

ICS 121 / Informatics 111 - Software Tools and Methods
Final Exam – Fall, 2005

Last Name: KEY First Name: _____

1. (15 points, 5 points each) Define the following terms, as used in software engineering:

Fault Seeding or Bebugging.

Putting a known number of faults into a program.

Fagan Inspection.

A series of steps in which several people carefully study a software object with the goal of finding and fixing errors and incorrect assumptions.

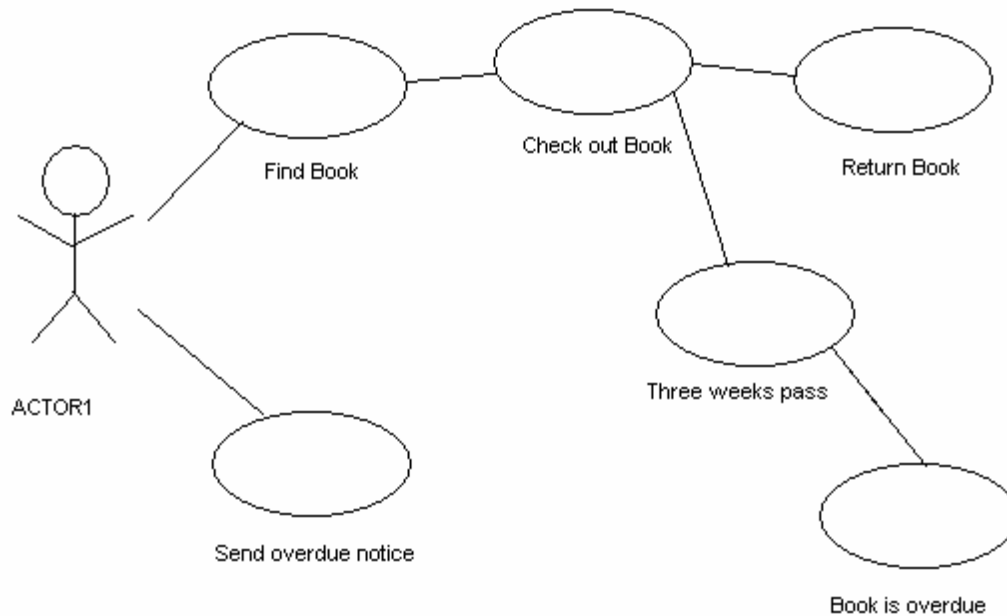
Multiplicities (in diagrams).

Indication of the number of objects in each of two classes participating in the relationship between the classes.

2. (14 points) The Rational Unified Process identifies four major phases. For each phase there are several activities and one milestone. Write down the names of the four phases, in order, and then draw lines from each phase to the associated activities and milestones.

Phase	Activity / Milestone
<u>1 Inception</u>	Life-cycle Architecture milestone 2 Rolling out fully functional system to customers 4 Building a system that operates successfully 3
<u>2 Elaboration</u>	Initial Operational Capacity milestone 3 Defining the scope of the system 1
<u>3 Construction</u>	Product Release milestone 4 Capturing a majority of functional requirements 2
<u>4 Transition</u>	Addressing significant risks on an ongoing basis 2 Life-cycle Objectives milestone 1 Outlining a candidate architecture 1

3. (16 points) Here is a Use Case Diagram:



Identify four specific and distinct errors in this Use Case Diagram. Don't worry about minor formatting issues.

Good answers:

ACTOR1 is not informative

Find Book – Check out Book – Return Book looks like a flow chart

Three weeks pass is not a Use Case

Book is overdue is not a Use Case, sounds like a FSM

Send overdue notice should have a different actor than the others

Not so good answers:

Lines should have arrow heads (2 pts. for this, it's optional)

Specifying a particular Use Case that is missing (4 pts for one of these, but no more points for additional ones; we don't know the domain well enough to critique the diagram on that basis)

Book is overdue is not a verb. (Maybe a good guideline, but isn't an error.)

Looks like a flow chart. (Not specific enough without naming names.)

4. (12 points) In the article *XP Distilled* by Miller and Collins, 12 practices of XP are discussed. Name and briefly describe three of those practices:

See the article. Each practice was worth 4 points.

i.

ii.

iii.

5. (4 points) Here's a quotation (not from this quarter's readings): "It is far better to adapt the technology to the user than to force the user to adapt to the technology" (Larry Marine). Which software tool that we studied this quarter do you think best follows this philosophy? Why?

A wide range of answers were accepted. 1 point for naming a tool, 3 points for explaining why.

6. (12 points) Abstraction is an important aspect of software design. How is abstraction used in UML? Identify one purpose of software design and explain how the UML use of abstraction you discussed furthers that purpose.

Many answers were possible. 4 points for "How is abstraction used in UML?" 4 points for a purpose of software design. (e.g. communication, clarifying one's thoughts, documentation, guiding the developers) 4 points for "how furthers?"

7. (12 points) Identify and briefly describe three quality assurance activities that are appropriate to perform during the Design phase of a software project:

See lecture slide #3 for week 5. 4 points for each activity. Common mistakes were naming activities that take place outside the Design phase (e.g. unit testing); naming activities that are not related (fairly directly) to quality assurance (e.g. interviewing the customer to assess requirements); and naming software qualities that are Good Things but which are not activities (nor directly related to quality assurance) (e.g. Information Hiding).

8. (15 points) At your new company you've rapidly won the informal title of "JUnit Guru" because of your deep knowledge of both JUnit and testing procedures. One day, a colleague says to you, "I'm having trouble developing JUnit test cases. They seem to be silly and unnecessary. For instance, I just wrote this method:

```
/** method isPrime
 * Returns true if parameter is a prime number.
 * Only needs to work on integers between
 * 2 and 20.
 */
public boolean isPrime(int n)
{
    if (n == 2 || n == 3 || n == 5 || n == 7 ||
        n == 11 || n == 13 || n == 17 || n == 19)
        return true;
    else
        return false;
}
```

Now I'm supposed to write a JUnit test case. What's the point? I'll just repeat in the test case the same code from the original isPrime method."

Write a short memo explaining to your coworker the benefits and proper use of JUnit. Talk specifically about this particular isPrime() method, and also refer to either

- a practice of Extreme Programming, or
- one of Brooks' four inherent properties of the irreducible essence of software engineering.

Lots of good memos. 5 points off if neither XP nor Brooks were discussed.

Many people mentioned that JUnit should be used to test how the isPrime() method will react when the parameter has an incorrect type, e.g. String. Very mysterious. I'll have to see if I can figure that out over the break.