SWEN90010 Sample exam questions

This is a collection of sample exam questions, not a sample exam itself. The final exam will be a two hour exam. The style of question will be similar to the questions in this handout.

Part A – Safety engineering

Question 1 [10 Marks]

Explain the purpose, the information derived from a typical analysis and the use of *hazard analysis* in the safety engineering process.

Question 2 [15 Marks]

Consider the following description of a nuclear power system, which controls the temperature of the nuclear core. The system consists of an array of sensors, three assessors, a voter, a controller, and an actuator. A system architecture is shown in Figure 1.

- There is an array of sensors monitoring the temperature of the core. Sensors are accurate to 0.1%, but can fail by not providing a reading, or by providing a reading outside of the 0.1% range.
- Three *assessors* each read the temperature signals sent from all sensors, and use a majority vote to determine the core temperature. Two sensor readings are considered sufficiently equal if they are within 0.4% of each other.

  If any assessor detects that less than half of the sensors are sufficiently equal to the others, it should send a shutdown signal to the voter. Otherwise, it sends its estimated temperature.
- The voter using a voting algorithm to detect any non-functioning assessor. If the voter detects that more than one assessor has failed, or it receives a shutdown signal from any assessor, it should send a shutdown signal to the controller. Otherwise, it sends its estimated temperature to the controller.
- The controller controls the actuator by lowering or raising cooling rods into the core based on the estimated temperature sent from the voter. If it receives a shutdown signal, it will lower the rods completely and raise an alert.
- The actuator controls the rods. The level of the rods should be in correlation with the core temperature; that is, if the temperature is high, the rods should be lowered further into the core than if the temperature is low.

Perform a Hazard and Operability Study (HAZOP) on the assessors of this system, paying attention to any safety concerns. You will need to tabulate consequences, potential causes, and risks.

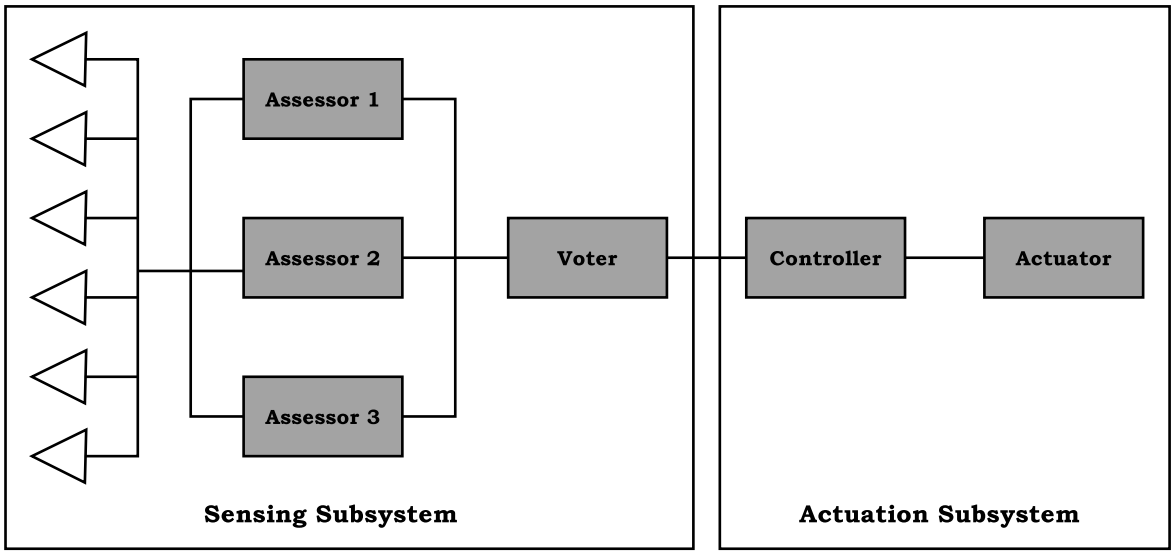

Figure 1: A system architecture for a nuclear reactor.

On an exam, any question about HAZOP would include the HAZOP guidewords.

Question 3 **[15 Marks]**

Choose one hazard from either of the previous two questions. Use fault-tree analysis to analyse causes of this hazard.

On an exam, any question on fault tree analysis will include the symbols used for fault trees.

Part B – Model-based specification

Consider a simple scheduling module for a single processor. A set of processes are running on the system, but the processor can execute only one at each time. Each process must be in exactly one of the following states:

1. *active*: currently executing on the processor;
2. *ready*: not executing, but ready to be executed; or
3. *waiting*: waiting on some other resource, so not ready to be executed.

At any point, there must be *at most* one active process. There should be no ready processes if there is no active process — that is, the processor must be in use as much as possible.

Question 4 [7 Marks]

Model the state and state invariant of the single processor scheduler using an Alloy signature and predicate respectively. The signature should model all active, ready, and waiting processors, and the invariant predicate should model the constraints between them; e.g. only one active process at at time.

Your solution should assume the existence a signature `ProcessID`, which is the set of all process IDs, declared as follows:

```
sig ProcessID {}
```

Question 5 [5 Marks]

New processes can be added into the scheduling system. A new process must not be an existing process in the system. When a new process is added into the system, by default it is in the *waiting* state.

Model an operation called `NewProcess` as an Alloy predicate, which takes, as input, a process ID (`ProcessID`) and adds the process to the scheduling system.

Question 6 [7 Marks]

Model an operation called `Ready` as an Alloy predicate, which takes, as input, a process ID, and switches this process out of the waiting state. The specified process must be a waiting process. If there is no active process, the specified process becomes the new active process. Otherwise, it becomes a *ready* process.

Question 7 [7 Marks]

Model an operation called `Swap` as an Alloy predicate, which models the case of an active process switching into a waiting process; that is, the process has finished executing for now. In addition to swapping out the active process, the scheduler should select *any* of the ready processes to become the new active process. If there are no ready processes, then there is no new active process. This operation should take no input.

Question 8 [12 Marks]

Note: this question requires you to understand a model in detail and to use Alloy to play around with the model, which is not representative of a good exam question. However, the style of question is representative of an exam question, and completing this question is good revision.

Consider the Alloy model at the end of this document, which models a key system for a hotel. The source is available on the LMS under the *Workshops* menu.

1. Read the model. Write one sentence to show why `NoBadEntry` seems as if it ought to be true.
2. Execute `check NoBadEntry`. Write one sentence explaining what has gone wrong.

3. Write a new fact that fixes the model. Execute `check NoBadEntry` again to make sure you have fixed the problem.

Part C – Short-answer cryptography and security questions

Question 9 [12 Marks]

(a) **[2 marks]** A message authentication code (MAC) takes a message m and a key k, and outputs a digest $MAC(m, k)$.

   (a) What is the main security property required of a MAC?

   (b) Suppose we have a hash function that is collision-resistant for single blocks, extended to long messages using the Merkle-Damgård construction (See Paar and Pelzl's textbook, Section 11.3, Figure 11.5[1]). Let $||$ indicate message concatenation, and h the hash function for multiple blocks. We construct a MAC as $MAC(m, k) = h(k||m)$. Is this secure? Why or why not?

(b) **[2 marks]** When computing a digital signature (*e.g.* using RSA), we usually hash the message before signing. If the hash function is not *collision-resistant*, how can an attacker break the security of the signature scheme?

(c) **[2 marks]** List two important things on an SSL/TLS certificate and two important checks your web browser should perform.

(d) **[2 marks]** Draw a picture of one of Google's certificate transparency logs. Explain how you can prove, to someone who knows the root hash, that your certificate is truly one of the leaves. If there are n certificates in the tree, use big-O notation to describe the size of the proof.

(e) **[2 marks]** Explain TLS proxying and the implications for the Western Australian election.

(f) **[2 marks]** An RSA public key consists of two items: an exponent e and a modulus N.

   (a) If you discover another person with the same e as your key, can you decrypt their messages? Explain why or why not.

   (b) If you discover another person with the same N as your key, can you decrypt their messages? Explain why or why not.

Question 10 [5 Marks]

(a) Consider a program for solving (unauthenticated) Byzantine Agremeent using the Exponential Information Gathering algorithm. Write the main parts of the algorithm out in your favourite programming language. (Assume you have scaffolding and library functions as necessary, *e.g.* for sending and receiving messages to and from other participants.)

(b) In order to tolerate one traitor, identify the minimum number of rounds your algorithm needs to run, and the minimum number of total participants.

[1] This is a sample exam, so you have to look it up yourself.

(c) Find a counterexample showing that agreement might fail if there is one less participant than necessary.

(d) Find a(nother) counterexample showing that agreeement might fail if there is one less round than necessary.

Part D – Fault-tolerant design

Question 11 [7 Marks]

What is design diversity and how does it contribute to software redundancy?

Question 12 [5 Marks]

Do recovery blocks require design diversity? Justify your answer.

Question 13 [7 Marks]

Systems that must detect up to n non-malicious failures require $2n+1$ redundant components. How many components do systems with malicious (or Byzantine) failures require to detect n failures? Why is this the case?

Part E – Correctness by construction

Question 14 [7 Marks]

You are working in a company that specialises in high-integrity software. SPARK is the implementation language of choice. A new manager arrives at the company and shows a preference for switching to Ada instead of using the SPARK safe subset. The manager's preference is based on a study that showed that more lines of code are required in safe programming language subsets than in the superset language to write the same program. The manager asserts that this increases the cost of the project.

Do you agree or disagree with the manager's assertion?

Question 15 [3 Marks]

Why did the designers of SPARK prohibit dynamic memory allocation?

Question 16 [12 Marks]

NOTE: Many of the following questions require you do to Hoare logic proofs over Ada programs. On a real exam, however, Hoare logic questions would be written in the simple programming language explained in lectures on Hoare logic.

```
procedure PowerOfTwo(N : in Integer; Result : out Integer) with
  Post => (Result = 2**N);
is
  K : Integer;
begin
  K := 0;
  Result := 1;
  while K /= N loop
    pragma Loop_Invariant (Result = 2**K);
    K := K + 1;
    Result := 2 * Result;
  end loop;
end PowerOfTwo;
```

Figure 2: An Ada program for calculating a power of two.

Figure 2 shows an Ada program for calculating the power of two of an integer. The postcondition is that `Result = 2**N`, in which `2**N` is defined as:

```
2**0      =  1
2**(N+1)  =  2*(2**N)
```

The loop invariant has been provided. Using Hoare logic, show that this program either establishes its contract, or fails to establish its contract.

Question 17 [12 Marks]

Use Hoare logic to establish whether the following program establishes is contract or not.

The loop invariant `Min(A, J, MinIndex)` is defined as follows:

```
Min(A, J, MinIndex)  =  for all I in Index range 1 ..  J => (A(I) >= A(MinIndex))
```

This states that the element at `MinIndex` is the smallest element found so far.

```
subtype Index is Integer range 1 .. 10;
type IntArray is array(Index) of Integer;

procedure FindMinIndex(A : in IntArray; MinIndex : out Index) with
   Post => (for all I in Index range 1 .. A'Length => (A(I) >= A(MinIndex)));
is
   J : Index;
begin
   -- find the minimum element in the list
   J := 1;
   MinIndex := 1;
   while J < A'Length loop
     pragma Loop_Invariant (Min(A, J, MinIndex));
     J := J + 1;
```

```
      if A(J) < A(MinIndex) then
        MinIndex := J;
      end if;
    end loop;
  end FindMinIndex;
```

Question 18 [6 Marks]

Figure 3 shows an Ada program for calculating the average of an array of integers. The program includes a contract with the postcondition:

```
(if A'Length = 0 then Result = 0 else
Result = summation(1, A'Length) / A'Length);
```

where `summation(X,Y)` = $\sum_{i=X}^{Y}$ `A`(i).

The program loops through all elements in the array, keeping a running tally in the variable `Sum`.

(4 marks) Write down a suitable loop invariant for this program, which is suitable to establish the postcondition required at the end of the loop:

```
Result = summation(1, A'Length)
```

(2 marks) Use Hoare logic to show that if this postcondition holds at the end of the loop, then the remainder of the program will establish the postcondition in its contract above.

Question 19 [20 Marks]

Figure 4 contains a SPARK implementation of a binary search algorithm. The program assumes that the list is sorted, and also that the target is in the list. The postcondition is that the variable `Index` is the index of the variable `Target` in the array `A`. The predicate `Sorted(A)` is defined as:

```
Sorted(A) = for all I in Index range 1 .. A'Length - 1 => (A(I) <= A(I+1))
```

The loop invariant has been provided. It states that the precondition always holds (which is true because the list `A` never changes) and that the target always remains in the range `[Low..Result]`.

Use Hoare logic to establish whether the following program establishes is contract or not.

```
type IntArray is array(<>) of Integer;

procedure ListAverage(A : in IntArray; Result : out Float) with
  Post => (if A'Length = 0 then Result = 0 else
           else  Result = summation(1, A'Length) / A'Length);
is
  Sum, I : Integer;
begin
  I := 0;
  Sum := 0;
  while I /= A'Length loop
    I := I + 1;
    Sum := Sum + A(I);
  end loop;

  if A'Length = 0 then
    Result := 0;
  else
    Result := Sum / A'Length;
  end if;
end ListAverage;
```

where `summation(X,Y)` = $\sum_{i=X}^{Y}$ `A(`i`)`.

Figure 3: An Ada program for calculating the average from an array of integers.

```
procedure BinarySearch(A : in IntArray;
                       Target : in Integer;
                       Result : out Index) with
  Pre => Sorted(A) and
           (for some I in Index range 1 .. A'Length => (A(I) = Target)),
  Post => A(Result) = Target;
is
  Low : Index := Index'First;
  Mid : Index;
begin
  Result := Index'Last;
  while (Low < Result) loop
    pragma Loop_Lnvariant (Sorted(A) and A(Low) <= Target and Target <= A(Result));
    Mid := (Low + Result) / 2;
    if A(Mid) < Target then
      Low := Mid + 1;
    else
      Result := Mid;
     end if;
  end loop;
end BinarySearch;
```

Figure 4: An Ada program implementing the binary search algorithm

```
module chapter6/hotel1 --- the model up to the top of page 191

open util/ordering[Time] as to
open util/ordering[Key] as ko

sig Key {}
sig Time {}

sig Room {
  keys: set Key,
  currentKey: keys one -> Time
  }

fact DisjointKeySets {
  -- each key belongs to at most one room
  Room<:keys   in   Room lone-> Key
  }

one sig FrontDesk {
  lastKey: (Room -> lone Key) -> Time,
  occupant: (Room -> Guest) -> Time
  }

sig Guest {
  keys: Key -> Time
  }

fun nextKey [k: Key, ks: set Key]: set Key {
  min [k.nexts & ks]
  }

pred init [t: Time] {
  no Guest.keys.t
  no FrontDesk.occupant.t
  all r: Room | FrontDesk.lastKey.t [r] = r.currentKey.t
  }

pred entry [t, t': Time, g: Guest, r: Room, k: Key] {
  k in g.keys.t
  let ck = r.currentKey |
    (k = ck.t and ck.t' = ck.t) or
    (k = nextKey[ck.t, r.keys] and ck.t' = k)
  noRoomChangeExcept [t, t', r]
  noGuestChangeExcept [t, t', none]
  noFrontDeskChange [t, t']
  }

pred noFrontDeskChange [t, t': Time] {
  FrontDesk.lastKey.t = FrontDesk.lastKey.t'
  FrontDesk.occupant.t = FrontDesk.occupant.t'
  }

pred noRoomChangeExcept [t, t': Time, rs: set Room] {
  all r: Room - rs | r.currentKey.t = r.currentKey.t'
  }
```

```
pred noGuestChangeExcept [t, t': Time, gs: set Guest] {
  all g: Guest - gs | g.keys.t = g.keys.t'
  }

pred checkout [t, t': Time, g: Guest] {
  let occ = FrontDesk.occupant {
    some occ.t.g
    occ.t' = occ.t - Room ->g
    }
  FrontDesk.lastKey.t = FrontDesk.lastKey.t'
  noRoomChangeExcept [t, t', none]
  noGuestChangeExcept [t, t', none]
  }

pred checkin [t, t': Time, g: Guest, r: Room, k: Key] {
  g.keys.t' = g.keys.t + k
  let occ = FrontDesk.occupant {
    no occ.t [r]
    occ.t' = occ.t + r -> g
    }
  let lk = FrontDesk.lastKey {
    lk.t' = lk.t ++ r -> k
    k = nextKey [lk.t [r], r.keys]
    }
  noRoomChangeExcept [t, t', none]
  noGuestChangeExcept [t, t', g]
  }

fact traces {
  init [first]
  all t: Time-last | let t' = t.next |
    some g: Guest, r: Room, k: Key |
      entry [t, t', g, r, k]
      or checkin [t, t', g, r, k]
      or checkout [t, t', g]
  }

assert NoBadEntry {
  all t: Time, r: Room, g: Guest, k: Key |
    let t' = t.next, o = FrontDesk.occupant.t[r] |
      entry [t, t', g, r, k] and some o => g in o
  }

// This generates a counterexample similar to Fig 6.6
check NoBadEntry for 3 but 2 Room, 2 Guest, 5 Time
```