Course: Software Engineering  
Multimedia University Cyberjaya,  

**ANSWERS TO ASSIGNMENT NO. 6**

1. In the article on Practical UML™: A Hands-On Introduction for Developers - by Randy Miller, at reference [http://bdn.borland.com/article/0,1410,31863,00.html](http://bdn.borland.com/article/0,1410,31863,00.html), you will find simple explanations of concepts in UML2 modeling used in software engineering. The core of the Unified Modeling Language is its nine (9) kinds of modeling diagrams listed below. In your own words, describe the diagrams and its use (i.e. when, where and in what instance you choose to specifically use it).

- **Use case diagrams**
- **Class diagrams**
- **Object diagrams**
- **Sequence diagrams**
- **Collaboration diagrams**
- **Statechart diagrams**
- **Activity diagrams**
- **Component diagrams**
- **Deployment diagrams**

**ANSWER**

**One possible place to understand the use of UML diagrams** (when and where) is from the famous reference e-book (PDF) named SWEBOK (Software Engineering Book of Knowledge at [http://www.swebok.org](http://www.swebok.org)). The latest version (2004) is available as Reading-62 on this course website.

Go to page 55 of 202 in the SWEBOK PDF file, Chapter 3 – Software Design, Section 5 – Software Design Notations. There are 2 sections on design notations, namely Structural Description and Behavior Description. The items below that marked blue and red are UML2 diagrams and non-UML2 diagrams, respectively.

**5.1. Structural Descriptions (static view)**

The following notations, mostly (but not always) graphical, describe and represent the structural aspects of a software design—that is, they describe the major components and how they are interconnected (static view):

- **Architecture description languages (ADLs):** textual, often formal, languages used to describe a software architecture in terms of components and connectors

- **Class and object diagrams:** used to represent a set of classes (and objects) and their interrelationships

- **Component diagrams:** used to represent a set of components (“physical and replaceable part[s] of a system that [conform] to and [provide] the realization of a set of interfaces” [Boo99]) and their interrelationships
Class responsibility collaborator cards (CRCs): used to denote the names of components (class), their responsibilities, and their collaborating components’ names.

Deployment diagrams: used to represent a set of (physical) nodes and their interrelationships, and, thus, to model the physical aspects of a system.

Entity-relationship diagrams (ERDs): used to represent conceptual models of data stored in information systems.

Interface description languages (IDLs): programming-like languages used to define the interfaces (names and types of exported operations) of software components.

Jackson structure diagrams: used to describe the data structures in terms of sequence, selection, and iteration.

Structure charts: used to describe the calling structure of programs (which module calls, and is called by, which other module).

5.2. Behavioral Descriptions (dynamic view)

The following notations and languages, some graphical and some textual, are used to describe the dynamic behavior of software and components. Many of these notations are useful mostly, but not exclusively, during detailed design.

Activity diagrams: used to show the control flow from activity (“ongoing non-atomic execution within a state machine”) to activity.

Collaboration diagrams: used to show the interactions that occur among a group of objects, where the emphasis is on the objects, their links, and the messages they exchange on these links.

Data flow diagrams (DFDs): used to show data flow among a set of processes.

Decision tables and diagrams: used to represent complex combinations of conditions and actions.

Flowcharts and structured flowcharts: used to represent the flow of control and the associated actions to be performed.

Sequence diagrams: used to show the interactions among a group of objects, with emphasis on the time-ordering of messages.

State transition and statechart diagrams: used to show the control flow from state to state in a state machine.

Formal specification languages: textual languages that use basic notions from mathematics (for example, logic, set, sequence) to rigorously and abstractly define software component interfaces and behavior, often in terms of pre- and post-conditions.
**Pseudocode and program design languages (PDLs):** structured-programming-like languages used to describe, generally at the detailed design stage, the behavior of a procedure or method

**For completeness sake and to avoid our confusion,** we will look at the separation of the Structural Description diagrams and Behavioral Description diagrams in UML2. And we shall refer to the actual UML2 Specification (latest ver 2.0 as of August 2005) which is available as Reading-63, also on this course website. We look at the real document (this is WRY's style ha..ha..ha..). Refer to the PDF file Reading-63, on page 675 of 709 (page 660 of doc) we see the figure below.

![Diagram]

The constructs contained in each of the thirteen UML diagrams is described in the Superstructure chapters as indicated below.

1. Activity Diagram - "Activities" on page 285
2. Class Diagram - "Classes" on page 21
3. Communication Diagram - "Interactions" on page 443
4. Component Diagram - "Components" on page 139
5. Composite Structure Diagram - "Composite Structures" on page 157
6. Deployment diagram - "Deployments" on page 189
7. Interaction Overview Diagram - "Interactions" on page 443
8. Object Diagram - "Classes" on page 21
9. Package Diagram - "Classes" on page 21
10. State Machine Diagram - "State Machines" on page 507
11. Sequence Diagram - "Interactions" on page 443
12. Timing Diagram - "Interactions" on page 443
13. Use Case Diagram - "Use Cases" on page 569
2. The Good Book provides a comprehensive idea of the process and transition relationships between the Analysis Model and the Design Model, through the process dimension as shown in Figure 9.4 (Chapter 9). Explain why the ER Model or the ER Diagram is not included in the figure. Hint: Go to the following URL http://www.smartdraw.com/examples/software-erd/department_full.htm

ANSWER

We know that Entity-relationship diagrams (ERDs) are used to represent conceptual models of data stored in information systems. One such diagram is shown below.

If you study the above ERD carefully, you will find that it is essentially a "class plus object" diagram in UML2. From the legend, the "Entity" is a class, the "Attribute" is an attribute, the Action is a relationship, etc.,

So in the Good Book, in Figure 9.4 (Chapter 9) the ER Model or the ERD is not included in the figure because class diagrams and data-flow diagrams have been included. It will be duplication (redundant) to put in the ERD.

3. Describe how we can use the different types of design patterns throughout the software design process. Why should we use them?

ANSWER

The Good Book in Chapter 9, on page 281 discussed using patterns in design. Design patterns can be used throughout the software design. There are a lot of examples of UML and non-UML diagrams in the Good Book, that is, if you browse through them. Ha..ha..ha.. Therefore, you should not have problems with
converting your project URS into design diagrams. Just search through the Good Book and the internet and look for them.

The different types of design patterns:

1. **Architectural patterns** – overall structure of the software. As an example for the Intelligent Home System, you may use the pattern in the Good Book, Figures 10.6, 10.7, 10.8 and 10.9, in Chapter 10. Why not?

2. **Design patterns** – patterns that address a specific element of the design. You can do “factoring” and produce a figure like 10.22. Factoring is a very popular word in software engineering, simply meaning, breaking down a big component into smaller ones with relationships to each other in some way. The Good Book states (page 312) “Factoring results in a program structure in which top-level components perform decision making and low-level components perform most input, computation, and output work.” And you can iterate (repeat several rounds) the factoring until you achieve the best solution. Or you can go to a design pattern book (e.g. Pattern-Oriented Software Architecture in 3 volumes) and study which pattern solves your problem. Why not?

3. **Idioms or Coding patterns** – language pattern when you do software coding that you can just follow or modify if need be to suit your situation. If you do Java buy a Java book with software codes, with CDs etc. Or just go to the tutorials or open source forums on the internet, and look for the software codes. There are Software Coding Standards available on the internet and in the course website (Lecture Notes and Reading-24) right? Why not?

**DESIGN PATTERNS ACCORDING TO SWEBOK (Reading-62, page 54 of 202)**

3.2. Design Patterns (microarchitectural patterns)

Succinctly described, a pattern is “a common solution to a common problem in a given context.” While architectural styles can be viewed as patterns describing the high-level organization of software (their macroarchitecture), other design patterns can be used to describe details at a lower, more local level (their microarchitecture).

- Creational patterns (for example, builder, factory, prototype, and singleton)
- Structural patterns (for example, adapter, bridge, composite, decorator, façade, flyweight, and proxy)
- Behavioral patterns (for example, command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor)

Why should we use design patterns? We use design patterns because a pattern is already “a common solution to a common problem in a given context.”

4. Relate our learning on real-time software applications in software engineering to our visit to the MMP/KVDT site with regards to the following subjects:
   - Business Operations to the importance of the URS
   - Class diagrams to electrical pumps, flow meters, pressure sensors, etc
- Statechart diagrams to the alarm monitoring screens, etc
- Component & Deployment diagrams to pumping stations, gantry etc,

Explain in your own words the four (4) direct relationships (actually many more) that have been identified above.

**ANSWER**

*In this case you must use the “Think” or seventh core principle in software engineering practice.*

1. **Business operations to the importance of the URS**
   The business operation of MMP/KVDT determines the software that have to be built for them, how they run their business, what they need to measure, what they need to monitor and control, how they like to control it, how they want to record it etc, meaning covering basically, the full fledged W5HH principle about their business. If this is not captured in the URS properly (high quality URS document), the software cannot be built to satisfy their requirements.

2. **Class diagrams to electrical pumps, flow meters, pressure sensors, etc**
   These devices are objects in the software for MPP/KVDT. The electrical pump for example, has attributes (ON, OF, UNKNOWN, flow rates, rotation speed, etc) and operations (START, STOP, ALARM STATUS, CHECK STATUS etc). You can model this using UML.

3. **Statechart diagrams to the alarm monitoring screens, etc**
   We saw in our visit different flow rate settings for the pumps, for example (LOW, LOW-LOW, NORMAL, HIGH, HIGH-HIGH, etc). If the flow goes out of range of the NORMAL state settings, we will hear beeping sounds and flashing alarms on the monitor. This can actually be modeled using statechart diagrams in UML.

4. **Component & Deployment diagrams to pumping stations, gantry etc,**
   There are 3 different pumping stations within the MPP/KVDT system located at various places. We can show these through components and deployment diagrams in UML, e.g. at each pumping station - what components are there and how they relate to the overall structure at MPP/KVDT.

5. Describe the three (3) characteristics: concurrency, persistence and distribution in the context of the architecture diagrams below.

**REFERENCES:**
- [http://java.sun.com/j2ee/white/connector.html](http://java.sun.com/j2ee/white/connector.html)
The Mincom Ellipse Service Oriented Architecture (SOA)

The Application Server for the Mincom Ellipse SOA above is the J2EE architecture using either the IBM Websphere or the BEA WebLogic application. A generic J2EE application architecture is shown below.

Figure 1-1 Multitiered Applications
The three (3) characteristics of concurrency, persistence and distribution were discussed in the Good Book in Chapter 10, page 296 under Architectural Patterns.

**Concurrency** is about software applications handling multiple tasks in a manner that simulates parallelism. In simple terms, many simultaneous requests to the software for some service the software must perform or respond (all at the same time, that’s why the word is concurrent). How can the software handle this? You must introduce a scheduler, like a transaction server or processor (explained in class). Therefore, in the context of the architecture diagrams above, the transaction servers can either be one of IBM-CICS or the BEA-Tuxedo. (You search on the internet for these 2 terms and learn on what it is all about).

**Persistence** is about information or data that continues to exist after the software process have been completed or the computer had been switched OFF. Simply said, persistence is about permanent storage of data or information, otherwise data that have been generated or calculated in the computer RAM (memory) will vanish after the process completes (garbage collection) or when the computer is switched OFF. So we will be jumping like crazy, if we lose important information. So people created computer or electronic databases, files etc that can be stored permanently on devices like hard disks, floppy disks, tapes, thumb drives etc, which later on you can read back if you wanted. The information is called persistent data because it persists forever on these devices (unless you delete them or the media deteriorates). Therefore, in the context of the architecture diagrams above, the persistent storage is the database in the EIS Tier (Database Server Machine).

**Distribution** is about addressing the manner in which systems or components within systems communicate with one another in a distributed environment. In our case, we spent 2 sessions in class explaining about software deployment through various communications (typically networked TCP/IP) over LAN, WAN, with routers, firewalls, proxy servers, VPN, Citrix Metaframe etc,. Therefore, in the context of the architecture diagrams above, the deployment of the software is the communications by the Integration Broker Hub (blue circle – for the top diagram) and the TCP/IP (understood) by the Client Tier, Web Tier, Business Tier and the EIS Tier (J2EE architecture - for the bottom diagram).