Methods in Software Engineering — Practice Exam

Prof. Dr. Gidon Ernst • Nian-Ze Lee, PhD • LMU Munich Summer Term 2022

Before the exam

- turn off your mobile phone and smartwatch now and store them away,
  if we catch you with a turned-on phone or similar we must regard it as cheating
- put any bags and jackets in the row in front of you

- write down your name and matriculation number

  Name:

  Matriculation Number:

When the exam time starts

- check that all 14 pages are included (the last page is spare)

Rules

- blue or black pen (not erasable, no pencil, no green/red),
- ok: mask, drinks, snack, ruler
- not ok: own paper, *any* other material, including:
  notes, books, calculators, dictionary

Time 90 minutes

Language

- you can answer both in English and/or German
- please ask if words or sentences are unclear

Grading ☐ please *do not* grade this exam ("entwerten")

Exercise	1	2	3	4	5	6	7	Σ
Points	of 7	of 12	of 6	of 14	of 10	of 11	of 15	of 75

Remarks

- the exam is subject to copyright and may not be distributed outside of this lecture
- this practice exam corresponds exactly to exam #1 held on August 8, 2022

1 Modern Software Development (7 points)

a) Briefly describe **version control systems**. (1 points)

b) Name **two** pieces of information that can occur in the description of an **issue**. (1 points)

c) What is a **regression** in the context of testing? Briefly describe or give an example. (1 points)

d) Briefly describe **continuous integration**. (1 points)

e) Name a technology that helps to create reproducible test environments. (1 points)

f) Name one **advantage** of unit tests over system tests. (1 points)

g) Name one **disadvantage** of unit tests over system tests. (1 points)

2 Domain-Specific Languages (DSLs) (12 points)

a) SQL is an example for a domain specific language, and so is the language defined in **b)**.

Give **one further** example: (1 points)

b) You are given the following grammar for propositional formulas in negation-normal form: You can assume that a suitable definition for `atomic-proposition` that accepts words like `A` or `isSunny` as strings.

```
formula     = conjunction | disjunction | literal;
literal     = atomic-proposition | "not" atomic-proposition;
conjunction = "(" formula "and" formula ")";
disjunction = "(" formula "or"  formula ")";
```

Here is an example formula accepted by the grammar:

```
(A and (B or not C))
```

Define a class hierarchy (UML, Java, or Python, just classes, attributes and associations, no methods), *or alternatively* Algebraic Data Types (Scala, Haskell, or similar), to represent the abstract syntax of the grammar. (6 points)

c) Give **one** use-case for the language from **b)** (e.g. an application scenario, or alternatively where it might be used in a larger language). (1 points)

d) Briefly describe the main distinction between "syntax" and "semantics" of a DSL? (1 points)

e) Describe the inputs and result of a suitable evaluation function for the language from **b)**, either by writing down its typed signature (parameters and return type in Java, Haskell, ...), or by describing the expected dynamic parameter and result type (cf. Python). (3 points)

Hint: Think about what kind of state you need to evaluate atomic propositions.

Do not give an implementation.

3 Refinement Types, Propositions as Types (6 points)

To express invariants over types, we have discussed the notion of "refinement" types.

a) Briefly describe what a refinement type is, i.e., what is different from a traditional type like `int`? (1 points)

b) Give an example for a refinement type, informally is ok. (1 points)

As a reminder, we have the following inductive data types that can represent logical conjunction $A \wedge B$ and disjunction $A \vee B$, as well as function types $A \to B$ that correspond to implication.

$$\texttt{data Pair}\ A\ B = (\mathit{fst} : A, \mathit{snd} : B)$$
$$\texttt{data Either}\ A\ B = \texttt{left}(a : A) \mid \texttt{right}(b : B)$$

Complete the missing parts below (type, function definition, formula). For example:

-) formula $A \implies A$
type $f : A \to A$
definition $f(x) = x$

c) formula $A \wedge (B \wedge C) \implies A \wedge C$

type (1 points)

definition (1 points)

d) formula (1 points)

type $\texttt{Pair}\ A\ B \to \texttt{Either}\ A\ B$

definition (1 points)

4 Component Design (14 points)

Consider a delivery service which picks up food from restaurants and delivers it to people's homes. We are concerned with the design of a web-service that is accessible to third-party web-sites or apps (such as an app for customers or an app for restaurants).

Specify the interface of the operation described below. Describe each **parameter** and **return value**. Explain how the underlined concepts below are represented in terms of their **type**, the respective **valid values**, and give an **example** for each.

The description can be informal (not tied to a particlar programming language) but should be precise (include all relevant details).

a) `GetRestaurantsForPostcode`: Returns the restaurants that are located within the area of a post code of interest. (4 points)

b) Between two addresses, find out the possible delivery options, which should each indicate: the estimated delivery time, and the fee/price for a devivery, and the maximal weight of the delivery. There may be multiple options for a each pair of addresses. (6 points)

The class `DatabaseConnection`, shown below, can be used to access a SQL data base. Assume that this class is already implemented. However, it is well-known that formatting SQL queries as a `String` has the security risk of injection attacks when programmers that use this class are not careful.

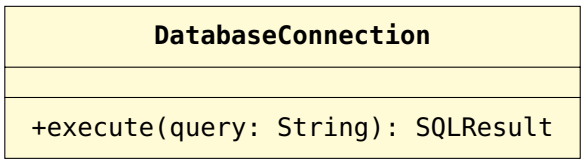
Therefore, we want to provide a method

```
+execute(query: SQLQuery): SQLResult
```

where `SQLQuery` is a structured object representing SQL queries, assuming we can convert an object of `SQLQuery` to its string representation.

We want to re-use class `DatabaseConnection` but unfortunately we cannot change it.

c) Extend the UML diagram below by a second class `BetterDatabaseConnection`, which realizes this change. (2 points)



d) Briefly discuss your design (there can be valid answers for all three choices below).

The change was incorporated using

☐ inheritance ☐ composition ☐ other, _____

because (2 points)

5 Liskov's Substitution Principle (10 points)

Consider two Java classes, **A** and **B** that both implement a common interface `BankAccount`, which should behave as you would expect from your own bank account.

```java
class A implements BankAccount {
  int balance = 0;

  void deposit(int amount) {
    balance += amount;
  }

  int getBalance() {
    return balance;
  }
}
```

```java
class B implements BankAccount {
  // an array with two integers,
  // initialized with 0's
  int[] transactions = new int[2];
  int counter = 0;

  void deposit(int amount) {
    assert(amount >= 0);
    transactions[counter] = amount;
    counter += 1;
  }

  int getBalance() {
    int sum = 0;
    for(int entry: transactions)
      sum += entry;
    return sum;
  }
}
```

There are multiple distinct violations of Liskov's substitution principle in the code above. Provide *histories*, which uncover these. Write *one event per line*.

Your histories should end in a mismatch in the result of **A** and **B**, where results are one of: return values of an operation resp. nothing (—) in case of `void`, or an exception. You do not need to include a call to a constructor as the first event.

a) Provide a history, in which a mismatch occurs because class **A** does not validate the inputs to one of its methods properly. (3 points)

step	operation name	parameter value	result for **A**	result for **B**
1.				
⋮				

b) How would you fix this problem in class **A**? (1 points)

c) Provide a history, in which the problem is in class **B**. (3 points)

step	operation name	parameter value	result for **A**	result for **B**
1.				
2.				
3.				
⋮				

d) How would you fix this problem in class **B**? (1 points)

e) Describe informally a possible scenario in development practice, where Liskov's substitution principle could be useful as a guide. (1 points)

f) We write $A \simeq B$ if two classes A and B satisfy Liskov's substitution principle. Draw a picture which expresses what this means for composition with a class C. (1 points)

6 Component and System Invariants (11 points)

a) Check all correct boxes: A component/class *invariant* is a/an (2 points)

☐ association (as in UML)

☐ variable

☐ formula

that may refer to

☐ the internal state

☐ method parameters

☐ execution history

b) Consider the following class, given in Java and in Python, where `N` is some unspecified positive number. Note, it is *different* from the shopping list in the lecture.

```java
class ShoppingList {
    int[] items = new int[N];
    int count = 0;

    void add(int item) {
/*(1)*/ assert <your task>;

        if(count != items.length) {
/*(2)*/     items[count] = item;
            count += 1;
        }
    }
}
```

```python
class ShoppingList:
  items = [0]*N # list of N entries
  count = 0

  def add(self, item):
    assert <your task>;

    if self.count != len(self.items):
      self.items[self.count] = item
      self.count += 1
```

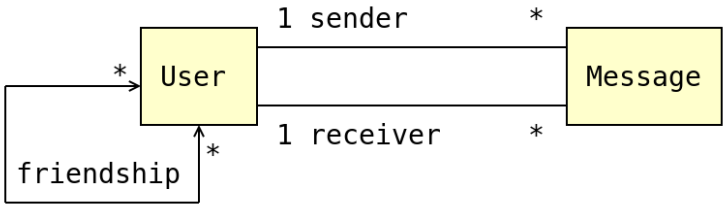
i. Read the subsequent question first. Then write down a formula mentioning `count` and `items`, which, when inserted as an assertion in line `(1)`, guarantees that the array/list access by index in line `(2)` is in bounds whenever it is actually executed. (1 points)

ii. Is your formula an invariant of the class **ShoppingList**? Note, depending on your answer to **i.**, either can be correct, but briefly explain why/why not. (2 points)

☐ yes ☐ no because

c) In some social network, users can send each other messages, but only if they are friends.

The following UML class diagram models users, messages, and the respective associations:



i. Which property of the `friendship` association must hold, such that it is possible to have a communication between two users? (1 points)

☐ reflexivity: a user is their own friend
☐ symmetry: friendship is always mutual
☐ transitivity: my friend's friend is also my own friend

ii. Describe informally an invariant that expresses that all three associations are consistent with each other. It is *not sufficient* to just repeat the shown multiplicities, or to just repeat the answer from **i.** (2 points)

d) Can an invariant *always* be expressed as part of the program source code? (1 points)

☐ yes ☐ no

Discuss one positive benefit and one potential negative **effect** of checking invariants as assertions in the source code (but do not answer: because it can resp. cannot be checked always).

i. positive (1 points)

ii. negative (1 points)

7 Mutation and Property-based Testing (15 points)

a) Briefly, what is the goal of mutation testing? (1 points)

The following function, given as Java and Python code, computes for two arrays/lists the length of the longest common prefix, i.e., how many common entries `a` and `b` have in the beginning.

```java
int lcp(int[] a, int[] b) {
    int k = 0;
    while(k < a.length &&
          k < b.length &&
          a[k] == b[k])
      { k += 1 }
    return k;
}
```

```python
def lcp(a, b):
    k = 0
    while (k < len(a) and
           k < len(b) and
           a[k] == b[k]):
        k += 1
    return k
```

You are furthermore given the following test-suite, in which we write arrays/lists using angular brackets e.g. ⟨ 1,2,3 ⟩ is array/list containing the elements 1,2,3 in that order.

input a	input b	result	comment
⟨ ⟩	⟨ ⟩	0	no elements to compare
⟨ 1,2 ⟩	⟨ 1,2 ⟩	2	two common elements in the beginning

b) Suggest a **mutation**, such as changing a constant or an operator, or adding/removing code fragments, that is *not* detected by the given test suite. (3 points)

in line: apply this mutation:

c) Propose **one new test case** that detects this mutation from **b)**. (3 points)

a = _____ and b = _____, with result = _____

If you have not answered **c)**, do suggest a test case nevertheless, you will get partial points if it uncovers *some* mutation, even if you do not say which one precisely.

d) Briefly, what is the goal of property-based testing? (1 points)

e) **Name** the three constituents that specify a *property* as discussed in the lecture. (1 points)

a.

b.

c.

f) Specify a property for the following informal statement: (3 points)

The first character of a string is the same as the last character of that string reversed.

Example: the first character of `"abc"` is the same as the last character of `"cba"`

Note: It is not required to use formal jqwik/Java/Python syntax but your intent should be clear. You may use `[ ]` for indexing strings. Your property should be as general as possible.

g) Similarly, specify a property for the following informal statement: (3 points)

If we concatenate a string to itself, then for each index in the first half of the result, the character at the corresponding index in the second half is the same.

Example: In `"abcabc"`, `a` at index 0 is the same as the character at index 3 etc…

Extra Page

If you use this page, please

- strike through those parts and solution attempts that should not be graded
- place a short note on the exercise sheet: “see extra page” or similar
- more paper is available on request, *please always return all sheets*