# Software testing

IN3240/IN3240

Part 2

Oppgavene er gitt i random rekkefølge

## Equivalence Prtitioning / Boundary Value Analasis (10 poeng)

### a) Løsningsforslag:

Gyldige partisjoner:

- [0, 217399]
- [217400, 306049]
- [306 050, 697149]
- [697150, 937900]
- [937901, 1350000]
- [1350001, → >

Ugyldig partisjon:

- <←, -1]

**Kommentar**: Ekvivalensklassene kan gjerne være beskrevet med ord, eller være angitt på andre måter. Hvis feilen i spesifikasjonen er oversett, skal det ikke trekke ned.

### b) Løsningsforslag:

Hvis vil kaller mengde av input-verdier (heltall) for A, kan betingelsene forklares på følgende måte:

1. Alle delmengdene må til sammen utgjøre hele mengden de har utgått fra, dvs. unionen av dem må være lik hele mengden $A$:

$$A = A_1 \cup A_2 \cup A_3 \cup \ldots \cup A_n$$

2. Delmengdene må være disjunkte, dvs. de kan ikke ha noen felles elementer. Det betyr at snitt mellom de ulike delmengdene må være tomme:

$$A_i \cap A_j = \emptyset \; \forall \; i \neq j$$

hvor $\emptyset$ er den tomme mengden og allkvantoren $\forall$ betyr *for alle*.

**Kommentar:** *Det holder at studentene kan enten forklare betingelsene med ord, eller med matematiske symboler.*

1. Vi ser at unionen av ekvivalensklassene i a) til sammen utgjør mengden av alle hele tall.
2. Vi ser av ingen av heltallene tilhører mer enn én ekvivalensklasse, dvs. ekvivalensklassene er disjunkte.

**c) Løsningsforslag:**

Programmet må behandle alle elementene innen hver enkelt ekvivalensklasse likt.

**d) Løsningsforslag:**

Det må velges ett element fra hver ekvivalensklasse, f.eks.

-1, 100 000, 250 000, 500 000, 750 000, 1 000 000, 5 000 000

NB! Det er ikke spørsmål om grenseverdier her.

**e) Løsningsforslag:**

Mengden av inputverdier må være sorterbare (slik at man kan avgjøre hvilke som ligger nærmest grensen mellom to ekvivalenspartisjoner).

**f) Løsningsforslag:**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 217 399 | 217 400 | 306 049 | 306 050 | 697 149 | 697 150 | 937 900 | 937 901 | 1 350 000 | 1,350,001 |

**g) Løsningsforslag:**

Erfaring har vist at feil har en tendens til å oppstå på grensen mellom ekvivalensklassene. En årsak kan være at spesifikasjoner ofte ikke er presist nok formulert.

**h) Løsningsforslag:**

Hvis man tester alle grenseverdiene, har man også dekket alle ekvivalensklassene. Grenseverdier er imidlertid ekstremverdier, og **vi ønsker også å teste verdier som vil bli brukt under normal bruk av programmet.**

## State transition testing (10 poeng)

**a) Løsningsforslag:**

Antall tilstander: 12

Antall overganger: 15

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 12

Tilstandsdekning: $\frac{Antall\ tilstander\ besøkt}{Totalt\ antall\ tilstander} = \frac{11}{12}$

Overgangsdekning: $\frac{Antall\ overganger\ passert\ (minst\ en\ gang)}{Totalt\ antall\ overganger} = \frac{10}{15}$

**Kommentar:** *Trengs ikke angis som prosent. Det holder å skrive det som en brøk.*

**b) Løsningsforslag:** Nei, Det er ikke mulig å gå gjennom alle overganger i et enkelt testtilfelle, ettersom overgangene mellom tilstandene forgrener seg ved flere steder uten at det går noen overganger tilbake. Vi trenger derfor **minst 2 testtilfeller** for å dekke alle overgangene, f. eks:

1 - 2 - 3 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 – 12 og 1 - 2 - 3 - 4 - 6 - 7 - 9 - 11 – 12

**c) Løsningsforslag:**

Den høyeste overgangsdekningen vi kan oppnå med kun ett testtilfellet **11/15**, f.eks.

1 - 2 - 3 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 – 12

**d) Løsningsforslag:**

Selv om vi har 100% tilstandsdekning, er vi ikke sikret 100% overgangsdekning. Hvis vi derimot har 100% overgangsdekning er vi automatisk sikret 100% tilstandsdekning fordi vi nødvendigvis må ha passert alle tilstandene hvis vi har passert alle overgangene.

## Decision table (10 poeng)

**Kommentar:** *Studentene har til en viss grad anledning til å selv tolke oppgaveteksten. For tilfellet der programmet skal sende informasjon om bankens spareprogram, blir søknaden om lån ikke innvilget. Dette kan løses på flere måter.*

**a) Løsningsforslag:**

Betingelser:

- **FJ**: Fast jobb
- **E:** Minimum 15 % egenkapital (15 % verdi av eiendom)
- **ML:** Ikke mer enn maksimalt lånebeløp 5 ganger inntekten.

Handlinger:

- **I**: Lånesøknaden innvilges
- **IR**: Lånesøknaden innvilges, men med redusert lånebeløp
- **SB**: Søkeren tildeles en saksbehandler.
- **AS**: (Lånesøknaden er avslått), men søkeren blir tilsendt informasjon om bankens spareprogram.
- **A**: Lånesøknaden er avslått **uten** informasjon om bankens spareprogram.

| | Rule 1 | Rule 2 | Rule 3 | Rule 4 | Rule 5 | Rule 6 | Rule 7 | Rule 8 |
|---|---|---|---|---|---|---|---|---|
| **FJ** | true | true | true | true | false | false | false | false |
| **E** | true | true | false | false | true | true | false | false |
| **ML** | true | false | true | false | true | false | true | false |
| **Action** | **I** | **IR** | **AS** | **A** | **SB** | **A** | **A** | **A** |

**b) Løsningsforslag:**

**Kommentar:** *Reduksjonen kan gjøres på flere måter. Her vises kun en av dem.*

Vi ser at regel 7 og 8 gir samme utfall, uavhengig av om lånets størrelse er innenfor lånerammen på 5 ganger inntekten eller ikke. Disse to reglene kan derfor slås sammen.

| | Rule 1 | Rule 2 | Rule 3 | Rule 4 | Rule 5 | Rule 6 | Rule 7/8 |
|---|---|---|---|---|---|---|---|
| **FJ** | true | true | true | true | false | false | false |
| **E** | true | true | false | false | true | true | false |
| **ML** | true | false | true | false | true | false | --- |
| **Action** | **I** | **IR** | **AS** | **A** | **SB** | **A** | **A** |
| | | | | | | | |

(Alternativt kan 4 og 7 eller 6 og 8 slås sammen)

**c) Løsningsforslag:**

Det første testtilfellet tester R1, det andre R5, og det tredje R2. Vi trenger følgelig fire testtilfeller til for å dekke alle reglene:

4. Et testtilfelle hvor søkeren hvor søkeren ikke har egenkapital, men tilfredsstiller de to andre betingelsene, og får tilsendt informasjon om bankens spareprogram (i tillegg til at søknaden blir avslått. (R3)
5. Et testtilfelle der søkeren har fast jobb, men ikke oppfyller de to andre betingelsene slik at søknaden blir avslått (R4).
6. Et testtilfelle hvor søkeren kun har egenkapital og ikke tilfredsstiller de to andre betingelsene (R6), slik at søknaden blir avslått (R6).
7. Et testtilfelle hvor søkeren verken har jobb eller egenkapital slik at søknaden blir avslått (R7/8).

**Kommentar**: *Oppgaven er feilaktig merket som oppgave c). Skulle vært oppgave d).*

**d) Løsningsforslag:**

Ettersom antall betingelser øker, øker også kompleksiteten ved å bruke beslutningstabeller. Antall mulige kombinasjoner av sannhetsverdier er $\mathbf{2^x}$, der *x* står for antall betingelser. Når vi legger til betingelsen «under 30 år», vil får vi $\mathbf{2^4 = 16}$ regler i beslutningstabellen (før en eventuell reduksjon er foretatt).

(Beslutningstabeller som testdesign-teknikk blir dermed mindre anvendbar hvis antall betingelser er høyt.)

## Text assignment

Ett av de syv testprinsippene kalles

*Feilslutning vedrørende fravær av feil* (Absence-of-errors fallacy).

Hva forteller dette prinsippet oss, og hvordan bør det påvirke testaktivitetene?

**Løsningsforslag:**

**Kommentar**: *Studentene trenger ikke å skrive like nye som her.*

Det er nærliggende å tro at et program som er testet og funnet i overensstemmelse med de krav og spesifikasjoner som er lagt til grunn, er en suksess. Men er dette nok? Nei, ikke nødvendigvis. Det er til syvende og sist sluttbrukerne som avgjør om programmet er vellykket. Om programmet ikke blir tatt i bruk, kan vi neppe betegne det som en suksess. Det er mange produkter som av ulike årsaker aldri ser dagens lys. Det kan være produkter som brukerne ikke etterspør lenger, som er for vanskelige å bruke, som tiden har løpt fra, eller produkter som har priset seg ut av markedet. Vi har sett eksempler på dataprogrammer som det har tatt så lang tid å utvikle at når de endelig lanseres, har nye teknologiske løsninger og plattformer overtatt markedet.

Et feilfritt program er ikke nok. Selv om et program er aldri så grundig testet og alle avdekkede feil er rettet, vil det aldri bli en suksess med mindre det tas i bruk etter hensikten og oppfyller brukerens behov og forventninger. Den avgjørende testen gjøres av den eller de som skal bruke programmet. Ved å ha god kontakt med markedet, sluttbrukeren og miljøet programmet skal brukes i, gjennom hele utviklingsforløpet, kan vi bidra til at programmet blir en suksess.

## Text assignment

Hvilke andre testprinsipper har vi, ifølge ISTQB?

**Løsningsforslag:**

**Kommentar:** *Trenger ikke ta med prinsippet som ble forklart i forrige oppgave. Rekkefølgen spiller ingen rolle, og ordlyden kan variere, da den avhengig av hvilken lærebok de har lest. For å få poeng, må det imidlertid fremgå hvilket prinsipp studenten sikter til. Det holder kun å nevne prinsippene, de trengs ikke forklares nærmere.*

Norsk:

Prinsipp 1. Testing kan ikke bevise at koden er feilfri

Prinsipp 2. Fullstendig testing er umulig

Prinsipp 3. Tidlig testing sparer tid og penger

Prinsipp 4. Feil samler seg i klynger

Prinsipp 5. Pass på sprøytemiddel-paradokset

Prinsipp 6. Testing er avhengig av konteksten

(Prinsipp 7. Feilslutning vedrørende fravær av feil)

Engelsk:

1. Testing reveals defects, not the absence of them

2. Exhaustive testing is impossible

3. Test early saves time and money

4. Defects cluster together

5. Beware the pesticide paradox

6. Testing is context dependent

(7. Absence-of-error fallacy)

## Text assignment

**a)** Forklar hva et statisk analyseverktøy er og hensikten med verktøyet.

**b)** Statisk analyseverktøy brukes hovedsakelig av én gruppe personer. Hvilken?

**c)** Gi minst tre eksempler på typer informasjon som statiske analyseverktøy kan gi.

**Løsningsforslag:**

**Kommentar:** *Svaret på a) trengs ikke å være like omfattende som løsningsforslaget.*

a) Statisk analyse analyserer kildekoden med hensyn til ulike kvalitetsaspekter ved koden **ute å kjøre den**. Hvis dette gjøres ved hjelp av et program, vil kildekoden være input til programmet, mens output vi gi en **kvalitetsvurdering** av koden ved hjelp av ulike parametere. Hensikten er **å sikre at kodebasen holder et akseptabelt nivå med hensyn til ulike kvalitetsaspekter** ved koden for derved å **minske risikoen for feil**. Statiske testteknikker oppdager feil, mangler og kvalitetsavvik som er vanskelige å oppdage under dynamisk testing, og bør derfor brukes i tillegg til dynamiske testteknikker.

b) Statisk analyse kan brukes hovedsakelig av **utviklere/programmerere** før og under implementeringen av kildekoden

c)

- Brudd på navnekonvensjoner og kodestandarder,
- Avdekke områder med høy kompleksitet
- Avhengigheter, inkonsistens eller andre mistenkeligheter ved koden

## Text assignment

Uavhengig testing, hvor testing utføres av personer som ikke har vært involvert i utviklingen av det som skal testes, har sine fordeler, men også sine ulemper.

**a)** Nevn minst to fordeler med uavhengig testing.

**b)** Nevn minst én ulempe ved uavhengig testing.

**c)** Gi eksempler på minst fire ulike grader av testuavhengighet.

**d)** Ranger de fire testnivåene i forhold til hvor utbredt uavhengig testing er, fra minst utbredt til mest utbredt.

**Løsningsforslag:**

a**)**

- **Kommentar:** (*trenger bare å nevne to ting*)
  - Det er **vanskelig å se feil i eget arbeid**, da vi lett kan overse feil fordi vi vet hva som skulle ha stått der.
  - En **uavhengig tester er objektiv**, og har ingen personlig interesse av utfallet av testen.
  - En uavhengig tester kan se programmet fra **et annet perspektiv**, og kan derved se **feil og mangler som utviklerne ikke ser**. Kan verifisere antagelser som er gjort programutviklingen.

b)

- En uavhengig tester er **isolert fra utviklingsteamet** og kan derved mangle kunnskaper og informasjon om systemet.
- Uavhengig testing av oppleves av negative **flaskehalser**
- **Utviklere kan miste ansvarsfølelsen for det som utvikles** fordi de vet at det skal testes senere.

c)

1. **Ingen uavhengighet**. F.eks. utviklere tester sin egen kode.

2. Arbeidet, f.eks. koden, **testes av en annen innen samme team**

3. Arbeidet, f.eks. programmet testes av et eget **testere fra en egen testavdeling innen samme organisasjon/firma.**

4. Arbeidet, f.eks. programmet testes av testere fra en uavhengig organisasjon eller av noen **utenfor organisasjonen/firmaet**.

d) Unit-testing →Integrasjonstesting → System-testing → Akseptansetesting

## Text assignment

**a)** Forklar hva *utforskende testing* (exploratory testing) er.

**b)** Hva kan begrense nytteeffekten av utforskende testing?

**c)** Nevn minst to situasjoner der det kan være hensiktsmessig å velge utforskende testing som teststrategi.

**Løsningsforslag:**

a) Utforskende testing er form for erfaringsbasert testing, og går ut på å utforske programmet og dets virkemåte i den hensikt å avdekke flest mulige feil. Teknikken baserer seg på først og fremst på testerens egen kompetanse, erfaringer og kunnskaper.

b) Begrensingen er nettopp at den baserer seg på testerens egen kompetanse, og hvis denne er mangelfull, vil nytteverdien av testingen bli tilsvarende redusert.

c) Utforskende testing brukes i situasjoner der annen mer systematisk testing ikke er et reelt alternativ, enten det skyldes
- Tidsnød
- Manglende eller mangelfulle spesifikasjoner av programmet.
- Brukes også som et supplement til mer formell testing, gjerne som en siste test etter at den formelle testingen er ferdig.

## Essay

**Statisk testing**

Statisk testing spiller en viktig rolle innen programvareutvikling. Vi deler den inn i to hovedtyper, *Reviews* (granskninger) og *Statisk analyse*.

Skriv en sammenhengende tekst om statisk testing, der du besvarer så mange som mulig av spørsmål nedenfor. Du står imidlertid fritt til å skrive mer om emnet enn de momentene som er fremhevet i spørsmålene.

- Hva er *statisk testing*?
- Hva er formålet og fordelene med statisk testing?
- Hvor i utviklingsprosessen utfører vi statisk testing?
- Hvilke(t) testprinsipp(er) oppfordrer indirekte til gjennomføring av statisk testing?
- Hva skiller statisk testing fra dynamisk testing?
- Kan statisk testing erstatte dynamisk testing?
- Hvilke review-typer har vi, og hva kjennetegner de ulike typene? Bruk gjerne eksempler.
- Hvilke roller har deltagerne i de ulike reviews-typene, og hva er rollenes ansvarsområder?
- Hvilke fem faser inngår i et formelt review?
- ...

**Løsningsforslag, eksempel på en god besvarelse.**

**What is Static Testing?**

Static testing is a form of testing where we test the test basis, such as requirements, documentation, code, and other work products, without executing code or seeing the system in action. This could include reviewing the user requirements and user stories, to ensure that they are coherent, clear of ambiguities, inconsistencies, contradictions, and so on. You could analyze the code without running it to find unreachable code, unused variables, unconventional variable names and syntax, adherence to industry standards or best practices, and so on.

**The Benefits of Static Testing**

The purpose of static testing is to validate and verify the various test items in the test basis, by uncovering defects early and throughout the development process. It allows us to already begin testing as soon as we have established some user requirements, before any code is even written. The benefit is that we can uncover defects that are simple and less costly to fix now, than if they were uncovered later in the development life cycle. Additionally, it uncovers different defects from other testing methods such as dynamic testing, which we take a closer look at in a bit. Also, static testing is highly adaptable to different levels of formality, scope and size of a project, and can be in some way useful in most projects.

**When do we Perform Static Testing?**

Static testing is primarily associated with the activities of the left side of the *V-model*. Meaning, we can perform static testing on user requirements, system requirements, global– and local design. This means that static testing can be performed early and throughout the development process, and can be iteratively worked on to control the process as it goes on. More generally, static testing can be done at any point there exists a test basis to be tested.

**Which Test Principle(s) Indirectly Encourage the Implementation of Static Testing?**

Static testing is closely tied to the testing principle of early testing (**P3**), as already outlined. Moreover, it is also tied to the principle that defects cluster together (**P4**). By uncovering defects early through static testing, we can control and update our plans to better prioritize areas where we have found defects, by allocating more time or resources to testing these areas. This prioritization is also indirectly encouraged by the principle that exhaustive testing is impossible (**P2**). The first principle of testing revealing defects, not the absence of them (**P1**) also encourages static testing, as it encourages testers to revisit existing test items since it might uncover defects we were not aware of the first time. The use of static testing is also encouraged by the principle of the pesticide paradox (**P5**), as varying your test methods by combining static and dynamic testing, or simply static reviews and static analysis, helps in finding defects that the other method is unlikely to find. As testing is context dependent (**P6**), it encourages static testing by adjusting the test methods and scope to the project, as static testing can help in refining the scope and requirements of a project (*It should be noted though, that this principle*

*might on the other in certain cases hand limit the use of static testing to match the context of the project*). Finally, it is also encouraged by the absence-of-error fallacy (**P7**) in that static testing plays a crucial role in *validation*, which is key in ensuring that we build the right thing rather than just building the wrong thing according to specification. All in all, every principle can in some way indirectly encourage the implementation of static testing.

**What Distinguishes Static Testing from Dynamic Testing?**

The main difference between static testing and dynamic testing is that dynamic testing is related to testing a system under execution. In dynamic testing we– for example– run our program and evaluate the results of certain inputs and outputs against what we expect it to output. We test how the system behaves in reality. On the other hand, as we have outlined already, static testing tests the system without executing it. In static testing we analyze the test items in the test basis to look for inconsistencies, contradictions, etc., and can review each of these items to varying degrees of formality to control for the direction of the the testing process.

**Can Static Testing Replace Dynamic Testing?**

So does that mean we can replace dynamic testing if we do enough static testing? **No.** It is not a binary case of one or the other, but rather they complement each other. Both types of testing reveals different defects. Static testing might resolve ambiguities that ensure in better validation of the system. Additionally, static testing can resolve questions pertaining to risk, and establish a course of action in relation to varying risk factors and risk levels. Meanwhile, dynamic testing might reveal contingencies, unforeseen events such as how the system behaves when a user inputs a wrong input, or if there are too many concurrent users on the system. Does it shut it down, become slow, or does it do something unexpected? These are questions that cannot be confidently answered unless it is tested in action— and as such, static testing is not sufficient on its own. Rather, knowing what incidents *might* occur and how to respond (Static testing) in combination with knowing what *actually* happens during a contingency (Dynamic testing) is better than just one or the other.

**Static Reviews - Types, Roles and Phases**

In addition to static analysis, we also have reviews. Reviews are a type of test activity where you revisit and go through a test item to refine, discuss, prioritize, and understand it. A review can vary in how formal it is, and its level of formality is dependent on the size, scope, limitations, requirement, etc. of the project. Below are common **types of reviews**, order from least to most formal:

- **Informal review**: this type of review can be done alone or with someone else. It is as implied, informal, and consists of revisiting a test item looking for potential defects or other issues. At this level of formality, it usually isn't documented, and this kind of review can be done by most people but is often used by developers.
- **Walkthrough**: this type of review is usually done in pairs (or more, but seldom so) and consists of an *author* explaining some test item, usually code, to their peer(s). This type of peer review is useful to sidestep author bias by getting a fresh pair of eyes on a test item. At this level, you might document something, but it is still optional.

- **Technical Review**: this type of review is more formal, and consists of more defined roles. There is the author of the test item, peers that ask questions, and might include a review leader that facilities the meeting and preparation material. It requires preparation beforehand, and the attendees come prepared to watch the author review a part of the software, and the attendees can ask questions. Both technical reviews and walkthroughs are commonly used to distribute knowledge about the product/project. At this level of formality the process is documented from beginning to end.
- **Inspection**: this type of review is the most formal of them all, and adheres to defined roles and responsibilities. As with a technical review, it consists of the author. This time there is also a *manager*, usually a product owner or similar, that allocates resources for the meeting to take place. A *review leader* that facilities the meeting by defining objectives, entry— and exit criteria, supplying preparation material, and creating post-review reports, as well as deciding who should be present. You also have a *moderator* who facilities productive conversation in the meeting, and ensures that the meeting stays on track and that the parties get heard. You have *attendees* who can ask questions and discuss with the author. Finally, you have a *scribe*, someone who takes notes during the review, making sure to highlight talking points and points of discussion. Inspections typically start with some established entry-criteria for facilitating the review, as well as some exit-criteria for ending the review.

In case you missed them, these are the **roles** mentioned so far:

- **Author** - The person under question that presents their work
- **Manager** - Allocates resources to allow for meetings to happen
- **Review leader** (or *Review Facilitator*) - Facilitates meetings, objectives and preparation material
- **Moderator** - Moderates the meetings to ensure productive discussion
- **Peers** - Attend the meeting prepared and ask questions to the author
- **Scribe** - Someone who makes notes during the meeting.

During a formal review, there are **multiple phases**.

1. **Preparation**: The first phase establishes the review process and includes setting the time and place for the meeting, and the review leader will collect and hand out preparation material to the other attendees.
2. **Review**: The actual meeting takes place, with members having come prepared and discussions are underway.
3. **Reporting**: The review leader collects the notes of the meeting, makes a report on the "results" of the meeting, establishing what was discussed and the outcomes.
4. **Rectification**: The actions points made during the review, which can include a decision that something needs to be done by certain people, is communicated to the relevant stakeholders, and they work on those action points to fix some problem.
5. **Follow-up**: The review leader follows up on the aforementioned stakeholders to verify whether the action points have been followed through and are fixed.