BINUS UNIVERSITY

People
Innovation
Excellence

COMP6115

Object Oriented Analysis and Design

Session #7

Moving on to Design

People
Innovation
Excellence

Learning Outcomes

LO1: Identify the basic concept of advance topic in Object Oriented Analysis and Design

LO2 : Use the knowledge to develop documentation for object oriented software analysis and design using Unified Modelling Language

LO3 : Analyze any problem in any software application and find out the alternative solutions using object oriented analysis and design approach

BINUS UNIVERSITY

People
Innovation
Excellence

Chapter 7:

Moving on to Design

Learning Objectives

1. Verifying and Validating the Analysis Models
2. Balancing between Functional, Structural, Behavioral Models
3. Evolving the Analysis Models into Design Models
4. Packages and Package Diagrams
5. Design Strategies: Custom Development, Packaged Software, Outsourcing
6. Selecting an Acquisition Strategy

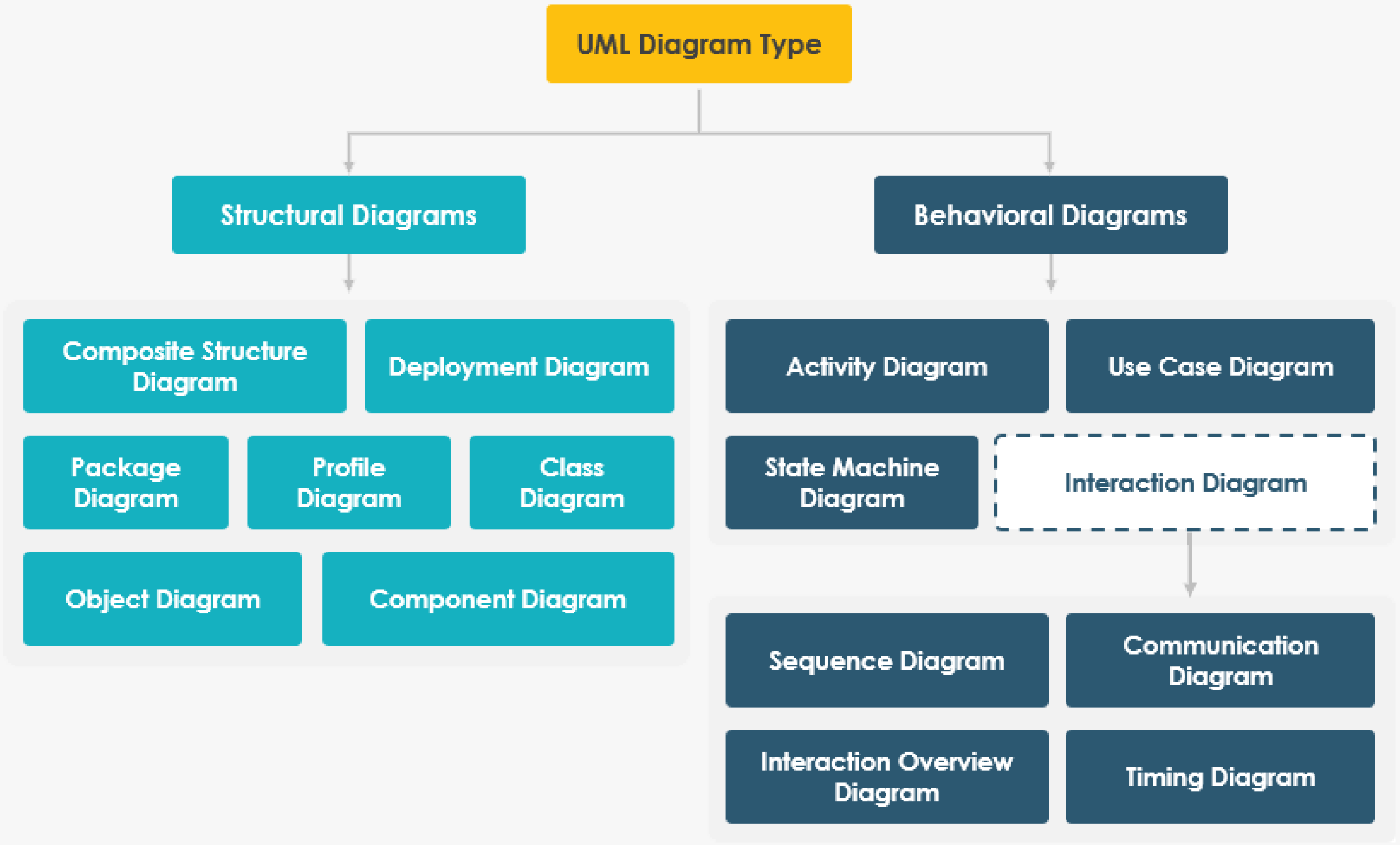
People
Innovation
Excellence

Introduction

- **Analysis determines the business needs**
- **Design activities focus on how to build the system**
  1. Major activity is to evolve the models into a design
  2. Goal is to create a blueprint for the design that makes sense to implement
  3. Determine how and where data will be stored
  4. Determine how the user will interface with the system (user interface, inputs and outputs)
  5. Decide on the physical architecture
- **Analysis and design phases are highly *interrelated* and may require much "going back and forth"**
  – Example: prototyping may uncover additional information

People
Innovation
Excellence

UML Diagram Types



- **Structural diagrams** show the things in the modeled system. In a more technical term, they show different objects in a system.
- **Behavioral diagrams** describe the internal behavior of a system. They describe how the objects interact with each other to create a functioning system.

People
Innovation
Excellence

VERIFYING AND VALIDATING THE ANALYSIS MODELS

The Design Process

- Verify and validate the analysis models
- Evolve the analysis models into design models
- Create packages and utilize package diagrams
- Decide upon a design strategy

Verifying & Validating the Analysis Models

- **Do the analysis models accurately represent the problem domain?**

  Test the consistency and relationship of each model

  Example:
  Activity Diagrams, Use-Case Descriptions, and Use-Case Diagrams should all describe the same functional requirements

- **Balance the models to ensure consistency between them**

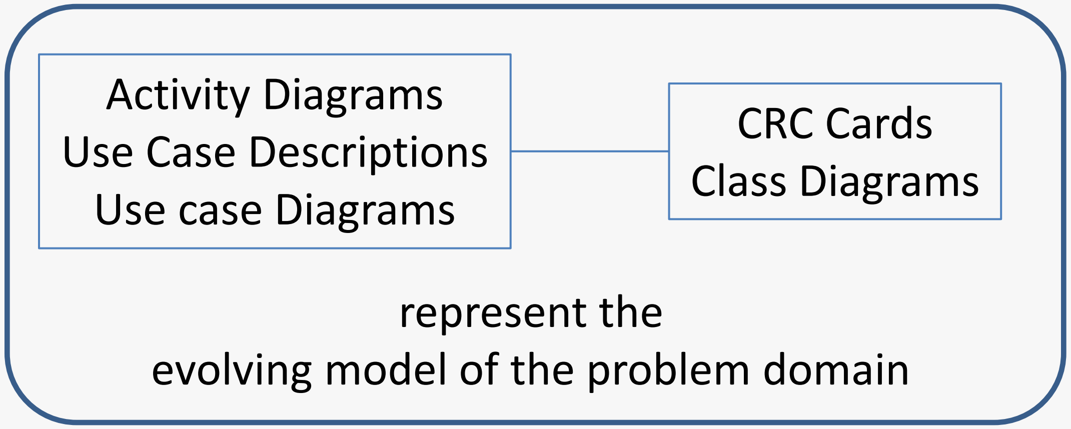
People
Innovation
Excellence

People
Innovation
Excellence

BALANCING BETWEEN FUNCTIONAL, STRUCTURAL, BEHAVIORAL MODELS

Balancing Functional & Structural Models

To **balance** the **functional** and **structural models**, we must **ensure** that the **two sets of models** are **consistent** with each other.



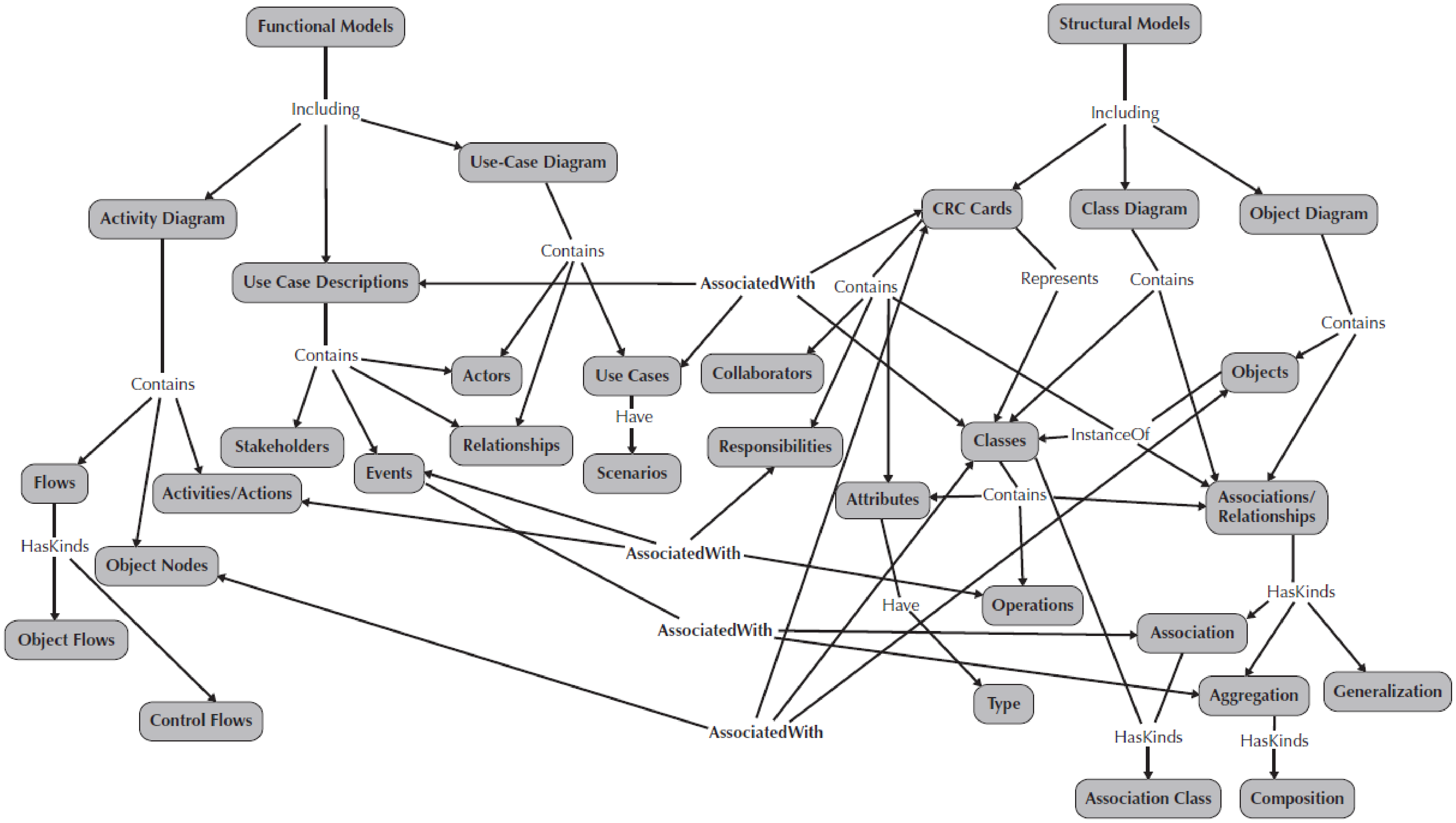


FIGURE 7-2 Relationships among Functional and Structural Models

To **balance** the **functional** and **structural models**, we must **ensure** that the **two sets of models** are **consistent** with each other.

Balancing Functional & **Structural Models**

1. Every **class on a class diagram and every CRC card** must be associated with at least **one use-case**, and vice versa.
2. Every activity in an **activity diagram** and every event in a **use-case description** should be related to one or more responsibilities on a **CRC card** and **one or more operations in a class on a class diagram** and vice versa.
3. Every **object node on an activity diagram** must be associated with an **instance of a class on a class diagram (i.e., an object) and a CRC card** or **an attribute contained in a class and on a CRC card**.
4. Every **attribute and association/aggregation relationships contained on a CRC card** (and connected to a class on a class diagram) should be **related to the subject or object of an event in a use-case description**.

People
Innovation
Excellence

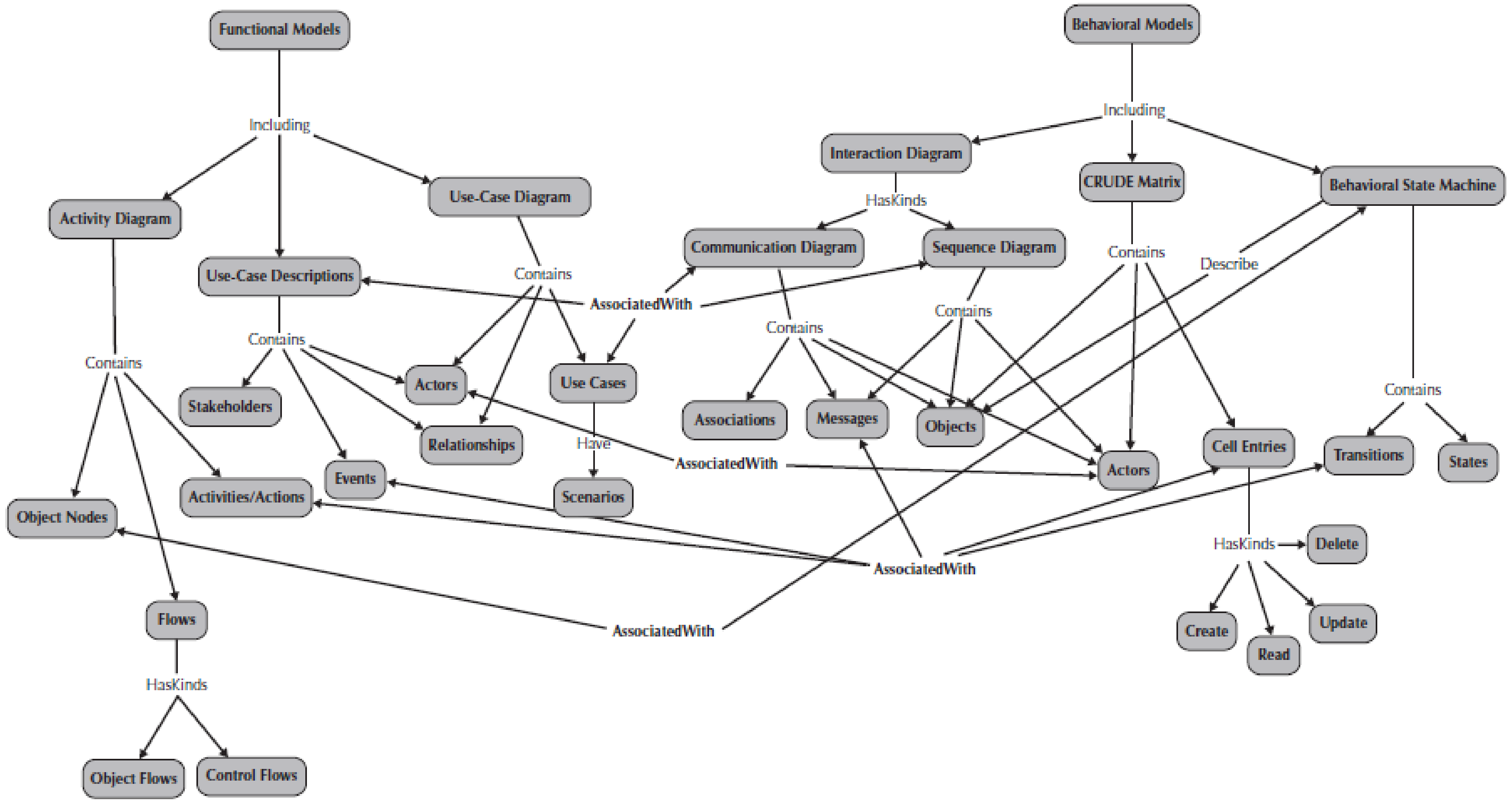


FIGURE 7-8 Relationships between Functional and Behavioral Models

As in balancing the functional and structural models, we must ensure the consistency of the two sets of models.

Balancing Functional & **Behavioral Models**

1. The **sequence** and **communication diagrams** must be **associated** with **a use case on the use-case diagram and a use-case description**.
2. Actors on **sequence diagrams, communication diagrams, and/or CRUDE matrices** must be **associated** with **actors on the use-case diagram or referenced in the use-case description**, and vice versa.
3. **Messages on sequence and communication diagrams**, **transitions on behavioral state machines**, and **entries in a CRUDE matrix** must be **related** to **activities and actions on an activity diagram** and **events listed in a use-case description**, and vice versa.
4. **All complex objects represented by an object node in an activity diagram** must have a **behavioral state machine** that **represents the object's lifecycle**, and vice versa.

Balancing Structural & Behavioral Models (cont.)

1. Objects that appear in a CRUDE matrix must be associated with classes that are represented by CRC cards and appear on the class diagram, and vice versa.
2. Because behavioral state machines represent the life cycle of complex objects, they must be associated with instances (objects) of classes on a class diagram and with a CRC card that represents the class of the instance.
3. Communication and sequence diagrams contain objects that must be an instantiation of a class that is represented by a CRC card and is located on a class diagram.
4. Messages contained on the sequence and communication diagrams, transitions on behavioral state machines, and cell entries on a CRUDE matrix must be associated with responsibilities and associations on CRC cards and operations in classes and associations connected to the classes on class diagrams.
5. The states in a behavioral state machine must be associated with different values of an attribute or set of attributes that describe an object.

Summary

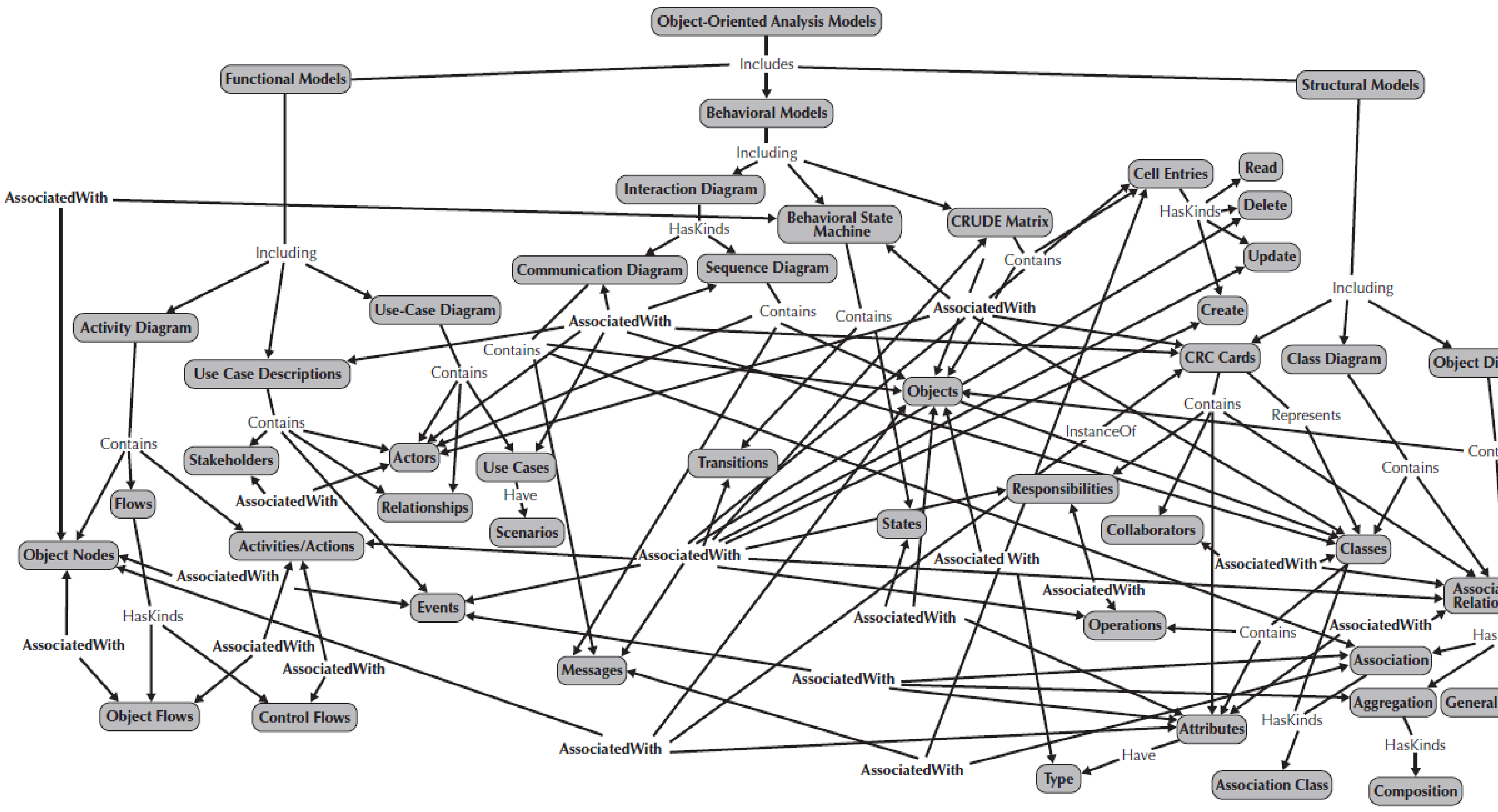


FIGURE 7-16 Interrelationships among Object-Oriented Analysis Models

People
Innovation
Excellence

EVOLVING THE ANALYSIS MODELS INTO DESIGN MODELS

Evolving the Analysis Models into Design Models

- **Now that we have successfully verified and validated our analysis models, we need to begin evolving them into appropriate design models.**

- Analysis models focused on functional requirements
- Design models must include non-functional requirements as well
  - System performance
  - System environment issues
    - Distributed vs. centralized processing
    - User interface
    - Database
- The system must be maintainable and affordable, efficient and effective
- Utilize **factoring**, **partitions** & **collaborations**, and **layers**

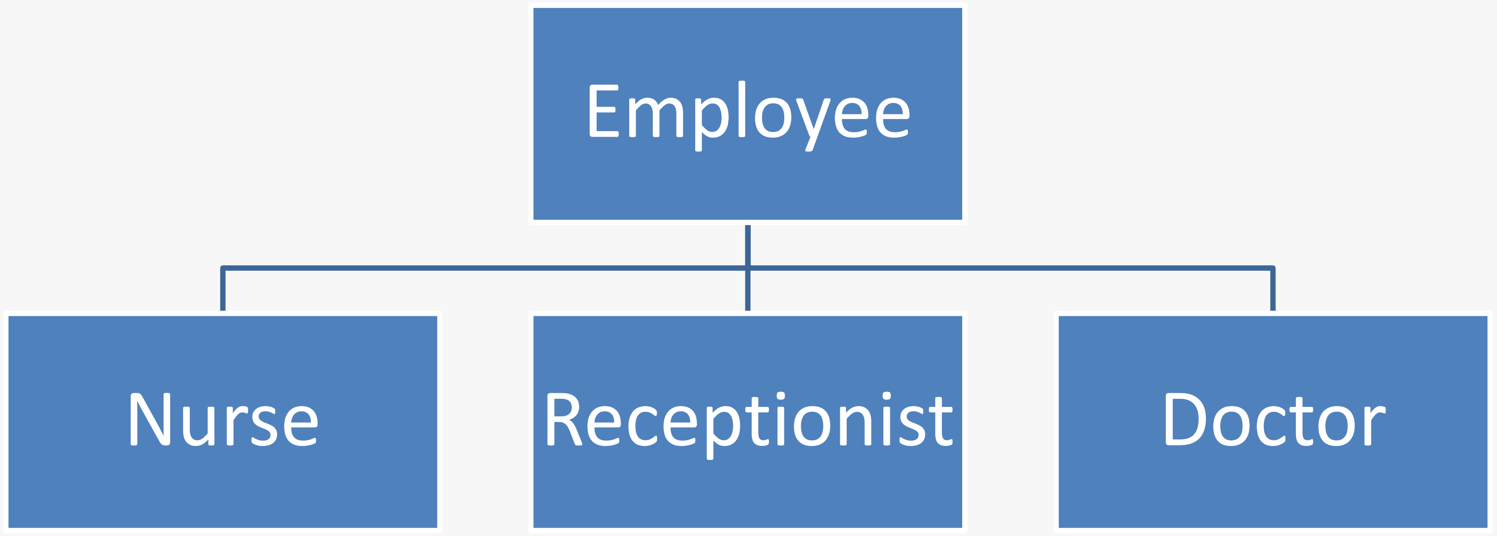
People
Innovation
Excellence

Factoring

- **Factoring is the process of separating out a module into a stand-alone module.**

- Creating modules that account for similarities and differences between units of interest
- New classes formed through a:
  - Generalization (a-kind-of) relationship, or a
  - Aggregation (has-parts) relationship
- Abstraction—create a higher level class (e.g., create an Employee class from a set of job positions)
- Refinement—create a detailed class (e.g., create a secretary or bookkeeper from the Employee class)

Generalization (a-kind-of) Relationship



People
Innovation
Excellence

Partitions and Collaborations

- **A partition is the object-oriented equivalent of a subsystem, where a subsystem is a decomposition of a larger system into its component systems**
- **e.g., an accounting information system could be functionally decomposed into an accounts-payable system, an accounts-receivable system, a payroll system, etc.**

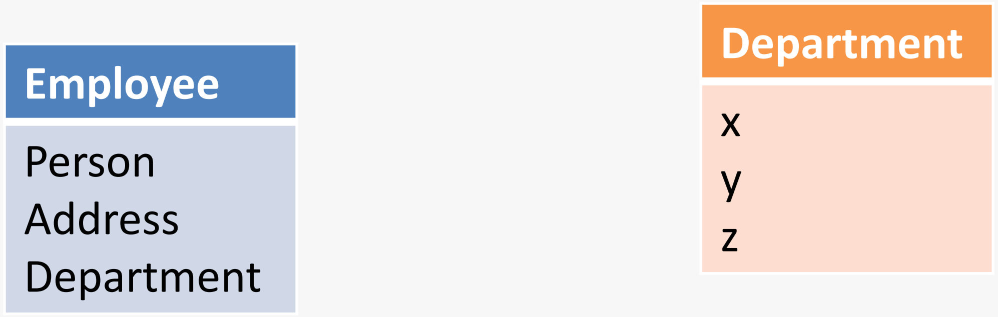
- Partition: create a sub-system of closely collaborating classes
  - Base partitions on patterns of activity (e.g., collaborations found in a communication diagram)
  - Greater coupling among classes may identify partitions (e.g., more messages passes between objects suggests that they belong in the same partition)
- Identifying partitions and collaborations determines which classes should be grouped together

People
Innovation
Excellence

People
Innovation
Excellence

Partitions and Collaborations

- For example, if attributes of a class have complex object types, such as Person, Address, or Department, and these object types were not modeled as associations in the class diagram, we need to recognize these implied associations.



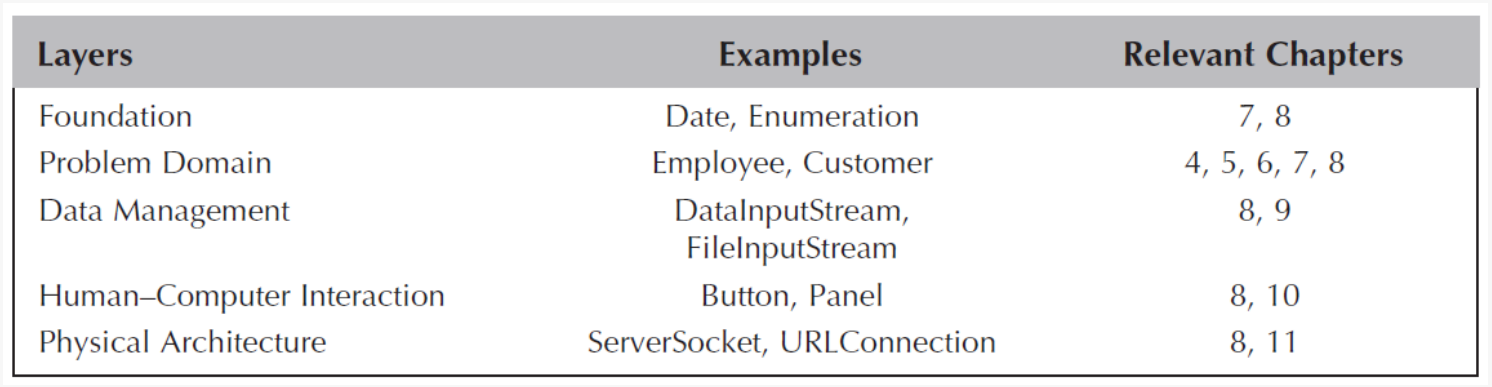
Layers

Until this point in the development of our system, we have focused only on the problem domain; we have totally ignored the system environment (data management, user interface, and physical architecture).

- System environment information must now be added
- Use layers to represent and separate elements of the software architecture
- A layer represents an element of the soft ware architecture of the evolving system.
  - Easier to understand a complex system
  - Example:
    - **Model-view-controller (MVC) architecture**
    - Separates application logic from user interface
  - Proposed layers:
    - Foundation (e.g., container classes)
    - Problem domain (e.g., encapsulation, inheritance, polymorphism)
    - **Data management** (e.g., data storage and retrieval)
    - **User interface** (e.g., data input forms)
    - Physical architecture (e.g., specific computers and networks)

People
Innovation
Excellence

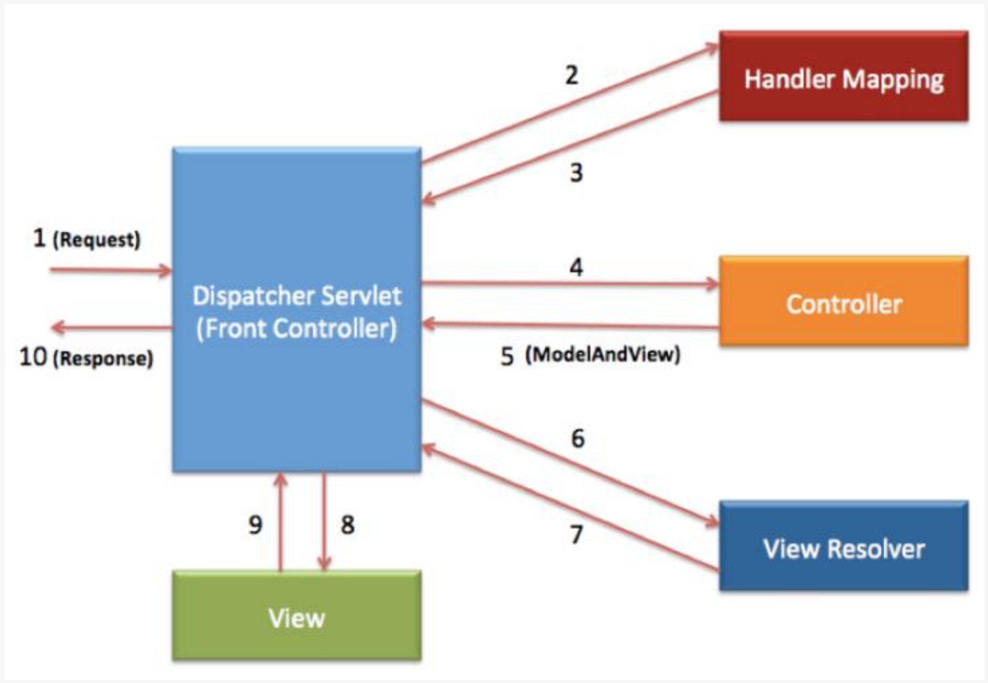
Layers



Layers	Examples	Relevant Chapters
Foundation	Date, Enumeration	7, 8
Problem Domain	Employee, Customer	4, 5, 6, 7, 8
Data Management	DataInputStream, FileInputStream	8, 9
Human–Computer Interaction	Button, Panel	8, 10
Physical Architecture	ServerSocket, URLConnection	8, 11

Layers

The idea of separating the different elements of the architecture into separate layers can be traced back to the MVC architecture of *Smalltalk*[1].



Spring MVC Framework Example

[1]See S. Lewis, *Th e Art and Science of Smalltalk: An Introduction to Object-Oriented Programming Using Visual-Works* (Englewood Cliff s, NJ: Prentice Hall, 1995).

People
Innovation
Excellence

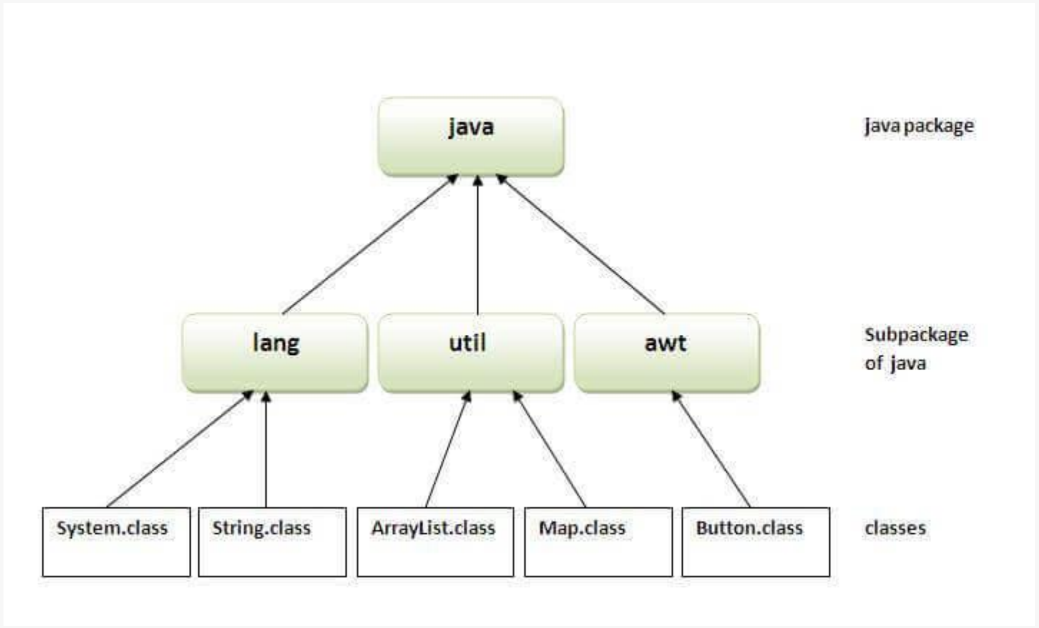
PACKAGES AND PACKAGE DIAGRAMS

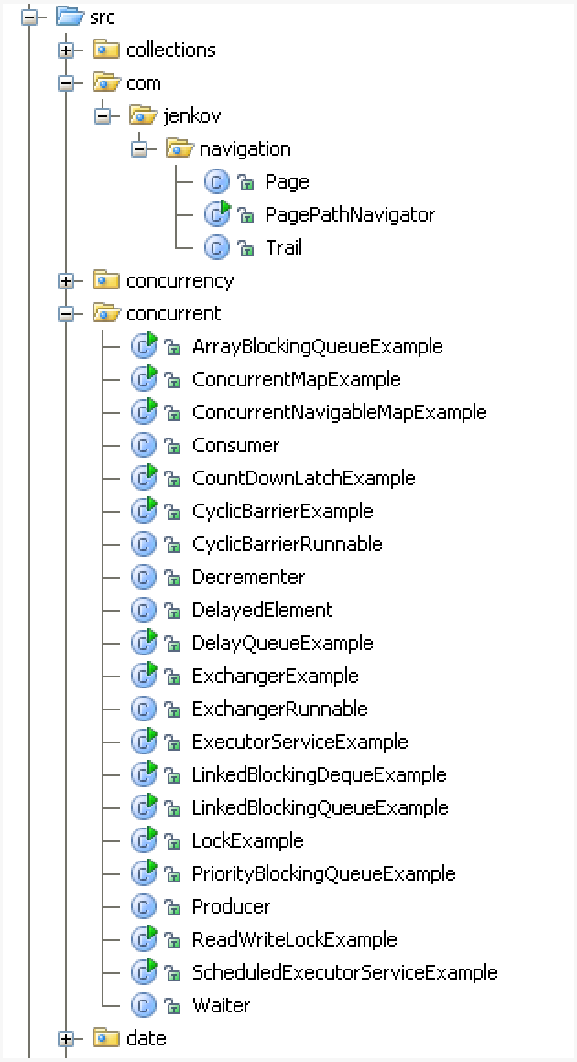
Packages and Package Diagrams

- **In UML, collaborations, partitions, and layers can be represented by a higher-level construct: a package.**

- Packages group together similar components (e.g., use-cases, class diagrams)
- Package diagrams show the packages and their relationships
  - Aggregation & association relationships are possible
  - Packages may be dependent upon one another
    - If one package is modified, others that depend on it may also require modification

Java Package Examples



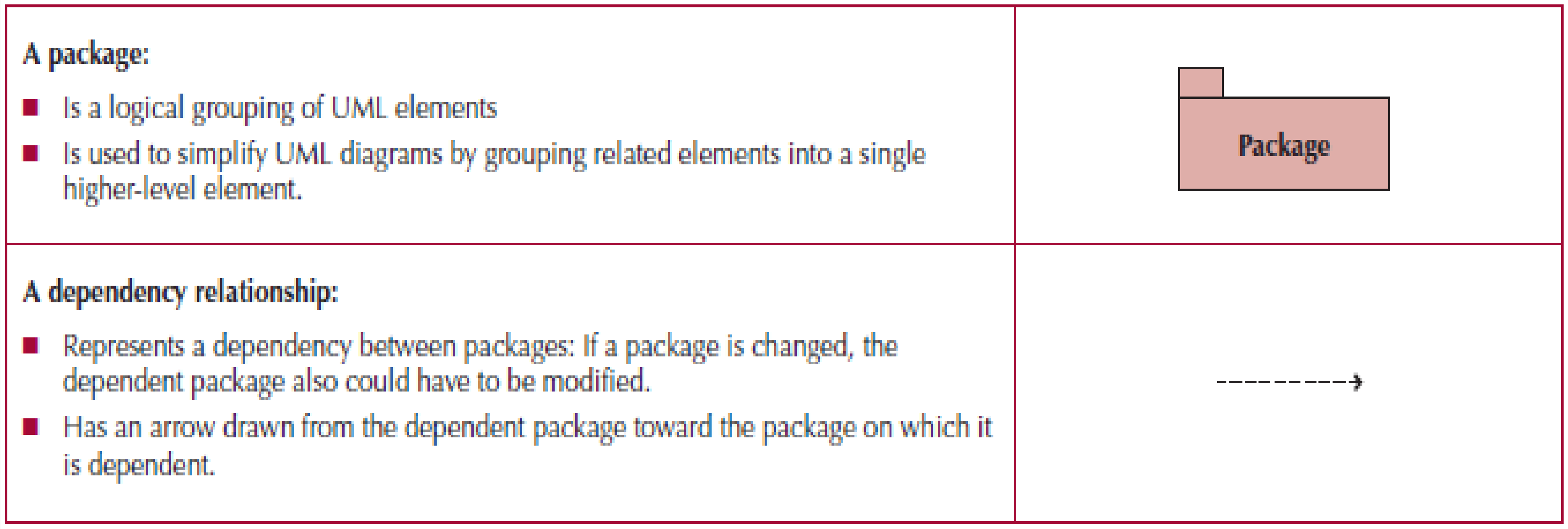


```
src
  collections
  com
    jenkov
      navigation
        Page
        PagePathNavigator
        Trail
  concurrency
  concurrent
    ArrayBlockingQueueExample
    ConcurrentMapExample
    ConcurrentNavigableMapExample
    Consumer
    CountDownLatchExample
    CyclicBarrierExample
    CyclicBarrierRunnable
    Decrementer
    DelayedElement
    DelayQueueExample
    ExchangerExample
    ExchangerRunnable
    ExecutorServiceExample
    LinkedBlockingDequeExample
    LinkedBlockingQueueExample
    LockExample
    PriorityBlockingQueueExample
    Producer
    ReadWriteLockExample
    ScheduledExecutorServiceExample
    Waiter
  date
```

People
Innovation
Excellence

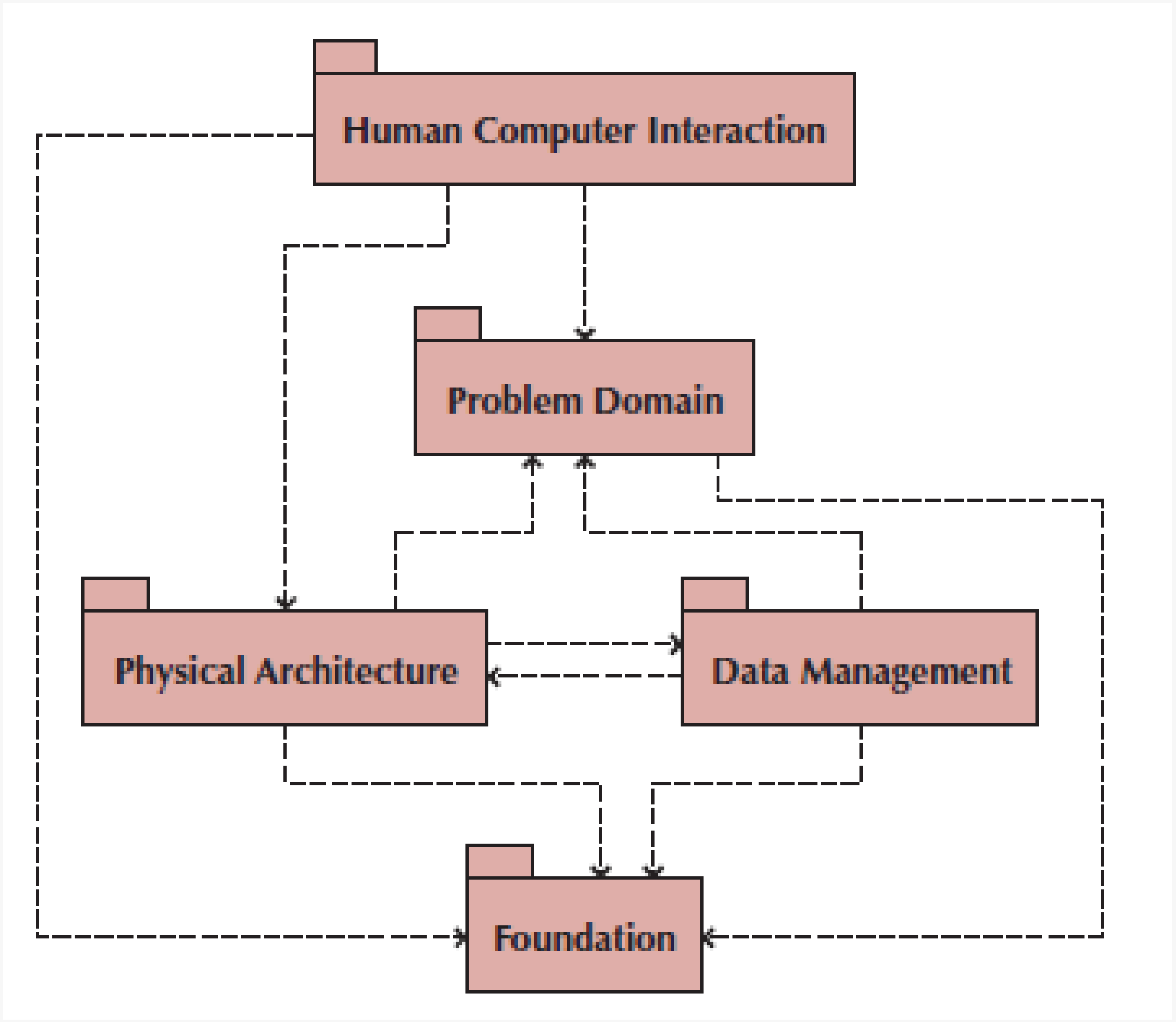
Package

- A general construct that groups units together
- Used to reduce complexity of models
- A package diagram shows packages only



A package: ■ Is a logical grouping of UML elements ■ Is used to simplify UML diagrams by grouping related elements into a single higher-level element.	Package
A dependency relationship: ■ Represents a dependency between packages: If a package is changed, the dependent package also could have to be modified. ■ Has an arrow drawn from the dependent package toward the package on which it is dependent.	$\dashrightarrow$

Package Diagram of Dependency Relationships among Layers



People
Innovation
Excellence

Guidelines for Building Package Diagrams

1. **Use package diagrams to logically organize designs.**
   - Specifically, use packages to group classes together when there is an inheritance, aggregation, or composition relationship between them or when the classes form a collaboration.
2. **In some cases, inheritance, aggregation, or association relationships exist between packages.**
   - In those cases, for readability purposes, try to support inheritance relationships vertically, with the package containing the superclass being placed above the package containing the subclass.
   - Use horizontal placement to support aggregation and association relationships, with the packages being placed side by side.
3. **When a dependency relationship exists on a diagram, it implies that there is at least one semantic relationship between elements of the two packages.**
   - The direction of the dependency is typically from the subclass to the superclass, from the whole to the part, and with contracts, from the client to the server.
   - In other words, a subclass is dependent on the existence of a superclass, a whole is dependent upon its parts existing, and a client can’t send a message to a nonexistent server.

Guidelines for Building Package Diagrams

4. **When using packages to group use cases together, be sure to include the actors and the associations that they have with the use cases grouped in the package.**
   – This will allow the diagram's user to better understand the context of the diagram.

5. **Give each package a simple, but descriptive name to provide the package diagram user with enough information to understand what the package encapsulates.**
   – Otherwise, the user will have to drill-down or open up the package to understand the package's purpose.

6. **Be sure that packages are cohesive.**
   – For a package to be cohesive, the classes contained in the package, in some sense, belong together. A simple, but not perfect, rule to follow when grouping classes together in a package is that the more the classes depend on each other, the more likely they belong together in a package.

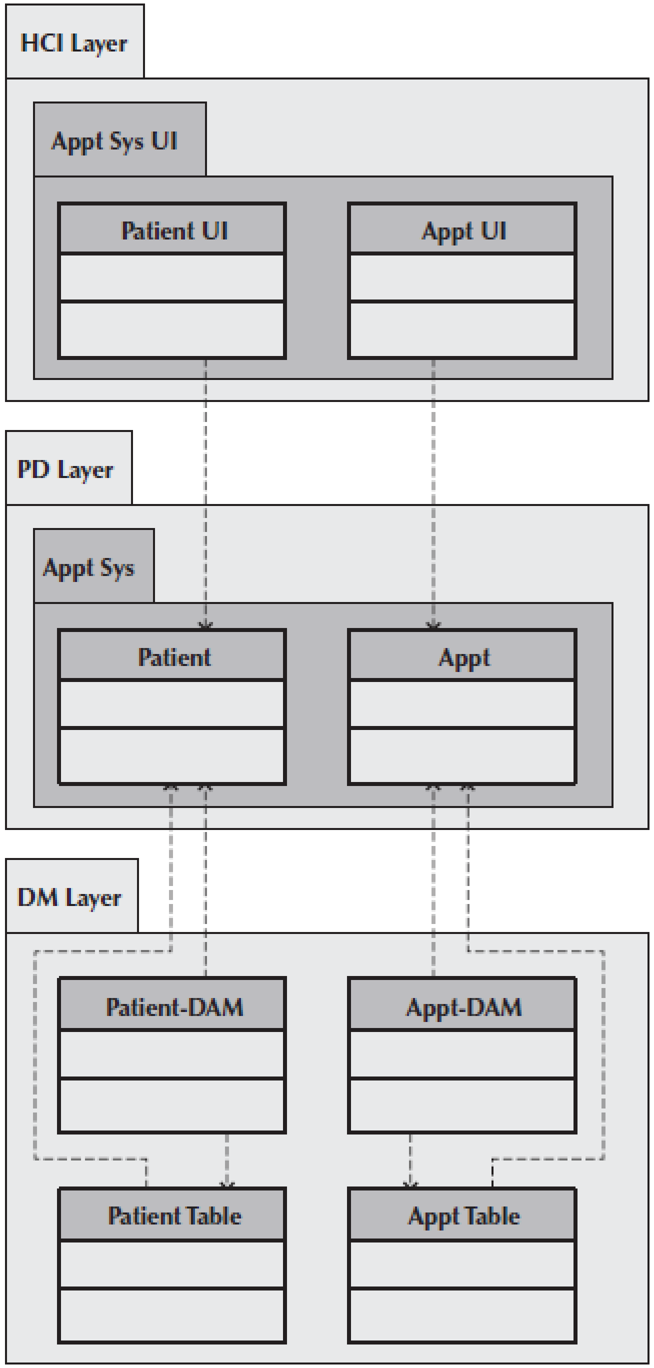


FIGURE 7-20
Partial Package Diagram of the Appointment System

BINUS UNIVERSITY

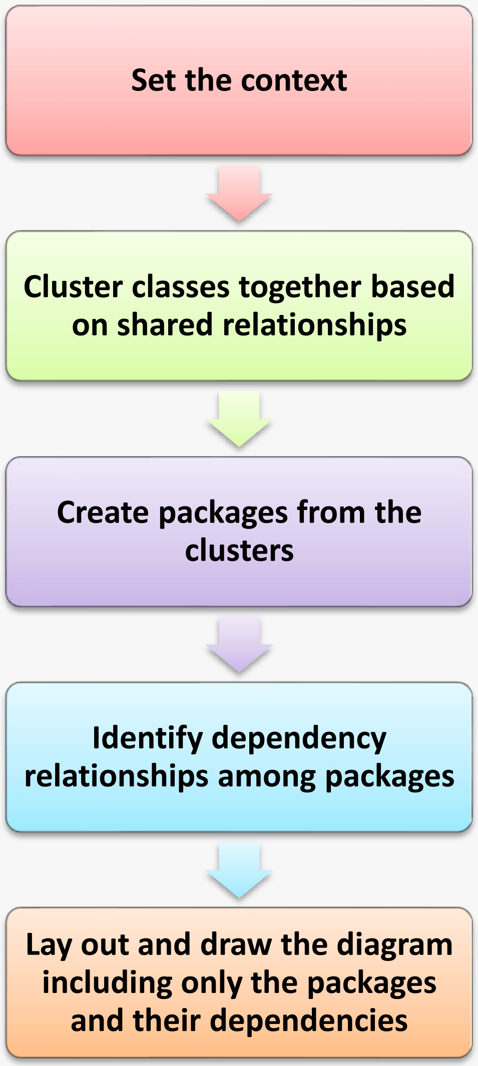
Guidelines for Building Package Diagrams

- Use them to logically organize your design
- Observe semantic relationships
  - Vertical positioning indicates inheritance
  - Horizontal positioning indicates aggregation and association
- Dependency relationships should also observe semantic relationships
- For use-case package diagrams, include the actors
- Use simple but descriptive names for each package
- Make packages cohesive

People
Innovation
Excellence

Building Package Diagrams

1. The first step is to set the context for the package diagram.
2. The second step is to cluster the classes together into partitions based on the relationships that the classes share.
3. The third step is to place the clustered classes together in a partition and model the partitions as packages.
4. The fourth step is to identify the dependency relationships among the packages.
5. The fifth step is to lay out and draw the diagram.


Set the context
Cluster classes together based on shared relationships
Create packages from the clusters
Identify dependency relationships among packages
Lay out and draw the diagram including only the packages and their dependencies

People
Innovation
Excellence

DESIGN STRATEGIES: CUSTOM DEVELOPMENT, PACKAGED SOFTWARE, OUTSOURCING

Design Strategies

Many project teams assume that custom development, or building a new system from scratch, is the best way to create a system.

➔ requires dedicated eff ort that involves long hours and hard work.

Alternatives:

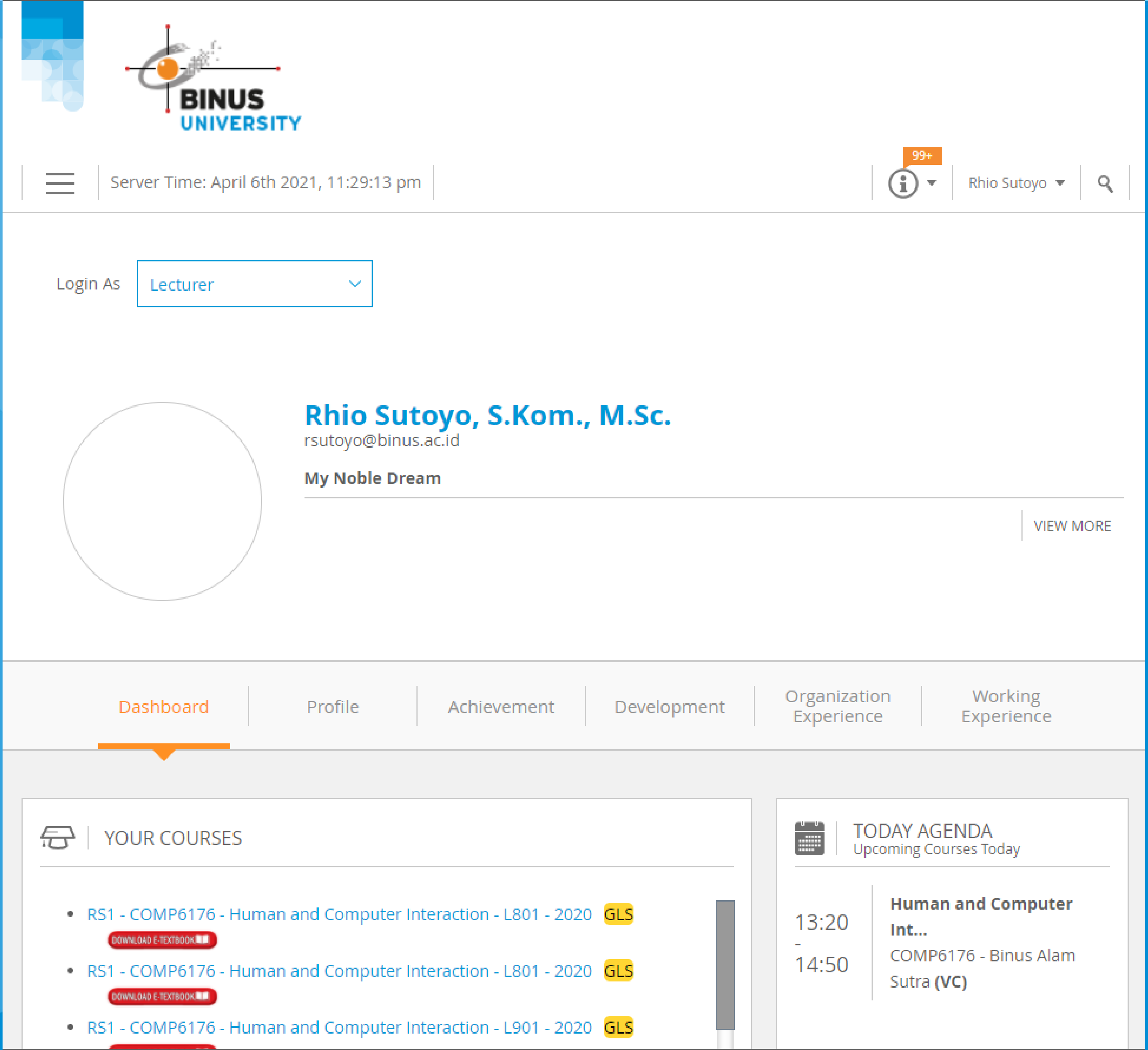
- Purchase packaged software
  - Office suites (e.g., word processors, spreadsheets, etc.)
  - Enterprise systems (e.g., SAP, PeopleSoft)
- Hire an external vendor (outsource)

People
Innovation
Excellence

Custom Development

- Allows for meeting highly specialized requirements
- Allows flexibility and creativity in solving problems
- Easier to change components
- Builds personnel skills
- May excessively burden the IT staff
- May add significant risk

Custom Development



Server Time: April 6th 2021, 11:29:13 pm

99+ Rhio Sutoyo

Login As: Lecturer

Rhio Sutoyo, S.Kom., M.Sc.
rsutoyo@binus.ac.id

My Noble Dream

VIEW MORE

Dashboard | Profile | Achievement | Development | Organization Experience | Working Experience

YOUR COURSES

- RS1 - COMP6176 - Human and Computer Interaction - L801 - 2020 GLS
  DOWNLOAD E-TEXTBOOK
- RS1 - COMP6176 - Human and Computer Interaction - L801 - 2020 GLS
  DOWNLOAD E-TEXTBOOK
- RS1 - COMP6176 - Human and Computer Interaction - L901 - 2020 GLS

TODAY AGENDA
Upcoming Courses Today

13:20 - 14:50 **Human and Computer Int...**
COMP6176 - Binus Alam Sutra **(VC)**

BINUS UNIVERSITY

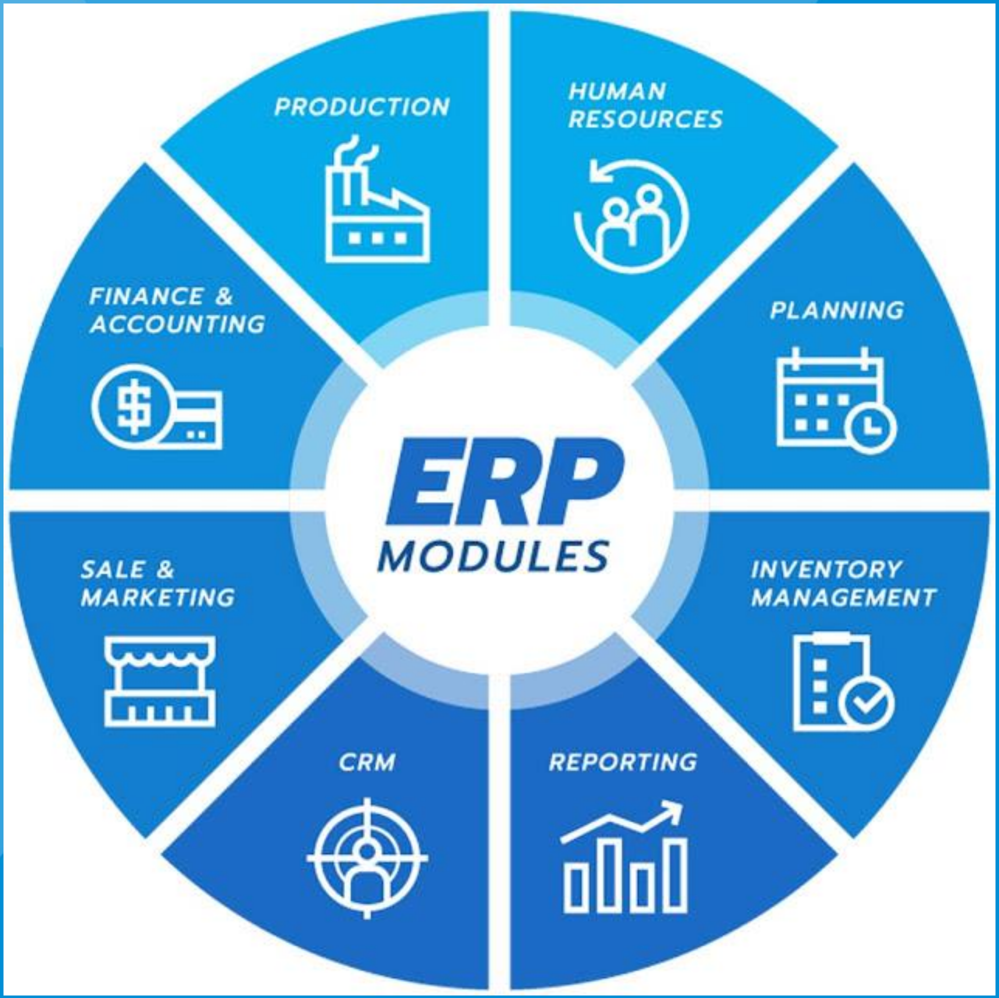
People
Innovation
Excellence

Packaged Software

- Software already written (e.g., accounting software)
- May be more efficient
- May be more thoroughly tested and proven
- May range from components to tools to entire enterprise systems
- Must accept functionality provided
- May require change in how the firm does business
- May require significant “customization” or “workarounds”

Packaged Software

Packaged Software



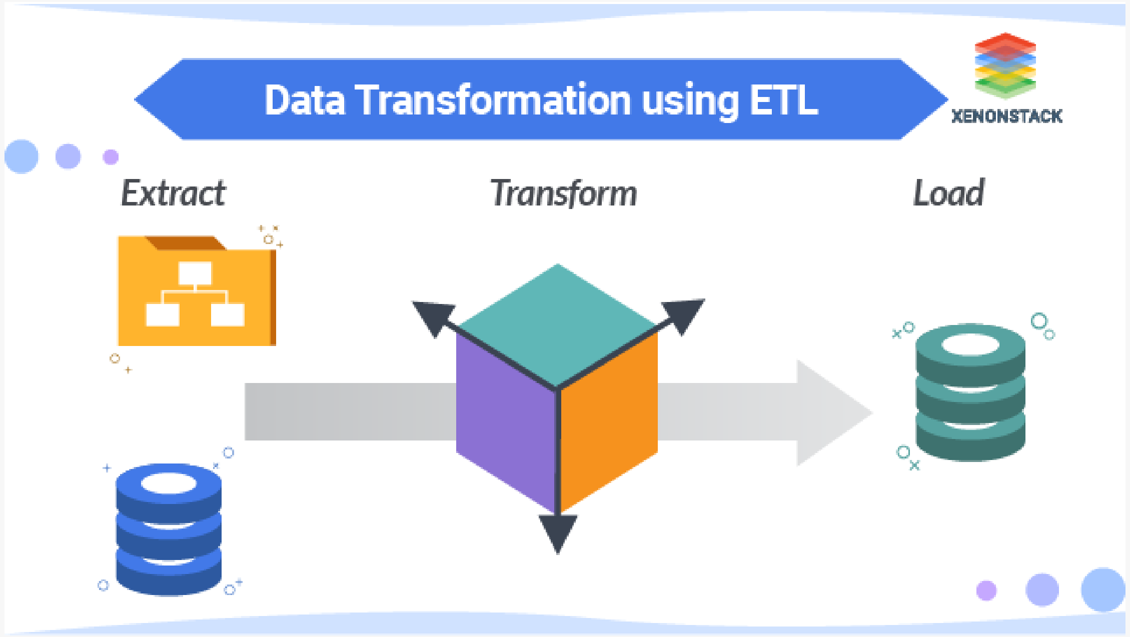
System Integration

- **Building a new system by combining packages, legacy systems, and new software**
  - Not uncommon to purchase off the shelf software and outsource its integration to existing systems
- **Key challenge is integrating data**
  - May require data transformations
  - New package may need to write data in the same format as a legacy system
- **Develop "object wrappers"**
  - Wraps the legacy system with an API to allow newer systems to communicate with it
  - Protects the investment in the legacy system

Human Capital

Operations

Marketing

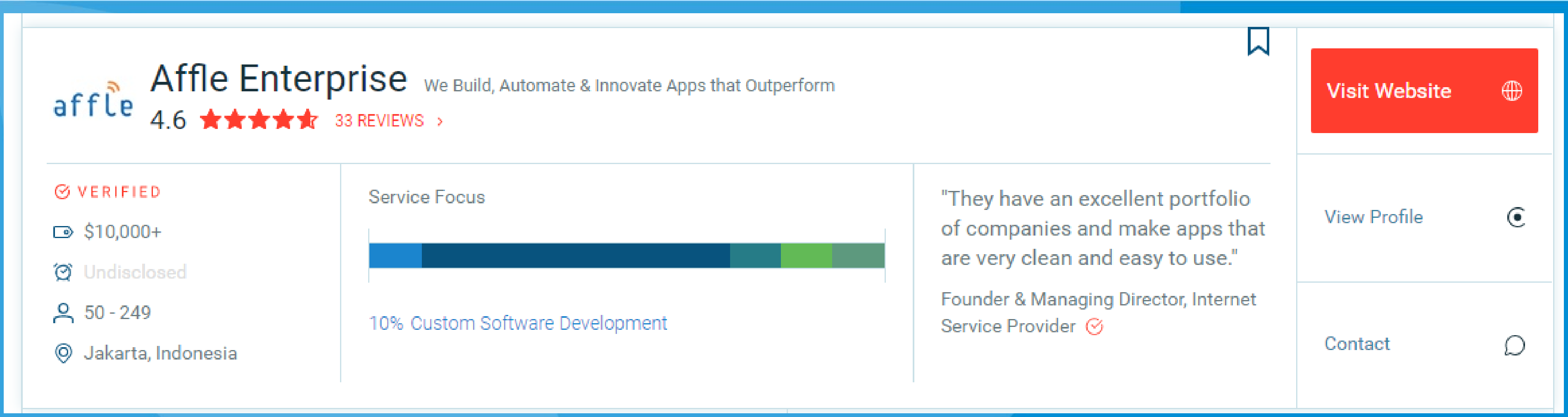


BINUS UNIVERSITY

Outsourcing

- Hire an external firm to create the system
  - Requires extensive two-way coordination, information exchange and trust
  - Disadvantages include loss of control, compromise confidential information, transfer of expertise
  - Carefully choose your vendor
  - Carefully prepare the contract and method of payment
- Contract types:
  - Time-and-arrangement: pay for all time and expenses
  - Fixed-price: pay an agreed upon price
  - Value-added: pay a percentage of benefits

People
Innovation
Excellence



Affle Enterprise We Build, Automate & Innovate Apps that Outperform

4.6 ★★★★★ 33 REVIEWS ›

VERIFIED

$10,000+

Undisclosed

50 - 249

Jakarta, Indonesia

Service Focus

10% Custom Software Development

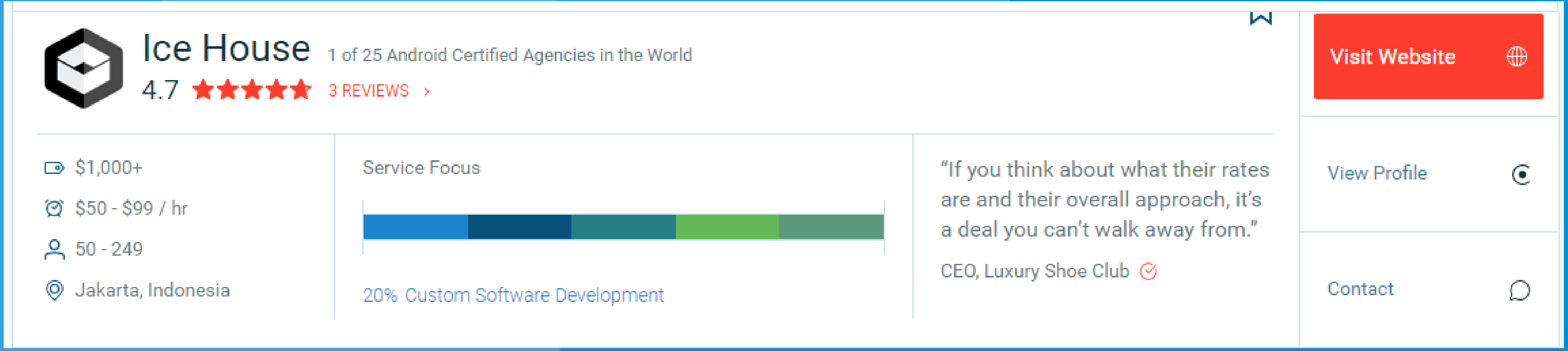
"They have an excellent portfolio of companies and make apps that are very clean and easy to use."

Founder & Managing Director, Internet Service Provider

Visit Website

View Profile

Contact



Ice House 1 of 25 Android Certified Agencies in the World

4.7 ★★★★★ 3 REVIEWS ›

$1,000+

$50 - $99 / hr

50 - 249

Jakarta, Indonesia

Service Focus

20% Custom Software Development

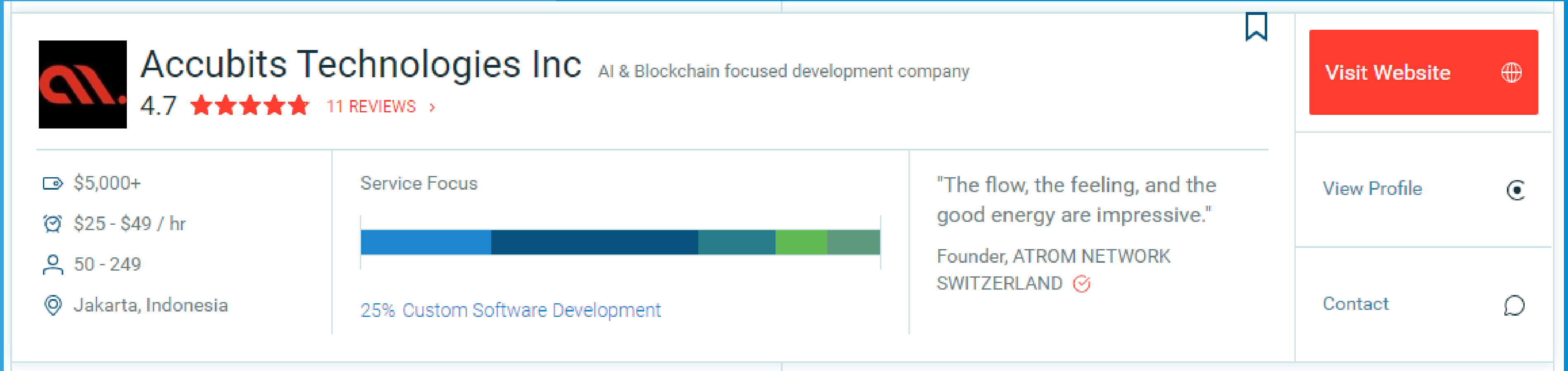
"If you think about what their rates are and their overall approach, it's a deal you can't walk away from."

CEO, Luxury Shoe Club

Visit Website

View Profile

Contact



Accubits Technologies Inc AI & Blockchain focused development company

4.7 ★★★★★ 11 REVIEWS ›

$5,000+

$25 - $49 / hr

50 - 249

Jakarta, Indonesia

Service Focus

25% Custom Software Development

"The flow, the feeling, and the good energy are impressive."

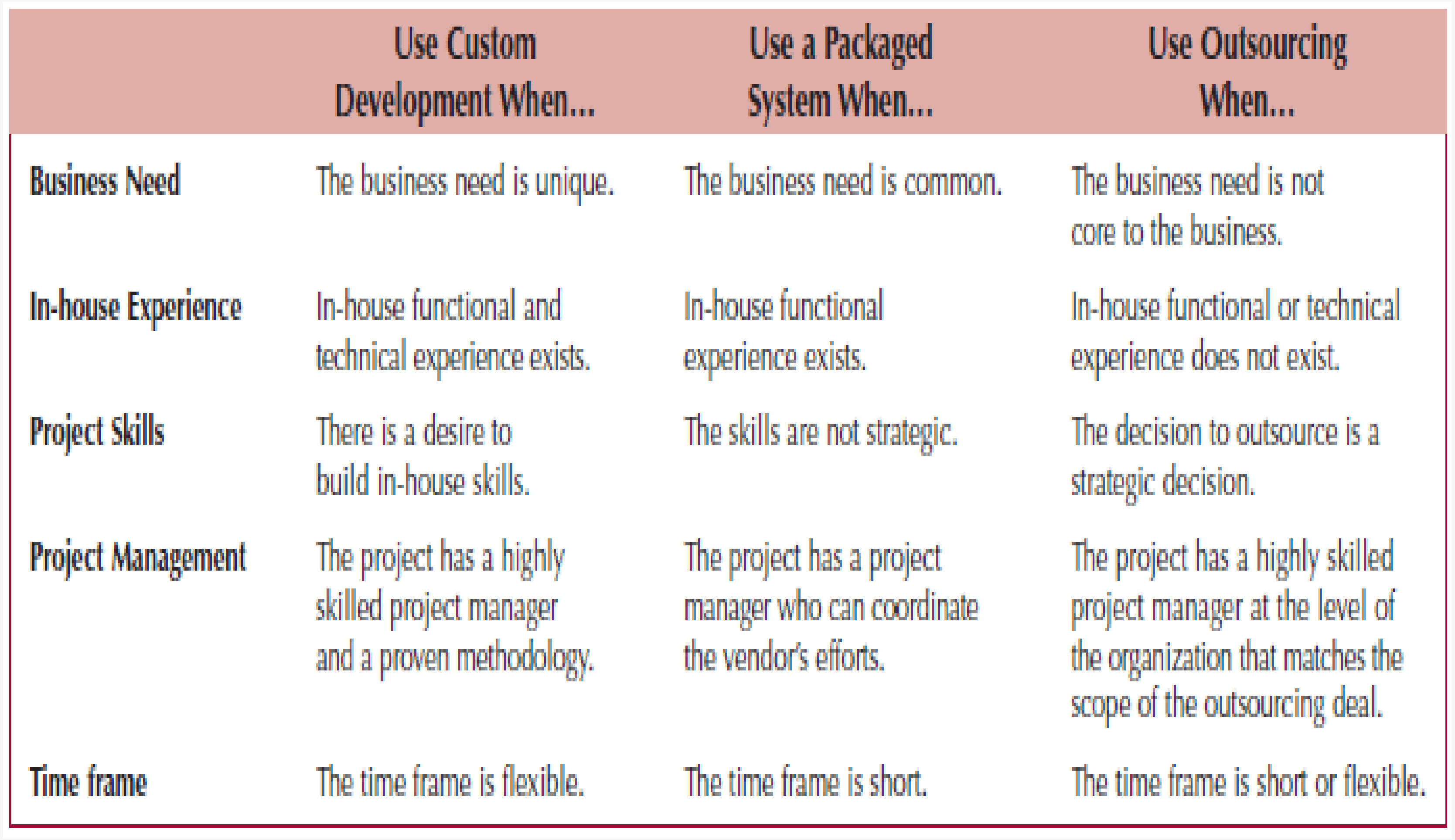
Founder, ATROM NETWORK SWITZERLAND

Visit Website

View Profile

Contact

Selecting a Design Strategy



	Use Custom Development When...	Use a Packaged System When...	Use Outsourcing When...
Business Need	The business need is unique.	The business need is common.	The business need is not core to the business.
In-house Experience	In-house functional and technical experience exists.	In-house functional experience exists.	In-house functional or technical experience does not exist.
Project Skills	There is a desire to build in-house skills.	The skills are not strategic.	The decision to outsource is a strategic decision.
Project Management	The project has a highly skilled project manager and a proven methodology.	The project has a project manager who can coordinate the vendor's efforts.	The project has a highly skilled project manager at the level of the organization that matches the scope of the outsourcing deal.
Time frame	The time frame is flexible.	The time frame is short.	The time frame is short or flexible.

BINUS UNIVERSITY

People
Innovation
Excellence

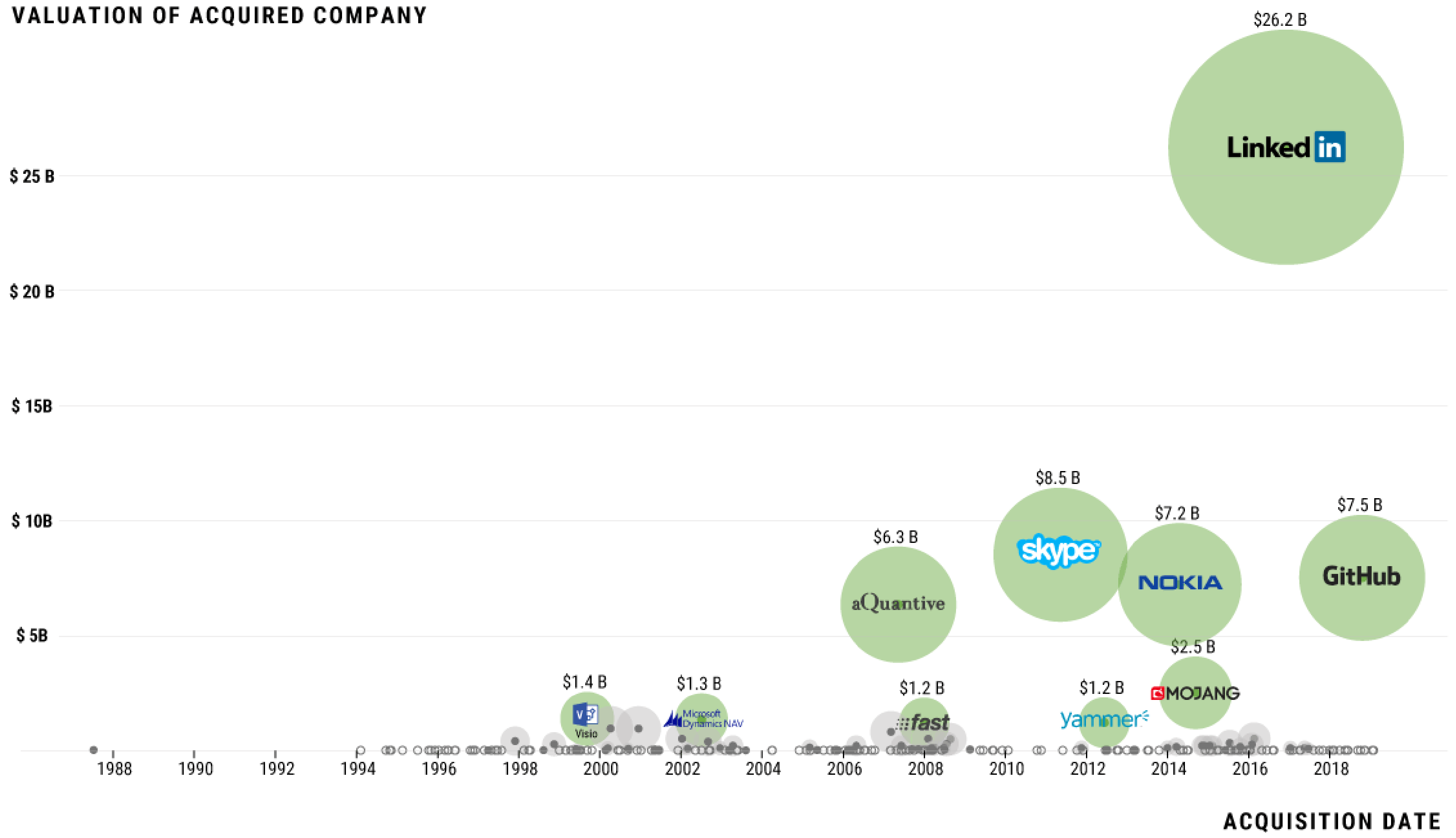
SELECTING AN ACQUISITION STRATEGY

SELECTING AN ACQUISITION STRATEGY

- Determine tools and skills needed for in-house development
- Identify existing packages that satisfy the users' needs
- Locate companies who can build it under contract
- Create an alternative matrix to organize the pros and cons of each possible choice
  - Incorporate technical, economic and organizational feasibility
  - Utilize an RFP or RFI to obtain cost & time estimates from potential vendors

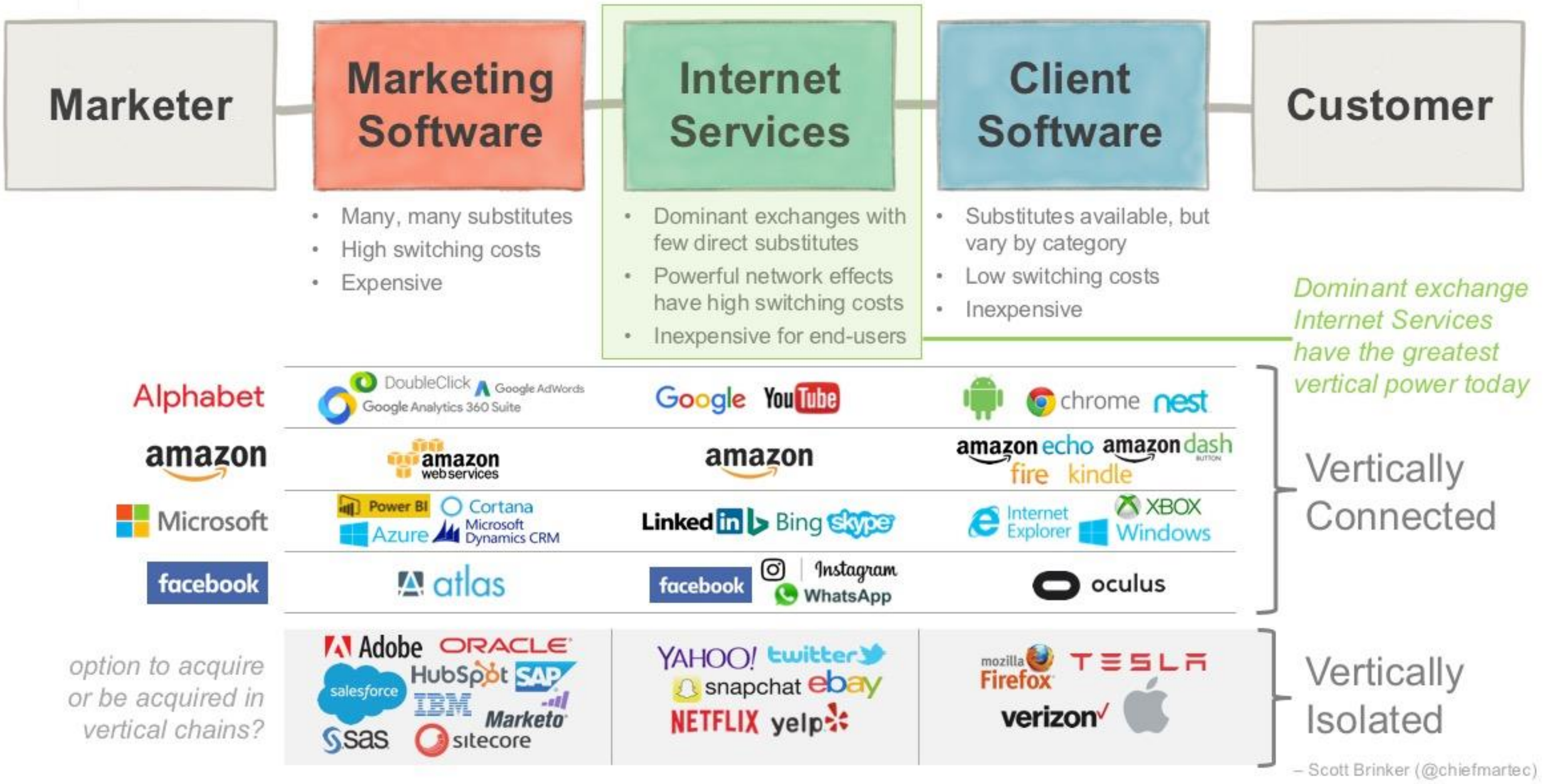
MICROSOFT'S ACQUISTION TIMELINE

Bubble size represents maximum valuation.
Clear bubbles represent undisclosed valuation.



CBINSIGHTS

Vertical Competition in the Digital Channel



Marketing Software:

- Many, many substitutes
- High switching costs
- Expensive

Internet Services:

- Dominant exchanges with few direct substitutes
- Powerful network effects have high switching costs
- Inexpensive for end-users

Client Software:

- Substitutes available, but vary by category
- Low switching costs
- Inexpensive

Dominant exchange Internet Services have the greatest vertical power today

option to acquire or be acquired in vertical chains?

– Scott Brinker (@chiefmartec)

People
Innovation
Excellence

SUMMARY

Summary

- Verifying and Validating the Analysis Models
- Evolving the Analysis Models into Design Models
- Packages and Package Diagrams
- Design Strategies
- Developing the Actual Design

References

Denis, Wixom,Tegarden. (2015). Systems Analysis and Design: An Object-Oriented Approach with UML. 5th edition. ISBN: 978-1-118-80467-4, John Wiley & Sons, Inc, Denver (USA)