Block III Mini Capstone

OBJECTIVES

1a. Identify basic facts and state general principles associated with software development documentation. Task(s): 3.13.1, 3.13.2

1b. Identify basic facts and state general principles pertaining to Object-Oriented Programming(OOP) concepts and design. Task(s): 2.3.1, 2.3.2

2a. Identify basic facts and state general principles about Iterative Development. Task(s): 3.9

2b. Identify basic facts and state general principles about DevOps. Task(s): 3.11, 3.12

INSTRUCTIONS

1. With the assistance of the instructor, determine the composition of the teams for the mini capstone.
2. Research and decide what your mini capstone will be. Either choose from the pre-determined list of examples for mini- capstone topics or choose a topic that will need to be approved by the instructor.
3. Develop user stories for your mini capstone.
4. From those user stories, encapsulate the actors and their behaviors by developing a use-case diagram.
5. Given the Use-Cases generated in the previous step, generate class diagrams.
6. Create a sample state machine for the mini capstone. Within the state machine attempt to identify missing functional and non-functional requirements.
7. As a team, determine an appropriate SDLC process (Waterfall, Iterative, Scrum, XP, etc.) for the mini capstone.
8. As a team, and to the specifications of the Rubric, complete the development of the capstone.
9. Development of the mini capstone is a three-day activity, with the scope of the project intended to be 100% complete by the third day. Correspond and work with the Instructor (the customer of your application) to be developed.
10. Upon completion, submit your mini capstone.

NOTE: The coding requirements and the written essay requirement of the mini-capstone rubric are required from each individual student. Design artifacts, including User Documentation and are shared points for the group. This is a shared responsibility project.

- Make sure that students are given the rubric of requirements to implement into the mini capstone.

- Before beginning, students will need to choose an SDLC process (such as waterfall, iterative, scrum, or XP) to implement while they complete the capstone.

- Student write-ups afterwards must include their reasoning for having chosen a specific SDLC process, and their conclusions about working with the chosen form. Instructors, if necessary, can play the part of a customer.

- Provisions were put into place to prevent individual students from completing all the work for their team. For example, the OOP requirements will be completed by all students, not just as a team. Also, for this mini-capstone, team composition should be decided by the Instructors. While students can choose their actual capstone teams, for this learning activity, working with unfamiliar students is a good learning process.

Suggested Mini Capstone Projects:

1. Libraries:
   - tkinter
   - pysimplegui
   - PyGame

   http://programarcadegames.com/

2. Games:
   Need to define number of human players and number of computers. (As an instructor, need to review the suggested automated portions of the game versus player input for the game. Number of human or computer players can determine complex or simple logic.)

   - Board Games:
     - Scrabble
     - Battleship - custom, more than basic Battleship
     - Backgammon
     - Monopoly
     - Connect 4
     - Mastermind
     - Stratego
     - Wordle
     - NOT Chess

   - Card Games:
     - Rummy
     - Hearts
     - Uno
     - NO Gambling Games
     - NOT Solitaire

- Interactive Games:
    - Codenames
    - Choose your own adventure
    - Escape Room

- Programs:
    - *Simulations:*
        - The game of life
        - Auto-battlers

INSTRUCTIONS:

1. Your instructor will assign you to a team.
2. Research together and decide what your mini capstone will be. Either choose from the pre-determined list of examples for mini-capstone topics or choose another topic, which will need to be approved by the instructor.
3. Develop user stories for your mini capstone.
4. From those user stories, encapsulate the actors and their behaviors by developing a use-case diagram.
5. Given the Use-Cases generated in the previous step, generate class diagrams.
6. Create a sample state machine for the mini capstone. Within the state machine attempt to identify missing functional and non-functional requirements.
7. As a team, determine an appropriate SDLC process (Waterfall, Iterative, Scrum, XP, etc.) for the mini capstone.
8. As a team, and to the specifications of the rubric, complete the development of the capstone.
9. Development of the mini capstone is a 23.5-hour activity, with the scope of the project intended to be 100% complete by the end of the PC. Correspond and work with the Instructor (the customer of your application) to be developed.
10. Upon completion, submit your mini capstone for grading.

Requirement Elements	Exceeds Expectations	Meets Expectations	TROLL
User/Software Documentation – Developed during the planning phase of the Mini Capstone. (1) User stories sufficiently account for requirements, actors, and define expected roles of users in the program. (2) Use-Case Diagram includes the system, actors, and their behaviors. (3) Class Diagram- Models the expected classes in the program, including the class names, class attributes, class methods, and the expected relations between those classes, including aggregation and composition. (4) State Machine- Accounts for program inputs, correctly defines states within the program, and correctly shows transitions between states	All four elements were included, there were no mistakes, and no instructor assistance was needed.	All four elements were included, there were no more than two mistakes (up to two per element), and no more than three instructor assists were needed.	One or more required elements was not included, or there were more than three mistakes, or more than three instructor assists were needed.
	10	**7**	**4**
Classes- (1) At least two complete classes were developed, (2) Proper implementation of __init__ constructors	Both elements were included, there were no mistakes, and no instructor assistance was needed.	Both elements were included, there were no more than two mistakes, and no more than two instructor assists were needed.	One or more required elements was not included, or there were more than two mistakes, and more than two instructor assists were needed.
	10	**7**	**4**
Objects- At least ten objects were initialized	At least ten objects were initialized, there were no mistakes, and no instructor assistance was needed.	At least seven objects were initialized, there were no more than two mistakes, and no more than two instructor assists were needed.	Less than seven objects were initialized, or there were more than two mistakes, and more

			than two instructor assists were needed.
	10	**7**	**4**
Functions and Methods-	At least 10 methods were written, there were no mistakes, and no instructor assistance was needed.	At least seven methods were written, there were no more than two mistakes, and no more than two instructor assists were needed.	Less than seven methods were written, or there were more than two mistakes, and more than two instructor assists were needed.
	10	**7**	**4**
Attributes and Properties	At least five instance attributes were written, there were no mistakes, and no instructor assistance was needed.	At least two class attributes were written, there were no more than two mistakes, and no more than two instructor assists were needed.	Less than two class attributes were written, or there were more than two mistakes, and more than two instructor assists were needed.
	10	**7**	**4**
Abstraction- Private attributes, getter methods, setter methods.	At least two private attributes were written, At least two getter methods were written, At least two setter methods were written, there were no mistakes, and no instructor assistance was needed. Students were able to verbalize how/why abstraction was applied in their program.	At least two private attributes were written, At least two getter methods were written, At least two setter methods were written, there were no more than two mistakes per element, and no more than three instructor assists were needed. Students, with minor difficulty, were able to verbalize how/why abstraction was applied in their program.	Not all requirements were met, or students made more than two mistakes per element, or students required more than three instructor assists, or students were unable to verbalize how/why abstraction was applied in their program.
	10	**7**	**4**
Encapsulation	Given the project requirements, appropriate classes were defined, there were no mistakes, and no instructor assistance was needed. Students were	Given the project requirements, appropriate classes were defined, there were no more than two mistakes, and no more than two instructor assists were needed. Students, with minor	Not all requirements were met, or students made more than two mistakes, or student required more than two instructor assistance, or

	able to verbalize how/why their program properly encapsulates the requirements.	difficulty, were able to verbalize how/why their program properly encapsulates the requirements.	students were unable to verbalize how/why their program encapsulates the requirements.
	10	**7**	**4**
Inheritance	Given the project requirements, students created at least three classes that need to inherit from/extend a parent class, there were no mistakes, and no instructor assistance was needed. Students were able to verbalize how/why inheritance is applied in their program.	Given the project requirements, students created at least three classes that need to inherit from/extend a parent class, there were no more than two mistakes, and no more than two instructor assists were needed. Students, with minor difficulty, were able to verbalize how/why inheritance is applied in their program.	Not all requirements were met, or students made more than two mistakes, or students needed more than two instructor assists, or students were unable to verbalize how/why inheritance is applied in their program.
	5	**4**	**2**
Polymorphism	Given the project requirements, at least three methods overwrite the behavior of a parent classes' methods, there were no mistakes, and no instructor assistance was needed. Students were able to verbalize how/why polymorphism is applied in their program.	Given the project requirements, at least three methods overwrite the behavior of a parent classes' methods, there were no more than two mistakes, and no more than two instructor assists were needed. Students were able to verbalize how/why polymorphism is applied in their program.	Not all requirements were met, or students made more than two mistake, or students needed more than two instructor assists, or students were unable to verbalize how/why polymorphism is applied in their program.
	5	**4**	**2**
Proper Naming Conventions- Class names, method names, variable names, file names	Proper naming conventions were used for class names, method names, variable names, and file names; there were no mistakes, and no instructor assists were needed.	Proper naming conventions were used for class names, method names, variable names, and file names; there were no more than two mistakes; and there were no more than two instructor assists.	Not all requirements were met, or students made more than two mistakes, or students required more than two instructor assists.

	5	4	2
Docstrings- All classes and all methods include a Docstring	Perfect with no errors and no instructor assists.	All classes and methods include a Docstring; there were no more than two errors, and no more than two instructor assists.	Not all methods include a Docstring, or there were more than two errors, or student needed more than two instructor assists.
	5	**4**	**3**
Comments- Does the student documentation include enough comments to understand logical sections of the program?	Yes	There are comments but not enough to enable understanding of the logical sections of the program.	No
	5	**4**	**0**
Extra Credit – Annotation (1) Parameter types Ex: var1:int, var2:str (2) Return Types Ex: ->int (3) Variable Types Ex: people:dict[int:list]	Student successfully completed all three elements.	Student successfully completed two of the three elements.	Student successfully completed one of the three elements.
	3	**2**	**1**
Extra Credit – Program Architecture (1) Indentation and Readability (2) Single Responsibility Principle (3) Eliminating Duplicate Code (4) Adherence to planned State Diagram, Refined one-to-one State Diagram as a software document artifact.	Student successfully completed all four elements.	Student successfully completed two of the four elements.	Student successfully completed one of the four elements.
	4	**2**	**1**
Extra Credit – User Documentation Creation of user support documentation. For example: user-manual or tutorial	Student successfully created one type of user documentation with no errors and no instructor assists.	Student successfully created one type of user documentation with no more than four errors and no more than two instructor assists.	Student successfully created one type of user documentation with no more than six errors and no more than three instructor assists.
	5	**3**	**1**
Total (**95** points without extra credit). (**12** extra-credit points possible)	**95**	**70**	**35**