

- knows/keeps information
- provides information

data encapsulation; get/set methods; private/internal methods

Interfacer (IF)

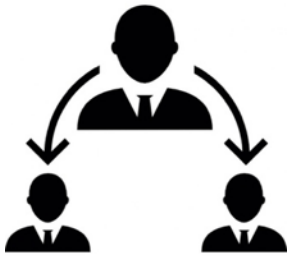
- transforms/converts information and requests b/t SW layers

GUI-related; storefront; API; extension points

Controller (CT)

- makes decision
- control complex tasks

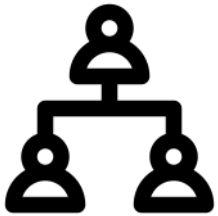
'controller', 'manager'; logic statements; knows IH, SP, CO



Coordinator (CO)

- delegates works
- forwards info requests

no/simple logic; knows requester & requestee



Structurer (ST)

- keeps/maintains relationship
- pool, collects, arranges objs

Collection; sort(); compare(); validate(); add(); remove(); ...

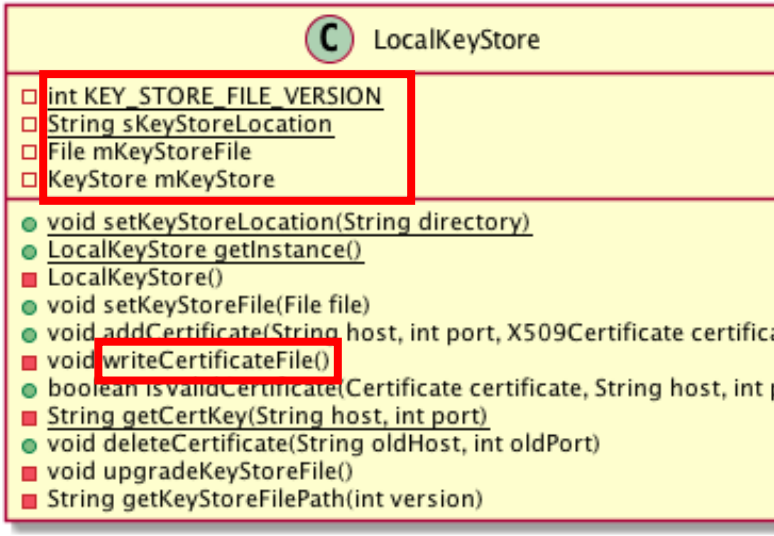
Information Holder (IH)

is a software element that

- keeps/knows information
- provides information to other elements

Example: An IH *class* might be characterized by:

- The class may just contains attributes
- Methods, if any, could be
  - Getters and setter
  - Persistence methods, eg. saving to database or implements Java's Serializable interface
  - Methods that are only used within the class



Service Provider (SP)

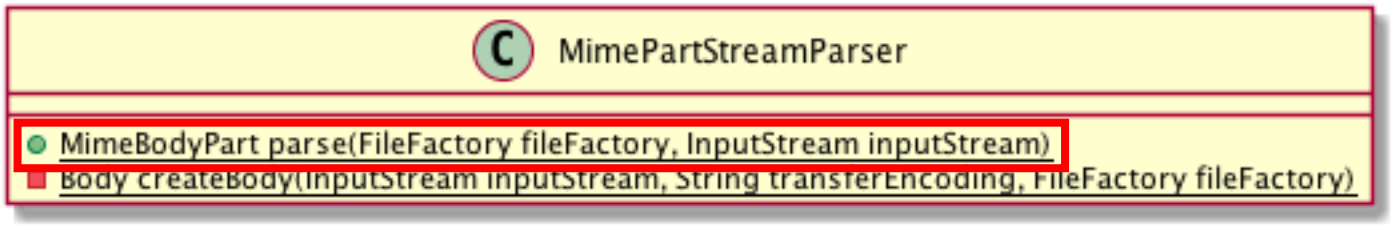
is a software element that

- performs specific works
- offers services to other elements on demand

A SP class can be characterized by:

- having name ended with “-er” (eg. *Provider*) or “-or” (eg. *Creator*, *Detector*)
- has methods and attributes are easily accessed by other classes (often static and public, or protected, not private)
- could be realization of a Interface
- decision making in methods should be at basic level, only to support specific work

Service Provider Class - Example

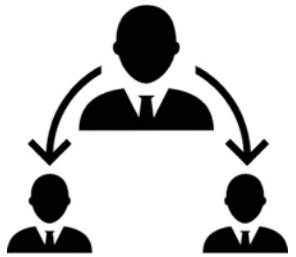



```
public static MimeBodyPart parse(FileFactory fileFactory, InputStream inputStream)
        throws MessagingException, IOException {
    MimeBodyPart parsedRootPart = new MimeBodyPart();

    MimeConfig parserConfig  = new MimeConfig();
    parserConfig.setMaxHeaderLen(-1);
    parserConfig.setMaxLineLen(-1);
    parserConfig.setMaxHeaderCount(-1);

    MimeStreamParser parser = new MimeStreamParser(parserConfig);
    parser.setContentHandler(new PartBuilder(fileFactory, parsedRootPart));
    parser.setRecurse();

    try {
        parser.parse(new EOLConvertingInputStream(inputStream));
    } catch (MimeException e) {
        throw new MessagingException("Failed to parse decrypted content", e);
    }

    return parsedRootPart;
}
```

```
public MimeBodyPart processData(InputStream is) throws IOException {
    try {
        FileFactory fileFactory =
                DecryptedFileProvider.getFileFactory(context);
        return MimePartStreamParser.parse(fileFactory, is);
    } catch (MessagingException e) {
        Timber.e(e, "Something went wrong while parsing the decrypted MIME part");
        //TODO: pass error to main thread and display error message to user
        return null;
    }
}
```

calls service

Coordinator (CO)



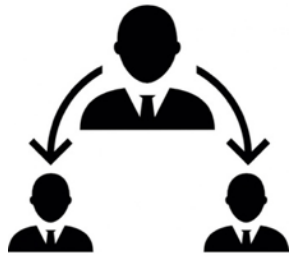
is a software element that

- does not make decisions
- delegates work to other objects
- forwards info/requests

Signs of a CO class:

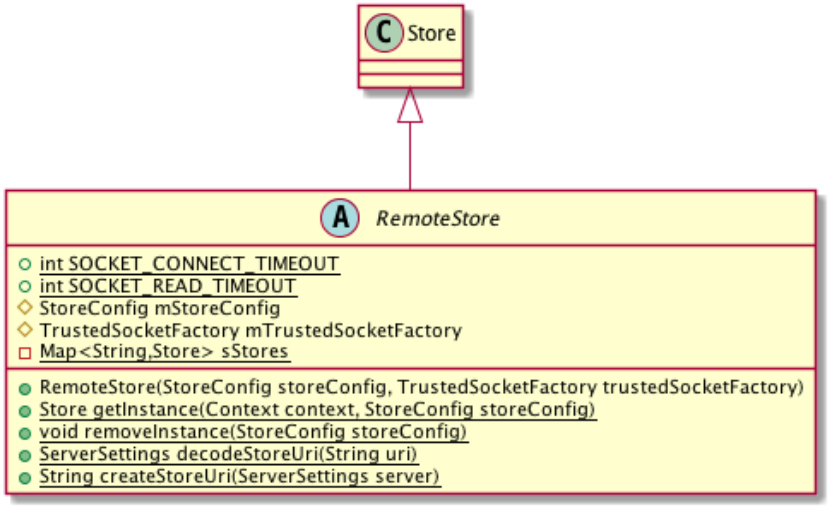
- Holding connection between working objects (SP, CT)
- Forwarding information and requests
  - it is important to define which classes are **requester** and **requestee**
  - information: method parameters; variables …
- When a Service Provider becomes too big, it evolves into Coordinator
  - Results of refactoring god classes

Coordinator Class - Example



```
public static String createStoreUri(ServerSettings server) {
    if (Type.IMAP == server.type) {
        return ImapStore.createUri(server);
    } else if (Type.POP3 == server.type) {
        return Pop3Store.createUri(server);
    } else if (Type.WebDAV == server.type) {
        return WebDavStore.createUri(server);
    } else {
        throw new IllegalArgumentException("Not a valid store URI");
    }
}
```

coordinates the works to ImapStore, Pop3Store, WebDavStore



Controller (CT)

is a software element that

- make decisions
- control complex tasks

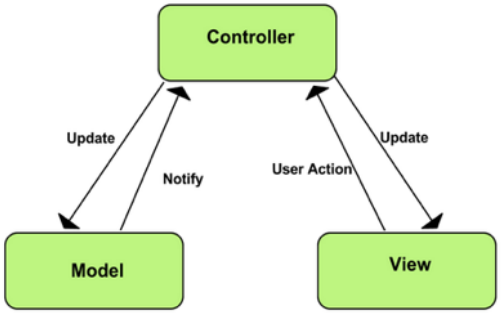
A CT class might be characterized by:

- having class name ended with “Controller”, “Manager”
- Should have access to information holders, coordinators, or service provider
- Its main responsibility is to make decision to control the flow of the application
  - Should contain condition statements (e.g. IF, IF ELSE, SWITCH CASE, x : ?)
  - The decision should be at the higher level than decision made at SP/CO.

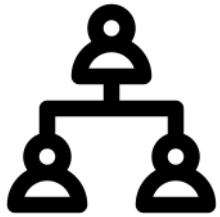
Controller Class - Example

```
public boolean checkRecipientsOkForSending() {
    recipientMvpView.recipientToTryPerformCompletion();
    recipientMvpView.recipientCcTryPerformCompletion();
    recipientMvpView.recipientBccTryPerformCompletion();

    if (recipientMvpView.recipientToHasUncompletedText()) {
        recipientMvpView.showToUncompletedError();
        return true;
    }

    if (recipientMvpView.recipientCcHasUncompletedText()) {
        recipientMvpView.showCcUncompletedError();
        return true;
    }

    if (recipientMvpView.recipientBccHasUncompletedText()) {
        recipientMvpView.showBccUncompletedError();
        return true;
    }

    if (getToAddresses().isEmpty() && getCcAddresses().isEmpty() && getBccAddresses().isEmpty()) {
        recipientMvpView.showNoRecipientsError();
        return true;
    }

    return false;
```

Delegating the work



Structurer (ST)



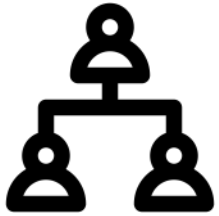
is a software element that

- maintains relationships between software components
- pools/collects/arranges a set of elements

A ST class might be characterized by:

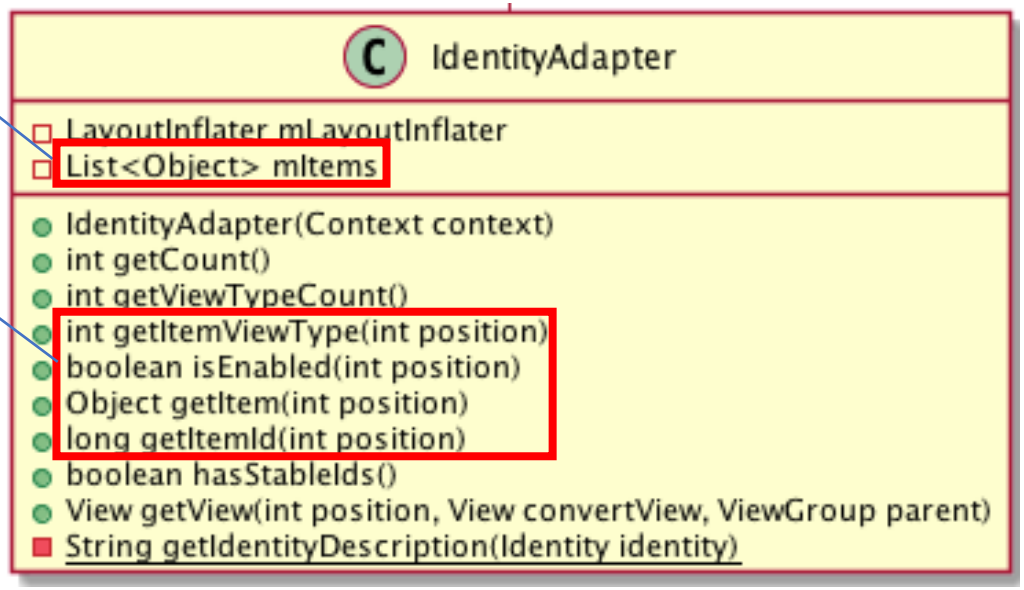
- extends Java's Collections framework
- contains a collection of objects (of other classes)
- has methods that maintaining relationships between objects in the collection
  - methods that manipulate the collection such as sort(), compare(), validate(), remove(), updates(), add(), delete() …
  - methods that give access to the objects such as get(index), next(), hasNext() ...

Structurer - Example



holds a collection

accessing to the collection




```java
@Override
public int getItemViewType(int position) {
    return (mItems.get(position) instanceof Account) ? 0 : 1;
}

@Override
public boolean isEnabled(int position) {
    return (mItems.get(position) instanceof IdentityContainer);
}

@Override
public Object getItem(int position) {
    return mItems.get(position);
}

@Override
public long getItemId(int position) {
    return position;
}
```

Interfacer

is a software element that

- transforms information or requests between distinct parts of the system
  - User interfacer interacts with the users of the system, e.g. GUI components
  - Internal interfacer exists between sub parts of the system, e.g. Data Management Tier
  - External interfacer communicates with external systems, e.g. API, extension points of the system

A ST class can be characterized by:

- Contains Java Swing, AWT, and other UI components
- Manage user interface and handle user interaction
  - In Android apps, this extends Activity classes
- Encapsulates functions or objects in the system by providing an Interface or an abstract class that can be used outside of the system
- If an interface is created but never implemented: may be this serves as an extension point for the system

Role stereotypes

Service Provider (SP)

- performs specific work
- offers services

'-er', '-or'; public static methods; might contains some logics to do specific tasks.

Information Holder (IH)

- knows/keeps information
- provides information

data encapsulation; get/set methods; private/internal methods

Interfacer (IF)

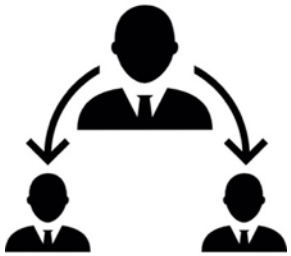
- transforms/converts information and requests btw SW layers

GUI-related; storefront; API; extension points

Controller (CT)

- makes decision
- control complex tasks

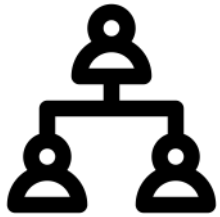
'controller', 'manager'; logic statements; knows IH, SP, CO



Coordinator (CO)

- delegates works
- forwards info/requests

no/simple logic; knows requester & requestee

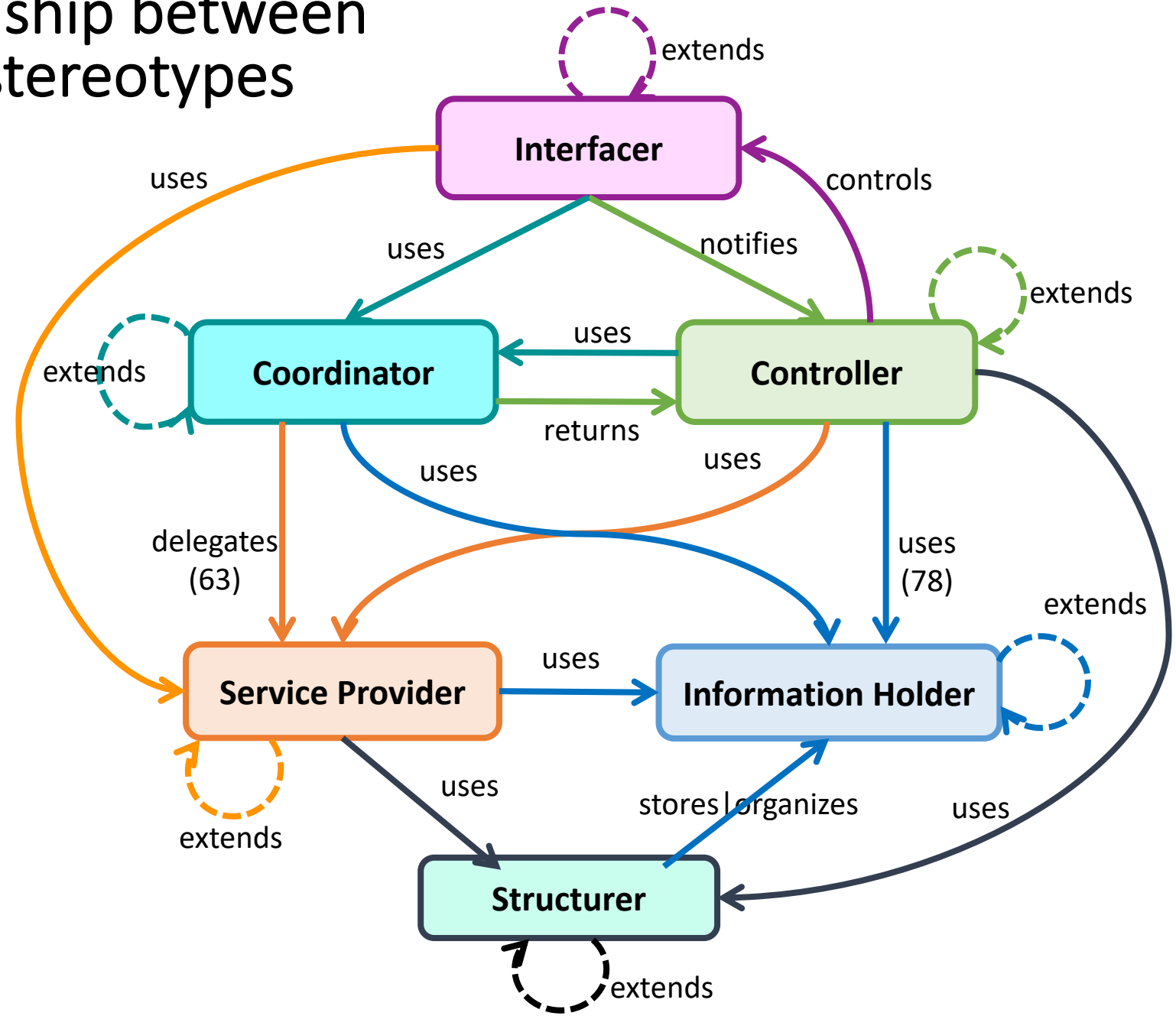


Structurer (ST)

- keeps/maintains relationship
- pool, collects, arranges objs

Collection; sort(); compare(); validate(); add(); remove(); …

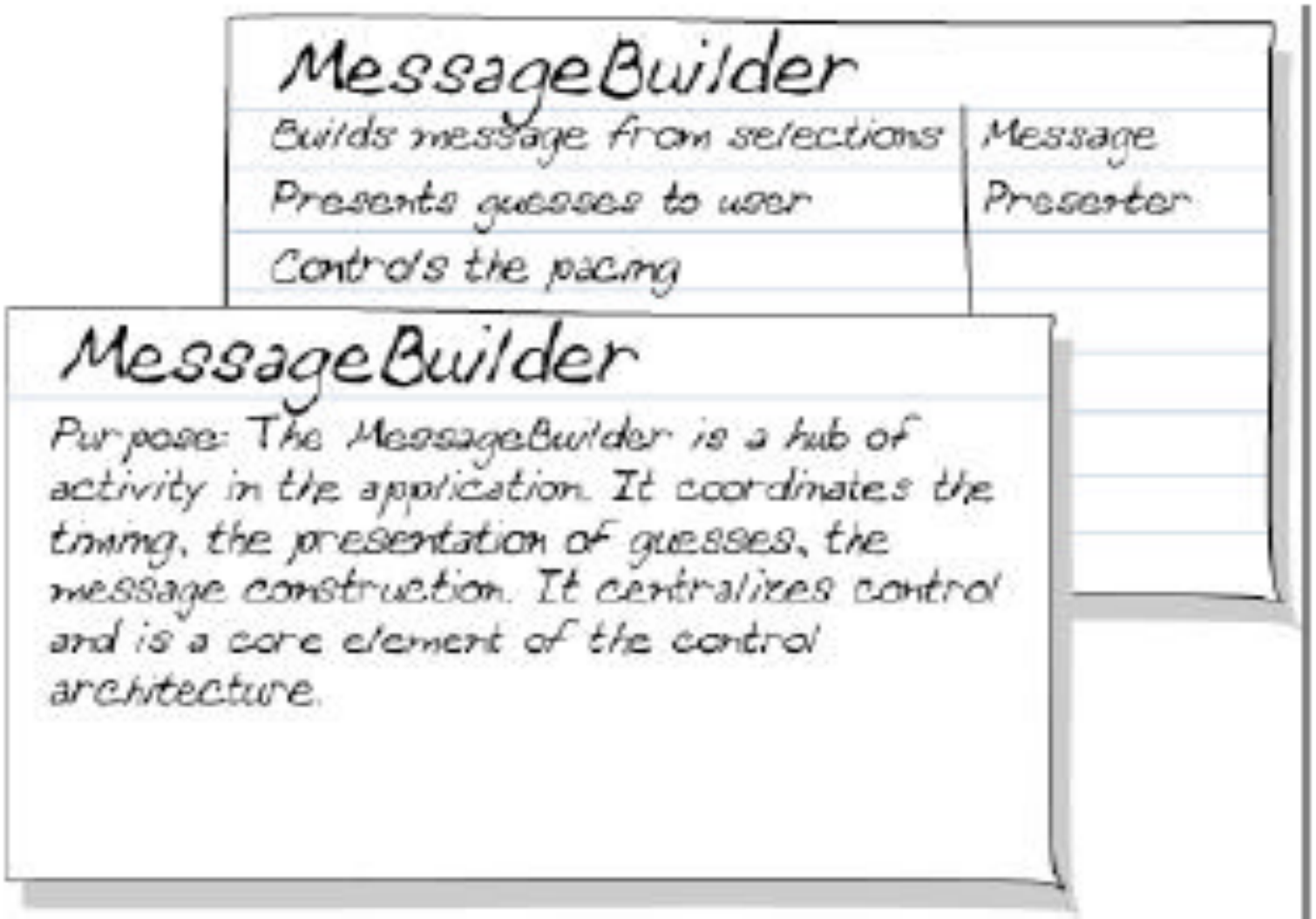
Relationship between role stereotypes



Analysing Responsibility and Collaborations of Objects using CRC Card

CRC Cards

Candidate, Responsibilities, Collaborators



CRC Card

Candidate:
Name of the object (component)

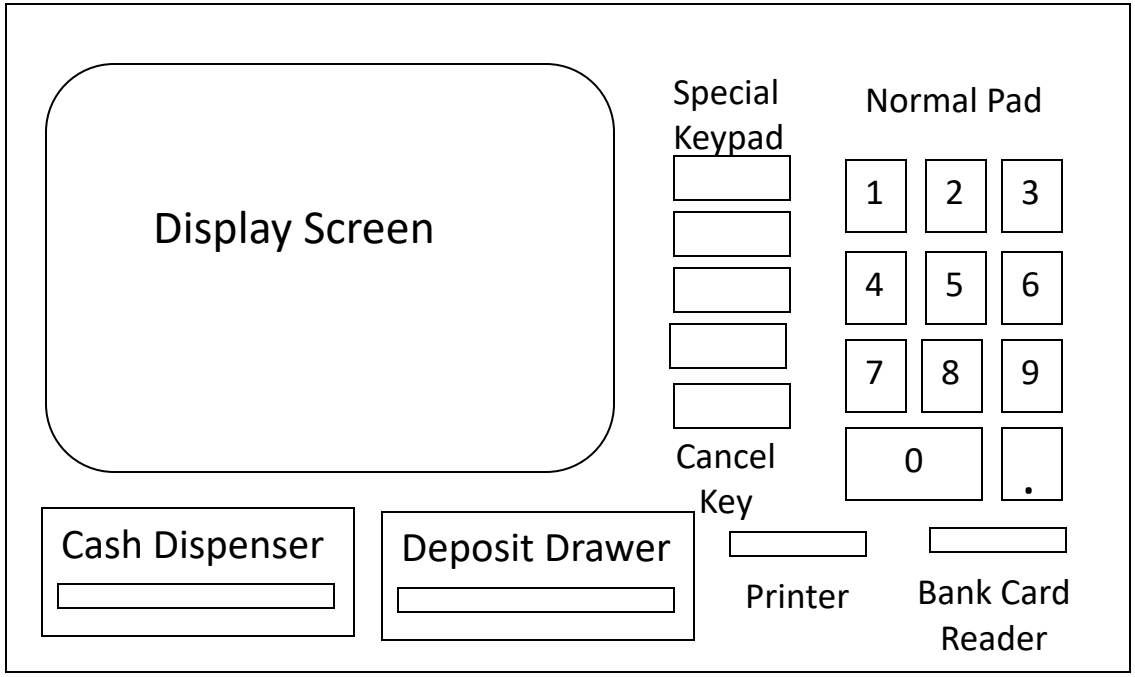
Responsibility

Message Builder	
Builds message from selection	Message
Presents guesses to users	Presenter
Controls the pacing	

Corresponsing **collaborator**

Message Builder
Purpose: The Message Builder is a hub of activity in the application. It coordinates the timing, the presentation of guesses, the message construction. It centralizes control and is a core element of the control architecture

Example: ATM system



Example: ATM system

An automated teller machine (ATM) is a machine through which bank customers can perform a number of financial transactions. The machine consists of a display screen, a bank card reader, input keys, a money dispenser slot, a deposit slot and a receipt printer. The main menu contains a list of the transactions that can be performed. These transactions include:

- deposit funds to an account
- withdraw funds from an account
- transfer funds from one account to the other
- query the balance of an account.

ATM class

The `ATM` class represents the teller machine. Its main operations are to create and initiate transactions. This class acts the following roles:

- a **Controller** role to both the `Financial Subsystem` and the `User Interface Subsystem`.

ATM Class	
Initiate Transaction	User Interface
Execute Transaction	User Interface

Financial Subsystem

- The `Financial Subsystem` implements the financial aspects of a customer's interaction with the `ATM`. Its main operations are to execute the following financial transactions; `deposit()`, `withdraw()`, `transfer()`, and `balance()` on customer accounts. There is one `Financial Subsystem` contract that must execute all the transactions. This subsystem acts as a **Service Provider** which provides banking services for `ATM Class`.

Financial Subsystem	
Deposit	ATM Class
Withdraw	ATM Class
Transfer	ATM Class
Balance	ATM Class

User Interface Subsystem

The `User Interface Subsystem` implements the interface between the ATM and the bank customer. The `User Interface Subsystem` has three responsibilities 1)To get numeric values from users. 2) Get users selection from menu. 3) To display messages and wait for events.

This subsystem acts as an **Interfacer** role to receive and transform requests from users to the system.

User Interface Subsystem	
Get numeric values	ATM Class, Financial Subsystem
Get users selection	ATM Class, Financial Subsystem
Display messages	ATM Class, Financial Subsystem

Does using role stereotype help in improving design quality?

Boeing Brewery Case (1)

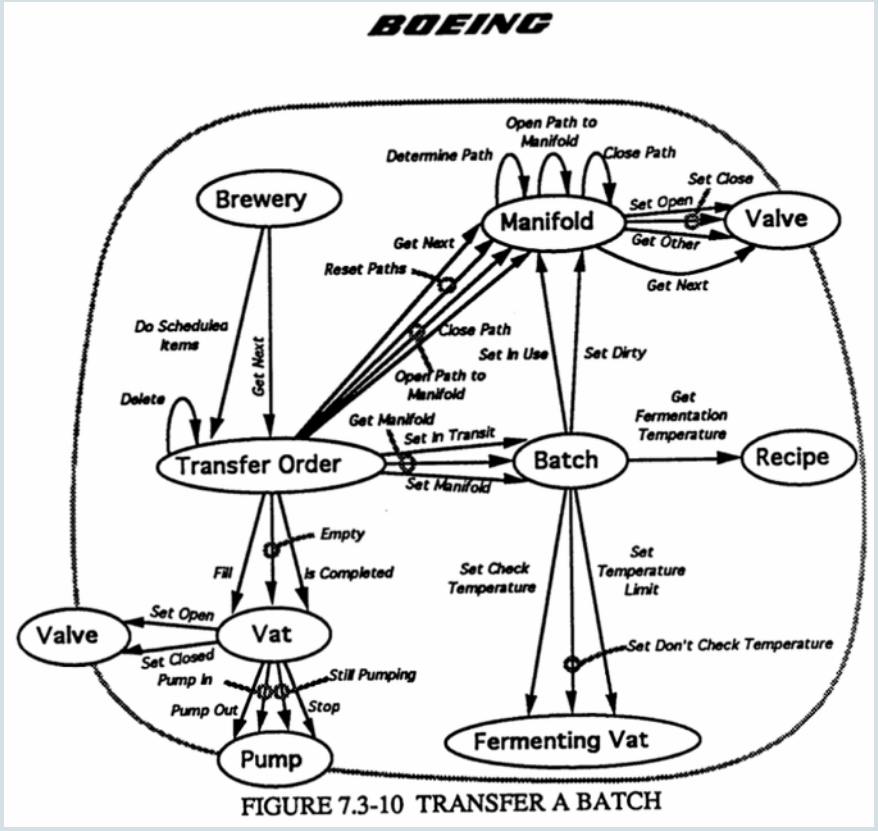


FIGURE 7.3-10 TRANSFER A BATCH

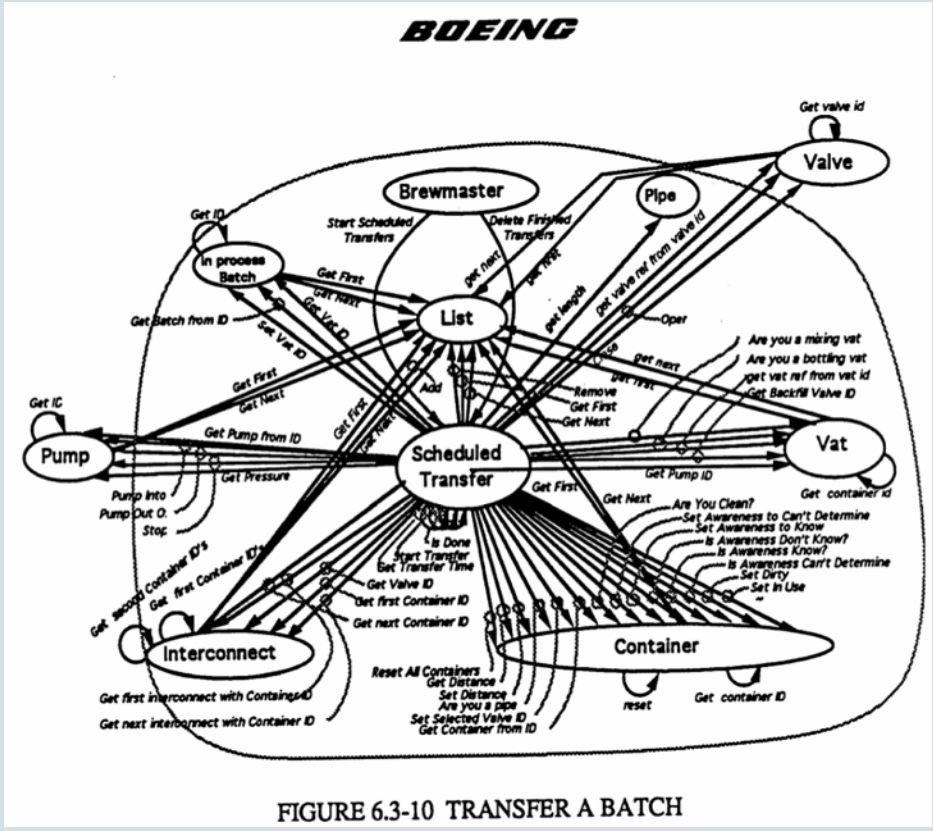


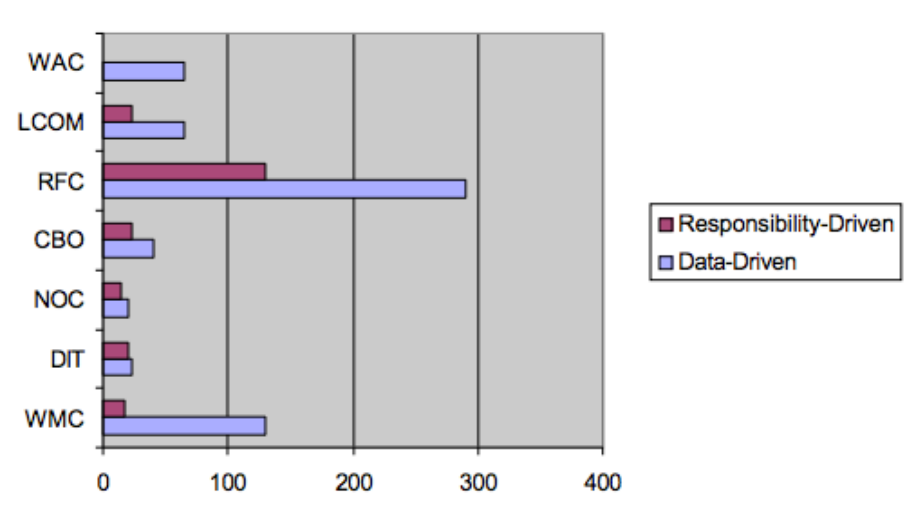
FIGURE 6.3-10 TRANSFER A BATCH

System 1: Responsibility-Focus

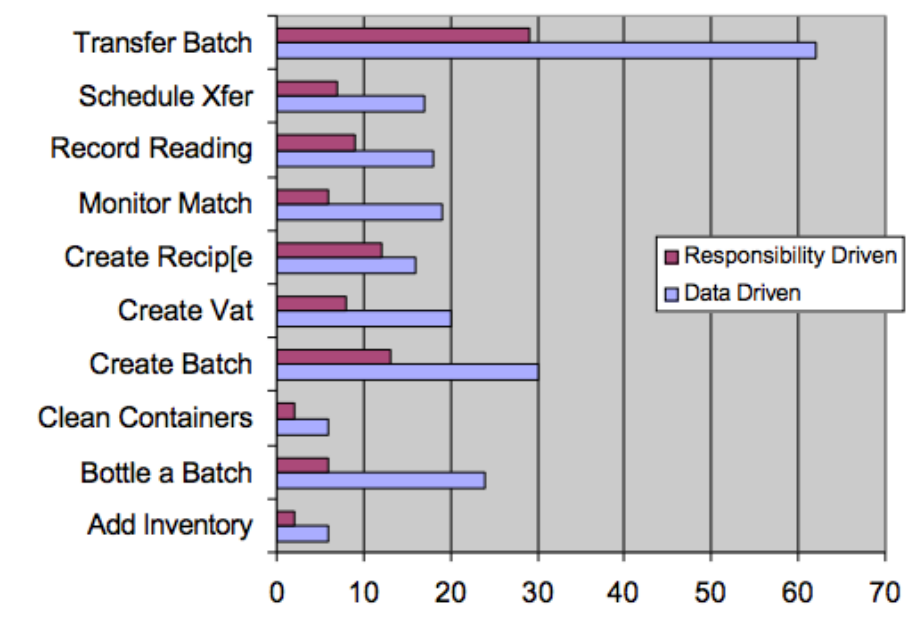
System 2: Data-Focus

Case description: R. Sharble and S. Cohen "The Object-Oriented Brewery: A Comparison of Two ObjectOriented Development Methods" Boeing Technical Report no. BC2-G4059, October, 1992.

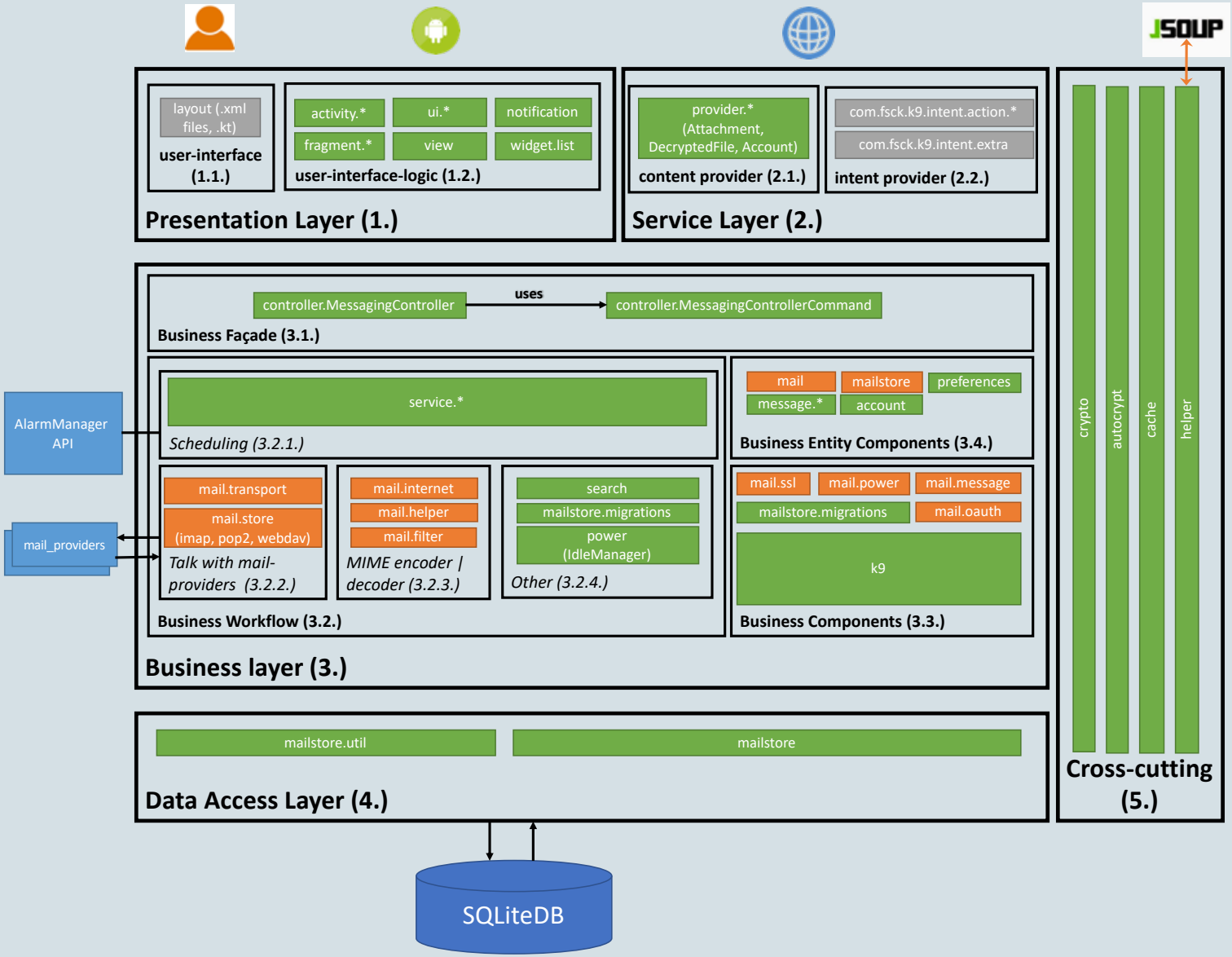
Boeing Brewery (2) – Design Quality Facts



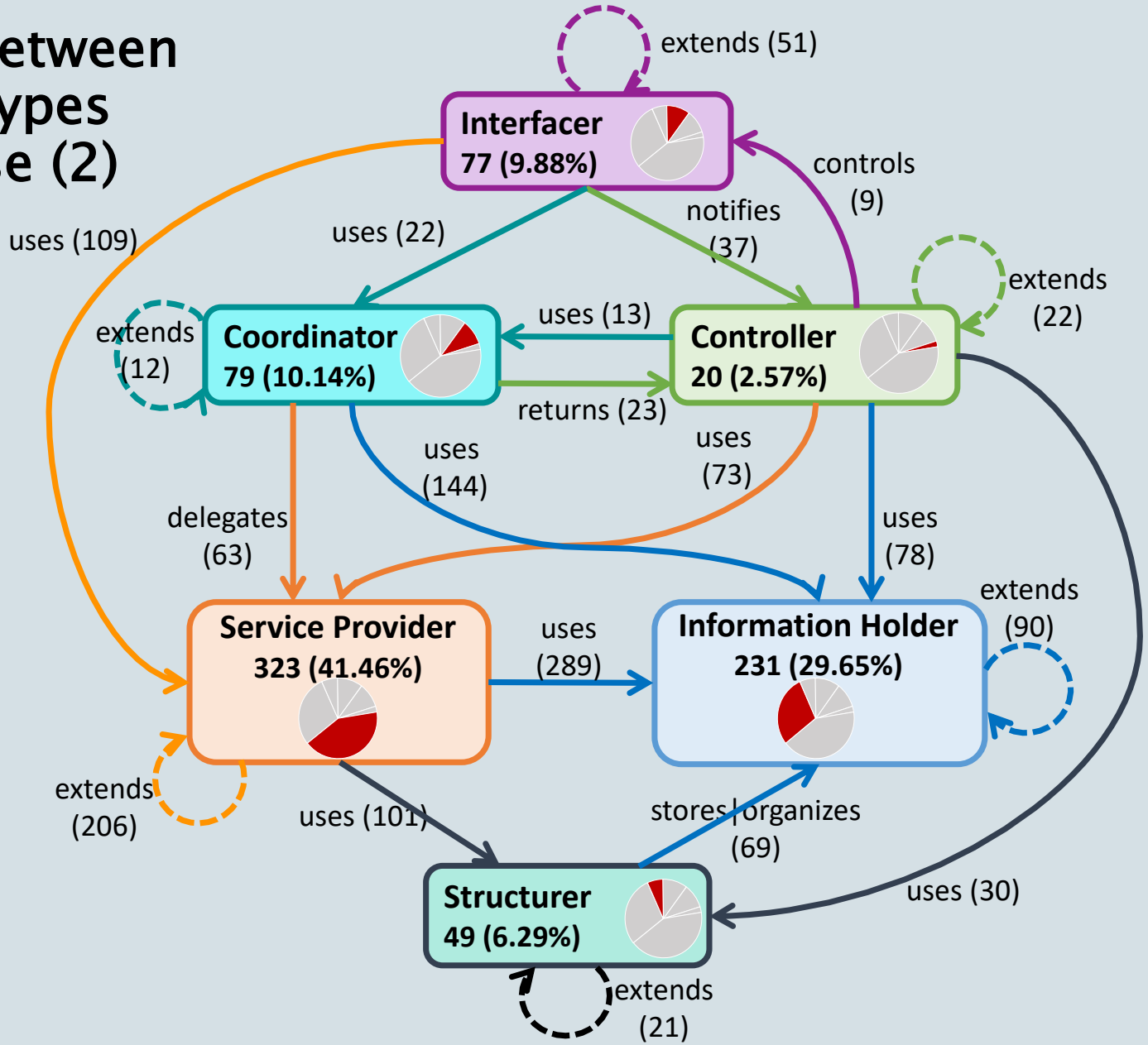
C-K metrics
Weighted Methods per Class (**WMC**)
Depth of Inheritance (**DIT**)
Number of Children (**NOC**)
Coupling between Objects (**CBO**)
Response For a Class (**RFC**)
Lack of Cohesion in Methods (**LCOM**)



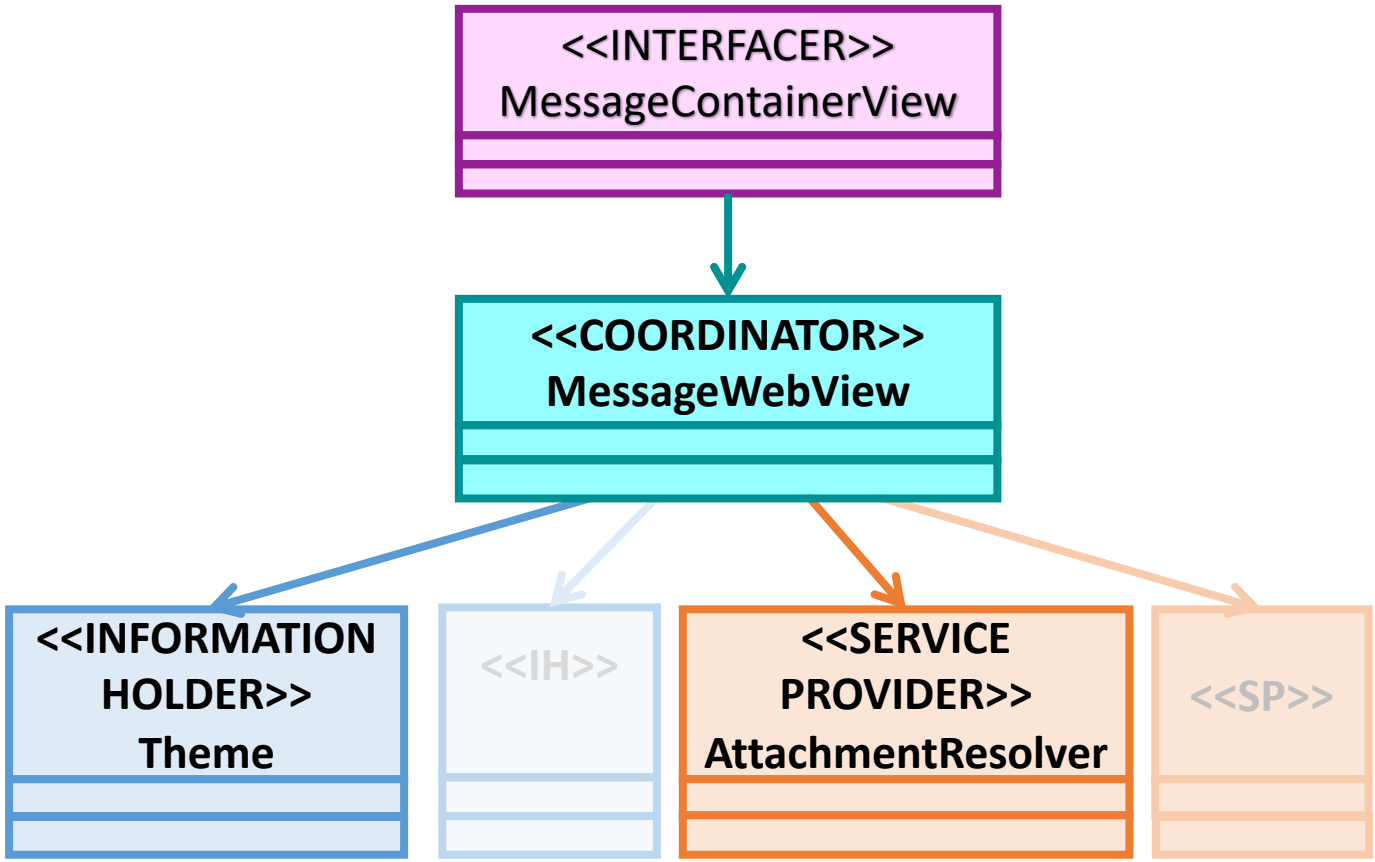
K9-Mail Case (1)



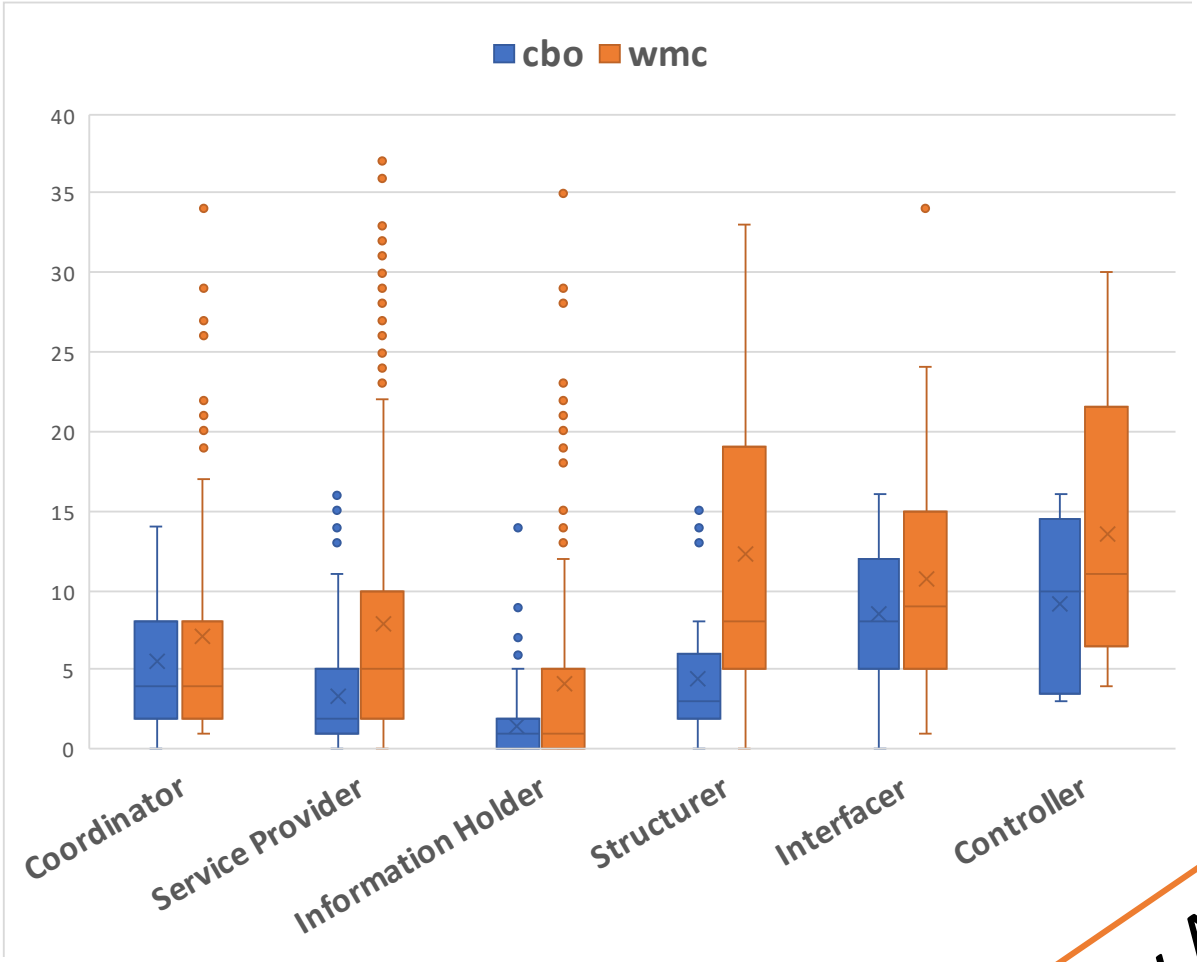
Relationship between role stereotypes K9-Mail Case (2)



K9-Mail Case (3)
Collaboration Patterns between Role Stereotypes



K9-Mail Case (4)
Design Metrics of Role Stereotypes



Quality Assurance

Summary Part II

- Having a concrete view on role/responsibility is vital to software design.
- Role stereotypes can be used as a tool for:
  - assigning roles to software elements (in design phase)
  - comprehending work breakdown and collaboration patterns in existing system
- Using CRC card when discussing/thinking of responsibilities and collaborations of an object (can be a component/subsystem/class)