Name_____

CPSC:480 Software Engineering Final

Do not open exam until instructed to do so!

Each question in section 1 is worth 2 points.

Each question in section 2 is 10 pts (2/item).

Each question in sections 3-5 is 5 or 10 pts.

Refer to attached descriptions for parts 3-5.

One sentence per description is adequate.

200 points total.

Do not open exam until instructed to do so!

Section 1 – Short answer

1. Describe a software engineering skill you've developed through this course.

2. Describe 1 similarity and 1 difference between software engineering vs other engineering disciplines.

3. Describe a condition or restriction found in a software license that might prevent some development teams from using the code with that license.

4. Describe one of the four types of software maintenance.

5. Describe a possible reason why a block of code might need to be refactored.

6. Define a corner case in software quality assurance.

7. Describe a bug you've observed in a piece of software you've developed or used.

8. Define the relationship between Mean Time To Recover and software downtime.

9. Describe the role of a Quality Assurance engineer within a software development team.

10. Describe one department or role outside of the software engineering team that might need to approve a release candidate for distribution, and why they might need to do so.

11. Describe a business task outside of product development that can be automated with software.

12. Describe one of the 12 Agile Alliance principles.

13. Describe the expected state of a software project at the “code complete” milestone.

14. Describe the software product or system currently operating that you believe has the largest impact on society today.

15. Version 4.3 of a product has been released. What is one factor that might be considered in the decision as to whether the next release is numbered as version 4.4 or version 5.0?

16. Describe a way in which software accessibility considerations may impact product design.

17. What is the difference between a functional requirement and a non-functional requirement?

18. Define the formula for risk exposure and explain the inputs to the formula.

19. Describe the difference between the *severity* and *priority* of a bug.

20. Describe one of Lehman's Laws of Software Evolution.

Section 2 – Five of a Kind

21. Describe five common components of a software requirements specification

22. Describe five roles in a software engineering project team.

23. Describe five cross-cutting concerns that might apply to a software product.

24. List five git commands and describe the purpose of each

25. Give five examples of elements that might be found in a coding standards document or style guide.

26. Describe five potential outcomes of triaging a bug report.

27. Describe five activities that occur for a software release after the code complete milestone.

28. Describe five artifacts that might be included in a software release other than the main program.

Section 3 – Requirements Engineering and Product Design

Refer to the project description attached to the exam for this section. You should also refer to the source code for part 4 for guidance on what components might be included in your models, but remember that the code is only a *partial* implementation; you will need to consider classes that are not addressed in the code.

29. (10 pts) Develop a class model for the IDE project.

30. (10 pts) Develop a behavioral model (state diagram) for the IDE project

Section 4 – Implementation and Testing

Refer to the project description and teammate's pull request contents attachments for this section.

31. (5 pts) Identify a block of code that demonstrates adherence to appropriate coding standards & how

32. (10 pts) What is the cyclomatic complexity of each function defined in the pull request? What code quality issues are present here that you would comment on in a code review? (You may mark up the code instead of writing here)

33. (5 pts) Does testRunHelloWorld pass (return true), assuming the build, execution, and output capture functions work as intended? Does it uncover any bugs?

34 (10 pts) Write one unit test case for the OnKeyPress function that validates a requirement. Describe in general terms how you would write additional tests to achieve adequate code coverage.

35. (10 pts) Refactor the code for the OnKeyPress functionality to improve the code quality and separation of concerns. Does your unit test still pass?

Section 5 – Project Management and SCM

36. (5 pts) The project management team would like to automate some parts of employee performance evaluation for developers on the IDE project using Lines of Code per day as a metric. Use pseudocode to list the steps a script could take to assist with this task by examining the repository commit history.

37. (5 pts) Describe a business risk related to the requirement to store credentials for a repository in the workspace file and the potential impact of that risk.

38. (5 pts) A last-minute customer request comes through to add support for breakpoints in debug mode (run-to-next-breakpoint on command, as an alternative to execute-next-step on command). Create a task list for this new requirement.

39. (5 pts) Describe a set of manual acceptance tests that could be performed to help validate that the finished product meets the requirements.

Extra Credit

You're given a function "getContents" that takes a directory path as a string and returns a list of strings representing the contents of that directory (empty list if the path doesn't exist or is a file rather than a directory). Write a function in the language of your choice to search the entire filesystem for all files matching a given filename, with the lowest possible cyclomatic complexity.

Grading

Page	Max Points	Grade
2	10	
3	10	
4	10	
5	10	
6	20	
7	20	
8	20	
9	20	
10	20	
11	15	
12	15	
13	10	
14	10	
15	10	
16 (EC)	10	
Total	200	

Project Description for Sections 3-5

An Integrated Development Environment (IDE) is a suite of tools for software development. You will participate in the software development lifecycle for a project to create a small, lightweight IDE. Common features of an IDE include:

- Text editor – Modify source code, test case, build definition, or other content
- File management – Display the file and folder structure for relevant content
- Version control repository integration – Pull and create pull requests to remote repo
- Build configuration and definition – Create program from source code
- Test framework integration – Execute pre-written automated tests and track results
- Debugger – Run program built in debug mode with program inspection capabilities

In a typical case, a user of the IDE would open a project, solution, or workspace file that represents all the content for the development project. We will call the file that our IDE uses to encapsulate the software project contents a *workspace*. The workspace will define:

- A set of source code files that represent the input to the build process.
- A set of pre-defined commands to build the software for Debug or Release mode.
- Connection and authentication Information for accessing an associated version control repository with contents matching the file structure known to the workspace file and IDE.
- A set of automated test cases that can be executed after a build.

These are the requirements for the basic IDE software project to be constructed. As a developer, I want to...

1. View the code from a selected file in my workspace in my editor and switch between code files.
2. Modify the code in a file open in my editor and see an indication when I've written code that won't build.
3. Create new code files and reorganize or delete files that I have.
4. Push changes to my branch on command and automatically issue a pull request to a remote repository containing the changes I've made in the editor.
5. Build my software in either debug mode or release mode on command.
6. View a list of automated test cases associated with my workspace.
7. Add, update, and delete automated test cases using the editor.
8. Run all test cases associated with my code on command and view the results.
9. Launch the debugger on command and step through my program one statement at a time, showing all of the values currently assigned in memory, and advance to the next step or halt the debugging process on command.
10. Switch between viewing my editor, file explorer, repository info, build definition, test cases, and current program state in debug mode (if active)

Teammate's Pull Request Contents for Section 4

Your teammate has submitted a pull request with the following C++ code for review.

```
#include <fstream>
#include <iostream>
#include <string>
#include <unordered_map>
#include <vector>

using namespace std;

// Use a list of identifiers for keys on the keyboard, and a map to
associate the name of the key
typedef int Keycode;
unordered_map<string, Keycode> Keycodes;

class editor {
public:
  string filepath;
  vector<string> lines;
  int cursorLine;
  int cursorColumn;
};

class workspace {
public:
  vector<string> buildCommands;
  void save(editor e) {
    ofstream file;
    file.open(e.filepath.c_str());
    for (string line : e.lines) {
      file << line;
    }
  }
};

class testCase {
public:
  bool (*funcToTest)(void** args);
  void** args;
  bool execute() {
    return funcToTest(args);
  }
};
```

```
class IDE {
public:
  int mode; //0 = editor, 1 = file explorer, 2 = build definition, 3 =
test cases, 4 = debugger
  editor editor;
  workspace openWorkspace;
  void PushCodeToRemoteRepo() {}
  void LaunchDebugMode() {}
  void ExecuteNextDebugLine() {}
  bool Build() {
    openWorkspace.save(editor);
    for (string command : this->openWorkspace.buildCommands) {
      int result = system(command.c_str());
      if (result < 0) { // error
        return false;
      }
    }
    return true;
  }
  void RunTests() {}
};

// Return true if an only if the key is a letter, number, or symbol
(so not an arrow key, escape, etc)
bool KeycodeIsCharacter(int key);
// Return true if and only if the key represents F1-F12 keys
bool KeycodeIsFunctionKey(int key);
// Return the symbol associated with the Keycode, as a string for
convenience
string KeycodeToCharacter(Keycode key, bool shift);
// GetOutputFromExecution returns the console output of running a
command on the command line
string GetOutputFromExecution(string command);

void onKeyPress(Keycode key, bool shift, bool ctrl, IDE ide) {
  if (ctrl) {  // Used to switch modes in the IDE
    for (int i = 0; i < 5; i++) {
      if (key == Keycodes[to_string(i)]) {
        ide.mode = i;
        return;
      }
    }
    if (key == Keycodes["S"]) {  // Ctrl+S=save
      ide.openWorkspace.save(ide.editor);
    }
    // TODO - Ctrl+N new file, etc
  }
```

```
  if (ide.mode == 0) {  // Typing into the editor
    editor e = ide.editor;
    if (key == Keycodes["Enter"]) {
      e.lines.push_back("");
    }
    if (key == Keycodes["Backspace"]) {
      if (e.cursorColumn == 0) {  // Special case for start of line
        e.lines.erase(e.lines.begin() + e.cursorLine);
        return;
      }
      string line = e.lines[e.cursorLine];
      line.erase(line.begin() + e.cursorColumn);
    }
    if (KeycodeIsCharacter(key)) {
      e.lines[e.cursorLine].insert(e.cursorColumn,
KeycodeToCharacter(key, shift));
    }
  }
  if (KeycodeIsFunctionKey(key)) {  // One of the commands specified
in requirements
    if (key == Keycodes["F4"]) {  // Push
      ide.PushCodeToRemoteRepo();
    }
    if (key == Keycodes["F5"]) {  // Debug
      ide.LaunchDebugMode();
    }
    if (key == Keycodes["F6"]) {  // Build
      ide.Build();
    }
    if (key == Keycodes["F7"]) {  // Run tests
      ide.RunTests();
    }
    if (key == Keycodes["F10"] && ide.mode == 4) {  // Step through
debug
      ide.ExecuteNextDebugLine();
    }
  }
}
```

```
bool runHelloWorld(void** args) {
  string output = GetOutputFromExecution("helloWorld.exe");
  return output.starts_with("Hello, world!"); // Starts-with instead
of == for flexibility
}

// This is a test case *FOR THE IDE PRODUCT CODE*, verifying that it
can save, build, and run a test case with a simple workspace
bool testRunHelloWorld() {
  workspace w = workspace();
  editor e = editor();
  e.lines = { "#include <iostream>", "int main() {", "std::cout <<
\"Hello, world!\n\";", "}" };
  e.filepath = "helloWorld.cpp";
  w.buildCommands = { "g++.exe helloWorld.cpp -o helloWorld.exe" };
  IDE ide = IDE();
  ide.openWorkspace = w;
  ide.editor = e;
  ide.Build();

  // This is a test case associated with the helloWorld workspace,
which gets executed to validate the IDE test execution functionality
  testCase test = testCase();
  test.funcToTest = runHelloWorld;
  test.args = NULL;
  // If we got through to here and the workspace test passes, the IDE
product is working and this IDE test passes
  return test.execute();
}
```