

Agile Testing

TestDrivenDevelopment
Af Poul Staal Vinje

Certified Scrum Master
Certified Scrum Practitioner

poul.vinje@gmail.com
www.agile-metoder.dk

Indhold

Agile Testing.....	1
1. Indledning	2
2. Eksempel.....	2
3. TDD forløbet.....	3
4. Resultatet af forløbet.....	3
5. Koden udvikles	4
6. Rytmen	5
7. Forskellen.....	5
8. Processen i TDD forløbet.....	6
9. Fordele ved TDD	6
10. Afslutning.....	7
Mine bøger	7

1. Indledning

Reaktionerne på artiklen om Agile testing i sidste nyhedsbrev var meget positive. Vi fortsætter derfor med emnet. Denne gang ser vi nærmere på TestDrivenDevelopment (TDD) der er et nøgleelement i Agile Testing.

I et agilt udviklingsforløb vil der være brug for mange små TDD forløb. Flertallet af dem skriver kun testcases til en enkelt Userstory, eller til en enkelt klasse, ad gangen. Det er et fornuftigt omfang.

2. Eksempel

Vi kan bruge dette simple eksempel på en Userstory;

Userstory på valg af gebyrregel på smålån.

Beregningen bruger en løbetid og et beløb som grundlag for valg af regel.

Løbetiden kan være 1-5 år.

Beløbet kan maksimalt være 300.000,- kr.

Lån på 1-3 år bruger regel 1 eller 2.

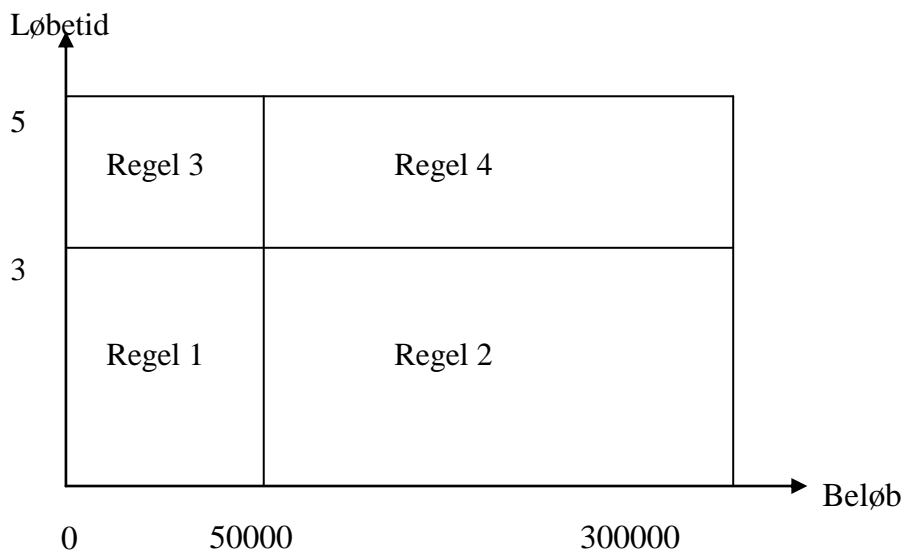
Lån på 4 eller 5 år bruger regel 3 eller 4.

Lån op til 50.000 bruger regel 1 eller 3

Lån mellem 50-300.000 bruger regel 2 eller 4.

Værdierne

Sammenhængen mellem værdierne kan afbildes således



Tegningen viser det gyldige værdiområde, gyldige ækvivalensklasser og grænseværdierne.

3. TDD forløbet

Generelt

Der arbejdes med Pair Testing, dvs. testcases udarbejdes i et samarbejde med en kollega.

Et forløb varer op til 2 timer. Det kan være mindre når opgaven ikke er så stor, men det brydes op i mindre forløb, hvis det bliver meget mere end et par timer.

En iteration indeholder så mange forløb der er nødvendige, med skiftende besætninger af udviklere, testere, og brugere.

Resultatet er testcases der bruges som grundlag til at skrive koden.

Afklaring

Der er som regel brug for afklaringer, og i dette tilfælde kunne det fx være:

Er vi sikker på at det kun kan være heltal?

Hvis ja, hvad skal komponenten gøre hvis der alligevel kommer 1,5 i løbetid?

Er 0 et validt beløb? Er 300000?

Skal komponenten validere løbetid og beløb. Og i så tilfælde, hvordan skal ugyldige behandles.

Selvom man kan gætte sig grænsetilfældene, så skal de afklares. I dette tilfælde beløb på 50000.

Overvejelser

Hvor mange af de gyldige ækvivalensklasser skal der skrives testcases til? Der er fire regler der skal dækkes, men man kan vælge at skrive testcases til alle fem år.

Hvor mange af de ugyldige ækvivalensklasser skal der skrives testcases til? Der er 12 ugyldige, hvis dette er almindelig forretningskritisk kode, men langt flere potentielle der skal afprøves hvis det er livskritisk kode. Altså hvis det kan skade mennesker eller andet vi under ingen omstændigheder ønsker at miste. Se mere om dette i min bog.

Men emnet lige nu er TDD, og vi afgør følgende i vores forløb, ved at konsultere en bruger eller en repræsentant for denne:

Beløb må ikke være 0.

Et beløb på 50000 hører til det lave interval. 50001 hører til det høje.

Et beløb på 300000 er gyldigt, 300001 er ikke.

År skal angives i heltal

År 3 hører til det lave interval

År 4 (faktisk 3 år og 1 dag, hvis vi brugte decimaler) hører til det høje interval

År 5 er også gyldig

4. Resultatet af forløbet

Følgende testcases er et muligt resultat af testcase planlægningen:

Nr.	Løbetid	Beløb	Forventet resultat
-----	---------	-------	--------------------

1	2	10000	Passerer, returnerer regel 1
2	2	60000	Passerer, returnerer regel 2
3	4	5000	Passerer, returnerer regel 3
4	4	160000	Passerer, returnerer regel 4
5	6	10000	Afvises, galt år
6	6	170000	Afvises, galt år
7	2	320000	Afvises, galt beløb
8	4	320000	Afvises, galt beløb
9	6	320000	Afvises, galt år og beløb
10	2	-10000	Afvises, galt beløb
11	4	-10000	Afvises, galt beløb
12	6	-10000	Afvises, galt år og beløb
13	-1	-10000	Afvises, galt år og beløb
14	-1	5000	Afvises, galt år
15	-1	150000	Afvises, galt år
16	-1	320000	Afvises, galt år og beløb
17	0	5000	Afvises, galt år
18	1	0	Afvises, galt beløb
19	0	0	Afvises, galt år og beløb
20	1	50000	Passerer, returnerer regel 1
21	1	50001	Passerer, returnerer regel 2
22	1	300000	Passerer, returnerer regel 2
23	1	300001	Afvises, galt beløb
24	1,5	20000	Galt år
25	1	20000,5	Galt beløb
26	A	1000	Galt år
27		1000	Galt år

Det er op til testerne om der ønskes flere testcases i det gyldige værdiområde, således at alle løbetider dækkes.

Det er op til testerne om der ønskes flere testcases af typen i nr. 26 og 27.

Det er op til testerne at lade et grænsetilfælde dække en ækvivalensklasse, fx at nr. 20 dækker nr. 1, at nr. 21 eller 22 dækker nr. 2, og at nr. 23 dækker nr. 7. Selv foretrækker jeg at have begge sæt.

Man kan vælge færre eller flere testcases end jeg har gjort her. Men de valgte testcases er nu de detaljerede krav til systemet.

5. Koden udvikles

Når testcases foreligger, kan udviklerne skrive koden på grundlag af dem. I Agile forløb vil det være i form af Pair Programming, og der vil ske en automatisering af testcases før udviklingen begynder.

Målet er klart for udviklerne, nemlig at koden er færdig når alle testcases opfører sig som forventet. Hvad enten det er, at de passerer med det forventede resultat, eller de afvises, ligeledes med det forventede resultat.

6. Rytmen

Rytmen mellem at formulere testcases og skrive den tilsvarende kode kan varieres. Fowler og Beck foretrækker hyppige skift, således at der formuleres en enkelt testcase, og der derefter skrives så lidt kode som der skal til for at få testcasen til at passere.

Fowler Rhythm:

Small test (Skriv en enkelt ny testcase)

Run test and fail with red bar

Small change (Mindst mulig ny kode, netop nok til at få testen til passere korrekt)

Run test and pass with green bar

Refactoring (forbedring af kode)

Run test – pass with green bar (Regressionstest)

Robert C. Martin (aka Uncle Bob) nøjes med at påpege at det kan være det rigtige at gøre. Yderpunkterne i TDD er at skrive testcases til en hel Userstory, og derefter koden, eller at vælge at skrive det mindst mulige test der skal til for at få applikationen til at ”Breake”. Hvorefter der skrives så meget kode der skal til for at få applikationen til at ”Passe”

I det første yderpunkt kan testere i par formulere testcases, i de sidste skal testere og udviklere, eller to udviklere, arbejde fysisk sammen.

De tre tommelfingerregler

Skriv kun kode til en test der fejler.

Skriv kun en enkelt ny testcase ad gangen.

Skriv kun så meget kode at det får den fejlende test til at passere.

7. Forskellen

Hvad er forskellen i en TDD drevet proces, i forhold til en proces hvor koden skrives på grundlag af Userstorien, og der derefter skrives testcases?

1. Processen drives af kravene til grænsefladen. Det er fra ’The Caller Perspective’.
2. Der bliver skrevet flere testcases i det ugyldige område.
3. Der bliver skrevet flere testcases på et tidligere tidspunkt i forløbet
4. Det styrker unittesten. Og opfordrer til at testcases bliver fastholdt og automatiseret.

Udviklere foretrækker nogle gange en Whitebox proces, hvor testcases defineres i forhold til koden, og deres kendskab til den. Risikoen er at koden er fejlfri fra udviklernes synspunkt, men ikke fra brugernes.

Nogle af de testcases der ikke skrives når det er udviklere der gør det efter kodning, vil ganske vist blive skrevet og udført senere. For eksempel i bruger- eller alfatest. Men der viser erfaringen at ikke alle bliver skrevet ned. De bliver ikke fastholdt. Automatisering bliver dermed mindre end mulig.

TDD er en Blackbox test. I sagens natur, koden findes jo ikke. Udviklernes er en Whitebox test. Man kan argumentere for at have begge dele. De giver hver deres form for dækning (coverage).

Martin Fowler

Martin Fowler var, sammen med Kent Beck, en af pionererne inden for TDD.

Martin Fowler havde disse mål med TestFirst:

1. Selvttestende kode, automatiseret
2. Det solide programmeringsgrundlag. Tænke interface til en komponent før implementering. Tænke brugen før udvikling af koden. Kode er implementering af detaljerede krav til interfacet.

8. Processen i TDD forløbet

Processen i disse to timers TDD forløb følger en af tre hovedformer:

1. Ustruktureret, kreativ, udforskende

Der er ingen særlig forberedelse fordi der netop er behov for at tænke 'sidelæns' og kreativt.

2. Struktureret

Dette er den almindeligste form. Der bruges et grundlag som processen struktureres henover. Det kan være Userstories, eller Use Case scenarier, med en eventuel afgrænsning i forhold til 'Happy Path', 'Sad Path', 'Bad Path', eller Use Case scenarier på et bestemt niveau, enten 'Cloud Level', 'Sea Level' eller 'Fish Level'.

Eller det strukturerede kan være i form af metoderne pr. klasse.

3. Rigid

I denne hovedform er der detaljerede modeller, for eksempel sekvensdiagrammer, eller andre UML modeller.

En erfaren tester kan alle tre hovedformer! Veksler gerne! Context-driven!

Se mere om Context-driven på www.context-driven-testing.com

9. Fordele ved TDD

Mine egne erfaringer, specielt i Scrum forløb, peger på disse fordele:

- 1) Færre fejl i det samlede forløb. Færre der begås, færre der skal findes, færre der skal rettes.

2) Bedre testmateriale, både til brug i det aktuelle testforløb og fremtidige. Automatiseret regressionstest kan vokse konstant.

3) Forbedring af det interne design. Coupling reduceres og Cohesion øges som resultat af et bedre Interface Design. Det bliver fra et 'Caller Perspective', og ikke en implementerings synsvinkel. Mikrodesignet bliver bedre, herunder navngivning af klasser, navngivning af metoder, klasseansvaret (Responsibility) og design af metoders parametre. TDD, rettet mod de komplekse dele af en applikation, resulterer i et simplere design.

4) Videreudviklingen lettes. TDD forbedrer mulighederne for Refactoring fordi resultatet af den kan checkes. Det øger sikkerheden og tiltroen til videreudvikling. Både i det aktuelle testforløb og i den langsigtede videreudvikling.

5) Forbedring af grundlaget, for eksempel Userstori eller klassebeskrivelsen. Det er en af de gratis sidegevinster, at de forbedres under vejs i TDD.

10. Afslutning

Næste artikel

Automatisering

Fordelene ved TDD forstærkes af automatisering af unittesten. Hvis der er fortsat lige stor interesse for Agile Testing, vil jeg fortsætte med en artikel der viser automatiseringen af de testcases der er resultatet her.

Relaterede emner

AcceptanceTestDrivenDesign

Denne artikel er om TDD. I Agile forløb kan man supplere med AcceptanceTestDrivenDesign (ATDD).

XP og Scrum

TDD skal foregå i en kontekst. Det kan være i en klassisk kontekst, eller det kan være i en Agile. for eksempel XP og/eller Scrum.

Iterativ udvikling

Iterativ udvikling kan omfatte videreudvikling af krav. Det vil sige at det ikke kun er hele nye funktioner der tilføjes i en iteration. Det kan omfatte videreudvikling af noget kode der allerede er leveret. I det tilfælde er det tæt på at være en forudsætning at have omfattende testcases, til brug for systematisk regressionstest.

Mine bøger

<http://www.vrpartners.dk/Litteratur.htm>

Reaktioner

Mail mig gerne med kommentarer og erfaringer.