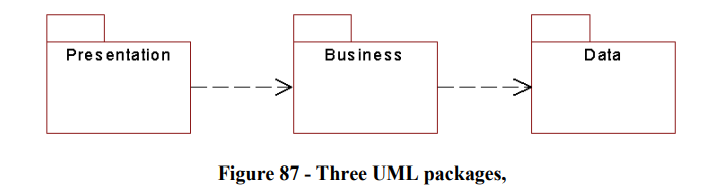
**Chapter 16**

**System Architecture - Large and Complex Systems**

**The UML Package Diagram**

* UML artifacts can be organized into logical containers called "UML packages," which can be displayed on a UML package diagram.
* Packages can contain related elements and can be arranged hierarchically, similar to folders in an operating system.
* The package diagram provides a high-level view of the system, and case tools allow users to explore the contents of the package.



The example shows the three-tier model of software development, where items in one package may be dependent on items in another package.

**Elements Inside a Package**

Packages in UML are commonly used to group related classes or use cases together, and the names of elements within a package must be unique.

However, packages allow for different teams to work on different parts of a system without worrying about name clashes between their elements, providing a significant benefit in terms of organization and collaboration.

**Why Packaging?**

• Group large systems into easier to manage subsystems

• Allow parallel iterative development

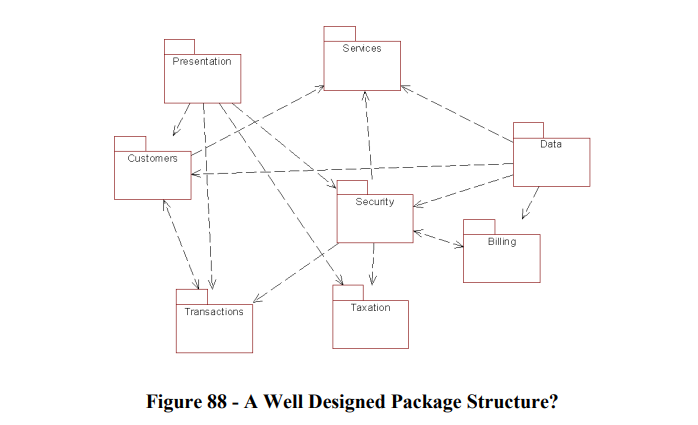
* Stand a chance of achieving code reuse(if we design each package well and provide clear interfaces between the packages)

*Comparision with class*

Unlike classes, which are small and difficult to reuse, packages offer the potential for creating larger, more comprehensive components that can be reused across different projects.

**Some Packaging Heuristics**

Package diagram can be used to partition classes into easy-to-understand and maintain packages. To design a well-structured package, several heuristics from the GRASP chapter apply, including Expert, High Cohesion, and Loose Coupling.



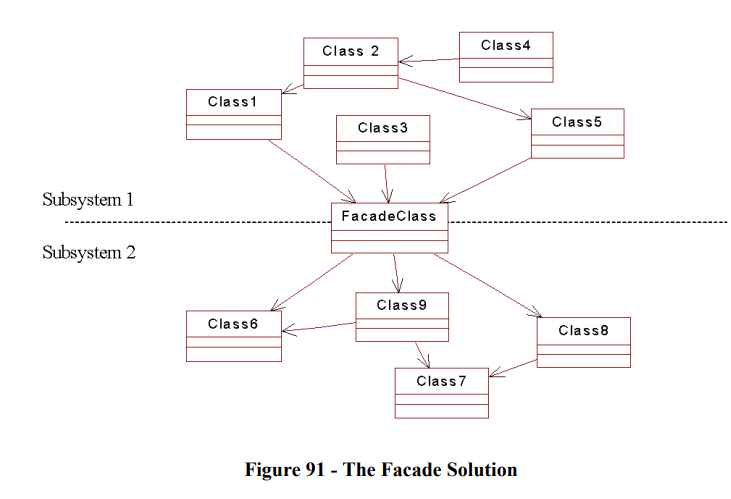
* **The Expert**: heuristic states that each class should belong to an obvious package. Heuristic ( tương đối)
* **High Cohesion**: suggests that a package should not do too much to avoid difficulty in understanding and reusing.
* **Loose Coupling**: emphasizes that dependencies between packages should be minimized, and too much cross-package communication can lead to a complicated and difficult-to-maintain system.

**Handling Cross Package Communication**

The dependency between classes in different subsystems/packages should be minimized to avoid difficulties in maintaining and changing the system.

The use of a Facade design pattern can help to simplify the process of replacing or changing subsystems/packages.

**The Facade Pattern**



Using a Facade class to act as an intermediary between two subsystems can improve the design of a system by reducing cross-subsystem communication and increasing modularity.

The Facade provides a collection of all public methods for the subsystem and ensures that all calls are directed through it.

If one subsystem needs to be replaced, only the Facade needs to be updated, rather than every class in the subsystem.

In Java, the use of package protection for classes can further enhance the modularity and encapsulation of the system.

**Architecture-Centric Development**

Lên kế hoạch từ đầu(thế mạnh

The Rational Unified Process advocates for architecture-centric development, which involves planning the system as a collection of subsystems early on in the project development.

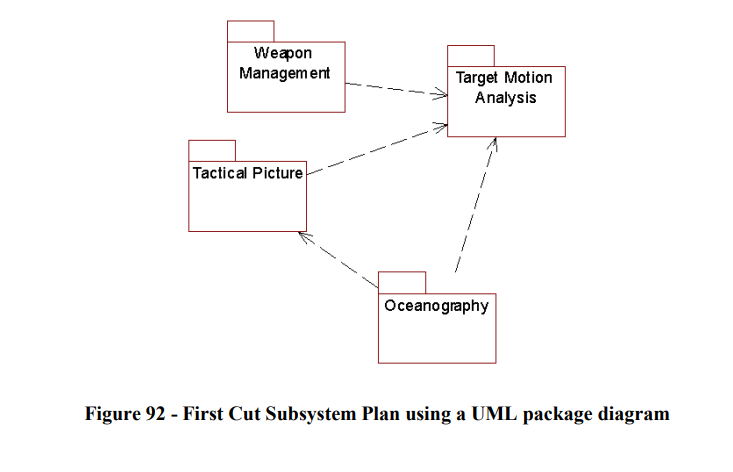
Small development teams are assigned to each subsystem, allowing them to work in parallel, independent of each other.

An architecture team is appointed to manage the architectural model and the package diagram, as well as the interfaces between subsystems.

Changes to the interfaces must be performed by the architecture team, as they maintain a constant high-level view of the system and are best placed to understand the impact of changes on the subsystems.

**Example**

The architecture team for a command and control system creates a package diagram as a first cut of the system architecture, identifying the major areas of functionality to be offered by the system.

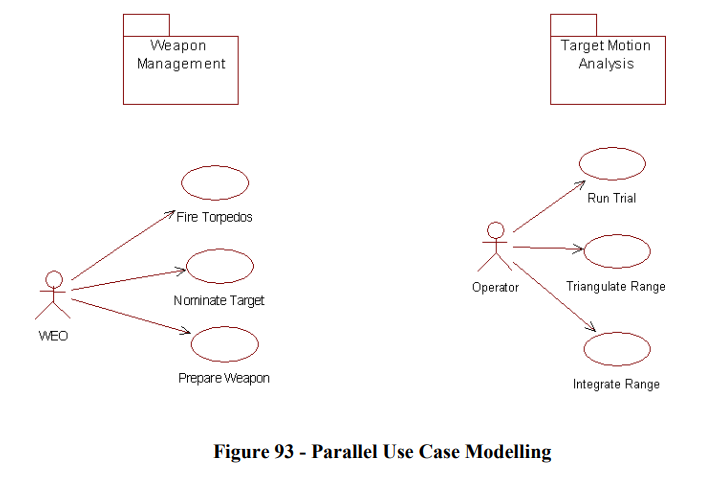


This diagram is not set in stone and will evolve and expand as the project progresses to contain the complexity of each subsystem.

The team would continue setting up subsystems until the size of each subsystem is easy to manage.

Use cases may then be built for each subsystem, treating each subsystem as a system in its own right.

Each subsystem is treated as a system in its own right, exactly as we did in the early stages of the book:



**Handling Large Use Cases**

In large scale software development, the use cases identified during the Elaboration phase may be too complex to be developed in a single iteration.

\**Solution*:

To address this, the solution is not to lengthen the iterations, but to break down the use cases into a series of easier-to-manage versions.

For instance, a large use case such as "Fire Torpedoes" can be divided into separate versions, with each version achievable in a single iteration.

The versions:

• Version 1 - allows the opening of bow caps

• Version 2 - allows interlocks to be set

• Version 3 - allows the discharge of weapons

\**Conclusion*:

This approach ensures that each version is easy to understand and manageable. The aim is to make sure that each version is completed within a single iteration.

**The Construction Phase**

The construction phase carries on as described in earlier chapters, but with each subsystem being developed, iteratively, by separate teams, working in parallel and as independently as possible.

At the end of each iteration, a phase of integration testing will take place, where the interfaces across subsystems are tested.

**Summary**

This chapter looked at some of the issues surrounding large scale system development. It is clear that although the UML is designed to be scaleable, transferring the Iterative Incremental Framework to large projects is far from a simple exercise. The best approach at the moment seems to be the Architecture Centric approach proposed by Rational Corp:

• Define subsystems from an early stage

• Keep complexity as manageable as possible

• Iterate in parallel but donít hack interfaces

• Appoint a central architecture team

The package model provided by the UML provides a way of containing the large complexity, and this model should be owned by the architecture team.

State diagran

Gruad : đk để trạng thái diễn ra