**Project Deliverable 5**

**Personal Software Process & Quality (PSP2, Unit Testing)**

**Points: 30**

**----------------------------------------------------------------------------------------------------**

**Instructions: INDIVIDUAL ASSIGNMENT (no collaboration)**

**Deliverable (30 points):**

Submit the following: YourASURiteID-ProjectDeliverable5.zip This compressed folder should contain the following files:

* All java files that you have so far including your test classes (they should be placed correctly in core, ui, and test folders). Build this assignment **using Deliverable 4 code**.
* Completed Time Log, Design form, Estimation worksheet, Defect Log, and Project Summary provided at the end of this assignment description
* A few screen shots showing results of your JUnit testing of the game, code coverage report, and answers to reflection questions written inline in this document
* Readme file (optional: submit if you have any special instructions for testing)

**----------------------------------------------------------------------------------------------------**

**Program Requirements**:

Conduct Unit testing for Game Logic class(es) and Computer Player class in the core package using JUnit for this deliverable. Create a separate package called test for the test class(es). Provide test methods for all important methods of the game logic and computer player module with different possible test cases using equivalence partitioning. You should have both success and failure cases. Use a code coverage tool such as EclEmma to generate a code coverage report for classes (only those classes where you provided unit tests) in the core package. You must achieve at least 90% code coverage (only Game Logic and Computer Player classes) through various test cases. Include the code coverage report in your submission.

**----------------------------------------------------------------------------------------------------**

**Personal Process:**

Follow a good personal process for implementing this game. You will be using PSP2 in this assignment. Estimate and track your effort and defects for unit testing code that you write and conduct a personal code review.

* Please use the time log (provided at the end of this document) to keep track of time spent in each phase of development.
* Please use the defect log (provided at the end of this document) to keep track of defects found and fixed in each phase of development.
* When you are done implementing and testing your program, complete the Project Summary form to summarize your effort and defects. Also answer the reflection questions listed below in Post-mortem phase.

Follow these steps in developing this game:

* **Plan**:
* understand the program specification and get any clarifications needed.
* estimate the time you are expecting to spend on unit testing task.
* estimate the defects you are expecting to inject in each phase for unit testing task.
* estimate the size of the program (only for new code that you will be adding)
* enter this information in the Estimation columns of the Project Summary form. Use your best guess based on your previous programming experience.
* **Design** – design the test classes and methods. Test case generation for various test methods needs to be done here. Keep track of time spent in this phase and log it. Also keep track of any defects found and log them. Use this phase to design different test methods and test cases.
* **Code** – implement the program. Keep track of time spent in this phase and log. Also keep track of any defects found and log them.
* **Code Review** – use the code review guidelines/checklist provided later in the document to conduct a personal review of your code and fix any issues found. Provide comments in the checklist about your findings.
* **Test** – Test your program thoroughly and fix bugs found. Goal here is to test all classes of core package thoroughly. Keep track of time spent in this phase and log. Also keep track of any defects found and log them.
* **Post Mortem** – Complete the actual columns of the project summary form and answer the following questions.
* How good was your time and defect estimate for various phases of software development?

My time estimate was off by about 110 minutes, which is not bad compared to previous predictions in earlier deliverables. My defect estimate for this project was nearly spot-on, by predicting 5 defects and only injecting 4 defects instead.

* How good was your program size estimate, i.e., was it close to actual?

My program size estimate did not end up being very close to actual. While I had predicted around 600 lines of code, I ended up only programming 194 lines of code to cover 90% of test cases. However, curiously enough, the overestimate balanced my overall program size, only being off by around 200 lines of code between my total lines of code and total estimate.

* How much code coverage did you achieve in the core package?

I was able to achieve 85% code coverage total in the core package. However, in the CheckersGameLogic and CheckersComputerPlayer classes, I was able to have 90% and 93% code coverage respectively.

**--------------------------------------------------------------------------------------------------**

**Grading Rubric**:

Unit Testing – 15 points

Test Results , Code coverage report, Postmortem reflection question responses – 5 points

PSP process – 10 points (Time log (2), Defect log (2), Estimation Worksheet & Design form (2), Code review (2), Project Summary (2))

**--------------------------------------------------------------------------------------------------**

**PSP Time Recording Log**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Start** | **Stop** | **Interruption Time** | **Delta**  **Time** | **Phase** | **Comments** |
| 4/27 | 12:00pm | 12:50pm | 0 min | 50 min | Planning | Performed research on JUnit and using it in IntelliJ IDEA. |
| 4/27 | 12:50pm | 1:00pm | 0 min | 10 min | Design | Created a design for the JUnit classes. |
| 4/27 | 2:45pm | 5:00pm | 15 min | 120 min | Coding | Created test methods for the Checkers computer player. Code coverage 83%. Interrupted by further research. |
| 4/27 | 5:00pm | 5:30pm | 0 min | 30 min | Testing | Tested computer player coverage testing after further changes. CheckersComputerPlayer now has 93% coverage in testing. |
| 4/27 | 5:30pm | 7:30pm | 5 min | 115 min | Coding | Started programming test cases for CheckersGameLogic. Code coverage reported to be 50%. |
| 4/27 | 8:30pm | 8:45pm | 0 min | 15 min | Coding | Finished creating code coverage cases for the remaining functions. Code coverage 88%. |
| 4/27 | 8:45pm | 9:00pm | 0 min | 15 min | Testing | Fixed uncovered cases in functions. Code coverage increased to 90%. |
| 4/27 | 9:30pm | 10:30pm | 15 min | 45 min | Postmortem | Finished all postmortem questions and PSP documentation. |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

* **Interruption time**: Record any interruption time that was not spent on the task. Write the reason for the interruption in the "Comment" column. If you have several interruptions, record them with plus signs (to remind you to total them).
* **Delta Time**: Enter the clock time you spent on the task, less the interrupt time.
* **Phase**: Enter the name or other designation of the programming phase being worked on. Example: Design or Code.
* **Comments**: Enter any other pertinent comments that might later remind you of any details or specifics regarding this activity.

**PSP1 Informal Size Estimating Procedure**

1. Study the requirements.

2. Sketch out a crude design.

3. Decompose the design into “estimatable” chunks.

4. Make a size estimate for each chunk, using a combination of:

\* visualization.

\* recollection of similar chunks that you’ve previously written

\* intuition.

5. Add the sizes of the individual chunks to get a total.

**Estimating Worksheet**

* Conceptual Design (sketch your high-level design here)

A screenshot of a computer

Description automatically generated

* Module Estimates

|  |  |
| --- | --- |
| **Module description** | **Estimated Size** |
| Checkers.core, which handles all game logic in the checkers system and computer player logic. | 900 LOC |
| Checkers.ui, which handles the text console and GUI visuals for the checkers game | 200 LOC |
| Checkers.test, which is comprised of all test methods for the game logic and computer player. | 600 LOC |
|  |  |
|  |  |
|  |  |
|  |  |

Total Estimated Size: \_\_\_1700 LOC\_\_\_\_\_

**PSP2 Project Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Time in Phase (minutes)** | **Estimated** | **Actual** | **To Date** | **To Date %** |
| Planning | 30 min | 50 min | 111 min | 3.675% |
| Design | 20 min | 10 min | 304 min | 10.07% |
| Code | 120 min | 250 min | 1718 min | 56.89% |
| Code Review | 30 min | 0 min | 94 min | 3.11% |
| Test | 60 min | 45 min | 655 min | 21.69% |
| Postmortem | 30 min | 45 min | 138 min | 4.57% |
| TOTAL | 290 min | 400 min | 3020 min | 100% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Defects Injected** | **Estimated** | **Actual** | **To Date** | **To Date %** |
| Planning | 0 | 0 | 0 | 0% |
| Design | 0 | 0 | 2 | 8% |
| Code | 5 | 4 | 21 | 84% |
| Code Review | 0 | 0 | 2 | 8% |
| Test | 0 | 0 | 0 | 0% |
| Postmortem | 0 | 0 | 0 | 0% |
| TOTAL | 5 | 4 | 25 | 100% |

**SUMMARY**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Estimated** | **Actual** | **To Date** |
| Program Size (LOC) | 600 LOC | 194 LOC | 1852 LOC |
| LOC/Hour | 124.14 | 29.1 | 179.343 |
| Defects/KLOC | 8.333 | 20.619 | 98.308 |

* LOC is lines of Code
* KLOC is Kilo lines of code (i.e. 1000 lines)

**PSP Design Form**

*Use this form to record whatever you do during the design phase of development. Include notes, class diagrams, flowcharts, formal design notation, or anything else you consider to be part of designing a solution that happens BEFORE you write program source code. Attach additional pages if necessary.*

**PSP Defect Recording Log**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sl. No.** | **Date** | **Defect Type** | **Defect Inject Phase** | **Defect Removal Phase** | **Fix Time** | **Fix Ref** | **Description** |
| 1 | 4/27 | 100 | Coding | Coding | 10 min | N/A | Had an issue running JUnit as I did not initially understand it. |
| 2 | 4/27 | 60 | Coding | Testing | 5 min | N/A | Did not cover a branch in CheckersComputerPlayer. Found that the branch was never reachable, so it was deleted. |
| 3 | 4/27 | 70 | Coding | Testing | 5 min | N/A | Encountered issues when working with text input file path. Issues were fixed with some workarounds. |
| 4 | 4/27 | 61 | Coding | Testing | 20 min | N/A | Did not cover enough test cases in setSquare(). |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Instructions**

* **Defect Type**: Use your best judgment in selecting which defect type applies from list provided below.
* **Defect Inject Phase**: Enter the phase when this defect was injected using your best judgment.
* **Defect Removal Phase**: Enter the phase during which you fixed the defect.
* **Fix Time**: Enter the time that you took to find and fix the defect.
* **Fix Ref**: If you or someone else injected this defect while fixing another defect, record the number of the improperly fixed defect. If you cannot identify the defect number, enter an X. If it is not related to any other defect, enter n/a.
* **Description**: Write a succinct description of the defect that is clear enough to later remind you about the error and help you to remember why you made it.

**PSP Defect Type Standard**

|  |  |  |
| --- | --- | --- |
| **Type Number** | **Type Name** | **Description** |
| 10 | Documentation | Comments, messages |
| 20 | Syntax | Spelling, punctuation, typos, instruction formats |
| 30 | Build, Package | Change management, library, version control |
| 40 | Assignment | Declaration, duplicate names, scope, limits |
| 50 | Interface | Procedure calls and references, I/O, user formats |
| 60 | Checking | Error messages, inadequate checks |
| 70 | Data | Structure, content |
| 80 | Function | Logic, pointers, loops, recursion, computation, function defects |
| 90 | System | Configuration, timing, memory |
| 100 | Environment | Design, compile, test, or other support system problems |

**Code Review Checklist – Java**

1. Specification / Design

[y] Is the functionality described in the specification fully implemented by the code?   
[y] Is there any excess functionality in the code but not described in the specification?

2. Initialization and Declarations

[y] Are all local and global variables initialized before use?   
[y] Are variables and class members of the correct type and appropriate mode   
[y] Are variables declared in the proper scope?   
[y] Is a constructor called when a new object is desired?   
[ y] Are all needed import statements included?

[ y] Names are simple and if possible short

[ y] There are no usages of ‘magic numbers’ (i.e, hard-coded values)

3. General

[ y] Code is easy to understand

[ y] Variable and Methods names are spelt correctly

[ y] There is no dead code (i.e., code inaccessible at Runtime)

[ y] Code is not repeated or duplicated

[ y] No empty blocks of code

4. Method Calls   
[ y] Are parameters presented in the correct order?   
[ y] Are parameters of the proper type for the method being called?  
[ y] Is the correct method being called, or should it be a different method with a similar name?   
[ y] Are method return values used properly? Cast to the needed type?

5. Arrays/Data structures   
[ y] Are there any off-by-one errors in array indexing?   
[ y] Can array indexes ever go out-of-bounds?   
[ y] Is a constructor called when a new array item is desired?

[ y] Ideal data structures are used

[ y] Collections are initialized with a specific estimated capacity

6. Object   
[ y] Are all objects (including Strings)  compared with "equals" and not "=="?

[ y] No object exists longer than necessary

[ y] Files/Sockets and other resources if used are properly closed even when an exception occurs in using them

7. Output Format   
[ y] Are there any spelling or grammatical errors in displayed output?   
[ y] Is the output formatted correctly in terms of line stepping and spacing?

8. Computation, Comparisons and Assignments   
[ y] Check order of computation/evaluation, operator precedence and parenthesizing   
[ y] Can the denominator of a division ever be zero?   
[ y] Is integer arithmetic, especially division, ever used inappropriately, causing unexpected truncation/rounding?   
[ y] Check each condition to be sure the proper relational and logical operators are used.   
[ y] If the test is an error-check, can the error condition actually be legitimate in some cases?   
[ y] Does the code rely on any implicit type conversions?

9. Exceptions

[ y] Are all relevant exceptions caught?   
[ y] Is the appropriate action taken for each catch block?

[ y] Are all appropriate exceptions thrown?

[ y] Are Catch clauses are fine-grained and catch specific exceptions?

10. Flow of Control

[ y] In a switch statement is every case terminated by break or return?   
[ y] Do all switch statements have a default branch?  
[ y] Check that nested if statements don't have “dangling else” problems.   
[ y] Are all loops correctly formed, with the appropriate initialization, increment and termination expressions?   
[ y] Are open-close parentheses and brace pairs properly situated and matched?

11. Files

[ y] Are all files properly declared and opened?   
[ y] Are all files closed properly, even in the case of an error?   
[ y] Are EOF conditions detected and handled correctly?   
[ y] Are all file exceptions caught?

12. Documentation

[ y] All methods are commented in clear language.

[ y] Comments exist and describe rationale or reasons for decisions in code

[ y] All public methods/interfaces/contracts are commented describing usage

[ y] All edge cases are described in comments

[ y] All unusual behavior or edge case handling is commented

[ y] Data structures and units of measurement are explained