Solution Software Modeling and Analysis - Exam

Given Name:	________________
Family Name:	________________
Matriculation Number:	________________

Please read this page carefully and fill in your name right away!

Examination date:	Wednesday, 19 December 2018
Version:	1.2
Time:	90 minutes
Total points:	60 (one point requires about 90 seconds of work)
Number of exercises:	5
Allowed materials:	a single ink-based pen (no pencil)

Please remember:

1. Some exercises are split over more than one page. Please quickly skim through all exercises first, before you proceed with the exam.
2. If you don't know something don't waste any time, go ahead and try to solve those questions at the end.
3. You are expected to answer each question concisely (brief and precise).
4. If you need more blank space for your answers, please use the back of your pages, and clearly state to which question your answer relates.
5. Please pay attention that your handwriting is easily readable by other people.
6. One final note: Cheating irreversibly leads to the grade 1.

Exercise 1 – General Questions (20 points | approx. 30 minutes)

Answer the following questions:

a) Why is bug prediction nowadays relevant? **(2 points)** **Answer:**

Modern software systems are way to large to test completely. Therefore a system to find the parts of code wich are more vulnerable to bugs is needed.

b) Why is duplicated code a problem in legacy software? **(2 points)** **Answer:**

Duplicated code arises naturally as a system evolves, as shortcut to implementing nearly identical code, or merging different versions of software systems. If the duplicated code is not eliminated by refactoring the common parts into suitable abstractions, maintenance quickly becomes a nightmare as the same code has to be fixed in many places.

c) Consider a method that detects dead code. Why are false negatives (= missed entities) of this method less problematic then false positives (= false alarms) when we want to automatically remove all code that was considered dead by this method? Explain! **(3 points)** **Answer:**

A false positive result for a method detecting dead code means that it categorized live, used code as dead. Removing this code would result in a malfunctioning program. False negative results on the other hand is code that is actually dead but categorized as alive by the method. Because we consider it alive we would not automatically remove it. False negative results means that our approach will fail at removing all dead code, but as long as we don't have false positives the program we modify will continue to work correctly.

d) Name and explain three code metrics of your choice. **(3 points)** **Answer:**

1) Cyclomatic Complexity: Equates complexity with control flow. The number of independent linear paths through a program. CC can be computed by representing control flow as a directed graph, and computing: CC = E - N + 2 where E = # edges and N = # nodes. Useful for analyzing test class coverage and time/effort required to maintain a class. 2) Depth of Inheritance Tree: The maximum depth level of a class in a given class hierarchy. The higher this value, the more methods the class inherits and the harder it is to evaluate its behaviour. 3) Access To Foreign Data: counts how many attributes from other classes are accessed directly from a given class. Gives insight to the interaction of a class with its environment.

e) What is the difference between intraprocedural analysis and interprocedural analysis? **(2 points)**
Answer:

Intraprocedural analysis only analyzes the body of a function, while interprocedural analysis analyzes the whole program with function calls.

f) What is the purpose of `ProtoObject` in the Pharo class hierarchy? **(2 points)** **Answer:**

ProtoObject is the root of the Pharo class hierarchy. While the Object class is usually functionally treated as the root of the class hierarchy (e.g. usually a class inherits from Object), the ProtoObject class is a light-weight class encapsulating a minimal set of messages must be understood by all objects. Since it is light-weight and the root of the hierarchy system behaviour can relatively easily be modified through modification of ProtoObject.

g) Name three reasons why the automation of data extraction in software analytics is important? **(3 points)** **Answer:**

Different data sources: Today, data can be gathered from lots of sources such as code, bug reports, mailing lists, customer feedback etc. Automation helps us to make the process of data extraction more efficient since we can profit from today's technology and high computation power. Furthermore, different metrics can be based on different sources, therefore an automated way to extract data from different sources is a necessity. To calculate e.g. the cyclo, we would require the source code, whereas for the number of commits we would use data from a repository.
Scalability: To count the number of methods for huge software systems with millions of lines of code is not feasible (well, you could count them, but that would be stupid and would take ages). Automation helps us to extract such data for large scale systems.
Time-sensitivity: Software systems change with time (Lehman's Law). Most of them quite frequently. E.g. Amazon deploys a new version of its system every few seconds. If it takes too long to gather and analyze the data, then the insights might not be relevant anymore, since the system already changed heavily in the meantime. Automation can speed up the process of data extraction.

Replication and comparison with historic data (and less error- prone): Gathering data manually is error-prone. Furthermore, different people might interpret data differently e.g. counting lines of code (do comments and empty lines count as well?). Therefore, automation makes sure that data is extracted always the same way and can be compared to previous extractions.

h) What is the difference between forward engineering and reverse engineering? **(3 points)** **Answer:**

Forward engineering is the process of moving from high-level abstractions and logical, implementation-independent designs to the low-level physical implementation of a system. Reverse-engineering is the opposite, we analyse the low-level physical implementation of a system and its components to create a highlevel representation of the system.

Exercise 2 – Smalltalk Coding (11 points | approx. 16.5 minutes)

Complete the provided (Pharo) Smalltalk code stubs.

a) Print out the message 'Hello World' to the Transcript window. **(3 points)** **Answer:**

```
________________ __________ : __________________________ .
```

Transcript show: 'Hello World'.

b) Sort the elements in the collection with respect to their `classDepth`. **(3 points)** **Answer:**

```
(Smalltalk allClasses) __________ : [ ________ | ___________________________ ] .
```

sorted: [:a :b | a classDepth > b classDepth]

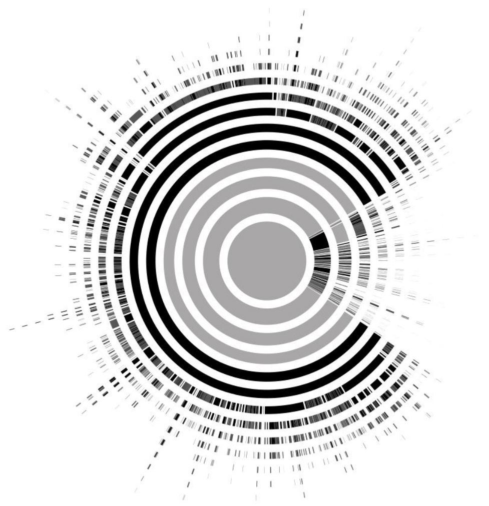
c) Complete the method below. The method is supposed to check if the input value in aString is equal to the string 'isValid'. If it is equal it should assign 'yes' to the result variable, otherwise 'no'. **(5 points)** **Answer:**

```
checkInput: aString
|result|
result := 'not available'.

________________________________

  _________ : ______________________

  _________ : ______________________ .

^ result.
```

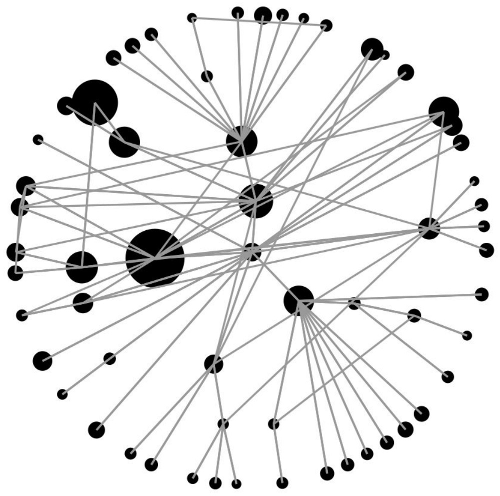
aString = 'isValid'
ifTrue: [result := 'yes']
ifFalse: [result := 'no'].

Exercise 3 – Visualizations (16 points | approx. 24 minutes)

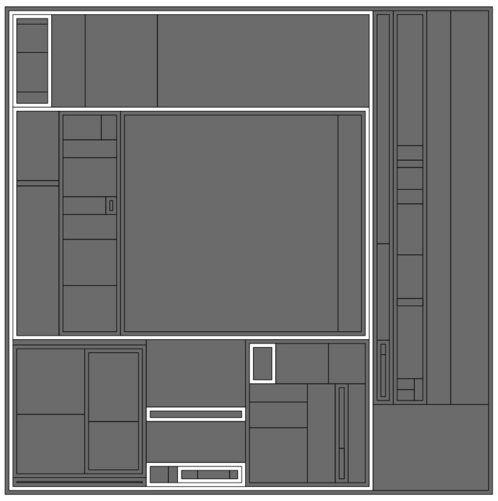
Consider the visualizations shown in Figure 1 and answer the questions.



(a) Sunburst visualization



(b) Node-link visualization



(c) Treemap visualization

Figure 1: Different code visualizations

You will find the questions on the next two pages.

Explanation of plots

The sunburst visualization in Figure 1a shows the distribution of lines of code for `Object` and all its subclasses. Hence, the class `Object` is in the center of the plot. The black segments are regular classes, while the grey segments represent classes that include test methods.

The node-link visualization in Figure 1b shows the dependencies between classes for the class `ArrayedCollection` and all its subclasses. Hence, the class `ArrayedCollection` is in the center of the plot. The size of the black circles (each representing a different class) is defined by the number of methods of the corresponding class.

The treemap visualization in Figure 1c shows the class hierarchy for `SequenceableCollection` and all its subclasses. Hence, the class `SequenceableCollection` covers the whole square and includes many smaller rectangles, each representing a subclass of `SequenceableCollection`. The size of the rectangles (each representing a different class) is defined by its number of methods. The white outlined rectangles represent classes that implement arrays.

Questions

a) Regarding the sunburst visualization: What could be a cause for the broken rings (the irregularity found in the right part of the visualization)? What is the problem with the test coverage? Why did the test coverage problem presumably occur? **(3 points)** **Answer:**

irregularity: due to classes not inheriting from main classes such as Class, ...
problem: test coverage is mainly present in core classes but not so much in classes at lower hierarchy levels
because the core classes are most important due to their frequent use, the developers spent presumably much time to keep the core functionality clean and correct

b) Name one drawback of the sunburst visualization shown in Figure 1a and explain why it is a drawback. **(2 points)** **Answer:**

waste of screen space due to the circular shape (more data could be visualized), the elements on the outer rings are overproporionally small (hardifies the interpretation), ...

c) Regarding the node-link visualization: Why does the central class `ArrayedCollection` only have a small circle in contrast to the one to the left (which represents the class `String`)? Is this a recommended practice? Justify! **(3 points)** **Answer:**

because it is rather an abstract class, its a shell that specifies basic behavior that every subclass has to follow. the implementation itself is done in the subclasses, for example, the string class which provides many different methods. That's why it is so large.
yes
it is recommended, because it increases the quality of the architecture: structural classes in the higher levels, implementational classes in the lower levels.

d) Name one drawback of the node-link visualization shown in Figure 1b and explain why it is a drawback. **(2 points)** **Answer:**

the lines are not well structured and crossing very much (confuses the reader), the black circles are presumably randomly aligned (waste of potential), waste of screen space due to the circular shape (more data could be visualized), ...

e) Regarding the treemap visualization: Why are not all subclasses outlined in white color? Why do nested white outlined classes exist? What could be an example of a class represented by a white outlined rectangle? **(3 points)** **Answer:**

Because not all are representing arrays
Because some array implementations apparently extend a base array class
*"*Array*" for example "IntegerArray"*

f) Name one drawback of the treemap visualization shown in Figure 1c and explain why it is a drawback. **(2 points)** **Answer:**

class inheritance is hard to compare between classes (it is hard to keep track of the "nested-ness"), the (nested) subclasses become scaled to the parent class size (biases the interpretation of an unaware reader), ...

g) What could be a more suitable visualization type for comparing the class hierarchies between the different classes for the treemap visualization in Figure 1c (except the two others, sunburst and node-link)? You do not have to refer to the visualization types available in Roassal, you can come up with anything as long as it is clear and reasonable. **(1 point)** **Answer:**

A tree with each class as node. Would improve clarity very much for comparing hierarchies between classes.

Exercise 4 – Static Analysis (7 points | approx. 10.5 minutes)

Consider the code in Figure 2 and answer the questions below.

```
int gcd(int a, int b) {
  int c = a;  int d = b; int e = 0; int f = 0;

  if (c == 0 && f == 0) {
    return d;
  }

  while (d != 0) {
    if (c > d) {
      c = c - d;
    } else {
      d = d - c;
    }
  }

  return c + 0;
}
```

Figure 2: Sample Java method

a) What does the method do? **(1 point)** **Answer:**

it calculates the greatest common divisor of two integer numbers

b) Describe three optimizations a Java compiler could apply to simplify the code in Figure 2 while maintaining its behavior. **(3 points)** **Answer:**

remove variable e (unused)
remove variable f and second if statement (no impact)
remove c + 0 in the return statement
replacing c with a, c gets removed
replacing d with b, d gets removed
moving the two int assignments beyond the first condition check

c) What is an intermediate representation (IR) language? What is its purpose? Provide an example of an IR language. **(3 points)** **Answer:**

an IR language is built closer to the hardware and less abstract. in other words, it uses a simpler vocabulary than high-level languages.
since it is less complex to parse and process, it is frequently used as a starting point for code analysis or code conversion.
Jimple, Java byte code, .net IR code, ...

Exercise 5 – Test Code Smells (6 points | approx. 9 minutes)

Consider the code in Figure 3 and answer the questions below.

```
public void test123() throws Throwable { {
  JSTerm jSTerm0 = new JSTerm();
  jSTerm0.makeVariable();
  jSTerm0.add((Object) "");
  jSTerm0.matches(jSTerm0);
  assertEquals(false, jSTerm0.isGround ());
  assertEquals(true, jSTerm0.isVariable());
}
```

Figure 3: Sample (Java) JUnit test case

a) Where do you find a test code smell? Why is it a test code smell? **(2 points)** **Answer:**

use of two assert statements in one test case
it complicates debugging of code that breaks this test, and also misleads developers

b) Do you find other bad coding habits? If yes, where do you find them, and what are the problems they could introduce? **(2 points)** **Answer:**

Yes, the use of ”test123” as test name. this is not providing any information about the content of the test to developers.

c) Provide two different reasons why testing can improve code quality. **(2 points)** **Answer:**

code becomes more modular due to the test separation
rethinking code while coding a test may lead to bugs otherwise undiscovered.