# Concordia University Department of Computer Science and Software Engineering

## Advanced Programming Practices SOEN 6441 --- Fall 2017

#### **Project Build 1 Grading**

#### Instructions for Incremental Code Build Presentation

You must deliver an operational version demonstrating a subset of the capacity of your system. This is about demonstrating that the code build is effectively aimed at solving specific project problems or completely implementing specific system features. The code build must not be just a "portion of the final project", but rather be something useful with a purpose on its own, that can be demonstrated by its operational usage.

The presentation should be organized as follows:

- 1. Brief presentation of the goal of the build.
- 2. Brief presentation of the architectural design of your project.
- 3. Demonstration of the functional requirements as listed on the following grading sheet.
- 4. Demonstration of the use of tools as listed on the following grading sheet.

You are graded according to how effectively you can demonstrate that the features are implemented. If you cannot really demonstrate the features through execution, you will have to prove that the features are implemented by explaining how your code implements the features, in which case you will get only partial marks.

During your presentation, you have to demonstrate that you are well-prepared for the presentation, and that you can easily provide clear explanations as questions are asked about the functioning of your code, or your required usage of the tools/techniques.

Before the presentation starts, you have to submit your current project code base to the Electronic Assignment Submission System under "project 1".

#### Identification

Team	Evaluator	Signature	Date

### Grading

Fluid exposition of knowledge of code base/clarity of explanations  Functional Requirements  30  Map editor  12  User-driven creation of map elements, such as country, continent, and connectivity between countries.  Saving a map to a file exactly as edited (using the "conquest" game map format).  Loading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.  Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).  22  Same Play  18  Implementation of a game driver implementing the game phases according to the Risk rules.  23  Startup phase  34  Sartup phase  36  Game starts by user selection of a user-saved map file, then loads the map as a connected graph.  37  User chooses the number of players, then all countries are randomly assigned to players.  38  19  19  19  19  19  19  19  19  19  1	Presentation	5			
Map editor  User-driven creation of map elements, such as country, continent, and connectivity between countries.  Saving a map to a file exactly as edited (using the "conquest" game map format).  Joading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.  Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).  Game Play  Inplementation of a game driver implementing the game phases according to the Risk rules.  Startup phase  Bame starts by user selection of a user-saved map file, then loads the map as a connected graph.  User chooses the number of players, then all countries are randomly assigned to players.  Players are allocated a number of initial armies, depending on the number of players.  In round-robin fashion, the players place one by one their given armies on their own countries.  Reinforcement phase  Salution of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  Fortification phase  Tortification phase  Torgramming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.	Effectiveness, structure and demonstrated preparation of the presentation				
Map editor User-driven creation of map elements, such as country, continent, and connectivity between countries.  4 User-driven creation of map elements, such as country, continent, and connectivity between countries.  5 aving a map to a file exactly as edited (using the "conquest" game map format).  3 Loading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.  4 Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).  5 degme Play  18 Implementation of a game driver implementing the game phases according to the Risk rules.  2 Startup phase  6 ame starts by user selection of a user-saved map file, then loads the map as a connected graph.  3 User chooses the number of players, then all countries are randomly assigned to players.  1 In round-robin fashion, the players place one by one their given armies on their own countries.  1 In round-robin fashion, the players place one by one their given armies on their own countries.  2 Reinforcement phase  5 Calculation of correct number of reinforcement armies according to the Risk rules.  2 Players place reinforcement armies on the map.  5 Tortification phase  3 Implementation of a valid fortification move according to the Risk rules.  3 Implementation of a valid fortification move according to the Risk rules.  3 Programming process  4 Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules and/or folders.  5 Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  4 API documentation—completed for all files, all classes and all methods.  4 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for	Fluid exposition of knowledge of code base/clarity of explanations				
User-driven creation of map elements, such as country, continent, and connectivity between countries.  Saving a map to a file exactly as edited (using the "conquest" game map format).  Loading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.  Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).  2 Game Play  In Implementation of a game driver implementing the game phases according to the Risk rules.  Startup phase  Bame starts by user selection of a user-saved map file, then loads the map as a connected graph.  3 User chooses the number of players, then all countries are randomly assigned to players.  Players are allocated a number of initial armies, depending on the number of players.  1 In round-robin fashion, the players place one by one their given armies on their own countries.  2 Players place reinforcement phase  Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  3 Fortification phase  Inplementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.	Functional Requirements	30			
Countries.  Saving a map to a file exactly as edited (using the "conquest" game map format).  Joading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.  Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).  Zempe Play  Inplementation of a game driver implementing the game phases according to the Risk rules.  Startup phase  Same starts by user selection of a user-saved map file, then loads the map as a connected graph.  Juser chooses the number of players, then all countries are randomly assigned to players.  Players are allocated a number of initial armies, depending on the number of players.  In round-robin fashion, the players place one by one their given armies on their own countries.  Zereinforcement phase  Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  3 Implementation of a valid fortification move according to the Risk rules.  Protrification phase  3 Implementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding	Map editor	12			
Loading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.  Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).  2 Game Play  Inglementation of a game driver implementing the game phases according to the Risk rules.  Startup phase  8 Game starts by user selection of a user-saved map file, then loads the map as a connected graph.  3 User chooses the number of players, then all countries are randomly assigned to players.  2 Players are allocated a number of initial armies, depending on the number of players.  1 In round-robin fashion, the players place one by one their given armies on their own countries.  2 Reinforcement phase  5 Calculation of correct number of reinforcement armies according to the Risk rules.  2 Players place reinforcement armies on the map.  3 Implementation of a valid fortification move according to the Risk rules.  3 Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	User-driven creation of map elements, such as country, continent, and connectivity between countries.	4			
Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).  2 Game Play 18 Implementation of a game driver implementing the game phases according to the Risk rules. 2 Startup phase 3 Game starts by user selection of a user-saved map file, then loads the map as a connected graph. 3 User chooses the number of players, then all countries are randomly assigned to players. 2 Players are allocated a number of initial armies, depending on the number of players. 1 In round-robin fashion, the players place one by one their given armies on their own countries. 2 Reinforcement phase 5 Calculation of correct number of reinforcement armies according to the Risk rules. 2 Players place reinforcement armies on the map. 3 Implementation of a valid fortification move according to the Risk rules. 3 Implementation of a valid fortification move according to the Risk rules. 3 Programming process 4 Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders. 3 Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for build 1.  a greading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Saving a map to a file exactly as edited (using the "conquest" game map format).	3			
Inplementation of a game driver implementing the game phases according to the Risk rules.  Startup phase  Game starts by user selection of a user-saved map file, then loads the map as a connected graph.  User chooses the number of players, then all countries are randomly assigned to players.  Players are allocated a number of initial armies, depending on the number of players.  In round-robin fashion, the players place one by one their given armies on their own countries.  Reinforcement phase  5 Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  3 Implementation of a valid fortification move according to the Risk rules.  3 Implementation of a valid fortification move according to the Risk rules.  3 Programming process  4 Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Intesting framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Loading a map from an existing "conquest" map file, then editing the map, or create a new map from scratch.				
Implementation of a game driver implementing the game phases according to the Risk rules.  Startup phase  Game starts by user selection of a user-saved map file, then loads the map as a connected graph.  User chooses the number of players, then all countries are randomly assigned to players.  Players are allocated a number of initial armies, depending on the number of players.  In round-robin fashion, the players place one by one their given armies on their own countries.  Reinforcement phase  5 Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  3 Implementation of a valid fortification move according to the Risk rules.  3 Implementation of a valid fortification move according to the Risk rules.  3 Implementation of a valid fortification move according to the Risk rules.  3 Programming process  4 Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Verification of map correctness upon loading and before saving (at least 3 types of incorrect maps).	2			
Startup phase  Game starts by user selection of a user-saved map file, then loads the map as a connected graph.  3 User chooses the number of players, then all countries are randomly assigned to players.  2 Players are allocated a number of initial armies, depending on the number of players.  1 In round-robin fashion, the players place one by one their given armies on their own countries.  2 Reinforcement phase  5 Calculation of correct number of reinforcement armies according to the Risk rules.  2 Players place reinforcement armies on the map.  5 Tortification phase  3 Implementation of a valid fortification move according to the Risk rules.  7 Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Game Play	18			
Game starts by user selection of a user-saved map file, then loads the map as a connected graph.  User chooses the number of players, then all countries are randomly assigned to players.  Players are allocated a number of initial armies, depending on the number of players.  In round-robin fashion, the players place one by one their given armies on their own countries.  Players place reinforcement phase  Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  Fortification phase  Implementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Implementation of a game driver implementing the game phases according to the Risk rules.	2			
User chooses the number of players, then all countries are randomly assigned to players.  Players are allocated a number of initial armies, depending on the number of players.  In round-robin fashion, the players place one by one their given armies on their own countries.  Reinforcement phase  Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  3 Portification phase  In players place reinforcement armies on the map.  Fortification phase  3 Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Startup phase	8			
Players are allocated a number of initial armies, depending on the number of players.  In round-robin fashion, the players place one by one their given armies on their own countries.  Reinforcement phase  Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  3 Implementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Game starts by user selection of a user-saved map file, then loads the map as a connected graph.	3			
In round-robin fashion, the players place one by one their given armies on their own countries.  Reinforcement phase  Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  Fortification phase  3 Implementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	User chooses the number of players, then all countries are randomly assigned to players.	2			
Reinforcement phase Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  3 Fortification phase 3 Implementation of a valid fortification move according to the Risk rules.  7 Programming process 7 Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Players are allocated a number of initial armies, depending on the number of players.	1			
Calculation of correct number of reinforcement armies according to the Risk rules.  Players place reinforcement armies on the map.  Fortification phase  Implementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	In round-robin fashion, the players place one by one their given armies on their own countries.	2			
Players place reinforcement armies on the map.  Fortification phase Implementation of a valid fortification move according to the Risk rules.  Programming process Ispanda design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Reinforcement phase	5			
Implementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Calculation of correct number of reinforcement armies according to the Risk rules.	2			
Implementation of a valid fortification move according to the Risk rules.  Programming process  Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Players place reinforcement armies on the map.	3			
Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  3 Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Fortification phase	3			
Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Implementation of a valid fortification move according to the Risk rules.	3			
and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.  Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	Programming process	15			
among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.  API documentation—completed for all files, all classes and all methods.  Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	<b>Architectural design</b> —short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.	3			
Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	<b>Software versioning repository</b> —well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.	3			
code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.  Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	API documentation—completed for <u>all</u> files, <u>all</u> classes and <u>all</u> methods.	3			
code layout, naming conventions and comments, absence of "commented out" code.	Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation; (2) calculation of number of reinforcement armies; (3) reading an invalid map file.	3			
Total	Coding standards—documented description of coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of "commented out" code.	3			
	Total	50			

Notes			