Task/Criterion (Note: each level reached if conditions met)	Preconditions	None	Unacceptable	Poor	Fair	Good	Excellent
Task 1: Justifying decisions (10)		(0) Not attempted	(2) A good amount of effort at attempting the required task(s), but mostly incorrectly.	(4) A good amount of effort at attempting required task(s) with mostly correct answers, but also some important mistakes in the reasoning, and understanding of the APIs and their alternatives unconvincing.	(6) Correct answers to both tasks, with very small mistakes in reasoning. Also, providing a few good links or references.	(8) All answers correct, with very small mistakes in reasoning. Providing links or references to support all statements.	(10) All answers correct and strongly supported by provided links and references, with reasoning showing having done a good amount of reading and having a good understanding of the APIs and alternatives.
Task 2: Construct code (25)		(0) Not attempted	(5) The code shows some effort, but it mostly does not match the provided class diagram and OR a considerable amount of it was not written (i.e. it covers few use cases from the team's task).	(10) The code does not correctly (i.e. meeting requirements) cover >= half of the required use cases OR it covers more (not all) but quality* is very low (i.e. unreadable),	(15) The code covers the use cases and mostly meets requirements. It matches the class and sequence diagrams or explanations are provided for very small differences to them. Quality could	(20) Like 'Fair' but only small aspects of code quality could be improved. Reasonable catching of invalid inputs using assertions.	(25) Like 'Good' but code fully meets requirements, of very high quality and impresses by extra effort in catching faults or invalid states OR writing how team would

* As described in Lecture 14.

				even if those that it covers do mostly respect the class and sequence diagrams.	be improved in some important aspects OR insufficient effort put into catching faults or invalid states using assertions.		consider non-functional requirements OR writing about code smells and how team would refactor OR other aspects going beyond the main instructions.
Task 3: Create system-level tests (20)	**At least "Poor" in Task 1 for reason **after** OR**	(0) Not attempted	(4) Some effort but not understanding what system tests should do.	(8) >= half of the use cases don't have system tests OR only testing one scenario per use case OR most tests don't pass.	(12) System tests for all use cases are provided, and there is some effort to test a few scenarios for each use case. The structuring of the tests, naming, comments, messages OR use of assert methods could be improved. Most tests pass.	(16) System tests for all use cases are provided and pass, and good effort at testing several scenarios for each use case. Structuring of the tests, naming, comments, messages and use of assert methods usually good with small issues.	(20) System tests for all use cases are provided and pass, and a high number of scenarios for each use case are tested. Structuring, naming, comments, messages and use of assert all excellent.
Task 4: Unit testing (15)	**At least "Poor" in Task 1 and having reasonably implemented**	(0) Not attempted	(3) Some effort but not understanding what unit tests should do.	(6) >= half of the required classes or >= half of their methods not having unit tests OR testing only	(9) Unit tests for all required classes and for (almost) all their methods are provided, and there is some effort to test	(12) Unit tests for all required classes and their methods are provided and pass, and good	(15) Unit tests for all required classes and their methods are provided and pass, and testing

* As described in Lecture 14.

	the required classes for this task.			one set of inputs (in one test) for each method OR most tests don’t pass.	a few classes of inputs for each method, however the structuring of the tests, naming, comments, messages OR use of assert methods could be improved. Most tests pass.	effort at testing several classes of inputs for each method. Structuring of the tests, naming, comments, messages and use of assert methods usually good with small issues.	a high number of classes of inputs for each method. High coverage (>70%) of all the classes being tested. Structuring, naming, comments, messages and use of assert all excellent.
Task 6: Reflection (15)	**At least ‘Poor in Tasks 2, 3.**	(0) Not attempted	(3) Little use of reflection – mostly broad unjustified, even dishonest statements	(6) Some use of reflection but generally too brief or only on team work/work.	(9) An attempt to use the given reflective model or similar on a few of the suggested topics for team work, or to touch on a few aspects of the work, or more but just in one. Min. half a page.	(12) A good attempt to use the given reflective model or similar on a good number of the topics for team work and work. Some discussion of tools for team work, the marking scheme, experience with tools/agile approaches/ agile practices with choices appropriate, good	(15) An excellent attempt showing a highly reflective team, touching on most of the proposed topics, the marking scheme, and a good number of tools/ agile approaches/ agile practices with choices appropriate, good understanding and proof of any tools used (screenshots).

* As described in Lecture 14.

						understanding and proof of any tools used (screenshots). Min. a page.	
Exceptionality	15 marks that can be awarded at the discretion of the marker if mark of >80 (max=85) reached for exceptional programming and testing skills, exceptional use of tools/ agile approaches/ agile practices, exceptional team work.						

* As described in Lecture 14.