**CIS 4328 – Information Systems Senior Project II**

**Project Deliverable 4**

**Due Date:** January 29

**Relevant Disciplines:** Design and Analysis, and Construction

**Artifacts to be delivered:**

1. Data Flow Diagram – Context Diagram
2. Data Flow Diagram – Level 0 Diagram
3. Architecture model
4. Detailed Design
   1. Package diagram
   2. Class diagram for each package
   3. Sequence diagram (optional, depends upon your project needs)
   4. Deployment diagram (optional, depends upon your project needs)
   5. Security mechanism (optional, depends upon your project needs)
5. Sprint Backlog
6. Fourth Iteration Product Release
7. Update artifacts delivered in earlier project deliverables
8. Data Flow Diagram (DFD) – Context Diagram

Data Flow Diagram (DFD) is a technique that depicts the business processes and the data that passes between them. Although the name data flow diagram implies a focus on data, this is not a case. The focus is mainly on the processes (i.e., set of business activities) that are performed. A process model is a formal way of representing how a business system operates. It illustrates the processes that are performed and how data moves among them. A process model can be used to document the current system (i.e., system as is) or the new system being developed (i.e., to be system). Elements of the data flow diagram are:

* External entity – is represented by a square
* Process – is represented by a rectangle with rounded corners
* Data store – is represented by an open-ended rectangle
* Data flow – is represented by an arrow

Context Diagram – The first DFD in every business process model is the context diagram. As the name implies, the context diagram shows the entire system in context with its environment. All process models have one context diagram. The context diagram shows the overall business processes as just one process and shows data flows to and from external entities.

Data stores are not usually included in the context diagram unless they are “owned” by systems or processes other than one being documented. For example, an information system used by the university library that records who have borrowed books would likely check the registrar’s student information database to see if a student is currently registered at the university. In this context diagram, the registrar’s student information data store could be shown on the context diagram, because it is external to the library system, but used by the library information system. However, alternatively, the “Registrar’s Student Information System” can be shown as an external entity.

To create a context diagram, simply draw one process symbol for the business process or system being modeled. Read through the use cases and add the inputs and outputs listed, as well as their sources and destinations. Usually, all the inputs and outputs will come from or go to external entities. Because there are sometimes so many inputs and outputs, we often combine several data flows into larger data flows.

Create **the context diagram for the new proposed information system** to be developed by your team.

For developing Data Flow Diagrams using Visio, select New, then “Software and Database,” then “Data Flow Model Diagram.” For Process symbol, in Visio search for “2 part function”. For Data Store, in Visio search for “Data Store.”

1. Data Flow Diagram (DFD) – Level 0 Diagram

A Level 0 diagram shows all the major high-level processes, data stores, external entities, and data flows among them. The purpose of the level 0 diagram is to decompose the context diagram and show more detail about the processes, how they are interrelated and data flow inside the system. All the process models have one and only one level 0 DFD.

To create a Level 0 diagram, each use case is considered as a process in the information system. Based on the use cases, include appropriate external entities, data store, and data flow between them.

Develop the **Level 0 diagram** **for the new proposed information system** to be developed by your team.

1. Architecture model

Like any other engineering disciplines, before designing software, certain architectural decisions need to be made. Drawing from civil engineering, the bridge designer would have to decide whether a suspension bridge, a cantilever bridge, a cable-stayed bridge or some other type should be chosen to satisfy the requirements. Following the above analogy, software designer (or architect) also needs to decide on the choice of architectures.

Architecture is equivalent to design at the highest level. The clear specification of software architectures, important for all applications, is indispensable in the case of multi-person development jobs. This is because large applications must be designed and implemented in parts (“modularized”) and then assembled. Architecture selection provides this modularization.

Deciding on the best architecture depends upon functional requirements captured in the Use-Case Model and non-functional requirements captured in the Supplementary Specifications. The decision on architectures is influenced by constraints imposed by the environment in which the software must operate; by the need to reuse existing assets; by the imposition of various standards; by the need for compatibility with existing systems, and so on. There are the preexisting set of architectural principles and policies which will guide the development, and which need to be elaborated and reified for the project. You can choose among following architectural patterns: the multi-layer architectural pattern, the client-server and other distributed architectural pattern, the broker architectural pattern, the transactional processing architectural pattern, the pipe and filter architectural pattern, the model view controller (MVC) architectural pattern, the service-oriented architectural pattern, and the message-oriented architectural pattern. Each pattern allows you to design flexible systems using components that are as independent of each other possible.

**Select appropriate architecture(s)** to be used for designing and developing your application. Describe the selected architecture(s). **Justify your architecture selection** with brief descriptions of the responsibilities and correspondences to your application for each architectural component. Provide rationale for your architectural choices based on relevant design principles considered, and outlining critical issues and trade-offs that were considered. Describe the software requirements and objectives that have some significant impact on the architecture; for example: security, privacy, use of an off-the-shelf product, portability, distribution, reuse, and so on. Also describe any other the special constraints consider for decision making: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.

More details on the software architectures can be found at:

<http://sce.uhcl.edu/helm/rationalunifiedprocess/process/workflow/ana_desi/co_swarch.htm>

1. Detailed design

For the detailed design, you will need to provide a complete specification of the entire software: every package, subsystem, and class. It will probably make sense to organize this according to the modules/packages identified in the package diagram.

A fundamental principle of software development is dividing a large system into subsystems or packages. A good decomposition makes the system more understandable and can be maintained easily. A package diagram consists of a small collection of packages, each of which can be expanded to larger set of classes. A package is a collection of modeling elements such as classes, instances, text, or other packages that are grouped because they are logically related. In the package diagram, a package is shown as a box, with a smaller box attached above its top left corner.

The overall structure of the design model can be understood easily, by grouping design classes into packages, preferably according to adopted architectural style, then showing how these packages related to one another. The package diagram shows the dependencies among the classes that belong to different packages. Two packages are dependent if there is a dependency between classes from different packages. Two classes are dependent, if one class sends a message to the other class, has data needed by the other class, or uses the other class as part of its parameter for a method. As with any good design, we will strive to minimize number dependencies between packages.

**Create a UML package diagram** that provides an overall structure of the design model for your application. Along with the diagram, you should briefly describe the purpose of the subsystem/packages identified in the package diagram. You should provide a rationale explaining your design decisions for organizing packages according to architectural style.

Guidelines for creating package diagrams can be found at:

<http://sce.uhcl.edu/helm/rationalunifiedprocess/process/modguide/md_despk.htm>

**Create UML class diagrams for each package** identified in the package diagram. Provide internal details for all classes include attributes (along with details of its visibility and data type) and methods (along with details of its visibility, return type, parameter listing, and exceptions thrown). You must include inheritance, aggregation/composition, association, and dependency relationships between classes as applicable. **If you have used any design patterns**, indicate this with an annotation in your class diagram. Along with the class diagram, provide a brief description of each class.

Guidelines for creating class diagrams can be found at:

<http://sce.uhcl.edu/helm/rationalunifiedprocess/process/modguide/md_desmd.htm>

If a particular class or package plays some vital role then perhaps you need to explain its workings in more detail, in particular, if it differs in some way from the most obvious way of doing things. You may consider including **pseudo-code for the more complex methods**. You may consider including **UML sequence diagrams** to clarify complex interactions between classes in the respective subsystem/package. Sequence diagrams provide details on the chronological order in which classes interact via messages exchanged and other event occurrences.

How to decide on the appropriateness for usage of the sequence diagrams? Use your judgment to determine which subsystem/packages are complex. As a guideline, consider including a sequence diagram for subsystem/package that has more than three classes or has more than five interactions (i.e., message exchanges or other events).

Guidelines for preparing sequence diagrams can be found at:

<http://sce.uhcl.edu/helm/rationalunifiedprocess/process/modguide/md_seqdm.htm>

If appropriate, i.e., hardware and software components of your application are distributed and communicate over a network, then include a **deployment diagram**.

A deployment diagram shows runtime configuration of hardware and where various instances of components that reside on them. A computational unit such as a computer, a processing card, a sensor or a device is represented as a node in the deployment diagram. A node is represented as a three-dimensional box. Each node can include one or more run-time software components. Connections between nodes show how communication takes place.

More details on the deployment diagram can be found at:

<http://sce.uhcl.edu/helm/rationalunifiedprocess/process/artifact/ar_dplmdl.htm>

Given the importance and complexity of security functions, most security controls in a typical information systems development are implemented by configuring security software underlying operating system and DBMS. Thus, information systems developers are limited mechanisms for securing communication between components and restricting access to confidential data. The mechanisms used for securing communications between users and components are secure channels, authentication, the integrity of the message and confidentiality. The access control mechanisms are used for restricting access to information and resources to authorized individuals.

If a particular software component is vulnerable to breach of system security, then **develop appropriate secured communication and access control mechanisms and policies** to ensure that the software component minimizes or eliminates the potential for breaches of system security.

If there are **any special features of your design,** then include additional descriptive text as necessary to explain those design features.

1. Sprint Backlog

A brief weekly meeting (weekly scrum) should be held by the team to inspect progress made on Sprint Backlog items. Report verbally as a team and individually to the instructor on progress made on the project.

For this deliverable, your team is expected to provide the sprint backlog for January month iteration and sprint backlog plan for February month iteration.

You will be documenting Sprint Backlog using JIRA system. Please note that each student is expected to add and maintain the progress made on relevant Sprint Backlog items in the JIRA system.

Remember to update product backlog and user stories in Kanban board in JIRA.

1. Fourth Iteration Product Release

As a part of this deliverable, your team has to construct major portions of the software functionality. While none of the functionality is expected to be fully constructed, it is expected that core functionality for each use case to be working and important non-functional requirements also completed. Each team member is expected to work on at least one use case. Thus, if your team has four members, then at a minimum demonstration of four software functions will be expected.

The deliverable format for fourth release prototype would be a **demonstration of the prototype with the instructor.** Each team is expected to schedule an appointment with the professor during the last week of January to demonstrate the fourth iteration product release from January 27 to 31. Each team member would be responsible for demonstrating their portion of the prototype. A signup sheet will be available via Canvas during the third week of January.

During the demonstration, for each software function, you are expected to address the following:

* Briefly describe specific use case and related non-functional requirements
* Briefly describe design decisions made relevant to the functionality supported by the use case
* Demonstrate the prototype developed so far to satisfy requirements

Please note that the prototype may not be complete, but it should still be executable. I do not expect a complete and fully functional prototype, but source code base should be addressing the stated requirements and progressing well to finish it on time.

Please submit other relevant artifacts via Canvas. Remember to upload any of the other relevant updated document such as use case specifications, class diagrams, sequence diagrams, and software development plan artifacts.