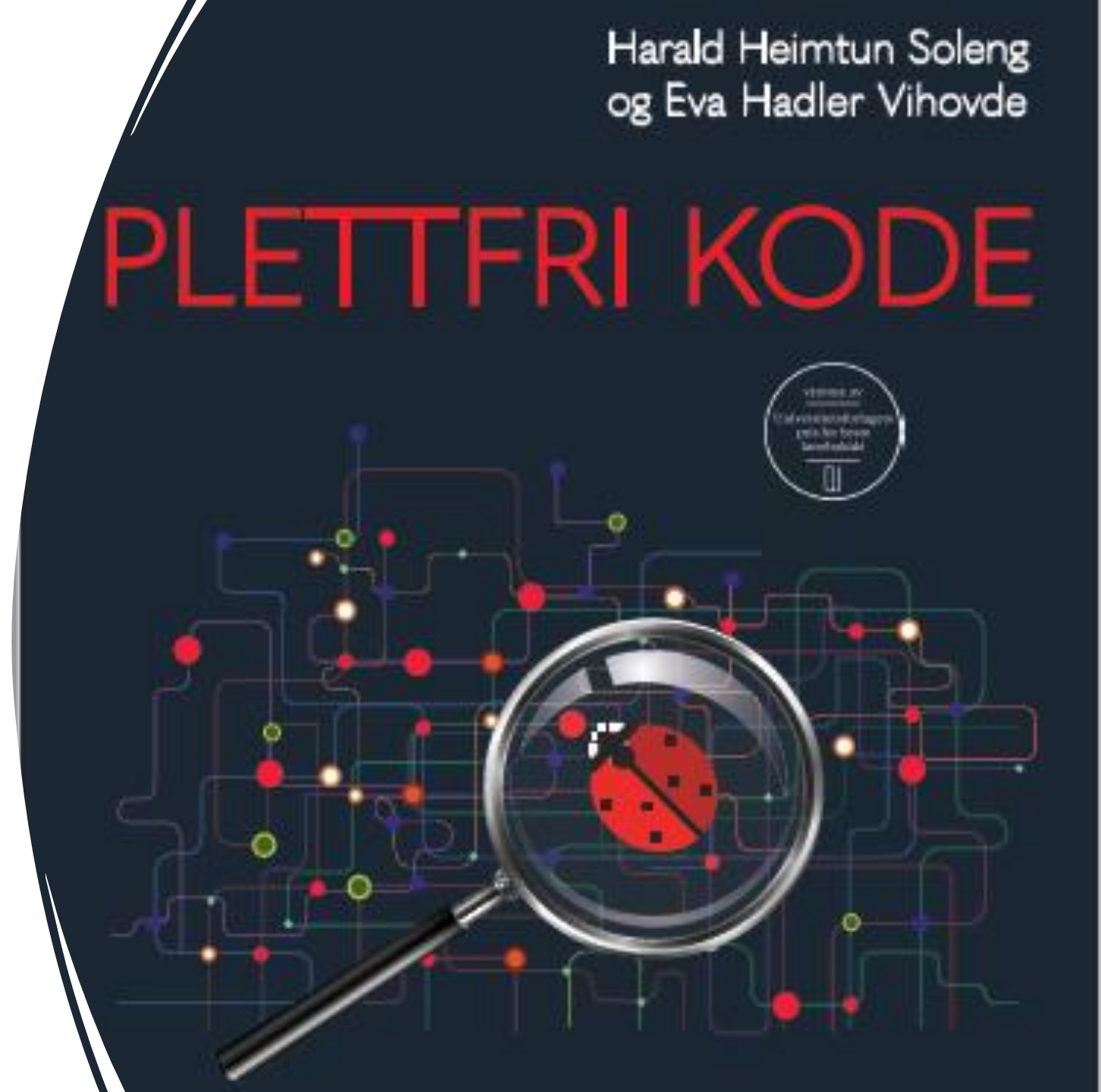


# Testing av programvare

Eva Hadler Vihovde – UiO / OsloMet

[Plettfri kode](#)

[Hvordan forhindre feil i programvare?](#)



# Kjente katastrofer



Ariane 5-raketten eksploderte som følge av heltallsoverflyt.  
Foto: ESA

# Mariner 1-krasj

Mariner-programmet (1959–1963) til NASA skjøt opp en serie interplanetariske romsonder for å undersøke Mars, Venus og Merkur. I 1962 feilet oppskytingen av Mariner 1 på grunn av en programmeringsfeil.

Forfatteren av spesifikasjonsdokumentet for styringssystemet overså symbolet for gjennomsnitt, en stripe, i en håndskrevet formel,

$$\overline{\dot{r}_n}$$

slik at formelen ble erstattet med

$$\dot{r}_n$$

I stedet for å korrigere ut fra variasjoner i en gjennomsnittsverdi, forsøkte styringssystemet å korrigere for hver lille tilfeldige variasjon. Alle de meningsløse korreksjonene endte med at raketten havarerte.

## Nesten-ulykke i bemannet romrakett

Mercury-programmet var det første bemannede amerikanske romprogrammet. Programmeringsspråket som ble brukt den gang, het Fortran, og i koden som beregnet raketts bane, fantes følgende linje:

```
      DO 17 I = 1.100  
        :   :  
17    CONTINUE
```

i stedet for

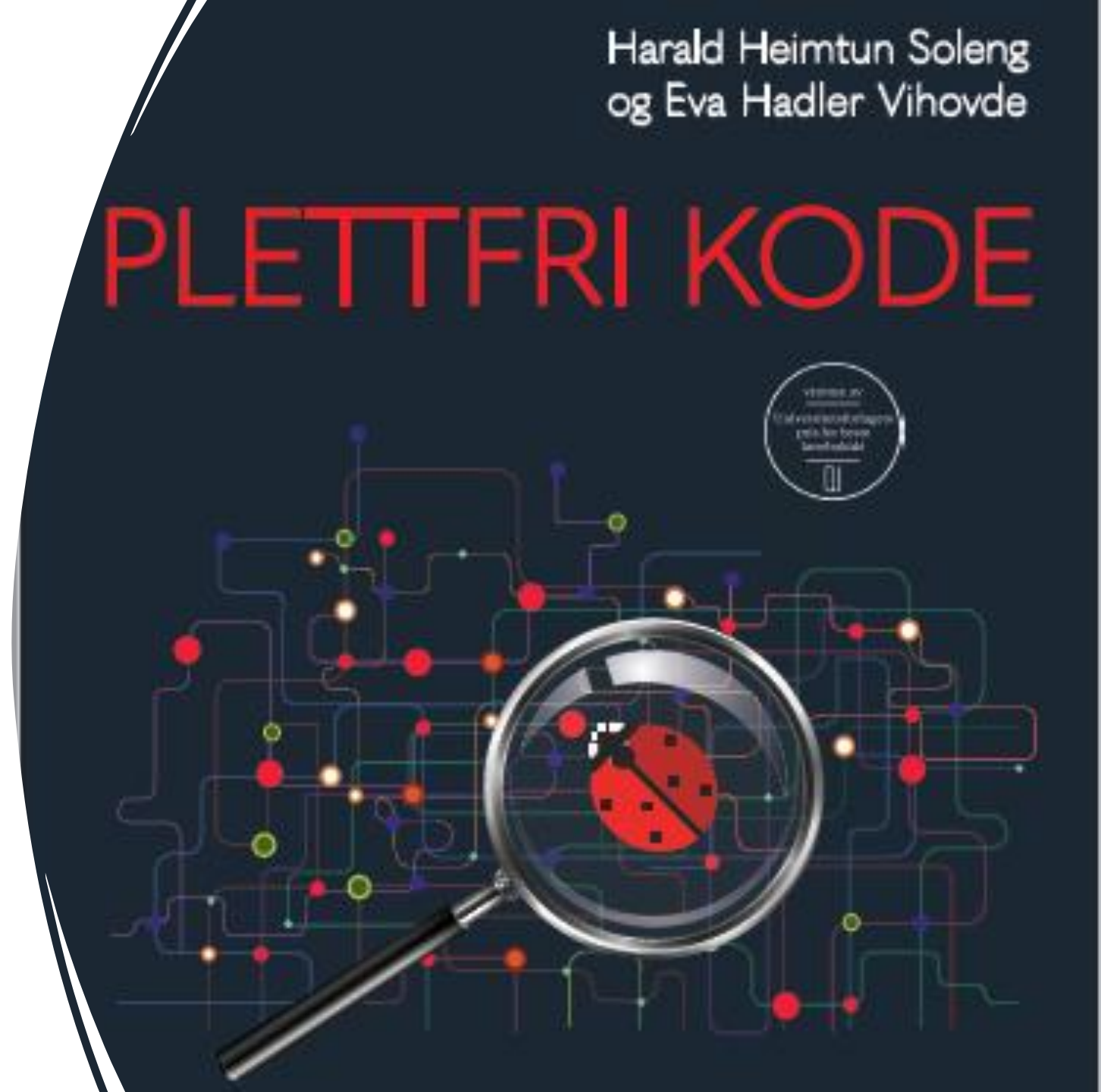
```
      DO 17 I = 1,100  
        :   :  
17    CONTINUE
```

# Testing av programvare

Eva Hadler Vihovde – UiO / OsloMet

[Plettfri kode](#)

[Hvordan forhindre feil i programvare?](#)



# Hvem er boken skrevet for?

- Utviklere
- Testledere
- Testere
- Prosjektledere
- Oppdragsgivere

# Innhold

1 Dødelige tegn og kommafeil

2 Kilder til feil

3 Syv prinsipper for testing

4 Arbeidsprosessen

5 Testing i produktets livssyklus

6 Å forhindre feil

7 Solid og testbar kode

8 Testprosessen

9 Statistiske testteknikker

10 Spesifikasjonsbaserte teknikker

11 Strukturbaserte testteknikker

12 Erfaringsbaserte testteknikker

13 Kildekodedokumentasjon

14 Praktisk testing

15 Testingsens psykologi

16 Automatisert testing

**17 Testguide for studentprosjekter**

A En fikspunktimplementering

# Agenda

- Syv prinsipper for testing
- Testing i produktets livssyklus
- Spesifikasjonsbaserte teknikker

# Agenda

- Syv prinsipper for testing
- Testing i produktets livssyklus
- Spesifikasjonsbaserte teknikker

# Syv prinsipper for testing

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

# Syv prinsipper for testing

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

# Syv prinsipper for testing

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

# Syv prinsipper for testing

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

# Syv prinsipper for testing

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

# Syv prinsipper for testing

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

# Syv prinsipper for testing

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

# Syv prinsipper for testing

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

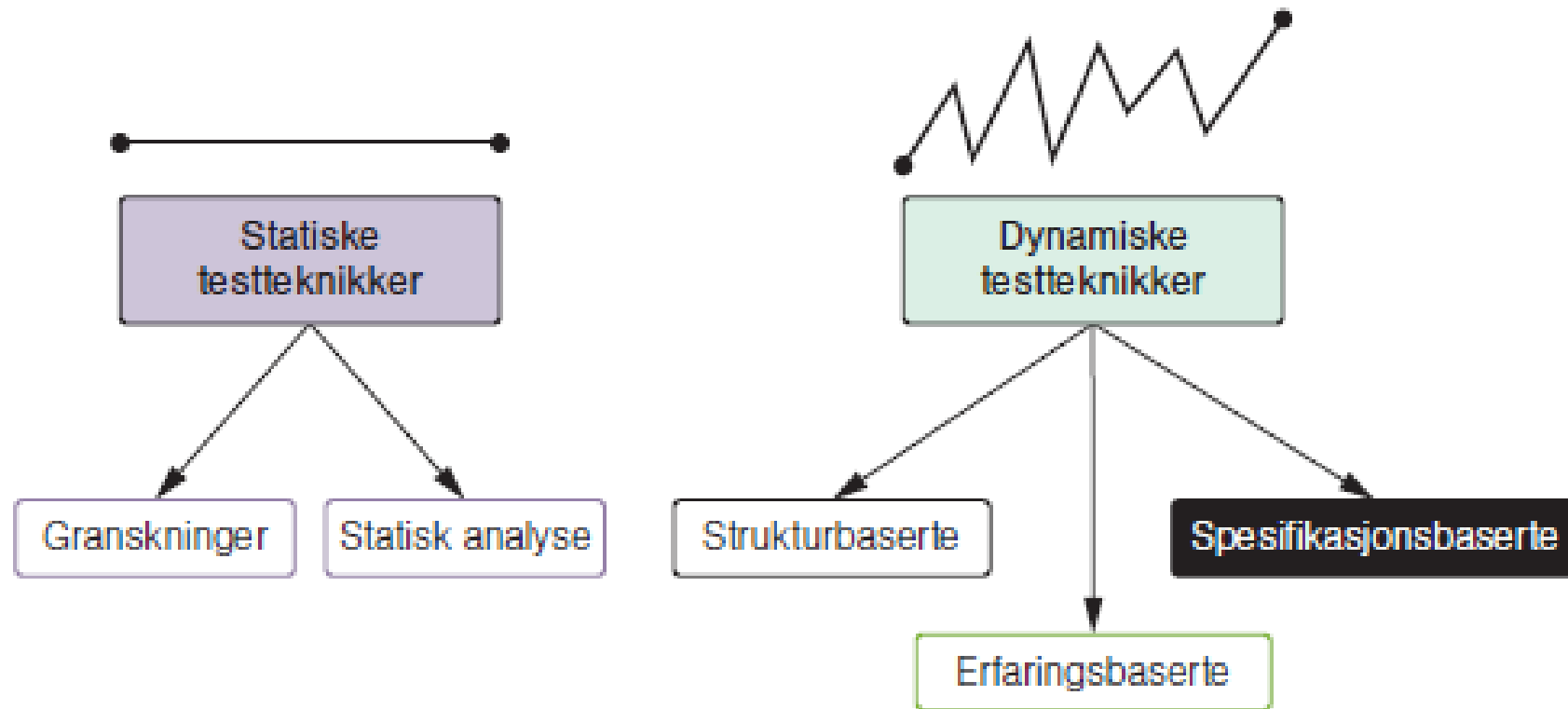
Prinsipp 6. Testing er avhengig av konteksten

Prinsipp 7. Feilslutning vedrørende fravær av feil

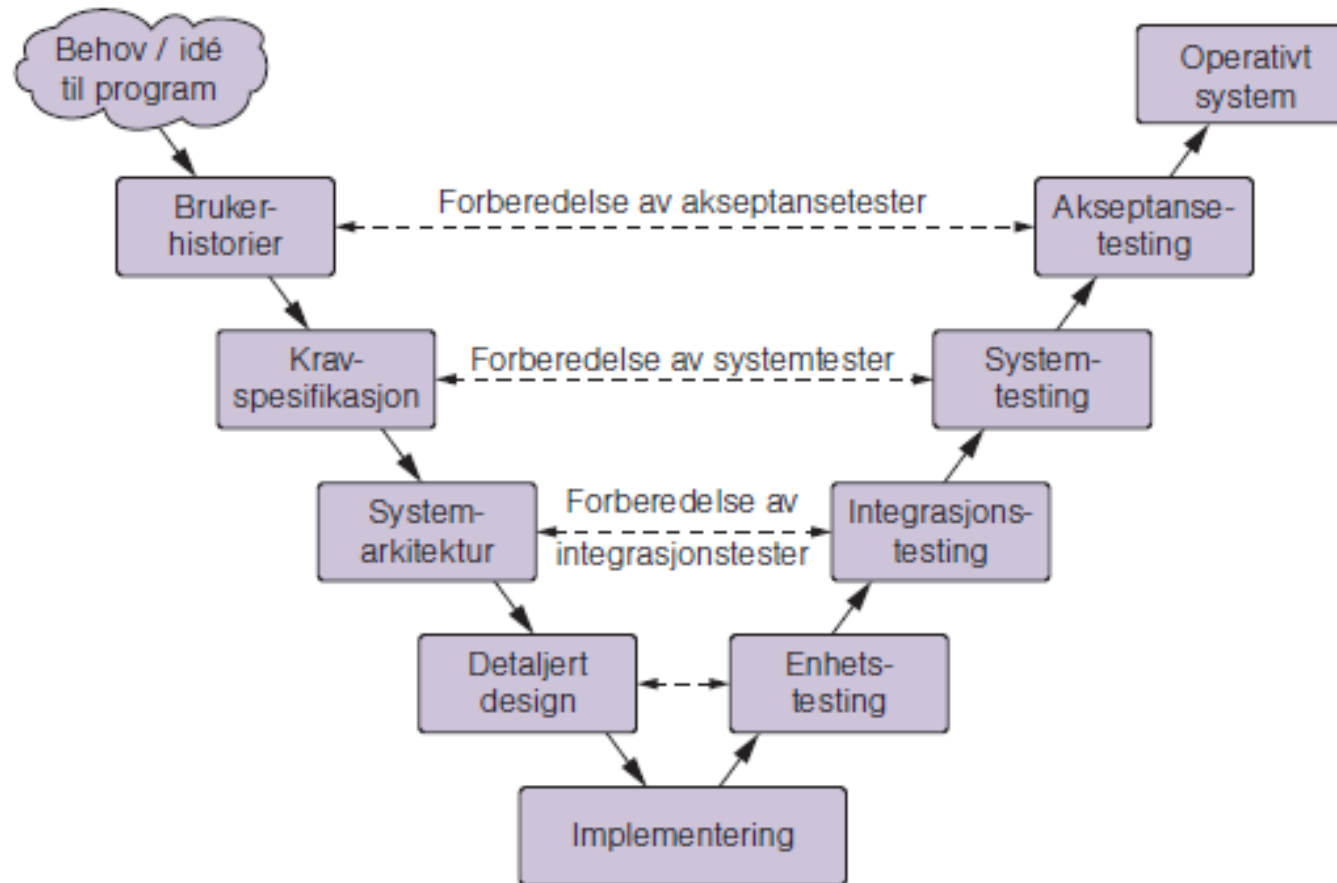
# Agenda

- Syv prinsipper for testing
- Testing i produktets livssyklus
- Pause
- Spesifikasjonsbaserte teknikker

# Statiske og dynamiske testteknikker



# V-modellen



# Startfasen

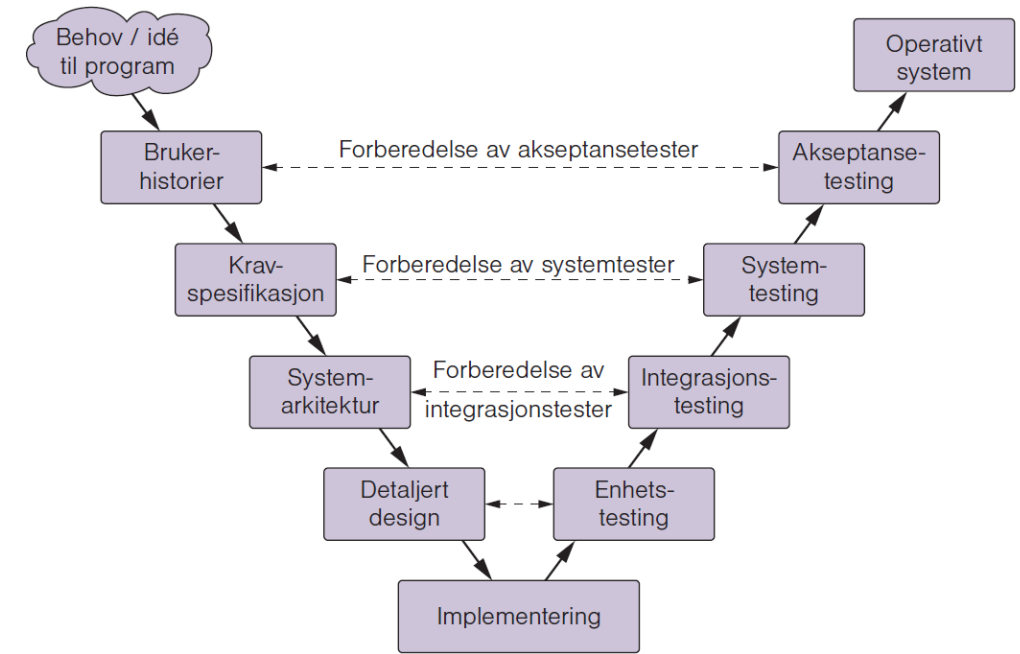
Er prosjektet liv laga?

Forstå oppdraget

Kartlegg konteksten

Bestem testingens mål og omfang

Programdokumentasjon og testrapport



Figur 5.2: V-modellen.

# Programmets fundament – kravspesifikasjonen

## Analysér testforholdene og design testtilfellene

Ekvivalensklasseinndeling og grenseverdianalyse

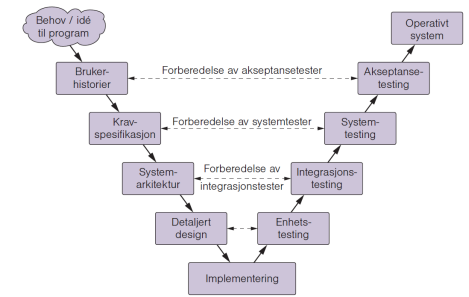
Tilstandsbasert testing

Beslutningstabeller

Brukerhistorier

Systemarkitektur

Detaljert design



Figur 5.2: V-modellen.

# Dynamisk testing gjennom fire testnivåer

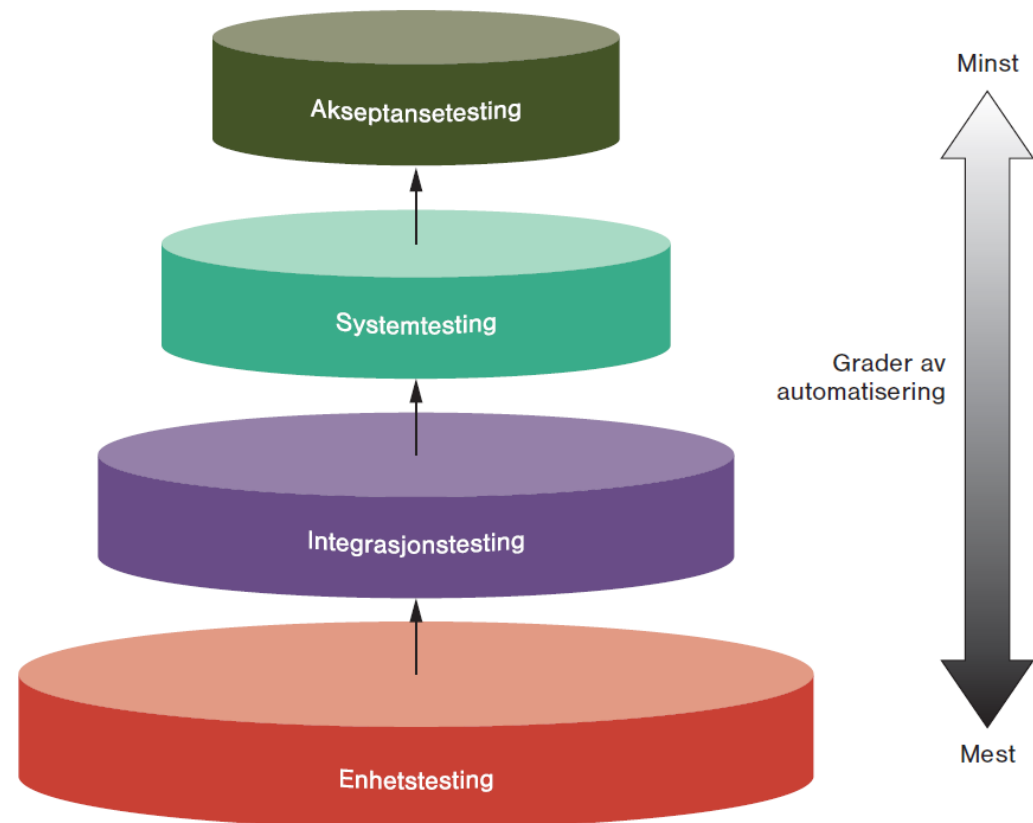
Enhetstesting

Integrasjonstesting

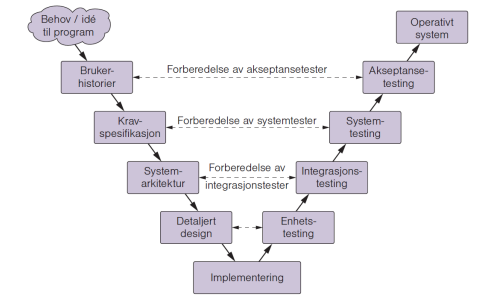
Systemtesting

Akseptansetesting

Regresjonstesting

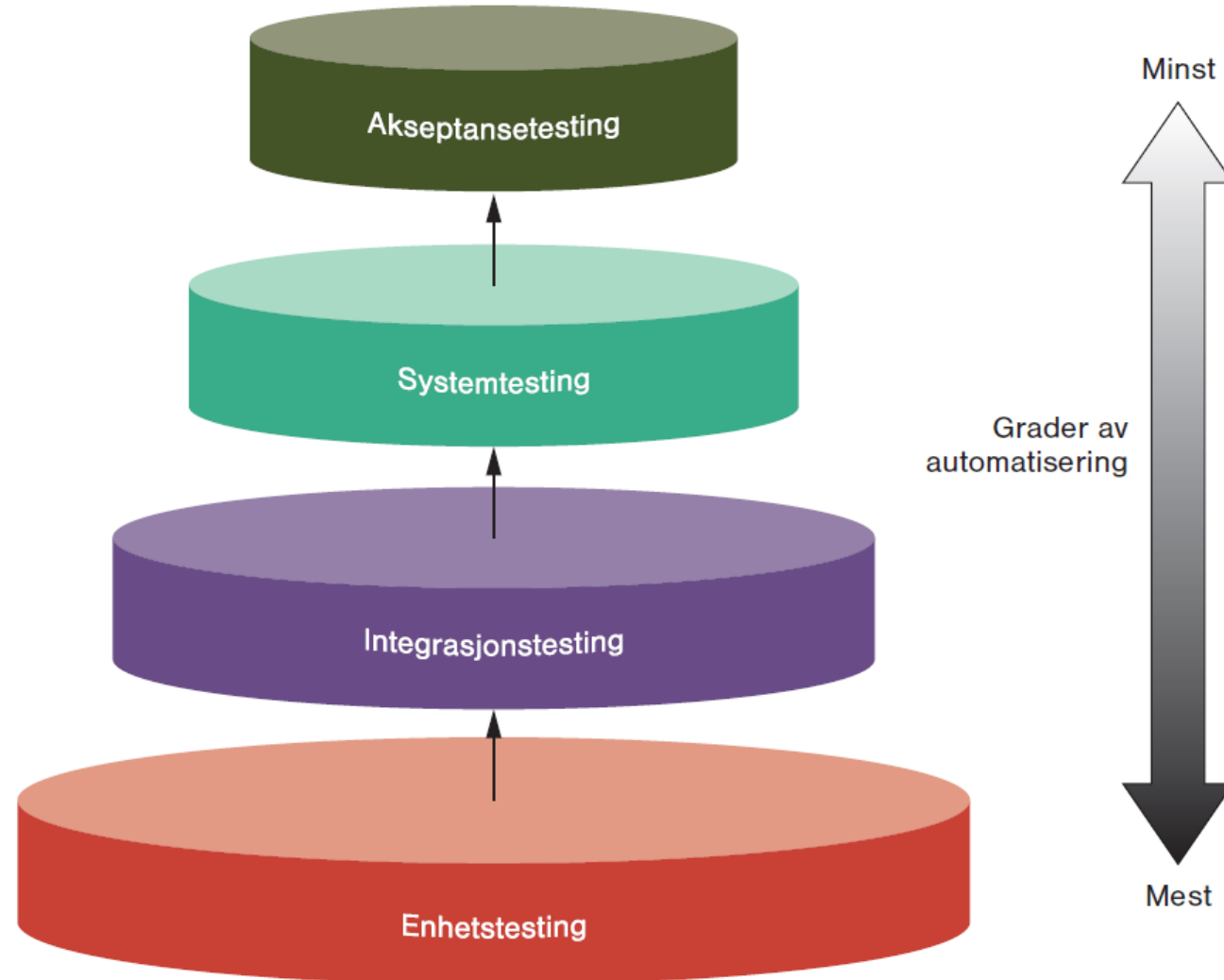


Figur 16.1: Anbefalt testautomatiseringsgrad.



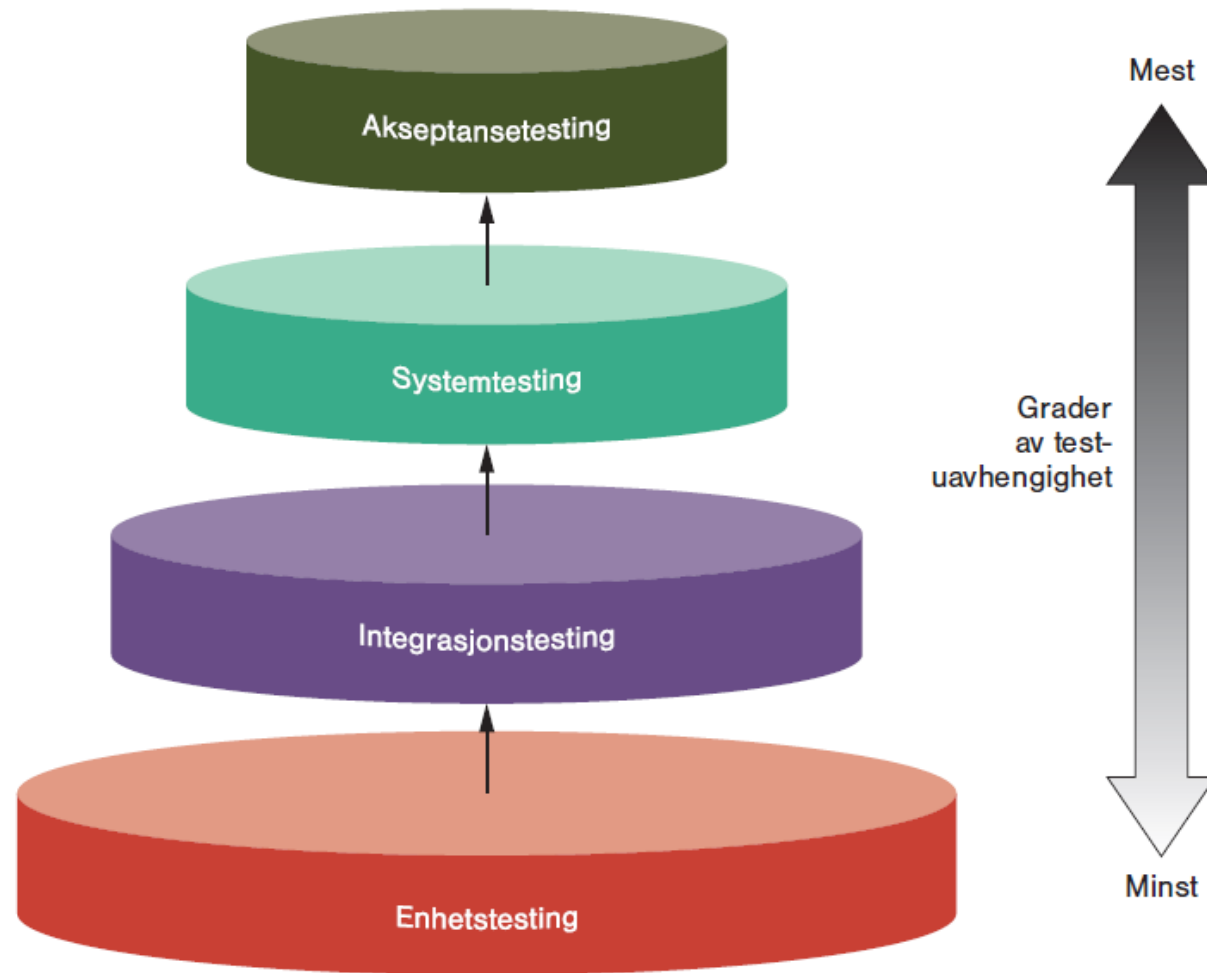
Figur 5.2: V-modellen.

# Testautomatisering



Figur 16.1: Anbefalt testautomatiseringsgrad.

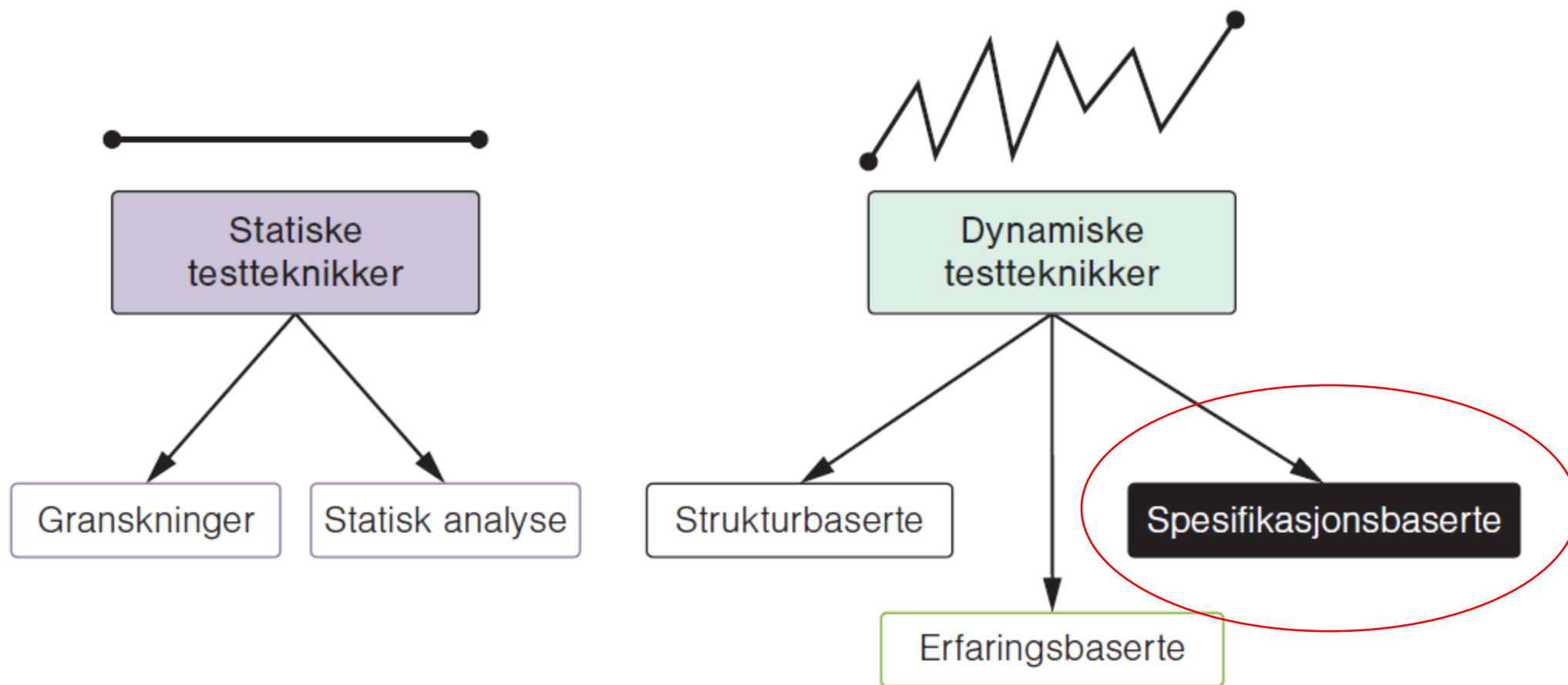
# Testuavhengighet



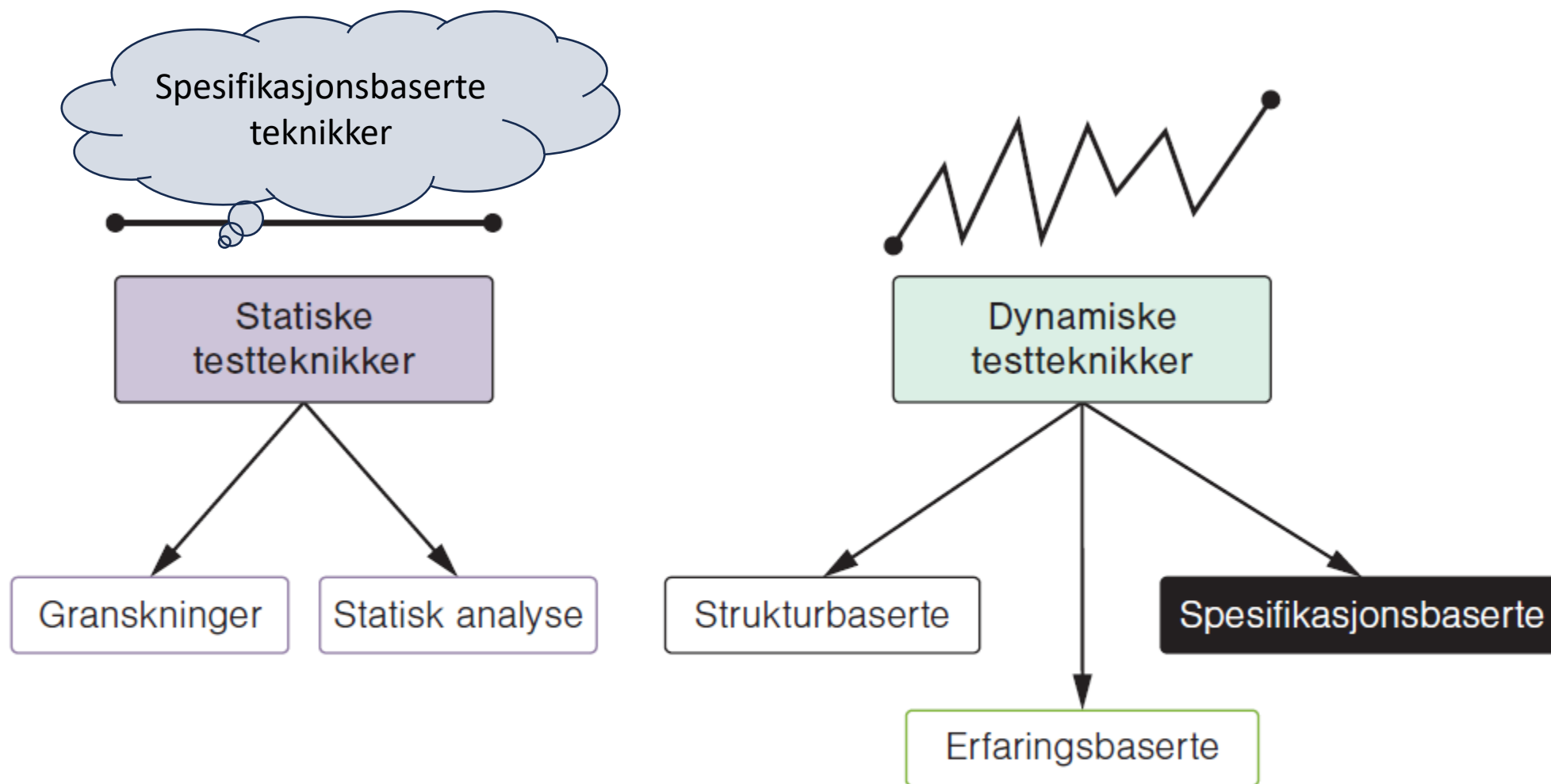
Figur 15.1: Grader av uavhengighet.

# Agenda

- Syv prinsipper for testing
- Testing i produktets livssyklus
- Spesifikasjonsbaserte teknikker

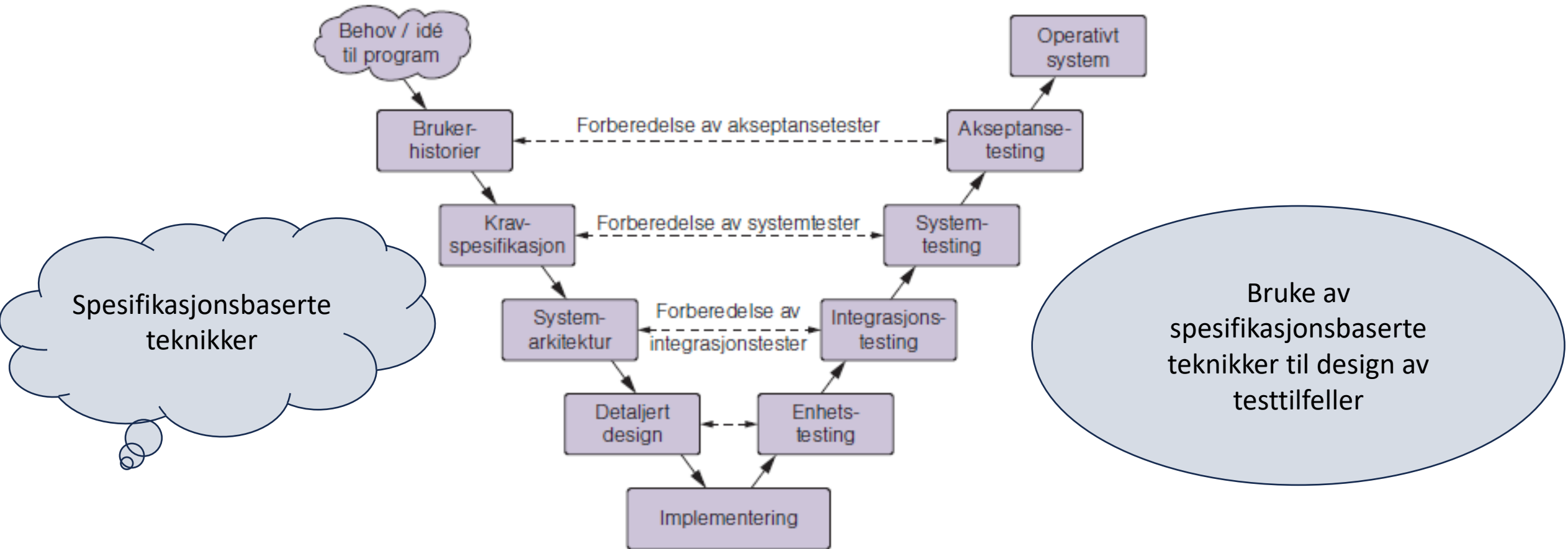


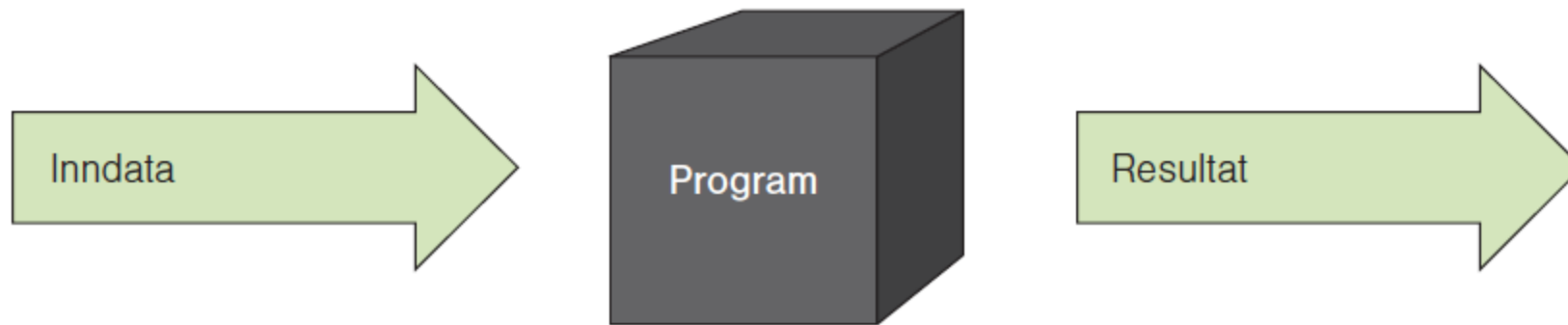
Figur 5.1: Statiske og dynamiske testteknikker.



Figur 5.1: Statiske og dynamiske testteknikker.

# V-modellen

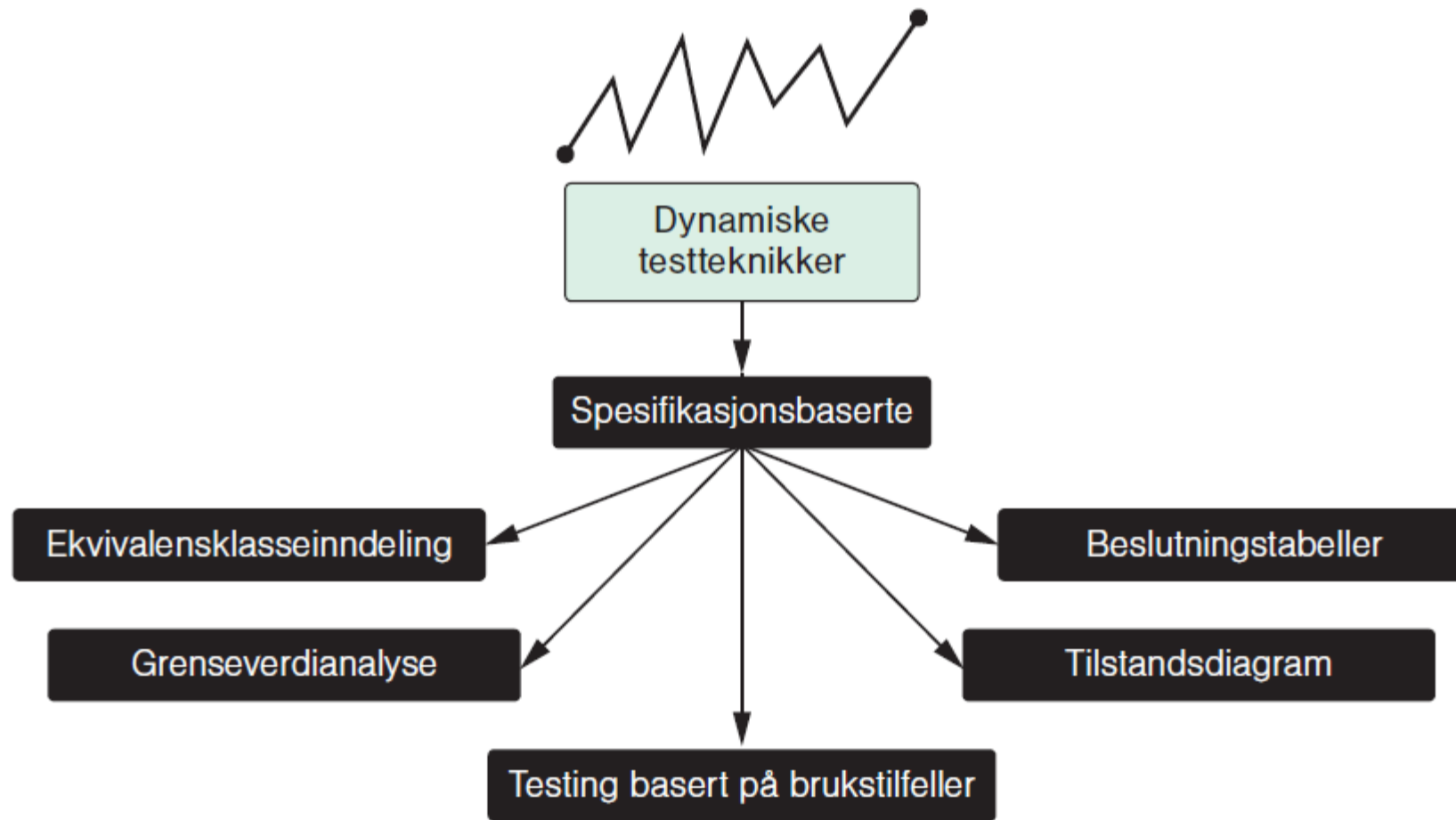




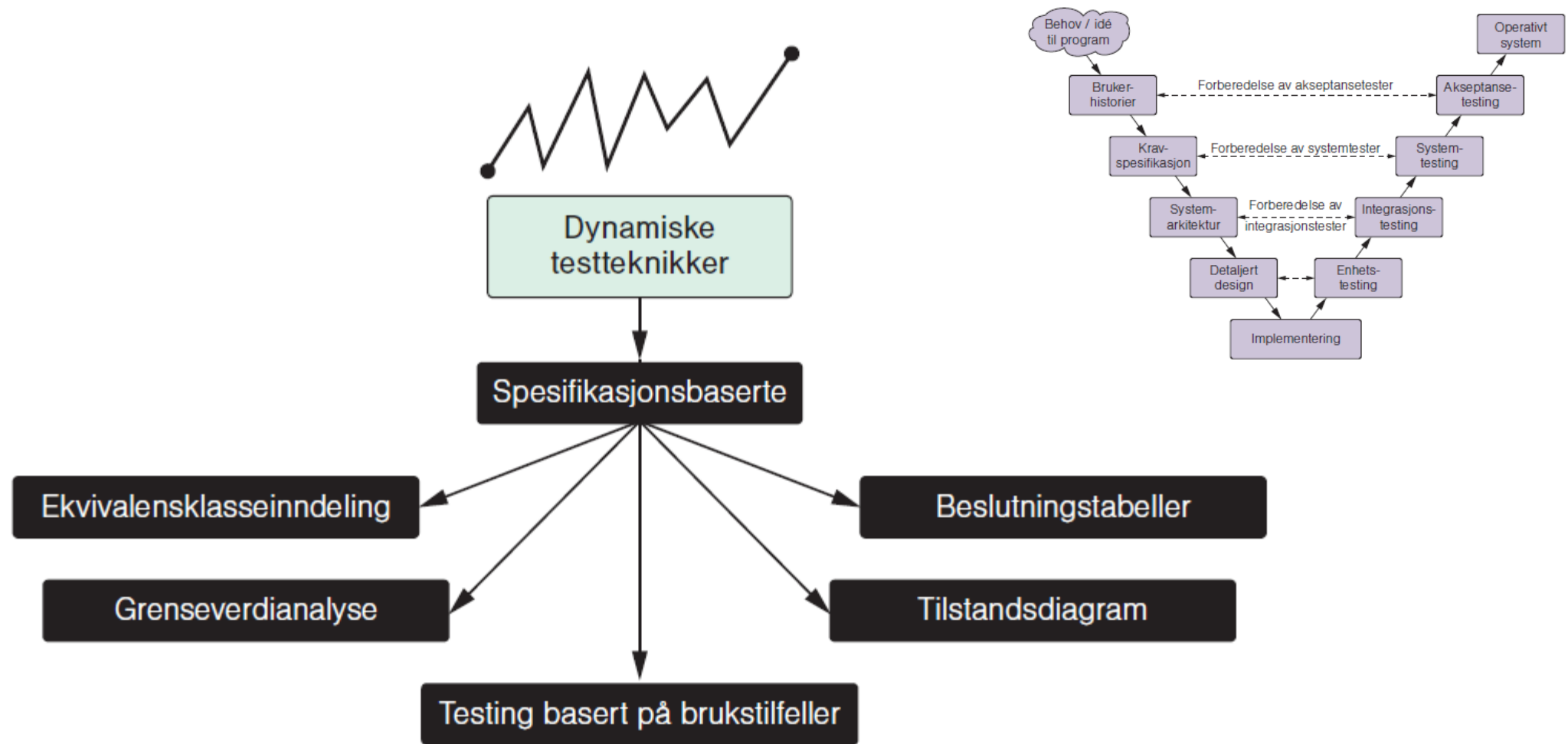
Figur 10.2: Svart boks-testing.



Figur 8.2: Testdesign.



Figur 10.1: Spesifikasjonsbaserte teknikker.



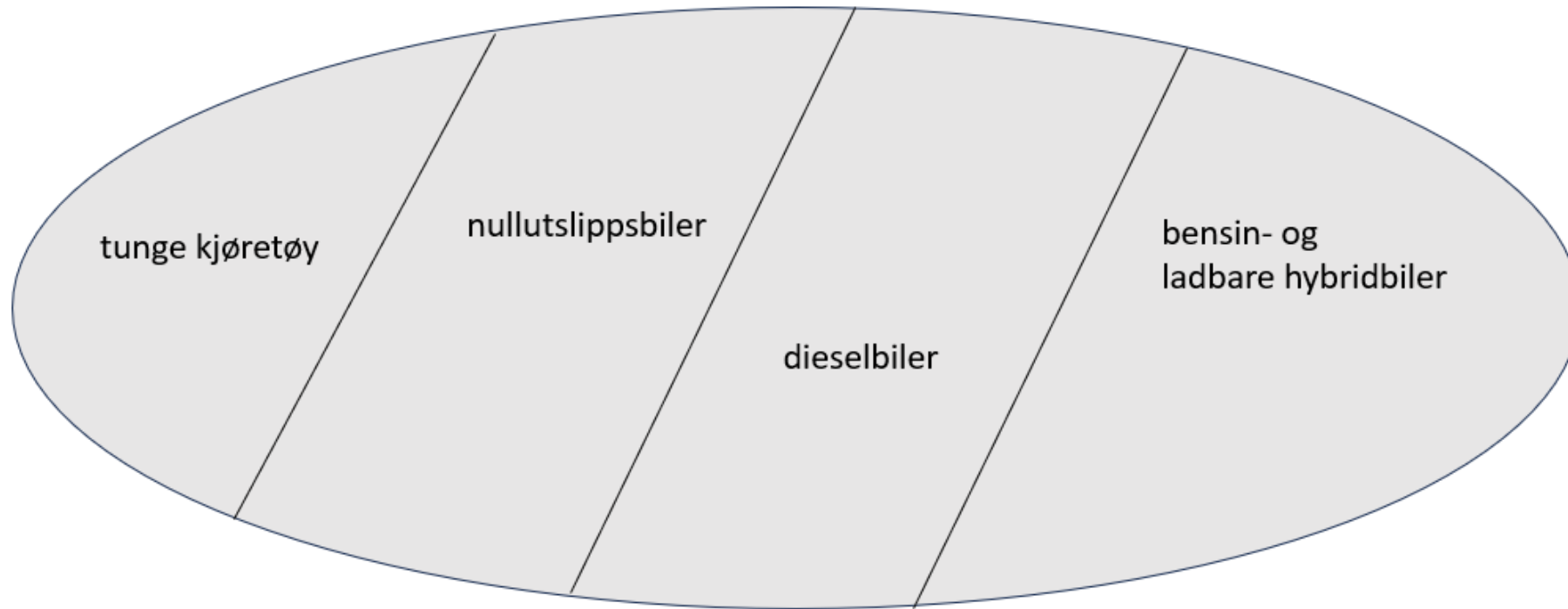
Figur 10.1: Spesifikasjonsbaserte teknikker.

# Ekvivalensklasseinndeling

Tenk deg at vi skal teste et program som drifter en bomring der ulike kjøretøy faktureres automatisk når bomringen passerer. Kjøretøyene er inndelt i fire grupper; bensin- og ladbare hybridbiler, dieserbiler, nullutslippsbiler eller tunge kjøretøy, dvs. kjøretøy over en gitt tillatt totalvekt. Alle kjøretøy innen hver gruppe betaler det samme, men prisene varierer fra gruppe til gruppe. For å teste om programmet beregner faktureringen riktig, er det naturlig å lage fire tester, dvs. en test for hver type kjøretøy. Vi kunne ha valgt en annen strategi, f.eks. testet hvert tiende kjøretøy som passerte bomstasjonen, men da ville vi risikere å måtte utføre mange flere tester før vi kunne være sikre på at faktureringen var riktig i alle situasjoner. Ved å velge kun ett kjøretøy fra hver gruppe kan vi greie oss med fire tester for å få dekket alle situasjonene.

For å bruke ekvivalensklasseinndeling må to krav være oppfylt:

1. Alle delmengdene må til sammen utgjøre hele mengden de har utgått
2. Delmengdene må være disjunkte, dvs. de kan ikke ha noen felles elementer.



Hvis man tester ett kjøretøy fra hver kjøretøygruppe (delmengde) opp når man 100% ekvivalensklassedekning.

# Grenseverdianalyse

La oss se på et annet eksempel. I en eksamen regnes det ut en poengsum for hver besvarelse. Tenk deg at du skal teste en metode som konverterer en poengsum mellom 0 og 100 til en bokstavkarakter. Poengsummene angis i hele tall, og konverteringen skal skje i henhold til følgende skala:

|      |       |       |       |       |        |
|------|-------|-------|-------|-------|--------|
| 0–39 | 40–49 | 50–59 | 60–79 | 80–89 | 90–100 |
| F    | E     | D     | C     | B     | A      |

NB! Hvis mengden som skal inndeles, inneholder elementer som kan sorteres i en rekkefølge, er det ikke tilstrekkelig å teste en verdi fra hver ekvivalensklasse. Da bør vi i tillegg bruke grenseverdianalyse.

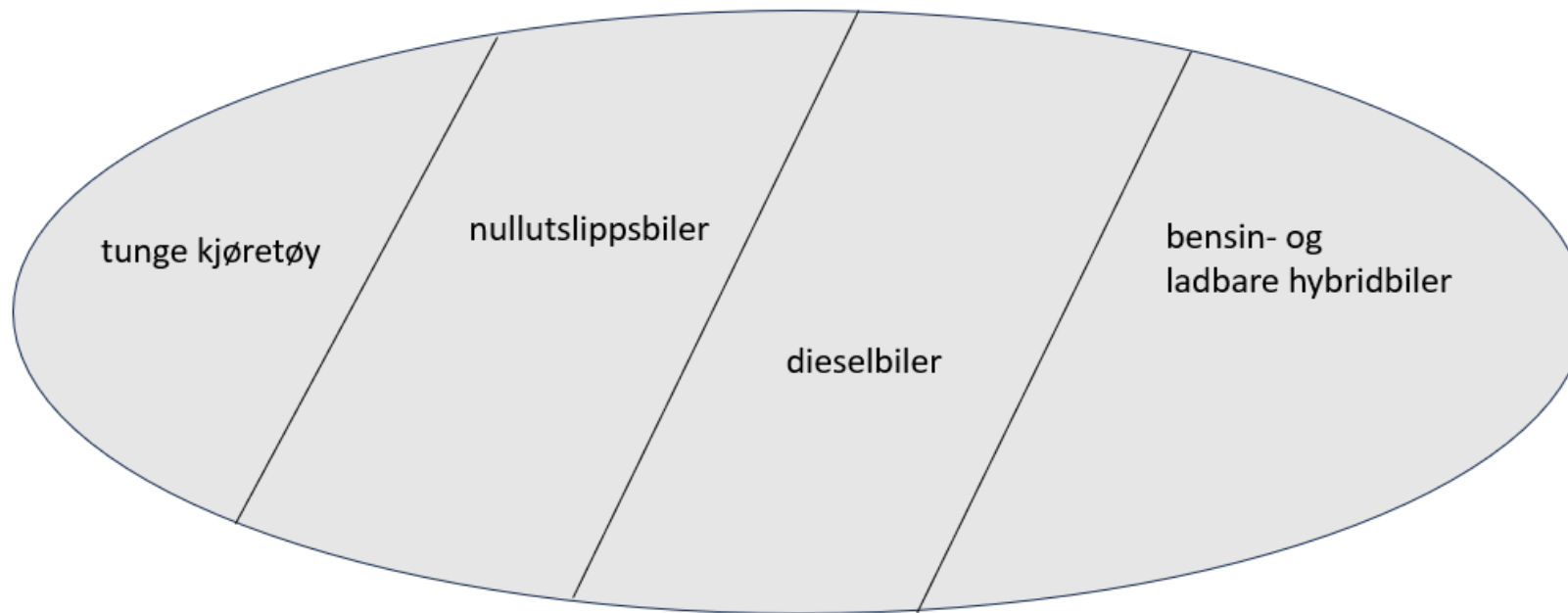
# Grenseverdianalyse og ekvivalensklasser

Vi får 8 ekvivalensklasser: 6 gyldige og 2 ugyldige.

På hver side av grensene mellom to ekvivalensklasser er det grenseverdier som bør testes.

|         |      |       |       |       |       |        |         |
|---------|------|-------|-------|-------|-------|--------|---------|
| ... -1  | 0-39 | 40-49 | 50-59 | 60-79 | 80-89 | 90-100 | 101 ... |
| Ugyldig | F    | E     | D     | C     | B     | A      | Ugyldig |

Som en tommelfingerregel bør vi teste en verdi fra midten av hver ekvivalensklasse, i tillegg til grenseverdiene.



Figur 10.3: Rushtider.

I bomring-eksempelet var det ingen grenseverdier. Men hvis vi innfører rushtidsavgift, får vi grenseverdier på hver side av alle tidsgrensene: 06:59, 07:00, 08:59, 09:00, osv. Disse må også testes.

# Beslutningstabelltesting

Teknikken med ekvivalensklasseinndeling og grenseverdianalyse er imidlertid ikke like godt egnet til testing av program- og forretningslogikk, der kombinasjoner av inndata resulterer i ulike handlinger (aksjoner) fra programmets side. Da kan beslutningstabeller være et godt supplement eller alternativ.

Sannhetsverdier for to betingelser:

|              |          |          |          |          |
|--------------|----------|----------|----------|----------|
| Betingelse 1 | <i>S</i> | <i>S</i> | <i>U</i> | <i>U</i> |
| Betingelse 2 | <i>S</i> | <i>U</i> | <i>S</i> | <i>U</i> |

Her sees en kodesnutt med tre betingelser nøstet i hverandre.

Er antall betingelser som er nøstet i hverandre høyt, er det fort gjort å miste oversikten over konsekvensene av alle de mulige kombinasjonene av sann og usann.

Antall mulige kombinasjoner er  $2^n$ , der  $n$  er antall betingelser.

```
while (<betingelse 1>
{
    while (<betingelse 2>)
    {
        if (<betingelse 3>)
        {
            <instruksjon 1>;
        }
        else
        {
            <instruksjon 2>;
        }
    }
    <instruksjon 3>;
}
<instruksjon 4>;
```

# Eksempel på bruk av beslutningstabell:

Tenk deg at en bank skal ta i bruk en webapplikasjon som skal behandle bankkundernes boliglånsøknader «online». For å få boliglån må imidlertid visse krav til jobb, inntekt og egenkapital være innfridd:

- Søkeren må ha fast inntekt.
- Søkeren må ha minimum 15 % egenkapital.
- Lånebeløpet kan maksimalt være fem ganger årsinntekten.

Hvis søkeren ikke har en fast inntekt, men tilfredsstiller de øvrige betingelsene, er det fortsatt mulig å få lån. Det krever imidlertid at søknaden behandles manuelt. I dette tilfellet vil programmet tildele søkeren en saksbehandler.

| <b>Spørsmål</b>                                         | <b>Navn på betingelse</b> |
|---------------------------------------------------------|---------------------------|
| Har søkeren fast inntekt?                               | Fast inntekt              |
| Har søkeren minimum 15 % egenkapital?                   | Nok egenkapital           |
| Er lånebeløpet ikke større enn fem ganger årsinntekten? | Passe lånebeløp           |

| <b>Handling</b>                      | <b>Navn på aksjon</b> |
|--------------------------------------|-----------------------|
| Lånesøknaden skal innvilges          | Innvilgelse           |
| Lånesøknaden skal avslås             | Avslag                |
| Søknaden sendes til en saksbehandler | Manuell behandling    |

| Regel nr.       | 1        | 2        | 3        | 4        | 5        | 6        | 7        | 8        |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| Fast inntekt    | <i>S</i> | <i>S</i> | <i>S</i> | <i>S</i> | <i>U</i> | <i>U</i> | <i>U</i> | <i>U</i> |
| Nok egenkapital | <i>S</i> | <i>S</i> | <i>U</i> | <i>U</i> | <i>S</i> | <i>S</i> | <i>U</i> | <i>U</i> |
| Passe lånebeløp | <i>S</i> | <i>U</i> | <i>S</i> | <i>U</i> | <i>S</i> | <i>U</i> | <i>S</i> | <i>U</i> |
|                 |          |          |          |          |          |          |          |          |
| Lån innvilges   | Ja       | Nei      | Nei      | Nei      | Saksb.   | Nei      | Nei      | Nei      |

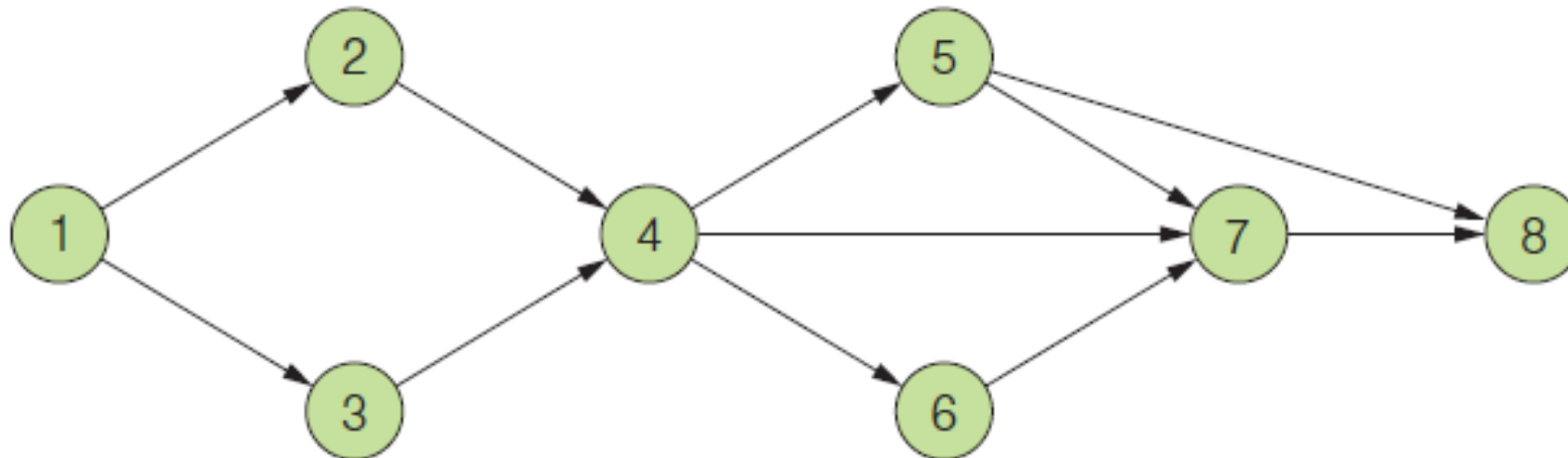
| Regel nr.       | 1        | 2        | 3        | 4        | 5        | 6        | 7        | 8        |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| Fast inntekt    | <i>S</i> | <i>S</i> | <i>S</i> | <i>S</i> | <i>U</i> | <i>U</i> | <i>U</i> | <i>U</i> |
| Nok egenkapital | <i>S</i> | <i>S</i> | <i>U</i> | <i>U</i> | <i>S</i> | <i>S</i> | <i>U</i> | <i>U</i> |
| Passe lånebeløp | <i>S</i> | <i>U</i> | <i>S</i> | <i>U</i> | <i>S</i> | <i>U</i> | <i>S</i> | <i>U</i> |
| Lån innvilges   | Ja       | Nei      | Nei      | Nei      | Saksb.   | Nei      | Nei      | Nei      |

| Regel nr.       | 1        | 2, 4, 6 og 8 | 3 og 7   | 5        |
|-----------------|----------|--------------|----------|----------|
| Fast inntekt    | <i>S</i> | —            | —        | <i>U</i> |
| Nok egenkapital | <i>S</i> | —            | <i>U</i> | <i>S</i> |
| Passe lånebeløp | <i>S</i> | <i>U</i>     | <i>S</i> | <i>S</i> |
| Lån innvilges   | Ja       | Nei          | Nei      | Saksb.   |

Hver regel angir et testtilfelle der inputverdiene må velges slik at betingelsene får de sannhetsverdier som regelen angir.

# Tilstandsbasert testing

Hvis vi kan betrakte programmet som et endelig antall tilstander, der programkontrollen beveger seg fra tilstand til tilstand, alt etter hva som skjer under kjøringen, kan programmert visualiseres i form av et tilstandsdiagram:



En sti fra starttilstanden (1), der pilene følges gjennom grafen, til slutttilstanden (8), angir et testtilfelle.

## Eksempel på tilstandsbasert testing

1: Du kan velge å sjekke inn med bookingnummer eller pass.

2: Du har valgt å sjekke inn med bookingnummer. Skriv inn nummeret og trykk «Neste».

Hvis nummeret ikke er riktig, går automaten tilbake til tilstand 2, hvor du kan prøve igjen.

Hvis nummeret er riktig, går automaten til tilstand 4.

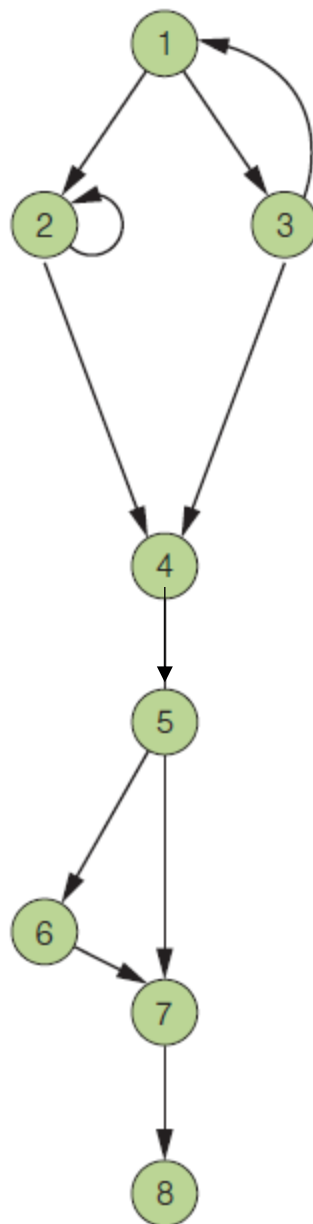
5: Du blir spurt om du vil reservere et sete eller endre seterreservasjonen (hvis du allerede har reservert sete).

Hvis du bekrefter, går automaten til tilstand 6.

Hvis du ikke bekrefter, går automaten til tilstand 7.

6: Du velger et ledig sete og trykker «Neste».

8: Automaten ønsker deg en behagelig flyreise.



3: Du har valgt å sjekke inn med pass. Plasser passet i leseren.

Hvis passet avvises, går automaten tilbake til tilstand 1.

Hvis passet godtas, går automaten til tilstand 4.

4: Du blir nå bedt om å taste inn antall bagasjeenheter du vil sjekke inn. Når antallet er tastet inn og du trykker «Neste», går automaten til tilstand 5.

7: Boardingkortet og eventuelle bagasjemerker blir skrevet ut. Når du har tatt begge deler, går automaten til tilstand 8.

# Testing basert på brukstilfeller

Testingen baserer seg på en beskrivelse den forventede bruken av programmet ut fra de arbeidsoppgaver sluttbrukeren skal bruke det til, der interaksjonen mellom bruker og maskin er beskrevet trinn for trinn.

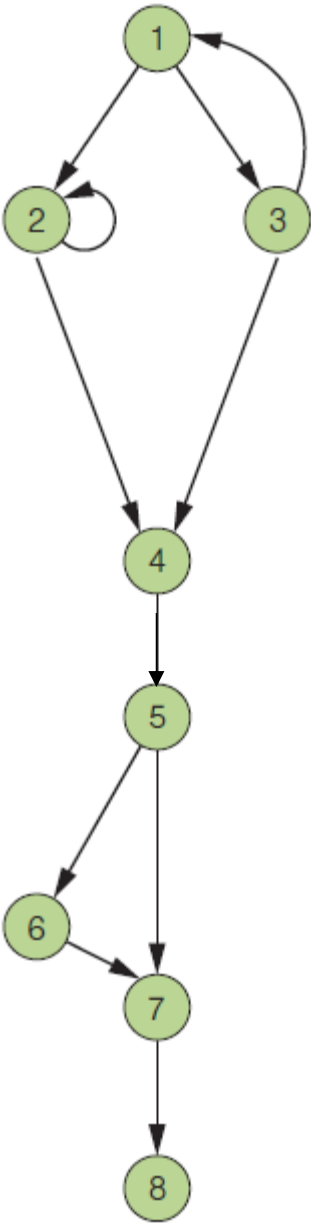
Testtilfellene kan utvikles fra

- Brukerhistorier
- Ekvivalensklasseinndeling
- Grenseverdianalyse
- Beslutningstabeller
- Tilstandsdiagram

Eksempel på testing basert på brukstilfeller, der testtilfellet er basert på et tilstandsdiagram.

Hovedscenario innsjekking med bookingnummer:

|               |       |              |                                           |
|---------------|-------|--------------|-------------------------------------------|
| Hovedscenario | Trinn | Aktør        | Beskrivelse                               |
|               | 1     | Automaten    | Ber den reisende velge innsjekkingsmetode |
|               | 2     | Den reisende | Velger å sjekke inn med bookingnummer     |
|               | 3     | Automaten    | Ber den reisende taste inn nummer         |
|               | 4     | Den reisende | Taster inn nummeret og trykker «Neste»    |
|               | 5     | Automaten    | Validerer nummeret. Nummeret godkjent     |
|               | 6     | Automaten    | Spør om antall kolli innsjekket bagasje   |
|               | 7     | Den reisende | Taster inn antallet og trykker «Neste»    |
|               | 8     | Automaten    | Spør om ny eller endret setereservasjon   |
|               | 9     | Den reisende | Velger «Ja»                               |
|               | 10    | Automaten    | Ber den reisende velge ledig sete         |
|               | 11    | Den reisende | Velger ledig sete og trykker «Neste»      |
|               | 12    | Automaten    | Skriver ut boardingkort og bagasjemerker  |
|               | 13    | Den reisende | Tar boardingkort og ev. bagasjemerke(r)   |
|               | 14    | Automaten    | Ønsker god reise                          |



|                                                                 |            |
|-----------------------------------------------------------------|------------|
| <b>17 Testguide for studentprosjekter</b>                       | <b>147</b> |
| 17.1 Generelle råd . . . . .                                    | 148        |
| 17.2 Startfasen . . . . .                                       | 149        |
| 17.3 Programmets fundament – kravspesifikasjonen . . . . .      | 151        |
| 17.4 Analyser testforholdene og design testtilfellene . . . . . | 152        |
| 17.5 Dynamisk testing gjennom fire testnivåer . . . . .         | 153        |
| 17.6 Regresjonstesting . . . . .                                | 155        |
| 17.7 En siste test . . . . .                                    | 155        |
| 17.8 Testing kan ikke bevise at koden er feilfri . . . . .      | 156        |
| 17.9 Avsluttende ord . . . . .                                  | 156        |

Takk for oppmerksomheten!