How to write SAD, DDD

HARVEY NASH
The Power of Talent

08 Nov 2011

Tuan Nguyen Manh

Agenda

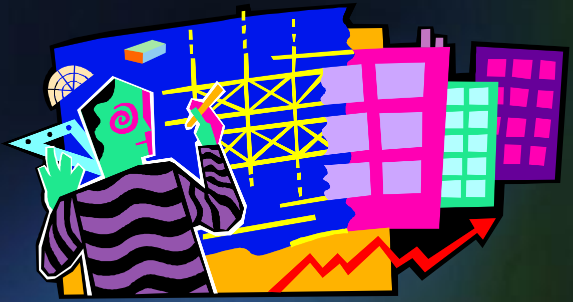
- Course Objectives
- Overview Software Architecture
- How to write SAD
- How to write DDD
- References
- Q&A

HARVEY NASH

Offshore Software Development Services

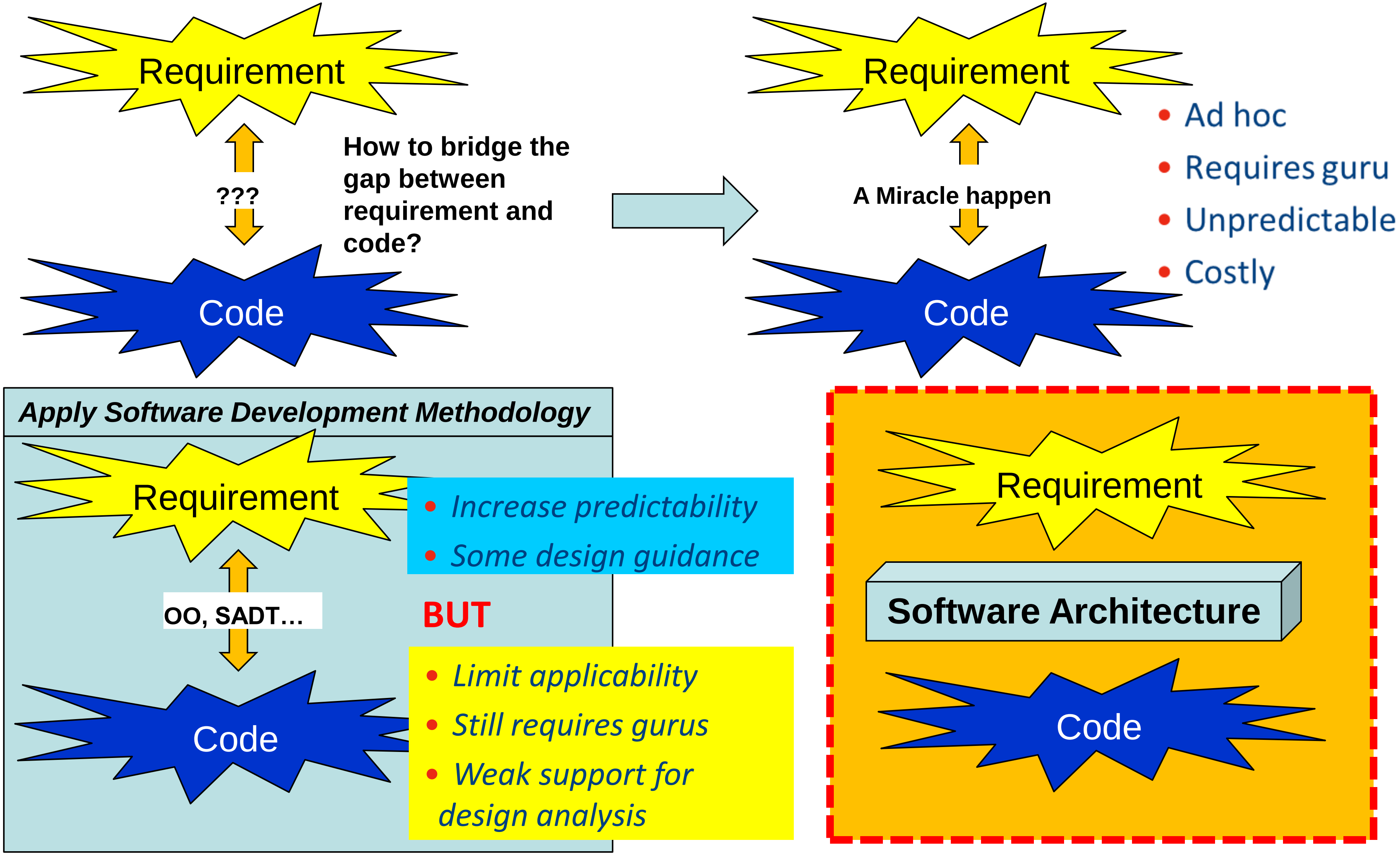
Course Objective

- To introduce software architectural and to discuss its importance
- To enhance the understandability of SAD (Software Architecture Document), DDD (Detail Design Document)
- How to write SAD, DDD that will be complete and unambiguous



Overview Software Architecture

Problem



Requirement

???

Code

How to bridge the gap between requirement and code?

Requirement

A Miracle happen

Code

- Ad hoc
- Requires guru
- Unpredictable
- Costly

Apply Software Development Methodology

Requirement

OO, SADT…

Code

- *Increase predictability*
- *Some design guidance*

BUT

- *Limit applicability*
- *Still requires gurus*
- *Weak support for design analysis*

Requirement

Software Architecture

Code

Example: The civil architecture

Can be built by one person

Requires

Minimal modeling

Simple process

Simple tools

Built most efficiently and timely by a team

Requires

Modeling

Well-defined process

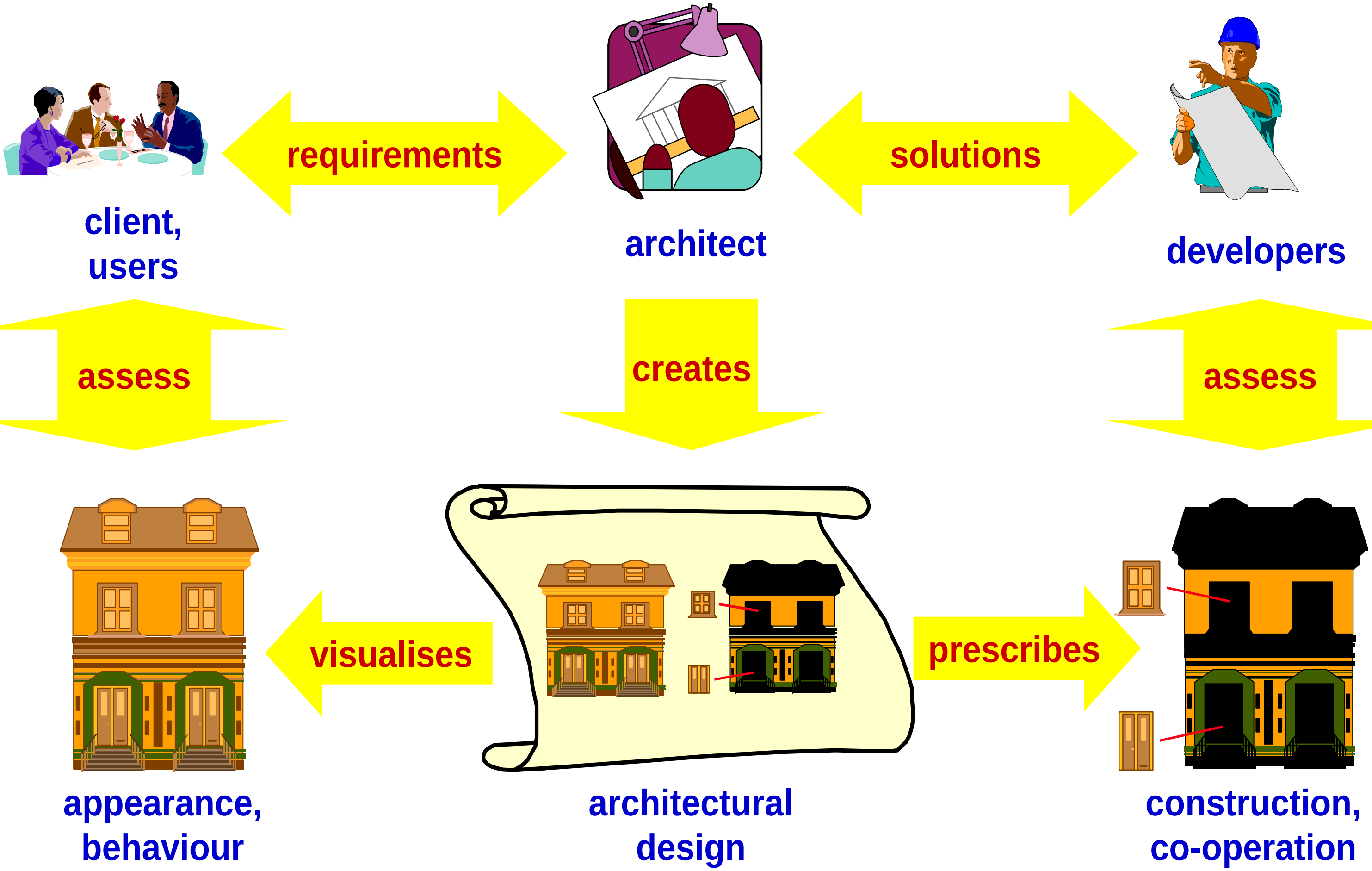
Power tools

A high rise

Differences

- Scale
- Process
- Cost
- Schedule
- Skills and development teams
- Materials and technologies
- Stakeholders
- Risks

The Role of the Architect



client,
users

requirements

architect

solutions

developers

assess

creates

assess

appearance,
behaviour

visualises

architectural
design

prescribes

construction,
co-operation

What is Software Architecture?

- **Boehm, et al, 1995:** A software architecture comprises:
  - A collection of software and system **components, connections, and constraints**
  - A collection of system **stakeholders' need** statements.
  - A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would **satisfy** the collection of system stakeholders' needs statements.
- **Eoin Woods**: Software architecture is the set of **design decisions** which, if made **incorrectly**, may cause your project to be **cancelled**.

Stakeholders and their Concerns

Management

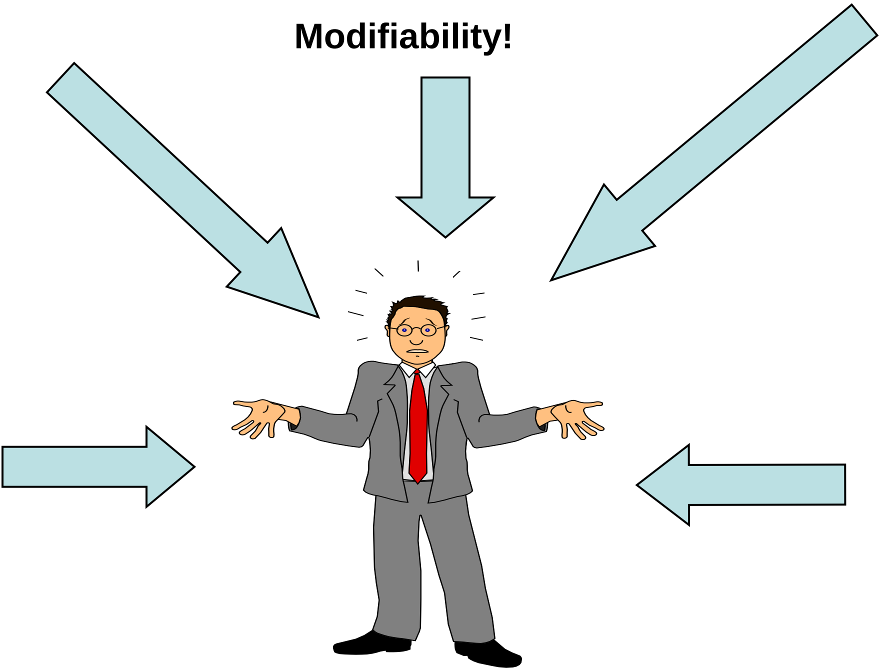
Low cost, keeping people employed!

Maintainers

Modifiability!

End-users

Behavior, performance, security, reliability!



Acquirers

Neat features, low cost, short time to market, parity with competing products!

Developers

Low cost, timely delivery, not change very often!

Software Architect

Why is Software Architecture Important?

- Architecture is the **vehicle** for **stakeholder communication**
- Architecture manifests the **earliest** set of design **decisions**
  - Earliest point at which the system to be built can be analyzed
  - Constraints on implementation
  - Setup organizational structure
  - Limit or enable quality attributes
- Architecture is a **transferable** abstraction of a system
  - Product lines share a common architecture
  - Allow for template-based development
  - Basis for large-scale reuse and training

The right architecture paves the way for system success.

The wrong architecture usually spells some form of disaster.

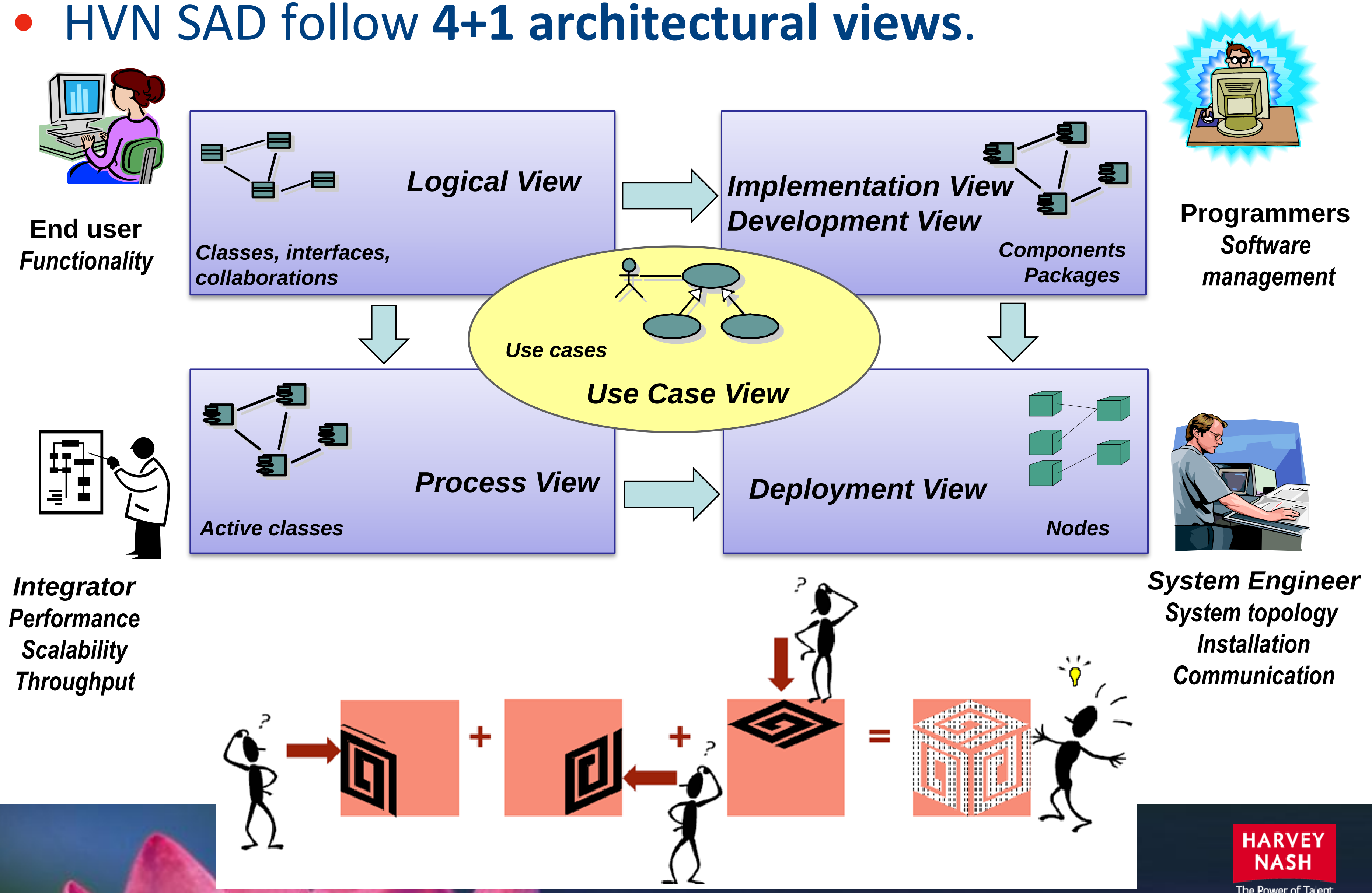
How to write SAD

The current situation SAD, DDD development at HVN

- **Advantage**
  - Available SAD, DDD template and some case study about SAD
    - HNVN_SD_002_01_Template_ArchitectureDesign
    - HNVN_SD_002_03_Template_DetailedDesign
    - HCM_IPS_PIMS_Software_Architecture_Design
    - Communisis Merlin - Software Architecture Document v0.3
    - HCM_SMI_DataMiner_ArchitectureDesign
  - Compliance with CMMI 3
    - **SAD**: Owner: Technical Leader – Approval: Technical Manager
    - **DDD**: Owner: Team Leader – Approval: Technical Leader
- **Disadvantage**
  - PM isn’t aware of effort and challenge during SAD, DDD development
    - Explanation and give some examples
    - Need to discuss TM if PM don’t aware or force on
  - Inconsistency between projects
    - **Tool**: Visio, VS2010, Enterprise Architect, StarUML…
      - StarUML
    - **Guideline**:
      - Training course

Software architecture document at HVN

- HVN SAD follow **4+1 architectural views.**



End user
Functionality

Logical View
Classes, interfaces, collaborations

Implementation View
Development View
Components
Packages

Programmers
Software management

Use cases
Use Case View

Process View
Active classes

Deployment View
Nodes

Integrator
Performance
Scalability
Throughput

System Engineer
System topology
Installation
Communication

Use case view

- Introduction
- Development of use case view
- Examples

Use case view - Introduction

Use case view

- **Capture system functionality** as seen by users
- Built in early stages of development
- Developed by **analysts and domain experts**
- **System behavior**, that is what functionality it must provide, is **documented** in a use case model

Use Case Model

- Illustrate the system's **intended functions** (use cases), its **surroundings** (actors), and **relationships** between the use cases and actors (use case diagrams).

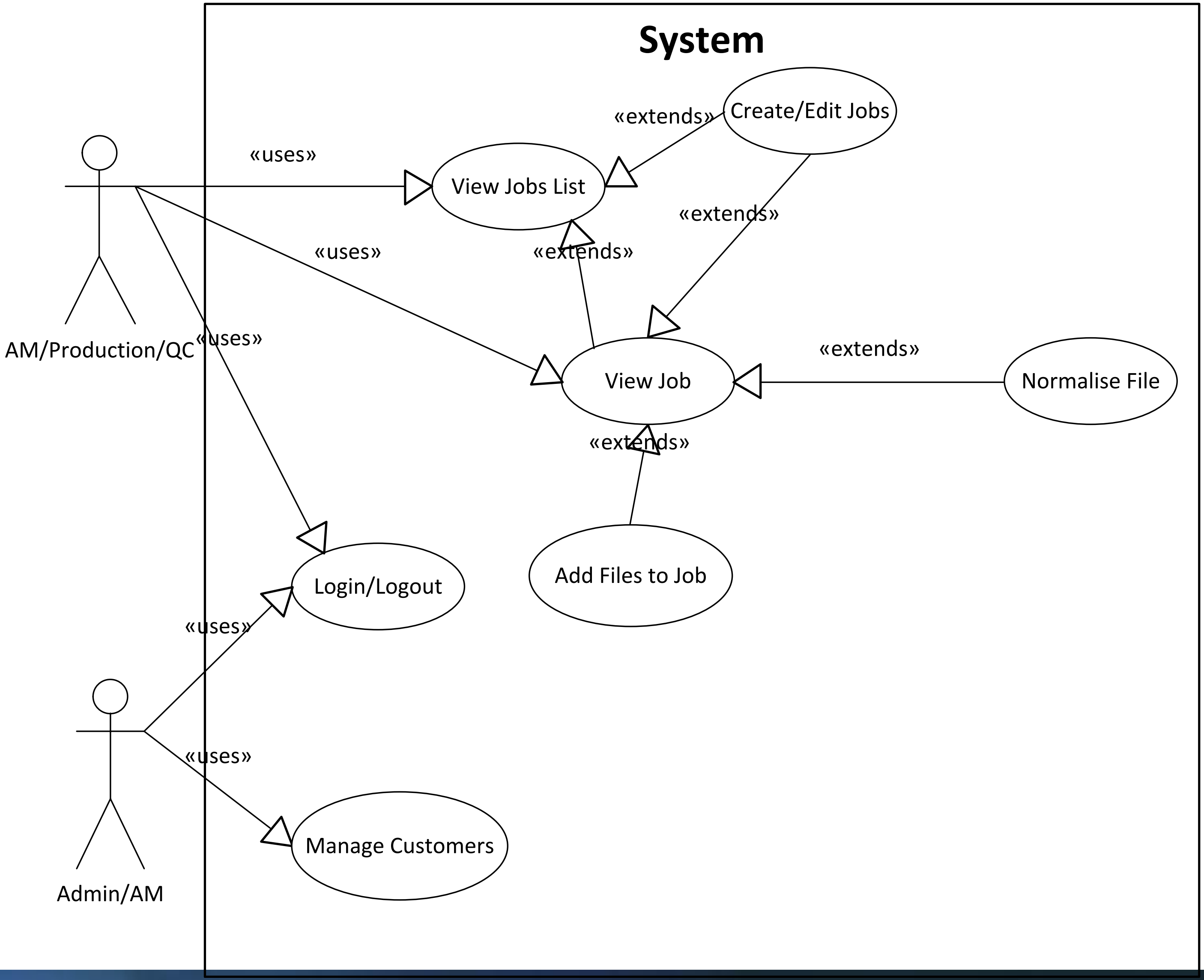
	Actor	**Use Case**
Actor	*Generalization*	*Communicate*
Use Case	*Communicate*	*Generalization* *Include* *Extend*

Use case view - Development of use case view

These are steps for development of use case view:

- Defining the system
- Finding Actors and Use Cases
- Use Case descriptions
- Defining relationships between Use Cases
- Verifying and validating the model

Use case view - Example



Logical view

- Introduction
- HVN Template
- Example

Logical view - Introduction

The purpose of the logical view is to **specify the functional requirements** of the system. The main artifact of the logical view is the design model:

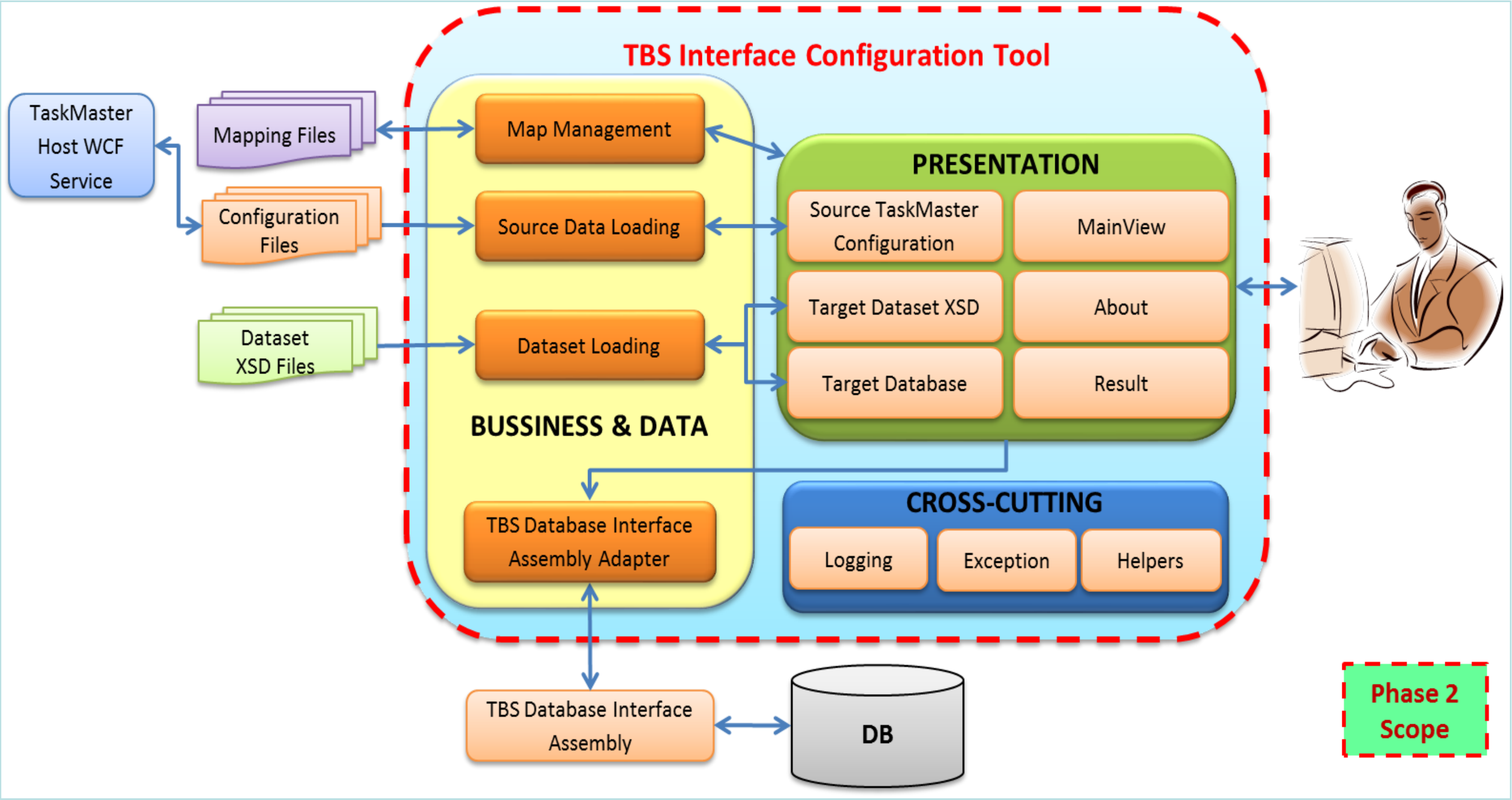
- Its **decomposition** into **subsystems and packages**
- For each significant package, its decomposition into classes and class utilities
- It should show any architecturally **significant classes** and describe their responsibilities, as well any key important **relationships, operations, and attributes**
- UML Diagrams used to represent the logical view include Class diagram, Communication diagram, Sequence diagram

DDD

Logical view – HVN Template

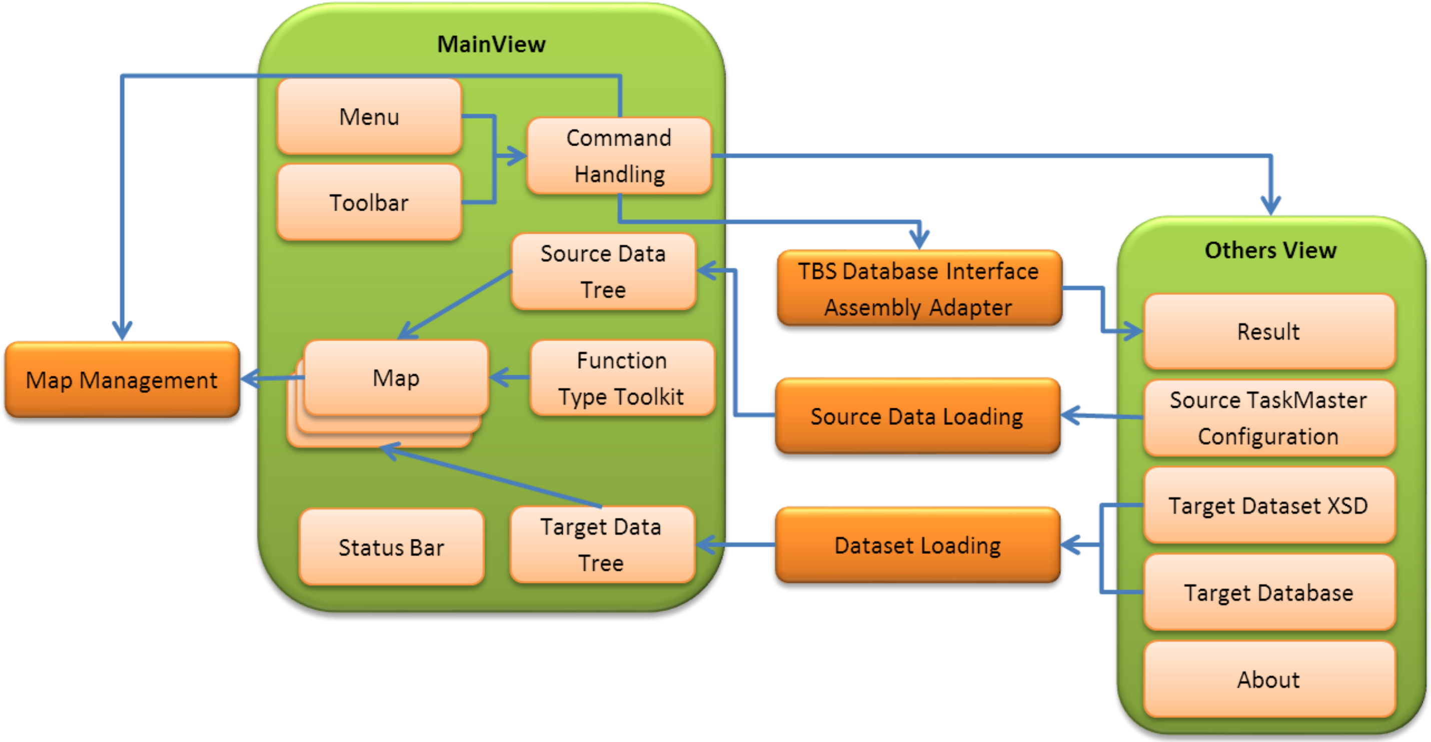
- Overview
- Description
- Common Sequence Diagram
- Architecturally Significant Design Packages

Logical view – HVN Template - Overview



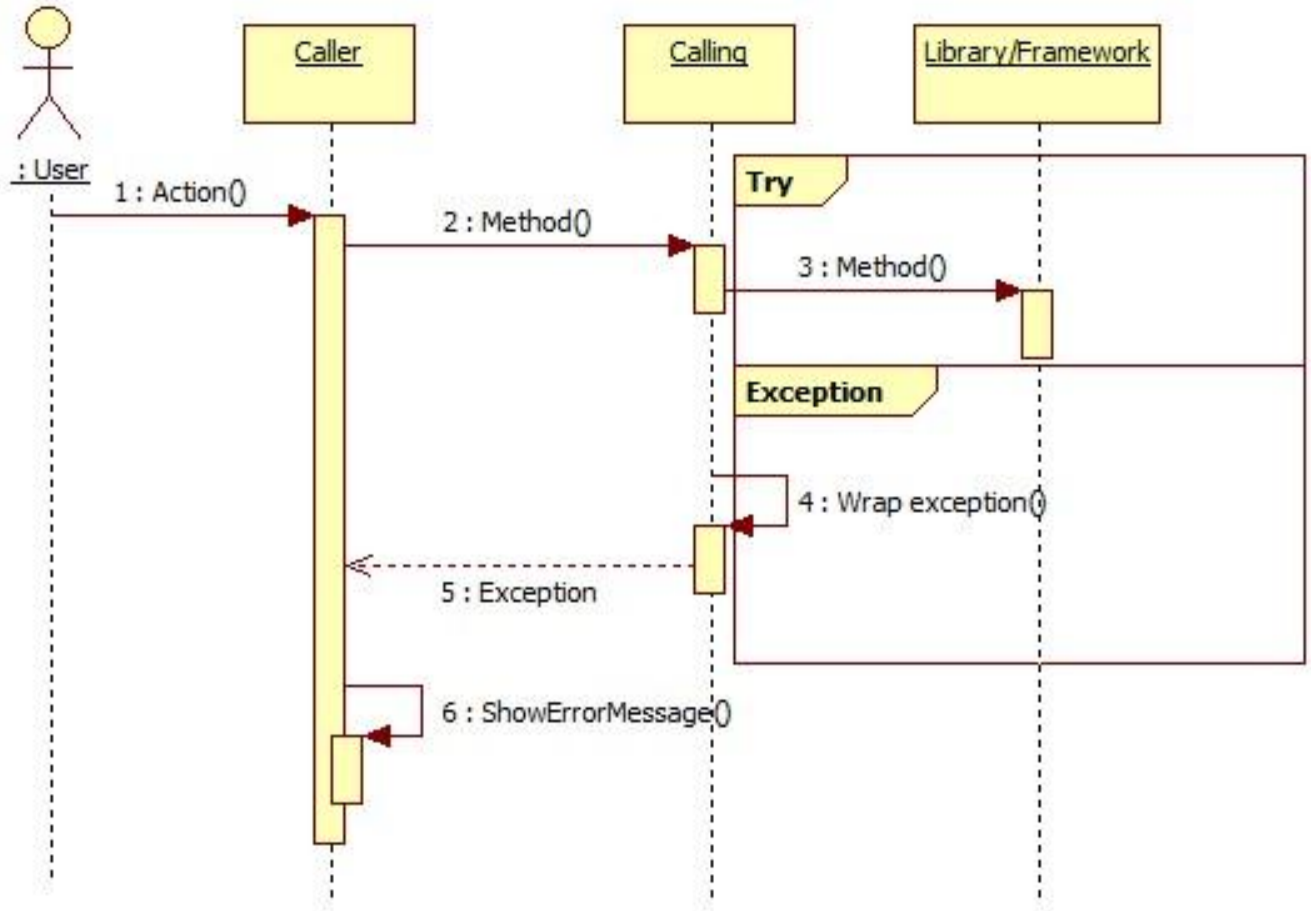
HARVEY NASH
The Power of Talent

Logical view – HVN Template - Description

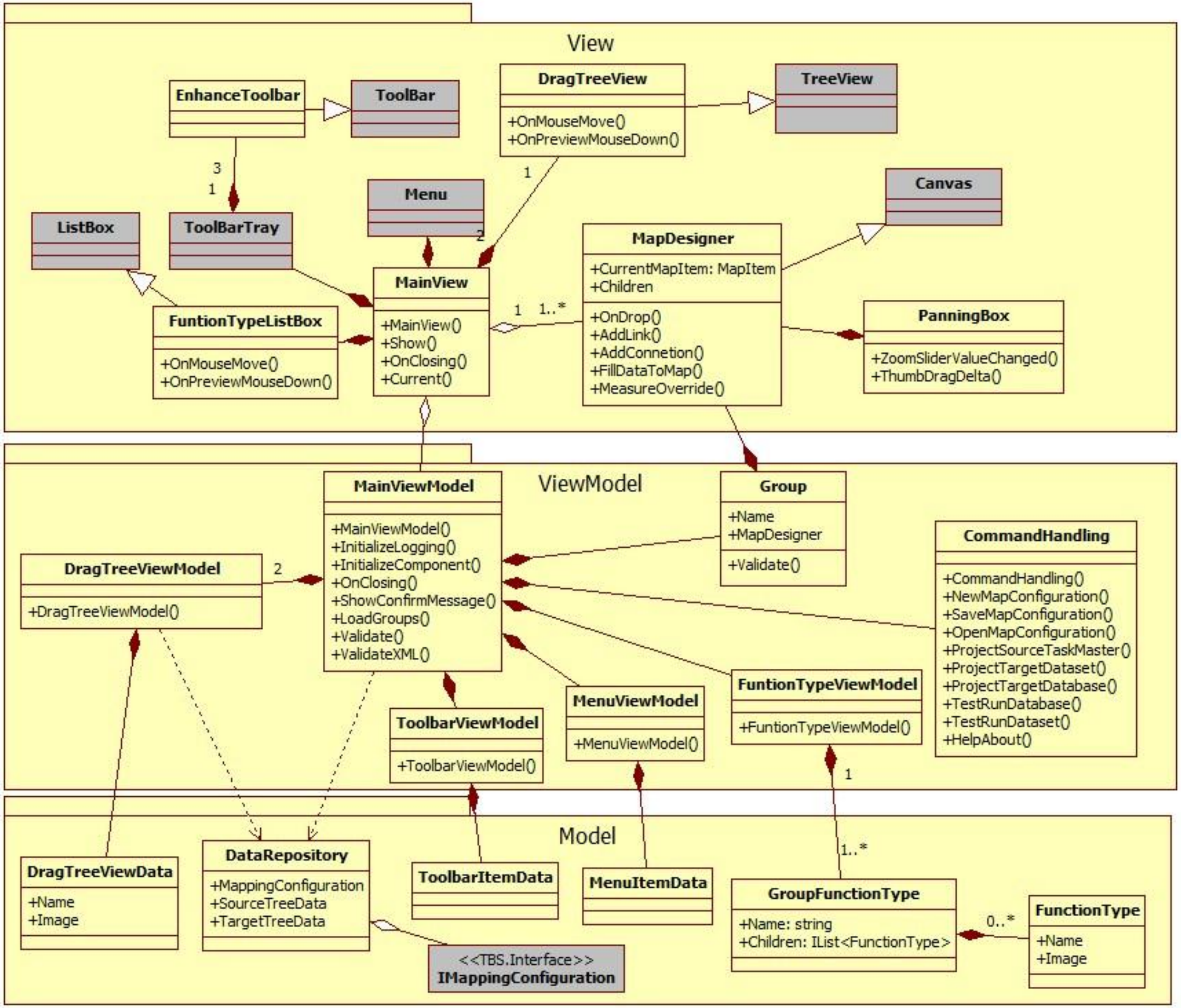


No	Item name	Description
1.	**Command Handling**	Receive actions from user via Menu and Toolbar and dispatch to appropriate components such: • **Map Management**: save/open/new mapping configuration file • **TBS Database Interface Assembly Adapter**: follow action to TBS Database Interface Assembly DLL • **Others view**: open appropriate views

Logical view – HVN Template - Common Sequence Diagram



Logical view – HVN Template - Design Packages



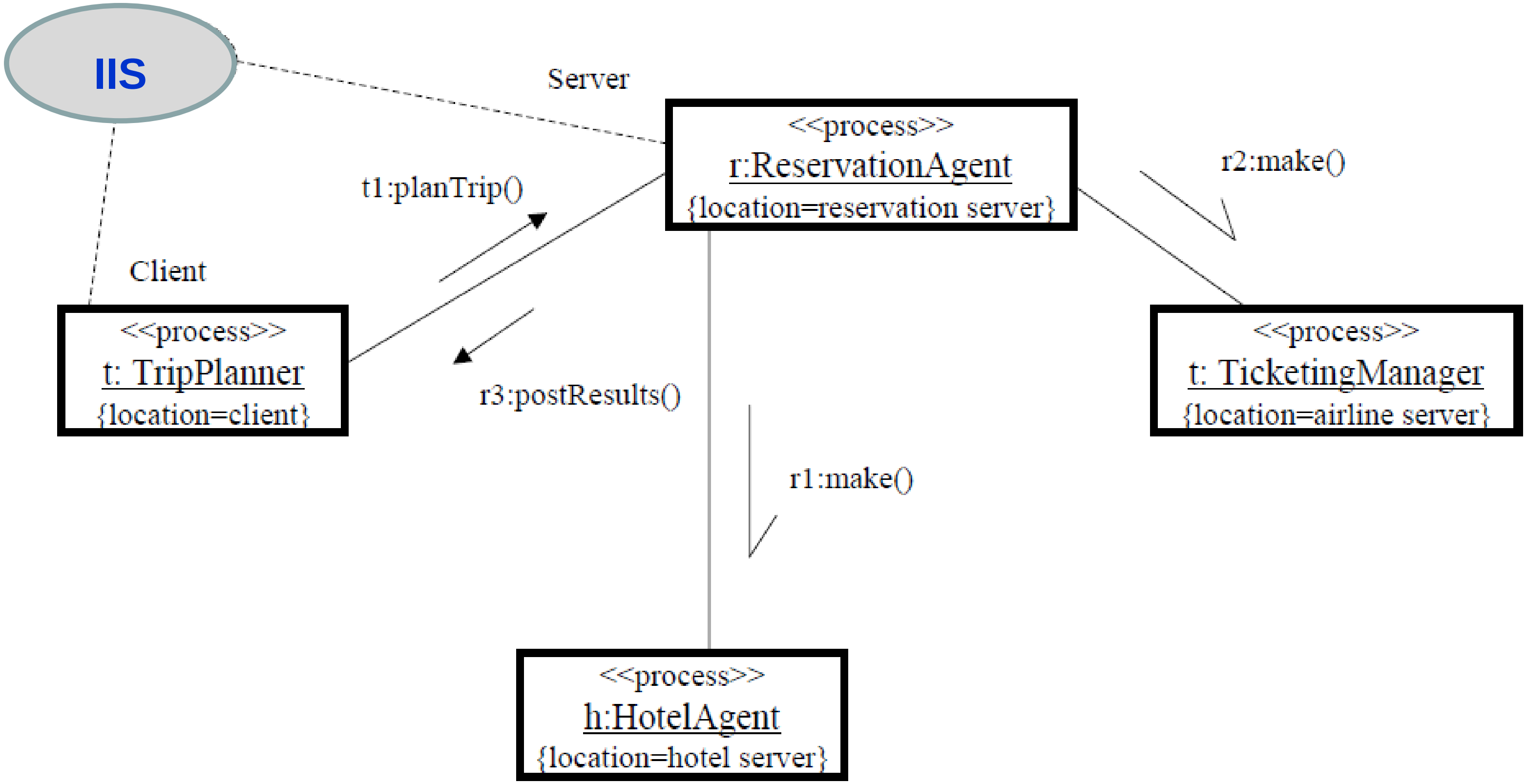
Process View

- Introduction
- Example

Process View - Introduction

- The process model deals with the **dynamic aspect of the system**, explains the system processes and how they communicate, and focuses on the runtime behavior of the system.
- Addresses issues:
  - Concurrency and parallelism (e.g. synchronization, deadlocks, time, events..)
  - System startup and shutdown.
  - Performance, scalability, and throughput of the system.
- Consists of the **processes** and **threads** that form the system's **concurrency**, and **synchronization** mechanisms, as well as their interactions.
  - **Process**: A heavyweight flow of control that can execute independently and concurrently with other processes.
  - **Threads**: A lightweight flow that can execute independently and concurrently with other threads in the same process.

Process View - Example



IIS

Server

Client

<<process>>
r:ReservationAgent
{location=reservation server}

t1:planTrip()

r2:make()

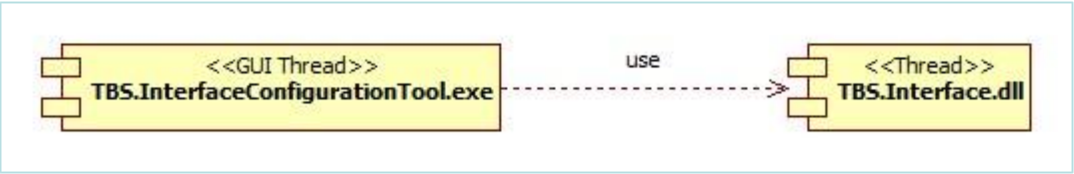
<<process>>
t: TripPlanner
{location=client}

r3:postResults()

<<process>>
t: TicketingManager
{location=airline server}

r1:make()

<<process>>
h:HotelAgent
{location=hotel server}



<<GUI Thread>>
TBS.InterfaceConfigurationTool.exe

use

<<Thread>>
TBS.Interface.dll

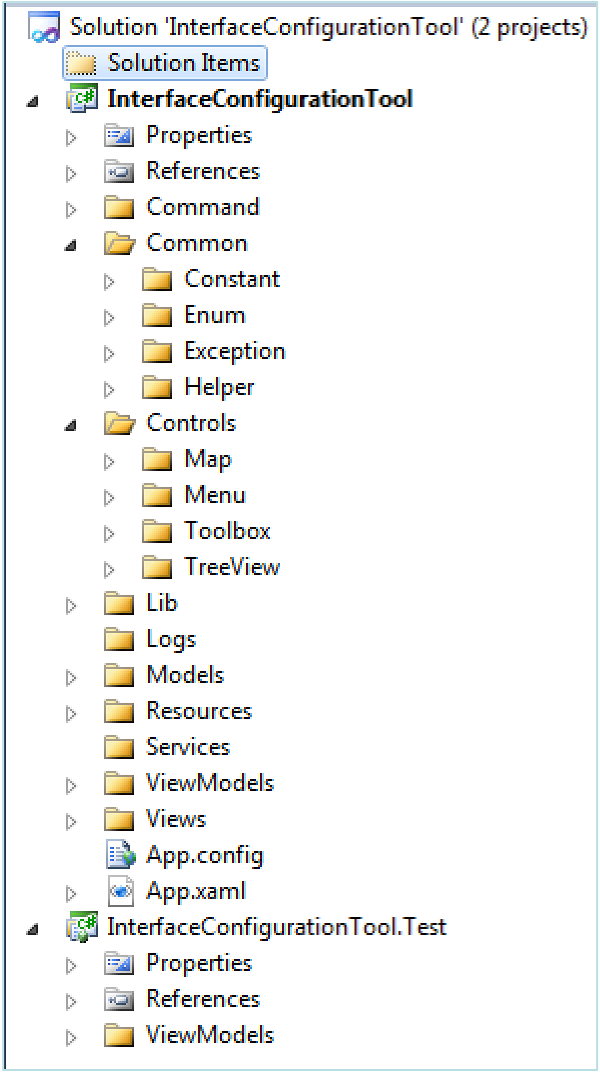
Development View / Implementation View

- Introduction
- Example

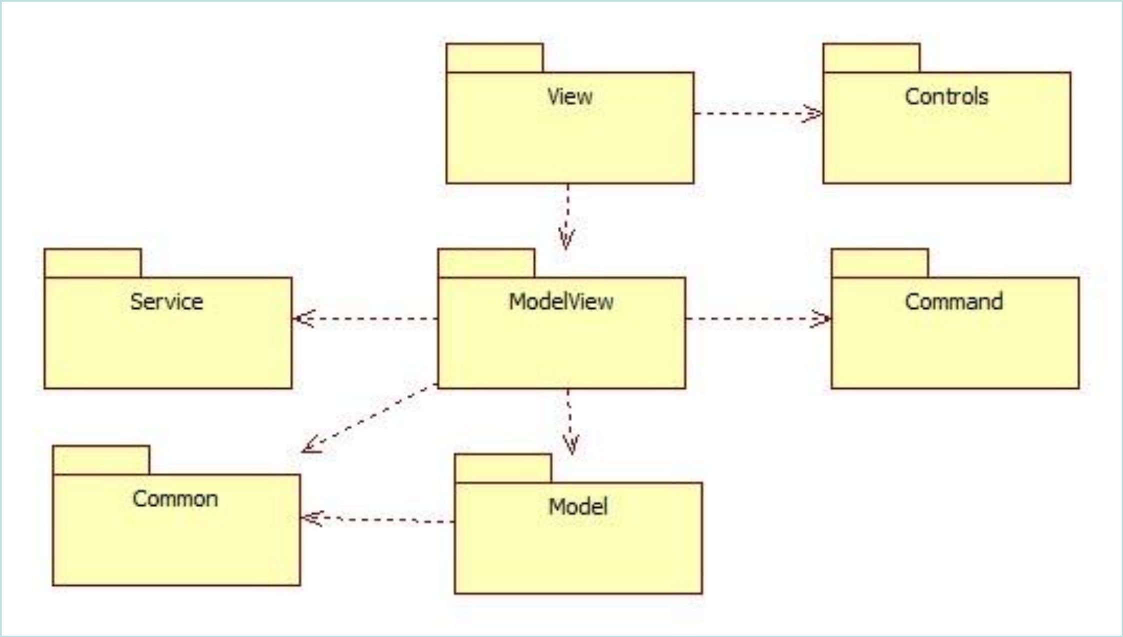
Development View - Introduction

- Describes the ***organization of static software modules*** (source code, data files, executable, documentation etc.)
- Describes a system from a **programmers perspective** and is concerned with software management
- Represents a **package diagram** that depicts how a system is spit up into logical groupings by showing dependencies among these groupings.

Development View - Example



```
Solution 'InterfaceConfigurationTool' (2 projects)
  Solution Items
  InterfaceConfigurationTool
    Properties
    References
    Command
    Common
      Constant
      Enum
      Exception
      Helper
    Controls
      Map
      Menu
      Toolbox
      TreeView
    Lib
    Logs
    Models
    Resources
    Services
    ViewModels
    Views
    App.config
    App.xaml
  InterfaceConfigurationTool.Test
    Properties
    References
    ViewModels
```



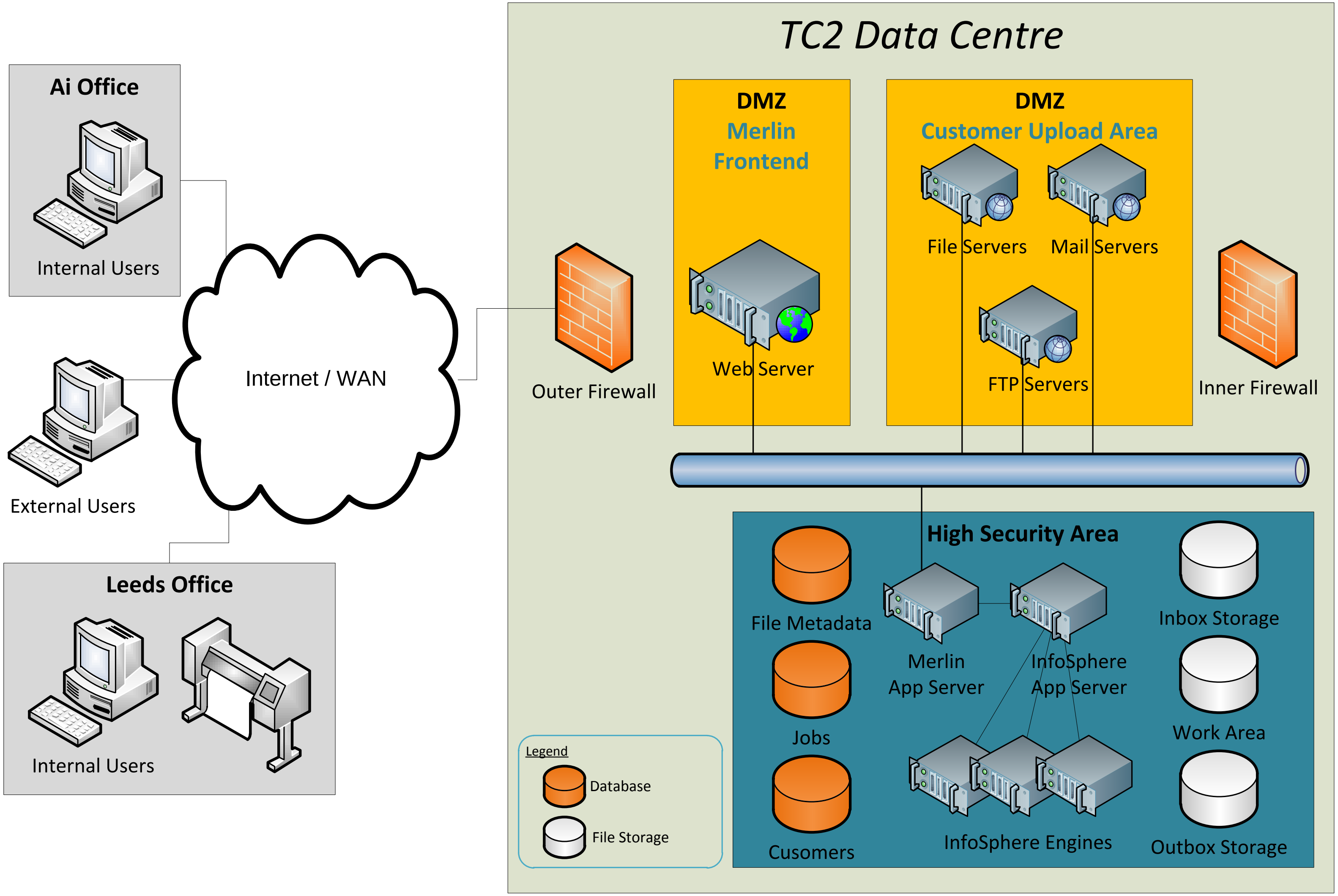
Physical View / Deployment View

- Introduction
- Example

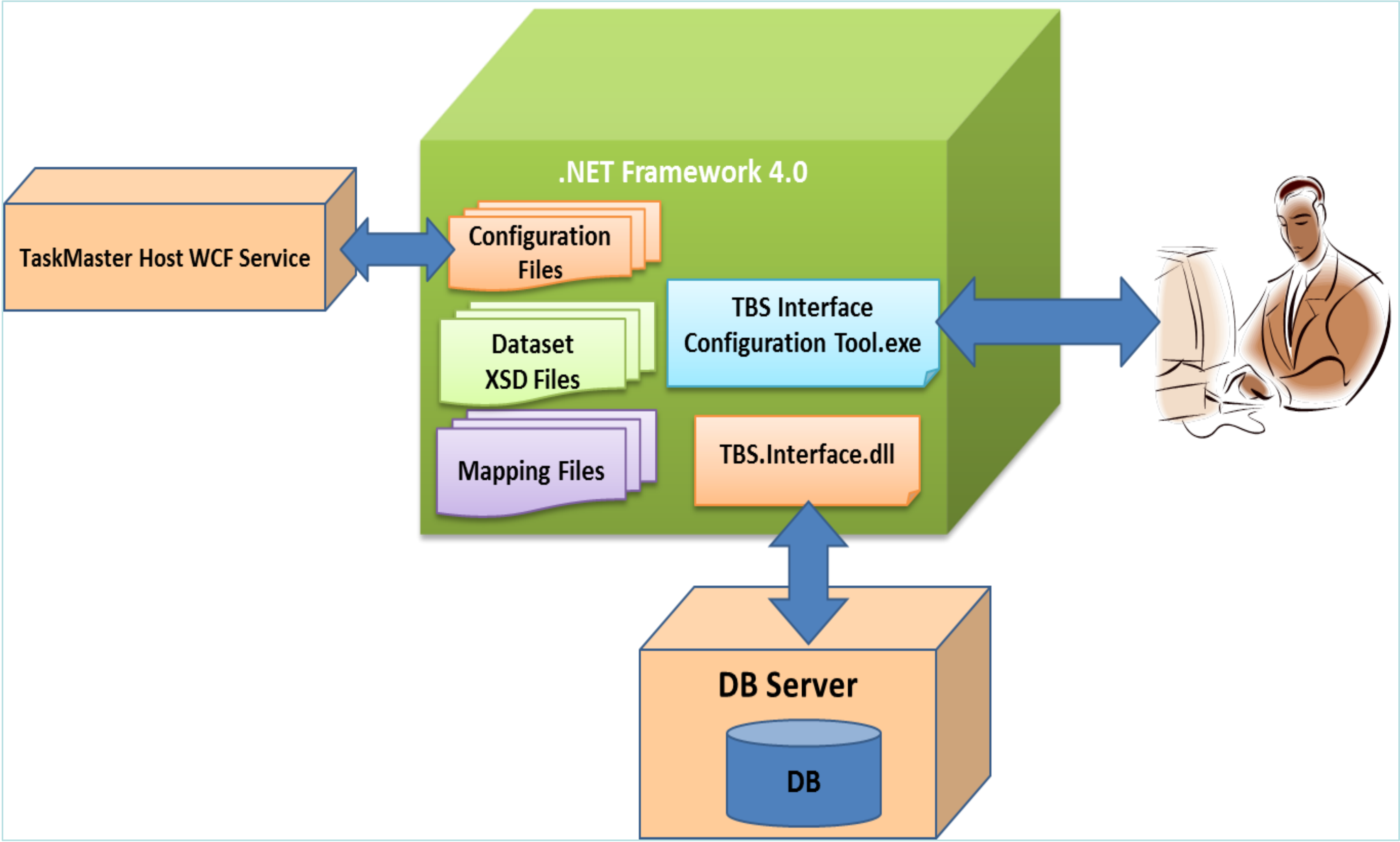
Physical View / Deployment View - Introduction

- Describes one or more **physical network** (hardware) configurations on which the software is deployed and run.
- At a minimum for each configuration it should indicate the **physical nodes** (computers, CPUs) that execute the software, and their interconnections (bus, LAN, point-to-point, and so on.).
- Describes the system from a **system engineer's point-of-view**. It is concerned with the **topology** of software components on the physical layer, as well as communication between these components.

Physical View / Deployment View - Example



Physical View / Deployment View – Example(cont)



How many views?

- **Simplified** models to fit the context
- **Not** all systems require **all views**:
  - **Single processor**: drop deployment view
  - **Single process**: drop process view
- Very Small program: drop implementation view
- Adding views:
  - Data view
  - Security view

How to write DDD

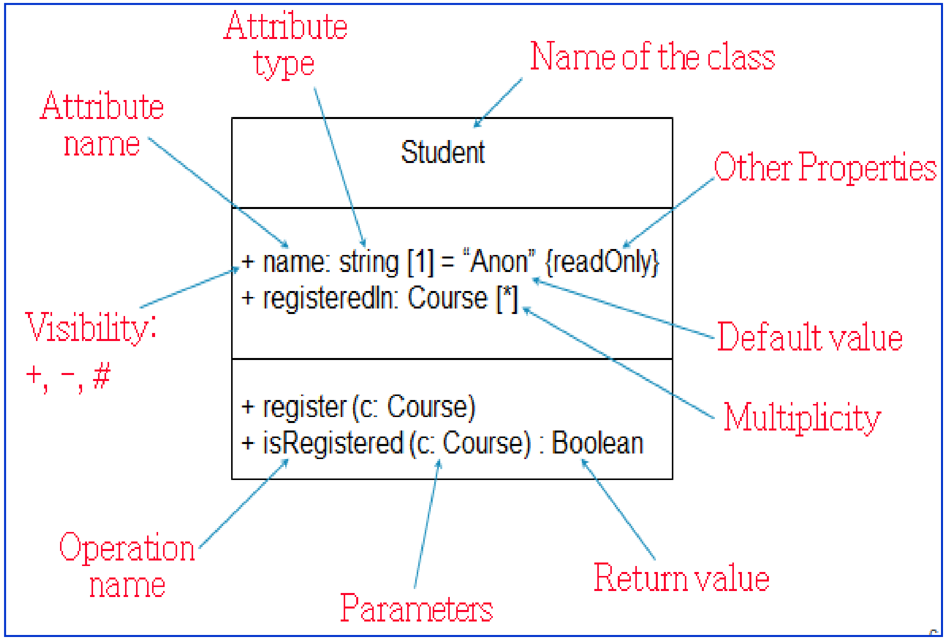
HVN’s DDD template customization

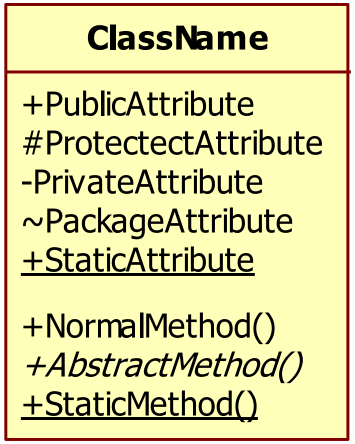
THE SPECIFIED USE CASE

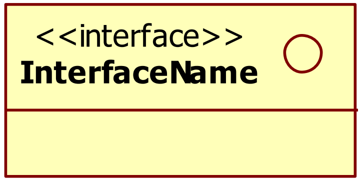
- Summary
- Use case diagram: can be ignore
  - It was included in UC document if UC existed
- Activity Diagram: **Optional**
- **Sequence Diagram**
- Screens Design: can be ignore
  - It was included in UC document if UC existed
- **Class Diagram**
- **Class Specification (s)**

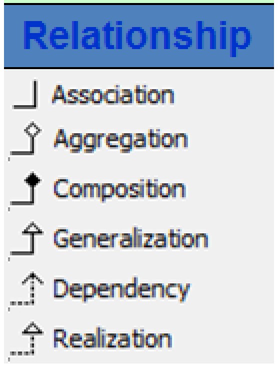
Class diagram

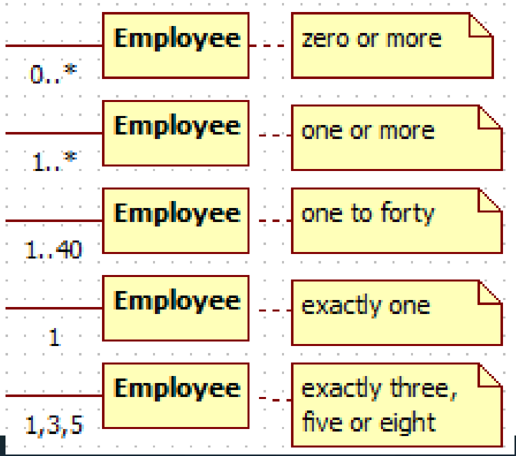
- **Class diagrams**, a type of static structure diagram that describe the structure of a system by showing the system’s ***class***, their ***attributes***, and ***relationships*** between





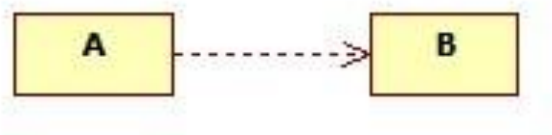


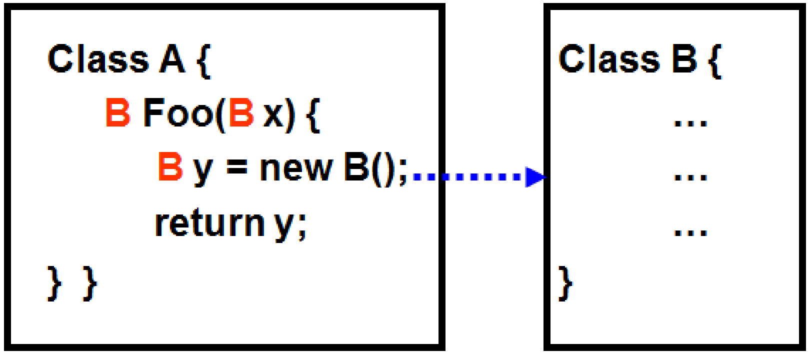




Class diagram - Dependency Relationship

- Change in specification of one class **can change the other class**. This can happen when one class is using another class.
- Method in Class A temporarily "uses a" object of type Class B
- Change in Class B may affect class A
- Dependence may be caused by
  - Local variable
  - Parameter
  - Return value
- Example



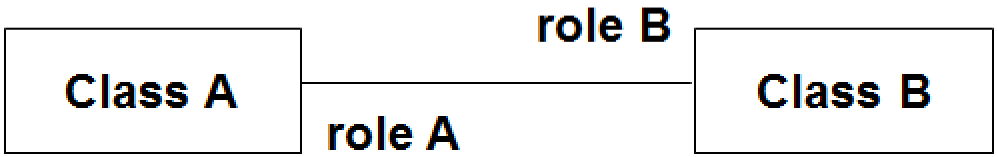


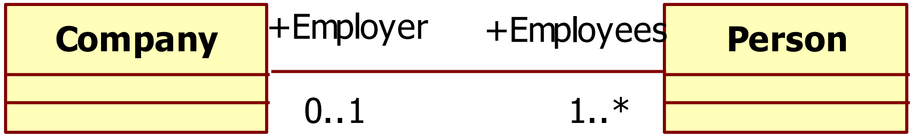
```
Class A {
    B Foo(B x) {
        B y = new B();
        return y;
} }
```

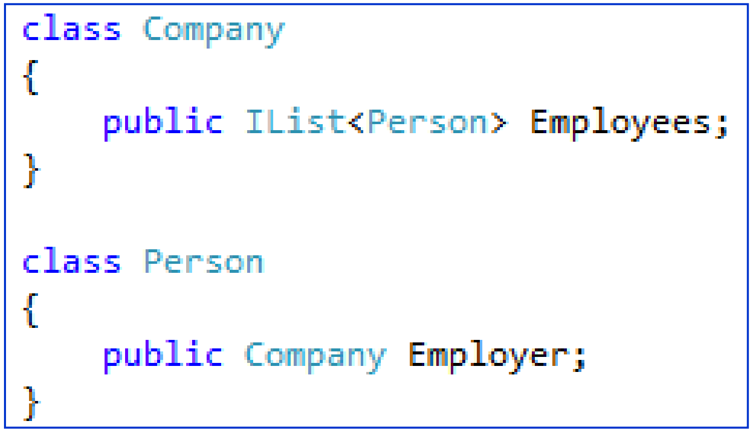
```
Class B {
        ...
        ...
        ...
}
```

Class diagram – Association Relationship

- Relationships between instances (objects) of classes
- Conceptual:
  - Associations can have two roles (bi-directional):
    - Roles have multiplicity (e.g., cardinality, constraints)
  - To restrict navigation to one direction only, an arrowhead is used to indicate the navigation direction



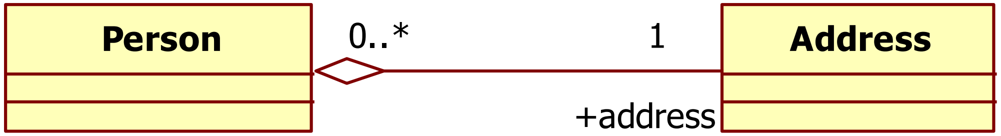


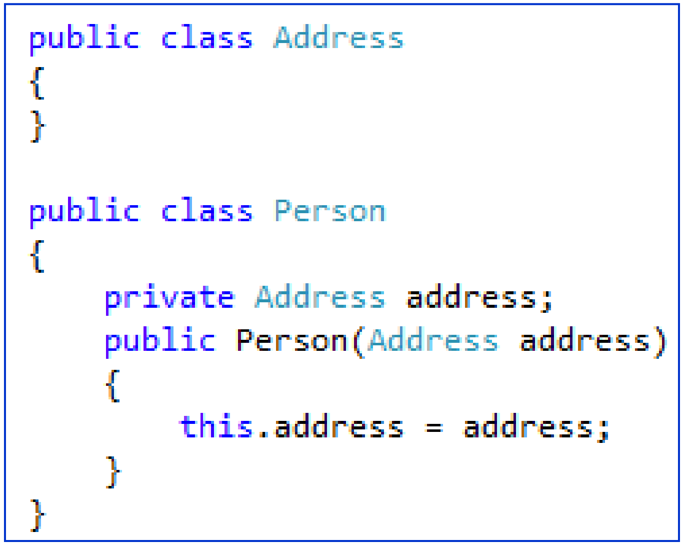


```
class Company
{
    public IList<Person> Employees;
}

class Person
{
    public Company Employer;
}
```

Class diagram – Aggregation Relationship

- A specialized form of ASSOCIATION in which a whole is related to its part(s).
- Is known as a “**part of**” or the “**has a**”
- Three ways to think about aggregations:
  - Whole-Parts
  - Container-Contents
  - Group-Members

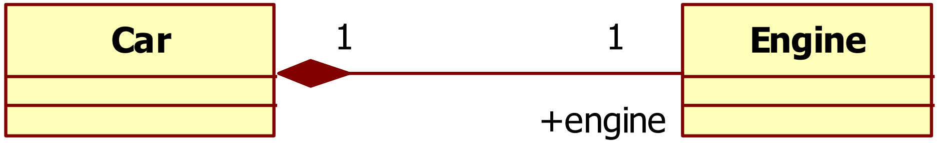


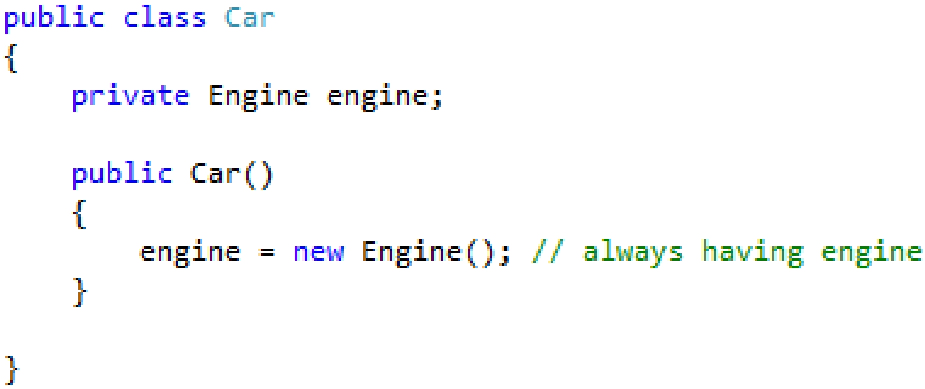


```
public class Address
{
}

public class Person
{
    private Address address;
    public Person(Address address)
    {
        this.address = address;
    }
}
```

Class diagram – Composition Relationship

- Is a **stronger** version of **AGGREGATION**
- The “part(s)” may belong to only **ONE whole**
- The part(s) are usually expected to **“live” and “die”** with the whole (“cascading delete”)



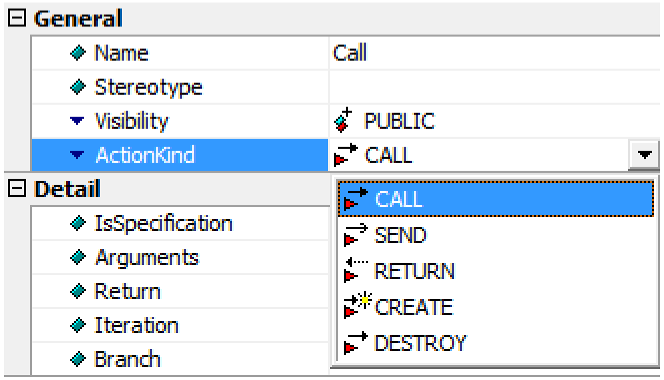


```
public class Car
{
    private Engine engine;

    public Car()
    {
        engine = new Engine(); // always having engine
    }

}
```

Sequence diagram & Activity diagram

- **Sequence diagrams:** Describes a pattern of interaction among objects, arranged in a **time order**; it shows the objects participating in the **interaction** by their "lifelines" and the **messages** that they send to each other.
- **Activity diagrams**: are essentially a flowchart, showing flow of control activity to activity.

Sequence Diagram

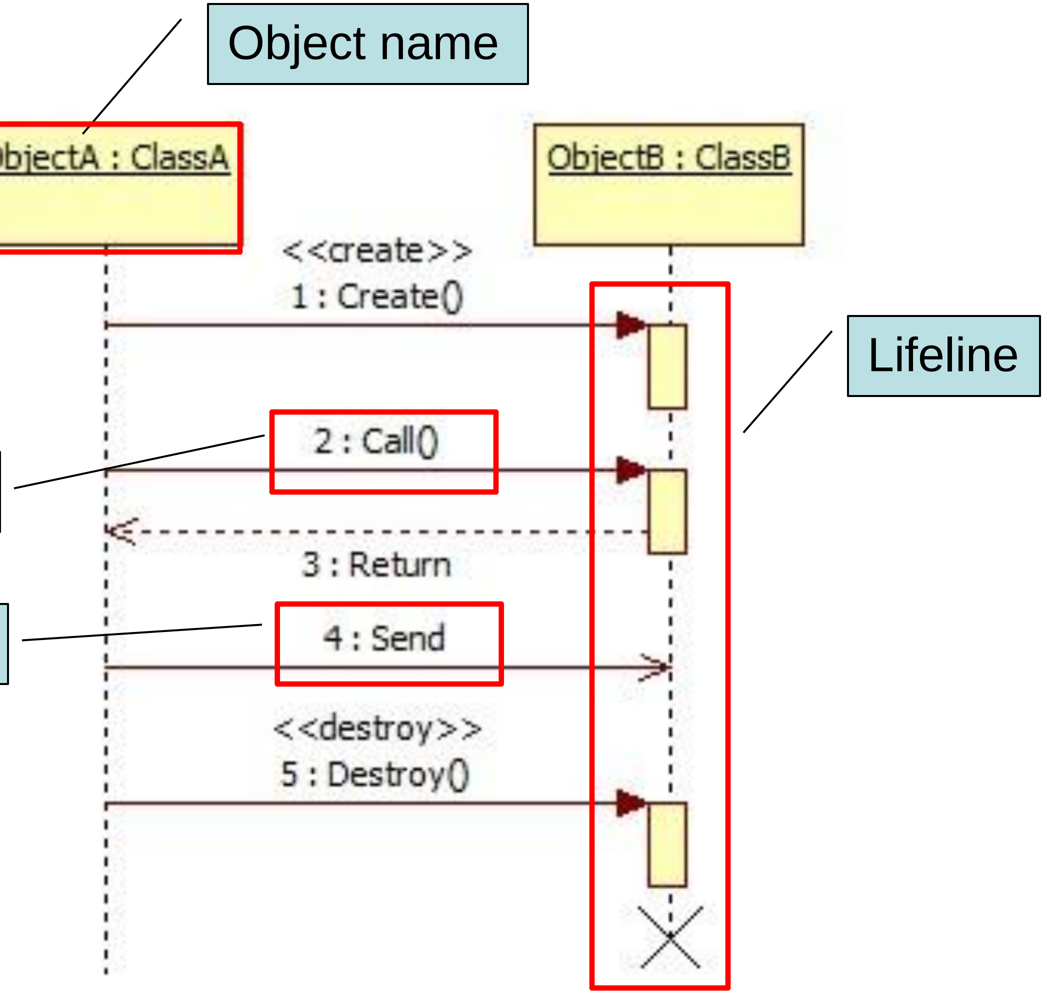


⊟ General

Name	Call
Stereotype	
Visibility	PUBLIC
ActionKind	CALL

⊟ Detail

IsSpecification
Arguments
Return
Iteration
Branch

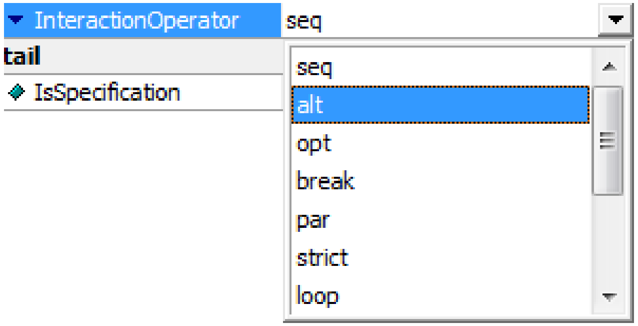


Object name

Lifeline

Synchronous Message

Asynchronous Message



InteractionOperator	seq

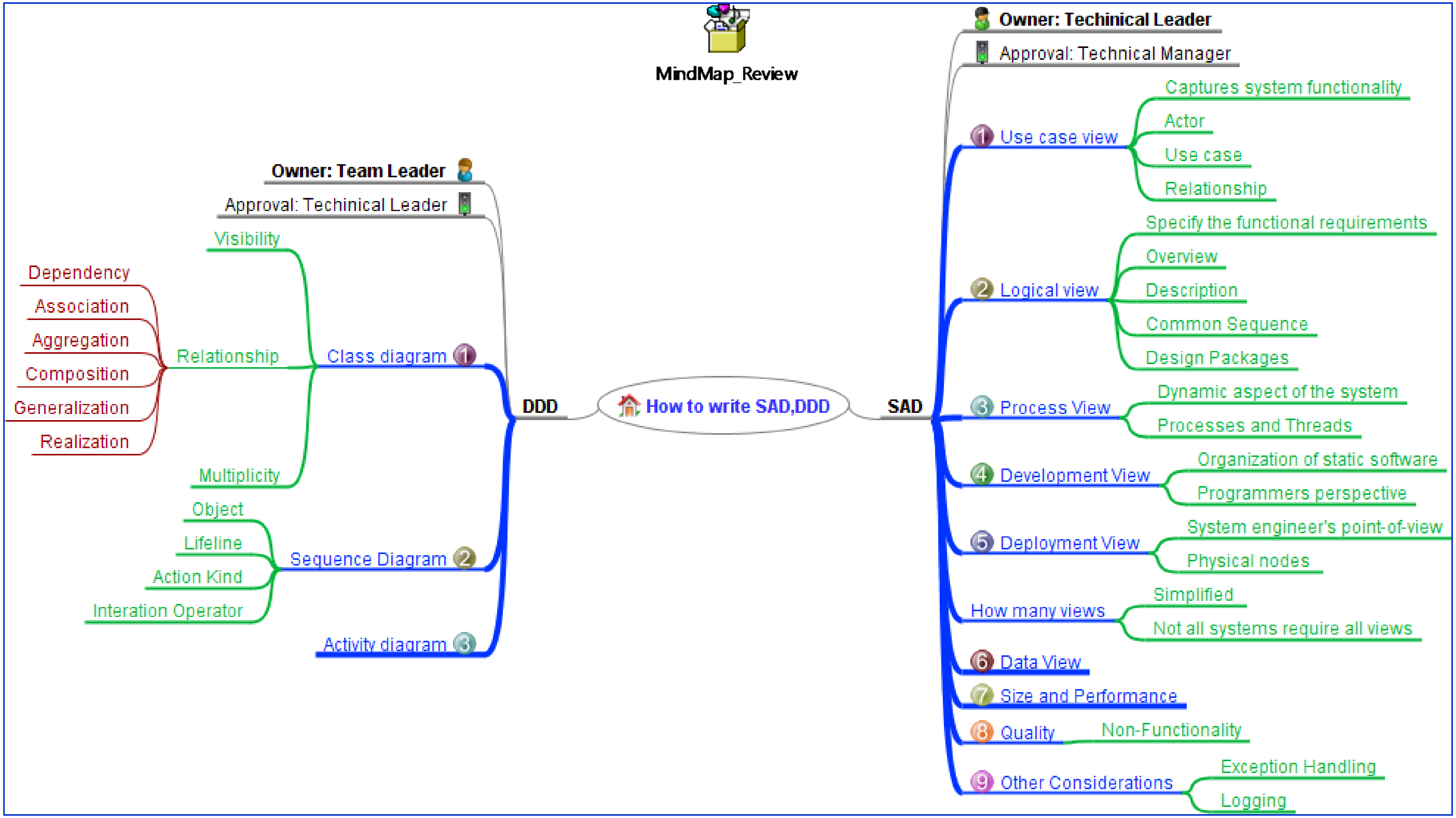
tail

IsSpecification

References

- http://www.iso-architecture.org/ieee-1471/docs/all-about-ieee-1471.pdf
- https://intranet.harveynash.vn/Process%20Asset%20Library/Forms/AllItems.aspx
- http://www-rohan.sdsu.edu/faculty/rnorman/course/ids306/norman-f99-306.htm
- http://people.uncw.edu/simmondsd/courses/Syllabi/csc450/lectures/lectures.htm
- http://*www.ece.uvic.ca/~itraore/seng422-05/notes/arch05-5.pdf*
- http://www.ncsta.gov/library/pp/Architecture%20Review%20Processes.ppt
- http://www.spiiras.nw.ru/rus/conferences/ict/Baranov270204.ppt
- http://www.cs.vu.nl/~hans/SEslides/arch.ppt
- http://www.ccs.neu.edu/home/lieber/courses/csg110/sp05/lectures/march31/arch_intro.ppt
- http://www.rgoarchitects.com/Files/Architecture.ppt

Summary



MindMap_Review

How to write SAD,DDD

- DDD
  - Owner: Team Leader
  - Approval: Techinical Leader
  - Class diagram 1
    - Visibility
    - Relationship
      - Dependency
      - Association
      - Aggregation
      - Composition
      - Generalization
      - Realization
    - Multiplicity
  - Sequence Diagram 2
    - Object
    - Lifeline
    - Action Kind
    - Interation Operator
  - Activity diagram 3
- SAD
  - Owner: Techinical Leader
  - Approval: Technical Manager
  - 1 Use case view
    - Captures system functionality
    - Actor
    - Use case
    - Relationship
  - 2 Logical view
    - Specify the functional requirements
    - Overview
    - Description
    - Common Sequence
    - Design Packages
  - 3 Process View
    - Dynamic aspect of the system
    - Processes and Threads
  - 4 Development View
    - Organization of static software
    - Programmers perspective
  - 5 Deployment View
    - System engineer's point-of-view
    - Physical nodes
  - How many views
    - Simplified
    - Not all systems require all views
  - 6 Data View
  - 7 Size and Performance
  - 8 Quality
    - Non-Functionality
  - 9 Other Considerations
    - Exception Handling
    - Logging

Q&A

The Need of Architecture???

The Winchester "Mystery" House

- **38 years** of construction – 147 builders **0 architects**
- 160 rooms – 40 bedrooms, 6 kitchens, 2 basements, 950 doors
- 65 doors to blank walls, 13 staircases abandoned, 24 skylights in floors
- No architectural blueprint exists

Appendix

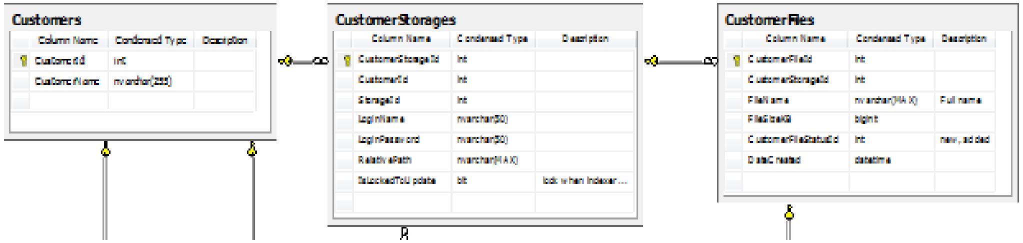
- Other Items in HVN’s SAD

Other Items in HVN’s SAD

- Data View
- Size and Performance
- Quality
- Other Considerations

Other Items in HVN's SAD – Data View

- Describes the **persistent data storage** perspective of the system
- This section is **optional** if there is little or no persistent data or the translation between the Design Model and the Data Model is trivial



Customers

	Column Name	Condensed Type	Description
🔑	CustomerId	int	
	CustomerName	nvarchar(255)	

CustomerStorages

	Column Name	Condensed Type	Description
🔑	CustomerStorageId	int	
	CustomerId	int	
	StorageId	int	
	LoginName	nvarchar(50)	
	LoginPassword	nvarchar(50)	
	RelativePath	nvarchar(MAX)	
	IsLockedToUpdate	bit	lock when Indexer ...

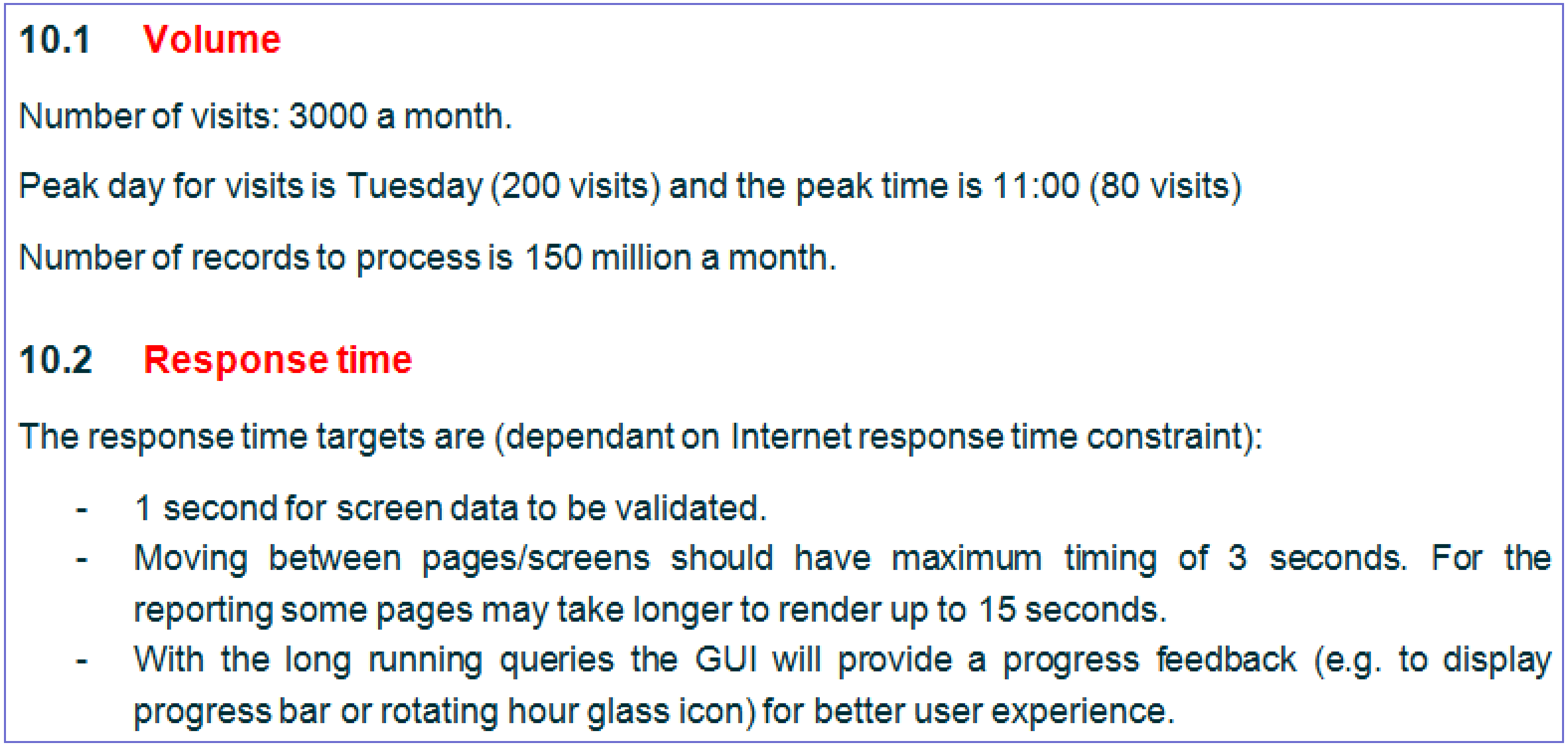
CustomerFiles

	Column Name	Condensed Type	Description
🔑	CustomerFileId	int	
	CustomerStorageId	int	
	FileName	nvarchar(MAX)	Full name
	FileSizeKB	bigint	
	CustomerFileStatusId	int	new, added
	DateCreated	datetime	

Customer Upload Area

Other Items in HVN’s SAD - Size and Performance

- Describes the **major dimensioning** characteristics of the software that impact the architecture, as well as the target **performance constraints**



10.1 Volume

Number of visits: 3000 a month.

Peak day for visits is Tuesday (200 visits) and the peak time is 11:00 (80 visits)

Number of records to process is 150 million a month.

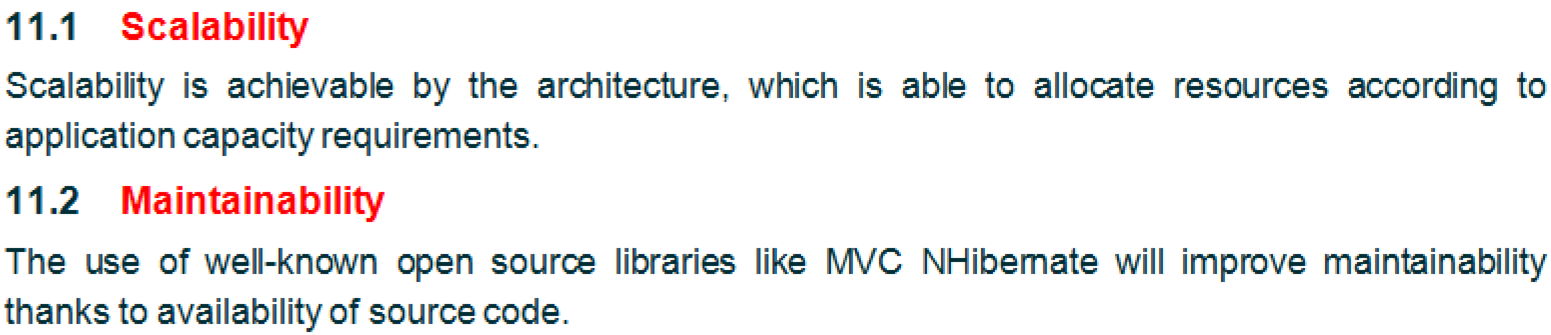
10.2 Response time

The response time targets are (dependant on Internet response time constraint):

- 1 second for screen data to be validated.
- Moving between pages/screens should have maximum timing of 3 seconds. For the reporting some pages may take longer to render up to 15 seconds.
- With the long running queries the GUI will provide a progress feedback (e.g. to display progress bar or rotating hour glass icon) for better user experience.

Other Items in HVN’s SAD - Quality

- Describes how the software architecture contributes to all capabilities (**other than functionality**) of the system: extensibility, reliability, portability, and so on.
- If these characteristics have **special significance**, for example safety, security or privacy implications, they should be clearly stated



11.1 Scalability

Scalability is achievable by the architecture, which is able to allocate resources according to application capacity requirements.

11.2 Maintainability

The use of well-known open source libraries like MVC NHibemate will improve maintainability thanks to availability of source code.

Other Items in HVN's SAD - Other Considerations

- Describes other approach/ solutions that were considered in selection process for the above architecture, i.e. a brief explanation of **advantages and disadvantages** of the selected architecture in comparison with others.
  - Exception Handling
  - Logging
  - …