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, 16 December 2020
Version:	1.0
Time:	60 minutes
Total points:	60 (one point requires about 1 minute of work)
Number of exercises:	6
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. Don't forget to answer every aspect of a question, *i.e.*, reply to every question mark.
3. If you don't know something don't waste any time, go ahead and try to solve those questions at the end.
4. You are expected to answer each question concisely (brief and precise).
5. Please pay attention that your handwriting is easily readable for other people.
6. Cheating irreversibly leads to the grade 1.

Exercise 1 – General Questions (15 points | approx. 15 minutes)

Answer the the following questions:

a) Which class represents the root of GT's class hierarchy? **(1 point)** **Answer:**

ProtoObject

b) Where can you find a meta-model in Pharo? **(1 point)** **Answer:**

Metaclasses are a typical manifestation of the meta-model concept. Moreover, the class "Class" can be considered a meta-model for class instances.

c) What is the benefit of using a meta-model? **(1 point)** **Answer:**

A meta-model improves the modularity and portability of a design due to better separation of concerns.

d) Which two reflection abilities do you know? What is the difference between both? **(2 points)** **Answer:**

Introspection (reading) and intercession (reading and writing). The latter also supports changes of state.

e) Briefly explain the concept of static, dynamic, and hybrid code analysis each in one sentence. **(3 points)** **Answer:**

Static analysis: analysis of (source) code
Dynamic analysis: analysis of program execution
Hybrid analysis: a combination of both

f) In what sense is GT a live programming environment? **(1 point)** **Answer:**

Everything changed in the GT environment will apply immediately to the whole system with all the consequences.

g) Which threats may arise due to users unrestricted access to GT's system classes? **(1 point)** **Answer:**

The system state may become corrupted due to flawed user input. Unexpected or wrong results might occur.

h) What is the syntax for a comment in Smalltalk? **(1 point)** **Answer:**

Comments are enclosed by the double quote character, e.g., *"I am a comment."*

i) What is the difference between a string (`'HelloWorld'`) and a symbol (`#HelloWorld`)? **(2 points)** **Answer:**

A symbol is unique, wheras a string is not.

j) Provide one argument for the use of GT for (software) prototyping and one against it. **(2 points)** **Answer:**

Yes, because it supports developers in efficiently crafting a solution.
No, because Smalltalk is not very common compared to mainstream languages such as Java or C, and it is hard to find people to collaborate with on a global project.

Exercise 2 – Smalltalk Coding (10 points | approx. 10 minutes)

Answer the following questions:

a) What does the code below do? **(2 points)**

```
(((SystemNavigation default allClasses collect: [ :eachClass |
  eachClass -> eachClass classDepth]) sorted:[ :a :b |
    a value > b value ]) asOrderedDictionary) keys first
```

Answer:

The code finds the class with the longest inheritance chain among all Smalltalk classes in the GT programming environment.

b) Fill in the blanks to complete the method that returns all method names in GT. **(4 points)**

```
(( ______________________ ________________ ________________

  collect:  [ _______________ | _______________ _________ ])

  sorted:   [ ____________ | ____________ ]).
```

Answer:

((SystemNavigation default allMethods
collect: [:eachMethod | eachMethod name])
sorted:[:a :b | a < b]).

c) When do you need to use the `<gtView>` pragma? How do you use it? **(2 points)** **Answer:**

The pragma declares the method as "viewable" in the GT inspector. Hence, we need to set it if we would like to create a custom view in the GT inspector. The pragma must appear after the method declaration but before the method body.

d) What is a "live document"? Which problem frequently found in traditional code documentation does it solve? **(2 points)** **Answer:**

A live document is a document that contains executable code. As a result, it can update itself based on the results of the contained code.
Outdated or inconsistent documentation is a serious problem. Live documents avoid discrepancies in the documentation by design, because they evolve with the code.

Exercise 3 – Dynamic Smalltalk Coding (8 points | approx. 8 minutes)

Assume a `String` instance contains the following message implementation.

```
addMethod:  aString
  self class compile:
    aString, ':  aString', String cr, '^ aString size.'.
```

a) What does the method `compile:` do? **(2 points)** **Answer:**

It compiles the provided string and adds a reference to the compiled code to the instance's method dictionary, if the string contains code for a method.

b) Assume

```
  s := String new addMethod:  'myMethod'.
```

has been executed in the Playground on such a modified `String` instance. What is the content of the `myMethod:` implementation? **(3 points)** **Answer:**

myMethod: aString
^ aString size.

c) Assume

```
  s myMethod:  'Hello World'.
```

has been subsequently executed in the Playground. What is the returned result? **(1 point)** **Answer:**

11 (a SmallInteger object)

d) Name one benefit of dynamic coding and explain why it is a benefit. **(2 points)** **Answer:**

Methods can be manipulated programmatically. This is a huge benefactor for maintenance and automatization, e.g., methods can be changed without restarting the execution environment, and hot patches to running code can be applied without any human interaction.

Exercise 4 – Software Metrics and Analyses (8 points | approx. 8 minutes)

Consider the code in Figure 1 and answer the questions below.

```
int gcd(int a, int b) {
  if (a == 0) {
    return b;
  }

  while (b != 0) {
    if (a > b) {
      a = a - b;
    } else {
      b = b - a;
    }
  }

  return a;
}
```

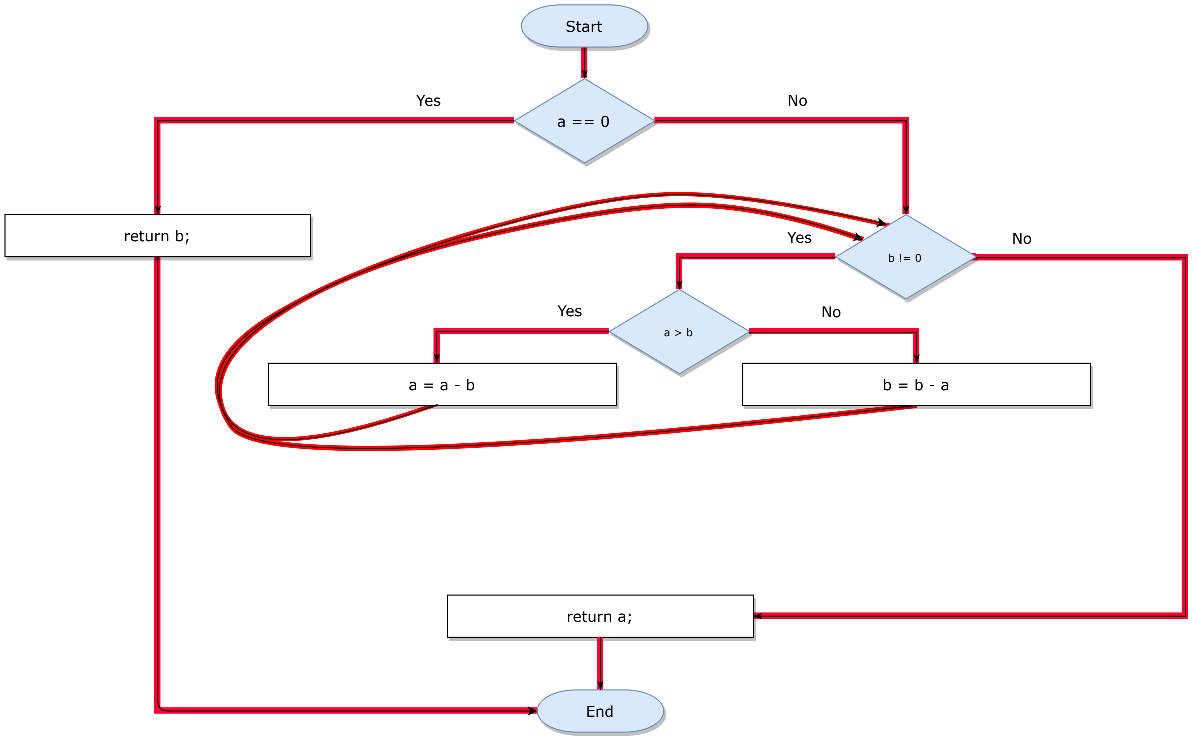
Figure 1: Sample Java method

a) Provide the common cyclomatic complexity formula we used in the lecture and explain each parameter of it. **(3 points)** **Answer:**

The formula is defined as $M = E - N + 2 * P$, *whereas* M *represents the resulting metric,*i.e., *the complexity value,* E *represents the number of edges of the CFG (possible different instruction flows),* N *represents the number of nodes in the CFG (instructions), and* P *represents the number of connected components (*$P = 1$ *for the analysis of a method).*

You will find more questions on the next page.

b) Draw the regular (non-interval) CFG for the code shown in Figure 1. **(4 points)** Answer:



c) What is the cyclomatic complexity of the code in Figure 1? **(1 point)** Answer:

4

Exercise 5 – Test Code Smells (10 points | approx. 10 minutes)

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

```
public class Compute {
  public static String mod(int number, int divisor) {
    if (number < divisor) {
      return number;
    } else {
      number -= divisor;
      return mod(number, divisor);
    }
  }
}

public void testMod() {
  // only run during night
  int currentHour = InternetTime.getCurrent(Calendar.HOUR);
  if (currentHour > 20) {
    int resultWithRemainder = Compute.mod(13,7);
    assertEquals(6, resultWithRemainder);
  }
}
```

Figure 2: Sample (Java) code with a JUnit test case

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

It computes the modulo operation.

b) The code of the static method contains multiple flaws. Identify one flaw and provide a pair of integer numbers that break the implementation, *i.e.*, lead to exceptions, errors, or wrong results. Provide the two numbers you found (*i.e.*, a `number` and a `divisor`) and briefly explain what the problem is. **(2 points)** **Answer:**

The code suffers from stack overflow errors when a divisor of 0 is provided, and it leads to stack overflow errors or broken results when the input contains negative integers. Because the bounds check is faulty, the recursion never stops (stack overflow) or skips inappropriately (negative numbers).

c) Write a test case for the flaw you just found. **(3 points)** **Answer:**

The provided solutions must be reasonable. They should exploit the discovered flaw. Two examples:

```
public void testModZero() {
    Assertions.assertThrows(StackOverflowException.class, () -> {
        resultWithRemainder = Compute.mod(13,0);
    });
}

public void testModNegativeNumber() {
    int resultWithRemainder = Compute.mod(-5,7);
    assertEquals(2, resultWithRemainder);
}
```

d) Do you find a *test code smell*? If yes, where do you find it, and what is problem it could introduce? **(2 points)** **Answer:**

Yes, the (i) use of "testMod" as test name. This misleads developers, because it does not comprehensively test the method and is too generic. (ii) An external resource is used within the test. Depending on its availability the test results could differ. A (iii) conditional in the test. This complicates the interpretation of results, because some code paths might not be tested.

e) Is *Feature Envy* considered a *code smell* or a *test code smell*? **(1 point)** **Answer:**

It is a code smell.

f) Is *Assertion Roulette* considered a *code smell* or a *test code smell*? **(1 point)** **Answer:**

It is a test code smell.

Exercise 6 – Fuzz Testing (9 points | approx. 9 minutes)

Please answer the following questions:

a) Briefly explain each of the terms *black box fuzzing*, *white box fuzzing*, and *grey box fuzzing*. **(3 points)** **Answer:**

Fuzzing is the process of creating new or altered input and then monitoring apps given the generated input.
Black box fuzzing: fuzzer knows nothing about the target application and its input.
White box fuzzing: fuzzer leverages data from program analysis and constraint solving to improve the quality of the results.
Grey box fuzzing: fuzzer has some knowledge about the program, i.e., *it is a combination of black and white box fuzzing.*

b) Answer the following questions regarding the `zzuf` tool:
(i) What is the purpose of the tool? **(1 point)**
(ii) Does the tool perform a static or dynamic code analysis? **(1 point)**
(iii) Is its output deterministic? **(1 point)** **Answer:**

(i) It is used to fuzz applications to find bugs or security vulnerabilities. zzuf is a transparent application input fuzzer and it can generate new input from existing data.
(ii) It is a dynamic code analyzer, because it needs to execute the application to gather results.
(iii) Yes, its output is deterministic.

c) Suppose you want to fuzz test a Smalltalk parser. What would be a suitable input string for a grey box fuzzer to start with? **(1 points)** **Answer:**

Any valid Smalltalk code snippet. For example:
Transcript show: 'Hello World'.

d) Suppose you use that input string. How could the input string look like after one iteration if you use a mutation-based fuzzer? Which mutation rule did you apply? **(2 points)** **Answer:**

Any answer is reasonable. The only requirement is that the provided code snippet must be altered and a rule must be followed (character substitutions, insertions, ..., randomness).

For example:
A rule that converts every non-letter character to a period would create this input after one iteration:
Transcript.show...Hello.World..