sopra steria

TECHNOLOGY MANAGEMENT
DANMARK

Fra manuel til automatiseret test med AI

En praktisk håndbog med **de fem trin, skabelonerne og faldgruberne**, fra den første kortlægning til en testsuite, der kører i jeres pipeline.

HÅNDBOGENS FEM TRIN

01

Kortlæg

Find de 20 %, der skal automatiseres først, og det, der bevidst bliver liggende.

02

Gør kravene testbare

Fra uklart acceptkriterie til noget, to personer ville teste ens.

03

Optag adfærden

Workshop-opskriften. Hvem, hvor længe, og hvad der kommer ud.

04

Generér og validér

Hvad en brugbar testcase indeholder, og hvem der godkender den.

05

Kør og mål

Pipeline, defect-flow og de seks tal, I skal tage baseline på.

SKREVET TIL

Testansvarlige, QA-leads og udviklingsleads

LÆSETID

20 minutter

INDEHOLDER

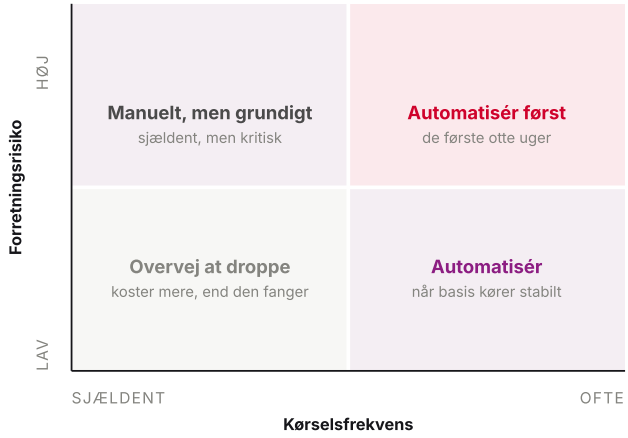
Skabeloner, et gennemarbejdet prompteksempel og en 8-ugers plan

Skal jeres ledelse overbevises først? Hent business casen i **Kom i gang med AI-drevet software testing** på soprasteria.dk.

Automatisér ikke alt. Automatisér det rigtige.

Den dyreste fejl i et automatiseringsprojekt er at gå bredt ud fra dag ét. Start med en kortlægning, der er god nok til at træffe beslutninger på. Den tager typisk **to til tre dage**, ikke to måneder.

Hvor skal I begynde?



Placér hvert testområde i matricen. Øverst til højre er den eneste kvadrant, I skal røre de første otte uger.

SÅDAN PRIORITERES DER

Giv hvert område en karakter fra 1 til 5 på forretningsrisiko, på hvor stabil brugerfladen er, og på hvor kompliceret testdata er. Gang de to første med antallet af kørsler om året, og divider med datakompleksiteten.

Formlen er grov med vilje. Den skal skabe en samtale om, hvad der betyder noget, ikke en facitliste. Flytter rækkefølgen sig, mens tallene diskuteres, er selve diskussionen den egentlige værdi.

De fleste opdager undervejs, at et område, alle troede var vigtigt, kører to gange om året.

SKEMA TIL KORTLÆGNING

TESTOMRÅDE	KØRSLER PR. ÅR	RISIKO 1–5	UI-STABILITET 1–5	TESTDATA 1–5

DET, DER IKKE SKAL AUTOMATISERES

- ○ **Eksplorativ test.** Værdien ligger i menneskets nysgerrighed, ikke i gentagelsen.
- ○ **UX- og designvurdering.** En maskine kan se, at knappen findes, men ikke at den er forvirrende.
- ○ **Tredjepartssystemer, I ikke kontrollerer.** Testen fejler, når leverandøren ændrer noget, og I kan intet stille op. Test jeres egen side af integrationen mod en stub.
- ○ **Engangstest.** En migrering, der kører én gang, betaler aldrig automatiseringen tilbage.

NÅR INGEN KAN SIGE, HVAD DER ER RIGTIGT

Automatisering kræver, at nogen kan afgøre, hvad det **rigtige** svar er. Er selv erfarne brugere i tvivl om, hvornår resultatet er korrekt, er det ikke et automatiseringsproblem. Det er et kravproblem, og det løses i trin 2.

OGSÅ NÅR SYSTEMET BYGGES FRA BUNDEN

Når tests regenereres frem for repareres, er en brugerflade i bevægelse ikke længere en grund til at vente. Start på krav- og API-niveau, og læg UI-tests oven på, når skærmbillederne falder på plads.

Et krav, to personer tolker ens

AI kan generere testcases fra et uklart krav. Den genererer bare de forkerte. Kvaliteten af alt, der kommer efter dette trin, er bundet op på, om acceptkriterierne er entydige. Det er også den korteste vej til gevinst.

FØR

Brugeren skal kunne ansøge om tilskud og få besked om resultatet.

Hvad er "besked"? Hvornår? Hvad hvis ansøgningen er ufuldstændig? Hvad med en bruger, der allerede har en åben ansøgning? Tre testere vil skrive tre forskellige testsuiter.

EFTER

Givet en verificeret bruger uden åben ansøgning
Når alle obligatoriske felter er udfyldt og ansøgningen indsendes
Så oprettes en sag med status "Modtaget", og en kvittering sendes til brugerens registrerede e-mail inden for fem minutter

Givet en bruger med en allerede åben ansøgning
Når hun forsøger at indsende igen
Så blokeres indsendelsen, og hun henvises til den åbne sag

Givet en ansøgning med manglende obligatoriske felter
Når der indsendes
Så markeres hvert manglende felt, og intet gemmes

TJEKLISTE: ER KRAVET TESTBART?

- [ ] Er startbetingelsen beskrevet, altså hvilken bruger og hvilken tilstand?
- [ ] Er handlingen entydig, og kan den udføres uden fortolkning?
- [ ] Er det forventede resultat observerbart udefra?
- [ ] Er der en tidsgrænse, hvis timing betyder noget?
- [ ] Er mindst ét negativt scenarie beskrevet?
- [ ] Kan en person uden domæneviden afgøre, om kriteriet er opfyldt?

EKSEMPELPROMPT: KRAVANALYSE

```
Du er testanalytiker. Gennemgå kravet
nedenfor og returnér tre ting:
1) en liste over uklarheder og implicitte
antagelser, hvert punkt med det spørgsmål,
der skal stilles til forretningen
2) et omskrevet krav som Givet/Når/Så-
acceptkriterier, ét pr. scenarie
3) de negative scenarier og grænsetilfælde,
kravet ikke nævner
Opfind ikke forretningsregler. Markér alt,
du er nødt til at antage, med [ANTAGELSE].

KRAV:
<indsæt krav>
```

Den sidste linje er den vigtigste. Uden den udfylder modellen hullerne selv, og antagelserne bliver usynlige.

Et uklart krav bliver ikke bedre af at blive automatiseret. Det bliver bare gentaget hurtigere, og fejlen står nu i 100 testcases i stedet for én.

Derfor er det også her, I får mest for indsatsen: en dags arbejde med acceptkriterierne på ét område sparer typisk flere uger senere i forløbet.

Optag, hvordan systemet faktisk bruges

Testcases skrevet ud fra dokumentationen tester det system, nogen troede, de byggede. Testcases genereret fra optaget adfærd tester **det system, der står i produktion**, og de forudsætter ikke en komplet kravspecifikation.

OPSKRIFT: CAPTURE-WORKSHOP PÅ TO TIMER

0:00	**Ramme og scope.** Hvilket flow, hvilken rolle, hvilket miljø. Aftal at der optages.
0:15	**Hovedflowet, uafbrudt.** Brugeren gennemfører opgaven, som hun plejer. Ingen spørgsmål undervejs.
0:45	**Samme flow, nu med variationer.** Hvad gør du, når feltet er tomt? Når kunden ringer midt i? Når systemet svarer langsomt?
1:15	**Testdata og forudsætninger.** Hvilke data skal findes, før flowet overhovedet kan starte?
1:40	**Opsamling.** Hvad blev ikke vist? Hvad gør de andre anderledes end dig?

Deltagere: én daglig bruger, én superbruger, én tester som facilitator og én teknisk observatør. Flere end det, og optagelsen bliver en diskussion i stedet for en observation.

FIRE FALDGRUBER

Kun happy path. Den erfarne bruger viser, hvordan det går godt. Undtagelserne skal fremprovokeres, så bed eksplicit om dem.

Trin, der springes over. Superbrugeren har genveje, hun ikke tænker over at nævne. Bed hende forklare højt, mens hun arbejder.

Produktionsdata i optagelsen. Optagelser af rigtige sager indeholder persondata. Brug testdata eller anonymisér, før materialet bruges videre.

Én bruger er ikke sandheden. To personer i samme rolle arbejder sjældent ens. Optag mindst to, hvis flowet er forretningskritisk.

HVAD AI OMSÆTTER OPTAGELSEN TIL

A
Handlinger
Hvert trin med det element, der blev interageret med, og den værdi, der blev indtastet.

B
Variationer
De steder, hvor flowet forgrener sig, og hvad der udløste forgreningen.

C
Forudsætninger
Den tilstand, systemet skal være i, før flowet kan gennemføres.

D
Testscenarier
Et udkast til scenarier, inklusive de grænsetilfælde adfærden afslører.

TRE TING PÅ PLADS FØR OPTAGELSEN

- ◯ **Et miljø, der ligner produktion.** Optages der i et miljø med halvdelen af integrationerne slået fra, optager I et flow, der ikke findes.
- ◯ **Testdata, der kan bruges igen.** Brugeren skal kunne gennemføre flowet mindst to gange, uden at nogen rydder op imellem.
- ◯ **En aftale om optagelsen.** Deltagerne skal vide, hvad der optages, hvor det gemmes, og hvor længe.

HVAD DER TYPISK DUKKER OP UNDERVEJS

De fleste capture-workshops afdækker mindst ét flow, som ingen vidste blev brugt: en genvej i systemet, et Excel-ark der føres sideløbende, eller en manuel rettelse direkte i databasen. Det er ikke støj, men den reelle proces. Er den ikke testdækket, er den heller ikke beskyttet mod regression.

Notér dem særskilt. De er ofte den mest værdifulde sidegevinst ved hele øvelsen.

Sådan ser en testcase ud, der kan automatiseres

En genereret testcase er kun brugbar, hvis den er **ensartet**. Ensartetheden er det, der gør, at den kan omsættes til kode automatisk, og at den kan regenereres i morgen i stedet for at blive repareret.

TC-0142	Indsend ansøgning med gyldige data
SPORBARHED	KRAV-17 · Acceptkriterie 1
RISIKO	Høj. Blokerer sagsoprettelse
FORUDSÆTNING	Verificeret bruger uden åben ansøgning. Testdata: bruger T-101.
TRIN	1 · Log ind som T-101 2 · Åbn "Ny ansøgning" 3 · Udfyld alle obligatoriske felter med datasæt D-04 4 · Indsend
FORVENTET	Sag oprettes med status "Modtaget". Kvittering på skærmen med sagsnummer. E-mail modtaget inden for fem minutter.
OPRYDNING	Slet oprettet sag, nulstil bruger T-101.

Opdigtet eksempel. Det er felterne, ikke indholdet, der skal være ens på tværs af hele suiten.

VALIDERINGSTJEKLISTE TIL FORRETNINGEN

Kan gennemføres af en fagperson uden teknisk baggrund. Det er her, den menneskelige kontrol ligger.

- [ ] Afspejler trinnene, hvordan opgaven faktisk løses?
- [ ] Er det forventede resultat rigtigt, også i detaljen?
- [ ] Mangler der et scenarie, I ved sker i virkeligheden?
- [ ] Er risikovurderingen rimelig?
- [ ] Er der testcases, der reelt tester det samme?

EKSEMPELPROMPT: TESTCASE-GENERERING

```
Generér testcases ud fra materialet
nedenfor. Brug præcis disse felter:
id, titel, sporbarhed, risiko,
forudsætning, trin, forventet
resultat, oprydning. Ét scenarie pr.
testcase. Hvert trin er én handling i
imperativ. Forventet resultat skal
kunne observeres udefra. Tilføj ikke
forretningsregler, der ikke står i
materialet, og markér manglende
information som [MANGLER].
```

Næste side viser prompten brugt på et konkret krav, med modellens svar og de rettelser, der fulgte.

TRE TEGN PÅ EN TESTCASE, DER SKAL RETUR

- ○ **Det forventede resultat er en gentagelse af handlingen.** "Ansøgningen indsendes" er ikke et resultat, men trinnet skrevet igen.
- ○ **Forudsætningen mangler.** Uden defineret starttilstand kan testen ikke køre to gange i træk med samme resultat.
- ○ **Flere scenarier i én testcase.** Fejler den, ved ingen hvad der gik galt.

HVOR LANG TID TAGER VALIDERINGEN?

Regn med 15–25 testcases i timen pr. fagperson, når formatet er ensartet. Det er den post, folk glemmer at planlægge, og den, der afgør, om trin fem kan begynde til tiden. Book den samtidig med capture-workshoppen.

Ét acceptkriterie hele vejen igennem

Prompten på side 5 kan enhver kopiere. Værdien ligger i det, der sker bagefter, når svaret skal læses igennem og rettes. Herunder er acceptkriterie to fra side 3 kørt gennem prompten, med modellens svar og de rettelser, en tester lavede.

DET, DER GÅR IND

Givet en bruger med en allerede åben ansøgning, **når** hun forsøger at indsende igen, **så** blokeres indsendelsen, og hun henvises til den åbne sag.

MODELLENS SVAR · 1 AF 2

TC-0143 Indsendelse blokeres ved åben ansøgning

FORUDSÆTNING	Bruger T-102 med én ansøgning i status "Modtaget".
TRIN	1 · Log ind som T-102 2 · Åbn "Ny ansøgning" 3 · Udfyld obligatoriske felter 4 · Indsend
FORVENTET	Indsendelsen gennemføres ikke, og brugeren henvises til den åbne sag.

MODELLENS SVAR · 2 AF 2

TC-0144 Bruger får besked om afvist indsendelse 3

FORUDSÆTNING	Bruger med åben ansøgning.
TRIN	1 · Log ind 2 · Indsend ny ansøgning
FORVENTET	2 Der vises en rød fejlbesked øverst på siden, og 1 brugeren modtager en e-mail om afvisningen.

DET, TESTEREN RETTEDE, OG HVORFOR

1. **Opfundet forretningsregel.** Kravet siger intet om en e-mail. Modellen udfyldte hullet selv, fordi det virkede rimeligt. Sætningen blev fjernet. Ønsker forretningen faktisk en kvitteringsmail ved afvisning, er det et nyt krav og ikke noget, en testcase må opfinde.
2. **Resultatet beskriver skærmen, ikke reglen.** "En rød fejlbesked øverst på siden" knækker ved næste designændring, og den siger intet om, hvorvidt forretningsreglen holder. Omskrevet til "indsendelsen gennemføres ikke".
3. **Samme scenarie to gange.** TC-0144 tester det, TC-0143 allerede dækker. Den blev slettet. To testcases på ét kriterium betyder næsten altid, at den ene er overflødig.

HVAD EKSEMPLET VISER

Prompten leverer strukturen og hovedparten af indholdet. Rettelserne er arbejdet, og de tre typer går igen fra testcase til testcase: opfundne regler, resultater der beskriver skærmen i stedet for reglen, og scenarier der overlapper. Ved man det på forhånd, kan valideringen køres som en tjekliste i stedet for som en diskussion.

Opdigtet eksempel. Modellens svar er konstrueret, så det rummer de tre fejltyper, vi oftest ser. Det er ikke et udtræk fra en konkret kørsel.

Fra godkendt testcase til kørende test

Det er her, mange forløb går i stå. En godkendt testcase er ikke en test, der kører, og omsætningen er ikke én knap. Den består af fire trin, og det meste af arbejdet ligger i ét af dem.

TRIN	HVAD DER SKER	HVOR MEGET GENERERES
Normalisering	Testcasens felter mappes til et fast, maskinlæsbart skema.	Fuldt automatisk, så længe formatet fra side 5 er overholdt.
Handlinger	Hvert trin bindes til en genbrugelig handling i et bibliotek, for eksempel "log ind som <bruger>".	Automatisk, når handlingen findes. Første gang skal den skrives. Det er her arbejdet ligger.
Elementer	Handlingerne bindes til konkrete felter og knapper i brugerfladen.	Automatisk med test-id'er. Uden dem er det håndarbejde hver eneste gang.
Kontroller	Det forventede resultat oversættes til noget, testen kan måle.	Delvist. Uklare eller sammensatte resultater kræver et menneske.

HVEM EJER HVAD

- **Testeren ejer testcasen.** Indhold, risiko og sporbarhed til kravet.
- **Automatiseringsudvikleren ejer biblioteket og skemaet.** Handlingerne og det format, alting mappes til.
- **Udviklingsteamet ejer test-id'erne.** Uden dem falder trin tre fra hinanden.

FEM KRAV, TESTCASEN SKAL OPFYLDE

- [ ] Ét scenarie, ikke flere i samme testcase
- [ ] En forudsætning, der kan genskabes hver gang
- [ ] Trin, der hver især er én handling
- [ ] Et resultat, der kan observeres udefra
- [ ] Navngivne testdata frem for "en gyldig bruger"

Det første område koster mest. Hovedparten af tiden går med at bygge handlingsbiblioteket op, og det er arbejde, I kun laver én gang. På område nummer to og tre findes de fleste handlinger allerede, og omsætningen bliver i praksis en generering med en gennemlæsning. Derfor skal det første område være afgrænset nok til, at I kommer i mål.

SÅDAN SER EN DÅRLIGT VEDLIGEHOLDT SUITE UD EFTER TRE MÅNEDER

SYMPTOM	ÅRSAG, OG HVAD DER FORHINDRER DET
Tests fejler, og udviklerne siger "sådan skal det slet ikke virke"	En opfundet forretningsregel slap gennem valideringen. Kræv sporbarhed til et krav på hver eneste testcase, og afvis dem uden.
Én UI-ændring vælter 30 tests	De forventede resultater beskriver skærmen i stedet for reglen. Omskriv dem, så de kan observeres uden at kende designet.
Suiten kan ikke længere regenereres, og folk retter i scripts igen	Felter er tilføjet ad hoc, og ingen ejer skemaet. Versionsstyr skemaet, og giv det én ejer.
Tests fejler om mandagen	De reserverede datasæt er blevet brugt manuelt i løbet af ugen. Skift til at oprette og rydde op i testen selv.

Hold testene stabile, når de først kører

Det afgørende skift er at holde op med at reparere tests og begynde at regenerere dem. Det kræver tre ting på plads: **stabile selectors, testdata I selv kontrollerer, og miljøer der opfører sig ens hver gang.**

SELECTOR-STRATEGI, PRIORITERET RÆKKEFØLGE

01 Dedikeret test-id på elementet
Kræver en aftale med udviklingsteamet, men er den eneste selector, der overlever et redesign.

02 Rolle og tilgængeligt navn
Knappen "Indsend", feltet med label "CPR-nummer". Gratis sidegevinst: tests fanger tilgængelighedsfejl.

03 Synlig tekst
Virker, indtil teksten ændres eller sproget skiftes.

04 CSS- eller XPath-sti
Elementet findes ud fra, hvor det ligger i sidens struktur: "det tredje inputfelt i den anden formular". Indsætter nogen et felt ovenover, peger stien et andet sted hen, og testen fejler, selvom funktionaliteten er uændret. Sidste udvej.

TESTDATA: TRE MØNSTRE

Opret og ryd op. Testen skaber selv det, den skal bruge, og fjerner det igen. Mest robust, kræver API-adgang.

Reserverede datasæt. Faste brugere og sager, der aldrig røres manuelt. Nemt at starte med, skrøbeligt over tid.

Anonymiseret produktionsudtræk. Realistisk dækning, men kræver en dokumenteret proces, og aldrig rå persondata i testmiljøet.

Hvad kører hvornår

Ved hvert commit	Hver nat	Før release
Unit- og API-tests samt et smoke-udvalg af UI-tests på det ændrede område.	Hele den automatiserede regressionssuite på et stabilt miljø med friske testdata.	Fuld regression, performancetest og de scenarier, der kræver integration til eksterne systemer.
Mål: under 10 minutter	**Mål: færdig før arbejdsdagens start**	**Mål: samme dag**

Tiderne er mål, ikke løfter. Overskrider commit-kørslen ti minutter, holder udviklerne op med at vente på den, og så er værdien væk.

REGENERÉR I STEDET FOR AT REPARERE

Når en test fejler efter en UI-ændring, er spørgsmålet ikke "hvordan retter jeg scriptet?", men "er testcasen stadig rigtig?". Er den det, genereres scriptet på ny fra den uændrede testcase. Er den ikke, opdateres testcasen, og alt nedenunder følger med.

FIRE TING, DER SKAL AFTALES MED UDVIKLING

- ◯ **Test-id'er er en del af definition of done.** Ikke noget, test beder om bagefter.
- ◯ **Miljøet er tilgængeligt, når pipelinen kører.** Et ustabilt testmiljø koster mere end manglende automatisering.
- ◯ **En fejlet test stopper pipelinen.** Kan man deploye, selvom testene fejler, bliver det hurtigt normalt at gøre det.
- ◯ **Ustabile tests fjernes samme uge.** En test, der fejler tilfældigt, ødelægger tilliden til hele suiten hurtigere end ti manglende tests.

Fejl, der lander hos den rigtige med det samme

Fejlrapportering er ofte den skjulte flaskehals. Når resultater samles automatisk på tværs af kørsler, kan gentagelser slås sammen, og defects kan prioriteres efter forretningsmæssig konsekvens frem for efter, hvem der råbte højest.

01
Saml
Fejl fra alle kørsler samles ét sted med log, skærmbillede og det trin, der fejlede.

02
Grupper
Samme underliggende fejl på tværs af 15 tests bliver til én defect, ikke 15.

03
Prioritér
Efter risikoen på den testcase, der fejlede, og hvor mange flows den blokerer.

04
Rapportér
Udkast til defekten skrives automatisk, med reproduktionstrin fra testcasen.

05
Godkend
En tester læser defekten igennem, retter den til og sender den videre til udvikling.

BASELINE INDEN OPSTART

MÅLEPUNKT	AKTUELT	MÅL, 6 MDR.
Andel af testscope, der er automatiseret		
Timer manuel regressionstest pr. release		
Tid fra fejl konstateres til færdig defektrapport		
Tid til at bygge testsuite for et nyt område		
Antal fejl fundet efter produktionssætning		
Timer brugt på at vedligeholde eksisterende tests		

Uden de seks tal målt *før* start bliver gevinsten en fornemmelse. Med dem har I en business case, der kan forsvares i en styregruppe.

HVORNÅR VIRKER DET?

Udviklerne venter på testen. Ikke fordi de skal, men fordi svaret er hurtigt nok til at være brugbart.

Ingen diskuterer, om en fejlet test er en reel fejl. Falske fejl er under kontrol, så når testene fejler, bliver det taget alvorligt.

Dækningen vokser, mens vedligeholdet står stille. Det er selve testen på, om regenerering faktisk er lykkedes.

RAPPORTÉR PÅ RISIKO, IKKE PÅ ANTAL

"412 testcases kørt, 9 fejlede" siger ingenting til en styregruppe. "Alle kritiske betalingsflows er testet og bestået; to fejl på rapportudtræk, begge med workaround" er en beslutning, de kan handle på.

Byg rapporteringen på testcasens risikofelt fra side 5. Det er derfor, feltet skal udfyldes, og ikke af ordenssans.

DEN KLASSISKE FÆLDE

Dækningsprocent bliver et mål i sig selv. 80 procent dækning på de forkerte områder er mindre værd end 30 procent på de rigtige. Mål på, hvor stor en del af **den forretningskritiske funktionalitet** der er dækket, ikke på hvor mange linjer kode testene rører.

Vælg efter jeres landskab, ikke efter markedet

Der findes ikke ét bedste værktøj, kun ét der passer til jeres landskab og til dem, der skal bruge det. De syv spørgsmål herunder afgør valget bedre end nogen sammenligningsmatrix.

01 Dækker det hele jeres stak: web, mobil, API og de legacy-systemer, ingen vil tale om?

02 Kan testene versionsstyres som kode, eller lever de i en lukket database?

03 Kan det køre i jeres pipeline uden en manuel handling?

04 Hvor svært bliver det at skifte værktøj senere, og kan testene flyttes med?

05 Hvem i teamet kan vedligeholde det om et år, hvis nøglepersonen er væk?

06 Hvor behandles jeres kode og testdata, når AI er involveret, og forlader noget af det organisationen?

07 Kan I få et revisionsspor ud: hvad blev testet, hvornår, med hvilket resultat, uden at skulle bygge det selv?

ARBEJDSDELINGEN MELLEM AI OG MENNESKER

KAN OVERLADES TIL AI	KRÆVER ET MENNESKE
Generere udkast til testcases fra krav og optagelser	Godkende, at en testcase afspejler den rigtige forretningsregel
Regenerere scripts, når en testcase er uændret	Beslutte, at en testcase skal ændres eller fjernes
Gruppere og foreslå prioritering af defects	Sætte den endelige sværhedsgrad på en defect i produktion
Køre test og rapportere resultater	Godkende en release på baggrund af testresultatet
Foreslå nye scenarier og grænsetilfælde	Afgøre, om et manglende scenarie er en risiko, I accepterer

TRE MÅDER AT BYGGE TESTENE PÅ

KATEGORI	SÅDAN SKRIVES TESTENE	SÅDAN GEMMES DE, OG HVAD DET BINDER JER TIL	PASSER NÅR
Kode	Af nogen, der programmerer, i samme sprog og arkiv som produktet.	Som kode i jeres eget repo. Binder jer ikke til nogen. Koden er jeres og kan tages med.	I har udviklere i eller tæt på testteamet, og stakken er moderne web og API.
Visuelt værktøj	Ved at optage et flow og rette det i en grafisk editor. Ingen kode.	I værktøjets eget format. Binder jer til leverandøren, medmindre der kan eksporteres til noget, I selv kan læse.	Testerne koder ikke, og de flows, I starter med, ligger stabilt.
Platform	I platformens egen testmanager, typisk af både testere og forretning.	I platformens database sammen med planer og resultater. Binder jer til leverandøren, som regel i flere år.	I har tunge standardsystemer eller legacy, og der er formelle krav til revisionsspor.

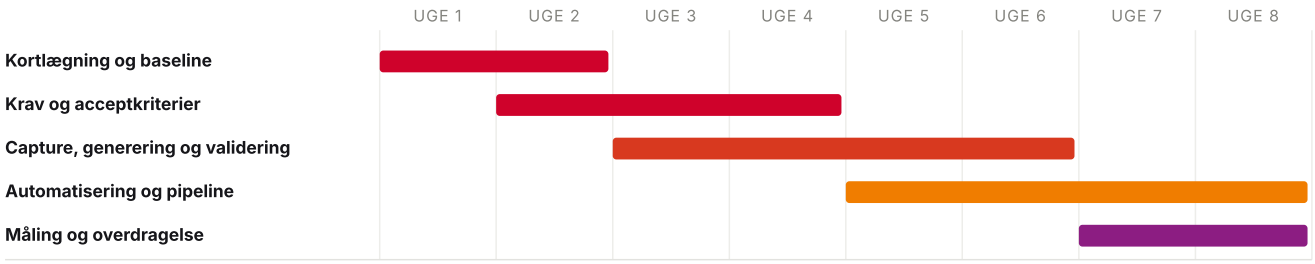
De fleste ender med to af dem, ikke én.

HVOR AGENTERNE KOMMER IND

Et agentbaseret lag er ikke et alternativ til de tre ovenfor. Det lægger sig oven på det, I har valgt, og forudsætter, at sporbarheden er på plads. Tag stilling til det, når de syv spørgsmål er besvaret.

Otte uger til den første testsuite er i drift

Planen herunder forudsætter ét afgrænset område og et team, der er dedikeret til opgaven i hele perioden. Den er stram med vilje, fordi momentum er det, der afgør, om indsatsen overlever.



STATUS VED UDGANGEN AF UGE 2

- ○ **Området er valgt** og afgrænset, og nogen ejer det.
- ○ **Baseline er målt** på de seks punkter fra side 9.
- ○ **Testmiljøet svarer**, og I ved, hvem der genstarter det.

FALDGRUBER OMKRING UGE 5

- ○ **Testdata løber tør**, fordi ingen ryddede op. Løs det i uge 3, ikke uge 5.
- ○ **Selectors knækker**, fordi test-id'erne aldrig blev en del af definition of done.

EFTER OTTE UGER

- ○ **En testsuite i drift** på ét område, der kører i jeres egen pipeline.
- ○ **Målt effekt** mod den baseline, I tog i uge 2.
- ○ **En arbejdsgang**, resten af organisationen kan kopiere.

NÅR DET FØRSTE OMRÅDE KØRER

Område nummer to tager typisk under halvdelen af tiden, fordi formatet, testdata-mønstrene og pipelinen allerede findes. Det er der, gevinsten begynder at skalere, og også der, testledelsen skal være på plads, så dækningen kan gøres op på tværs.

Signe Rohde
Senior Manager
Head of Technology Management
Sopra Steria Danmark

signe.rohde@soprasteria.com
+45 30 61 42 89

Test Automation: hurtigere kvalitetssikring med AI →
Agentisk QA: næste niveau af test →
Test Management: strategisk testledelse og kvalitetssikring →

Start med et proof of concept på ét område

Et afgrænset forløb på et udvalgt flow, hvor I ser kvaliteten af de genererede testcases i jeres eget system, inden I skalerer indsatsen.

soprasteria.dk