Appendix

Frameworks and Tools

Testing frameworks for JS involved in this thesis:

- Jasmine[1]
- JsTestDriver[2]
- Sinon.JS[3]
- qUnit[4]
- Karma[5]
- Mocha[6]
- Buster.JS[7]

A full evaluation of available JS frameworks was outside the scope of the thesis (see the Scope and Delimitations section of the report).

Testing frameworks for SML:

- QCheck[8]
- smlunit[9] [10] [11]
- tests.sml[12]
- sml-testing[13]
- SMLTest[14]
- smell[15]
- smell-spec[16]
- stupid[17]

[1] http://pivotal.github.com/jasmine/
[2] https://code.google.com/p/js-test-driver/
[3] http://sinonjs.org/
[4] http://qunitjs.com/
[5] http://karma-runner.github.io/
[6] http://visionmedia.github.com/mocha/
[7] http://docs.busterjs.org/
[8] https://github.com/league/qcheck
[9] https://github.com/dellsystem/smlunit
[10] https://github.com/smlsharp/SMLUnit
[11] https://github.com/CyberLight/smlunit
[12] https://github.com/spacemanaki/tests.sml
[13] https://github.com/kvalle/sml-testing
[14] https://github.com/Yankovsky/SMLTest
[15] https://github.com/Morriphi/smell
[16] https://github.com/davidpdrsn/smell-spec
[17] https://github.com/teebrz/stupid

QCheck is the most popular SML testing framework, and there are three different ones that are all called smlunit. The smlsharp version of SMLUnit is tailored for a non-standard SML distribution, smell-spec is the only one with BDD syntax and stupid is unique in offering watch capabilities so that the tests are run when files change. All except QCheck and the dellsystem version of smlunit are less than one year old (2013) and none are under particularly active development.

Transcript of Interview with Johannes Edelstam

1. Vad är dina erfarenheter av hur folk testar sitt JavaScript?

För 1,5-2 år sen, höll jag i en träff om testning av JavaScript. Jag hade handuppräckning och frågade "Hur många testar sitt JavaScript?". Då var det bara tre personer som hade gjort det överhuvudtaget. Nu, nästan två år senare, är i princip alla händer i luften. Trenden har gått från "Vad konstig du är som testar ditt JavaScript" till "Vad kör du för tools" på bara ett och ett halvt år.

2. Vad tror du har lett till det?

Verktyg, dels att de har blivit bättre och att det finns fler exempel på hur man gör. Folk var inte emot testning eller test-driven utveckling, många Ruby-utvecklare hade redan positiva erfarenheter av att testa, men hade inställningen att JavaScript var för inriktat på gränssnitt för att det ska gå att testa det.

3. Det tycker jag att man fortfarande kan se lite grann.

Samtidigt är det konstigt att säga så, för i ett Ruby-projekt testas saker och ting rigoröst, trots att det är en massa gränssnitt som du testar igenom. Det handlar bara om att det är dålig tooling, inte att det inte behövs göras.

En annan aspekt kan vara att folk helt enkelt skriver kass kod. Om man i ett läge inte vet hur man ska testa en viss sak så kan det vara så att det inte går för att koden inte har skrivits på ett sätt som gör den testbar.

4. Jag funderar på om att använda ramverk som strukturerar upp koden är en lösning för att faktiskt kunna testa kod bättre.

Vad tänker du på för ramverk?

5. Det jag hittills hunnit skaffa mig erfarenhet av är AngularJS, som har vyer och controllers. Controllers är enkla att testa.

Precis.

6. Sen det här spelet som jag tänkte köra en workshop om framöver, det kanske jag inte har berättat något om. Jag tänkte köra med ett arkadspel, Asteroids, och TDD:a lite på det.

Schysst.

7. Det är inte uppbyggt kring något ramverk, men är uppdelat i klasser och känns i allmänhet som rätt så schysst kod. Då går det att testa. Det fanns inga tester, så det

har jag lagt till nu i efterhand (vilket har varit lärorikt). Jag tror att om man inte är såpass duktig som han som skrev det var, och faktiskt är så disciplinerad att man ser till att dela upp så att det bildas klasser med metoder som går att testa, så blir det kaos.

Verkligen.

8. Det där med Ruby vs JavaScript har jag sett innan också, jag inleder min rapport med det just nu, ett liknande citat som det du just sa, bara att det var från när det fortfarande bara var tre händer.

Det är vanligt och gäller överallt, folk gillar att hitta ursäkter till varför man inte ska testa grejer. Det är en liknande paradox som den att man tror att man blir så effektiv när man multitaskar, man luras att tro att man begränsas av testerna. Det som folk ofta menar när de säger att de begränsas av testerna är att de i själva verket inte riktigt vet vart de ska. Om man inte vet vad det är man ska bygga och hur allting ska fungera så tar det väldigt lång tid om man ska test-driva någon form av lekstuga. Det blir rätt stökigt, så där handlar det om att hitta bra egna principer för hur man tar sig fram till den punkt då man vet vad något ska bli. När man väl har en klar bild av vart man vill så är det ofta ganska lätt att börja skriva tester och det väcker bra tankar som man inte har stött på när man tagit sig fram till sin målbild.

Det som ofta blir när man skriver tester är att man lägger mycket av sin tid och sitt fokus i att skriva testerna på ett bra sätt och känslan blir att man inte skriver lika mycket av produktionskoden. Vilket är en bra sak! För då ägnar man en massa tid åt att fundera på problemet och hur man ska boxa in det. Det leder till eleganta och små lösningar, för att man lyckats skapa sig en uppfattning om vad problemet är.

Framförallt blir man bättre på att göra begränsningar, om man kodar utan tester så är det lätt att man kommer på att en viss funktion skulle kunna göra flera saker, medan om man skriver tester för koden så är det enklare att inse att en extra parameter kan innebära att det behövs dubbelt så många tester. Just i JavaScript är det ett vanligt misstag att frångå single purpose-principen och det kan man alltså undvika med hjälp av tester.

9. Ja, i JavaScript är det ju enkelt att skriva funktioner som tar olika antal parametrar och bara kör på oavsett hur de anropas.

Jag skulle säga att jag testar betydligt mer av just logik-kod än till exempel grafer. Det är mest för att de ändras snabbt och risken för fel är lägre. Det är värre att ha ett konsekvent logik-fel än om en stapel blir lite för kort.

10. Du tänker på Tink?

Ja, precis. Det är samma mönster som när jag testdriver Ruby-kod. Man driver beteende, inte utseende. Sen tror jag också att det har att göra med att det är dåliga tools, det är omständigt att skriva hållbara och värdefulla tester för att en svg har rätt parametrar. I situationer där något kan vara rätt ändå, fast det inte är precis som man skrev, så uppstår bräckliga tester om man inte har tagit hänsyn till detta när man skrivit sina tester. Dåliga tester kännetecknas av att man behöver ta bort dem för att

kunna komma vidare i utvecklingen. Det är en svår gränsdragning var man ska lägga sin energi.

11. Jag läste ett inlägg på twitter alldeles nyss av Kent Beck, om att han hade spenderat 6 timmar på att skriva ett test. Han hade researchat och konstaterat att det var ett komplicerat domän och någon påpekade att detta var anledningen till att folk inte skriver tester, att andra hade gett upp här. Kent Beck kontrade med att om han inte vet hur han ska skriva testet så vet han verkligen inte hur han ska skriva koden.

Exakt! Det där är jätteintressant, jag tror att folk hänger upp sig för mycket på att det är att skriva testerna som är svårt. Om man sätter sig och skriver testerna först så är det där man springer på problemet. Någon annan hade säkert lagt samma sex timmar fast på att stirra på produktionskod.

Sen kände jag att det har lite med vana att göra också. Nu när jag skrivit tester till det här spelet på senare tid, så går det mycket fortare att se vad som går att testa och hur testkoden kan skrivas. I början blev jag frustrerad över de nybörjarmisstag jag gjorde.

Det är en viss startsträcka så att det tar längre tid att komma igång, vilket avskräcker folk. Det där har blivit mycket bättre genom verktyg som Yeoman och Brunch, där man får projekt med tester och allting uppsatta, det är bara att börja skriva ungefär som i Ruby on rails. Sådant gör att folk blir mer vänligt inställda till tester.

I början när jag kollade på det här så var det väldigt experimentellt att överhuvudtaget köra tester i terminalen. Innan PhantomJS var bra så fanns det headless drivers som var svåra att använda.

12. Hur bra är PhantomJS nu?

Nu är det bra, jag kör testerna i princip jämt genom PhantomJS. Det kunde fortfarande vara bättre, men för det mesta så är det nog testramverken som inte använder det på rätt sätt. Vissa test failures måste man fortfarande dra upp i browsern för att få bra felrapportering. Vi kör våra tester i PhantomJS i byggen på Jenkins, så det är en del av byggflödet.

13. Det jag har gjort nu är att jag har använt JsTestDriver, det känns som att det kan vara knepigt att få in det i Jenkins.

Det är det säkert, men där handlar det om att man måste skilja på tester som testar ren logik där värdet av att testa i en massa olika webbläsare kanske inte är sådär gigantiskt stort.

14. Nej, mina tester skulle jag kunna köra headless.

Exakt, och då kan man lika gärna köra dem i något abstrakt som till exempel phantom, för du får samma resultat och det är mycket smidigare. Jag har i och för sig inte provat att sätta upp JsTestDriver och har nog aldrig kört det faktiskt och har hållt mig till andra verktyg.

15. Det är egentligen bara att man har en jar-fil som man använder som lokal server och så skickar man requests dit. Det är smidigt att använda men oklart hur det skulle

funka med Jenkins, eftersom man behöver hooka upp webbläsare mot servern för att ha någonstans att köra testerna. Att få det att ske automatiskt är kanske inte trivialt. Dessutom så har det varit lite buggigt, att om något test handlar i en oändlig loop så har jag varit tvungen att stänga ner den fliken i webbläsaren, starta om servern och återansluta. När jag satt med Mac innan så var jag tvungen att starta om servern varje gång datorn gått i vänteläge, typiskt sådant som man som användare blir frustrerad av.

Sånt kan vara väldigt irriterande och det är sådana saker som gör att folk inte vill hålla på med tester.

16. Du hade använt Mocha rätt mycket, det tänkte jag titta på sen.

Mm, vad kör du nu?

17. Nu kör jag JsTestDriver och Jasmine.

Det är väldigt likt, Jasmine och Mocha.

18. Så du kör Mocha istället för Jasmine?

Ja.

19. Är Mocha en test driver? För Jasmine kan väl inte köra testerna i en webbläsare, utan är i första hand ett assertion framework?

Det var så länge sen jag körde det, men ja man kanske behöver ha en driver till det för att kunna köra i webbläsaren.

20. Jag har uppfattat det som att Mocha är lite mer kompetent som driver och liknande.

Ja, det är det. Det är enkelt att dra upp tester i webbläsaren om man vill ha det.

Jag gillar nästan bättre att skriva tester i JavaScript än i Ruby, det finns så mycket praktiska språk-features.

21. Att manuellt kunna definiera om en funktion.

Precis, och hur lätt det är att göra nya fake-objekt, det är så enkelt. Ofta blir mocknings- och stubbningsramverk ganska överflödiga, förutom till att göra vissa call-assertions som man använder det till, men i övrigt så försöker jag att alltid hålla mig till fakes som är skrivna för hand för att det blir lättare för någon annan som sätter sig in i det.

När nästa person tittar på koden så är det en fördel om den personen inte behöver lära sig ett helt ramverk eller sätta sig in i hur DSL:er är uppbyggda bara för att förstå testerna. Om det bara är vanlig JS-kod så blir det lätt för folk att förstå vad det är som händer, så på det sättet är JavaScript väldigt väl lämpat för att skriva tester till.

Sen kan det vara så att hela Node-rörelsen också har bidragit till att det testas mer.

22. Det har väl också ändrat vad JavaScript används till, rätt mycket?

Ja, det har det garanterat.

23. Argumentet "det är gränssnitt så det är svårt att testa" håller inte riktigt. Jag fokuserar på klientsidan nu, för att jag var tvungen att avgränsa på något sätt och för att det känns som att problemen ligger på gränssnitt snarare än Node och liknande.

Om vi tar ett login-formulär som exempel, med en vy-klass som du kan anropa render på. Alla dependencies, inklusive templates, som den använder, laddas via RequireJS. Så i mitt test så använder jag RequireJS för att få in den filen och då är den redo att köra och det går att anropa render. Här gäller det att ha sina dependencies rätt, för om den vet om en massa saker utanför så uppstår situationer där man behöver mocka väldigt mycket.

Det är ytterligare en fördel med tester, att det blir tydligt vad som krävs för att kunna använda ett visst objekt och man blir uppmärksam på dependencies som inte ska vara där. Bara genom att anropa render och göra en submit på formuläret så hanteras det i vyn och man kan testa att rätt tjänster anropas, m.m. Igen så handlar det om att man måste hitta bra modularitet i hur man lägger upp det, så att man inte gör anrop direkt till ett API från en vy, utan ha ett servicelager emellan så att man kan swappa ut testerna och kolla att den går mot servicelagret. Det är sånt som folk fastnar på, att man har för dålig separation.

24. Vid själva utvecklingen av det här, vad skriver du tester för först då? Skriver du tester för vyerna innan du har de här tjänsterna igång?

Det beror på, det typiska man vill testa här är en validering. I det fallet skulle jag sätta upp formuläret först och sen fixa ett failande test som letar efter en validering som inte finns. Först därefter skriva själva valideringen. Man får ta det till en bra nivå, det känns inte jättevärt att testdriva varje steg av boilerplate-grejer när man vet exakt vad det ska bli och det nästan är copypaste-varning på det man gör. Då ger det mer att testdriva när man börjar komma till nästa nivå. Det känner man själv, när man måste börja tänka efter.

25. Det gäller att höra de varningssignalerna.

Ja, det här tror jag också är det svåra med testning, att det handlar om avvägningar. Det är som allt man gör med kod, att man måste ha med sunt förnuft när man gör det. Folk gillar att sätta upp principer och det tror jag att andra blir provocerade av. Principer behöver användas i sitt sammanhang och man får inte bli för religiös kring dem. Man behöver känna när det ger något och när det är i vägen.

26. Mm, det jag hoppas få som slutsats i mitt examensarbete är något i stil med "om du är i den här situationen, gör så här". "Om du gör en sån här app, använd ett ramverk av den här typen".

Ja, men då tror jag tyvärr att du kommer att landa i att "är du i den här situationen, använd ditt sunda förnuft".

*27. *båda skrattar**

För det där tycker jag att man ser rätt tydligt, som nybörjare så behöver man kunna reglerna för att bryta mot dem. De som är mer seniora kan reglerna och är inte så rädda att bryta mot dem när de är i vägen. Man måste vara väldigt pragmatisk med sina

principer. Det är om man inte är det som det uppstår konflikter av typen ”att skriva tester tar så mycket tid från det riktiga jobbet”. Om det inte ger dig något värde att skriva tester, gör inte det då! Det är ingen idé att göra det om du inte tror på det. Om du har något bättre sätt, go ahead! Om 20 år så är det kanske inte TDD som är grejen längre för att det kommit fram mycket smartare sätt och det kanske är ditt sätt.

Man måste se testning som ett verktyg man har, något man kan ta fram och stödja sig på i vissa situationer. Man kan också hamna i att man blir så inne i BDD och TDD att man slutar tänka. Att tänka på arkitektur blir någonting fult, man ska skriva tester först, det är det enda man får göra först. Att stå vid en whiteboard och rita blir plötsligt att gå händelserna i förväg och det tror jag är farligt.

Man ska inte glömma att precis som ens kod lätt blir ens baby så kan ens tester också bli det. Man kan skriva tester som man är riktigt nöjd med och då kommer det att ta emot att kasta den koden. Det ska man akta sig för, att bli för kär i sin kod. Då måste man ibland tänka till lite först, innan man bara sätter sig och hamrar igång. Där måste man hitta sin avvägning i vad man tror på.

28. Jag funderar nu på vilken nivå man väljer att hålla testerna på. De flesta av testerna jag skrev till asteroids-applikationen återspeglade hur jag förväntade mig att programmet skulle bete sig. Sedan var det någon kod som var rätt dåligt modulariserad och då hamnade testerna väldigt mycket på detaljnivå och det blev väldigt många tester som var svåra att överblicka. Testerna kan ju bli som en kravspecifikation och det är svårt att läsa dem som en sådan när det blir för mycket. Jag funderar på hur man egentligen ska tänka, ska man hålla en jämn nivå? Om man börjar inse att ok, nu skriver jag väldigt mycket tester på detaljnivå, jag kanske borde slänga de här testerna och skriva om koden istället?

Ja, egentligen så tror jag att det där är sådant som man verkligen måste känna efter vid varje situation. Om man till exempel har en jättekomplicerad och ointuitiv prisberäkning, för all del, skriv hur mycket tester du vill, men jag ser inget egenvärde i att ha en 100-procentig test coverage, för det slinker ändå igenom saker. Det finns alltid scenarios som du inte har tänkt på. Det är förmodligen det fallet du inte har skrivit ett test för som skapar problem, och det är för att du inte kom på det när du skrev testerna. Det finns ingen större mening med att försöka trycka in alla fall, utan det viktiga är att man skapar en känsla av att täcka in det ganska bra, att man har rimliga fall och framförallt tänka på den som kommer efter.

Om du öppnar en fil med 700 rader testkod så kommer du inte att engagera dig på samma sätt för det är för svårt att förstå sig på all den koden. Där måste man verkligen använda sunt förnuft och det var också någon tweet jag läste att om du ber en utvecklare review:a 10 rader kod så kommer du att få hur mycket feedback som helst, om du ber en utvecklare review:a 500 rader kod så kommer han eller hon att säga att det ser bra ut. Så är det ju, det är för mycket att sätta sig in i och bland annat därför så ska man inte vara rädd för att ta bort tester. Kasta gamla tester som inte behövs längre.

De bästa utvecklarna som jag har jobbat med har haft ett netto att det försvinner kod med varje commit, så att det kommer till grejer men samtidigt så försvinner kod. Det är imponerande, och har att göra med att de är bra på att se vad som inte behövs och

hur man kan tänka om, ibland genom att ändra på vissa grundantaganden för att få koden att bli renare. Med det vill jag inte säga att man ska gå över till någon sorts kodgolfande, men det är ändå intressant att det kan bli resultatet.

29. Vi kom in lite på att lägga till tester till kod som redan finns, vilket jag upplever är rätt så jobbigt och förstår att du tycker är lite dumt. Men är det inte också risk för dubbelarbete om man tänker att man bara skriver tester för ny kod och ersätter gammal kod som man inser inte funkar så bra?

Det beror på vad den gamla koden är i för skick.

30. Det kan vara lätt att tro att den är i sämre skick än den egentligen är. Om man tar över kod som någon annan har skrivit.

Javisst, om jag skulle ersätta ett banksystem, då är det klart att jag skulle börja försöka skriva tester för precis den lilla biten man ska in och härja i. Men med ny kod så tänker jag också som så att om man ska lägga till någonting nytt eller bara är inne och roddar, då kan man se till att lägga till tester för just det man gör. Då gör man det som en del i det nya man skriver. Det är svårt, för det är inte alltid det går. En del gammal kod är så ruttet skriven att testerna ändå blir så komplicerade att man måste mocka precis allting för att ens komma till objektet och logiken där bygger på så krångliga kedjor av anrop att testerna blir bräckliga och svåra att förstå oavsett. Frågan är om det ger så mycket då. Å andra sidan, i fallet med banksystemet, så måste det bli rätt. Då måste man hitta ett sätt att verifiera att systemet fortfarande beter sig rätt. Jag har nog aldrig varit i ett sådant fall att jag suttit med något så kritiskt som ett banksystem eller en cancerlaser eller liknande där jag behövt ersätta gammal kod, men det finns bra böcker på det där, som tar upp de situationerna.

De gånger det inte går att skriva meningsfulla tester så tror jag att det beror på att koden inte är skriven för att vara testbar. Oftast när folk pratar om legacy JavaScript så är det kod som ändå inte är banksystems-kritisk. Då skulle jag säga att en bättre approach är att testa allt nytt man skriver än att försöka täcka in allt gammalt.

31. Min tanke nu var att i min workshop/kompetensdragning så vill jag ge möjlighet att se om man har råkat ha sönder någonting gammalt och se den effekten av testerna. Jag tänkte ställa några frågor efteråt, ”hjälpte testerna er någonting?”

Det är nog bra, för det är ju också ett jättevärde med tester, när man släpper in nytt folk i ett projekt. Det är alltid så att folk tenderar att ha sönder grejer när de är inne och härjar i dem för första gången, vilket inte är så konstigt.

32. Och framförallt är rädda att ha sönder saker.

Ja, precis. Finns det då en bra testsvit så är det bara att köra. Jag tror att det är ett bra sätt att komma in i nya projekt också, genom att det finns en bra testsvit som man kan börja bygga vidare på så blir man tryggare i det man gör och kan imitera hur de tidigare testerna har skrivits. Då kan man lita på det man skriver, att det blir rätt.

33. Vinsten med att lägga till tester i efterhand så som jag har gjort är att jag verkligen fått lära mig, det känns nästan som att det är jag som har skrivit koden jag testat, trots att jag bara har varit och fingrat på några få rader i den.

Verkligen, man får en helt annan uppfattning för den. Men det är en utmaning att skriva bra kod som blir testbar. Sen det här problemet som du också var inne på, att det kan finnas integrationstestnings-problematiker, det kanske jag skulle vilja säga är det svåraste. Fallen då man har egna API:er. Hur man ska testa det rakt igenom.

34. Du skrev i ditt mail att du tyckte att det var knepigt att testa privata API:er för att de ändras så mycket.

Säg att jag integrerar mot Twitter eller någonting, deras API:er. Det är lätt att testa, genom att kolla att de anropas på rätt sätt. Det är väldokumenterade, stora, bra API:er. Det som är problemet är att hålla hastighet när man ändrar på sina egna API:er. Ett bra sätt att göra det internt är att du har ett test för någon typ av domän-modell som definierar vad som gäller för en viss sak, till exempel hur ett cykelhjul ska fungera. Då har man en uppsättning tester som testar att ett objekt är ett giltigt cykelhjul, och så kan man köra samma tester på sin fake av ett cykelhjul för att verifiera att den också är ett giltigt sådant. Denna fake kan sedan användas i andra test, som gör saker med cykelhjul. Om API:et för cykelhjulet ändras så kommer testerna för fakeobjektet också att faila, eftersom samma tester används på det riktiga objektet som på fakeobjektet. Då uppdateras den tills den uppfyller testerna för att vara ett giltigt cykelhjul och då kommer de andra testerna som involverar fakeobjektet att börja faila.

Egentligen skulle man vilja göra ungefär samma sak med sina egna API:er, genom att med tester definiera vad som är ett giltigt respons. Då kan man ha samma tester till att kolla att API-endpointen ger ett giltigt respons och att ens fakerespons är ett giltigt respons. Problemet man ofta har är att man separerar det där, istället för att ha en delad testsvit. Det blir oerhört komplicerat, framförallt om man skriver i olika språk. Det är ett problem som jag vet att många har och som jag inte har sett någon riktigt bra lösning på.

35. Och det är lite mitt emellan Selenium och enhetstester då?

Ja, precis.

36. Hur stort är behovet av att ha de här testerna?

Tyvärr så är det stort. Dels är det något som ändras snabbt och så är det någonting som påverkas mycket. Det är ju egentligen ingen skillnad på de testerna och tester mot andra API:er som du har i din kod, bara att de ligger i samma kodbas. Det är ju precis lika viktiga API:er.

På samma sätt, när du gör ändringar i en endpoint som du använder på något specifikt ställe så kan du missa att den även används på två andra ställen. Testerna kommer inte att uppmärksamma dig på detta för där är anropen fakeade till att härma endpointens tidigare beteende. Så behovet är ungefär samma och uppstår när ett team hanterar all kod. När ett API ligger utanför ens kontroll så har man tillgång till dokumentation och endpoints ändras inte utan vidare, så då kan man använda sig av fixtures för hur ett response ser ut och kan dessutom ofta använda versionerade API:er. Det är knappast troligt att någon på twitter får för sig att utan vidare ändra så att till exempel users får heta followers istället, utan att det blir en ny version, då skulle ju allt gå sönder.

Ur samarbetssynpunkt, när det gjorts en lokal överenskommelse som inte alla fått ta

del av så kan det vara knepigt för andra att veta vad som har beslutats om det inte finns tester som specificerar detta. Alternativet att ha ett kravdokument är inte så lockande.

37. Det är väl mycket det som många ser som vitsen med tester nu, att de blir som ett kravdokument fast ett bra sådant, som man inte behöver läsa utan man kör det istället.

Jo men verkligen, så är det ju.

38. Vad skulle du ge för tips till någon som är ovan vid att testa? Om vi säger att du skulle ha träffat mig för ett halvår sen, innan jag skrivit mitt första JavaScript-test?

Testa inte gammal kod, haha. Idag skulle jag säga ”kör Yeoman, sätt upp ett projekt, prova lite”. Det finns tyvärr fortfarande rätt så lite skrivet om det tror jag, inga riktigt bra resurser om JavaScript-testning, men jag skulle nog rekommendera att läsa clean code-böckerna och förstå vad det är du ska ha testning till. Utan den grundläggande förståelsen så är det svårt att se värdet i det och man hamnar lätt i tankesätt som ”men vadå, trots att jag har skrivit det här testet så kan det ju ändå bli buggar”. Utan att förstå konceptet testning och varför man gör det så har man svårt att väga olika verktyg mot varandra och motivera de val man gör.

39. Sen pratade vi också om att involvera alla i testningen. Det kan kanske vara en hjälp för en enskild programmerare att istället för att behöva kämpa för att få testa sin kod och hetsas till ett högre tempo ha testning som en del av det man förväntas göra.

Ja, absolut. Där handlar det igen om att vara överens om att det är viktigt och att det är ett sätt man vill jobba på, annars tror jag att det är svårt.

40. Vilka ser du som viktiga att involvera då? Teamet gissar jag är fundamentalt, men vill man även få ut det till produktägare, till chef, till kund, till användare? Var vill man lägga detaljnivån?

Det där beror så mycket på vad det är för kund och så vidare. Men framförallt så kommer det i arbetet att speca upp stories och liknande, för även om man kan komma bort från kravspecen så tror jag inte att man kommer bort från kravarbetet. Först och främst, hur funkar prisalgoritmen, hur är prissättningen, vart kommer grejerna ifrån? Där har man inget annat val än att jobba med produktägare och kund i att ta fram vad som ska hända, så det är snarare att vara involverade i story-arbetet tillsammans. Där tar man fram input till testerna så att man är överens om den. Sen tror jag inte så mycket på att man ska sitta och gå igenom testerna tillsammans med kunden för det brukar inte ge så mycket och blir ett onödigt steg.

41. Cucumber och liknande?

Ja, precis. Det finns ju bra exempel på det, men man ska veta vad man håller på med om man gör det.

42. Vi hade en tanke i vårt tidigare projekt om att låta användare skriva selenium-tester genom att ha ett plugin i firefox som genererar tester, så varje gång någon hittar något som inte funkar så skapas ett test för det.

Det är ju ascoolt.

43. Det blev aldrig så.

Det vore tufft om man kunde göra det, att en användare får repetera vad man gjorde. Det kan potentiellt bli hur häftigt som helst.

44. Hur ser du på selenium-tester i allmänhet, hur mycket tycker du att man ska använda det och till vad?

Det där är svårt, jag har kört det ganska mycket förut men det är mest för att jag inte har kunnat få ihop en runtime annars. I en gammal webb där man var tvungen att requesta sida för sida så är det ditt enda sätt att testa flöden. Nu när jag jobbar med en SPA (single page application) så kan man testa flöden ändå, då är man inte lika beroende av selenium. Jag tror inte heller på att testa flöden utöver att speca ner den mest grundläggande funktionaliteten.

*45. *Lunch**

46. Jag tycker att det var kul att du ser JavaScript som vilket annat språk som helst, för det är det verkligen inte alla som gör.

Nej, det är lite stående på mitt jobb också, när man hittar diverse bra grejer och reaktionen blir "Ah, det är nästan som att programmera på riktigt" och man får flika in att de borde vara tysta.

*47. *skratt**

Det har att göra med vad man gör i språket, om man använder ett språk till riktiga saker så blir det ju till ett riktigt språk.

48. Sen så funderar jag på alla de här ramverken, vad det är som du brukar tänka när du väljer vilka du ska använda, tycker du att det är svårt att veta vad de kan och så?

Jag går ofta fram och tillbaka där, det viktigaste är att man inte sätter sig i situationer där man måste jobba mot ramverket. Det leder ofta till dåligt resultat för att "det blir en massa kod som ingen förstår som motverkar en massa annan kod som ingen förstår" (obegriplig kod). Det där är ett problem som man nästan alltid hamnar i med de här magic bullet-ramverken, som Ruby on Rails till exempel. Man inbillar sig att man har ett ramverks som gör allt åt en, så att man kan göra en blogg på en kvart eller liknande. Så kan det också vara i små projekt, och då är det bra. När man däremot använder det i ett större projekt så uppstår lätt en känsla av att något inte är riktigt som det ska. Man lägger oproportionerligt mycket tid på att använda ramverket till saker det inte är tänkt för från början. Frågor som "hur gör man det här i ramverket?" uppstår och när man kommit till det stadiet så är man verkligen låst i ramverket. Det blir att man försöker anpassa sig till hur ramverket är tänkt att användas snarare än att man tar fram sin lösning utefter den vision man har.

I fallet med rails så tror jag att en stor del av problemet är att man har "felriktade dependency-pilar", man har bakat ihop persistence med domänobjekten till en kombination som är svår att reda ut.

Det viktigaste när man väljer ramverk är att hitta ett ramverk där man känner att man har frihet att strukturera koden som man behöver. Det är samtidigt något som man behöver känna sig fram med.

49. Jag tänker på det du sa om integrationstestning, att man ofta har separerat det, det skulle kanske kunna vara en ramverksfråga också? Om man har ett ramverk där man får uppfattningen av att man behöver strukturera koden på ett visst sätt så kan det hindra en från att skriva integrationstester för saker som tvingats isär av ramverket?

Det är ett stort problem och något man inte vill hamna i, att man inte kan testa grejer för att ramverket säger så. Det är verkligen grunden i att välja ramverk att man måste kunna undvika det.

50. Du sa att du gärna har ganska små ramverk.

Ja, och där kan man nämna backbone som ett exempel. Jag gillar att det håller sig ur vägen (unobtrusive). Den ger förslag till hur man ska göra vissa grejer som har med infrastruktur att göra och sen så får jag själv bestämma till exempel hur modeller ska läsas upp. Tyvärr är det inte alltid man kan bestämma vilket ramverk man ska ha och ibland så måste man välja stora ramverk för att man är beroende av ett stort CMS för något projekt eller liknande och då är det inte så mycket att göra egentligen.

51. Det är väl ofta så inom valtech att kunden säger "jag vill ha EPiServer" eller liknande.

Ja och då får man gilla läget mer, men det är klart, får man välja själv så... Ska det vara någonting litet tycker jag.

52. Jag har inte hunnit skaffa mig stenkoll på alla de här. Grunt, har du använt det?

Grunt är ett byggverktyg skulle man kunna säga. Vi använder det för att köra testerna, då tar det hand om att starta en lokal server och att packa ihop en distribution. Det är något av en motsvarighet till Rubys Rake eller Javas Maven. Det är viktigt för att det ska bli av att man kör tester och liknande, det måste vara lätt och tydligt hur man ändrar och därför behöver man bra byggverktyg.

53. Sen tänkte jag på stubbning med Sinon, vanillaJS och egentligen Jasmine också.

VanillaJS är ju bara ett skämt, man driver med...

54. När du själv skriver funktioner och ersätter...

Ja, men återigen så är det samma princip som i att välja små ramverk, när det gäller mockning och stubbning så tycker jag att det är bra att använda JavaScript så långt det går. Vilket jag egentligen tycker gäller för alla språk, även i Ruby använder jag i första hand det som finns inbyggt i språket.

55. Jag har läst en bok som är skriven av Christian Johansen som gjort SinonJS, Test-Driven JavaScript Development, där hela boken går ut på att man använder sig av manuell stubbning. Du sparar en kopia av en funktion, skriver över den med en egen och återställer efter testet. Det är först i sista kapitlet som han skriver "och förresten, det finns det här ramverket som jag råkar ha skrivit".

Det ligger ju jättemycket i det, för det är samma sak där som för andra ramverk. Annars sitter du där och slåss mot ett stubbningsramverk, helt ovärt, det är tid du aldrig kommer att få tillbaka. Dessutom blir det bräckligt, ju fler dependencies du har desto bräckligare blir det.

56. Jag insåg ganska sent att Jasmine har spies, och du kan välja om de ska anropa funktioner du stubbar eller inte, så varför använde jag sinonJS då? Sen insåg jag att Jasmine verkar återställa alla metoder du stubbar automatiskt efteråt, men det betyder ju också att jag inte kan stubba en sak, återställa och stubba på ett annat sätt i ett test. Låt oss säga att jag har en testsvit med en describe som jag har 20 test i och alla förutom ett stubbar en viss metod på ett visst sätt, då blir det knepigt. Det kanske går att komma runt, jag har inte försökt. Där funderar jag på vad man ska ge för rekommendation, för det är ändå det jag vill göra någonstans, "om du inte kommer göra det här och det här fancy grejerna så behöver du inte sinonJS, du kan nöja dig med Jasmine och att stubba manuellt".

Jag tror att det är bra med exempel, där man visar hur man kan lösa olika fall.

57. Tidigt i mitt arbete så insåg jag att de exempel som finns är ofta väldigt grundläggande, "så här testar du ett hello world". Men det är ju inte det som folk är oroliga över.

Det där är också problemet, för ofta vill man ju demonstrera en princip som är lika sann för hello world som jättestora JavaScript-applikationer. Folk gör det ofta väldigt lätt för sig så det vore nog bra att försöka hitta lite svårare fall, men inte krångla till det onödigt. Ofta gäller samma principer, tricket är ju att lyckas separera det så bra att ditt test blir som ett hello world.

Om jag skulle skriva en bok så skulle jag inte vilja ha med mina tester. De skulle ju innebära en massa stubs som lyssnar på metodanrop som går härs och tvärs. För det är inte något jag är speciellt nöjd med utan det är så men jag önskar att det vore bättre skrivet. Så det finns inte någon situation där jag kan rekommendera till någon annan att "skriv det här testet" utan snarare "var smartare än vad jag är, skriv bättre kod och lös problemet på ett bättre sätt".

Till slut så kommer man ner till problemet att man måste hitta bra abstraktioner. När det är precis rätt så blir det elegant. Därför tror jag att många exempel är så korta, för man är inte nöjd med de komplicerade testerna, det är inget man vill skryta med.

58. Sen är det ju så att när man väl har insett varför man vill testa och bestämt sig för att göra det så kommer man väl komma fram, men det vore skönt om jag kunde göra den resan lite enklare för folk.

Nej men ta fram bra exempel för det där om sånt som du hittar på vägen, det tror jag är bra.

59. Just ja, du skrev spikes i vår mailkonversation, om hur du gör när du inte vet hur du ska testa någonting.

Det är ett TDD-begrepp. Säg till exempel att man ska bygga något och det finns många osäkerhetsfaktorer kring vad slutresultatet ska bli och vad som kommer att fungera. Då är det användbart att ha versionshantering med Git, det är bara att skapa en ny branch

och köra loss, skriv inte test utan gör bara. Kasta in grejer, prova, gör, kör hårt. Sen när du har gaffat ihop din prototyp och den börjar fungera något sånär, då går du ur den branchen, gör en ny branch från samma ursprungscommit och börjar om fast med tester. Det är att göra en spike.

Man vill inte ha för långa spikes, för då blir det bökigt. Man ska inte sitta i två veckor och sen kasta och göra om, utan det handlar bara om att hitta en riktning och en känsla för hur man ska lösa problemet. Ofta är det så att man tror att det borde gå om man tänker på ett visst sätt och då kan man göra en snabb validering av det. Ofta kommer man till ett läge då man känner att det kommer att gå att göra på ett visst sätt, man är inte klar men börjar se hur saker och ting kommer att hänga ihop. Då är det läge att avsluta sin spike och börja om med tester.

60. Hur undviker man känslan då av att man gör dubbelarbete?

En motfråga är, vilken kod sätter du dig och skriver en gång? Du itererar, det är bara att du tar en iteration tidigare, en spike kan vara den första iterationen av din kod. Det är jämt att ”typing is not the bottleneck”, det är inte att skriva koden som tar tid, utan det som tar tid är att komma på hur man ska lösa problemet. Har man väl den lösningen, om man fattar vad som är rätt svar, då är det ganska lätt att skriva tester för det och sen skriva koden. Det där är intressant, för där kan man testa sig själv om det blir samma lösning om man testdriver något som när man bara häver ur sig något.

61. Det är väl en väldigt bra grej. Jag tror att det är något som många upplever med TDD, att de inte har det här verktyget och då står de där och försöker komma på tester för något som de inte har en aning om hur de tänker implementera.

Där har vi det igen, principer är inte till för att begränsa dig. Om man låter sig begränsas så hamnar man i läget där man känner sig oproduktiv, man måste ju tro på det man gör. Eller i varje fall vilja testa det, sen när man kan reglerna så får man bryta mot dem.

62. Du hade skrivit i ditt mail att du bara testar det publika API:et.

Ja, och då menade jag publikt API i termer av att jag aldrig skulle skriva ett test för en privat metod. Nu har man ju inte riktigt det i JavaScript, men...

63. Jag var nyligen hos Sony-mobile teamet och pratade med Kristoffer och han ville verkligen testa en privat metod. Det vi kom fram till var att vi kunde exponera den i testmiljön och gömma den i produktionsmiljön.

Jo, men... Jag tycker ändå inte att det är ett bra test. För det är en implementationsdetalj, det är som att testa variabelnamn. Om det blir för stort så kanske man exponerar fel saker. Man kanske vill ha ett annat API, dela upp i två objekt till exempel, som har API:n sinsemellan. Jag tror att... det finns säkert undantag. Vet man vad man gör, by all means, testa privata metoder, man kommer inte att hamna i TDD-domstolen för det. Men generellt så är det en varningsflagga när man vill skriva ett test för en privat metod, då är det läge att tänka efter.

64. Min reaktion var att han kanske kan skapa en klass för det som han vill testa. Då var det tydligen tio rader kod och han var rätt nöjd med att det såg ut så.

Jag hade nog inte gjort så, men det är ju en pågående diskussion och folk säger att de har ett case för att göra det och visst jag kanske någon gång hamnar i ett case där jag känner att nu måste jag testa den här privata metoden, men jag har inte varit där än.

65. Jag hade planer på att skriva tester för det vi gjorde i våras med konsultprofil-sidan, och där hade vi mycket jQuery och använde module pattern. På ett felaktigt sätt förmodligen. När jag sedan tittade på det här och frågade mig hur jag ska kunna testa det så var det verkligen som du sa innan om legacy JavaScript att det inte gick att testa alls. De enda testerna jag kunde skriva var "existerar den här funktionen?" och det är ju i princip att testa variabelnamn.

Ja och det är ju ganska pointless (meningslöst) egentligen, det kommer du inte ifrån. Den viktiga grejen med att inte testa privata metoder är att tester är till för att kolla att saker blir rätt och i övrigt är du fri att göra vad du vill. Jag tänker att problemet med hans privata metod det är när du kommer in i kodbasen och har ett mycket elegantare sätt att lösa hans problem på, då finns det ett test som säger att den här metoden måste fungera så här och då kan du ju inte göra din eleganta lösning för den måste ju tydligen göra så. Då måste du gå till någon som vet och fråga vad tanken bakom testet är för att i bästa fall kunna våga ta bort det. Det är faran med att testa för djupt.

66. Det kan jag nog känna med de tester jag skriver nu också, inför det här workshopen, att det kan eventuellt uppstå en sådan situation att jag har skrivit ett test utifrån en specifik implementation och missat poängen med den. Att hitta rätt nivå att lägga sig på är inte lätt. Framförallt inte när man skriver tester i efterhand, vilket han ju också gjorde.

Nej, det är ju inte det. Då är det rätt hopplöst. Man får även skilja på konceptet privat metod som i att det står private i koden och att det är en privat metod. I JavaScript så kan man inte göra en metod privat, men den kan ändå vara tänkt att användas som om det vore det och inte är till för att exponera. Det är bra om detta framgår i testerna.

*67. Du kan testa att en metod är privat? *skratt**

Även om du skulle fälla in (inline) den metoden direkt i din kod så skulle testerna fortfarande gå igenom. Koden gör fortfarande rätt även fast den är skriven på ett annat sätt.

68. Kodduplicering, när man ser till att hålla testen korta så är det kanske inte ett lika stort problem om det bara är tre rader i varje test.

Jag tycker att där är läsbarhet viktigast. Man ska inte behöva läsa 700 rader testkod för att komma till kärnan med det.

69. Det är många som säger att DRY-principen inte behöver tillämpas på tester och det är samtidigt många som säger att tester behöver vara maintainable.

Det är absolut en avvägning, men jag tror helt klart att caset är annorlunda jämfört med när man skriver produktionskod. Det är också det här med premature optimization, att folk använder DRY-principen fel, både i test och i produktion. Om man tolkar det

som att samma kodrad inte får förekomma två gånger i ett projekt så har man missat poängen, det handlar ju snarare om att man inte ska duplicera funktionalitet. Om två olika saker råkar som en del av deras funktionalitet göra samma sak så betyder inte det att det ska dras ut till en metod, alltid. Det är likadant med tester, ofta är det så att saker ser väldigt lika ut, som att de gör samma sak, men ofta är så inte fallet. Då hamnar man i dåliga abstraktioner som i själva verket gör testerna mindre maintainable.

70. Det tycker jag ofta att man kan märka, man tänker "ok, om jag ska ta ut det här till en separat metod, vad ska jag döpa den metoden till?" Om namnet börjar innehålla villkor "om det är så här gör så här" och liknande så kanske det inte är rätt sak att göra.

Där kan jag ofta testa mig själv genom att göra koddupliceringss-spåret och i efterhand fråga mig hur lika de olika delarna blev. Ofta visar det sig att det inte var så mycket samma grej som man skulle göra i de olika situationerna. Så det är en princip som har förstörts lite, det har på vissa håll gått för långt och övergått till att man kodgolfar.

71. Ja det är ju ofta man kan minska antalet rader kod och det är ju det folk börjar sikta på nu så. Det är väl det du menar med kodgolfning?

Ja, precis, och det är inte alltid så bra.

72. Hur mycket tid har vi kvar förresten?

Jag måste nog börja röra på mig.

73. Jag kan maila frågor om det är så.

Ja men gör det, absolut. Hade du något mer?

74. Nej, inget viktigt.

Då ska jag nog börja dra mig. Men vad kul, det blir spännande att se, du får gärna skicka ditt arbete till mig när du är klar.

Post interview mail conversation with Johannes Edelstam

75. Vad var det för event du körde handuppräckning på?

Tror det var ett sthlm.js event!

76. Hur anser du att sannolikheten för förändringar bör påverka testningsambitionerna?

Vet man att det är en prototyp man bygger kanske man inte ska ha lika hög ambition som när man bygger nästa cancer laser.

77. På vilket sätt underlättar require.js vid testning?

I och med att du har ett strukturerat projekt med separata och isolerade moduler slipper du den dependency soppa som oftast gör saker svåra att testa. Dessutom blir det enkelt att fakea objekt!

78. Hur påverkas behovet av mockning av hur man skriver sina tester?

Skulle snarare säga att det beror på hur du skriver produktionskoden. Om du lyckas decoupla din kod ordentligt så behöver du generellt färre mocka.

79. När förekommer "det riktiga objektet" vid testning av privata APIer?

Jag tycker inte man ska testa privata API:er :) Men om du menar att man internt har ett backend som pratar med ett frontend genom ett internt API så är det ett problem att skriva integrationstester som testar hela systemet på ett enkelt sätt utan att sätta upp komplexa seedningar och liknande. I vissa fall går det att komma runt (man behöver inte en massa state i backend t.ex.), men inte alltid.

Mail conversation with Patrik Stenmark

Vad har du använt JavaScript till under din yrkeskarriär?

Allt från enklare "få små saker att röra sig och animeras" på små siter till stora applikationer (single page application-style).

Vad ser du som de främsta fördelarna, respektive nackdelarna, med språket?

Fördel: Det finns i alla webbläsare.

Nackdelar: Det är ju helt konstigt språk. Objektmodellen är schizofren (varför finns new-operatorn?). ===/==. Mycket ceremoni för ett dynamiskt typat språk. typeof NaN === "number", och andra sköna inkonsistenta saker.

Vilka verktyg och ramverk har du använt mest?

I produktion: Bara backbone.js som ramverk. Labbat med halvstora grejer i Ember.js och Angular. Testning via Jasmine och Buster.js. Sinon.js som mock/double/spy/whatever-ramverk.

Vad är din syn på testning av JavaScript jämfört med Ruby, är det svårare/enklare, lika viktigt, ovanligare?

Beror på. Testa saker som har med DOM att göra är svårt, men det är minst lika svårt att testa något som är skrivet i Ruby som har ett UI.

Jag skulle snarare säga att det är vad man ska testa snarare än språket som avgör om det är svårt/lätt. Väl strukturerad testdriven JS-kod är mycket lättare att testa än fulhackad organiskt framväxt Ruby. Vettig struktur gör att det blir lätt att testa.

Tester är för mig inte främst ett verktyg för verifiering av funktionalitet utan ett verktyg som hjälper mig att skriva väldesignad kod. Testerna tvingar mig att fundera på beroenden mellan delar av systemet och gör att jag känner mig säker på att saker funkar trots att jag refaktoriserat kod.

Viktigheten beror också på och har inget med JS eller inte JS att göra. Är det javascript som är byggt för att hantera ett komplicerat betalflöde i en webbshot så är det viktigt. Är det för att animera företagets logga, not so much.

Det är definitivt ovanligare. Min erfarenhet är att JS-kodare ofta gillar quick-and-dirty, fulhacklösningar och inte tycker det behövs nån direkt struktur. Denna typ av kod är i

stort sett omöjlig att testa (enhetstesta) och därför är det många som är av åsikten att "javascript går inte att testa".

Vad inom testning i allmänhet tycker du är särskilt svårt? Vilka är de främsta utmaningarna du har stått på inom testning och hur hanterade du dem?

Gränssnitt är alltid svårt. Även saker som är beroende på externa saker, i JS-fallet kan detta t ex vara twitter eller facebooks API:er. Eller ett egetutvecklat API som används av JS.

En lösning på detta är att använda sig av någon form av mockning, men då får man istället problemet att den mockade datan kan komma i osynk med den riktiga datan.

Kod som beror på DOMen. Detta går att göra mycket mindre genom att flytta så mycket logik som möjligt från DOM-beroende saker till icke-DOM-beroende och testa detta och sedan bara ha ett tunt gränssnitt som använder sig av detta underliggande. Problemet ligger i att DOM-beroende tester blir långsamma och beroende av HTML-strukturen på siten.

Vad ser du för problem i samband med testning av JavaScript och gränssnitt?

Tja, se ovan :)

Vad anser du kännetecknar bra tester?

Snabba. Tydliga. Både att de är tydligt vad de testar och att setup-biten är lätt att förstå. Inte för mycket "magi". Jag tycker det är bättre att duplicera lite kod i ett test än att bygga komplicerade hjälpfunktioner eller arvsstrukturer för att göra testerna mer "DRY".

Sen så klart det förhoppningsvis uppenbara, de ska gå sönder när appen inte fungerar :)

Hur går du tillväga för att avgöra vad du ska testa?

Mest magkänsla :) Men det jag funderar på är väl

- Hur viktig är denna funktionalitet?
- Hur sannolikt är det att den kommer ändras i framtiden?
- Hur komplext blir det?
- Hur mycket av lösningen är rent GUI och hur mycket är "logik"?

Vad ser du som de främsta fördelarna med testning?

Bättre design på koden eftersom testerna (Om man "lyssnar" på dem) tvingar att man tänker på beroenden mellan delar av koden och mellan "logik" och GUI och man kan refaktorisera koden. Man kan lita (mer) på att de ändringar man gör inte har sönder funktionalitet. Om man ser till att skriva tester för buggar kan man se till att en bugg bara uppstår en gång.

Hur påverkar testning sättet du arbetar på i ett team och tillsammans med en kund?

Jag tycker nog att det viktigaste som konsult är att anpassa sig till teamet/kunden. Skulle jag komma till ett greenfield-projekt eller till ett projekt där jag ska utveckla en väldigt isolerad del av en site (Ex: Kartan på antagning.se) skulle jag pusha väldigt hårt för att vi skulle börja jobba testdrivet.

Är det en redan existerande kodbas skulle jag pusha för att försöka få den testbar, men inte lika hårt eftersom ROI:n på detta är mycket lägre på kort sikt. Då skulle jag nog försöka få till någonting i alla fall, men anpassa mig till den existerande kodbasen och teamet.

Transcript of Interview with Patrik Stenmark

1. Du skrev i ditt mail till mig att en nackdel med JavaScript är att det är "mycket ceremoni för ett dynamiskt typat språk". Vad menade du med det?

Jag borde kanske ha skrivit funktionellt språk snarare än dynamiskt typat. Jag menar att det till exempel är ganska klumpigt syntaxmässigt att skicka med en anonym funktion i ett anrop. Du måste skriva function med parenteser, klammer, i vissa fall semikolon efter definitionen, 20 rader nedanför, i andra fall ska det inte vara det, beroende på om det är ett objekt eller bara en funktion du definierar. Man behöver fundera rätt mycket på syntax.

2. Det känner jag igen som har suttit rätt mycket med python, där tycker jag att det är bättre.

Jag är ju Ruby-utvecklare om jag får välja själv. Python är ännu renare, för i Ruby finns det en del saker som man får fundera på, men i JavaScript så är det så i stort sett hela tiden. Om man till exempel ska flytta en funktion från att vara anonym till att bli en egen funktion för att den har blivit för stor, då är det inte bara att flytta den utan man behöver även fundera på om den hamnar i ett objekt så att det måste finnas kommatecken på rätt ställe eller om det blir en vanlig funktionsdefinition som ska avslutas med semikolon. Om den flyttas sist i ett objekt så ska det inte vara kommatecken, för då blir det syntaxfel i vissa versioner av Internet Explorer. Det är mycket sånt som jag tycker blir väldigt jobbigt. Visserligen finns JSLint och JSHint och liknande som kan hjälpa till, men det är fortfarande lite för mycket saker som man inte borde behöva tänka på.

3. Du skrev att du har använt JavaScript till både små saker och till lite större. Tycker du att det lämpar sig bra till stora applikationer?

Nej, *skratt*, egentligen inte. Jag tycker inte alls om JavaScript som språk, jag tycker att det är alldeles för mycket specialfall och konstiga quirks. Objectmodellen är väldigt konstig och känns schizofren i att man har ett prototyp-baserat system men också en new-operator som emulerar new-operatorn i Java, fast inte riktigt. Det är mycket sådana designbeslut som jag tycker är ganska konstiga. Däremot så är det ju ett språk som finns tillgängligt överallt, så det gör att det ändå funkar att hålla på med. Om det hade varit så att alla webbläsare hade stöd att exekvera Ruby utan att först ladda hem emscripten så skulle jag definitivt föredra det. Sen så finns det ju ramverk och grejer som hjälper

till och gör att det fungerar bättre, men i grunden så nej, jag tycker inte att det är ett lämpligt språk att skriva stora applikationer i.

4. Så det är mest att det är enda alternativet?

Ja.

5. Nu är jag inte helt insatt, men Single Page Applications (SPA) känns som en typisk grej där JavaScript är enda valet.

Ja, där har du ju inte något alternativ egentligen. I varje fall så måste slutprodukten vara JavaScript. Den största grejen jag har gjort, som är en SPA, skrev jag i Coffeescript, men det är ju JavaScript.

6. Hur ser du på testning när det kommer till stora program, blir det mer relevant då?

Ja, definitivt.

7. Tror du att man klarar sig utan det överhuvudtaget?

Ja klarar sig gör man ju, men utifrån min erfarenhet med en tidigare applikation som jag har jobbat med, som tog in statistik på ena sidan och byggde upp grafer som gick att filtrera på olika parametrar, så blev jag flera gånger hjälpt av testerna, särskilt när ny funktionalitet skulle läggas till. Då fanns det en bekräftelse på att jag inte hade haft sönder någonting tidigare och testerna manade mig till att skriva lite mer välstrukturerat.

Från vad jag sett i andra projekt så när man inte har tester så blir det bara en stor soppa till slut där varenda ändring man gör tar jättelång tid och har sönder fyra andra funktioner. För att man inte var medveten om att en viss jQuery-handler måste köras före en annan och så har man bytt plats på lite kod för att öka läsbarheten och helt plötsligt funkar ingenting längre.

8. Jag vet inte hur mycket du har hållit på med node?

Ingenting.

9. Det jag funderat lite på är hur attityden gentemot testning har förändrats genom tiden, det känns som att node har varit en milstolpe som gjort att man börjat utveckla i JavaScript mer seriöst.

Det har ju påverkat testning inom webb-JavaScript också, för trots att jag aldrig har kodat för node så har jag ändå haft nytta av att ha en JavaScript-runtime tillgänglig som gjort att man kunnat köra testverktyg. Tidigare behövde man skriva en html-sida och ladda i webbläsaren för att köra sina JavaScript-tester. Det var ganska meckigt, att sätta upp en ny testsvit var att sätta upp en ny html-sida. Det node har gjort för mig är att jag kan köra mina tester i terminal med PhantomJS eller Selenium beroende på vilken nivå jag vill lägga det på. Så på sätt och vis har jag använt node, men jag har inte kodat node.

10. Varför tror du att JavaScript-kodare ofta föredrar quick-and-dirty?

Jag skyller på jQuery. *Skratt* Det har mycket att göra med hur JavaScript började, för 10 år sen så hade du inte kunnat skriva en SPA för du hade knappt AJAX men jag kan tänka mig att många har lärt sig JavaScript för att kunna animera en knapp, vilket är enkelt att göra genom att skriva några få rader jQuery i en fil och inkludera den. Sen utvecklas det till att man vill ha lite mer menyer som expanderar och andra effekter vilket i sin tur övergår till att man implementerar faktiska funktioner i JavaScript. Då har man läst tutorials på Internet som berättar att man skapar en viss handler i vilken man läser in något från DOMen och manipulerar det och använder en selektor för att hämta ut ett visst element och i det läget så är testbarheten i stort sett förlorad även om man har en utvecklarbakgrund sedan tidigare.

Sen så tror jag att många JavaScript-utvecklare kommer från designer-sidan, som inte har kunskaperna som krävs för att riktigt veta vad det är de gör. De är designers som har lärt sig jQuery och kan det, utan att egentligen vara så insatta i programmering i stort. Sen så sitter de och lappar ihop saker och kan absolut vara asgrymma på jQuery och åstadkomma jättehäftiga effekter och bra saker på så sätt, men de har inte strukturen och koddesign-tänket som man får som utvecklare när man bygger stora saker. Det har jag märkt när jag jobbat med frontend-utvecklare att de tycker att det är helt rimligt att ha en fil med 3000 rader kod med en funktion som är 250 rader lång som bara innehåller DOM-uthämtning och AJAX-request och allting i en stor soppa. De tycker inte att det är något konstigt överhuvudtaget, för det är så de har lärt sig. Tidigare har det varit väldigt lite information på Internet om hur man gör. Tutorials, dokumentation och liknande visar inte det utan det är de snabba lösningarna som visas oftast.

11. Så, det har varit lite information om hur man gör det bra.

Ja.

12. I ditt mail nämnde du några ramverk som du använt: Backbone, Ember, Angular. Framförallt Backbone, vad jag fattade det som. Och så lite testning men Buster, Jasmine och Sinon. När tror du att de är lämpliga att använda sig av?

Idag skulle jag nog inte använda Backbone överhuvudtaget, det var jättebra när det kom men det känns som att det hade en hel del brister som har stannat kvar och sen så har det kommit en massa andra som har sett Backbone och tyckt att det varit bra men att vissa saker saknats och därför gjort något nytt som är snäppet bättre.

Det finns olika komplexitetsnivåer på det, Backbone är ju i den lägre skalan, det är väldigt simpelt egentligen. Den typen av ramverk känns rimliga när det är små applikationer. Kanske som en del av en större site som hålls isolerad från övriga delar av siten. Däremot så om man ska bygga en SPA-style så skulle jag nog inte gå mot Backbone utan istället utgå från hur komplext det är och välja Angular eller CanJS om det var av medelstorlek eller Ember om det gäller en applikation i stil med Gmail och avancerade forum-mjukvaror, som ska konkurrera med desktop-applikationer. Att dra fram Ember för att göra en liten Todo-lista är bara overkill och onödigt.

13. Du tror inte att det är risk att det är i vägen när det är så stort?

Jo, det är det, men jag tror också att när det är en så komplex applikation så är det värt att ha ett ramverk som hjälper till rätt mycket, i synnerhet när man måste använda

JavaScript. Men absolut, jag är ju Rails-utvecklare och i den världen så är jag väldigt mycket inne på att försöka frikoppla så mycket som möjligt från Rails när jag bygger saker. Jag har inte sett det göras ordentligt eller på ett bra sätt i JavaScript-världen ännu. Det kanske går, jag kanske ändrar mig om ett par år eller någonting, men just nu så känns det som att det enklaste och mest effektiva är att ha ett ramverk som hjälper till rätt mycket. Sen kan man säkert hamna i att man får slåss lite mot det eller anpassa vad man vill göra till vad ramverket klarar av.

14. Vad är de största vinsterna med de här då, vad är det man får hjälp med?

När det gäller Ember så får du ju strukturen på applikationen, hur den ska byggas upp. Allting bygger på någon slags state machine som styrs av vilken URL du är på. Sen att du får en riktig MVC, eller i varje fall riktigare än de flesta andra ramverk, och att du har databindningarna både på ren data och på det som kallas computer properties, så du kan ha en funktion som genererar ett värde från andra värden som du också kan binda till DOM-element, vilket gör att mycket av situationen där en händelse ska trigga en mängd andra händelser blir enklare.

När det gäller de enklare verktygen så får man främst organisation och hjälp med boilerplate-kodning. Man vet vad vyer och controllers förväntas innehålla, vilket ger struktur.

15. Känner du att det hjälper för att skriva tester också?

Tankepaus Ja, det gör det ju. Det får lite samma funktion, de hjälper varandra kan man säga. Du kan inte skriva tester för dåligt strukturerad kod utan att det blir komplicerat, så det hjälper ju dig att du redan har någon form av struktur, men jag tror också att testerna hjälper dig att få ännu bättre struktur även om du använder Backbone eller Ember. För du kan ju fortfarande skriva dålig kod i de ramverken, men då hjälper testerna till att förhindra det.

Om du inte hade något ramverk från början så skulle du nog hamna i en bra struktur ändå om du bara körde testdrivet. Jag påstår inte att det blir bra bara man har tester, men testerna kan hjälpa dig även om du inte har ett ramverk. Båda hjälper från varsitt håll.

16. Hur mycket har du testat just gränssnitt då?

En del, men du har det inte varit så mycket i JavaScript utan mer Selenium från Ruby. (Jag har skrivit tester med Jasmine som körs med Selenium aldrig så att alla tester körs via JavaScript.) Det är väldigt sällan värt det, för allting blir så känsligt för förändringar i utseende, det är långsamt och så fort du har någon form av asynkronitet så får du tester som failar ibland. Det kan bli så att de failar vissa tider på dygnet och kontentan blir att de gör mer skada än nytta för att de inte går att lita på och tar så lång tid att köra att man till slut inte orkar vänta på testerna.

Däremot så tror jag att det är bra att ha väldigt övergripande för de viktigaste flödena. I mitt nuvarande projekt så har vi Selenium-tester för de flöden som har med pengar att göra trots att de involverar en del JavaScript, men vi testar inte att dropdown-menyer, växling mellan olika vyer och liknande fungerar utan bara det absolut viktigaste som ett smoke test.

17. Vad menar du med smoke test?

Att prova att gå till sajten, se om den exploderar ungefär. Väldigt övergripande för att se om standardfallet fungerar som det ska. Testa siten och se om det ryker från den eller inte. *Skratt*

18. Så det är just Selenium du har använt för gränssnitt då?

Ja, eller nej, inte bara. Jag har gjort en del genom att ladda in någon form av template i någon testhtml-sida och sen köra jQuery mot den div:en. Då hamnar man i det som jag nämnde i mitt mail, att man behöver hålla det i synk med vad som finns i produktion. Det kan bli så att man ändrar i designen på siten men att testerna fortfarande går igenom eftersom de är baserade på templates som du har skapat trots att siten har blivit trasig. Då har testerna körts av Selenium på en CI-maskin men det har inte varit Selenium som har klickat runt saker utan det används bara för att ladda sidan och sen har jQuery gjort resten.

19. Jag hade några följdfrågor här men de är inte så relevanta nu. Vi pratade i vår mailkonversation om externa APIer, du tyckte att det var svårt att testa dem. Vad är det som gör det svårt?

Det är ju lite samma sak som med gränssnitt, att du har ett beroende på något som du själv inte har full kontroll över. Facebook är till exempel kända för att ändra i sina APIer lite hur som helst och när som helst, när de känner för det. Det finns inget vettigt sätt att testa dem mot live-miljön, för du vill inte att du går in och like:ar något tusen gånger per dag för att du kör testerna hela tiden. Då kan man förstås göra någon slags mock och bara testa att rätt metoder anropas och så där, men då har man problemet att om Facebook ändrar i sitt API så kommer ens tester fortfarande att gå igenom men siten kommer inte att fungera.

Samma sak gäller för vilka APIer du än använder, det kan även vara så med APIer du har själv, för du vill ju inte köra dina enhetstester mot en riktig server för då är de inte riktigt enhetstester längre utan har blivit någonting annat. Det är enklare när det är APIer man själv har kontroll över för då kan man sätta upp någon slags integrationstestmiljö som man använder bara inför release eller liknande för att kontrollera att de mockade testerna fortfarande är uppdaterade. Så gjorde jag när jag arbetade med det där statistikhanteringsverktyget som jag nämnde tidigare och det fungerade riktigt bra, kanske för att det var ett API som jag själv byggde och hade full kontroll över. Så fort det blir något som man själv inte har kontroll över så blir det mer komplicerat.

20. Det är väl så att privata APIer ändras ännu oftare och att det ytterligare späder på problemen med att den mockade datan kommer i osynk med den riktiga datan, så som du skrev i ditt mail till mig. När jag intervjuade Johannes Edelstam så föreslog han att man kan ha tester som kontrollerar att ett objekt är ett korrekt request eller ett korrekt response, och köra dem både på de riktiga objekten och på din fake som du har när du mockar ett API. Då märker du att om testerna slutar gå igenom på de riktiga objekten då behöver du ändra på dem och då kommer de att börja faila på faken, och då ändrar du på faken, och då kommer de tester som faken används i att faila. Har du använt dig av det någonting?

Inte i JavaScript-världen, men i Ruby-världen så gör jag det ofta nu för tiden. Man behöver oftast komma ut på Internet för att kunna göra det, vilket ibland är ett problem och ibland inte. Jag hade någon betalningslösning en gång där det kostade pengar att göra testbetalningar, och då vill man förstås inte göra en testbetalning varje gång man kör sina tester. Men ibland så är det inga problem, om det till exempel gäller en sökning på Twitter så är det säkert inga problem för det får du göra ganska många. Jag tycker absolut att det är en bra idé och jag använder den själv när jag jobbar med backend-kod, jag har dock inte provat det i JavaScript.

21. Du skrev i ditt mail att DOM-beroende tester blir långsamma. Var det just Selenium du tänkte på då?

Oavsett om du kör Selenium eller PhantomJS eller någonting så är de ju långsammare än rena JavaScript-tester.

22. Är det för att det är stora objekt som de skapar eller...?

Ja, och DOM:en är ganska tung att jobba med, antar jag. Jag vet faktiskt inte varför det går långsamt även när du kör Phantom, men det blir väl mer att göra helt enkelt. Det kanske inte är så jobbigt att köra ett test, men om du har 100 tester som är oberoende och varje test tar 100 ms så är du ändå uppe i 10 s körtid. I mitt nuvarande projekt så har vi väl 6-700 enhetstester och om alla de skulle ta 100 ms så skulle vi ju ha blivit gråhåriga vid det här laget. På enhetstestnivå så blir det rätt många tester, i varje fall för mig.

23. Vad strävar du efter då när du har så många tester, försöker du täcka in många olika fall vill du bara ha ett test som talar om vad en sak ska göra?

Nej, på enhetstestnivå så vill jag ju täcka in allt i den mån det går. Det är ett slags mål som jag vet inte riktigt går att uppnå, men ändå det jag satsar på.

24. Just för att kunna känna dig trygg sen när du gör ändringar?

Ja, precis. När jag ändrar och framförallt när andra ändrar. Om jag sitter själv på ett projekt så har jag oftast koll på vad som händer, men när man är flera personer, kanske geografiskt separerade som vi är i vårt nuvarande projekt med fem personer i Stockholm och två på Malta, så är det en stor fördel om man kan förvissa sig om att ingen annan har haft sönder någonting genom att se att testerna fortfarande går igenom.

25. Jag såg att du skrev på intranätet att du kom tillbaka från semestern och fick sätta dig och fixa tester, var det någon annan som hade...?

Ja, eller, det var någon som hade stängt av testerna för att det skulle fortsätta vara grönt i Jenkins. Jag köpte inte riktigt det tankesättet. Så fort man har tester som ibland inte fungerar, så kan man lika gärna kasta dem, för om de spontanfailar så att man inte kan lita på dem så blir det bara onödigt jobb.

26. Var det det som var fallet nu?

Ja, en av testerna hade lite för låg timeout tror jag att det var, så ibland hann den inte svara. Istället för att lösa det problemet så hade de bara stängt av alla testerna.

27. Det är väl viktigt att folk har samma ambitioner, kanske.

Mm.

28. Du listade lite olika saker som du brukar ta hänsyn till när du avgör vad du ska testa. Du sa att du gick mycket på känsla också, men. Hur relevant något var, det tolkade jag som att det var just funktionaliteten, hur relevant den var. Att om det är väldigt relevant så är det större chans att du testar.

Ja, som jag sa tidigare. Betalningsflöden kör jag i Selenium för att vara hundra på att det fungerar, men en liten dropdown-meny som visar kontaktinformation kanske jag bara fulhackar ihop med jQuery utan att skriva något test för det. Sen finns det en skala därimellan.

29. Sannolikhet till förändring, om något förändras ofta, är det en anledning till att testa det?

Ja, precis. Om vi vet att vi kommer att lägga till funktioner och ta bort och ändra så vill man se till så att det man har byggt fungerar, men om man vet att det är något man gör en gång och som sen kommer att ligga kvar där, då tjänar man nog på att manuellt testa det så att det funkar och sen kommer det inte att förändras så därför kan man låta det vara. Den är väldigt svår, för man kan ju idag tro att något inte kommer att förändras och sen kommer en produktägare in imorgon och säger att det visst behöver ändras. Det är ganska vanligt. I de flesta fall skulle jag skriva något väldigt lätt happy path-test bara för att se till att ha den infrastrukturen på plats och förhoppningsvis så skulle personen som gör de här ändringarna inse att det var dåligt med tester just där och skriva dem då. Så försöker jag göra, att när jag ska göra en ändring på något som är otestat så försöker jag, om det går och passar in i projektkulturen, lägga upp det så att det får någon slags test runt sig.

*30. Det vore ju grymt att ha en sån i teamet. *Skratt**

Mm, men det kräver ju att teamet gör det, inte att en person gör det, för annars kommer den personen att bli galen efter ett tag. Om ingen annan gör det så kommer det att kännas som att alla andra har sönder saker.

31. Hur mycket känner du att du får med dig andra när du gör så här? Behöver du slåss för att det ska bli som du vill?

Det har varit väldigt olika på olika ställen. Allt från ”va, ska man köra testerna också?” till folk som aldrig har kört testdrivet tidigare men som när man berättar fördelarna reagerar ”oh, det här är ju det bästa jag har hört talats om, det är klart vi ska göra det här”, och sen de som redan är medvetna om att det är bra såklart. Jag har varit tvungen att anpassa mig, på några projekt har jag velat göra mycket mer men ingen annan gör det och då tar jag hellre och gör mindre just för att jag annars blir den som inte producerar någon kod, jag skulle bara sitta och skriva tester. Som konsult så tycker jag att det viktigaste är att man anpassar sig till den redan existerande team-kulturen även om jag aldrig skulle sluta försöka övertala folk.

32. Men om man hamnar i en sådan situation att man nästan bara skriver tester, då låter det som att man skriver tester till existerande kod?

Ja, precis. Det skrev jag väl också, att om det bara var ett nytt greenfield-projekt så

skulle jag pusha ganska hårt för att få någon form av testkultur i teamet, men om man kommer in någonstans där det finns en massa kod som kanske inte är sådär välskriven, det är då det tar tid att skriva testerna. Då måste man ofta skriva om saker för att få dem testbara.

Om man är den enda som vill göra det så kan det bli lite konstig stämning i teamet. Man sitter i två dagar och skriver om kod så att den går att testa, sen sitter man i två dagar och skriver tester och sen ägnar man en dag åt att faktiskt göra det man skulle göra från början. Om man däremot har ett team där alla är med på det så tror jag att det är en väldigt bra sak för du kan komma till ett läge då det går långsamt första tiden men ju längre man kommer desto snabbare går utvecklingen för att man blir säkrare på att man inte har sönder saker och man kan skriva om kod så att den blir mer lättförstådd. Det är i så fall en långtidsinvestering, som man inte kan göra som enskild team-medlem.

33. Du nämnde komplexitet också, det framgick inte om det var komplex kod du syftade på eller om det var komplexa tester. Jag gissar att om det är väldigt komplex kod så vill man testa den och om det blir komplexa tester så vill man kanske inte ha dem.

Ja, precis. Det skrev jag väl också någonstans om det här med bra och dåliga tester, att man vill ha så lättförstådda och så enkla okomplicerade tester som möjligt. Om det är ett komplicerat flöde så blir ju testerna oftast komplicerade, men man kan ändå sträva efter att ha så enkla tester som möjligt. Vad gäller att bestämma vad som ska testas så är det nog ändå i första hand komplexiteten i det som ska testas som avgör, både i termer av komplexa algoritmer och affärsregler, det kan vara ganska enkel kod men många olika regler som spelar in. Ju mer komplext desto mer tester.

34. Du skrev någonstans att man ska "lyssna" på testerna. Vad menar du med det?

Det går ihop med att jag tycker att tester är mer ett designverktyg än ett verifikationsverktyg. Låt oss säga att jag har ett test som kräver 70 rader setup-kod för att få alla beroenden uppsatta och jag måste initiera 17 olika objekt med 10 parametrar var bara för att kunna testa att jag får ut "Hello World" på skärmen, då kanske jag har en dålig arkitektur som jag borde förenkla eller förändra på något sätt. När man börjar hamna där så gäller det att lyssna på att det är någonting som börjar bli jobbigt, ta ett steg tillbaka och börja fundera på vad det egentligen är man håller på med och om det går att förenkla. Kanske introducera något nytt objekt som tar hand om lite av det ansvar som jag håller på med nu, eller ändra ansvarsområden för de redan existerande objekten, istället för att bara köra på och pressa in ännu mer saker.

35. Hur tror du att det här hänger ihop med hur enkelt det är att köra tester?

Jag vet inte om det hänger ihop så jättemycket med att köra testerna, eller, på någon nivå gör det väl det.

36. Hur resultatet presenteras.

Ja, alltså...

37. Eller är det mer vid själva skrivandet av testerna som man behöver känna av sådant här?

Ja, fast det är både och, det finns olika nivåer på beroenden. När det handlar om att

sätta upp ett enskilt test så handlar det ofta om beroenden mellan klasser och objekt, där har det inte så mycket att göra med att köra dem, men man kan ju däremot ha beroenden på en högre nivå: ”Jag måste ha en Java-applikationsserver igång för att kunna köra de här testerna och den måste vara konfigurerad på det här sättet med de här inställningarna”. Det tycker jag också är ett beroende på någonting som kräver en massa setup, men då är det ju på en högre nivå. Jag tror att man vill undvika den typen av beroenden så långt som möjligt.

I idealfallet så tycker jag att man ska kunna checka ut koden och köra ett kommando så ska testerna köra. Det är inte alltid lätt att hamna där, men det är det som är målet, på samma sätt som jag tycker att 100 % test coverage är mål som man ska sträva mot så tycker jag att man ska sträva mot att ha tester som man kan köra när som helst utan att behöva starta upp servrar och liknande. Om man vill ha ordentliga integrationstester så är det oftast svårt, eller omöjligt, men det ska ändå inte kräva att man startar upp en specifik tomcat-server manuellt utan allt ska skötas av test runnern.

38. Jo, att automatisera kan verkligen vara guld värt, då blir det att man kör testerna oftare också, så att man verkligen får ut mer värde av dem, mer feedback. Vi pratade lite om hur det påverkar sättet man jobbar på, om man har tester jämfört med om man inte har det. Om vi tänker oss att du har en kund som tycker att testning är en bra sak, hur påverkar det hur du jobbar?

Om man bortser från alla andra faktorer, så tror jag att man kan jobba mycket mer med refaktorisering och kodkvalitet om man har en vettig testkultur, för att man kan vara mer säker på att förändringar man gör inte har sönder någonting. Det tror jag betalar sig i längden, för man kan hålla högre hastighet.

39. Man blir mer flexibel också.

Ja, den andra biten är att man förhoppningsvis får en bättre arkitektur som gör leder till ett mer flexibelt system. Man kan ta till sig nya krav istället för att vissa saker inte går att göra i det system som man har byggt. Ny funktionalitet kan fortfarande ta tid att implementera, men förhoppningsvis så finns det alltid en möjlighet. Det finns förstås en mängd andra faktorer som också spelar in för att man ska kunna åstadkomma hög kodkvalitet, men en bra testkultur är helt klart ett steg i rätt riktning. Man behöver fortfarande anstränga sig, men det blir enklare att göra det med testning.

40. Vad tror du om att använda tester som ett kommunikationsredskap? Både gentemot kund och nya utvecklare.

Det låter väldigt bra. *Skratt* Jag har aldrig upplevt att det har fungerat.

41. Om du tittar på någon annans kod och det finns tester, kollar du på dem först då eller försöker du förstå koden först?

Jag brukar kolla på testerna, men de brukar väldigt sällan vara skrivna på ett sätt som gör det uppenbart.

42. Det tänkte jag på innan när du sa det här med att man försöker täcka in så mycket som möjligt med sina enhetstester, att det skulle kunna finnas en vits med att skriva ganska enkla enhetstester som inte alls har ambitionen att hitta alla buggar utan istället

är tänkta mer som en specifikation.

Jag tycker att om man har ett vettigt testverktyg så kan man dra upp det på olika nivåer, både Buster och Jasmine har ju nästlade describe-block, så att man kan skilja grundfunktionaliteten från specialfallen. Det är något som jag vill bli bättre på för det är riktigt bra när det fungerar, som i vissa Open Source-projekt. Jag har aldrig sett ett kundprojekt där det har varit så.

RSpec och Cucumber har så att deras webbsite är deras features i ett mer webbvänligt format, vilket är häftigt. Vi har även boken Specification by Example av Gojko Adzic som tar upp mycket av det. Jag tror att det på vissa ställen skulle vara väldigt värdefullt att tänka så, i synnerhet ut mot resten av organisationen men även på enhetstestnivå. Jag vill träna på det mer och faktiskt kunna få till sådana tester som gör att man kan förstå hur allting fungerar direkt utifrån att köra spec-outputen.

Testning är sjukt svårt, jag har försökt köra någon form av testdriven utveckling i sex år och jag tycker inte att jag riktigt kan det ännu. Det kan vara så att dåligt skriva tester är sämre än inga tester, för de är ofta så beroende av implementationen att fördelarna av att kunna ändra på saker uteblir för att en ändring av implementationen gör att testerna slutar fungera. Då har du tester som bara är i vägen, och samma sak blir det när du inte lyssnar på dina tester. Du hamnar i att du har beroenden mellan allting, men i testerna så sätts allting upp och därför går de ändå igenom. Då blir det så att du ändrar en sak och allting ändå går sönder, så att man blir frustrerad över att behöva fixa tester som slutar fungera av helt orelaterade anledningar. I de lägena är det kanske bättre att inte ha några tester alls, och det var där jag var för kanske fem år sedan. Jag hittade kod från den tiden för ett år sedan där vi hade Ruby-tester med 160 raders setup-block för att sätta upp nästan alla objekt i hela applikationen. Jag minns att vi svor dagligen över de testerna för att de kändes i vägen och onödiga. Det är svårt. Det är ännu svårare i JavaScript, för där har man en DOM som har en tendens att komma in överallt.

43. Sista frågan var just vad du skulle ge för tips till någon som vill börja testa sin JavaScript.

Försök inte att göra det på redan existerande kod, är väl det första. I princip enda sättet du kan testa en jQuery-byggd site är med Selenium eller en liknande helt integrerad lösning. Det är inte värt att försöka gå in och skriva enhetstester med en DOM inblandad. Sen skulle jag ta det steg för steg, så att man skriver tester för de delar man är inne och meckar i.

Vad gäller att lära sig så har Katas fungerat väldigt bra för mig. Genom att köra kodkatas så kan man lära sig grunderna, som behövs för att man inte ska få det väldigt jobbigt. Börja med de enkla fallen, som att testa en strängomvändare och liknande. Då slipper du lära dig alla saker samtidigt, och kan istället fokusera på att lära dig testramverket först och sen lägga på saker allt eftersom. Det är också väldigt givande om du har någon som kan mycket som du kan sitta tillsammans med.

Börja någonstans, använd tester i egna projekt eller kundprojekt om det finns tillräckligt stöd från resten av teamet. Någonting nytt då, inte något som redan har massor med JavaScript, för då blir det bara jobbigt.

44. Jag tänker också på det nu när jag kommer in i Live, där det just nu finns en ambition om att börja testa mer. Hur det kommer att gå till där, vad tror du skulle vara lämpligt?

Det är svårt att säga allmänt.

45. Det jag känner spontant är väl just det att man ser till att skriva tester för det man går in och ändrar.

Läs refactoring-boken och jobba med att kontinuerligt förbättra kodbasen, för det räcker inte med att skriva ett test och sedan känna sig nöjd. Annars kommer du att ha en testsvit på flera hundra tester som tar flera minuter att köra lokalt och då kommer ingen att orka köra dem. Du kanske har en CI-maskin som kör testerna varje natt.

46. Selenium tänker jag försöka undvika, förutom för det absolut mest kritiska.

Det är oftast dit du måste gå om du inte börjar förbättra kodstrukturen, annars blir det ofta så att man inte kommer undan att behöva kolla i sin DOM någonstans. Jag ska inte säga att du inte ska vara rädd för att strukturera om kod, för det kommer man att vara, men gör det ändå. Det spelar egentligen ingen roll om det är .NET-kod eller JavaScript-kod, samma principer gäller ändå. Se till att automatisera allt, för annars kommer ingen att orka köra testerna.

Börja med inställningen att stoppa produktionslinan om testerna går sönder, som i Toyota-fabrikerna. Inte skjuta på att fixa tester som gått sönder. Det har med kulturen att göra, man måste få alla med på det. Det är problem jag haft på många ställen att folk säger att de vill göra det och är väldigt positiva när man pratar om det men sedan så märker man att vissa tester på byggservern har varit röda i nästan ett dygn och får ett svävande svar om att det ska fixas imorgon när man frågar varför det blivit så. Då hamnar man i att ingen litar på testerna efter ett tag, för att man inte vet om det är rött för att något är trasigt eller för att testerna inte har uppdaterats på grund av att någon inte förstod någonting. Så fort man får den osäkerheten så går testernas värde ner ganska hårt.

48. Tack ska du ha!

Transcript of Interview with Marcus Ahnve

1. Vad har du använt JavaScript i för sammanhang?

Webb, alltså webbapplikationer.

2. Mycket frontend, eller backend också?

Ja, nej, inte alls, jag programmerar allt möjligt, jag flyger lite fram och tillbaka. Jag kommer från backend, men har tvingats lära mig frontend också.

Senaste gången vi körde testdrivet JavaScript var hos en kund (av sekretesskäl kallar vi kunden för X) för tre år sedan. Då tog vi till oss idéer som... (oavslutad mening) Vi hade en del testdriven JavaScript hos en annan kund också (vi kallar kunden för Y).

Det var två relativt avancerade JavaScript-delar.

3. Vad var det som gjorde dem avancerade då?

Det var ganska avancerad vylogik, kunderna frågade efter saker som vi inte hade sett förut, som troligtvis inte hade gjorts tidigare av någon. Kund X skickade nyhetsmeddelanden som kunde gå iväg på två språk, engelska och svenska. När man öppnade ett sådant så skulle det komma upp en flik på sidan med svenska som förvalt språk och möjlighet att lägga till engelska. När man valde engelska så skulle en ny flik komma fram med nya fält, som skulle gå att ta bort.

Det svåra var i första hand att hantera Rails. Med de ramverk som finns idag så hade vi gjort på ett annat sätt, förmodligen med AJAX eller något enklare, men det fanns inte dokumenterat hur man kunde göra det vi behövde göra. Vad vi skulle vara tvungna att göra för att få det här att fungera med Rails var att inputfälten var tvungna att heta något eller ha ett index för att bli tolkat på rätt sätt av Rails. Vi förrenderade HTML och la i ett hidden-fält för att få tillgång till fältens namn, som bestäms av Rails namnkonventioner. När en ny flik skulle skapas, så kopierades allt och ändrades med reguljära uttryck utifrån det högsta id:t just nu och liknande.

4. Och det här gjorde ni testdrivet?

Ja, precis. Då hade vi varit på XP-konferensen i Trondheim (Agile Processes in Software Engineering and Extreme Programming 11th International Conference, XP 2010, Trondheim, Norway, June 1-4, 2010, Proceedings) där det var två som berättade om hur de körde testdriven JavaScript, vilket vi inte hade hört talats om innan överhuvudtaget. Problemet med testdriven JavaScript är ju DOM:en. De var de första jag såg som separerade DOM-access från resten av koden genom att lägga det i ett separat lager. De gjorde inga jQuery-anrop i affärslogiken, utan updaterade vyn som en separat sak. Det fanns då vy-lager där all jQuery och annat hamnade. Det var jag och Jimmy Larsson som satt där och jag minns att vi frågade oss varför vi inte hade tänkt på det förut. Man behöver inte skriva callback-soppa bara för att det är JavaScript.

En sak till som var ännu värre (än anpassningarna som behövdes för att de komplicerade vyerna skulle fungera med Rails) var att vi använde rich text editors, alltså för webben. Det som var grejen var att vi använda två editorer per flik. Jag minns inte vad det var för någon men det var en av de mer vanliga varianterna. Dels använde vi två på samma sida, vilket inte var speciellt vanligt, men framförallt så var det problematiskt att instansiera dem dynamiskt, det var det ingen som hade gjort förut. En sådan editor tar en textarea, gömmer den, och ersätter med ett gigantiskt JavaScript. Sen när du trycker på submit så finns den en master-klass som känner till alla de här, läser ut all text, populerar textarean och skickar submit. Det där var riktigt svårt att hålla ordning på, för det var helt odokumenterat. Det var inte så svårt när vi listade ut hur allt fungerade, men innan dess var det mycket problem med att se till att viss kod kördes överhuvudtaget. Just den biten var kanske inte testdriven, men det tillkom tester på det efter ett tag. Även fast det egentligen bara var textinmatning så blev det ganska komplext ändå.

5. Minns du hur ni testade det?

JsTestRunner körde vi.

6. Jag tänker själva testerna, hur ni...

Vi hade dem integrerade i Rake, genom att lägga till en test-target för JavaScript-testerna som kördes automatiskt. De gick väldigt fort. Jag minns inte om vi la till Evergreen (https://github.com/jnicklas/evergreen) på slutet eller inte. Det var då som Jonas Nicklas på eLabs i Göteborg byggde den här pluginen. Det var någonting med en browser inblandat som skulle startas varje gång testerna skulle köras. Jag tror faktiskt att vi ersatte alla tester efter ett tag med Jasmine.

7. Vad var det som fick er att göra det?

Jasmine är mycket mer läsbart än JsTestRunner och ganska snyggt gjort i jämförelse. Nuförtiden så är folk rätt förtjusta i Jasmine, jag tycker personligen att Buster ser lite bättre ut.

8. Mm, och det var just för att få mer läsbara tester?

Ja, precis. JsTestRunner gjorde att testerna blev väldigt tekniska. Framförallt i JavaScript är verktygen viktiga och där är det stor skillnad idag, att det numera finns vettiga ramverk som CanJS och Angular, som ger en tydlig uppdelning för vad det är du testar.

För tre år sen var det fortfarande mycket hackande av callback-soppa. När folk skrev tester i JavaScript så var inte det mycket annat än att man kickade igång en browser och kollade att rätt sak hände, det fanns ingen separation mellan event och action. Det är den stora skillnaden idag, nu tror jag att det skulla vara lättare att skriva tester. På den tiden var det mer så att om man hade varit riktigt duktig så hade man kunnat komma på att skriva ett CanJS själv *skratt* men det gjorde vi inte.

9. Svårt att motivera gentemot kunden också kanske?

Äh, givet hur mycket tid vi la ner så. Jag är inget vidare på JavaScript, men har gjort ganska avancerade grejer i det ändå måste jag säga.

10. Den här konferensen, var den under tiden som ni höll på med det här?

Nej, det var precis innan. Det var rent flyt, vi insåg fördelarna med att testa JavaScript och bestämde oss för att göra det.

11. Vad hade ni för nytta av testerna?

Trygghet, regression, det gamla vanliga. En känsla för att de viktigaste sakerna fungerade. Även att tvinga fram vettig modularisering, det är ofta det som jag tycker är en av huvudfunktionerna med tester. Om man upplever att något inte går att testa så är det i princip uteslutande så att det är fel på koden.

Det var rätt svårt, jag minns att vi hade en kille som i övrigt är en väldigt duktig utvecklare men som hade svårt att bryta upp sättet som han skrev JavaScript på. Han chokade fullständigt, sa att han inte fattade och började skriva callback-soppa istället. Det var det han var van vid, det var så JavaScript skulle se ut. Function, function, function, function...

12. Hur tror du att trenden var, för några år sen, har det förändrats nu med de nya verktygen?

Ja! Jag har inte någon särskilt positiv syn JavaScript-communityt så som det såg ut, det har tummats mycket på kodkvalitet och det har funnits en attityd som gått ut på ungefär "skit i det, det funkar". Jag skulle säga att testdrivet kom från Ruby-hållet, från backend-programmerare som klev in i frontend-programmering. Det stämmer väl med många inom Rails-communityt, Rails har ju väldigt mycket tester och när de började skriva sina skript-grejer så insåg de att de inte hade några tester för det och Jasmine kom av...

13. RSpec?

Precis, och de som skrivit Jasmine är... är inte det FortBot? Jo det är det. (Pivotal Labs?) Alltså, det är ju Rails-gäng. Väldigt många är det så med: BusterJS, PhantomJS, allt det där kommer från folk som har hållit på med Rails och är vana vid att skriva tester. Det existerande JavaScript-communityt har fortfarande en del folk som till exempel person A (onödigt att nämna namn) som inte är så intresserad av tester så länge koden fungerar. Så det beror helt på var man kommer ifrån.

14. När jag började med mina efterforskningar så fick jag intrycket från allt jag läste att det inte testades mycket alls och sen när jag pratade med Johannes Edelstam så lät det på honom som att det börjar komma igång nu.

Jo, Johannes, som för övrigt var med i projektet för kund Y, är ju väldigt duktig. Ska jag vara helt ärlig så är det ju oftast så att om du åker ut och tittar på vissa projekt som drivs av mindre ambitiösa företag så ser du mindre av testning. Jag tror att det handlar om varifrån du kommer, om man mest har ägnat sig åt HTML, CSS och PHP så har man troligtvis inte skrivit så mycket tester, kommer man från Rails, backend och är van vid det och dessutom skriver frontend då funderar man mer över var man ska lägga testerna.

15. Frågan är då ifall det har förändrats

Ja, jag tycker att det har förändrats. En skillnad är att det överhuvudtaget finns. Jag tittar på Angular som säger att det går jätteenkelt att testa. CanJS, som jag tycker bättre om, är också väldigt enkelt att testa. För mig är det mycket positivt.

16. Vad tycker du kännetecknar bra tester?

Utan att läsa utifrån någon form av definition av bra tester som Pragmatic Programmer.

17. Utifrån din erfarenhet

Jag tycker att tester ska vara snabba, små och testa en sak. Idealet är en assert per test, men det behöver inte alltid vara så. De ska vara meningsfulla, det finns en del människor som skriver tester som inte är det. Deskriptiva, jag är förtjust i idén bakom BDD, att testerna ska beskriva ett beteende i systemet, varför det gör saker. Folk som inte tänker i BDD-termer har en tendens att skriva tester som testar metoder, till exempel testAddition.

18. Vad är det du menar när du säger att tester ska vara meningsfulla?

De måste ha någon form av relation till det affärsvärde som systemet löser. De måste på något sätt beskriva någonting. Ett exempel skulle kunna vara att någon ska skriva en metod som heter add(x, y) och bestämmer sig för att skriva ett test som heter testAddXY med tre asserts som kollar att 3+2 blir 5 osv. BDD-sättet är ju då snarare att säga att 2+3 ska bli 5, det är ett exempel. Fördelen är att när ett test failar så får du ut vettiga felmeddelanden. Om du har ett felmeddelande som säger testAdd failed så är det svårt att veta varför det failade och man måste lusläsa testkoden för att förstå. Just testAdd kanske är lite väl rudimentärt, men det kan till exempel vara testUpdateUser failed, då undrar man vad som menas med att uppdatera en användare. Om testet däremot heter "User should be saved to database" så är det mycket tydligare vad som gick fel.

19. Det är väl också mycket det att man har möjlighet att specificera förutsättningarna: Given, When och de där.

Ja, kontext är väldigt bra, det är också en fördel med BDD. Särskilt i Java kan den delen saknas i traditionella TDD-ramverk, det är inte ett lika stort problem i Python och Ruby, för då kan du skriva fler testklasser i samma fil. Då är det inte ett lika stort problem att du är begränsad till en setup och en teardown per test. I Java har du ofta så att det är en klass som mappar fram och tillbaka, då har det åtminstone för mig inneburit problem att det bara finns en setup.

20. Den blir ganska stor

Ja, jag kan göra en koppling till J. B. Rainsberger, som har skrivit JUnit Recipes: Practical Methods for Programmer Testing, en av de bästa böckerna om jUnit, för nästan 10 år sedan. Han körde en dragning på XP 2011 i Madrid om hur man jobbar med BDD i vanliga jUnit. Hans poäng är att man ska sluta testa klasser och testa beteenden på system istället, möjligen implementerat genom att testa klasser men i första hand så ska man sträva efter att beskriva systemets beteende. Det är egentligen det som jag har velat göra med Cucumber, att få hjälp med att testa utifrån. En liknelse jag brukar göra är att du ska undvika att sitta bredvid processorn med sladdar och grejer och tänka att du testar tekniska saker för att du är en tekniker. Det är mycket intressantare vad systemet gör för att fylla affärsvärde.

Jag börjar luta åt att vi bör skriva mer och mer enhetstester som beskriver affärsvärden men som implementeras på enhets-nivå. Jag tror att det finns problem i att vi bara beskriver affärsvärden i integrationstester och att vi fokuserar för mycket på implementationsdetaljer och lågnivå-saker i våra enhetstester.

21. Hur uppnår man det, jag tänker att man behöver vara rätt så konkret när man skriver enhetstester? Exemplet tidigare om att uppdatera en användare i databasen är väl ganska bra i och för sig.

Det finns ju en poäng med integrationstester, men de ska nog i första hand vara happy path och smoketest, men det ska mest handla om ifall grejerna sitter ihop. Jag kan fortfarande bli irriterad på gamla "Uncle Bob-människor" som tycker att man aldrig ska testa GUI:t. Varför inte då, om det är det du vill testa? Om GUI:t är viktigt så tycker jag att man ska testa det. Min gissning är att attityden beror på att man är inlåst på

att man skriver en viss sorts system, och generaliserar för mycket.

De som började skriva TDD en gång i tiden jobbade i första hand i finansbranchen, där var GUI:t oviktigt och affärsregler oerhört viktiga, så de skrev mycket tester för just affärsregler. Sedan så kom TDD ut i webbvärlden där affärsreglerna inte är så framträdande men istället så har man en stor mängd flöden, som är viktiga. Att då utgå från att man inte ska testa GUI:t blir närapå absurt, för resten är ju ofta närapå trivialt så då behöver man knappt några tester.

Att man behöver enhetstester är en annan sak. Om vi tar CanJS som exempel, där du kan ha en explicit vy som tar en template och genererar HTML. Det går att testa genom att parsea den genererade HTML-koden och se om specifika värden dyker upp. Man kanske behöver mocka vyn, men i övrigt så är det precis så man vill ha det.

22. Jag har bara provat Angular än så länge, jag ska nog kika lite på Can sen.

Can är snyggt och jag tycker om det för att det är så modulärt. De har routing, som går att använda helt oberoende av allt annat i ramverket. Om man vill använda deras controllers så går det också, och blir enkelt. Det tog lite tid för mig att förstå deras dokumentation därför att när jag läste om deras routing så tog det ett tag att komma till avsnittet om hur man gör om man även vill använda controllers, och då kändes allt mer intuitivt. Samtidigt är det bra att de separerat på det viset, eftersom det blir mer modulärt, lättviktigt och fint, vilket jag tycker om.

Jag har tittat lite på Angular och fått det beskrivet för mig. Den bild jag fått är att det är rätt så mycket ”all or nothing”, det går inte att plocka bara de delar man behöver och vill ha. CanJS skiljer sig på det sättet att du kan använda det för att strukturera upp din jQuery-kod med lite mustache-templates och sen är det bra så. Sen är det snygg kod, om man dessutom lägger in requireJS så blir det ruskigt snyggt.

23. Drog ni några lärdomar från projektet med kund Y?

Ja, lärdomen var väl att när man väl såg Backbone och liknande komma så insåg man rätt snabbt att de fyllde behov som man tidigare behövde brottas med. När man skriver stora mängder JavaScript-kod så blir det oerhört viktigt att den är välstrukturerad. Johannes kom in i det projektet och fixade saker som hade med JavaScriptet att göra långt efteråt att det hade skrivits. Han sa att det hade hjälpt honom att det var testdrivet.

Vi tog över projekt från en annan organisation och det var en märklig upplevelse, ungefär som att han som hade skrivit koden hade läst en kurs i hur man strukturerar JavaScript men inte förstått varför man ska göra det. Det var inte speciellt svåra saker som koden skulle göra, man hade bara gjort dumheter såsom flera hundra rader kod enbart för validering av fält.

Klasser som dolde funktioner och publicerade publika grejer i slutet, jättestrukturerat och jättefint bara det att när man till slut lyckades ta sig igenom det här så insåg man att det ändå var hårdkodat mot specifika fält och det gjorde det bara jobbigt, det hade varit enklare att hantera en callback-soppa, då hade man i varje fall kunnat gå in och dra isär den. Det fanns naturligtvis inga tester, och det slutade med att vi var tvungna att göra ganska mycket fulhack. En av de fulaste saker jag gjort någonsin faktiskt var då när jag gick in mitt i hans 300-rader långa funktion och la till ett villkor att valideringen

endast skulle köras om ID:t hade ett visst värde. Det blev hårdkodat för att jag inte orkade bryta ut det när det var så fruktansvärt mycket kod.

Andra grejer han hade gjort var att en knapp tändes på onBlur om ett visst inmatat ID var korrekt. När kunden kom med ett önskemål om att fältet skulle vara förifyllt så försökte vi lösa det genom att skicka in det med en url-parameter och populera fältet men återigen hade vi situationen att det var hårdkodat. Så vi tog faktiskt page.onLoad och anropade onBlur. Jag reagerade med att fråga mig själv om man överhuvudtaget fick göra så, är det lagligt? Det gick bra och löste problemet men det kändes ju inte bra. Det var ett praktexempel på hur JavaScript-communityt kan fungera. Han ville nog väl men det blev väldigt fel.

Det finns ett problem med folk som kodar JavaScript och aldrig har sett någon annan kod, som definierar sig själva som frontend-utvecklare och har attityden att de inte rör backend-kod. Jag tror att det är en nackdel för kodare generellt sätt, för hur du kodar. Det är nyttigt att få se andra språk, kunna använda det och få den inputen. Om du går in för att enbart läsa och skriva JavaScript i resten av ditt liv så kommer du att få så begränsad input att du inte förstår varför du ska strukturera din kod och utan de insikterna så blir det inte bra även om du försöker.

24. Så budskapet blir att det är bra att strukturera sin kod, men att man behöver se till att man vet varför man gör det.

Ja, absolut.

25. Ett liknande resonemang är att det är bra att testa, men att man bör se till att man vet varför man gör det. Se till att alla vet varför.

Ja, men samtidigt så beror det på. När du är nybörjare så saknar du den kontextuella kunskapen som krävs för riktig förståelse. Då får man finna sig i att göra som man blir tillsagd, som i kampsportsträning där du som nybörjare inte ska ifrågasätta varför din tränare säger åt dig att slå på ett visst sätt. Det är först när du kommer upp i en högre nivå och har gjort något länge som du kan börja ifrågasätta och till slut även komma på egna förbättringar. Att bara testa när man förstår varför gäller egentligen bara de utvecklare som vet varför de ska testa. Om du aldrig har testat så kan du inte avgöra när du bör göra det och när du inte bör göra det. Jag skulle föredra att man alltid testar mot att man aldrig gör det, tills man kommer till den punkt att man vet när man inte ska göra det.

Idealet är förstås att sätta nybörjare med erfarna människor, så kan de erfarna bestämma och förklara varför det ser ut som det gör. Det blir många iterationer och mycket jobb om en nybörjare ska behöva återuppfinna hjulet helt själv.

26. Vad är din syn på testning av JavaScript jämfört med andra språk, är det någon skillnad i hur du skriver tester?

Ja, det borde vara det. Hittills har det nog inte varit det, jag har jobbat väldigt objektorienterat och jobbar nu gradvis med att bli en bättre människa och skriva mer funktionell kod. Förhoppningsvis kommer det att leda till att jag skriver mer funktionellt baserade tester, och därigenom slippa hålla på och instansiera klasser och liknande. Annars är det inte så stor skillnad, jag försöker att tänka likadant.

27. Har du upplevt att det varit svårare eller enklare?

Om det har varit svårare så har det har nog mest att göra med att jag är bättre på TDD än på att skriva JavaScript. Nuförtiden behöver det inte vara någon skillnad, för tre år sen var det definitivt svårare för det fanns inte några exempel att titta på. Det var inte lika väl dokumenterat hur man hanterar specifika fall och det fanns mindre vettiga ramverk. Förr så hade du en massa jQuery som kördes på document.onLoad och då var det verkligen inte lätt att veta hur man skulle bära sig åt för att testa den koden. Det var märkligt att ingen kom på idén att ta fram ett MVC-ramverk för att lösa problemen.

28. Och sen exploderade det med sådana ramverk.

Ja, precis.

29. Vi pratade lite om ambitioner med testning tidigare, om man väljer att fokusera på integrationstester eller enhetstester, min fråga som jag hade här sen innan var: vilka brukar vara dina ambitioner när det kommer till testning?

Jag brukar testa utifrån och in. På så sätt kan jag identifiera vad det egentligen är jag vill göra. Det behöver inte utgå från klienten, däremot så strävar jag efter att tänka i termer av APIer, vad varje del ska göra.

Ett praktexempel var en gång när jag var med och arrangerade en konferens för Agila Sverige, då vi skulle göra en schema-snurra. Jag gjorde det klassiskt objektorienterade misstaget att jag började modellera upp en jättesnygg objektmodell för schemat som skulle gälla i två dagar med två parallella tracks, med olika tider och lokaler. När jag väl skulle börja använda modellen i HTML så insåg jag att det var helt värdelöst. Jag blev så arg på mig själv, jag höll ju kurser i BDD och annan testdriven utveckling och gjorde det nybörjarmisstaget själv. Det jag borde ha gjort var att börja skriva gränssnittet och fråga mig vilka APIer jag vill ha, istället för att börja med något i förhoppningen om att det kommer att behövas. Det tankesättet går att tillämpa generellt när det gäller utveckling.

30. Så de första testerna du skriver är...?

Jag skulle göra som så här, att om jag skriver en HTML-sida så börjar jag med att skriva HTML-koden snabbt och enkelt och utan tester. Sedan kommer jag till en punkt då jag inser att jag skulle vilja ha ett API-anrop, idag om jag får välja så händer det i en mustache-template. Då skriver jag kod som anropar något som inte finns, och ett test för det. När man väl kommit dit så vet man att det är en komponent som behövs och då kan man börja skriva den, och även testa den, eftersom man då vet vad den ska göra.

31. Man har ofta som princip att alla tester ska gå igenom hela tiden. Det finns många bra orsaker till det, man ska kunna lita på testerna och så. Samtidigt kan jag känna att om man har det här tillvägagångssättet att man jobbar mer top-down, så skulle man vilja skriva tester som är på hög abstraktionsnivå och i princip kräver att systemet är klart innan det går igenom.

Absolut. Så gör jag jämnt. Och om du tittar på gammal klassisk XP-terminologi så kan

ett integrationstest faila ganska länge. I RSpec så finns ett koncept som heter Pending, man låter den vara grön till exempel genom att säga att den ska faila därför att det inte är klart. När du väl är färdig så behöver du ändra på testet. Alternativt så kan du lista den som Not Done eller något. Pending tycker jag är bra, för då får du bra indikationer, men sen så ska du väl inte ha så värst många pending liggandes.

32. Det kan vara bra för att få en tydlig riktning kan jag tänka mig.

Jag tycker det, jag tycker att det är en av de bästa sätten att jobba på. Däremot så kräver det ganska mycket disciplin. Om du ska skriva en helt vanlig HTML-sida eller gör något som i princip enbart är boilerplate och du vet med dig att det kommer att vara många ögon på den koden i projektet så kanske det inte är värt att skriva ett långsamt integrationstest för det.

Det är väldigt trevligt med mustache och liknande som gör att du får ut logiken i riktig kod istället för att du har en massa märkliga if-satser i vyerna.

33. Du nämnde att det kräver mycket disciplin. Det har ju lite att göra med ens personlighet som utvecklare, tror du att testning är något för alla? Om man inte är en sådan person som har den disciplinen.

Nej, det tror jag inte. Jag tänkte säga ja, men sen insåg jag att det är fel. Om vi tar Notch (Markus Persson, skapare av Minecraft) som exempel, han skriver inga tester. Det vore fruktansvärt drygt och dumt av mig om jag sa att han borde skriva tester. Nej jag tror inte att det är för alla, jag tror att det går att skriva alldeles utmärkt bra kod utan det, däremot så tror jag att du skriver bättre, mer underhållsbar och mer modulariserad kod med tester. Det kan mycket väl vara så att du är en tillräckligt duktig programmerare, som skriver modulär och vacker kod även utan tester, men för mig är det i varje fall så att det är ganska uppenbart när folk inte använder tester.

Ett praktexempel är Android, det är uppenbart att de som gjorde Android-API:et inte använde sig av tester, för då hade de inte gjort det så som det är gjort.

34. Vad är det man ser det på, rent konkret?

Det är en gigantisk arvshierarki, som gamla klassiska J2EE, där du inte kan instansiera en servlet med mindre än att du har en applikationsserver tillgänglig eftersom den krävs längre upp i arvskedjan, eller JSP-sidor som ska kompileras av en parser som är inbyggd i din applikationsserver. Det här kom att utmanas av Freemarker som fungerar mer fristående. Mata in data, ange vad för template du vill ha och så får du HTML som du kan göra vad du vill med.

35. Lite enklare att testa.

Väldigt mycket enklare att testa, väldigt mycket enklare att förstå vad den gör. Det finns fantastiskt många människor som vill hjälpa stackars programmerare att inte förstå vad det är som händer.

36. Vilka ser du som de största riskerna med testning då? Om vi fortsätter på spåret om vilka som borde testa och inte. Det beror kanske lite på vilket projekt det är.

Det finns två risker med testning, det ena är att man litar för mycket på det, att man

tror att det räcker med att skriva tester och att man inte behöver göra något annat. Att det gör all manuell testning överflödig. En annan risk uppstår i och med att tester precis som all annan kod blir gammal och dålig om du inte underhåller det.

Jag kom till ett projekt med Microsoft-utvecklare en gång där chefsarkitekten mycket stolt sa att de hade 2000 fitness-tester. Vad trevligt. Vad roligt. Och så skulle jag köra dem, och fick 1000 fails. När jag frågade varför så svarade han att testdatabasen inte var ordentligt uppsatt. Då frågade jag när den senast var uppsatt så att alla tester fungerade och det var det ingen som mindes. Då är det bara slöseri med pengar. Det hade varit bättre att inte skriva testerna överhuvudtaget.

Sen har du det att du behöver ha en vettig testmiljö, som fungerar. Det är inget fel på Microsoft-utvecklare, men de har en tendens att göra vissa saker på fel sätt ibland, som till exempel att alla delar på en databas för att man inte orkar sätta upp SQL Server till alla. Det kan uppstå absurda situationer där en utvecklare meddelar de andra att testerna kommer att sluta fungera för att personen ifråga ska lägga till en tabell eller ändra namn på en kolumn. Det beror ofta på att det är jobbigt att ge alla varsin och att de inte får byta till en annan databas.

37. I de projekt som du håller på med nu, skulle du vilja använda testning i alla?

Ja, absolut. I mitt nuvarande projekt så är det inte överdrivet mycket JavaScript just nu men det kommer nog att bli det så jag hoppas att det börjar testas även om jag går vidare till ett annat projekt.

38. Jag har tänkt göra någon slags analys av ekonomiska aspekter av testning, hur mycket det lönar sig, men det är lite svårt att mäta tycker jag.

Det är sjukt svårt, jag kan bara gå på magkänsla. Det bästa exemplet jag har är från när jag jobbade på en startup för ganska så länge sedan, där vi sålde mjukvara som skulle köras på Websphere-servrar, på ett visst antal databaser och ett visst antal operativsystem. Vi hade i princip 100 procent test coverage, och testservrar uppsatta med olika konfigurationsvarianter som kördes vid varje bygge och gav oss en tabell över vilka tester som gick igenom på vilka maskiner. Vår VD sa att det var vår bästa investering någonsin, för han visste alltid var produkten var vilket gav en enorm trygghet när han gav sig ut för att möta kunder. Det är i princip omöjligt att sätta fasta pengar på det, det är många som inte ens vet om att deras utvecklare testar.

Det har mycket att göra med vad som händer om det uppstår ett fel, graden av allvarlighet om en bugg uppstår i produktion. Om du har ett system som styr kylvattnet i ett kärnkraftverk, navigering av flygplan eller liknande så är det inte möjligt att förlita sig på enhetstester och continuous delivery. Man måste ha både automatiska tester och rigorösa manuella testningsrutiner, med flera olika personer som tittar på det regelbundet och med olika perspektiv, annars blir det livsfarligt.

Kund X är ett praktexempel. Samtidigt som vi hade vårt projekt hos dem så hade vi ett projekt hos en mediekoncern där det förelåg ungefär samma ekonomiska risk kopplad till buggar. Med systemet vi byggde åt kund X så skickades det ut finansiella nyheter åt börsbolag, vissa av dem var noterade på Nasdaq OMX vilket innebar att varje finansiell nyhet enligt lag var tvungen att nå ett visst antal människor, vilket var

anledningen till att systemet fanns överhuvudtaget. Det stora med det här systemet var inte att vi la ut information på vår egen webbsida utan att det skickades till 7000 ställen: finansinspektionen, dagens industri, bloomberg, ... Allt gick ut till alla ställen. Man kunde ange exakt tid då ett pressmeddelande skulle skickas. Om det skulle gå fel så innebär det rejäla böter för att marknaden inte blivit informerad, för det är så stora pengar inblandade.

Bortsett från att vår produktägare hos kund X var inne och tittade inför varje release som ett smoketest så hade vi ingen manuell testning med protokoll eller liknande. Vi hade en testmiljö och en produktionsmiljö, med möjlighet att rulla tillbaka ändringar från produktion vid behov. Att släppa ut till produktion gick väldigt fort. Jag levererade koden och så hade de en driftavdelning som insisterade på att göra saker själva, som ofta gjorde fel. Men det hade egentligen inte med vår testning att göra. Då litade vi på testerna.

Att jämföra med historien hos mediekoncernen, där projektledaren var en gammal testledare. Man ska alltid komma ihåg att tänka på vad folk bryr sig om när de träffar sina "peers" (kamrater, jämlikar). För det pinsammaste man kan säga på en presskonferens som testare är ju att man givit ut ett system med odokumenterade fel. Den ekonomiska risken i det här projektet var i första hand att man kunde gå miste om prenumerationskunder, vilket troligtvis var mindre allvarligt än i fallet med kund X. Ändå så hade de en process som i varenda release låg i form av en "hardening sprint" som gick på att testa i två veckor så att allt var perfekt innan det gick iväg. En av mina kollegor som jobbade i det projektet sa att det var som att sitta i en Ferrari och köra i 30 km/h.

Som affärsutvecklare måste man räkna på vad en bugg kostar i jämförelse med att skjuta på releasen i två veckor. Alternativet är att släppa funktionalitet så fort som möjligt och rulla tillbaka om det uppstår en allvarlig bugg. Det finns en felaktig konservatism i många delar av vår bransch som säger att det inte får finnas buggar. Om man frågar en person med den inställningen varför man inte bara kan rulla tillbaka så tenderar man att få som svar att det är för mycket jobb inblandat med att göra en release, och anledningen till att det är så mycket jobb är att man är så noggrann för att undvika att det blir fel. Ett cirkelresonemang som leder till att varje release blir jobbigare än föregående. Jag tycker att det ska vara löjligt enkelt att göra releaser, så enkelt som möjligt. Naturligtvis finns det exempel som banker och liknande verksamhet där man kan förlora extrema mängder pengar på en allvarlig bugg, där man behöver vara mer försiktig men i allmänhet så är det så här.

Jag noterade igår att SEB lyckats deploya en version av sin Internetbank som det inte gick att logga in på med dosan, enda sättet var om man hade mobilt bankID. De kan inte ha haft tester. När man matade in sitt personnummer och tryckte tabb så fylldes password-rutan med personnumret, hoppade du ner och skrev i passwordrutan så fylldes personnummerrutan. Det kan inte ha varit testat överhuvudtaget, inte ens manuellt. Nu var det här förvisso Chrome på Linux men det lär ändå ha varit så att det inte fungerade på något Chrome och då blir jag ännu mer nervös om det är så att de bara testade på explorer. Man vet aldrig, vår bransch upphör aldrig att överraska.

39. Jag tänker att det här kan ha att göra med val av verktyg, att man behöver ta med i beräkningarna om de möjliggör testning på olika plattformar. Med Selenium kan man

parallellisera upp sina tester på en grid och olika test drivers är olika kompetenta när det kommer till det där.

Jag har inte haft tillfälle att bygga upp någonting stort, men i en större organisation så skulle jag nog vilja sätta upp någonting sådant.

40. Jag har tänkt att jag kanske borde intervjua personer med mindre koppling till valtech delvis av den anledningen.

Här på valtech så har jag i första hand testat i Chrome och Firefox och sen fixat till det till Explorer på slutet. Förhoppningsvis så är vi av med IE7 snart, men det är ju ändå inte förrän i IE10 som det börjar bli bra.

Ibland kan man lyckas, vi lyckades fantastiskt bra hos kund X och det var ändå tre år sedan. De skrev "optimerat för Chrome, Safari och Firefox", jag ville att det skulle stå "byggt med moderna webbstandarder". På den tiden fanns IE6 fortfarande i betydligt större utsträckning än idag och det vi gjorde var att vi lät rundade hörn vara kantiga och även om det här inte har med tester att göra så är det återigen en kostnadsfråga. Det blir en affärsfråga att avgöra om det är värt att sitta och lägga in gif-bilder för varenda bild.

41. Som ska transformeras och positioneras så att de hamnar rätt, och fungera i alla operativsystem.

Ja, eller så använder du CSS för rundade hörn på ett sätt som inte fungerar i gamla versioner av explorer. Då kanske kunden svarar att det inte går för sig och tycker att det är viktigt trots att det bara gäller 2 procent av besökarna, då brukar det hjälpa att komma med en uppskattning av vad det faktiskt kostar att implementera dessa rundade hörn ordentligt, det brukar få kunden att säga "de får fan ha kantiga hörn".

42. Då gäller det att kunna säga vad det kostar.

Ja, men det är inte så svårt. Det blir ju aldrig exakt, men man kan ändå säga att om man ska hålla en sådan sak vid liv genom hela projektets gång så kan det mycket väl innebära att det läggs totalt fyra veckors arbetstid på att se till att de rundade hörnen ser ut som de ska i IE6. Då brukar reaktionen vara att man inte vill ha det.

43. Det är intressant det här med kulturen, att det gick bra med kund X men att projektledaren med testarbakgrund hos mediekoncernen kunde ställa till med sådana problem.

Ja, jag vet ju inte hur de gjorde med just rundade hörn, men...

44. Nej men just med processerna vid release.

Ja, man ska alltid komma ihåg att folk bryr sig väldigt mycket om sitt eget skrå. Vad kommer folk vilja visa upp nästa gång de byter jobb? En AD kommer alltid att vara fokuserad på att det ser bra ut, om de gör något som inte ser bra ut så kan de inte visa upp det, UX:are samma sak. Testare vill inte ha något med buggar. Alla vill göra sin grej och har sin yrkesstolthet.

Egentligen är allt underordnat målet att tjäna så mycket pengar som möjligt. Folk har mer eller mindre svårt att förstå det. Jag har sprungit på UXare och ADs som hävdar

att det de gör är affärsvärdet. Nej det är det inte, svarar jag då. Jo, det är det, det är ju vad folk upplever, användarupplevelsen är viktigast av allt. Nej det är den inte. Jo, men annars kommer folk inte att tycka om det! Jo men då är det dåligt, då börjar vi prata ekonomi, men huruvida folk upplever det som vackert är egentligen ointressant eller i varje fall underordnat huruvida de sen bestämmer sig för att betala pengar för det och att den som tillhandahåller tjänsten tjänar något på det. Nu har vi kanske lämnat JavaScript-testning lite långt bort.

45. Det är ändå intressant, just kulturen och det tänker jag nog skriva om i uppsatsen att folk behöver vara överens om hur mycket man ska testa, vad det är man ska testa, så att det inte blir konflikter inom teamet och med kunden.

Det kommer det att bli ändå. Jag har väldigt svårt att se ett JavaScript-projekt som lyckas på det sättet. Här på valtech kan man nog få till det ganska bra i och med att det finns gott om JavaScript-utvecklare som har förstått och vet vad testning handlar om. Om du däremot tittar på ett godtyckligt normalt frontend-projekt så kommer det oftast att finnas svårigheter att driva igenom en testningskultur, eftersom JavaScript-communityt har sett ut så traditionellt sett. Den testkultur som finns kommer från folk som har programmerat mycket i andra språk, till största del i varje fall.

46. Vad skulle du ge för tips till någon som tycker att det är riktigt svårt, som menar på att det inte går att testa gränssnitt? Att det bara blir långsamt och jobbigt?

Jobba med någon som vet hur man gör. Jag tror inte att det finns några riktigt vettiga böcker om att testa JavaScript så antingen det, eller så låter man helt enkelt bli.

47. Vad tror du om kod-katas?

Grymma, det är bara att köra. Det är faktiskt intressant, jag har nog aldrig sett en kodkata gjord i frontend-kod, det kan jag ta upp med Emelie, det är en bra tanke.

48. Då tycker jag att vi avslutar med den tanken.

Transcript of interview with Per Rovegård

1. Vad har du för erfarenhet av testning av JavaScript?

Du syftar på automatiserade tester antar jag. Jag har jobbat på ett JavaScript-projekt senaste året med AngularJS, med ett antal 1000 JavaScript-tester som använder Jasmine-ramverket. Dessförinnan har jag ägnat mig åt Node.js med Mocha.

2. Har du så att alla de testerna går igenom när ni kör dem?

Javisst, annars är det inte ett riktigt bygge.

3. Jag har hört historier där det inte varit så, det har varit ganska spännande.

Ja, det är ganska vanligt att inte alla tester går grönt, men vi har gjort så att det inte ska gå att deploya utan att testerna går igenom.

4. Vad tyckte du om Mocha versus Jasmine?

Det var ett tag sen jag använde Mocha men jag vill minnas att de är ganska snarlika. På senare tid har jag blivit lite trött på Jasmine så jag har varit sugen på att byta till qUnit eller Mocha.

5. Vad är det som gjort att du blivit trött på Jasmine?

I Jasmine har man nästlade describes och its och med stora testfiler och många tester som körs så kan det bli ganska mycket till slut. Det enda du har stöd för är att exkludera specar. Det är inte överdrivet komplicerat att hacka ihop sätt att köra enskilda describes, men det har inte tagits in i kodbasen till jasmine och det har gjort att jag upplever det som att utvecklingen har blivit fientlig mot nya features från communityn som till exempel den gamla patchen för ddescribe/iit som du säkert sett och en del annat såsom beforeAll.

I andra testramverk finns ofta möjlighet att göra en fixture-setup, som körs en gång för alla före samtliga tester. Man kan föra en filosofisk diskussion kring huruvida det är bra eller dåligt. En fixture-setup tvingar dig att skriva tester som inte påverkar varandra i termer av delat state. Det leder också till att testerna går snabbare att köra, vilket kan vara användbart för tester som har en setup som tar lång tid att köra. I Jasmine finns inte beforeAll eller någon motsvarighet och de försök att komma runt det som finns fungerar inte så bra.

Nu har jag förvisso inte använt Jasmine 2, utan enbart 1.3 alltså senaste, men de har en ny på gång och det är möjligt att de har fått in det där men jag har inte sett det i så fall. Problemet med existerande lösningar som extendar Jasmine för att få beforeAll uppstår när du nästlar och inte har beforeAll i hela kedjan "uppåt".

Detta har lett till att jag funderar på att byta till något annat.

6. Jag har använt Jasmine med JsTestDriver och delat upp testerna i olika filer och därigenom hadft möjligheten att välja vilka filer jag vill köra.

Ja just det. Men det är fortfarande så att om du har en testfil med 100 tester i och du ska debugga ett fel, så kanske du bara vill köra ett av dem.

7. Har du upptäckt någon fördel med Angular ur ett testnings-perspektiv?

Fördelen är att Angular-gänget gillar tester vilket innebär att det är tämligen enkelt att testdriva sin kod. Min upplevelse har varit positiv förutom två saker. Angular använder en IOC-container, dependency injection för services och annat, vilket generellt kan vara komplicerat när det kommer till testning. Du har en massa regler för hur saker instantieras som i viss mån kan konfigureras men i många fall följer defaultregler vilket kan leda till konstigheter i testningen genom att du inte får injicera på det sätt du vill. Dessutom är det väldigt tight kopplat till just Jasmine, AngularMocks är inne i Jasmine och stuvar om för att göra det smidigare och enklare att testa men det betyder också att du binder väldigt mycket till just detta testramverk.

8. Det som hette Testacular innan, Karma, använder ni det?

Det finns en nackdel med Karma för oss som har över 3000 tester. Vi använder en testdriver som heter Chutzpah med fördelen att den kan köra tester parallellt. Säg att vi har våra tester uppdelade i ett 30-tal spec-filer, då kör den dem parallellt vilket

innebär att det blir en ganska snabb testkörning. Vi kör alltihop i PhantomJS. Fördelen med Karma är ju att du kan ställa in och köra i Chrome, IE, Firefox, osv. men de har ännu inte stöd för att parallellisera. Senast jag kollade, före sommaren, så fanns en issue för att fixa det stödet men jag vet inte om det har kommit ännu. Det är såklart bra att köra testerna i olika browsers för även om du följer ECMAScript standard så har olika browsers olika beteenden.

9. Johannes berättade att han också bara kör PhantomJS, för att han inte har upplevt att det funnits något behov av annat ännu. Jag förstår att om man har riktigt stora applikationer så behöver man göra det.

Har du stöd för uråldriga browsers, som IE8, som inte stödjer ECMAScript 5, så vill du ha in shim och liknande. Det fångar du inte om du enbart kör dina tester i PhantomJS, utan det är först när du deployar på en testserver som du märker att det inte fungerar i en gammal browser.

Även om allting verkar fungera så kan det uppstå problem med olika sorteringsordning i olika browsers. Om du gör en sort på en array eller localeCompare på strängar så kan du få olika resultat i olika browsers. Dina sorteringar blir inte stabila, resultatlistorna kan till exempel skilja sig mellan IE och Chrome. Om man kör sina tester i olika browsers så är chansen stor att man fångar en sådan sak.

10. När anser du att det är särskilt lämpligt med testning av JavaScript? I allmänhet, inte bara i Angular.

Alltid, eftersom JavaScript inte är starkt typat så är min åsikt att de enda gånger du kan skippa att testa din JavaScript är i egna hobbyprojekt. Gör du något seriöst så måste du ha tester, allting annat är dömt att misslyckas. Det tycker jag å andra sidan om allting, även om du frågar mig om .NET. Du kan inte ändra någonting om du inte har tester, det är enda sättet att garantera att du har ett stabilt beteende.

11. Har du sett något trendskifte?

Tidigare har JavaScript handlat mycket om att få till häftiga effekter i browsern, animationer och annat som har med utseendet att göra. I och med inträdet av Angular och andra MVC-ramverk så har man nu en annan möjlighet att skala och strukturera koden till applikationer som körs i browsern och i och med det så blir det ett större fokus på testning. Åtminstone bland de som ser poängen i att skriva kvalitativ applikationskod.

Jag har varit med om en hel del projekt där de inte har haft tester men där de har börjat smyga in det då och då. Det blir ganska uppenbart, jag var inne i ett JavaScript coverage-projekt där jag skapade ett tiotal issues på saker som inte fungerade och som slutade fungera från version till version och till sist förstod utvecklaren att han kanske skulle ta och börja med lite testning trots allt.

Det är en fråga dels om att applikationsutvecklingen kräver det och dels om att enskilda utvecklare inser poängen med det.

12. Vad ser du som de viktigaste förutsättningarna för att lyckas med JavaScript-testning?

Det är en generell fråga som gäller för testning i allmänhet. Bra testfall är en viktig punkt, vilket tenderar att komma naturligt om du använder TDD på rätt sätt. I och med att JavaScript inte har något naturligt sätt att organisera filer med namespaces eller assemblies så kan både applikationsfilerna och specarna bli svåra att hålla reda på. Det kräver att man har en struktur för hur man lägger upp sina testfiler.

13. När jag nämner dig i min rapport så behöver jag kunna tala om vem du är. Då tänkte jag nämna att du har doktorsexamen i mjukvaruutveckling, att du driver programatically speaking, att du kommer att prata på sthlm.js... Har du pratat på liknande grejer tidigare?

Inte JavaScript just, jag ska prata på JavaZone i Oslo i september, jag pratade på Microsoft TechEd i New Orleans och Madrid i juni. Sen har jag pratat på lite lokala konferenser här i Karlskrona där jag bor. Inte så mycket mer än så länge.

14. Handlar JavaZone om enbart Java?

Nej, när jag kollade i programmet så såg jag att det fanns lite om Angular, så de verkar ha lite om JavaScript också.

15. Hur länge har du hållit på med JavaScript?

Jag höll på med JavaScript för kanske 12 år sedan, i en browser som hette Netscape Communicator, väldigt basic grejer. Projektet jag håller på med nu startade för lite mer än ett år sedan och det är första gången som jag skrivit riktig applikationskod i JavaScript. Dessförinnan har det i första hand varit hobbyprojekt i Node.js och liknande. Så det är väl inte jättemånga år så, men det har varit rätt intensivt senaste året. Jag har inte någon erfarenhet av andra ramverk än Angular, som Ember och Knockout.

16. Är det på factor10 som du har det här projektet?

Ja.

17. Hur har ditt tidigare jobb på Ericsson påverkat din syn på testning?

När jag jobbade på Ericsson så var det inom telecom, jag jobbade med motorn som ser till att du debiteras för samtal på telefonnätet. Det är förstås viktigt att den beter sig på rätt sätt så att inte operatören råkar ut för extra kostnader genom att ta felaktigt betalt. Vikten av ordentliga tester blev därför väldigt stor.

Man kan kanske argumentera för att testning är bransch-specifikt för telekom, banker och liknande, samtidigt kan jag inte komma på någon bransch där det skulle vara oviktigt att ha koll på sin kod. Till och med när jag kört hobbyprojekt så har jag upplevt att så fort du fuskar med en annan utvecklingsapproach och inte kör tester, då måste du köra din applikation och klicka dig fram vilket ger extremt långa feedback-cykler. Med testning så kortar du ner de cyklerna väsentligt.

Transcript of second interview with Per Rovegård

18. Du nämnde att det har blivit mer testning i och med MVC-ramverken, men att det också finns de som inte bryr sig så mycket om den kvalitet som tester kan medföra. Jag

funderar kring vad dessa personer har för argument, ett av dem är att det är svårt och i vissa fall inte går att testa användargränssnitt och utseende. Hur ser du på det?

Det går att göra om man vill. Det är svårt, men det går att testa en del. Jag tror snarare att det är så att de som inte gör det helt enkelt inte kan. De har inte förstått vinsten med det, tycker att det är svårt och vet inte var de ska börja. De tror att det är snabbare att bara hacka.

19. Hur ser du själv på sådant som är nära användargränssnittet? Känner du att det är värt att testa sådant?

Ja, det kan finnas en poäng i det. Det finns ett par sätt att göra det på, Selenium WebDriver i webb-sammanhang, sen har Angular en e2e-stöd (end-to-end testing) som jag dock inte har använt själv. Det är tänkt att testa på strukturen, det är DOMen du testar på då.

Det är rätt bra att ha tester som kan köra viktiga flöden, så att de håller. Annars är enda sättet att fånga eventuella felaktigheter att sitta och klicka igenom GUIs efter deploy och problemet som programmerare är att man tenderar att klicka igenom sådant som man vet fungerar.

Så jag tycker att det finns ett värde, sen kan det naturligtvis vara svårt. Om vi tar Selenium WebDriver, eller FEST Swing på Javasidan, så finns det ett problem med att de kan vara instabila av anledningar som du inte har kontroll över. Vi har kört Selenium WebDriver en del och ibland så går tester rött och nästa gång så går de grönt därför att integrationen inte fungerar som den ska.

20. Ofta tar de ganska lång tid att köra också.

Ett fåtal sådana för att testa det viktiga tror jag är bra.

21. Jag kan tänka mig att vissa tycker att testning är tråkigt, att det finns en attityd.

Här kommer vi till det som jag skrev till dig, du har ju två typer av testning. Det ena är den manuella testningen, den är jättetråkig, men det andra att skriva tester och jobba testdrivet det är mer en del av programmeringen. Det är ett verktyg för att du får bättre kod.

22. Jag tänker bara på automatiserade tester.

Ja, precis, och det är ett verktyg för att få bättre kod, kvalitet. Jag tycker även att det är det är det överlägset effektivaste sättet att jobba på, det går mycket snabbare. Den enda gången jag inte har använt det är när man prototypar.

23. Jag hade nyligen en intervju med en gränssnittsutvecklare och han uppvisade en osäkerhet kring hur man faktiskt gör. Det vore kul att kunna råda bot på det på något plan genom min rapport, men det är svårt att hitta de där konkreta tipsen som går att tillämpa direkt utan att man behöver lägga ner två månader på det.

Det beror på vad man är ute efter, vad man vill åstadkomma med testerna. Vill du kontrollera att användare kan gå igenom ett visst flöde så är det väl ändå ok att ha

WebDriver-tester på det viktigaste. Och med Angular, e2e och andra tester, att kolla att DOM-elementen faktiskt kommer ut är ju rätt så vettigt.

Just nu sitter jag och skriver tester på en hjälp-popup som är baserad på jQuery och Angular. Man kan lockas att tänka att det inte går att testa en bootstrap popover som det heter, men det gör det för man kan kolla i DOM-strukturen att de appendas (läggs till) på rätt ställe, får rätt klasser på sig och liknande. Så det är inte särskilt komplicerat alls egentligen.

24. Och det blir stabila tester då?

Ja visst, det är phantomJS, den har ett DOM-träd den också, som bygger på WebKit.

25. Jag tänker att det är viktigt att undvika onödiga beroenden i testerna, så att de inte går sönder av fel anledningar. Jag är helt med på att de kommer att gå igenom varje gång du kör dem nu, men när det börjar ske ändringar i andra delar av applikationen, som ändrar andra delar av DOMen och kanske strukturerar om, att det kan bli ett problem då.

Fast i det här fallet så är det ju en isolerad företeelse. I Angular kan du komponentifiera med direktiv och har en avgränsad funktionalitet som du kan testa och använda dig av. Om du ser till att den inte beror på en massa annat och vet att den fungerar så undviker du sådana problem.

26. Vad menade du i vår förra intervju när du sa att det finns projekt som inte har tester men där de börjat smyga in det då och då?

Om du kollar på Open Source-communityn så vet utvecklarna att det borde finnas tester, men att det inte alltid finns ännu. De vill ha det men vet inte hur. Det har att göra med hur populärt projektet är och vilken skala det är på. Att nya versioner kan ha sönder gammalt tror jag är en sådan sak som får folk att förstå att tester är ett sätt att undvika återkommande buggar och att saker som har fungerat tidigare slutar funka.

27. Så ibland händer det att enskilda programmerare väljer att lägga in tester i projekt som inte har det sen tidigare?

Så kan det ju vara, att du kommer till ett Open Source-projekt och vill göra ett bidrag till det. Då kan ju du välja att lägga till tester, så kan maintainern ta det och kanske inse vitsen med det och själv börja använda det. Men det handlar återigen mycket om personliga attityder.

28. Du nämnde att man behöver ha ett tänk kring hur man lägger upp sina testfiler. Vad menade du med det?

Det var att JavaScript inte har något färdig stöd för namnrymder, paket och liknande som finns i Java och .NET till exempel. Du har inte samma fördefinierade strukturer. Där får du hitta på något eget eller använda dig av RequireJS eller liknande. Det är en förutsättning för att lyckas med testning, att man har tänkt igenom hur man vill lägga upp sina tester. Vilken nivå man vill lägga sig på, om man vill undvika att ha testfiler med flera hundra specar så behöver man fundera över hur man ska åstadkomma det.

29. Har du provat att testa asynkron kod någonting? Har du använt Jasmines runs och waitsFor?

Ja. Det är lite fult, men jag har använt det lite. En fin sak med Angular är att du kan överrida de servicarna som finns, till exempel timeout, som gör det möjligt att exekvera saker senare. I testerna kan du då använda en synkron version, så att testerna får ett synkront beteende.

30. Uppstår det inte ett problem då i och med att du inte testar koden på det sätt som den kommer att köras i produktion?

Det beror lite på vad du är ute efter att testa. Det kan vara ett problem, så man får göra det när man vet vad man håller på med, när man vet att det inte påverkar.

31. Ser du det som ett problem i allmänhet att testa asynkron kod, är det svårare än att testa synkron kod?

Inte om du kör testdrivet och tänker på det från början. Att i efterhand testa asynkron kod kan vara krångligt.

32. Är det något annat som du tycker är krångligt med testning? Jag vill lägga fokus på det som folk faktiskt tycker är svårt.

Något som alltid är svårt är att hantera den data som ingår i testerna. Särskilt om testerna är av integrationskaraktär, då vill du kanske ha riktig data som hämtas från en databas eller filer eller liknande, samtidigt så vill du ha möjlighet att se datan i testerna, så där uppstår direkt en konflikt. Tittar du på ett test så är det ofta lättare att förstå testet och veta om det är korrekt om du kan se vilken data det använder, och om anropen till andra metoder görs korrekt.

Det finns ett mönster som heter Object Mother, som går ut på att du har en speciell klass som skapar dina objekt. Det är smidigt för att det gör dina tester renare men du gör det svårare att se vilken data som faktiskt används, plus att du skapar beroenden mellan testerna genom den här objektmodern. Samma problem uppstår om du använder riktig data i dina tester, det är bra på så sätt att det är äkta data, inte konstruerad, samtidigt så uppstår ett beroende mot saker och ting som kan ändras. Du kan tänka dig om du har en databas med ett gäng personer i med adresser och liknande. Då kan det hända att du gör en schemaändring och att testerna börjar fallera efter det, inte för att det de är tänkta att testa har slutat att fungera utan för att datan de berodde på inte ser ut som innan. Så det finns helt klart en svårighet i att hantera testdata.

En annan problematik är när du har en klient-server arkitektur, där servern producerar JSON-data som klienten konsumerar. Det blir lätt så att du har tester på serversidan och på klientsidan, men inga tester som baseras på att både klienten och servern körs. Så hur vet du att interfacet mellan klienten och servern är korrekt? Man kan tänka sig ett JSON-schema men den tekniken är inte riktigt mogen, så där finns också en utmaning.

33. Har ni vidtagit några konkreta åtgärder för att öka testbarheten mer än TDD?

Behövs det någon annan menar du?

34. Jag tänker att man kommer till vissa insikter när man jobbar testdrivet, kan du komma på några saker som TDD har lett till?

TDD leder till att du får avgränsade enheter. Vi kan ta ett exempel. Nu i sommar har jag skrivit ett plugin till ps3 media server, som ju ursprungligen utvecklades för att serva content till playstation 3, men som nu klarar en massa andra enheter. Den är extremt dåligt testad, de har inga automatiska tester, och när man är inne och roddar så märker man att allt sitter ihop med allting annat, i en enda stor röra. Om du vill använda en liten del så går inte det, för den sitter samman med 10 andra delar som inte går att få bort. Gör man saker testdrivet så blir det istället av naturliga skäl mer löskopplat därför för att testa en liten sak så, testar du bara den då, och då kan du inte ha en massa jobbiga beroenden. Så TDD leder till att saker och ting blir löskopplade och fristående, så att du sen komponenar ihop det på ett annat sätt.

35. Har du någon erfarenhet av kodkatas? Om man vill omsätta TDD-principer i praktiken så kanske det är ett bra ställe att börja på, tänker jag.

Ja, jag har erfarenhet av kodkatas. Kanske kan det vara så. Jag har gjort några enstaka och även lett någon övning som man skulle kunna kalla för kodkata. Problemet som jag ser det är att man tenderar att välja saker och ting där det är väldigt rakt fram (straight forward) att använda TDD och så. När man sen kommer till sitt riktiga projekt så är det plötsligt inte lika svart eller vitt längre. Men visst, för att förstå principerna så är det givetvis bra. Absolut.

36. Vad har du för filosofi kring coverage? Du sa att ni hade flera tusen tester, då är jag nyfiken på vad det i första hand är ni testar.

Allt. Eller vad menar du? All funktionalitet. Arbetar du testdrivet så får du en väldigt god coverage av sig själv. Jag tycker inte att det finns något självändamål att sträva efter 100 % coverage, om inte annat för att det i vissa språk kan vara svårt att uppnå. Ta Java som exempel, där du måste fånga vissa Exceptions. Du måste fånga IOException när du läser en fil, men du vet att enda gången det kommer hända är om filen är korrupt på något sätt och det är väldigt svårt att återskapa i testerna, så därför kanske just den koden inte går att testa så väl. Då tar det kanske för mycket tid att försöka få testtäckning just där och då kan det vara bättre att acceptera att man inte har full testtäckning. Men i allmänhet så anser jag att allt normalbeteende ska vara täckt av tester.

Jag brukar ha som princip att om jag går in och gör en förändring, och jag ser till att testtäcka den, och alla befintliga tester går gröna, då är allting safe. Är det sen så att något beteende går sönder i produktion då är det den personen som skrev den koden som gick sönder från början som är skyldig till att inte ha skrivit tillräckligt bra tester. Man har ett ansvar att se till att den kod man skriver är så väl testad att någon annan ska kunna göra ändringar och märka om något slutar fungera genom att ett test går rött. Det är ett ansvarsskifte, då får inte skylla på den som checkade in koden som ledde till att någonting hände i produktion, utan det är den som slarvade med testerna som bär skulden.

37. Hur ser du på när tester går röda utan att man egentligen gjort något fel? Hur ofta händer det för dig?

I projektet jag är på nu så händer det uteslutande i WebDriver-testerna. När jag jobbade på Ericsson så jobbade vi väldigt mycket med ett ramverk som heter FEST Swing, som är baserat på Javas robot-interface. Det är också peka-klicka, fast i ett Swing-GUI. Det var samma sak där, ibland gick tester rött för att maskinen som körde testerna var hårt lastad, så att det blev längre delays vilket fick testerna att balla ur med timeouts. Vi hade stora problem med att knapparna inte fick fokus ordentligt. Jag vet att även systerprodukter ofta hade problem med tester som gick rött. Problemet i det läget är att man slutar lita på testerna.

Om du vet att det alltid är några tester som går rött, så slutar du bry dig, och när ett test plötsligt går rött för att något är fel på riktigt så ser du inte det. Så att ha röda tester och acceptera det, det är inte bra. Inte alls bra. Då får du ha en separat testsvit för dem, som heter instabila tester. Där du räknar med att det ska vara stabilt måste det alltid vara grönt. Man kan då diskutera, om man har tester som ibland går rött, då kanske man inte ska ha kvar de testerna, för de är inte tillförlitliga ändå. De har inget värde.

38. Det jag tänkte på är om man råkar skriva testerna så att de blir implementationsberoende och någon sedan gör en ändring som borde vara ok.

Ah, du menar att man lägger det på fel nivå? Sådant är såklart jobbigt. Om du skriver enhetstester så tenderar de att bli ganska implementationsberoende, medan integrationstester brukar gå att göra ganska oberoende av implementationen. Samtidigt så tenderar integrationstester att vara långsamma och inte lika precisa i sin formulering. Så det är en avvägning där helt klart.

Det är ju jobbigt när du vill ändra ett beteende och har en massa tester som... Eller, du vill bibehålla ett beteende men implementera det på ett annat sätt, och det finns en massa tester som är beroende på exakt hur det är implementerat. Då får man bita i det sura äpplet.

39. Och skriva om dem.

Ja.

40. Ser du något mönster i de tester ni har, alltså att det förekommer tester som är skrivna på ungefär samma sätt? Finns det olika grupperingar av tester, olika typer av tester? Då tänker jag i första hand på enhetstesterna, om de går att dela upp sinsemellan.

I vår svit så har vi blandat integrationstester och enhetstester. Testar du Angular controllers till exempel så ser jag det som en form av integrationstest. Medan enstaka domänklasser blir det enhetstester på. Vi har vissa tester som testar DOM-struktur. Vissa som testar just Angular controllers, interaction med scope och sådana saker. Sedan andra tester som testar ren vanilj-kod, som inte har med Angular att göra alls. Man skulle kunna dela upp det bättre, att tänka mer på hur man strukturerar testerna är en lärdom, det har inte vi riktigt gjort. Hade vi gjort det från början så hade jag kanske delat upp det lite grann och sagt att här är tester som är beroende på Angular och här är tester som kör ren vanilj-JavaScript.

41. Är det något annat som du skulle ha gjort annorlunda om du skulle börja om

nu?

Ja, jag skulle nog ha valt något annat än Jasmine. Samtidigt så är Angular så knutet till Jasmine. Jag tänker att någon borde ha skrivit någon liten kodsnutt som gör att du kan använda Mocha eller qUnit till Angular också. Det finns säkert någon som har gjort det.

42. Vad är du riktigt nöjd med?

Jag tror att vi har totalt 5000 tester, alltså 4000 tester i JavaScript och cirka 1000 på serversidan. Den är ganska tunn, serversidan. Det är en trygghet. Gör jag en ändring, en refaktorering eller ett tillägg av någonting, och alla 5000 tester går igenom, då är allting frid och fröjd. Då behöver jag inte sitta och klicka runt i GUI:t, för jag vet att allting fungerar som det ska. Det blir en inre frid *skratt*, är det grönt så är det lugnt.

Mail conversation with Per Rovegård after the interview

43. Hur tror du att kundens förväntningar tenderar att påverka hur mycket man testar? Om man får betalt per timme eller har en budget som man känner sig stressad av så kanske man låter bli att testa av den anledningen. Särskilt om man inte är van vid att testa. Hur tycker du att man bör gå tillväga för att ta sig ur en sådan situation?

Jag är van vid att göra TDD, och då är TDD ett verktyg för att leverera kod med hög kvalitet. För mig är TDD ett nödvändigt verktyg, likt ett tangentbord. Kunden eller chefen eller någon annan har inte att göra med vilka verktyg jag använder för att producera kod. Med andra ord ingår tiden att skriva tester i estimatet för kodningen. Det finns inget separat estimat för att skriva tester.

Sen är det så att när jag har mycket att göra inför en leverans och stressnivån är hög, då är TDD/testning det överlägset snabbaste sättet för mig att leverera något jag vet fungerar. All form av koda-och-kör-och-hoppas-det-funkar går bort.

44. Vad har du för tankar kring parprogrammering? Är det något du har ägnat dig åt mycket i dina projekt? Under vilka former i så fall? Vilken roll har testning spelat då?

Har inte så stor erfarenhet. Teorin är bra. :-)

45. Gränssnittsutvecklaren som jag nämnde ansåg att det finns en stor skillnad på webbapplikationer och mer enkla infositer, som är det som han har ägnat sig mest åt. Han uttryckte det så här: "De flesta info-siter är mest jQuery, DOM-manipulation så jag är inte säker på om man har någon särskild nytta av testramverk för JavaScript på den typen av grejer som man oftast gör. Det är nog snarare Selenium-tester som man kan använda för att täcka upp det mesta med. Min känsla är att JavaScript-testning är mer för skript-bibliotek eller om man skriver mer applikationskod." I vår första intervju (fråga 10) så uttryckte du att det alltid är lämpligt med testning av JavaScript. Står du fast vid denna synpunkt?

Ja, om man bryr sig om kvaliteten på det man gör.

46. Vad var det du menade med precision när vi pratade om integrationstester och närhet till implementation (fråga 38)?

Enhetstester kan testa många detaljer därför att de har stor frihet i att variera indatan. De kan vara väldigt precisa.

Integrationstester har typiskt högre uppsättnings- och körtid och är begränsade till faktorer såsom i nuläget möjlig användarinteraktion och kanske äkta data. Därför lämpar sig de mer för testing på högre nivå. Ska man testa precist med integrationstester får man ofta ganska slöa testsviter. Det är min erferenhet åtminstone. YMMV (Your Mileage May Wary)!

47. Vad ser du för vinster med BDD? Finns det någon del av BDD som du tycker mindre om?

Det beror på vad du menar med BDD. Jasmine utger sig för att vara ett BDD-ramverk, men för mig är det ett TDD-ramverk. BDD får full kraft om du jobbar med ett verktyg som RBehave, NBehave, SpecFlow, Cucumber, Lettuce etc. där du kan omsätta acceptanskriterier i scenarier uttryckta i naturligt språk som sedan mappas mot kod. BDD för mig är ett verktyg som överbryggar gapet mellan utvecklare och kravställare/-domänexperter, och är således ett utmärkt verktyg om du jobbar med DDD (Domain Driven Development).

Transcript of interview with Henrik Ekelöf

1. Vad har du för erfarenhet av testning av JavaScript?

Väldigt lite, jag har labbat med det privat, aldrig använt det i ett kundprojekt.

2. Hur kommer det sig?

Jag vet inte, de flesta info-siter är mest jQuery, DOM-manipulation så jag är inte säker på om man har någon särskild nytta av testramverk för JavaScript på den typen av grejer som man oftast gör. Det är nog snarare Selenium-tester som man kan använda för att täcka upp det mesta med.

Min känsla är att JavaScript-testning är mer för skript-bibliotek eller om man skriver mer applikationskod.

3. Så du har inte använt något MVC-ramverk?

Nej, aldrig. Inte i kundprojekt. Vi har diskuterat testning i olika projekt men jag har inte upplevt att det varit någon som tyckt att det varit värt att göra.

4. Har du varit med om att det funnits någon som verkligen velat testa och haft bra argument för det?

Nej, jag har aldrig jobbat med någon sådan. Jag har hört folk som haft sådana synpunkter men aldrig jobbat i något projekt tillsammans med någon som har haft det.

5. Hur mycket tror du att det har att göra med just kulturen, man kan ju tänka att det har att göra med att man gör just info-siter och inte så mycket applikationssiter, men

också att det kan ha att göra med just vilka man jobbar med. Hur viktigt tror du att det är?

Det är klart att vad utvecklarna vill göra styr mycket. Det inverkar definitivt, men jag är fortfarande inte säker på att det ger så mycket. De vanligaste fallen man skriptar i den här typen av siter är att det ska ploppa upp en ruta, man ska kunna göra några saker i rutan, kanske posta något med AJAX och få den att stänga sig. De funktioner man skriver behöver inte testas så mycket, man använder ju bibliotek som är testade i sig och man gör mest anrop till dem, så jag är skeptisk till att lägga tid på att skriva tester till den typen av skript.

6. Hur testar ni att det fungerar då, med sådana siter? Klickar ni runt, är det ofta som ni öppnar upp browsern för att kolla hur det ser ut?

Ja, absolut, det gör man. Annars är det ju Selenium som kan plocka om det blir fel på något JavaScript, så märker man det många gånger i de testerna. Sen är det den manuella testningen, JSHint kör vi, det är ingen testning men det hjälper en att kolla att man inte skrivit trasig kod. Det är ett slags kvalitetsverktyg som vi har kört i alla projekt som jag har varit med i de senaste åren.

7. Hur tror du att strukturen av koden påverkar ens benägenhet att testa?

Det är självklart, om man kan ha en väldigt bra struktur, kanske objektorienterad, att det blir smidigare att testa. Sen är det kanske inte alltid som man vill skriva det så, det känns som att man krånglar till koden för att göra den testbar.

8. Det kanske blir en effekt av testningen snarare än att det leder till testning. Jag tror att det har mycket att göra med det du säger om olika typer av siter, att de jag intervjuat tidigare har pratat mycket om att arbete med TDD höjer kvaliteten på koden, men samtidigt så om man vet att det inte kommer att vara ett problem... Har du underhållit någon sida under lång tid, behövt lägga till ny funktionalitet i efterhand och så?

Ja absolut, vårdguiden var ett sådant projekt. Jag var delaktig i det under lång tid och vi hade löpande releaser och även underhåll på befintlig kod. Där körde vi enbart Selenium-tester för JavaScript-koden, och givetvis enhetstester på backend-koden. Selenium-testerna plockade väl en del grejer men det var mest stök med dem tycker jag för att de kanske var lite dåligt skrivna.

Det är nog en konst att kunna det där, både att skriva koden på rätt sätt och att skriva testerna på rätt sätt. Jag behärskar det inte, så det är nog en del av det. Det är egentligen ingen annan som jag jobbat med som har gjort det heller.

9. Det har väl lite med tekniken att göra, det var några år sen som det projektet startade? Jag tror att Selenium har blivit lite bättre sen dess. Jag minns också i vårt projekt att vi hade problem med att testerna failade av konstiga anledningar.

Vi fick mycket fel för att det var tester på innehåll som ändrades, då har man gjort fel när man skrivit testerna.

10. Hur tror du man skulle göra ett bra Selenium-test?

Försöka efterapa det som folk gör på siten. Se till att skriva tester så att man inte är beroende av exakta formuleringar i texter och rubriker.

11. Det kanske beror lite på vad man har för stories och så också tänker jag. Man kan försöka knyta testerna till det?

Oh, ja. Men om man bygger ett flöde för till exempel att jämföra olika vårdcentraler så får man försöka klicka så som man tror att besökarna kommer att klicka sig genom det. Lista ut var man kan tänkas klicka fel eller var det kan gå fel av andra anledningar.

12. Vad har du för tankar kring tiden det tar att köra testerna?

Det är dåligt när det tar en kvart innan man vet om bygget gick igenom. Det är klart att ju snabbare man kan få det att gå desto bättre. Samtidigt så är det många saker som tar tid. Nu när man börjar med Sass, Compass och liknande tekniker så tar det plötsligt tid innan man kan ladda om sidan när man jobbar för att den ska bygga klart CSSerna först. Så det är svårt när projekten växer och man får mer och mer kod, att allt tar längre tid. Så det är en viktig grej att optimera.

13. Det är en vits med enhetstester, att de går snabbare att köra. Så att man istället för att vänta i en kvart på Selenium-testerna har några få Selenium-tester och sedan kompletterar med enhetstester. Men jag är med på att det inte är helt trivialt att veta när det är värt det.

Det jag kanske saknar är väl att jag inte vet hur man skulle göra i ett typiskt projekt som vi har och även om man kan leda reda på saker på nätet och lista ut själv hur man borde göra så vore det bra med enkla och bra exempel på hur man kan strukturera koden och då inte för applikationssiter utan mer exempel på hur man kan göra om man vill köra testdrivet på en vanlig infosite, där man använder jQuery och bara manipulerar DOM. I stil med ”Det kan vara värt att testa de här sakerna på det här sättet, och då behöver man skriva koden så här”, enkla praktiska exempel vore intressant att läsa för att kunna ta ställning till om jag tycker att det är bra.

14. Javisst, sen är det säkert värdefullt också att jobba med andra som är vana vid att testa. Det är väl som i alla sammanhang att man tjänar på att sätta någon som är nybörjare inom ett område med en erfaren, för att få ut så mycket som möjligt av det. Där har vi även en möjlighet med kodkatas, om man kör en ganska enkel övning. Sådana är å andra sidan ofta baserade på ganska låg nivå, strängmanipulation och liknande, rena logiksaker. Man kanske skulle kunna hitta en kodkata som är mer fokuserad på gränssnitt och öva sig på testning med en sådan, lösa samma problem flera gånger och upptäcka nya sätt att göra det på. Även där kanske göra det tillsammans med någon, så att man kan diskutera lösningen. Det är jätteintressant, om det är värt det, hur man ska göra. Det har varit lite mitt problem i det här exjobbet att jag har inte kunnat ge ett definitivt svar på om det är värt att testa eller inte för att det beror lite på vad det är för applikation.

Sen är det så att när man är några stycken i ett projekt och många är inne i koden, så är det svårt som det är bara att få folk att skriva skript som går igenom med JSHint-valideringen. Det är nog en ganska lång startsträcka på att få folk att skriva kod enligt något visst mönster för att det ska bli testbart, för de flesta är inte tillräckligt bra,

eller intresserade av att skriva JavaScript bra, man vill bara göra någonting snabbt oftast.

15. Hur tror du att det är kopplat till vilken bakgrund man har?

Eller intresse för JavaScript, gränssnitt...

16. Jag tänker att om man kommer från en design-sida eller från att koda assembler och C, hur sådant påverkar. Jag har fått intrycket av att folk som har kört mycket Ruby (on Rails) har en tendens att testa mer. För att folk gör det i den communityn helt enkelt. Att det på sätt och viss är en community-fråga.

Det kan säkert vara en del av det. Min erfarenhet är att de som inte i första hand är webbmänniskor utan med systemutvecklarbakgrund som skriver C# och använder EPiServer har synen att HTML, CSS och JavaScript är något som finns vid sidan av deras egentliga intresse, programmeringen. Då har man kanske inte rätt förkunskaper för att förstå att det påverkar mycket om man optimerar skript-koden. Känslan för hur man kan skriva en selektor och undvika att göra det 20 gånger i rad på samma objekt i jQuery. Man tänker inte på vissa grundläggande saker. Om man inte ens kan de grejerna i JavaScript så känns det långt bort att man ska slänga på ett speciellt sätt att skriva kod på för att få det testbart.

17. Tror du att testning kan vara i vägen för att skriva effektiv kod?

Nej, effektiv kod, det vet jag inte. Det kan säkert på många sätt bli bättre men jag tror också att det i vissa fall kan bli onödigt krångligt. Jag tror att det blir i vägen för en del för att skriva JavaScript överhuvudtaget för man kan det inte tillräckligt bra.

18. Samtidigt tänker jag att testning kan vara ett stöd för den som inte är så bra. Då kan man titta på hur andra gör och hela tiden få bekräftelse på om man har gjort rätt om testerna går igenom.

Ja, kanske. Absolut.

19. Har du sett något trendskifte, någon förändring inom det här? Hos folk som håller på med JavaScript, känns det som att fler pratar om att testa sin kod nu än innan? Kommer det upp ofta?

Ja lite, men det känns fortfarande som att det är långt borta innan vi kommer att göra det i alla projekt tror jag.

20. Mm, är det så att man inte vet var man ska börja riktigt, eller helt enkelt vill bli klar så snabbt som möjligt, eller vad ser du som största anledningen till det?

Att man inte vet hur det ska gå till, att man inte vet hur man behöver skriva koden i så fall, att man inte vet att man får någon vinst på att göra det. Mest att man inte har kunskapen och inte vet om man vinner något på att försöka göra det.

21. Saken är ju den att om man tittar på en typisk infosite med jQuery och mycket callbacks och liknande, då är det svårt att skriva tester i efterhand. Det vet jag från den koden jag kollat på under mitt exjobb, att det kan vara väldigt svårt att titta på existerande kod och inse hur man ska testa den. Det kräver nästan att man tänker på det före.

Ja, det tror jag också.

22. För att kunna presentera dig på ett bra sätt i min rapport så behöver jag veta hur du titulerar dig och hur länge du har hållit på med JavaScript.

Jag är gränssnittsutvecklare på valtech. Jag har ägnat mig åt JavaScript professionellt i cirka 7 år, lite längre om man räknar med hobby-hack men säg 7 år.

23. Och ganska intensivt?

Ja, det får man väl säga. De senaste åren i varje fall. Det har ju blivit mer och mer JavaScript hela tiden. De siter som jag byggde 2005-2006 hade relativt lite JavaScript. Idag använder man det hej vilt.

24. Har du varit engagerad i någon blogg eller hållit föredrag eller så?

Lite grann på valtech, men inte om JavaScript i första hand utan andra webbrelaterade saker.

25. Du höll ju i en grej för oss i talangprogrammet.

Ja, det var JavaScript som jag pratade om. Ja, det har jag gjort. Just det, bra. *skratt* Men jag är inte superaktiv vare sig bloggare eller liknande. Jag har konto på twitter men skriver inte så mycket numera. Jag läser rätt mycket däremot.

26. Du är inte engagerad i något open source-projekt eller så?

Nej, det är jag inte.

27. Tack ska du ha, det är värdefullt att få ett nytt perspektiv för de jag pratat med innan har varit personer med bakgrund inom Ruby och annan backend som gått in med inställningen att det är självklart att man ska testa.

Det finns en skillnad där jämfört med Microsoft-gänget, och vi som jobbar som gränssnittare är oftast med Microsoft-gänget för att Java Open Systems-gänget brukar göra allting själva, så vi hamnar aldrig i de projekten. Tyvärr, annars kanske vi skulle ha lärt oss mer om testning. Så jag tror att om du skulle prata med flera gränssnittare så kommer de antagligen inte att ha testat någon kod heller. Jag känner inte till att det görs i något Microsoft-projekt alls.

Mail conversation with Fredrik Wendt

Previous to this mail, I had presented myself and my work and given Fredrik questions. His answers are given in Swedish below.

Hej.

Jag svarar lite huller om buller här - känns som att det hade varit lättare att ta i en intervju/över telefon. :-)

Till att börja vill jag säga att jag ser väldigt få som funderar över VAD/VARFÖR de testar sin JavaScript-kod (utgår från webapp-världen, inte server side JS). Bara för att det går att få ett mätetal på kodtäckning så är det alltför vanligt att folk skall dra upp

feta maskiner där hela systemet kan köra, Selenium i någon form (webdriver etc) och långa körningar med många browsers. GUI-test? Nja.

GUI-kod (JavaScript) är ett lager precis som t ex JDBC/Linq och de flesta utvecklare tycker att det kan vara bra att testa denna, med hjälp av enhetstest. Därav följer att enheterna skall isoleras, ofta med hjälp av mockning eller stubbning.

Att testa sitt GUI-lager (JavaScript) kan man mena då följer samma princip, dvs vi skall ha hög kodtäckningsgrad på vårt GUI-lager och vi skall kunna köra enhetstesten i isolation, dvs vi mockar/stubbar bort server side. Om man börjar "mocka bort" hela browsern också, då drar de flesta lite på munnen och tycker att det blir lite för mycket isolation - de litar fortfarande inte på att browsers är så pass enhetliga, eller iaf enhetliga under ett ramverk (jQuery?), att de vågar lita på att någon lyckas göra en abstraktion av "browsern" (ofta DOM:en + events) att det känns tillförlitligt, att man litar på testen. (Och då är det ju mycket tveksamt med test, om man inte tror att de gör något bra eller fyller någon värdefull funktion.)

Så, men den bakgrunden biter jag tag i dina frågor en och en. :-)

Brasklapp: Jag utgår hela tiden nedan från ett scenario där "JavaScript-kod" avser kod i en webapp, där JS-koden körs i användarens browser. För annan form av kod så blir vissa frågor konstiga, eller så har jag missförstått vad du vill uppnå med din uppsats. :-)

1. På din blogg framgår det att du i slutet av oktober 2011, alltså för snart 2 år sedan, höll en dragning om JsTestDriver (JSTD), JsHamcrest och JsMockito. Jag har själv använt mig av JSTD en del och känner igen mig i problemen med att det ibland inte framfår tillräckligt tydligt när något test inte körs och att man kan behöva rensa webbläsar-cachen för att få ett korrekt beteende. Vad är din syn på JSTD såhär i efterhand, är det något du fortfarande använder? Varför/varför inte?

Jag sitter inte längre i projekt med samma webbtyngd. Jag använde JSTD flera gånger när jag testdrev fram "logik". Jag använde inte JSTD för att testa GUIt, utan snarare kollade att Observable-pattern fungerade i olika sammanhang. Applikationen hade en meddelandebus och på den skulle olika saker hända beroende på vilka meddelanden som kom och hur applikationen var konfigurerad. Ofta slutade det med ett par ajax-anrop som skulle skickas (vilka fångades upp och verifierades att de kom), men när svaret kom och datan fylldes på någonstans i någon modell så använde jag inte JSTD för att kolla att View-koden ritade upp saker rätt utifrån Modellen (och de event den skickade).

För att kolla View-koden användes tyvärr just Selenium Core(!) och ett hemmaknackat ramverk för att köra tester på många browsers parallellt. Det fungerade fint, men det fanns inget mockat back-end så det var en hel del uppsättning av databaser, tabeller osv för att göra körningar.

2. Vad gäller JsHamcrest så har jag inte någon personlig erfarenhet av att använda det. Jag får intrycket av att det kan användas till bland annat filtrering av datamängder och som grund till JsMockito. Har du något att tillägga om JsHamcrest och vad du har använt det till? När har man som mest nytta av det, i testningssammanhang? Känner du till alternativ som kan vara värda att överväga?

Känner inte till alternativ. Hamcrest är ordlek med bokstäverna i Matchers - det JsHamcrest och andra Hamcrest portningar (Java, Python, ...) gör är - i testsammahang - att tillföra matcher-funktioner som gör testkoden mer läsbar.

PSEUDOKOD:

```
testThat(anObject).hasProperty('color');
testThat(anObject).hasProperty('color').withValue('green');
```

Minns inte syntax för JsHamcrest och nätet är kass just nu på tåget (på väg mot Sthlm för att hålla kurs).

3. JsMockito har jag heller ingen erfarenhet av. Är det på grund av bakgrund inom Java eller något annat som gjorde att du bestämde dig för att använda dig av det? Har du någon uppfattning om hur det står sig jämfört med till exempel sinon.JS, de inbyggda stubbnings- funktionerna i Jasmine och "VanillaJS" (dvs. manuell stubbning helt utan ramverk)? Jag tänker särskilt på aspekter som hur enkelt det är att lära sig och använda, hur läsbar koden blir och hur setup oc teardown av fake-objekten fungerar.

För mig som java-människa var det lätt att lära sig och sättet att stubba- och mocka (dvs INTE tre-stegsmodellen: prepare, replay, verify) tilltalar mig då det tenderar att ge mycket mindre brus i testkoden.

Jag har använt Jasmine mycket lite, så pass att jag inte skall uttala mig om hur de mäter sig mot varandra.

Jag har använt mig av VanillaJS och överlagrat funktioner på objekt. Fördelen med ett bra mocknings/stubbningsramverk är att felmeddelandena kan vara mycket användbara. Istället för att testet bara blir rött och man behöver börja debugga, kan "expected call to method setColor with parameters 'blue', but got call to method setColor with parameters 'green'" vara mycket insiktsfullt och "snabbt" ge tillräckligt med information för att kunna bedöma vad som är fel.

4. Vilka verktyg använder du idag för testning av JavaScript och varför?

Inga. Föräldraledig. Använder mycket lite JavaScript för närvarande, men jag är intresserad av vad Angular har att erbjuda. (Samtidigt är jag besviken över hur begränsat Angular är, och hur omoget Ember är. När jag satte mig ner för att skriva om ett par applikationer i våras blev jag verkligen bedrövad över Embers überdåliga dokumentation (så fort man skrapar lite på ytan blir det stopp) och naïviteten i Angular (nested views?!).)

5. Har du någon tanke kring hur ramverk som Angular, CanJS och Ember kan inverka på testbarhet hos koden man skriver? Vilka andra åtgärder ser du som viktiga för att åstadkomma hög testbarhet?

Återigen - man skall inte skriva test "för att det går" och att "ramverket gör att vi kan testa ...". GUI-kod är inget magiskt och borde inte behandlas som det heller.

6. Vad jag förstår det som så har du engagerat dig i att anordna coding dojos och i att hjälpa andra med att göra det. Har du haft sådana med fokus på JavaScript och vilken roll har testning i så fall fyllt? Anser du att aktiviteter av den här typen är lämpliga för

att lära sig att programmera testdrivet i JavaScript? Vad behöver man komplettera med för att det ska generera värde inom en organisation?

Nej, inte för JavaScript. Bara Python och Java och C#. "Testning" har varit TDD. Acceptanstestning är ett område jag önskade jag hade mer praktisk erfarenhet av.

Dojos, om de utförs rätt, har ett enormt värde för organisationer. Detta är något jag talat om på många konferenser (hur man faktiskt levererar värde och inte bara pizza och något att dricka). Se http://www.slideshare.net/fwendt/coding-dojos-p-arbetstid för värde av dojos.

7. I vilka fall tror du att det lönar sig att skriva tester för sin JavaScript-kod?

I samma fall som för all annan kod - det är som sagt inget magiskt bara för att det är ett annat språk som sitter närmast användaren. Tänk en C#- eller VB.Net-applikation som pratar med en NodeJS-tjänst - hur/varför skall man testa GUI-delen?

(Påminner om att jag utgår från att du menar JavaScript-kod som körs i användarens browser.)

8. Upplever du att testningskulturen skiljer sig bland JavaScript-utvecklare jämfört med till exempel Java, Ruby och C#? Vad tror du att det beror på?

1. Det är VÄLDIGT stora generaliseringar det där.

1b. Jag kan tänka mig att "profiler" inom respektive communitys har olika inställning till testning, och att det finns subkulturer inom communitys som kan avgränsas till användare av olika språk, men det är VÄLDIGT stora generaliseringar.

1bb. Jag kan tänka mig att tillgången på "lätt att använda"-verktyg, utformning av ramverk och "reference projects" eller användandet av templates som skapar upp tomma testklasser från början påverkar utvecklare, mer än språk eller egentliga möjligheter att utföra bra testning.

2. Nej, det finns större variationer inom grupperna än det finns homogena trender och skillnader mellan dem.

3. Ja, jag tycker mig se fler som arbetar med Ruby som skriver olika former av pre-produktionstest för sin kod.

Det finns grupper som INTE skriver enhetstest, utan istället ser till att ha ordentligt med feedback i sin applikation som berättar hur systemet mår. Med bra monitorering och snabba sätt att rulla tillbaka utrullade system/kod/versioner, så kan pre-produktionstest ibland vara omotiverad kostnad. Detta diskuterades bland annat på CITCONf i Budapest okt 2012. (Bör finnas anteckningar att googla upp, Dave Squirrel (osäker på förnamnet) var det som anordnade sessionen.)

9. Har du uppfattat någon form av trendskifte när det kommer till testning av JavaScript? Hur skulle du spå att utvecklingen kommer att se ut framöver?

Trenden, generellt sett, är att det är fler som bryr sig om att skriva test. De som brydde sig tidigare är mer angelägna om att det skall vara "bra" test, med olika tolkning av "bra". Exempeltolkning är "kunna läsas av Business Analyst-folk", skall ge "värde"

(vad nu "värde" är), skall passa ihop med en övergripande teststrategi, skall vara enkla att kasta, skall vara tydliga att härföra från krav, skall vara skrivna så att de väl dokumenterar förväntat beteende av enheten/systemet under test ...

Två bra trender helt enkelt.

En liten ny trend är som sagt de som helt skippar pre-produktionstest. Deras arbetsuppgifter är små och de kan driftsätta dem och slå på/av dem i produktion = liten risk och "det är inte förrän i produktion som man verkligen kan veta om det fungerar eller inte, oavsett nivå av pre-produktionstestning".

En annan trend är att fler och fler tar sig högra i stegen mot att test även omfattar A/B-testning. Du känner säkert till Netflix t ex och hur de testar olika features mot olika delar av sin användarbas, och jämför resultaten mellan varandra för att avgöra vilka features som ger bolaget "värde" (likes, genomförda transaktioner, antal clicks to "action", ...)

10. Vad anser du att enhetstestning (med rigorös mockning/stubbning) tillför i jämförelse med att testa flöden och integration av APIer? I vilka sammanhang lämpar sig de olika sätten att testa på?

Nu börjar jag bli lite trött märker jag, så med risk för att missa hälften:

Välskrivna test dokumenterar hur enheten/funktionen/systemet under test förväntas bete sig (i programmerarnas huvud), är de välskrivna så verifierar testen att koden också faktiskt beter sig så som utvecklarna som skrev produktionskoden TROR att den beter sig.

Den tillåter att man, givet att API:er man ansluter sig mot är väldokumenterade och stabila, kan parallellisera utvecklingen. Om vi har 5 features som skall gå mot samma datalagringstjänst, men den tjänsten ännu inte finns, så kan man definiera ett API och därefter sätta igång att jobba på 5 fronter parallellt. (Finns givetvis många risker med att göra så också, som att API:et saknar funktioner som behövs om det designas up-front, men det finns risker med allt. :-)

Jag har erfarenhet från olika variationer av detta sätt att arbeta med integration också, och ibland är det så att konsumentdelen helt enkelt rent schemamässigt har möjlighet att göra "sitt" arbete nu och producentdelen (som tillhandahåller implementationen av tjänsten) pga "real life (ledigheter, sjukdom, dödsfall, dålig organisation/projekt, ...)" behöver skrivas senare. Att jobba med mockning då gör att en hela applikation kan utvecklas till den grad att den ser ut att fungera, medan den faktiska implementationen av bakänden skrivs senare. Detta har jag flera gånger sett fungera utomordentligt bra (gick raka spåret ut till betalande kunders produktion), och utomordentligt dåligt (bakänden klarade inte av lasten som genererades). Oavsett om det gick bra eller dåligt - att jobba med mockning och stubbning MÖJLIGGÖR att man kan fortsätta utveckla/utforska sin applikation.

(Det går givetvis tvärs emot "full slice"-tänket inom agil utveckling, men verkligheten får ibland ha företräde över optimala situationer.)

11. Har du stött på eller känner till några "olösta problem" inom testning av JavaScript,

områden där det är oklart hur man bör gå tillväga för att skriva bra tester?

Inget jag kommer på nu på tåget.

12. Vad skulle du ge för tips till någon som upplever att testning av JavaScript är problematiskt?

Det är en för bred fråga för att förstå problemet - i vilken kontext? Jag hade bett personen förklara vad han/hon menar, i vilket sammanhang är det svårt att testa? "Ge exempel på vad du tycker är svårt!"

13. Jag hoppas att det inte var för många frågor, haha... Om det är så, så får du be mig att avgränsa eller anpassa längden på svaren. Du får gärna fokusera på de frågor som du tycker känns mest intressanta. :)

Jag fokuserade på det jag tyckte jag har något att säga om. Hoppas det hjälper och du får gärna höra av dig igen, eller ringa och ställa följdfrågor.

DescribeSML source code

The DescribeSML testing framework that was developed as part of this thesis is designed to work with Standard ML of New Jersey (SML/NJ). It is divided into three structures that provide matchers (Expect), a running utility (Describe) and a reporter (SpecReporter). They can be used separately if desired. The running utility allows for nested describe blocks as in the JS frameworks Jasmine and Mocha (or RSpec in Ruby). They can be loaded using the CM package manager of SML/NJ.

expect.sml

```
structure Expect = struct
fun expect it f = f(it)

local
    fun checkWith operator value result =
        if operator(value, result) then "pass" else "FAIL"
in
    fun toEqual value = checkWith (op =) value

    fun toNotEqual value = checkWith (op <>) value

    val toBe = toEqual

    val toNotBe = toNotEqual
end

local
    fun checkStrWith operator relation result value =
        if operator(value, result) then
            "pass"
        else
            concat ["FAIL: expected \"", result, "\" to ", relation, " \"", value, "\""]
in
    fun toEqualStr result = checkStrWith (op =) "equal" result
```

```
25
26      fun toNotEqualStr result = checkStrWith (op <>) "NOT equal" result
27
28      val toBeStr = toEqualStr
29
30      val toNotBeStr = toNotEqualStr
31  end
32
33  local
34      fun checkIntWith operator relation result value =
35          if operator(value, result) then
36              "pass"
37          else
38              concat ["FAIL: expected \"", Int.toString(result), "\" to ",
    relation, " \"", Int.toString(value), "\""]
39  in
40      fun toEqualInt result = checkIntWith (op =) "equal" result
41
42      fun toNotEqualInt result = checkIntWith (op <>) "NOT equal" result
43
44      val toBeInt = toEqualInt
45
46      val toNotBeInt = toNotEqualInt
47  end
48
49  local
50      fun checkListWith relation result value =
51      let
52          fun checkNth n = n >= 0 andalso length result > n andalso List.nth(
    result, n) = value
53          val containsValue = List.exists (fn a => a = value) result
54          val success = case relation of
55              "contain" => containsValue
56            | "not contain" => not containsValue
57            | "begin with" => checkNth 0
58            | "not begin with" => not (checkNth 0)
59            | "end with" => checkNth (length result - 1)
60            | "not end with" => not (checkNth (length result - 1))
61      in
62          if success then
63              "pass"
64          else
65              concat ["FAIL: Expected List to ", relation, " value"]
66      end
67  in
68      fun toContain result = checkListWith "contain" result
69
70      fun toNotContain result = checkListWith "not contain" result
71
72      fun toBeginWith result = checkListWith "begin with" result
73
74      fun toNotBeginWith result = checkListWith "not begin with" result
75
76      fun toEndWith result = checkListWith "end with" result
77
78      fun toNotEndWith result = checkListWith "not end with" result
79  end
```

```
80
81 local
82     structure RE = RegExpFn (
83         structure P = AwkSyntax
84         structure E = BackTrackEngine)
85 in
86     fun toMatch result value =
87         case StringCvt.scanString (RE.find (RE.compileString value)) result
     of
88             NONE => concat ["FAIL: expected \"", result, "\" to match \"",
    value, "\""]
89           | SOME match => "pass"
90 end
91
92 fun toThrow callback exc1 =
93     (callback(); "FAIL: did not raise " ^ (exnName exc1))
94     handle exc2 => if exnName exc2 = exnName exc1 andalso exnMessage exc2 =
     exnMessage exc1 then
95         "pass"
96     else
97         "FAIL: raised " ^ (exnName exc2) ^ " with message \"" ^ (exnMessage
     exc2) ^ "\" instead of " ^ (exnName exc1) ^ " with message \"" ^ (
    exnMessage exc1) ^ "\""
98 end
```

describe.sml

```
1  structure Describe = struct
2  fun suite((description, status)) =
3      print (concat ["\n", description, status, "\n\n"])
4
5  fun describe sut specs =
6      let
7          val failures = List.filter (fn (_, result) => SpecReporter.
    isFailure result) specs
8          val resultReport = map (SpecReporter.report "concise") specs
9          val failureReport = map (SpecReporter.report "verbose") failures
10     in
11         (concat ["Ran ",
12             Int.toString (length specs),
13             " specs for ",
14             sut,
15             ":\n\n",
16             concat resultReport,
17             concat (if length failures > 0
18                 then "\n\n" :: failureReport
19                 else ["\n"]),
20             "\nFailures: ",
21             Int.toString(length failures),
22             "\n==========\n\n"],
23         (if length failures > 0
24             then "FAIL: a nested spec failed, see report above."
25             else "pass"))
26     end
27
28 fun should(description, spec) =
29     (description, spec())
```

```
    handle exc => (description, "FAIL: raised " ^ exnName exc)
end
```

spec–reporter.sml

```
structure SpecReporter = struct
fun isFailure result =
    String.substring(result, 0, 4) = "FAIL"

fun report "verbose" (description, result) =
    concat ["should ", description, ": ", result, "\n"]
  | report _ (_, result) =
    if isFailure result then "!" else "."
end
```