VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

EXAMINATIONS — 2004

MID-YEAR

COMP 462

OBJECT-ORIENTED PARADIGMS

Time Allowed: 3 Hours

Instructions:

- *Read each question carefully before attempting it.*
- This examination will be marked out of **120** marks.
- Answer all questions. Each question has the same value, and should take approximately 30 minutes to answer.
- You may answer the questions in any order. Make sure you clearly identify the question you are answering.
- Many of the questions require you to discuss an issue, or to express and justify an opinion. For such questions, you will be assessed on your answer, the *evidence* you present, and any *insight* based on these.
- Some of the questions ask for examples from object-oriented languages. Your answers need only refer to object-oriented languages discussed in the course, but you may refer to other languages if you wish.
- Non-electronic foreign language-English dictionaries are permitted.

Question 1. Object-Orientation [20 marks]

In Smalltalk, *"everything is an object"*.

Discuss how this principle is realised in the design of at least *two* other object-oriented languages covered in the course — such Simula, Java, Self, or C++, or other object-oriented languages you may have studied.

Question 2. Design Techniques and Methodologies [20 marks]

(a) [10 marks] *CRC cards* are used in both XP and RDD. Explain how and why.

(b) [10 marks] In XP, you write *user stories*; in many other object-oriented methodologies you write *use cases*. What are the differences between user stories and use cases? Why?

Question 3. Software Quality [20 marks]

Explain why (or why not) *metrics*, *heuristics*, and *patterns* can stop a software project turning into a *Big Ball of Mud*. Your answer should include at least one example each of a metric, an heuristic, and a pattern, explaining how that example helps (or doesn't!).

Question 4. Software Design [20 marks]

In *Designing Reusable Classes*, Ralph Johnson and Brian Foote include the following design rules (amongst many others):

- **Rule 2:** Eliminate case analysis.
- **Rule 4:** Reduce the size of methods.
- **Rule 7:** Minimize accesses to variables.
- **Rule 12:** Send messages to components instead of to self.

For *each* rule, explain why it should lead to a better object-oriented software design.

Question 5. Frameworks and Patterns [20 marks]

Which comes first, frameworks or patterns?

You may wish to consider that:

- Patterns can be discovered by examining the design of frameworks.
- Frameworks can be constructed by using design patterns.
- Patterns can be used to describe frameworks.
- Frameworks can be used to give examples of design patterns.

Question 6. Programming Languages [20 marks]

Macrosoft corporation has introduced the design of a new programming language C◦, pronounced "See Through". C◦ is basically the same as Java, but has no inheritance or interface declarations (that is, no `extends` or `implements` keywords).

Rather, any C◦ class declaration can incorporate any number of `include` statements. These `include` statements copy all the field and method declarations from the included class into the new class being declared.

For example, in the code below, all the fields and methods in the `Animal` class will be copied into the `Cat` class. Then all the fields and methods in the `Cat` and `Fish` classes (including the ones copied in from `Animal`) will be copied into the `CatFish` class. The `Cat`, `Fish`, and `CatFish` classes have other fields and methods declared directly, of course.

```
class Animal {
 public void eat() {};
};


class Cat {                          class Fish {
 include Animal;                      public void swim() {};
 public void purr() {};               protected String name = "fish";
 protected String name = "cat";      }
};


class CatFish {
 include Cat;
 include Fish;
 protected String colour = "blue";
 public bool feelers() {return true;};
}
```

Critique the design of the `include` feature in C◦. You could discuss C◦'s support (or lack of support) for: Code Reuse, Multiple Inheritance, Subtyping, Subclassing, Polymorphism, and other relevant issues.

****