UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2013 EXAMINATIONS

CSC302H1S

Duration – 3 hours

No Aids Allowed

Department: Computer Science
Instructor: Matt Medland

This test counts for 35% of your final grade

Name: _____
(Please underline last name)

Student Number: _____

Question Marks

1 _____ /10
2 _____ /35
3 _____ /15
4 _____ /15
5 _____ /25
6 _____ /25
7 _____ /30

Total _____ /155 = _____ %

1. [Large Software Systems, 10 marks Total]
(a) [Measuring Software Size – 5 marks] The focus of this course is developing large software systems. As discussed in lecture, there are many ways one might think about the size of a software development project or system. List at least five (5) metrics that could be used to classify the size of a software system.

Lines of code (LOC), number of developers, person-hours of effort, number of users, number of processors running the software, number of bugs, etc.

(b) [A Better Definition – 5 marks] Size alone, as measured by one or more of the metrics listed in (a) above, does not necessarily mean that the software is an appropriate candidate for the tools and techniques taught in this course. Give a better definition of the type of software system we are concerned with in this course.

Any software that is considered non-trivial, and that benefits from proper planning and tools. This software is usually used by someone other than the developer. The software is usually subject to multiple releases and revisions. The software is classified as E-type (Embedded) by Lehman.

2. [UML Diagrams, 35 marks Total]

(a) [Reverse Engineering UML Class Diagrams – 20 marks] Given the three classes below, draw a UML class diagram showing all fields, modifiers, types, operations and relationships between them. Use page 4 (and the scratch paper at the back of this package, if necessary) to draw your class diagram.

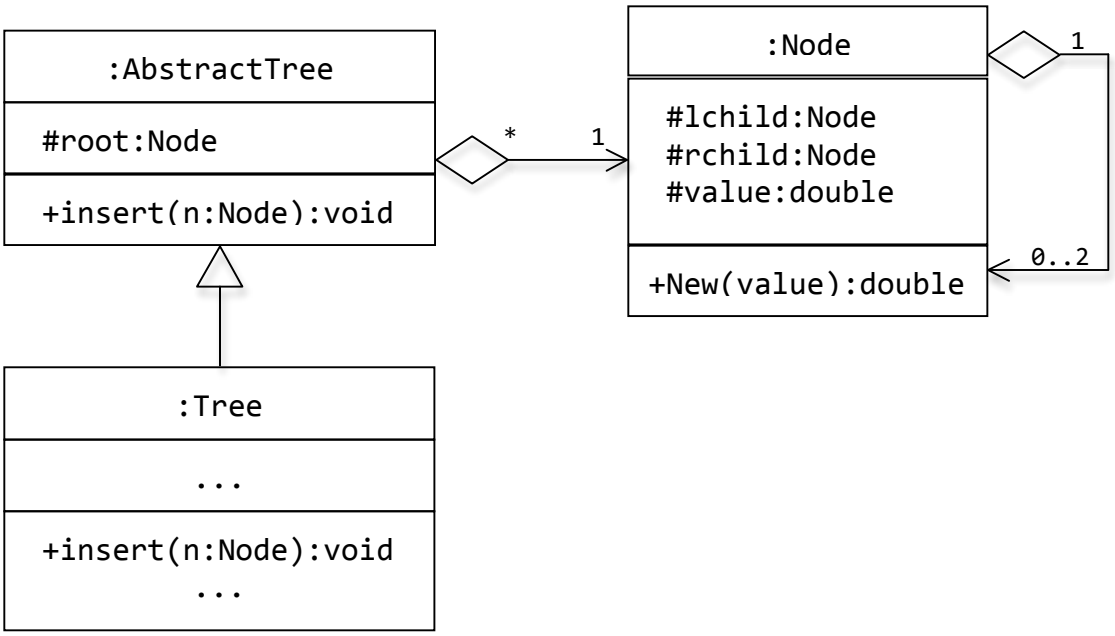
```
public class Node {
    protected Node lchild, rchild;
    protected double value;

    public Node(double value) {
        lchild = null;
        rchild = null;
        this.value = value;
    }
}
```

```
public abstract class AbstractTree {
    protected Node root;
    public abstract void insert(Node n);
}
```

```
public class Tree extends AbstractTree {
    public void insert(Node n) { ... }
    ...
}
```

[answer question 2(a) here]



Marks for: (1) all attributes identified (2) all operations identified (3) at least one constructor, preferably for the Node class shown (4) all visibility modifiers shown (5) operations parameter and return types shown (6) attribute types (7) all three classes shown with class names (8) Tree->AbstractTree generalization (9) AbstractTree<>->Node aggregation, and (10) correct multiplicities (11) Node<>->Node aggregation, and (12) correct multiplicities.

(b) [Sequence Diagram, Interaction Frames – 10 marks] Within a sequence diagram, there is often a need to group a set of messages together. These groupings are sometimes referred to as *Interaction Frames*, or *Combined Fragments*. The most common reason for using one of these frames is to show alternate, or repeated execution based on a given *guard expression*. We had an example of this in the midterm test where we used a **loop** frame with guard **[!isDone()]** while modeling the Iterator pattern. Give the definition for the following five (5) most common frame types:

- alt

Conditional execution like an if-statement with an else clause.

- opt

Like an if with no else.

- par

Messages in this frame can occur in any order or at the same time.

- loop

Iterative execution of statements in frame.

- region

Ctitical section or “region” of code that can only have one thread executing the messages at a time.

(c) [Sequence Diagram Uses – 5 marks] As software designers, sequence diagrams are very useful tools to help shed light on possible implementation problems, and for getting one step closer to implementation from design. List a couple development activities where sequence diagrams can help us, and how. Although not required, it may help to think of some of the definitions from the previous question.

ex. Comparing design options: sequence diagrams graphically depict alternative behaviors and messages using alt frames.

Finding bottlenecks and potential performance issues. A sequence diagram can show if one object has a disproportionate number of messages routed through it, and hence it may be responsible for too much and could become a performance issue.

Elaborating use cases. Shows how the user interface functions and how the user expects to interact with the system.

Depicts opportunities for multi-threading and parallelism. Use case diagrams do not show this explicitly, but sequence diagrams can use **par** frames.

Other reasonable answers possible

3. [Software Processes, 15 marks Total]

(a) [Waterfall – 5 marks] List some of the problems or drawbacks associated with the waterfall model.

Static requirements, ignores that they may change. Lack of user involvement between specification and UAT at the very end. Unrealistic separation of specification from design. Awkward, and inefficient that each step must be complete before moving on to the next. Often progress is tracked using Gantt charts.

Other reasonable answers possible.

(b) [Gantt Charts – 10 marks] Gantt charts are often used to do project planning, and in particular they are popular when using a waterfall-like software process. What are some of the potential problems associated with using Gantt charts when managing a software development project?

They contain too much detail for early stages, ex. who is assigned to what task, the prerequisites between tasks, etc. Gantt charts take a very long time to develop and are very difficult to change. Developers can (more or less) be used interchangeably so it does not matter who is assigned to what throughout the entire release cycle. Gantt charts only contain implementation tasks, not used for testing or other activities.

4. [Risk Assessment, 15 Marks Total]
Ideally, we would like to assess risk by quantitatively calculating each individual risk exposure and the leverage of each mitigating action. In some cases this is either impractical or impossible. In the cases where a quantitative calculation is not feasible, describe what types of risk assessment can be done, and what tools we can use. Give an example risk exposure matrix using the Therac-25 X-ray device discussed in lecture.

Qualitative analysis is the alternative if quantitative is not possible. Explain qualitative analysis a bit...

		Likelihood of Occurrence		
		Very Likely	**Possible**	**Unlikely**
Undesirable Outcome	**Patient dies**	Catastrophic	Catastrophic	Severe
	Suffers permanent injury	Catastrophic	Severe	Severe
	Patient injured	Severe	Severe	High
	Temporary discomfort	High	Moderate	Low
	X-Ray takes too long	Moderate	Low	Low

Marks given if the undesirable outcomes make sense given the context (i.e. they are bad effects on patient). The risk scale (ex. High, medium, low) must be appropriate as well.

5. [Testing, 25 Marks Total]

(a) [Usability Testing – 10 marks] During verification & validation, or V&V, we often have users test a prototype and observe their activities. This is often referred to as usability testing. As discussed in lecture, it was suggested by Jakob Neilson that the best number of users for such activities is between three (3) and five (5). However, it is not necessarily true that 3-5 is the correct number all the time. List some factors to consider when determining the number of testers for a particular system, and why they are important.

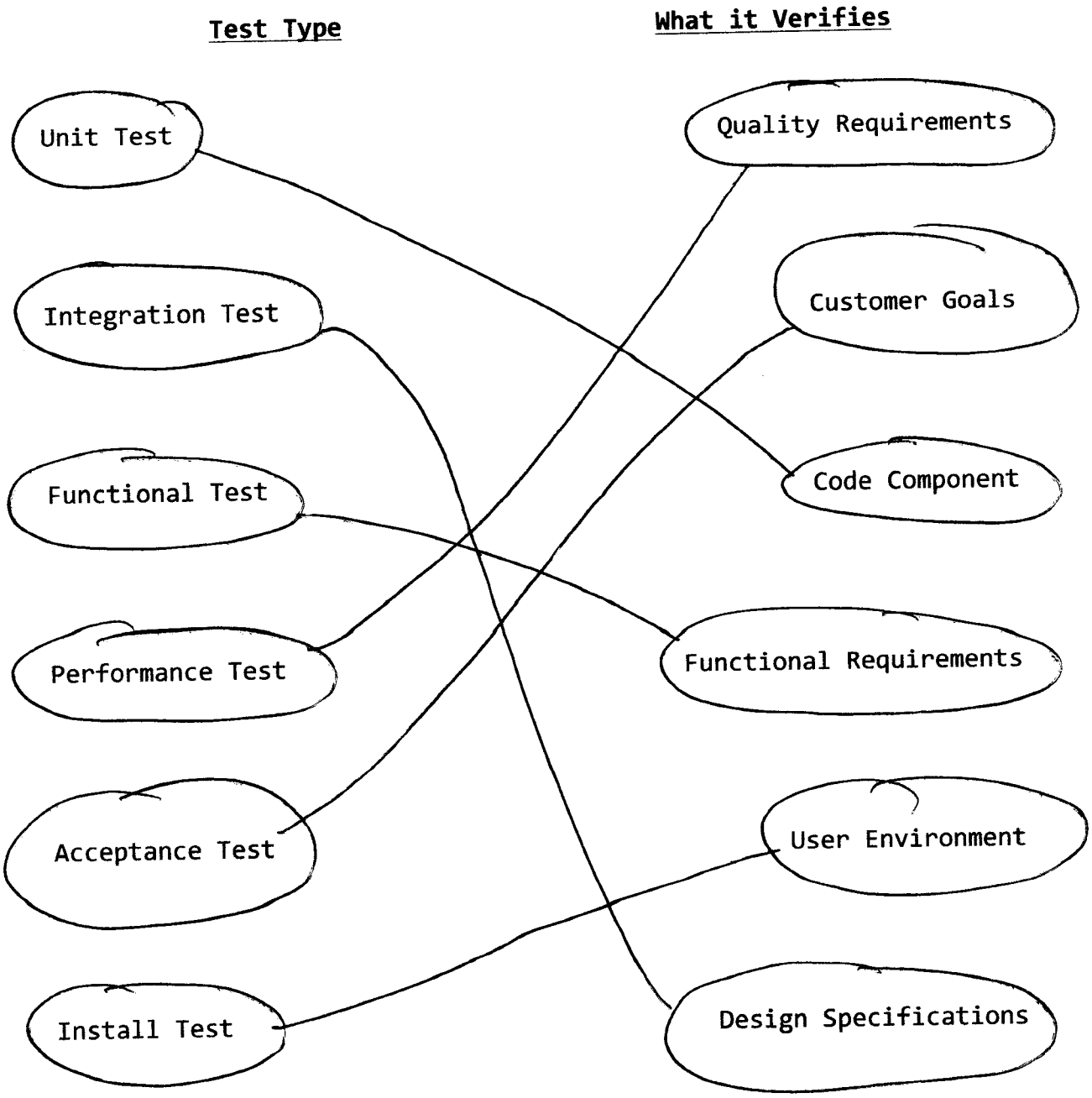
Age of the system and its current stability level. If it is a brand new system that has never been released before there are likely quite a few easy to find bugs. May want to limit to only a few people at first expecting each of them to find large numbers of defects.

Expert level required to use the system. If you are developing a flight simulator you would likely get a lot more valuable feedback from a single pilot than from numerous members of the general public.

Cost of each user. Again using the flight simulator example, a pilot is likely an expensive resource to have testing a system. A small number of skilled pilots will likely find the same bugs as a large number of pilots, so best to keep the numbers low to save money. Neilson’s advice may be appropriate in this case.

Many other acceptable answers.

(b) [Types of Testing - 10 marks] For each type of test listed below draw a line to the item or concept it is used to verify.



(c) [Test Coverage – 5 marks] Briefly describe the differences between structural coverage (white box) and functional coverage (black box) testing. Emphasis here is on Coverage.

Structural coverage means that every part of the code, including all conditional branches, is covered by some test case.
Functional coverage typically means that every type of input is tested, including valid input, edge (or corner) cases and error cases.

6. [Software Quality, 25 Marks Total]

(a) [Capability Maturity Model – 15 marks] Name, and give a brief description of, each of the five (5) levels of the Capability Maturity Model (CMM).

One mark for the name, and two for a good description.

1. Initial: Ad-hoc development with no estimation or project management to speak of.

2. Repeatable: Process depends on individuals in the group.

3. Defined: There is a documented process in place and it is used throughout the software development team/organization.

4. Managed: Process metrics are measured. For example, estimate vs. actual for feature sizing.

5. Optimizing: Feedback into process for continued improvement.

(b) [Six Sigma & Software – 10 marks] List some reasons why Six Sigma might not be appropriate for judging software.

Software processes are “fuzzy” and depend on human behavior. Hard to establish and measure a degree of conformance. Can’t actually accurately measure the number of faults in a piece of software. Prescribes a failure level of 0.0034 faults/KLOC, where the shuttle had 0.1 faults/KLOC and cost approx. $1,000 per LOC! So not cost effective, or even attainable?

7. [Release Planning, 30 Marks Total]

(a) [Balanced Resources – 5 marks] A development plan is like a financial balance sheet. In finance, the sheet is balanced if your payables are equal to your receivables. What is the counterpart to this in a software development plan?

Your capacity to do development work must be at least as great as the amount of (estimated) work you have to do.

(b) [The Capacity Constraint – 10 marks] In the lectures, we defined three (3) quantities that relate to capacity and requirement. Namely *F*, *N* and *T*. These quantities are related to each other by the equation $F = N \times T$ or, preferably $F \leq N \times T$. Give a definition for each of *F*, *N* and *T* and describe how they relate to software development capacity and requirement.

F is the total amount of feature work planned for a given release. It is also, by definition, the requirement.

N is the number of developers (8 hour developer-day equivalent's) assigned to do feature work on a given release.

T is the amount of time (in days) for a given release.

N × T is the capacity, or rather, the amount of work the team can accomplish with the given amount of time. N × T should be greater than F at all times or else the capacity constraint is violated.

(c) [Balancing the Plan – 10 marks] If at some point the plan becomes unbalanced (i.e. the capacity constraint is violated) there are some mitigating actions that can be taken. List some of these actions and their expected effectiveness & consequences.

1. Cut features. This decreases the requirement (*F*) and can bring a plan back into balance but has the negative impact of delivering less functionality to the customer.

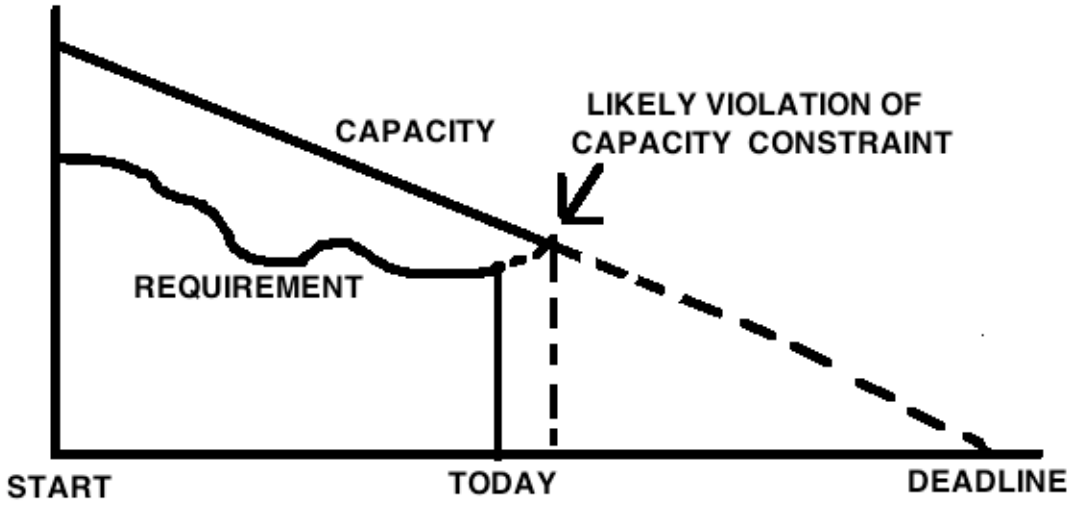
2. Extend the deadline for the release. This increases capacity by giving the existing development team more time (increased *T*) but has the negative impact of delaying the availability of the new functionality to the users.

3. Add developers to the plan. This will increase capacity by adding more effective developers to the plan (*N*). This has the positive effect of delivering on time and the negative effect of costing more in terms of developer salary. This strategy rarely works in practice because the new developers have a learning curve that may nullify their positive contribution.

(d) [When to Take Action – 5 marks (+5 bonus)] It is not usually a good idea to wait until a plan is out of balance (i.e. capacity constraint is violated) before taking a mitigating action. Name a type of graph that can help to make this decision earlier, and describe how to keep it up to date with current and historical information. Bonus points for drawing an example.

The burndown graph (or any other equivalent variant) can help the project manager determine ahead of time when a project could get in danger and react before a crisis is upon them. The burndown graph can be used to track progress and essentially take a derivative of progress to release. This derivative, or rate in the change of requirement, can help the manager predict when a problem may occur (i.e. capacity constraint violation) and react early.

Up to 5 bonus points for a good burndown graph example:



[scratch paper]

[scratch paper]

[scratch paper]