Module Title: Inf2C-SE
Exam Diet (Dec/April/Aug): Dec 2015
Brief notes on answers:

1. (a) *Bookwork.* Other techniques discussed in lecture include using interviews, facilitated meetings, observation and prototypes.

   2 marks for each of 2 techniques.

   (b) *Application of knowledge, Problem solving.* Use cases include:
   1. recognising an item placed in the fridge;
   2. recognising an item removed,
   3. Updating the highlighting of items on the display when new items become near their use-by dates.
   The main actor is the user for 1-2. In the tutorials and coursework, the notion was introduced of treating a system timer as an actor which initiates use cases at specific times. So for 3, the expectation is that the timer will figure as an actor. The user might also figure as a secondary actor, since the user is expected to read the display.

   3 marks for suitable use cases, 1 for the user as an actor, 1 for the timer as an actor, and 2 for the general well-formedness of the diagram.

   (c) One answer is to have the following classes:

   - Fridge: findItemsNearUseByDate(), addItem(), removeItem()
   - Timer: generateTimePulse()
   - CameraSystem: addItem(), removeItem()
   - Display: showItemsWithHighlights()
   - Item: description, useByDate.

   with the latter 4 all associated to the Fridge class. In tutorials and the coursework, the idea of was introduced of having classes for I/O devices supporting operations for primary I/O events. 2.5 marks for the classes, 2.5 for the operations, 1.5 for the associations, and 1.5 for suitable multiplicities. The marks are both for the existence of the respective entities and the correct UML syntax.

   (d) *Bookwork.* From lecture:

   - *Inception* ends with commitment from the project sponsor to go ahead: business case for the project and its basic feasibility and scope known.
   - *Elaboration* ends with basic architecture of the system in place, a plan for construction agreed, all significant risks identified,
   - *Construction* ends with a beta-release system .

   1 point for each name, 1 point for each deliverable example.

2. (a) (i). *Bookwork/Application of Knowledge*
TFD and TDD processes rely on rerunning tests frequently after each feature addition, in order to check the code is in a good state, that code changes have not inadvertently introduced problems elsewhere in the code. Automation of tests is essential to make this frequent rerunning of tests practical.
The value of frequent test rerunning is not confined to TFD and TDD. The same argument applies whether the tests are written before or after each code increment. And even when tests and code are not development incrementally together, rerunning tests is needed after each bug fix for the same reason.
Marking:
2 marks for why automation is needed for TFD and TDD.
2 marks for sensible remarks on the wider value of test automation

(ii). *Problem solving.*

```
import static org.junit.Assert.*;
import org.junit.Test;
public class StopWatchTest {
  @Test
  public void testIncrement() {
    StopWatch s = new StopWatch(2,59);
    s.increment();
    assertTrue (s.minutes == 3 && s.seconds == 0);
  }
}
```

1 mark for test class and method, 1 mark for object creation, 1 for calling the method, 1 for checking result. No penalty for omitting the import(s).
Problem solving

(iii).
```
//@ ensures seconds == (\old{seconds} + 1) % 60;

//@ invariant 0 <= seconds && seconds < 60 && 0 <= minutes;
```
2 marks for correct post-condition, 2 for the invariant.

(iv). *Bookwork.* Run-time checking can usually only check a very small fraction of possible inputs to methods, whereas static checking formally proves methods correct for all inputs. The main problem with static checking is that it doesn't always work. The checking needs might be beyond the checking technology.
1.5 marks for the advantage, 1.5 marks for the problem.

(b) *Bookwork/Application of knowledge:* The other security requirements mentioned in class are:

- integrity – information cannot be tampered with
- authentication – parties are who they say they are
- authorisation – parties can only do what they should be able to do
- non-repudiability – agreements made cannot be denied later, c.f., digital signatures.

Marking:
1 mark for each of two names,

2 marks for each reasonable example of a violation.

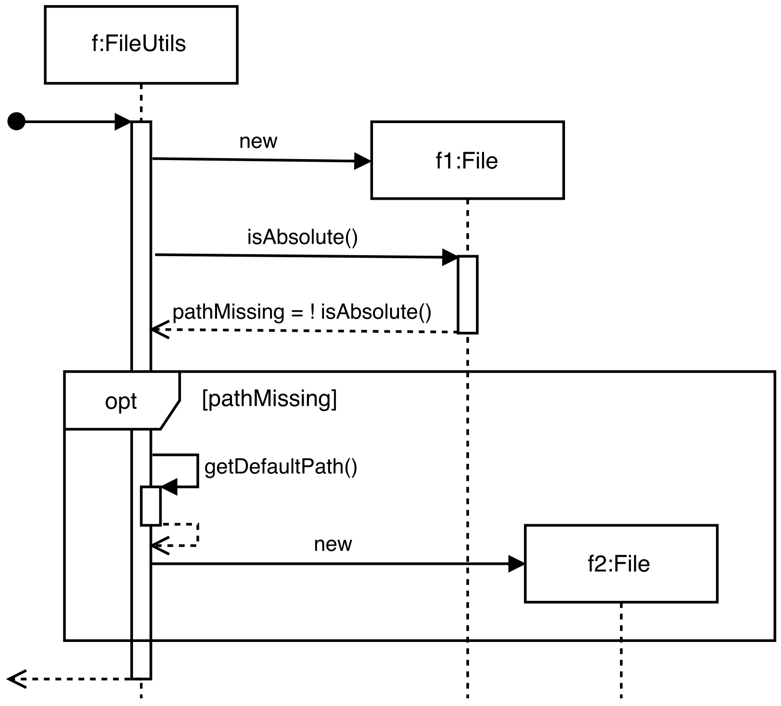
(c) *Bookwork, Application of knowledge*

(i). Modifications or adaptations of software released under a copyleft open-source licence such as GPL must also themselves be licensed under the same licence.

(ii). It depends on whether the library was distributed under the GPL or the Lesser GPL. The latter would allow release of the app without the source code, for former not.

Marking: 2 marks for each part.

3. (a) *Problem solving:* The sequence diagram should closely resemble this:



Issues to look for in the diagram include:

1 Basic diagram syntax captured (object rectangles with dashed lifelines descending below. Horizontal solid arrow messages and dashed arrow responses.)

2 Correct ordering of messages and messages going between correct objects.

3 Local variable values set on method returns.

4 Lifelines of constructed File objects start at constructor invocations.

5 Use of UML *opt* block with pathMissing condition for if statement.

6 File object constructed in opt block is distinct from file object constructed before.

7 Activation bar for getDefaultPath invocation half overlapping activation bar for getAbsoluteFile invocation

2 marks for each of first two. 1 mark for each of further 5 points.

(b) (i). *Bookwork:* Refactoring is particularly important in agile processes because otherwise the rapid incremental iterations can lead to hard to read and hard to maintain code.

(ii). *Bookwork:* Extract Method is good for making long methods shorter and pulling out reusable blocks of code from methods.

(iii). *Problem solving*

```
public class SignalProc{

  public void clip(Signal s, Signal lo, Signal hi) {
      if (s.compareTo(lo) == -1) {
        s.set(lo);
      } else if (s.compareTo(hi) == 1) {
        s.set(hi);
    }
  }
  public int[] clipWaveform(Signal[] wf, Signal lo, Signal hi) {
    for (i = 0; i != wf.length; i++) {
      clip(wf[i],lo,hi);
    }
  }
}
```

4 marks for something sensible.
1 further marks if the solution recognises that, because `Signal` is a reference type, the extracted method can just take the `s` argument as a single argument for both reading and writing.

(c) *Bookwork*

(i). The main ones highlighted in class were CVS and SVN. Distributed VCSes such as Git and Mercurial also do.

(ii).
A. Alice checks her modified file into the repository.
B. Bob executes an update operation on his version of the file in order to pull in Alice's changes.
C. the VC system reports that the merge of Alice's changes in the repository with Bob's changes failed, and usually produces a file showing the conflicting segments.
D. Bob edits the file to resolve the conflicts.
E. Bob checks in the file.

1 point for anything vaguely sensible and 1 point for each of the above 5 key steps.