EXAMINER : Mr Sebastian Coope

PAPER CODE NO.

**COMP 319**

DEPARTMENT: Computer Science



**EXAMINATION January 2022**

**COMP319 : SOFTWARE ENGINEERING II**

**INSTRUCTIONS TO CANDIDATES**

**Answer ALL questions**

**This exam consists of two parts a times MCQ quiz and this written paper**

**You MAY use a calculator in this exam**

**Question 1**  **[30 Marks]**

Look at the following description for a Java class used as a document checker.

The class is used to open, decode and process documents in a number of different file formats, for example docx or ASCII text.

For each document it needs to be able to:

1. Count the words in the document
2. Spell check the document returning a list of the words that it thinks are in error and for each word its line and column number in the document.

The code should support the following methods and attributes:

String lines[]; // Contents of the document

void open() // Tries to open the document for processing

// It then processes the document into an array of

// Strings and stores them in attribute lines (see

above)

// Each string is a line in the document

This method will throw an exception if the file does not exist

This method will throw an exception if the file is in an unsupported format

The format of the document will be described by the extension of the file name, so a file named essay.pdf will be in the portable document format.

The filename will be passed to the Document class when constructing new instances of documents.

Striing [] decodeDocument(InputStream inputStream)

Reads the document from the input stream and decodes it into lines of text

int countWords() // Returns the number of words in the document

SpellingMistakes [] spellcheck() // Spell checks the document and returns

the mistakes.

SpellingMistake is a class partially defined as follows

public class SpellingMistake {

private String word; // word in question

private int row; // row position in document

print int column; // column position in document

}

You should complete this class with constructors, getters and setters as part of your solution.

Please provide a Java solution to this problem using the Factory object pattern.

In your answer include a description of the Factory pattern **[6 Marks]** and code implementation of your solution to the problem written in Java. **[24 Marks]**

In your code you should use the Façade structure and provide access to the spell checking service using a singleton class called SpellChecker.

The code examples do not have to show the internal workings of the methods that involve decoding the documents or performing the spell checking. For your example assume there are 3 document formats, pdf, docx and txt.

Your factory class should produce an instance of a different concrete class for each format. Make sure your solution correctly follows the Myers open/closed principle.

You will need in your solution, an interface definition, an abstract class definition and 3 concrete class definitions.

**Explanation of Factory pattern [6 Marks]**

**Factory pattern**

**The factory pattern is used then an objects exact type is no know at compile time but will**

**ideally be decided later on at run time. [2 Marks]**

**The factory class has a method which takes in a parameter and decides at runtime which object instance to created. [2 Marks]**

**Each of the objects created confirms to a particular interface, so that they provide a similar service for a different context. For example for the document spell checking problem, the spell checking is the task required but it has to be done for different document formats. [2 Marks]**

// Java code 24 marks  
// 2 Marks for Exception class  
// 4 Marks for IDocument interface  
// 12 Marks for Document abstract base class  
// 3 Marks for concrete classes  
// 3 Marks for Singleton

**package** spellcheck;

**public** **class** BadDocumentNameException **extends** Exception {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** BadDocumentNameException(String message) {

**super**(message);

}

}

**package** spellcheck;

**import** java.io.FileNotFoundException;

**interface** IDocument {

**void** open() **throws** FileNotFoundException;

SpellingMistake [] spellcheck();

**int** countWords();

}

**Document.java**

**package** spellcheck;

**import** java.io.InputStream;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**abstract** **class** Document **implements** IDocument {

**private** String filename;

**private** String lines[];

**abstract** String [] decodeDocument(InputStream inputStream);

@Override

**public** **final** **void** open() **throws** FileNotFoundException {

InputStream inputStream = **new** FileInputStream(filename);

lines=decodeDocument(inputStream);

}

@Override

**public** **final** SpellingMistake[] spellcheck() {

// TO DO add in spell checking code

**return**(**null**);

}

@Override

**public** **final** **int** countWords() {

// **TODO** add in word counting code

**return** 0;

}

**final** **static** Document getDocument(String filename) **throws** BadDocumentNameException {

Document document=**null**;

**if** (filename==**null**) {

**throw** **new** BadDocumentNameException("Filename invalid, filename null");

}

filename=filename.toLowerCase();

**if** (filename.endsWith("pdf")) {

document=**new** DocumentPDF();

}

**if** (filename.endsWith("docx")) {

document=**new** DocumentDOCX();

}

**if** (filename.endsWith("txt")) {

document=**new** DocumentTXT();

}

// TO DO... add more documents in here

**if** (document==**null**) {

**throw** **new** Exception("Filename invalid, document postfix unrecognised");

}

document.filename=filename; // used by document processing code

**return**(document);

}

}

**package** spellcheck;

**import** java.io.InputStream;

**final** **class** DocumentDOCX **extends** Document {

@Override

String[] decodeDocument(InputStream inputStream) {

System.***out***.println("Decoding DOCX");

**return** **null**;

}

}

**package** spellcheck;

**import** java.io.InputStream;

**final** **class** DocumentPDF **extends** Document {

@Override

String[] decodeDocument(InputStream inputStream) {

System.***out***.println("Decoding PDF document");

**return** **null**;

}

}

**package** spellcheck;

**import** java.io.InputStream;

**final** **class** DocumentTXT **extends** Document {

@Override

String[] decodeDocument(InputStream inputStream) {

System.***out***.println("Decoding TXT document");

**return** **null**;

}

}

**package** spellcheck;

**import** java.io.FileNotFoundException;

**public** **final** **class** SpellChecker {

**private** SpellChecker instance=**new** SpellChecker();

/\*\* Private constructor for singleton

\*

\*/

**private** SpellChecker() {

}

**public** SpellChecker getInstance() {

**return**(instance);

}

**public** SpellingMistake [] spellCheck(String filename) **throws** FileNotFoundException, BadDocumentNameException {

Document document=Document.*getDocument*(filename);

document.open();

**return**(document.spellcheck());

}

**public** **int** wordCount(String filename) **throws** BadDocumentNameException, FileNotFoundException {

Document document=Document.*getDocument*(filename);

document.open();

**return**(document.countWords());

}

}

**package** spellcheck;

/\*\*

\* This class has to pass the spelling mistakes bac

\* **@author** coopes

\*

\*/

**public** **class** SpellingMistake {

**public** SpellingMistake(String word, **int** row, **int** column) {

**this**.word = word;

**this**.row = row;

**this**.column = column;

}

**private** String word; // word in question

**private** **int** row; // row position in document

**private** **int** column; // column position in document

**public** String getWord() {

**return** word;

}

**public** **int** getRow() {

**return** row;

}

**public** **int** getColumn() {

**return** column;

}

}

**Question 2** **[40 Marks]**

A web development company has a number of issues with its software development processes. This has been leading to poor customer satisfaction and after a detailed internal review the following issues were revealed.

1. Much of the code was poorly commented, complex and hard to follow

b) When developers left the company it was hard to other developers to   
 work on their code and sometimes the more complex parts of the

code have to re-written when used in new projects.

c) A lot of the projects seemed to be released with many bugs in the Beta

release stage, this leads to lack of confidence in the products.

d) Customers complained that the products were not to their satisfaction

with poor validation, so features which were not very important were

included in the final cut, while mission critical features were left out

e) The costing of projects are led by the sales team with advice from the

development team. Often the projects are under-costed leading to

either losses for the project or the project needing negotiating at the

later stages leading to poor customer satisfaction.

Explain how this organisation could apply techniques recommended in Agile, SCRUM and XP to deal with these issues. For each section you need to justify how your solution will help with the given problem. To get full marks please support your answer with relevant research if available.

Please structure your answer under 5 headings a,b,c,d and 5. You should include a bibliography.

**Marking criteria: 6 Marks for each part a, b, c, d and e. 10 marks for research and bibliography.**

**YOUR ANSWER, WILL BE PLAGERISM CHECKED, PLEASE DO NOT CUT/PASTE FROM ANY SOURCE.**

**If you need to quote that is ok but you should put in quotations and include source”.**

**Guidelines for word lengths no more than 600 words.**

**MODELS ANSWER**

A)

Pair programming, part of the extreme programming (XP) architecture, would help resolve these

issues. This is where two programmers work on the same computer, with one writing the code,

and one observing and reviewing the code/making suggestions. With pair programming,

developers are more focused, motivated, and much less likely to make use of poor

programming practices, due to ‘pair pressure’ - not wanting to let their partner down [1]. The

code produced will generally be of a higher quality, will have fewer defects/bugs, and will be

easier to understand.

However, pair programming isn’t without problems. Two programmers may solve the problem

faster than one, but in terms of programmer-hours, it is still likely inefficient. Some developers

find pair programming exhausting “because it requires heavy concentration” [2], and a study on

university students found that “pair programming was difficult when the skill differences between

the pair members were large” [2], meaning it might not always be the right choice.

The ‘user stories’ method of planning, which is used in the XP and SCRUM architectures, could

also help solve these problems. Using this approach, features are split down into smaller,

independent pieces of functionality, written from the perspective of the user, which are

implemented individually. As a result, the code will be produced as a number of smaller, simpler

parts, that are more modular/independent and easily testable. Code coupling is also reduced,

meaning the code is much easier to reuse. Overall, this style of splitting up tasks helps produce

much less complex and more readable code.

User stories aren’t without their downsides. A common criticism is that they don’t account for

non-functional requirements [3], which can cause an issue with cost or size estimation. They are

also informal and vague when compared to more traditional methods, such as use cases.

B)

Pair programming is part of XP, as discussed in part A. As well as the previously mentioned

benefits, programming in pairs also increases code ownership. Using pair programming

throughout development means that more people in the team will understand the code, simply

because more people have worked on it. If one developer leaves the company, there is likely

another person there who understands their code, because they worked on it with them.

The ‘user stories’ method of planning used in SCRUM and XP would also help to reduce this

problem. As described in part A, this splits complex problems and features into much smaller

independent pieces of functionality, resulting in less complex, easier to follow code, which is

more modular and has less coupling (so it is much easier to reuse). This approach does have

flaws, as discussed in part A.

C)

XP’s test-driven development style would be useful here. This is where tests are developed

before the code is written, and the code must pass all tests before it is considered complete.

This produces a very high quality and well-tested codebase, because most bugs are found and

fixed during development, before the code is released. Testing like this accumulates confidence

in the product over time, both for the developers and for the customer [4]. However, one study

has found that students find this style of development challenging [2], so perhaps it isn’t suitable

for the more inexperienced developers.

Pair programming (discussed in parts A and B) is another possible approach that could fix these

issues. With pair programming, mistakes are spotted much faster, and overall code correctness

increases, especially on more complex tasks [5]. As explained in part A, ‘pair pressure’ means

developers are less likely to implement poor practices in their code, meaning defects are less

likely and easier to spot. Inexperienced developers in particular perform much better with pair

programming, with one study finding a 73% increase in correctness [5]. Overall, pair

programming leads to higher quality code, with better design decisions and less bugs, although

it isn’t without problems, as discussed in part A.

D)

In SCRUM, the customer is part of the sprint planning meetings, in which they prioritise user

stories from the work backlog (a list of all user stories in the project). Involving the customer in

the planning process like this would resolve a lot of these problems, as the features the

customer considers critical would be prioritised by them during these meetings and would be

completed very early on in development. The less important features can be completed last, as

it is less important if these don’t make it into the final release. Customer satisfaction is increased

when using scrum, because the customer has much more involvement in the project, and more

control over the final product [6].

In XP, due to the test-driven development style (described in part C), the customer defines

acceptance tests for all functionality, and all code must pass testing before it can be released.

This way, simple issues like poor validation would not pass testing, and would not make it into a

software release. XP also prioritizes user stories during the planning phase, meaning the

dependent core code and high priority stories are completed first. However, XP expects the

customer to be on-site at all times, something that studies have found to be “stressful and

unsustainable for long periods” [2].

E)

Poker planning is often used for cost estimation in Agile architectures such as SCRUM, and

would help solve this issue. Poker planning works by having everyone provide their own

prediction at the same time, meaning predictions can’t be biased by other predictions made

before them. It helps eliminate anchoring, a problem where one member of the team attempts to

persuade the other team members to follow their estimate - such as a member of the sales team

underestimating the work in an attempt to win a contract. As well as this, studies show that

poker planning improves estimation accuracy when the team has previous experience with

similar tasks [7], reducing the chance of under-costing in the first place.

SCRUM and XP involve the customer very heavily in the development process, meaning any

potential cost issues can be resolved quickly as they arise, rather than near the end of

development. Because user stories are small, they are easier to accurately predict, and can be

used as a more precise measure of progress. This means that any potential problems with

meeting the original estimate will be made clear much earlier into the project.

Because SCRUM especially is very transparent, customer trust and satisfaction is in general a lot higher throughout development. One study found that customers gained more respect for developers with SCRUM projects, as it helped them understand “how easy it is for expectations

and results to differ without clear instructions and regular communication” [6]. When the customer is so heavily involved in the project, they are more likely to be understanding if there are problems - compared to other, more rigid, development styles, where they have very little

indication of progress before they are told the project will overrun.

However, Agile methodologies do have drawbacks in this area. With Agile, it can be hard to form and agree on a contract, due to the lack of formal documentation when compared to other approaches (such as waterfall). Because the requirements can change during the project, there is also the concern of feature creep (where the customer keeps asking for more features), which

can result in the project overrunning. Agile methods have shown to be hard to introduce for large and complex projects [2], and these projects are generally more difficult to estimate, so switching to agile might not always be the right approach for this specific problem.

Bibliography

[1] L. A. Williams and R. R. Kessler, "The effects of "pair-pressure" and "pair-learning" on

software engineering education," Thirteenth Conference on Software Engineering

Education and Training, Austin, TX, USA, 2000, pp. 59-65, doi:

10.1109/CSEE.2000.827023.

[2] T. Dyba and T. Dingsoyr, "What Do We Know about Agile Software Development?," in

IEEE Software, vol. 26, no. 5, pp. 6-9, Sept.-Oct. 2009, doi: 10.1109/MS.2009.145.

[3] F. Paetsch, A. Eberlein and F. Maurer, "Requirements engineering and agile software

development," WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on

Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003., Linz, Austria,

2003, pp. 308-313, doi: 10.1109/ENABL.2003.1231428.

[4] K. Beck, "Embracing change with extreme programming," in Computer, vol. 32, no. 10,

pp. 70-77, Oct. 1999, doi: 10.1109/2.796139.

[5] E. Arisholm, H. Gallis, T. Dyba and D. I. K. Sjoberg, "Evaluating Pair Programming with

Respect to System Complexity and Programmer Expertise," in IEEE Transactions on

Software Engineering, vol. 33, no. 2, pp. 65-86, Feb. 2007, doi: 10.1109/TSE.2007.17.

[6] C. Mann and F. Maurer, "A case study on the impact of scrum on overtime and

customer satisfaction," Agile Development Conference (ADC'05), Denver, CO, USA,

2005, pp. 70-79, doi: 10.1109/ADC.2005.1.

[7] N. C. Haugen, "An empirical study of using planning poker for user story estimation,"

AGILE 2006 (AGILE'06), Minneapolis, MN, 2006, pp. 9-34, doi:

10.1109/AGILE.2006.16.