NTNU
Norwegian University of Science and Technology

ENGLISH

Faculty of Physics, Informatics and Mathematics

Department of Computer and Information Sciences

Sensurfrist: 2010-06-29

Exam in
TDT 4242 Software Requirements and Testing

Tuesday June 8, 2010
9:00 am – 1:00 pm

Aids allowed C:
Pocket calculators allowed
Specified material allowed.
All printed and handwritten material is specified as allowed

Contact person during the exam:
Professor Tor Stålhane, phone 73594484

Good Luck!

Introduction

In this exam you can score a maximum of 70 points. The rest of the total possible score of 100 points for the semester comes from the compulsory exercises.

If you feel that any of the problems require information that you do not find in the text, then you should

- Document the necessary assumptions
- Explain why you need them

Your answers should be brief and to the point.

Problem 1 – Requirements engineering (20 points)

Toga Islands airspace is divided into four zones (centres), and each zone is divided into sectors. Also within each zone are portions of airspace, about 50 miles (80.5 km) in diameter, called TRACON (Terminal Radar Approach CONtrol) airspaces. Within each TRACON airspace there is a number of airports, each of which has its own airspace with a 5-mile (8-km) radius.

For jet powered airliners flying within Toga airspace, the cruising altitudes typically vary between approximately 25,000 feet and 40,000 feet. All other engines are only allowed to fly at a cruising altitude below 25,000 feet. Allowable cruising altitude is dependent on the speed of the aircraft.

Consider the goal model fragment of an air traffic control system for Toga Islands in the diagram in appendix 1. The parent goal roughly states that aircraft routes should be sufficiently separated. The refinement is intended to produce sub-goals for continually separating routes based on different kinds of conflicts that can rise among routes. The goal Achieve[RouteConflictsPredicted] roughly states that the air traffic control system should be able to predict route conflicts associated with aircraft flying within Toga airspace.

1a — 15 points

1. Identify the parent goal of the goal Maintain[SufficientRouteSeparation].

Maintain[SafeFlight] is considered to be the right answer. With logical reasons, marks should also be given for answers such as: Achive[RouteOptimisation] Maintain[RouteEfficiency]

2. Based on the description provided above, identify two refinement nodes from the sub-goal Achieve[RouteConflictsPredicted].

Ideally it is expected that students will use the parameters in the description above (zone, sector, airport_airspace, cruising_altitute, type_of_engine, speed_of_aircraft) to derive subgoal, concrete requirement, design constraint or an expectation. Example:

- ***Maintain[WorstCaseConflictDistance] (sub-goal)***
- ***Achieve[CompareSpeedofAircraft if same zone and same cruising_altitude] (concrete requirement)***
- ***Achieve[CompareEngineType if same sector and cruising_altitude] (concrete requirement)***

1b — 5 points

For each of the refinement nodes identified in (1a-2) explain whether this is a sub-goal, concrete requirement, design constraint or an expectation. Explain why you give each node their characteristic.

Reasons should align with the definition of goals, concrete requirement, design constraint or expectation:

- ***Subgoal:***
  - ***Is an objective the system under consideration should achieve***
  - ***Ranges from high-level strategic to low-level technical concerns over a system***
  - ***A goal that requires the cooperation of many agents.***
- ***Concrete Requirement:***
  - ***Normally atomic and cannot be decomposed into further sub-goals.***
  - ***A goal under the responsibility of a single agent in the software-to-be.***
- ***Assumption (expectation):***
  - ***A goal under the responsibility of a single agent in the environment of the software-to-be.***
  - ***Assumptions cannot be enforced by the software-to-be***
- ***Design constraint***
  - ***Refers to some limitation on the conditions under which a system is developed, or on the requirements of the system.***

Problem 2 – Testing methods (20 points)

Some developers in our company have just developed a large and important method – a switch based implementation of a state machine. We have access to all code and all documentation – among other things the use cases, the sequence diagrams and the state charts for all components. In addition, we have a tool that can automatically draw the code graph for the method. The final graph is annotated with the predicates of each branching point.

2a – Methods choice – 10 points

1. There are several ways to define the necessary tests – e.g. black box testing, white box testing, round-trip path tree and domain testing. Select two test methods that can be used for the situation described above and give a short discussion of their strong and weak sides. Select one of these two methods for the scenario described above. Explain why you choose this method.

Two useful methods are round-trip path tree and full path coverage.

Full path coverage consists of the following steps:

- ***Create a table with one row for each predicate value combination – e.g. FFF, FFT, FTF, FTT, TFF, TFT, TTF, and TTT for three predicates.***
- ***Select input data that realizes each predicate value combination.***
- ***Run all the tests and check the results***

Strong points: full path coverage, simple to implement, we have some too support for this
Weak points: may create a several redundant test cases

Round trip patch tree consist of the following steps:

- ***Generate the roundtrip path tree***

- ***Generate one test set for each legal set of predicates – each complete branch.***
- ***Generate one test set for each illegal set of predicates***
- ***Run the tests and check the results***

Strong points: can test both good data and error handling, can discover sneak paths
Weak points: manual creation of the roundtrip path tree

The only weak point for the roundtrip path tree is the manual creation of the tree. If the state diagram is small – e.g. less than 15 states – I would select the round trip path method, otherwise I would select the full path coverage method, since this has some tool support.

2. Why is 100% code coverage in most cases not good enough as a stop criterion for testing? Which coverage measure would you use and why?

Code coverage is not a sufficient coverage measure since it is possible to have full code coverage and still miss out a lot of paths. As a minimum I would choose full path coverage since this will guarantee that all paths through the code are executed at least once. Loops will need special treatment – e.g. all loops executed at least 0 times, 12 times and 20 (many) times.

2b – Testing methods – 10 points

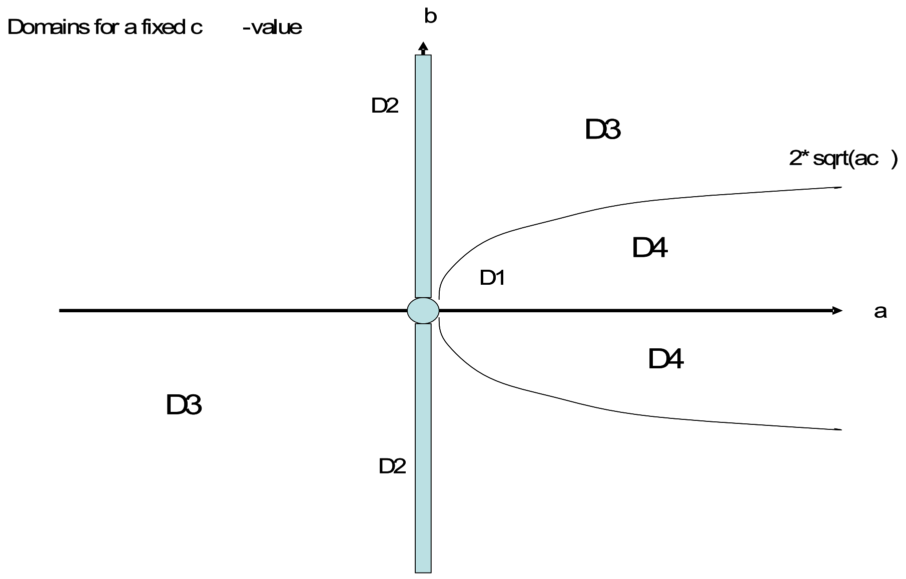
1. Explain how you would use domain testing in the enclosed code example – see appendix 3.

The program in appendix 3 has four input domains:

- ***D1: a = b = 0, which is a line in the three-dimensional space – F1***
- ***D2: a = 0 and b >< 0, which a plane in the three-dimensional space – F2***
- ***D3: b*b >= 4*ac – F3***
- ***D4: b*b < 4*ac – F4***

Note that with these domain definitions, a = 0, b = 0 formally belongs to both D1 and D3. The division b>/(2*a) requires, however, that a >< 0. Thus, it would be more appropriate to define:

- ***D3: b*b >= 4*ac and a >< 0.***
- ***D4: b*b < 4*ac and a >< 0.***



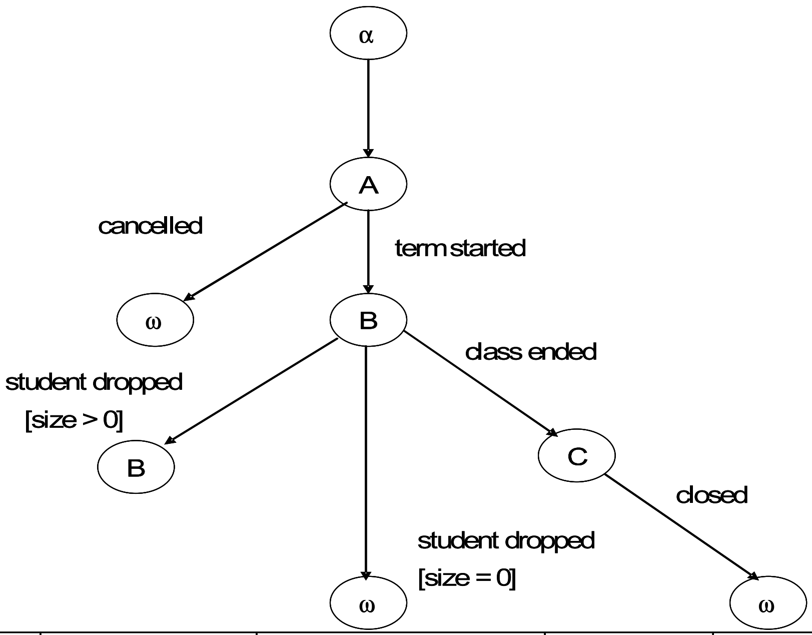
The standard domain testing assumption is that: data in Di => result from Fi.
The domain diagram can be drawn in a two-dimensional space for any fixed c-value.

- ***Tests for D1: ON-point a = b = 0 =>F1, OFF-points: a=0, b = 0.0001 => not F1, a = 0.0001, b = 0 => not F1***
- ***Tests for D2: ON-point: a = 0, b = 0.0001 => F2, OFF-points: a = b = 0 => not F2, a = 0.0001, b = 0 => not F2***
- ***Tests for D3: ON-point: a = 1, b = 2, c = 1 => F3, OFF-points: a = 1, b = 1, c = 1 => not F3, a = 4, b = 3, c = 1 => not F3***
- ***Tests for D4: ON-point: a = 1, b = 1.001, c = 1 => F4, OFF-points: a = 1, b = 2.1, c = 1 => not F4, a = -1, b = 1, c 0 1 => not F4.***

Alternatively, we could define the four domains based on the four predicates, named so that Pi corresponds to Di in the list above. This would give us D1: T---, D2: FT--, D3: FFTF and D4: FFFT.

2. Use the round trip path tree method to generate the testes needed for the state diagram shown in appendix 2.

See diagram below.



ID	*Start state*	*Event*	*Condition*	*Result*	*New state*
1	***α***	***Constructor***	***-***	***-***	***A***
2	***A***	***Cancelled***	***-***	***-***	***ω***
3	***A***	***Term started***	***-***	***-***	***B***
4	***B***	***Student dropped***	***Size = 0***	***-***	***ω***
5	***D***	***Student dropped***	***Size > 0***	***-***	***B***

Note that the table is just an example – it is not complete.

Problem 3 – Testing management (30 points)

Our company has a large amount of expertise on testing. Thus, testing other companies' products has become a major source of income for us. Our testers are especially good at using all types of assertions during testing.

A large company that specializes in development of software for industrial automation has put a large test job out for tender. The code will be developed in Java. The test job contains the following activities:

- Testing of all classes before integration. Unit testing of individual methods has already been done
- Integrating and testing the subsystems
- Testing the total system, consisting of all the subsystems integrated.
- Construct the first version and necessary infrastructure for a regression test.

In addition to convincing the other company that we can deliver according to contract, it is also important for us to define a believable set of completeness and success criteria for the tests.

We have access to all code and all documentation – among other things the use cases, the sequence diagrams and the state charts for all components.

Describe the following:

1. The way we will argue that the outsourcing company can have confidence in our results and recommendations – e.g. our recommendations on delivery / no delivery.

The argument of confidence is based on three pillars:

- ***Who we are: description of our testing experience – e.g. our wide experience with testing methods and tools***
- ***What we have done: a description of our track record – i.e. the testing we earlier have done for other companies.***
- ***How we will do the job: the methods and tools we will use for each job described in the list above plus a description of the test strategy and test plan.***

2. The test strategy that we will use for the
    a. Integration test
    b. Acceptance test.

Acceptance test:
The acceptance test for functional requirements will take the users' goals as its starting point. We will test each goal by identifying all functions that participate in the system's ability to meet this goal and then

- **Test each function**
- **Have the customer participate in assessing whether the combined functionality will enable him to meet his goals.**

Non-functional requirements will be tested by

- **Deriving measurable requirements from each non-functional requirement. This will be done by using the "What do you mean by…" approach as many times as necessary.**
- **Testing the measurable requirements**
- **Checking that each measurable requirement meets the customers requirements and expectations.**

Integration test:
We will use interaction assertions for integration testing – assertTrue, assertFalse, assertSame, assertEquals. These assertions will check that each component give the info that its environment expected due to the requirements at this level.

3. The acceptance criteria that we will use for both of the abovementioned tests.

Integration test: The integration acceptance criterion will be that

- **When a subsystem – part that is integrated – is evoked, all assertion will behave as expected. During the integration test, all subsystems will be evoked at least once.**

Acceptance test: two acceptance criteria:

- **100% requirements coverage – all requirements that are derived from the customer's goals are covered. This will include all error handling.**
- **The results of the acceptance test is accepted in full by both the developers and their customers**

Appendix 1 – Goal model fragment

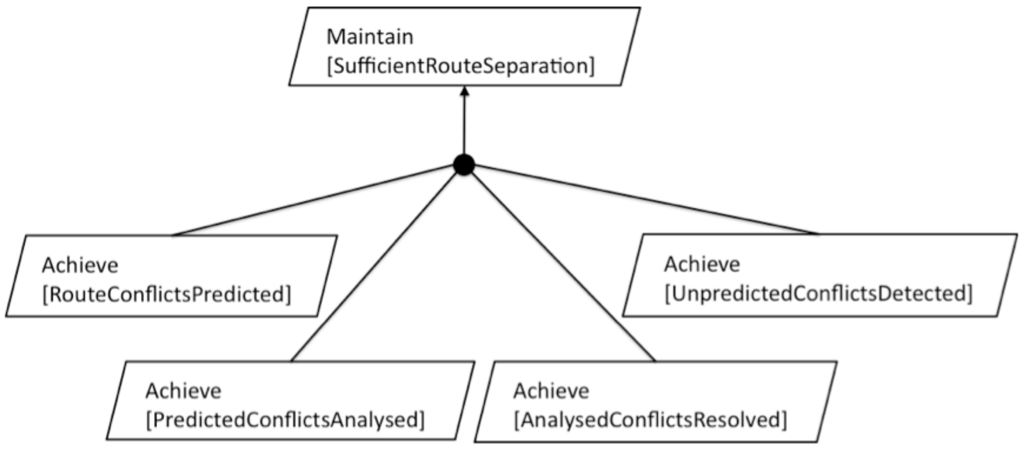
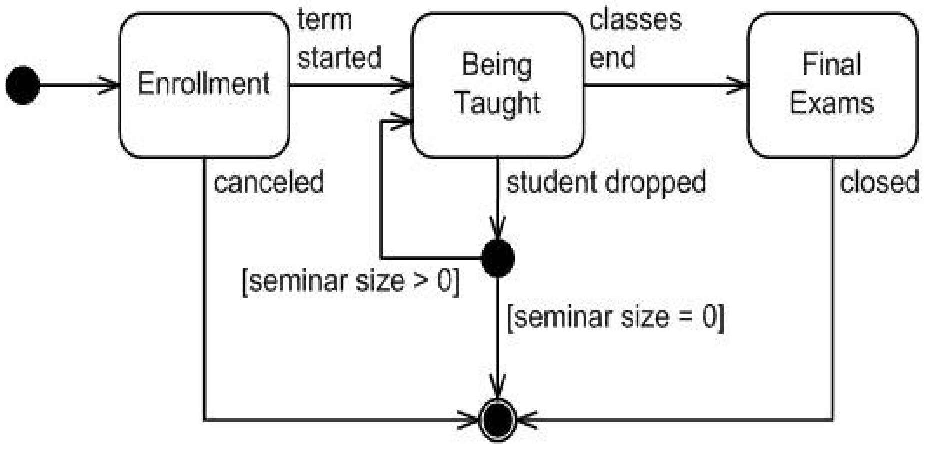

Maintain
[SufficientRouteSeparation]
Achieve
[RouteConflictsPredicted]
Achieve
[UnpredictedConflictsDetected]
Achieve
[PredictedConflictsAnalysed]
Achieve
[AnalysedConflictsResolved]

Figure 1 Goal Model fragment for Toga air traffic control system

Appendix 2 – State diagram



Appendix 3 – Code fragment

Pseudo code for solving a second degree equation $aX^2 + bX + c = 0$.

```
equation_solver(a, b, c, x1, x2)
{
 real a, b, c.x1, x2

if (a = 0 and b = 0) then
{
   write(" no solution")
  exit
}

if (a = 0 and b =/= 0) then
{
  x1 = c / b
  x2 = x1
  exit
}

rad = b*b - 4*a*c

if  (rad >= 0) then
{
  x1 = -b/(2*a) - sqrt(rad)
  x2 = -b/(2*a) + sqrt(rad)
  exit
}
else
{
   write ("no real roots")
  exit
}

}
```