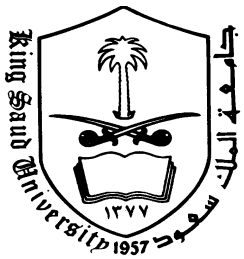

King Saud University
College of Computer and Information Sciences
Department of Software Engineering

SWE 497 – Graduation Project Evaluation Form

Grades Distribution (Assigned by Advisor)

Student Name: _____
Student Id: _____
Project Title:
_____

Project Advisor: _____

Marks Obtained:	**/100**

Item	Marks	Justification
A1: Written Report [45%]		
Overall Organization and Presentation	**/5**	
English	**/5**	
Technical Content		
Abstract/ Introduction/ Problem Definition	**/4**	
System design update [updated requirements, use cases, class diagram, architecture]	**/6**	
System implementation [code view, logic description, walkthrough]	**/11**	
Testing	**/10**	
Conclusions and Future Work / Ethics principles adopted [proper references and citation, no plagiarism]	**/4**	
Sub-Total (A1)	**/45**	
A2: Oral Presentation [15%]		
Clarity of Purpose	**/2**	
Technical Content [Identify problems faced/ solution approaches/ depth of concept]	**/5**	
Presentation Skills [Spoke clearly/ made eye contact/ stayed in time limit/ looked at audience]	**/5**	
Discussion [Participated in question answering/ interactive]	**/3**	
Sub-Total (A2)	**/15**	
A3: Competency, Dedication, and Team Work [15%]		
Regular meeting with advisor/Team work	**/5**	
Timely submission of required deliverables	**/10**	
Sub-Total (A3)	**/15**	
A4: Project Demonstration [25%]		
Demonstration of system functionality	**/25**	

Sub-Total (A4)	**/25**	
Total Marks Earned [A1+A2+A3+A4]:	**/100**	

Comments by Advisor:

Name of Advisor: **Signature**: **Date**:

Checklist SWE 497 Final Report Evaluation

The following are characteristics of excellent work on the corresponding key criteria in the evaluation of the written report part of SWE 497 – Graduation Project Evaluation Form.

Overall Organization and Presentation

English

- Report should be coherent and well organized following the provided report template.
- Grammar, spelling, punctuation and formatting should be flawless, which allows the reader to focus on the message.
- Figures and tables should be numbered appropriately and captions should be used to explain the corresponding figures and tables.

System Analysis

Requirements:

- Important requirements should not be missed.
- Requirement statements should be clear (there should be no ambiguity).
- Requirement statements should be written using a consistent style.
- Non-functional requirements should be measurable (fit criteria should be defined).
- Requirements should be well-organized. Merely listing a large number of requirements without any organizing effort should be avoided.
- Types of requirements should be defined (functional, non-functional, and design constraints)
- Requirements should be abstract (avoiding making unnecessary design decisions).

Use Case Model:

- All use cases should be found. The elaborated use cases should meet all functional requirements.
- Use cases should have unique, intuitive, and explanatory names.
- The UC diagram should use correct UML notation.
- Use case relationships should be correct (e.g., *includes* and *extends* relationships).
- Use case descriptions are not required.

Design and Architecture:

Architecture Design:

- Logical layers/subsystem design should be clearly defined.
- The design should show relationships/connections between layers/subsystems.
- Appropriate architectural style should be chosen. The choice should be justified.
- Alternative styles should be discussed (if applicable).
- Constraints that affect the way the architecture can be implemented should be discussed (if applicable).

Detailed Design:

- Detailed information of each class should be clearly defined such as:
    - Data attributes
    - Operations (parameter and return types should be defined)
    - Data structures
    - Algorithms (if applicable)
- Database design should describe the necessary tables and columns.

Testing:

- The test approach should be thoroughly described. For each major set of features, the approach should specify the major activities, techniques, and tools which will be used.
- Functional testing should include fairly enough test cases.
- Some functional testing should be derived from use cases. Some test cases should be generated to cover enough scenarios of some use cases.
- Some functional testing should be derived based on inputs for some screens. For each input, equivalence classes should be defined and boundary values should be used to define the test cases.
- Some unit testing and code coverage (white box testing) should be demonstrated.
- Some usability testing should be demonstrated.
- A tool such as JUNIT, Clover, or Code Cover should be used for unit testing.
- Screen shots should be used to show results for some test cases.
- Screen shots should be used to show the output of applying the testing tools.

System Implementation[1]:

- Technologies used should be adequately described. These include database systems, programming languages, development platforms, runtime environments, servers, operating systems, etc.

- Mapping of code to requirements or use cases should be adequately described. Mapping description should include the names of methods or classes that implement the corresponding requirement or use case.

- In the Implementation Details subsection, the students should present a walkthrough through the main program logic. The students should include several screen shots showing the GUI interface and code snippets showing the program logic.

- The walkthrough should show the starting point and then should cover the program logic for the most important functions.

- The walkthrough description should be enough to help the reader understand the source code.

[1] This section is the most important section in the report.