Software Engineering – Demo Exam

Name: _____

Student ID (Matrikelnummer): _____

Course of study (Studiengang): _____ **since** _____

Duration: 120 minutes (2 hours)[1]

Exam material: Writing instruments (pens and pencils); one A4 page with handwritten notes (please hand in with exam). The supervisors can hand out extra pages if needed.

Language: You can provide answers in English or German.

Sprache: Sie können Antworten auf Deutsch oder Englisch geben.

Assistance: Any questions? Ask the supervisors!

This exam has **15 pages.** Please check that all pages are present.

In this exam, you can achieve a maximum of **75 points**.
The exam is passed with **40 points** or more.

Problem	Max #points	Achieved #points
1 Requirements and Design	35	
2 Functional Testing	10	
3 Structural Testing	20	
4 Mixed Bag	10	
Total	75	

Points

Grade

Notes

[1] Note: this time applies to this demo exam. Real exams may cover more and take longer.

1 Requirements and Design [35 points]

The Software Engineering Chair at Saarland University has decided to set up a new coffee management system. These are the (informal) requirements:

1. A coffee machine dispenses coffee at the press of a button.
2. There are different coffee flavors (espresso, cappuccino, or latte macchiato) with different prices.
3. Every user has an account on the coffee management system.
4. Every user logs on to the system using some identification (a password or picture).
5. Users choose the coffees they had (or will have) and mark them as "dispensed".
6. The price for the coffee is automatically deducted from their account.
7. A special user (the "administrator") can recharge user's accounts.

a) [4 points] Consider the scenario

1. A student gets a coffee.

Create a *use case* (in Pressman style) for the above scenario including alternatives and exceptions.

b) [10 points] Design a *user interface* for the above use case as a series of *screen shot sketches*—again, including alternatives and exceptions.

c) [15 points] Develop a *class model* for the coffee management system:

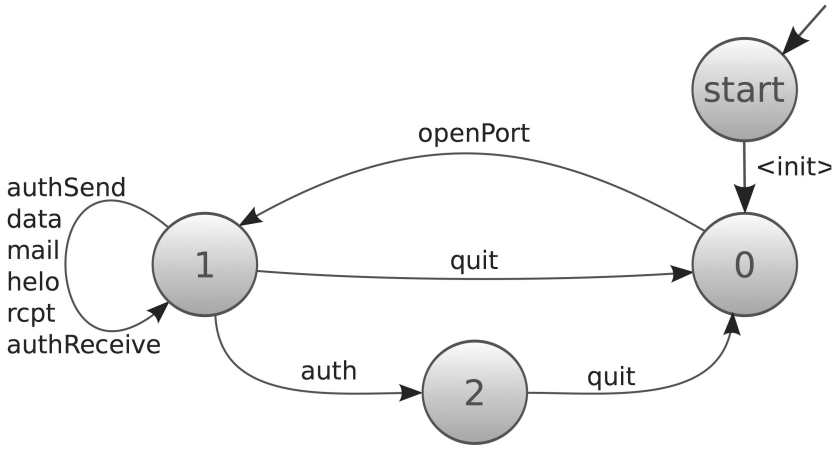
1. Start with CRC (Class-Responsibility-Collaborators) descriptions
2. Provide a UML system model listing
   - classes
   - attributes
   - essential methods, and
   - relationships.

Name: Student ID (Matnr):

Name: Student ID (Matnr):

2 Functional Testing [10 points]

Consider the enclosed description of the *SMTP protocol*, the most frequently used protocol for sending emails. The *states* are the states of the SMTP client; the *transitions* stand for messages sent. (Note: each of `authSend`, `data`, etc. is an individual transition.)



a) Design a set of test cases (as sequences of messages) that achieve *state coverage* with as little effort as possible.

b) Design a set of test cases (as sequences of messages) that achieve *transition coverage* with as little effort as possible.

Use the following format for sequences of messages:

```
<init> openPort helo data quit
```

3 Structural Testing [20 points]

The cgi_decode() function translates CGI encoding to plain ascii text:

```
/**
 * @title cgi_decode
 * @desc
 * Translate a string from the CGI encoding to plain ascii text
 * '+' becomes space, %xx becomes byte with hex value xx,
 * other alphanumeric characters map to themselves
 *
 * returns 0 for success, positive for erroneous input
 * 1 = bad hexadecimal digit
 */
int cgi_decode(char *encoded, char *decoded)
{
     char *eptr = encoded;
     char *dptr = decoded;
     int ok = 0;

     while (*eptr)  /* loop to end of string ('\0' character) */
     {
          char c;
          c = *eptr;
          if (c == '+') {  /* '+' maps to blank */
              *dptr = ' ';
          } else if (c == '%') { /* '%xx' is hex for char xx */
              int digit_high = Hex_Values[*(++eptr)];
              int digit_low  = Hex_Values[*(++eptr)];
              if (digit_high == -1 || digit_low == -1)
                  ok = 1; /* Bad return code */
              else
                  *dptr = 16 * digit_high + digit_low;
          } else { /* All other characters map to themselves */
              *dptr = *eptr;
          }
          ++dptr; ++eptr;
     }
     *dptr = '\0';   /* Null terminator for string */
     return ok;
}
```

a) For each of the following coverage criteria, provide a set of inputs to cgi_decode that achieves 100% coverage:

1. [4 points] Statement coverage
2. [4 points] Branch coverage
3. [4 points] Branch and condition coverage
4. [4 points] MC/DC coverage
5. [4 points] Loop boundary coverage

Use the following format:

```
cgi_decode("foo")
cgi_decode("bar")
```

4 Mixed Bag [10 points]

Please evaluate the following statements and check "T" if they are true, or "F" if they are false.
Grading: $+2$ points for each correct, -2 points for each wrong answer.
Non-checked or ambiguous answers are graded with ± 0 points.

1. [T] [F] *Functional testing* tests against the specification.
2. [T] [F] *Structural testing* attempts to find missing functionality.
3. [T] [F] The OO *hierarchy* principle defines team structure.
4. [T] [F] *Weyuker's hypothesis* states that coverage criteria are only intuitively defined.
5. [T] [F] *Code and Fix* is an important debugging pattern.