Software Architecture Assessment Instrument

Assessor Name(s): _________________________
Project Team #____

Purpose: The purpose of this instrument is to help assess the Software Architecture projects carried out by the students in the cs471 ’04 class at the University of Western Ontario. The assessments will be used in an empirical study on software architectures.

Background: Bredeyemer consulting states that a Software Architecture should be:

- **Good** — i.e., it is technically sound and clearly represented.
- **Right** — i.e., it meets the needs and objectives of key stakeholders.
- **Successful** — i.e., it is actually used in developing systems that deliver strategic advantage.

This assessment instrument is meant to assess the Architecture with respect to only the first two attributes listed since the architectures in question will not be implemented, or judged, in any real-world setting. The minimum amount of work that the instrument is based on is taken from the document “Minimum Project Documentation”. This document was given to the students to state the expected type and quantity of the various architecting artifacts that should be in the final product. Also, the students used documentation templates from the course textbook to complete their project so these templates are also a source for this instrument.

Instructions:

Use the accompanying Microsoft Excel template to rate each statement according to the following scale except where otherwise instructed. Each statement refers to the ‘level of agreement’ of the statement. The scale is: 6 – very strongly agree; 5 – strongly agree; 4 – mostly agree; 3 – neither agree nor disagree; 2 – mostly disagree; 1 – strongly disagree; 0 – very strongly disagree. The template also has two columns for “Evaluator Confidence”, “Rationale”, and “Suggestions”. In the confidence column, mark your confidence level from 1-10 (where 1 is very little confidence and 10 is extremely confident) for each of your responses. In the rationale box, provide where necessary a

---

1 The format and style of this instrument is based on an assessment manual from the Social Sciences in the field of Psychology. The reference for the manual is: J. Fortin and C. Cuerrier: Evaluating a Mentoring Program, Canadian Cataloguing in Publication Data, 2003.


justification to your rating. The suggestions column is where you can input feedback about the instrument for any given statement.

1. Domain Work

The domain work includes tasks that are based on understanding the requirements and the domain of the system. They deal with issues that are not part of the design of the system, but more with modelling and understanding the problem definition of the system to be built.

1.1 The context diagram(s)

1. Minimum one context diagram showing overall system within its environment (2 – Has more than one context diagram for different sections, 1 – meets this requirement, 0 – no context model)

2. Models show links to all external institutions and entities in the problem domain (Completeness) (should be 4 or 5 links in the ideal solution)
   - The Banking system
   - ATM, internet phone banking, direct staff access, and automated phone banking
   - Other financial institutions such as: other banks, stock exchange, government institutions
   - Other users such as managers, maintainers, and

3. Model(s) (possibly explained by supporting description) are easy to read and understand

4. The context model(s) are too complex and detailed for the level of abstraction they are representing.

1.2 Use Cases (Enter N/A if there is no work on the use cases)

1. Existence of use cases for key functionality such as withdraw and deposit funds, transfer funds, check account balance, and edit personal information.

2. Clear and logical models

3. Use cases are rooted in the requirements

4. The use case models, components, or links are superfluous

5. Appropriate labelling of links between elements in the models

2. Requirements-Architecture Work

Kazman, Bass and Clements identify the quality scenario work and tactics determination to be the tasks that lie in the link between Requirements and Architecture. Other researchers have proposed other RE-SA methods that would involve a different set of RE-SA tasks, but since the subjects of our study used Kazman et al.’s ADD process, we are using their definition of RE-SA tasks.

2.1 Quality work (attributes and scenarios) (Enter N/A if there are no quality scenarios)
1. **Explicit quality scenarios that are representative of the problem domain**
2. **For each scenario, a maximum of six elements of quality scenarios are “reasonable”:** (for each of the elements of a scenario, give a ‘1’ if it is reasonable and a ‘0’ if it is not. Total will be given out of six)
   - Source
   - Stimulus
   - Environment
   - Artefact
   - Response
   - Response measure

### 3. Architecting

Architecting forms the bulk of the tasks the subjects had to perform. It involves high-level design focused on creating the structure of the system, and the relationships within this structure. Everything from the conceptual models to the corresponding documentation is included in Architecting tasks.

#### 3.1 Architectural Structure

#### 3.1.1 Module view

“In the module view, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. There is less emphasis on how the resulting software manifests itself at runtime. The module structure allows us to answer questions such as, “What is the primary functional responsibility assigned to each module?” What other software elements is a module allowed to use? What other software does it actually use?”

1. **Appropriate use of architectural patterns (either selected or created) to satisfy quality attributes and tactics.** Please refer to BCK (chapter 3-7) textbook for discussion on patterns and how they relate to quality attributes and tactics.
2. **Three levels of decomposition for the main functions:** (3 – three levels, 2 – two levels, 1 – 1 level, 0 – no levels)
   - deposit (cheque or cash)
   - withdraw
   - check balance
   - transfer funds between accounts
   - view/print recent transactions
   - pay bills
   - edit personal information

3. **At least two levels of module decomposition for “other” features such as:**
   - 3.1 user interface functionality: (2 – two levels, 1 – one level, 0 – no level)
     - Representation of differences between ATM, Internet Banking, Staff Access, and Telephone Banking
3.2 security features: (2 – two levels, 1 – one level, 0 – no level)
   - Encryption/decryption, authentication, and audit trail, among others
4. At least one level for database functionality (1 – yes, 0 – no)
5. The different levels of decomposition are consistent with one another:
   - 5.1 Features well-defined modules whose functional responsibilities are allocated on the principles of information hiding and separation of concerns
   - 5.2 Sub-modules coverage of the parent module functionality (possibly also include the info-hiding/separation-of-concern question of #11 here)
   - 5.3 Sub-module links show the data flowing up/down the links (see Sommerville’s SEng book on module decomposition)
   - 5.4 The sub-module dataflow (in/out) are consistent with the in/out dataflows of the parent (see Sommerville/Pressman SEng books)
6. Diagrams are readable (not “messy”)
7. Elements in a model are given appropriate names

3.1.2. Deployment view

“This view is meant to show the relationship between the software elements and the elements in one or more external environments in which the software is created and executed. They answer questions such as:
   - What processor does each software element execute on?
   - In what files is each element stored during development, testing, and system building?
   - What is the assignment of software elements to development teams?”

1. Minimum one deployment structure for a particular level of decomposition (2 – goes beyond the required one section, 1 – one section, 0 – none)
2. Appropriateness of the patterns selected/created with respect to the quality attributes and tactics
3. Deployment view is centred on appropriate issues (such as, network topology, assignment of software units to processors, middleware, etc.) based on fulfillment of quality attributes (1 – yes, 0 – no)
4. Understandability – the model is conceptually clear
5. Readability – the model is well-labelled and clear, use of key for notation
6. Logical displacement and labelling of links (relations) between the elements

3.1.3 Component and Connector view

“In this view, the elements are runtime components (which are the principal units of computation) and connectors, which are the interactions among the components. Component-and-connector structures help answer questions such as:
   - What are the major executing components and how do they interact?
   - What are the major shared data stores?
   - Which parts of the system are replicated?
   - How does data progress through the system?
• What parts of the system can run in parallel?

1. **Minimum one C&C structure section for a particular level of decomposition (2 – goes beyond the required one section, 1 – one section, 0 – none)**
2. **Appropriateness of the patterns selected/created with respect to the quality attributes and tactics**
3. **C&C view is centred on an appropriate issue based on fulfillment of quality attributes (e.g. showing concurrency for performance, timing properties, data-flow, etc.) (1 – yes, 0 – no)**
4. **Understandability – the model is conceptually clear**
5. **Readability – the model is well-labelled and clear**
6. **Logical displacement and labelling of links (relations) between the elements**

### 3.2 Overall Architecture

1. **Buildability:**
   a. Architecture amenable to be assigned to separate development groups for implementation and, subsequently, amenable to incremental integration and incremental testing.
2. **The architecture depends on a specific version of a commercial product (1: yes, 0: no)**
   a. If yes, architecture is structured so that changing to a different product is straightforward and inexpensive
3. **The various views (module, C&C, and deployment) all map to each other in a seamless, non-conflicting way. They depict different aspects of the system.**

### 3.3 Documenting an Architecture

#### 3.3.1 Interfaces

1. **Completeness – has interface description for the lowest levels of decomposition of the main features listed below (put a 1 for exist, 0 for not described)**
   a. Print Reports for bank manager
   b. withdraw money
   c. deposit money
   d. transfer funds
   e. check balance
   f. cancel card
   g. postdate transactions
   h. pay bills
   i. order cheques and bonds
   j. edit personal information
   k. request stop payments
2. **Interface description includes public services of a module**
3. **Interface description includes information an element needs in order to perform a function**
4. Interfaces are accurate and correct to the extent they can be at this level of abstraction
5. Interfaces are written in a format (can be any) that is readable and understandable

3.3.2 Behaviour

1. All the common functionality is represented using sequence, state, and/or activity diagrams (completeness)
2. Behaviour diagrams maintain consistency with the rest of architecture
3. Redundancy (functions that are similar in behaviour, such as withdraw, deposit and check balance, are not represented multiple times)
4. Models are technically correct
5. Architectural behaviour is depicted (meaning the behaviour across elements, not within a given element) (3 – all the time, 2 – most of the time, 1 – very little, 0 – no architectural behaviour)
6. Diagrams are clear and easy to read

3.3.3 Descriptions

This section is for the textual descriptions of the views. The elements and their relations are described to complement the graphical models that are given.

1. Sound grammar and spelling
2. Describes enough information to understand the system at this level of abstraction
3. Completeness – existence of descriptions of all elements and their relations

3.3.4 Architecture Background (rationale, assumptions, analysis of results, and design alternatives)

This section contains all the reasoning description about the corresponding sections of the architecture. Items such as rationale, assumption, analysis of results, and design alternatives should all be detailed in this section.

1. Rationale is based on quality attributes trying to achieve and how (the tactics)
2. Quality attribute tradeoffs are made explicit
3. Discusses possible design alternatives and why they were dismissed
4. Sound grammar and spelling
5. Appropriate explicit assumptions should be documented for each view

3.3.5 Overall Documentation

1. Existence of: (enter 1 for yes, 0 for no for each of the following documentation elements)
   a. Page numbering
b. Table of contents
c. Section headers
d. Glossary of terms
e. References are used when necessary

2. Documentation across views section (see pages 215-218 in BCK course textbook)
3. Models have key to describe the notation used for modelling.
4. Consistency of documentation across all sections. Different individuals might be responsible for different sections of the documentation, so is there differences in the format, structure, or writing style of the various sections?
5. The documentation is well structured, organized and clear