Certifierad testare

Kursplan för grundnivå
Agil testare

Version 2014

Swedish Software Testing Board
International Software Testing Qualifications Board

V1.03

*Senaste versionen av detta dokument kan hämtas från **http://www.sstb.se**. Versioner av detta dokument hämtade från andra webbsidor än **http://www.sstb.se** kan vara en äldre version.*

Detta dokument baseras på ISTQBs dokument “Foundation Level Extension Syllabus, Agile Tester” och har översatts till svenska av Swedish Software Testing Board.
Medverkande i översättning och granskning har varit Tobias Ahlgren, Beata Karpinska, Ninna Morin och Ingvar Nordström. Dokumentet motsvarar den internationella certifieringen “ISTQB® Cerified Tester Foundation Level Extension, Agile Tester”. Enligt ISTQB gäller följande villkor:

1) Individer och kursarrangörer får använda denna kursplan som bas för kurser om referenser ges till författarna, ISTQB och SSTB som copyright-ägare av dokumentet och att all annonsering av kurser först kan omnämna kursplanen efter det att en officiell ackreditering av kursmaterialet har gjorts av SSTB (ett av ISTQB erkänt nationellt organ).
2) Varje individ eller grupp av individer får använda denna kursplan som bas för artiklar, böcker eller andra skrifter om författarna, ISTQB och SSTB är nämnda som källan och copyright-ägare av kursplanen.
3) Varje av ISTQB erkänd nationell styrelse, har rättighet att översätta och licensiera kursplanen och dess översättning till andra kurshållare.

SSTB, Swedish Software Testing Board, är av ISTQB en erkänd nationell styrelse och har därmed övertagit rättigheterna för den svenska översättningen.

Harmonisering med ”ISTQB: Certified Tester Foundation Level Extension Syllabus, Agile Tester”.

Foundation Level Extension Syllabus, Agile Tester:

Foundation Level Extension Agile Tester Working Group: Rex Black (ordförande), Bertrand Cornanguer (vice ordförande), Gerry Coleman (ansvarig för inlärningsmål), Debra Friedenberg (ansvarig för examinering), Alon Linetzki (ansvarig för marknadsföring), Tauhida Parveen (redaktör) och Leo van der Aalst (utvecklingsansvarig).

Författare: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh och Stephan Weber.

Interna granskare: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael och Erik van Veenendaal; 2013-2014.

Versionshistorik

Version	Datum	Kommentarer
Kursplan 2014 1.0	2014-10-05	Godkänd version
Kursplan 2014 1.01	2015-02-26	Mindre korrigeringar
Kursplan 2014 1.02	2015-03-31	Formaterat till A4
Kursplan 2014 1.03	2015-08-26	Referenser till böcker tillagt

Innehållsförteckning

Versionshistorik 3
Innehållsförteckning 4
Introduktion till denna kursplan 6
Avsikten med detta dokument 6
Översikt 6
Inlärningsmål som kan examineras 6
1. Agil programvaruutveckling – 150 min. 7
1.1 Grunderna för agil programvaruutveckling 8
1.1.1 Agil programvaruutveckling och det agila manifestet 8
1.1.2 Hela teamets ansvar 9
1.1.3 Tidig och regelbunden återkoppling 9
1.2 Aspekter av agila angreppssätt 10
1.2.1 Angreppssätt för agil programvaruutveckling 10
1.2.2 Gemensamt framtagande av användarberättelser 12
1.2.3 Retrospektivmöten 13
1.2.4 Kontinuerlig integration 13
1.2.5 Leverans- och iterationsplanering 14
2. Grundläggande principer, arbetssätt och processer för agil testning – 105 min. 17
2.1 Skillnaderna mellan traditionell testning och agil testning 18
2.1.1 Test- och utvecklingsuppgifter 18
2.1.2 Projektarbetsprodukter 19
2.1.3 Testnivåer 20
2.1.4 Test- och konfigurationshantering 20
2.1.5 Organisationsalternativ för oberoende testning 21
2.2 Teststatus i agila projekt 21
2.2.1 Kommunicera teststatus, testprogress och produktkvalitet 21
2.2.2 Hantering av regressionsrisker genom utveckling av manuella och automatiserade testfall 22
2.3 Testarens roll och färdigheter i ett agilt team 24
2.3.1 Den agila testarens kompetens 24
2.3.2 Testarens roll i ett agilt team 24
3. Metoder, tekniker och verktyg för agil testning – 480 min. 26
3.1 Metoder för agil testning 27
3.1.1 Testdriven utveckling, acceptanstestdriven utveckling och beteendedriven utveckling 27
3.1.2 Testpyramiden 28
3.1.3 Testkvadranter, testnivåer och testtyper 28
3.1.4 Testarens roll 29
3.2 Utvärdering av kvalitetsrisker och testuppskattning 30
3.2.1 Utvärdering av kvalitetsrisker i agila projekt 30
3.2.2 Uppskatta mängden testarbete utifrån innehåll och risk 31
3.3 Tekniker i agila projekt 32
3.3.1 Acceptanskriterier, tillräcklig täckningsgrad och annan information för testning 32
3.3.2 Använda acceptanstestdriven utveckling 34
3.3.3 Black-box-testdesign för funktionella och icke-funktionella egenskaper 35
3.3.4 Utforskande testning och agil testning 35
3.4 Verktyg i agila projekt 36
3.4.1 Verktyg för ärendehantering och spårning 37
3.4.2 Verktyg för kommunikation och informationsdelning 37
3.4.3 Verktyg för programvarubyggen och distribution 38
3.4.4 Konfigurationshanteringsverktyg 38
3.4.5 Verktyg för design, implementering och exekvering av tester 38
3.4.6 Verktyg för molntjänster och virtualisering 39
Referenser 40
Standarder 40
Dokument från ISTQB 40
Böcker 40
Agila termer 40
Övriga referenser 41

Index 42

Introduktion till denna kursplan

Avsikten med detta dokument

Denna kursplan utgör grunden för certifiering av agila testare på grundnivå av International Software Testing Qualifications Board. ISTQB® tillhandahåller den här kursplanen i följande syften:

- Till nationella ISTQB-styrelser för översättning till lokala språk och för ackreditering av utbildningsleverantörer. De nationella styrelserna kan anpassa kursplanen efter specifika nationella behov och anpassa referenserna efter lokala förhållanden.
- Till examineringsorgan, som kan härleda examineringsfrågor på sina respektive språk med hänsyn tagen till inlärningsmålen för varje kursplan.
- Till utbildningsleverantörer, att användas som material för kurser och som vägledning för att hitta lämpliga undervisningsmetoder
- Till de som vill certifieras, så att de kan förbereda sig för examineringen (antingen som en del av en utbildning eller fristående).
- Till programvaru- och systemtekniker i hela världen som ett hjälpmedel för att öka den samlade kunskapen om programvaru- och systemtestning, och som grund för böcker och artiklar.

ISTQB® kan också tillåta andra organ och enskilda att använda kursplanen i andra syften förutsatt att de inhämtar skriftligt godkännande i förväg.

Översikt

Översiktsdokumentet för agil testning på grundnivå [ISTQB_FA_OVIEW] innehåller följande information:

- Affärsnyttan för kursplanen
- Sammanfattning av kursplanen
- Kopplingar mellan kursplanerna
- Beskrivning av kognitiva nivåer (K-nivåer)
- Bilagor

Inlärningsmål som kan examineras

Inlärningsmålen ligger till grund för affärsutfallen och används för att skapa det prov som används för att examinera den som vill certifieras som agil testare på grundnivå. Generellt kan alla delar av den här kursplanen examineras på nivå K1, det vill säga genom att kandidaten ska kunna känna igen och komma ihåg termer och koncept. De specifika inlärningsmålen på nivåerna K1, K2 och K3 finns i början av respektive kapitel.

1. Agil programvaruutveckling – 150 min.

Nyckelord

Agilt manifest, agil programvaruutveckling, modell för inkrementell utveckling, modell för iterativ utveckling, livscykel för programvara, automatiserade tester, testbas, testdriven utveckling, testorakel, användarberättelse

Inlärningsmål för agil programvaruutveckling

1.1 Grunderna för agil programvaruutveckling

FA-1.1.1 (K1) Komma ihåg grundkonceptet för agil programvaruutveckling baserat på det agila manifestet
FA-1.1.2 (K2) Förstå fördelarna med hela teamets ansvar (whole team approach)
FA-1.1.3 (K2) Förstå fördelarna med tidig och regelbunden återkoppling

1.2 Aspekter av agila angreppssätt

FA-1.2.1 (K1) Komma ihåg angreppssätt för agil programvaruutveckling
FA-1.2.2 (K3) Skriva testbara användarberättelser i samarbete med utvecklare och verksamhetsrepresentanter
FA-1.2.3 (K2) Förstå hur retrospektiv kan användas som ett verktyg för processförbättring i agila projekt
FA-1.2.4 (K2) Förstå nyttan och syftet med kontinuerlig integration
FA-1.2.5 (K1) Känna till skillnaderna mellan iterations- och leveransplanering samt hur en testare skapar mervärde under dessa aktiviteter

1.1 Grunderna för agil programvaruutveckling

En testare i ett agilt projekt arbetar på ett annat sätt än testare i traditionella projekt. Testaren måste vara införstådd med de värderingar och principer som ligger till grund för agila projekt, och inse att testare tar en viktig del av hela teamets ansvar tillsammans med utvecklare och representanter från verksamheten. Teammedlemmarna i ett agilt projekt kommunicerar ofta med varandra. Denna kommunikation sker så tidigt som möjligt, så att man kan eliminera defekter tidigt och förbättra produktkvaliteten.

1.1.1 Agil programvaruutveckling och det agila manifestet

2001 samlades en grupp representanter för några av de vanligaste så kallade agila metoderna (lättviktsmetoderna) för programvaruutveckling och sammanställde en gemensam uppsättning värderingar och principer som har blivit välkänt som Manifestet för agil programvaruutveckling, eller det agila manifestet. Det agila manifestet innehåller fyra grundvärderingar:

- Individer och interaktioner *framför* processer och verktyg
- Fungerande programvara *framför* omfattande dokumentation
- Kundsamarbete *framför* kontraktsförhandling
- Anpassning till förändring *framför* att följa en plan

Det agila manifestet utgår från att det visserligen finns värde i punkterna till höger, men punkterna till vänster värdesätts mer.

Individer och interaktioner

Agil utveckling är människocentrerad. Programvara skapas av team, som arbetar så effektivt som möjligt genom kontinuerlig kommunikation och interaktion i stället för att förlita sig på verktyg eller processer.

Fungerande programvara

Ur kundperspektiv är fungerande programvara betydligt mer användbart och värdefullt än överdrivet detaljerad dokumentation, och med fungerande programvara kan utvecklingsteamet också få snabb återkoppling. Fungerande programvara kan också göras tillgänglig på ett tidigt stadium under utvecklingscykeln (även om den har begränsad funktionalitet). Agil utveckling innebär alltså att ledtiderna kan kortas betydligt. Agil utveckling är därför särskilt användbart på snabbföränderliga marknader där problemen och/eller lösningarna inte är helt tydliga eller där ett företag vill arbeta innovativt inom nya problemområden.

Kundsamarbete

Många kunder har svårt att ge tydliga kravspecifikationer för det system de behöver. Ett direktsamarbete med kunden ökar sannolikheten för en bättre förståelse av kundens behov. Det kan förstås vara viktigt att ha kontrakt med kunden, men ett regelbundet och välfungerande samarbete är viktigare för att åstadkomma ett lyckat projekt.

Anpassning till förändring

Förändring är en ständigt närvarande faktor i programvaruprojekt. Projektet och dess målsättningar kan påverkas av en mängd faktorer, till exempel marknadsförutsättningarna, lagar och regler, vad konkurrenterna gör och teknikutvecklingen. Alla dessa faktorer måste beaktas under utvecklingsprocessen. Därför är det viktigare att bygga in flexibilitet i alla arbetssätt än att ha en fastställd plan som man följer strikt.

Principer

Kärnan i det agila manifestet uttrycks i tolv principer:

- Vår högsta prioritet är att tillfredsställa kunden genom tidig och kontinuerlig leverans av värdefull programvara.
- Välkomna förändrade krav, även sent under utvecklingen. Agila metoder utnyttjar förändring för att åstadkomma konkurrensfördelar för kunden.

- Leverera fungerande programvara ofta, med ett par veckors till ett par månaders mellanrum, ju oftare desto bättre.
- Verksamhetskunniga och utvecklare måste arbeta tillsammans dagligen under hela projektet.
- Bygg projekt kring motiverade individer. Ge dem de förutsättningar och det stöd de behöver, och lita på att de får jobbet gjort.
- Kommunikation ansikte mot ansikte är det bästa och effektivaste sättet att förmedla information, både till och inom utvecklingsteamet.
- Fungerande programvara är främsta måttet på framsteg.
- Agila metoder främjar uthållighet. Sponsorer, utvecklare och användare ska kunna hålla jämn utvecklingstakt under obegränsad tid.
- Kontinuerligt fokus på förstklassig teknik och bra design stärker anpassningsförmågan.
- Enkelhet – konsten att maximera mängden arbete som inte görs – är grundläggande.
- Bäst arkitektur, krav och design växer fram med självorganiserande team.
- Med jämna mellanrum reflekterar teamet över hur det kan bli mer effektivt och justerar sitt beteende därefter.

De olika agila metoderna beskriver olika arbetssätt där dessa värderingar och principer omsätts i handling.

1.1.2 Hela teamets ansvar

Hela teamets ansvar (whole team approach) innebär att alla som har kunskaper och kompetens som behövs för att projektet ska lyckas ska ingå i teamet. Teamet ska innefatta representanter från kunden och andra intressenter som avgör vilka funktioner som ska ingå i produkten. Teamet bör vara relativt litet: framgångsrika team har vanligtvis mellan tre och nio medlemmar. Hela teamet bör ha samma arbetsplats eftersom det underlättar kommunikation och samarbete. Hela teamets ansvar betonas genom dagliga ståmöten (se avsnitt 2.2.1) med samtliga teammedlemmar där man pratar om de framsteg som har gjorts i arbetet och flaggar för eventuella hinder. Hela teamets ansvar ger ett mer dynamiskt och effektivt arbetssätt inom teamet.

Hela teamets ansvar (whole team approach) för produktutveckling är en av de främsta beståndsdelarna i agil utveckling. Här är några av fördelarna:

- Bättre kommunikation och samarbete inom teamet
- Gör det lättare att dra nytta av all kunskap och kompetens inom teamet så att projektet fungerar bättre
- Gör att alla delar på ansvaret för att skapa kvalitet

I agila projekt ansvarar hela teamet tillsammans för kvaliteten. Hela teamets ansvar (whole team approach) innebär att testare, utvecklare och verksamhetsrepresentanter samarbetar under varje steg av utvecklingsprocessen. Testarna har ett nära, kontinuerligt samarbete med såväl utvecklare som verksamhetsrepresentanter för att se till att önskad kvalitet uppnås. Detta innefattar att ge stöd till och samarbeta med verksamhets-/användarrepresentanterna för att hjälpa dem att skapa lämpliga acceptanstester, att samarbeta med utvecklare för att fastställa en teststrategi och att fatta beslut om hur automatiserade tester ska användas. På så sätt kan testarna sprida sina testkunskaper till övriga teammedlemmar och påverka produktutvecklingen.

Hela teamet deltar i alla rådslag eller möten där produktfunktioner presenteras, analyseras eller uppskattas/beräknas. Att engagera testare, utvecklare och verksamhetsrepresentanter i alla funktionsdiskussioner på det här sättet kallas för ”Power of Three” [Crispin08].

1.1.3 Tidig och regelbunden återkoppling

Agila projekt delas in i korta iterationer så att projektteamet kan få tidig, kontinuerlig återkoppling om produktkvaliteten under hela utvecklingscykeln. Ett sätt att få snabb återkoppling är att arbeta med kontinuerlig integration (se avsnitt 1.2.4).

Om man använder en sekventiell utvecklingsmetod ser kunden ofta inte produkten förrän i slutet av projektet. I det skedet kan utvecklingsteamet oftast inte göra något åt eventuella problem som kunden upptäcker. Genom att inhämta kontinuerlig återkoppling från kunden under hela projektförloppet kan agila team ta till sig informationen från kunden för att göra förändringar under hela produktutvecklingsprocessen. Tidig och regelbunden återkoppling hjälper teamet att fokusera på de funktioner som har störst värde för verksamheten (eller som medför störst risker) och leverera dessa först till kunden. Dessutom fungerar teamet bättre eftersom alla får en tydlig bild av vad teamet kan – och inte kan – göra. Till exempel: hur mycket jobb hinner vi med under en sprint eller iteration? Vad kan hjälpa oss att jobba snabbare? Finns det något som drar ned tempot?

Här är några av fördelarna med tidig och regelbunden återkoppling:

- Lättare att undvika missförstånd om behov och krav, som annars kanske inte hade upptäckts förrän senare i utvecklingscykeln när det blir dyrare att korrigera produkten.
- Tydligare kravspecifikationer av vilka funktioner eller features kunden behöver, så att de kan levereras tidigt. På så sätt matchas produkten bättre mot kundens faktiska behov.
- Bättre chans att upptäcka (via kontinuerlig integration), isolera och åtgärda kvalitetsproblem i ett tidigt skede.
- Det agila teamet får en bättre bild av sin kapacitet och vad de kan leverera.
- Gör det lättare att hålla en god arbetstakt i teamet.

1.2 Aspekter av agila angreppssätt

Det finns flera olika agila angreppssätt som kan användas inom företag. De flesta agila företag arbetar till exempel med gemensamt framtagande av användarberättelser, retrospektivmöten, kontinuerlig integration och planering av varje iteration samt av slutleveransen. I det här avsnittet beskrivs några agila angreppssätt.

1.2.1 Angreppssätt för agil programvaruutveckling

Det finns flera olika agila angreppssätt som kan användas för att förverkliga värderingarna och principerna i det agila manifestet på olika sätt. I den här kursplanen beskrivs tre av dessa agila angreppssätt: extrem programmering (XP), Scrum och Kanban.

Extrem programmering (XP)
Extrem programmering (XP) lanserades av Kent Beck [Beck04] och är ett angreppssätt för agil programvaruutveckling som utgår från ett antal värderingar, principer och arbetssätt.

XP bygger på fem värderingar: kommunikation, enkelhet, återkoppling, mod och respekt.

Dessutom används en uppsättning principer som vägledning för XP: mänsklighet, ekonomi, ömsesidiga fördelar, självliknande (self-similarity), förbättring, mångfald, reflektion, flöde, möjligheter, redundans, felsymptom, kvalitet, små steg (baby steps) och accepterat ansvar.

För XP används tretton huvudsakliga arbetssätt: sitt tillsammans, hela teamets ansvar, en informativ arbetsyta, energiskt arbete, parprogrammering, berättelser, veckocykel, kvartalscykel, slack, tiominutersbygge, kontinuerlig integration, testdriven utveckling och inkrementell design.

Många av de angreppssätt som används för agil programvaruutveckling är inspirerade av värderingarna och principerna bakom XP. De agila team som arbetar enligt Scrum använder till exempel ofta arbetssätten från XP.

Scrum
Scrum är ett ramverk för agilt arbete som innefattar följande verktyg och arbetssätt [Schwaber01]:

- Sprint: Ett Scrum-projekt delas in i iterationer (sprintar) med fast längd (vanligtvis två till fyra veckor).
- Produktinkrement: Resultatet av varje sprint är en potentiellt lanserbar/levererbar produkt (som kallas för ett inkrement).

- Produktbacklogg: Produktägaren ansvarar för en prioriterad lista med planerade uppgifter (som kallas för produktbacklogg). Produktbackloggen utvecklas för varje sprint (detta kallas för underhåll av backloggen).
- Sprintbacklogg: I början av varje sprint väljer Scrum-teamet ut ett antal högprioriterade objekt (denna uppsättning kallas för sprintbackloggen) från produktbackloggen. Eftersom det är Scrum-teamet och inte produktägaren som väljer ut de objekt som man ska arbeta med under sprinten drivs den här processen underifrån snarare än uppifrån.
- Definition of Done: För att se till att det finns en potentiellt levererbar produkt i slutet av varje sprint ska Scrum-teamet diskutera och definiera lämpliga villkor för att sprinten ska anses vara färdig: Definition of Done. Diskussionen fördjupar samtidigt teamets förståelse av objekten i backloggen och av produktkraven.
- Timeboxing: Endast de arbetsuppgifter, produktdelar eller funktioner som teamet förväntar sig att kunna slutföra inom sprinten ska ingå i sprintbackloggen. Om utvecklingsteamet inte hinner klart med en arbetsuppgift inom ramen för en sprint flyttas de associerade produktfunktionerna från sprinten och uppgiften flyttas tillbaka till produktbackloggen. Detta kallas för timeboxing och gäller inte bara arbetsuppgifter utan även andra aspekter (t.ex. start- och sluttider för möten).
- Transparens: Utvecklingsteamet rapporterar om sitt arbete och uppdaterar sprintstatus varje dag under det så kallade dagliga Scrum- eller ståmötet. På så sätt blir innehållet och förloppet inom sprinten, inklusive testresultat, synliga för såväl teamet som ledningen och andra intressenter. Utvecklingsteamet kan till exempel åskådliggöra status för sprinten på en whiteboard-tavla.

Inom Scrum finns tre roller:

- Scrum Master: ser till att alla följer arbetssätten och reglerna för Scrum och åtgärdar alla eventuella problem, resursbrister och andra hinder som kan göra det svårare för teamet att följa arbetssätten och reglerna. Den här personen fungerar som en coach snarare än som teamledare.
- Produktägare: representerar kunden och skapar, underhåller och prioriterar produktbackloggen. Den här personen är inte teamledare.
- Utvecklingsteam: utvecklar och testar produkten. Det här teamet är självorganiserat: det finns ingen teamledare, utan teamet fattar alla beslut gemensamt. Teamet är också tvärfunktionellt (se avsnitt 2.3.2 och avsnitt 3.1.4).

Till skillnad från XP föreskriver Scrum inte några specifika metoder för programvaruutveckling (t.ex. testdriven utveckling). Dessutom innefattar Scrum inga riktlinjer för hur testning ska genomföras i Scrum-projekt.

Kanban

Kanban [Anderson13] är ett arbetssätt för styrning som kan användas i agila projekt. Det generella syftet med Kanban är att synliggöra och optimera arbetsflödet inom en mervärdeskedja. Kanban bygger på tre instrument [Linz14]:

- Kanbantavla: Mervärdeskedjan kan synliggöras genom en Kanbantavla. I varje kolumn visas ett arbetsområde som utgörs av en uppsättning relaterade aktiviteter, till exempel utveckling eller testning. Det som ska utvecklas eller de uppgifter som ska utföras symboliseras av markörer som flyttas från vänster till höger över tavlan, via alla områden.
- Begränsning av pågående arbete: Det finns en begränsning för hur många aktiva arbetsuppgifter som får pågå parallellt. Detta styrs genom att ett område och/eller hela tavlan som mest får innehålla ett visst antal markörer. När det finns ledig kapacitet inom ett område tar medarbetaren en markör från det föregående området.
- Ledtid: Kanban används för att optimera det kontinuerliga flödet av arbetsuppgifter genom att minimera (den genomsnittliga) ledtiden för hela mervärdeskedjan.

Kanban har vissa likheter med Scrum. Inom båda dessa ramverk använder man visualisering av pågående arbetsuppgifter (t.ex. på en offentlig whiteboard-tavla) för att göra projektinnehållet och -förloppet transparent. Uppgifter som ännu inte schemalagts finns i en backlogg och flyttas ut på Kanbantavlan så snart det finns ledig plats (produktionskapacitet).

Iterationer (sprintar) kan användas i Kanban, men det är inte obligatoriskt. Kanbanprocessen tillåter driftsättning av leverabler var för sig, i stället för som delar av en slutleverans. Timeboxing för synkronisering är också valfritt till skillnad från Scrum, där alla uppgifter inom en sprint synkroniseras.

1.2.2 Gemensamt framtagande av användarberättelser

Bristande specifikationer är en vanlig anledning till att projekt misslyckas. Problem med specifikationer kan bero på att användarna inte inser vad de verkligen behöver, att det saknas en heltäckande vision för systemet, överflödiga eller motstridiga funktioner och andra typer av kommunikationsproblem. Vid agil utveckling skrivs användarberättelser för att synliggöra olika behov för såväl utvecklare som testare och verksamhetsrepresentanter. Under en sekventiell utvecklingsprocess åstadkommer man den här samsynen kring funktioner genom formell granskning efter att kravspecifikationerna har skrivits. Under en agil utvecklingsprocess skapar man i stället samsyn genom täta, informella granskningar som pågår parallellt med utformningen av kravspecifikationerna.

Användarberättelserna måste utgå från såväl funktionella som icke-funktionella egenskaper. Varje berättelse ska också innefatta acceptanskriterier för dessa egenskaper. De här kriterierna definieras genom samarbete mellan verksamhetsrepresentanter, utvecklare och testare. De ger utvecklarna och testarna en bättre överblick över den funktion som verksamhetsrepresentanterna ska validera. I det agila teamet anses en uppgift ha slutförts när de definierade acceptanskriterierna har uppfyllts.

Ofta kan testaren med sitt unika perspektiv bidra till användarberättelsen genom att identifiera detaljer som saknas eller krav som gäller icke-funktionella egenskaper. Testaren kan också bidra genom att ställa öppna frågor till verksamhetsrepresentanterna om användarberättelsen, föreslå olika sätt att testa användarberättelsen och bekräfta acceptanskriterierna.

Vid det gemensamma framtagandet av användarberättelsen kan man t.ex. använda tekniker som brainstorming och mindmapping. Testaren kan använda INVEST-tekniken [INVEST]:

- Independent (oberoende)
- Negotiable (förändringsbar)
- Valuable (värdefull)
- Estimable (uppskattningsbar)
- Small (liten)
- Testable (testbar)

Konceptet 3C [Jeffries00] definierar en användarberättelse som en samverkan av tre delar:

- Card (berättelsekort): Kortet är det fysiska medium som används för en användarberättelse. Här definieras kravet, hur viktigt det är, hur lång tid utvecklings- och testarbetet förväntas ta samt acceptanskriterierna för berättelsen. Beskrivningen måste vara korrekt eftersom den används i produktbackloggen.
- Conversation (konversation): I konversationen beskrivs hur programvaran kommer att användas. Konversationen kan vara skriftlig eller muntlig. Testare har en annan utgångspunkt än utvecklare och verksamhetsrepresentanter [ISTQB_FL_SYL] och kan därför bidra med värdefull information under utbytet av tankar, åsikter och erfarenheter. Konversationen inleds under leveransplaneringsfasen och fortsätter när användarberättelsen schemaläggs.
- Confirmation (bekräftelse): De acceptanskriterier som diskuteras under konversationen används för att avgöra när användarberättelsen är färdig. Acceptanskriterierna kan täcka flera användarberättelser. Både positiva och negativa tester ska utföras för att avgöra om kriterierna är uppfyllda. Under bekräftelsefasen kan olika deltagare inta testarrollen, till exempel utvecklare eller specialister som fokuserar på prestanda, säkerhet, interoperabilitet och andra kvalitetsfaktorer. För att en användarberättelse ska bekräftas som färdig ska de definierade acceptanskriterierna testas så att man kan se att de är uppfyllda.

Olika agila team dokumenterar sina användarberättelser på olika sätt. Oavsett hur man väljer att dokumentera användarberättelser ska dokumentationen vara koncis, tillräcklig och innefatta all nödvändig information.

1.2.3 Retrospektivmöten

Vid agil utveckling används retrospektivmöten: en typ av möte som hålls i slutet av varje iteration för att diskutera vad som har fungerat bra, vad som kan förbättras och hur man kan dra nytta av de framgångsrika delarna i kommande iterationer. Vid retrospektivmöten pratar man om processen, människorna som har deltagit, organisationer, relationer och verktyg. Regelbundna retrospektivmöten som åtföljs av lämpliga uppföljningsaktiviteter är avgörande för ett fungerande självorganiserande team och för att arbetet med utveckling och testning ska kunna förbättras kontinuerligt.

Retrospektivmöten kan leda till beslut om testrelaterade förbättringar som kan öka effektiviteten och produktiviteten, ge ökad kvalitet i testfallen och ökad teamnöjdhet. Dessutom kan man prata om testbarheten hos programvarorna, användarberättelserna, features och systemgränssnitten. Rotorsaksanalyser av defekter kan leda till förbättrade processer för testning och utveckling. Generellt bör ett team endast implementera ett fåtal förbättringar under en iteration, så att förbättringsprocessen sker gradvis och kontrollerat.

Exakt när och hur retrospektivmötet ska hållas beror på vilken agil metod som används. Verksamhetsrepresentanter och teammedlemmarna är deltagare i mötet och en koordinator ordnar och leder mötet. I vissa fall kan teamet också bjuda in ytterligare deltagare till mötet.

Testarna bör ha en framträdande roll under retrospektivmöten. Testarna ingår i teamet och bidrar med sitt unika perspektiv [ISTQB_FL_SYL], avsnitt 1.5. Testning utförs i varje iteration och är avgörande för att projektet ska bli framgångsrikt. Alla teammedlemmar, såväl testare som andra, kan bidra med indata om både test och andra aktiviteter.

Retrospektivmöten ska hållas i professionell anda och i en miljö som präglas av ömsesidigt förtroende. Kännetecknen på ett framgångsrikt retrospektivmöte är samma som för andra typer av granskningar, enligt beskrivningen i kursplanen för grundnivå [ISTQB_FL_SYL], avsnitt 3.2.

1.2.4 Kontinuerlig integration

För att ett produktinkrement ska kunna levereras måste teamet ha åstadkommit tillförlitlig, fungerande, integrerad programvara vid slutet av varje iteration. Kontinuerlig integration är ett sätt att åstadkomma detta. Det innebär att alla genomförda förändringar slås samman och alla ändrade komponenter integreras regelbundet, minst en gång per dag. Konfigurationshantering, kompilering, programvarubygge, distribution och testning kombineras i en enda automatiserad och upprepningsbar process. Eftersom utvecklarna ständigt och kontinuerligt integrerar sitt arbete, skapar byggen och testar kan defekter i koden upptäckas tidigare.

Efter att utvecklarna har programmerat, avlusat och checkat in koden på en gemensam lagringsplats för källkod vidtar den kontinuerliga integrationsprocessen med följande automatiserade aktiviteter:

- Statisk kodanalys: exekvera statisk kodanalys och rapportera resultaten
- Kompilering: kompilering och länkning av koden för att generera körbara filer
- Komponenttest: genomföra komponenttester, kontroller av kodtäckning och rapportera testresultaten
- Distribution: installera bygget i en testmiljö
- Integrationstest: genomföra integrationstester och rapportera resultaten
- Rapport (dashboard): publicera status för alla dessa aktiviteter på en offentlig plats, eller skicka statusmeddelanden via e-post till teamet

En automatiserad bygg- och testprocess genomförs dagligen så att eventuella integrationsfel upptäcks tidigt och snabbt. Kontinuerlig integration gör att agila testare kan köra automatiserade tester regelbundet, i vissa fall som en del av själva processen för kontinuerlig integration, och skicka snabb återkoppling till teamet om kodens kvalitet. De här testresultaten är synliga för alla teammedlemmar, framför allt om automatiserade rapporter integreras i processen. Automatiserad regressionstestning kan göras kontinuerligt under hela iterationen. Väl utformade automatiserade regressionstester innefattar så mycket av funktionaliteten som möjligt, inklusive användarberättelser som levererats under tidigare iterationer. En hög täckningsgrad i de automatiserade regressionstesterna underlättar bygge (och testning) av stora, integrerade system. När regressionstestningen automatiseras kan agila testare i

stället koncentrera sig på att manuellt testa nya funktioner, införda ändringar och utföra omtestning av felrättningar.

Utöver automatiserad testning använder företag som arbetar med kontinuerlig integration ofta byggverktyg för att implementera en kontinuerlig kvalitetskontroll. Sådana verktyg kan användas för att köra komponent- och integrationstester och dessutom ytterligare statiska och dynamiska tester, för att mäta och profilera prestanda, extrahera och formatera dokumentation från källkoden och underlätta manuella kvalitetssäkringsprocesser. Den här ständiga, kontinuerliga kvalitetskontrollen ersätter den traditionella modellen där kvalitetskontroll utförs efter att allt utvecklingsarbete har slutförts. Syftet är att förbättra produktkvaliteten och förkorta tiden fram till leverans.

Byggverktyg kan länkas till verktyg för automatisk distribution som kan hämta rätt bygge från servern för kontinuerlig integration eller byggservern och distribuera det i en eller flera utvecklings-, test- eller produktionsmiljöer. På så sätt kan man minska risken för misstag och förseningar som annars kan uppstå om specialiserad personal eller programmerare måste installera byggena.

Kontinuerlig integration kan ge följande fördelar:
- Möjliggör tidigare upptäckt och enklare rotorsaksanalys vid integrationsproblem och motstridiga ändringar
- Ger utvecklingsteamet regelbunden återkoppling om hur koden fungerar
- Gör att den version av programvaran som testas aldrig är mer än en dag äldre än den version som utvecklas
- Minskar regressionsrisken som uppstår när koden omstruktureras vid snabba omtestningar av kodbasen som görs efter varje (liten) uppsättning med ändringar
- Gör arbetet tryggare: alla kan vara säkra på att varje dags utvecklingsarbete vilar på en stabil grund
- Tydliggör varje framsteg mot färdigställandet av produktinkrementet: en morot för både utvecklare och testare
- Eliminerar de schemarisker som är förknippade med big-bang-integration
- Gör att körbar programvara är tillgänglig hela tiden under hela iterationen för testning, demonstrationer eller utbildning
- Minskar mängden repetitiva manuella testuppgifter
- Ger snabb återkoppling om beslut som fattats för att förbättra kvaliteten och testningen

Kontinuerlig integration medför dock också en del risker och utmaningar:
- Verktyg för kontinuerlig integration måste introduceras och underhållas
- Processen för kontinuerlig integration måste definieras och etableras
- Automatiserade tester kräver ytterligare resurser och rutinerna kan vara komplexa att etablera
- Omfattande testtäckning är ett måste för att uppnå fördelarna med automatiserad testning
- Risk för att teamet förlitar sig alltför mycket på komponenttester och arbetar för lite med system- och acceptanstester

Kontinuerlig integration förutsätter att verktyg används, bland annat verktyg för testning, för automatisering av byggprocessen och för versionskontroll.

1.2.5 Leverans- och iterationsplanering

Planering är en kontinuerlig aktivitet (vilket också nämns i kursplanen för grundnivå, [ISTQB_FL_SYL]) och detta gäller även agila livscykler. Under agila livscykler sker två typer av planering: leveransplanering och iterationsplanering.

Horisonten vid leveransplanering ligger vid den planerade leveransen av en produkt, vanligtvis ett antal månader framåt i tiden från projektstarten. Leveransplanering innebär att man definierar och omdefinierar produktbackloggen och det kan också innebära att större användarberättelser delas in i en uppsättning mindre berättelser. Leveransplanering utgör grunden för ett angreppssätt för test och en testplan som innefattar alla iterationer. Leveransplanering sker på en hög nivå.

Under leveransplaneringen etablerar och prioriterar verksamhetsrepresentanterna användarberättelserna för leveransen i samarbete med resten av teamet (se avsnitt 1.2.2). Utifrån de här användarberättelserna identifieras projekt- och kvalitetsrisker och en övergripande uppskattning av arbetsmängden genomförs (se avsnitt 3.2).

Testarna deltar också i leveransplaneringen och bidrar med mervärde framför allt vid följande uppgifter:
- Definierar testbara användarberättelser, inklusive acceptanskriterier
- Ger information vid analyser av projekt- och kvalitetsrisker
- Uppskattar hur mycket testarbete som krävs för användarberättelserna
- Definierar nödvändiga testnivåer
- Planerar testningen för leveransen

När leveransplaneringen har slutförts inleds iterationsplaneringen för den första iterationen. Iterationsplaneringen sträcker sig till slutet av den aktuella iterationen och planeringen är kopplad till backloggen för iterationen.

Under iterationsplaneringen väljer teamet användarberättelser från den prioriterade produktbackloggen, gör dem mer detaljerade, genomför en riskanalys för dem och uppskattar hur mycket arbete som krävs för varje användarberättelse. Om en användarberättelse är för vag och det inte går att förtydliga den kan teamet välja att avvisa den och använda nästa användarberättelse i prioritetsordningen. Verksamhetsrepresentanterna måste besvara teamets frågor om varje användarberättelse så att teamet förstår vad de ska göra och hur varje berättelse ska testas.

Hur många berättelser som väljs ut beror på den fastställda hastigheten för teamet och den uppskattade storleken på de valda användarberättelserna. När innehållet i iterationen har fastställts delas användarberättelserna upp i uppgifter som utförs av teammedlemmarna.

Testarna deltar också i iterationsplaneringen och bidrar med mervärde framför allt vid följande uppgifter:
- Bidrar till den detaljerade riskanalysen av användarberättelser
- Fastställer testbarheten för användarberättelserna
- Skapar acceptanstester för användarberättelserna
- Delar upp användarberättelserna i uppgifter (främst testuppgifter)
- Uppskattar hur mycket testarbete som krävs för alla testuppgifter
- Identifierar funktionella och icke-funktionella aspekter av systemet som ska testas
- Stödjer och deltar i testautomatisering på flera olika testnivåer

Leveransplanen kan förändras allt eftersom projektet fortskrider, och det kan bli aktuellt att ändra enskilda användarberättelser i produktbacklogen. Sådana förändringar kan föranledas av interna eller externa faktorer. Interna faktorer kan till exempel vara leveranskapacitet, hastighet och tekniska problem. Externa faktorer kan till exempel vara att nya marknader, affärsmöjligheter eller konkurrenter dyker upp, eller att verksamheten hotas på ett sätt som kan påverka leveransmål och/eller deadlines. Även iterationsplaner kan ändras under loppet av en iteration. Det kan till exempel visa sig att en användarberättelse som bedömdes som enkel under uppskattningsfasen är mer komplex än förväntat.

De här förändringarna kan innebära utmaningar för testarna. Testarna måste hela tiden ha en helhetsbild av hela leveransen i huvudet för att planera testningen, och de måste ha en tillräcklig testbas och testorakel i varje iteration för att utveckla testningen, enligt beskrivningen i kursplanen för grundnivå: [ISTQB_FL_SYL], avsnitt 1.4. Testarna måste ha tillgång till all nödvändig information i ett tidigt skede och samtidigt måste de vara beredda på förändringar, i enlighet med de agila principerna. Det här är en svår balansgång som kräver noggrant genomtänkta beslut om teststrategier och testdokumentation. Läs mer om utmaningar för agil testning i [Black09], kapitel 12.

Leverans- och iterationsplanering ska innefatta testplanering och planering av utvecklingsarbete. Här är några testrelaterade frågor som bör tas upp:
- Vad som ska testas, hur omfattande tester som ska göras, målsättningar för testningen (samtliga dessa beslut ska också motiveras).
- Vilka teammedlemmar som ska utföra testuppgifterna.
- Vilken testmiljö och vilka testdata som behövs, när de behövs och huruvida tillägg eller ändringar i testmiljön och/eller testdata kommer att göras före eller under projektet.

- Tidsplanering och ordningsföljd för testuppgifterna, eventuella beroenden mellan dem och förutsättningar för alla testuppgifter för funktionella och icke-funktionella egenskaper (t.ex. hur ofta regressionstester ska köras, vilka funktioner som är beroende av andra funktioner eller testdata osv.), inklusive på vilket sätt testuppgifterna är kopplade till och beroende av utvecklingsuppgifterna.
- Vilka projekt- och kvalitetsrisker som ska behandlas (se avsnitt 3.2.1).

Dessutom bör teamet under det generella uppskattningsarbetet beakta hur lång tid och hur mycket arbete som krävs för att utföra de nödvändiga testuppgifterna.

2. Grundläggande principer, arbetssätt och processer för agil testning – 105 min.

Nyckelord
verifiering av bygge, konfigurationselement, konfigurationshantering

Inlärningsmål för grundläggande principer, arbetssätt och processer för agil testning

2.1 Skillnaderna mellan traditionell testning och agil testning

FA-2.1.1 (K2) Beskriva skillnaderna mellan testuppgifter i agila projekt och andra typer av projekt
FA-2.1.2 (K2) Beskriva hur test- och utvecklingsuppgifter integreras i agila projekt
FA-2.1.3 (K2) Beskriva vilken funktion oberoende testning har i agila projekt

2.2 Teststatus i agila projekt

FA-2.2.1 (K2) Beskriva de verktyg och metoder som används för att kommunicera status för testning i agila projekt, inklusive testprogress och produktkvalitet
FA-2.2.2 (K2) Beskriva processen för hur tester kan utvecklas under loppet av flera iterationer och förklara varför automatiserade tester är viktigt för att hantera regressionsrisken i agila projekt

2.3 Testarens roll och färdigheter i ett agilt team

FA-2.3.1 (K2) Förstå vilka färdigheter (sociala, domän- och testrelaterade) som krävs av en testare i ett agilt team
FA-2.3.2 (K2) Förstå testarens roll inom ett agilt team

2.1 Skillnaderna mellan traditionell testning och agil testning

I kursplanen för grundnivå [ISTQB_FL_SYL] och i [Black09] beskrivs hur testuppgifter är relaterade till utvecklingsuppgifter och hur testningen varierar mellan olika livscykelmodeller. För att testare ska kunna arbeta effektivt krävs en förståelse för skillnaden mellan testning i traditionella livscykelmodeller (t.ex. sekventiella modeller som V-modellen eller iterativa modeller som RUP) och agila livscykelmodeller. Agila modeller skiljer sig från traditionella modeller på flera sätt: test- och utvecklingsuppgifter integreras på ett annat sätt, andra projektarbetsprodukter, namn, start- och avslutskriterier för olika testnivåer används, och verktyg och oberoende testning används på andra sätt.

Testaren behöver inse att olika organisationer använder livscykler på mycket varierande sätt. Eventuella avvikelser från den ideala agila livscykeln (se avsnitt 1.1) kan bero på en genomtänkt anpassning av arbetssätten. Förmågan att anpassa sig efter förutsättningarna i det aktuella projektet, inklusive de metoder som används för programvaruutveckling, är en viktig framgångsfaktor för testare.

2.1.1 Test- och utvecklingsuppgifter

En av de främsta skillnaderna mellan traditionella livscykler och agila livscykler är konceptet med mycket korta iterationer, där varje iteration ger fungerande programvara med värdefulla funktioner för intressenterna. I början av projektet genomförs en så kallad leveransplanering. Därefter följer ett antal iterationer. Varje iteration inleds med en iterationsplanering. När iterationens omfattning har fastställts utvecklas de valda användarberättelserna och de integreras med systemet och testas. Iterationerna är mycket dynamiska och utvecklings-, integrerings- och testuppgifter utförs parallellt och överlappande inom varje iteration. Testning pågår under hela iterationen och inte enbart som en slutlig kontroll.

Testare, utvecklare och intressenter bidrar alla till testprocessen, precis som i traditionella livscykler. Utvecklarna gör komponenttester när de utvecklar funktioner baserat på användarberättelserna, och därefter testas funktionerna av testarna. Intressenterna testar också berättelserna under implementeringsfasen. Intressenterna kan använda skriftliga testfall eller helt enkelt experimentera med funktionen på olika sätt för att ge snabb återkoppling till utvecklingsteamet.

I vissa fall kan regelbundna stabiliseringsiterationer användas för att åtgärda eventuella kvardröjande defekter och andra typer av tekniska skulder. Det bästa angreppssättet är dock att ingen funktion ska anses vara färdig förrän den har integrerats och testats tillsammans med systemet [Goucher09]. Ett annat bra arbetssätt är att ta itu med defekter som finns kvar från en iteration i början av nästa iteration, som en del av backloggen för den iterationen (denna metod kallas för att ”fixa buggarna först”). Vissa menar dock att detta riskerar att göra arbetsmängden inom iterationen svårbedömd, och att det blir svårare att uppskatta när övriga delar av iterationen kan slutföras. I slutet av projektet, när alla iterationer är klara, kan man planera in en uppsättning leveransaktiviteter för att göra programvaran klar för leverans. Ibland görs dock leveranser löpande i slutet av varje iteration.

Om riskbaserad testning används som en av teststrategierna görs en riskanalys på hög nivå under leveransplaneringen, där testarna ofta är drivande. De specifika kvalitetsriskerna som är förknippade med varje iteration identifieras och utvärderas dock under iterationsplaneringen. Den här riskanalysen kan påverka ordningsföljden inom utvecklingsarbetet och prioriteringen och omfattningen av funktionstestningen. Dessutom påverkas uppskattningen av hur mycket testarbete som krävs för varje funktion (se avsnitt 3.2).

Vid vissa agila arbetsmetoder (till exempel extrem programmering) används partestning. Det kan till exempel innebära att två testare testar en funktion tillsammans, eller att en testare och en utvecklare samarbetar för att utveckla och testa en funktion. Partestning kan vara svårt om alla inte sitter på samma plats, men det finns processer och verktyg som kan underlätta samarbete på distans. Mer information om distribuerat arbete finns i [ISTQB_ALTM_SYL], avsnitt 2.8.
Testare kan också fungera som coacher för testning och kvalitet inom teamet, dela med sig av sina testkunskaper och stödja kvalitetssäkringen inom teamet. Detta främjar en känsla av att alla äger produktkvaliteten.

Automatiserade tester på alla testnivåer förekommer i många agila team. Detta kan innebära att testare får arbeta med att skapa, exekvera, övervaka och underhålla automatiserade tester och resultat. Eftersom automatiserade tester används i så stor utsträckning utgörs en stor del av den manuella

testningen i agila projekt av erfarenhets- och defektbaserad testning som programvaruattacker, utforskande testning och felgissning (se [ISTQB_ALTA_SYL], avsnitt 3.3 och 3.4 samt [ISTQB_FL_SYL], avsnitt 4.5). Utvecklarna fokuserar på att skapa komponenttester, och testarna bör därför fokusera på att skapa automatiserade integrationstester, systemtester och systemintegrationstester. I agila projekt är det därför vanligt med testare som har gedigna tekniska kunskaper och erfarenhet av automatiserade tester.

En av de viktigaste agila principerna är att ändringar kan ske när som helst under projektet. Därför är det vanligt med lättviktsdokumentation för arbetsprodukter inom agila projekt. Alla ändringar av befintliga funktioner påverkar testningen, framför allt regressionstestningen. Automatiserad testning kan underlätta det extra testarbete som förändringar kan medföra. Det är dock viktigt att mängden förändringar inte överskrider projektteamets förmåga att hantera de risker som följer av förändringarna.

2.1.2 Projektarbetsprodukter

De projektarbetsprodukter som är av störst intresse för agila testare kan delas in i tre huvudkategorier:

1. Verksamhetsorienterade arbetsprodukter som beskriver vad som behövs (t.ex. kravspecifikationer) och hur de ska användas (t.ex. användardokumentation)
2. Utvecklingsarbetsprodukter som beskriver hur systemet är uppbyggt (t.ex. relationsdiagram över databasobjekt), som utgör själva systemimplementeringen (t.ex. kod) eller som utvärderar enskilda koddelar (t.ex. automatiserade komponenttester)
3. Testarbetsprodukter som beskriver hur systemet testas (t.ex. teststrategier och planer), som utgör själva testningen av systemet (t.ex. manuella och automatiserade tester) eller som presenterar testresultat (t.ex. test-dashboards, se avsnitt 2.2.1)

I agila projekt strävar man ofta efter att inte skapa alltför stora mängder dokumentation. I stället fokuserar man på att skapa fungerande programvara och automatiserade tester som visar att programvaran uppfyller kraven. Den här strävan efter att minska mängden dokumentation gäller endast dokumentation som inte ger kunden mervärde. I framgångsrika agila projekt hittar man balansen mellan att förbättra effektiviteten genom att minska mängden dokumentation och samtidigt tillhandahålla tillräcklig dokumentation som stöd för verksamheten, testning, utveckling och underhåll. Under leveransplaneringen måste teamet fatta beslut om vilka arbetsprodukter som är nödvändiga och vilken nivå av dokumentation som behövs för dem.

Två vanliga verksamhetsorienterade arbetsprodukter i agila projekt är användarberättelser och acceptanskriterier. Användarberättelser fungerar som kravspecifikationer i agila projekt. De förklarar hur systemet ska fungera med utgångspunkt från en enda, avgränsad feature eller funktion. En användarberättelse bör definiera en funktion som är tillräckligt liten för att kunna slutföras under en enda iteration. En större samling relaterade funktioner eller en grupp underfunktioner som utgör en komplex funktion kan kallas för ”epic”. En epic kan innefatta användarberättelser för olika utvecklingsteam. En användarberättelse kan till exempel beskriva vad som krävs på API-nivå (mellanprogram) medan en annan berättelse beskriver vad som behövs på gränssnittsnivå (applikation). De här samlingarna kan utvecklas under en serie iterationer. Varje epic och dess användarberättelser ska vara kopplad till acceptanskriterier.

En vanlig arbetsprodukt för en utvecklare i ett agilt projekt är kod. Agila utvecklare skapar också ofta automatiserade komponenttester. Sådana tester kan skapas efter att kod har utvecklats. I vissa fall arbetar dock utvecklarna gradvis med att skapa tester redan innan en viss del av koden skrivs, så att man sedan kan verifiera att den skrivna koden fungerar så som det är tänkt. Det här angreppssättet kallas för ”test first” eller testdriven utveckling, men i själva verket bör testerna snarare ses som en typ av exekverbara designspecifikationer på låg nivå än som verkliga tester [Beck02].

En vanlig arbetsprodukt för en testare i ett agilt projekt är automatiserade tester samt dokument som testplaner, kataloger över kvalitetsrisker, manuella tester, felrapporter och testresultatloggar. Precis som i traditionella livscykelmodeller ska de här dokumenten skrivas enligt en lättviktsprincip, så att de inte blir alltför omfattande. Testare skapar också mätetal för test utifrån felrapporter och testresultatloggar, enligt samma lättviktsprincip som dokumentationen.

I vissa agila modeller (framför allt för reglerade, säkerhetskritiska, distribuerade eller mycket komplexa projekt och produkter) behöver de här arbetsprodukterna formaliseras mer. Vissa teams behöver till

exempel omvandla användarberättelser och acceptanskriterier i enlighet med mer formella kravspecifikationer. Vertikala och horisontella spårbarhetsrapporter kan skapas för att uppfylla revisionskrav, regelverk och andra krav.

2.1.3 Testnivåer

Testnivåer är testuppgifter som är logiskt relaterade till varandra, ofta utifrån hur moget eller färdigt det testade objektet är.

I sekventiella livscykelmodeller definieras testnivåerna ofta så att avslutskriterierna för en nivå ingår i startkriterierna för nästa nivå. I vissa iterativa modeller gäller dock inte den här regeln. Testnivåerna kan överlappa varandra. Krav- och designspecifikationer och utvecklingsuppgifter kan också överlappa testnivåerna.

I vissa agila livscykelmodeller uppstår överlappning på grund av förändringar i krav, design och kod som kan uppkomma när som helst under en iteration. Inom Scrum tillåts i teorin inga förändringar i användarberättelserna efter iterationsplaneringen, men i praktiken sker detta ändå ibland. Under en iteration genomgår en användarberättelse vanligtvis följande testuppgifter:

- Komponenttestning, som vanligtvis utförs av utvecklaren
- Funktionsacceptanstestning, som ibland delas upp i två uppgifter:
    - Funktionsverifieringstestning, som ofta är automatiserad och kan utföras av utvecklare eller testare. Detta kan innefatta testning mot användarberättelsens acceptanskriterier
    - Funktionsvalideringstestning, som ofta görs manuellt. Här kan utvecklare, testare och intressenter samarbeta för att avgöra om funktionen är klar att använda, för att tydliggöra de framsteg som görs och för att inhämta återkoppling från intressenterna

Dessutom pågår ofta regressionstestning parallellt under hela iterationen. Detta innebär att automatiserade komponenttester och funktionsverifieringstester från den aktuella iterationen och tidigare iterationer utförs på nytt, vanligtvis inom ett ramverk för kontinuerlig integration.

I vissa agila projekt kan det finnas en systemtestnivå som inleds så snart den första användarberättelsen är klar för denna typ av testning. Detta kan innefatta tester av funktionella egenskaper, tester av icke-funktionella egenskaper för prestanda, tillförlitlighet och användbarhet och andra relevanta testtyper.

Agila team kan använda olika typer av acceptanstestning (termen används här enligt förklaringen i kursplanen för grundnivå [ISTQB_FL_SYL]). Intern alfatestning och extern betatestning kan förekomma antingen i slutet av varje iteration, efter att varje iteration har slutförts eller efter att en serie iterationer har slutförts. Användaracceptanstester, driftsacceptanstester och acceptanstester för kontrakt och förordningar kan också förekomma antingen i slutet av varje iteration, efter att varje iteration har slutförts eller efter att en serie iterationer har slutförts.

2.1.4 Test- och konfigurationshantering

I agila projekt används ofta automatiserade verktyg i stor utsträckning för att utveckla, testa och hantera programvara. Utvecklarna använder verktyg för statisk analys, komponenttestning och kodtäckning. Utvecklarna checkar regelbundet in koden och komponenttester i ett konfigurationshanteringssystem med automatiserade ramverk för byggen och testning. Inom de här ramverken kan ny programvara integreras kontinuerligt (kontinuerlig integration) med systemet, och statiska analyser och komponenttester körs regelbundet varje gång ny programvara checkas in [Kubaczkowski].

De automatiserade testerna kan även innefatta funktionella tester på integrations- och systemtestnivå. Sådana funktionella automatiserade tester kan skapas med exekveringsplattformar för funktionell testning, open source-verktyg för funktionstestning av användargränssnitt eller kommersiella verktyg, och de kan integreras med de automatiserade tester som körs som en del av ramverket för kontinuerlig integration. I vissa fall kan funktionella tester köras separat från (och mer sällan än) komponenttesterna, eftersom funktionella tester kan ta lång tid. Komponenttester kan till exempel köras varje gång ny programvara checkas in, medan längre funktionstester körs endast ett par gånger i veckan.

Ett mål med automatiserade tester är att bekräfta att bygget fungerar och kan installeras. Om ett automatiserat test misslyckas ska teamet åtgärda den bakomliggande defekten innan koden checkas

in nästa gång. Detta kräver ett medvetet arbete med testrapportering i realtid för att göra testresultaten synliga. På så sätt minskar man risken för dyra, ineffektiva cykler enligt modellen “bygga-installera-misslyckas-bygga om-installera om” som är vanliga i traditionella projekt, eftersom ändringar som får bygget att sluta fungera eller gör att det inte kan installeras upptäcks snabbt.

Automatiserade test- och byggverktyg underlättar hantering av de regressionsrisker som uppstår när förändringar sker ofta, något som kännetecknar agila projekt. Man bör dock inte förlita sig helt och hållet på automatiserad komponenttestning för att hantera de här riskerna eftersom komponenttestning oftast är mindre effektivt för att identifiera defekter [Jones11]. Automatiserade tester på integrations- och systemtestnivå måste också utföras.

2.1.5 Organisationsalternativ för oberoende testning

I kursplanen för grundnivå [ISTQB_FL_SYL] nämns att oberoende testare ofta är mer effektiva när det gäller att hitta defekter. I vissa agila team skapar utvecklarna många av testerna i form av automatiserade tester. En eller flera testare kan ingå i teamet och utföra mycket av testarbetet. Det finns dock risk för att testare som ingår i teamet inte har det oberoende och den objektivitet som behövs.

Andra agila team väljer därför att anlita helt oberoende, separata testteam och tilldelar arbete till dem efter behov under de sista dagarna av varje iteration. På så sätt säkerställs testarnas oberoende och möjligheterna till en verkligt objektiv utvärdering av programvaran ökar. Tidspress, bristande förståelse för de nya funktionerna i produkten och problem i relationen till intressenter och utvecklare kan dock orsaka problem med det här angreppssättet.

Ett tredje alternativ är att ha ett oberoende, separat testteam där testarna tilldelas jobb inom agila team på längre sikt redan i början av projektet. På så sätt kan testarna fortsätta att vara oberoende men samtidigt få en god förståelse för produkten och bygga upp goda relationer med övriga teammedlemmar. Dessutom kan det oberoende testteamet anlita specialiserade testare utanför det agila teamet för arbete med mer långsiktiga och/eller iterationsoberoende uppgifter, som utveckling av automatiserade testverktyg, testning av icke-funktionella egenskaper, att skapa och stödja testmiljöer och testdata och utföra testnivåuppgifter som inte passar in inom en iteration (t.ex. systemintegrationstestning).

2.2 Teststatus i agila projekt

Förändringstakten är hög inom agila projekt. Det innebär att teststatus, testprogressen och produktkvaliteten är under ständig utveckling, och att testarna därför måste hitta bra sätt att förmedla information till teamet så att de kan fatta de beslut som krävs för att varje iteration ska kunna slutföras med gott resultat. Förändringar kan dessutom påverka befintliga funktioner från tidigare iterationer. Därför måste både manuella och automatiserade tester uppdateras för att hantera regressionsrisker.

2.2.1 Kommunicera teststatus, testprogress och produktkvalitet

Arbetet inom agila team fortskrider stegvis genom att fungerande programvara skapas i slutet av varje iteration. För att teamet ska kunna avgöra när de kan åstadkomma färdig programvara måste de noggrant följa förloppet för alla arbetsobjekt inom iterationen och leveransen. Testare i agila team kan använda olika metoder för att registrera testprogress och teststatus, inklusive resultat av automatiserade tester, förloppet för testuppgifter och användarberättelser på den agila aktivitetstavlan samt burndown charts som visar teamets progress. Den här informationen kan sedan förmedlas till resten av teamet via till exempel wiki-dashboards och dashboard-liknande e-postmeddelanden samt verbalt vid ståmöten. Agila team kan använda verktyg där statusrapporter skapas automatiskt utifrån testresultat och uppgiftsprogress, rapporter som i sin tur skickas direkt till wiki-dashboards och e-postmeddelanden. Den här metoden innebär också att mätetal samlas in från testprocessen vilket kan vara användbart vid processförbättring. Genom att teststatus till stor del förmedlas automatiskt får testarna tid över för att utforma och exekvera fler testfall.

Teamet kan använda så kallade burndown charts för att spåra förloppet för hela leveransen och inom varje iteration. Ett burndown chart [Crispin08] motsvarar den arbetsmängd som återstår i relation till den allokerade tiden för leveransen eller iterationen.

För att skapa en ögonblicklig och detaljerad visuell återgivning av aktuell status för hela teamet, inklusive teststatus, kan teamet använda agila aktivitetstavlor. På aktivitetstavlan visas berättelsekorten, utvecklings- och testuppgifter och andra uppgifter som skapats under iterationsplaneringen (se avsnitt 1.2.5), ofta med färgkodning som visar de olika uppgiftstyperna. Under iterationen spåras arbetsprogressen genom att uppgifterna flyttas till olika kolumner på aktivitetstavlan, till exempel *att göra*, *pågående arbete*, *ska verifieras* och *färdigt*. Agila team kan använda verktyg för att uppdatera och underhålla berättelsekort och aktivitetstavlor, och på så sätt automatisera arbetet med dashboards och statusuppdateringar.

Testuppgifterna på aktivitetstavlan är kopplade till de acceptanskriterier som definierats för användarberättelserna. När skript för automatiserade tester, manuella tester och utforskande tester för en testningsuppgift når godkänd status flyttas uppgiften till kolumnen *färdigt* på aktivitetstavlan. Hela teamet granskar regelbundet status för aktivitetstavlan, ofta under de dagliga ståmötena, för att se till att uppgifterna flyttas över tavlan i en godtagbar takt. Om någon uppgift (till exempel en testuppgift) inte ändrar status över en längre tid tittar teamet närmare på uppgiften och tar itu med eventuella problem.

Vid de dagliga ståmötena deltar alla medlemmar från det agila teamet, inklusive testarna. Vid det här mötet berättar alla om aktuell status för det arbete de gör. Varje medlem svarar på följande frågor [Agile Alliance Guide]:

- Vad har du slutfört sedan det förra mötet?
- Vad planerar du att slutföra fram till nästa möte?
- Vilka hinder finns det för ditt arbete?

På de dagliga ståmötena går man igenom alla eventuella problem som kan hindra testarbetet så att hela teamet är medvetna om dem och kan vidta åtgärder.

För att förbättra den allmänna produktkvaliteten genomför många agila team undersökningar om kundnöjdhet för att inhämta återkoppling om hur väl produkten uppfyller kundernas förväntningar. Teamet kan också använda andra mätetal som liknar de som används i traditionella utvecklingscykler, till exempel andelen godkända/underkända tester, antal upptäckta defekter, resultat av om- och regressionstestning, defekttäthet, hittade och korrigerade defekter, kravtäckningsgrad, risktäckningsgrad, kodtäckningsgrad och kodomsättning (code churn), för att förbättra produktkvaliteten. Oavsett vilken typ av livscykelmodell som används ska alla mätetal vara relevanta och underlätta beslutsfattande. Mätetal ska inte användas för att belöna, bestraffa eller hänga ut enskilda teammedlemmar.

2.2.2 Hantering av regressionsrisker genom utveckling av manuella och automatiserade testfall

I ett agilt projekt växer produkten gradvis för varje slutförd iteration. Därför utökas också testningens omfattning. Utöver att testa de kodförändringar som utförts under den pågående iterationen behöver testarna också verifiera att ingen regression har påverkat funktioner som utvecklats och testats i tidigare iterationer. Risken för regression vid agil utveckling är hög eftersom kodomsättningen är stor (många kodrader läggs till, ändras och tas bort mellan olika versioner). Eftersom anpassning till förändring är en grundläggande agil princip kan man även göra ändringar i tidigare levererade funktioner för att uppfylla företagets behov. För att upprätthålla en god hastighet utan att en alltför stor teknisk skuld uppstår är det avgörande att teamen lägger tid på att utforma automatiserade tester på alla testnivåer så tidigt som möjligt i projektet. Det är också avgörande att alla testtillgångar som automatiserade tester, manuella testfall, testdata och andra testobjekt hålls uppdaterade för varje iteration. En stark rekommendation är att alla testtillgångar underhållas i ett konfigurationshanteringsverktyg för att möjliggöra versionskontroll, se till att alla teammedlemmar enkelt kan komma åt tillgångarna och göra det lättare att göra ändringar efter behov när funktionaliteten förändras, samtidigt som äldre information bevaras.

Eftersom det sällan är möjligt att upprepa alla tester, framför allt i agila projekt med stor tidspress, måste testarna allokera tid inom varje iteration för att granska manuella och automatiserade testfall från tidigare iterationer och den pågående iterationen för att välja ut testfall som kan vara lämpade att användas under regressionstestningen, och för att avveckla testfall som inte längre är relevanta. Tester som har skrivits i tidigare iterationer för att verifiera specifika funktioner kan vara mindre användbara i senare iterationer om funktionerna ändras, eller om nya funktioner introduceras som påverkar de äldre funktionerna.

Under granskningen av testfall ska testarna överväga vilka fall som passar för automatisering. Teamet bör automatisera så många tester som möjligt från tidigare iterationer och/eller leveranser som har påverkats av ändringar i den pågående iterationen. På så sätt kan man automatisera regressionstestningen och minska regressionsrisken med mindre arbetsinsats än vad manuell regressionstestning skulle kräva. På så sätt får testarna mer tid över för att göra grundligare tester av nya funktioner i den pågående iterationen.

Det är mycket viktigt att testarna snabbt kan identifiera och uppdatera testfall från tidigare iterationer och/eller leveranser som påverkas av de förändringar som har utförts i den pågående iterationen. Definitioner av hur teamet ska designa, skriva och lagra testfall ska tas fram under leveransplaneringen. Goda arbetssätt för testdesign och implementering bör införas tidigt och användas konsekvent. Effekten av dålig testdesign och implementeringsmetoder förstärks av tidspressen och de ständiga förändringarna inom varje iteration.

Användning av testautomatisering på alla testnivåer gör att det agila teamet kan ge snabb återkoppling om produktkvaliteten. Välskrivna automatiserade tester är en levande dokumentation av att systemet fungerar [Crispin08]. Genom att checka in automatiserade tester och resultaten av dem i systemet för konfigurationshantering i enlighet med versionerna för de olika produktbyggena kan det agila teamet se vilken funktionalitet som har testats och testresultaten för alla byggen och alla tidpunkter.

Automatiserade komponenttester körs innan källkoden checkas in i huvudspåret i konfigurationshanteringssystemet för att se till att koden inte skadar programvarubygget. För att minska risken för byggstopp (vilket kan orsaka problem och fördröjningar för hel teamet) bör kod aldrig checkas in förrän alla automatiserade komponenttester är godkända. Automatiserade komponenttester ger omedelbar återkoppling om kvaliteten på koden och bygget, men inte om produktkvaliteten.

Automatiserade acceptanstester körs regelbundet som en del av den kontinuerliga integrationen för det fullständiga systembygget. Dessa tester körs alltid på ett fullständigt systembygge minst en gång per dag, men körs vanligtvis inte varje gång kod checkas in eftersom de tar längre tid än automatiserade komponenttester vilket kan fördröja kodincheckningen. Testresultaten av automatiserade acceptanstester ger återkoppling om produktkvaliteten med avseende på eventuell regression mot det senaste bygget, men de ger ingen information om den generella produktkvaliteten.

Automatiserade tester kan köras kontinuerligt på systemet. En delmängd med automatiserade tester som rör kritiska systemfunktioner och integrationspunkter bör skapas omedelbart efter att ett nytt bygge har lagts in i testmiljön. Dessa tester kallas sammantaget för verifiering av bygge. Resultaten av dessa tester ger omedelbar återkoppling om programvaran efter distributionen så att teamen inte förlorar tid på att testa byggen som inte fungerar.

De automatiserade tester som ingår i uppsättningen med regressionstester körs i allmänhet varje dag på det dagliga bygget i miljön för kontinuerlig integration, och dessutom varje gång ett nytt bygge distribueras till testmiljön. Om ett automatiserat regressionstest blir underkänt avbryter teamet arbetet och utreder varför testet underkändes. Problemet kan bero på planerade funktionsändringar som utförts under den aktuella iterationen. I så fall kan testet och/eller användarberättelsen behöva uppdateras enligt de nya acceptanskriterierna. Det kan också hända att testet behöver tas bort om ett annat test som täcker in de nya ändringarna har skapats. Om testet underkänns på grund av en defekt ska teamet åtgärda defekten innan de arbetar vidare med nya features.

Utöver testautomatisering kan även följande testaktiviteter automatiseras:

- Generering av testdata
- Inläsning av testdata i system
- Distribution av byggen till testmiljöer
- Återställning av en testmiljö (t.ex. databasen eller webbplatsdatafiler) till en fastställd konfiguration
- Jämförelse av utdata

Automatisering av de här uppgifterna minskar overheadkostnaderna och frigör tid så att teamet kan fokusera på att utveckla och testa nya funktioner.

2.3 Testarens roll och färdigheter i ett agilt team

I ett agilt team är det viktigt att testarna har ett nära samarbete med alla övriga teammedlemmar och intressenter. Detta innebär att testaren måste ha vissa kompetenser och utföra vissa uppgifter inom det agila teamet.

2.3.1 Den agila testarens kompetens

En agil testare bör ha de färdigheter som beskrivs i kursplanen för grundnivå [ISTQB_FL_SYL]. Utöver dessa färdigheter måste testare i ett agilt team ha kunskaper om testautomatisering, testdriven utveckling, acceptanstestdriven utveckling, white-box- och black-box-testning samt erfarenhetsbaserad testning.

Eftersom agil metodik i hög grad bygger på samarbete, kommunikation och interaktion mellan teammedlemmarna och mellan teamet och externa intressenter bör testare i agila team ha god social kompetens. Testare i agila team bör:

- vara positiva och lösningsorienterade gentemot teammedlemmar och intressenter
- tänka kritiskt, skeptiskt och kvalitetsorienterat om produkten
- aktivt arbeta för att inhämta information från intressenter (i stället för att helt förlita sig på skriftliga specifikationer)
- utvärdera och rapportera om testresultat, testprogress och produktkvalitet på ett korrekt sätt
- arbeta effektivt för att definiera testbara användarberättelser, framför allt acceptanskriterier, tillsammans med kundrepresentanter och intressenter
- samarbeta inom teamet och arbeta parvis med programmerare och andra teammedlemmar
- reagera snabbt på förändringar genom att t.ex. ändra, lägga till eller förbättra testfall
- planera och organisera sitt eget arbete

Kontinuerlig kompetensutveckling, även inom social kompetens, är viktigt för alla testare inklusive de som arbetar i agila team.

2.3.2 Testarens roll i ett agilt team

En testare i ett agilt team ansvarar för att utföra uppgifter som ger återkoppling om teststatus, testprogress och produktkvalitet men också om processkvalitet. Utöver de uppgifter som beskrivs på andra platser i den här kursplanen handlar det om följande uppgifter:

- att förstå, implementera och uppdatera teststrategin
- att mäta och rapportera täckningsgrad för testningen inom alla tillämpliga täckningsgradsaspekter
- att se till att rätt testverktyg används på rätt sätt
- att konfigurera, använda och hantera testmiljöer och testdata
- att rapportera defekter och samarbeta med teamet för att åtgärda dem
- att handleda andra teammedlemmar inom relevanta delar av testningen
- att se till att lämpliga testuppgifter schemaläggs under leverans- och iterationsplanering
- att aktivt samarbeta med utvecklare och intressenter för att tydliggöra krav, framför allt vad gäller testbarhet, överensstämmelse och fullständighet
- att delta aktivt i retrospektivmöten med teamet och ge förslag på (och implementera) förbättringar

Inom ett agilt team delar alla medlemmar på ansvaret för produktkvaliteten och alla bidrar till testrelaterade uppgifter.

Här är några testrelaterade organisationsrisker för agila team:

- Om testarna arbetar alltför nära utvecklarna kan de tappa det kritiska tänkande som en testare behöver
- Om testarna blir vana vid eller håller tyst om ineffektiva eller lågkvalitativa arbetssätt inom teamet
- Om testarna inte kan hålla jämna steg med förändringstakten inom iterationer med hård tidspress

För att motverka de här riskerna kan teamet överväga de olika åtgärder för att bevara/öka testarnas oberoende som beskrivs i avsnitt 2.1.5.

3. Metoder, tekniker och verktyg för agil testning – 480 min.

Nyckelord
acceptanskriterier, utforskande testning, prestandatestning, produktrisk, kvalitetsrisk, regressionstestning, angreppssätt för test, testcharter, testuppskattning, automatiserad testexekvering, teststrategi, testdriven utveckling, ramverk för komponenttest

Inlärningsmål för metoder, tekniker och verktyg för agil testning

3.1 Metoder för agil testning

FA-3.1.1 (K1) Komma ihåg koncepten bakom testdriven utveckling, acceptanstestdriven utveckling och beteendedriven utveckling
FA-3.1.2 (K1) Komma ihåg koncepten bakom testpyramiden
FA-3.1.3 (K2) Sammanfatta testkvadranter och hur de är kopplade till testnivåer och testtyper
FA-3.1.4 (K3) Utöva rollen som testare i ett Scrum-team i ett agilt projekt

3.2 Utvärdering av kvalitetsrisker och testuppskattning

FA-3.2.1 (K3) Utvärdera kvalitetsrisker inom ett agilt projekt
FA-3.2.2 (K3) Uppskatta mängden testarbete utifrån innehållet i iterationen och kvalitetsriskerna

3.3 Tekniker i agila projekt

FA-3.3.1 (K3) Tolka relevant information som stöd för testuppgifter
FA-3.3.2 (K2) Förklara för intressenter hur man definierar testbara acceptanskriterier
FA-3.3.3 (K3) Skriva testfall för acceptanstestdriven utveckling utifrån en given användarberättelse
FA-3.3.4 (K3) Skriva testfall för såväl funktionella som icke-funktionella egenskaper med black-box-testdesign utifrån givna användarberättelser
FA-3.3.5 (K3) Utföra utforskande testning som en del av testarbetet inom ett agilt projekt

3.4 Verktyg i agila projekt

FA-3.4.1 (K1) Komma ihåg de olika verktyg som är tillgängliga för testare utifrån hur de kan användas och vilka typer av uppgifter i agila projekt som de lämpar sig för

3.1 Metoder för agil testning

Det finns vissa testarbetssätt som kan vara till nytta i alla utvecklingsprojekt (både agila och andra typer) för att skapa produkter av hög kvalitet. Några exempel: att skriva tester i förväg för att beskriva programmets korrekta beteende, att tidigt fokusera på att förebygga, identifiera och åtgärda defekter och att se till att rätt typer av tester körs vid rätt tillfälle och på rätt testnivå. En agil testare bör sträva efter att införa de här arbetssätten så tidigt som möjligt. Testare i agila projekt spelar en viktig roll för att se till att dessa arbetssätt för testning används under hela livscykeln.

3.1.1 Testdriven utveckling, acceptanstestdriven utveckling och beteendedriven utveckling

Testdriven utveckling, acceptanstestdriven utveckling och beteendedriven utveckling är tre kompletterande tekniker som används inom agila team för att utföra testning på olika testnivåer. Alla dessa tekniker är exempel på en grundläggande testprincip: att det är viktigt att testa och kvalitetssäkra i ett så tidigt skede som möjligt, eftersom testerna definieras innan koden skrivs.

Testdriven utveckling
Testdriven utveckling (TDD) används för att utveckla kod med vägledning av automatiserade testfall. Så här fungerar processen för testdriven utveckling:

- Skapa ett test som beskriver hur programmeraren har tänkt sig den önskade funktionaliteten hos en liten kodbit
- Kör testet, som förstås blir underkänt eftersom koden inte finns
- Skriv koden, kör testet och upprepa tills testet blir godkänt
- Omstrukturera koden efter att testet blivit godkänt och kör testet på nytt för att säkerställa att det blir godkänt även för den omstrukturerade koden
- Upprepa processen för nästa del av koden och kör också om alla tidigare tester utöver det nya testet

Testerna som skapas ligger främst på komponenttestnivå och är kodfokuserade, men man kan också skriva tester på integrations- eller systemtestnivå. Testdriven utveckling blev populärt tillsammans med konceptet extrem programmering (XP) [Beck02], men det används även inom andra agila modeller och ibland också i sekventiella livscykler. Den här tekniken hjälper utvecklaren att fokusera på ett tydligt definierat förväntat resultat. Testerna är automatiserade och används vid kontinuerlig integration.

Acceptanstestdriven utveckling
Acceptanstestdriven utveckling (ATDD) [Adzic09] innebär att acceptanskriterier och tester definieras i samband med att användarberättelser skrivs (se avsnitt 1.2.2). Acceptanstestdriven utveckling bygger på samarbete och ger alla intressenter en god förståelse för hur programvarukomponenten ska fungera och för vad utvecklare, testare och intressenter behöver göra för att se till att komponenten fungerar korrekt. Processen för acceptanstestdriven utveckling beskrivs i avsnitt 3.3.2.

Vid acceptanstestdriven utveckling skapas återanvändbara tester för regressionstestning. Specifika verktyg används som stöd för att skapa och exekvera sådana tester, ofta inom ramarna för den kontinuerliga integrationsprocessen. De här verktygen kan anslutas till data- och tjänstelager i applikationen vilket gör att testerna kan exekveras på system- eller acceptanstestnivå. Acceptanstestdriven utveckling gör att man snabbt kan åtgärda defekter, validera funktionsbeteende och avgöra om acceptanskriterierna för funktionen har uppfyllts.

Beteendedriven utveckling
Beteendedriven utveckling (BDD) [Chelimsky10] gör att utvecklaren kan fokusera på att testa koden med utgångspunkt från programvarans förväntade beteende. Eftersom testerna baseras på programvarans förväntade beteende är de oftast lätta att förstå för andra teammedlemmar och intressenter.

Specifika ramverk för beteendedriven utveckling (BDD) kan användas för att definiera acceptanskriterier baserat på formatet givet-när-så (given/when/then):

Givet en inledande kontext,
när en händelse inträffar,
så ska specifika utfall säkerställas.

Utifrån de kraven genereras kod inom ramverket för beteendedriven utveckling (BDD) som utvecklarna kan använda för att skapa testfall. Beteendedriven utveckling (BDD) gör det lättare för utvecklaren att samarbeta med andra intressenter, inklusive testare, för att definiera exakta komponenttester med fokus på verksamhetsbehov.

3.1.2 Testpyramiden

Ett programvarusystem kan testas på olika nivåer. De vanligaste testnivåerna som används är (nedifrån och upp): komponent, integration, system och acceptans (se [ISTQB_FL_SYL], avsnitt 2.2). Testpyramiden betonar att det bör finnas ett stort antal tester på de lägre nivåerna (längst ned i pyramiden) och därefter allt färre tester allteftersom utvecklingsarbetet fortskrider (högre upp i pyramiden). Tester på komponent- och integrationstestnivå är vanligtvis automatiserade och skapas med API-baserade verktyg. På system- och acceptanstestnivå skapas automatiserade tester med hjälp av användargränssnittsbaserade verktyg. Konceptet för testpyramiden bygger på testprincipen om kvalitetssäkring och testning i ett tidigt skede (så att defekter kan elimineras så tidigt som möjligt i livscykeln).

3.1.3 Testkvadranter, testnivåer och testtyper

Testkvadranter, som definierats av Brian Marick [Crispin08], samordnar testnivåerna med lämpliga testtyper inom den agila modellen. Testkvadrantmodellen och dess varianter hjälper till att säkerställa att alla viktiga testtyper och testnivåer ingår i utvecklingslivscykeln. Modellen ger också ett sätt att särskilja och beskriva testtyperna för alla intressenter, inklusive utvecklare, testare och verksamhetsrepresentanter.

I testkvadrantmodellen kan tester vara kundorienterade (inriktade på användaren) eller teknikorienterade (inriktade på utvecklaren). Vissa tester understödjer det agila teamets arbete och bekräftar programvarans beteende. Andra tester kan verifiera produkten. Tester kan vara helt manuella, helt automatiska, en kombination av manuella och automatiska eller manuella men med stöd av verktyg. De fyra kvadranterna är som följer:

- Kvadrant Q1 är på komponenttestnivå, teknikorienterade och stödjer utvecklarna. Den här kvadranten innehåller komponenttester. Testerna bör vara automatiserade och ingå i den kontinuerliga integrationsprocessen.
- Kvadrant Q2 är på systemtestnivå, kundorienterade och bekräftar produktens beteende. Den här kvadranten innehåller funktionstester, exempel, test av användarberättelser, prototyper för användarupplevelser och simuleringar. Testerna kontrollerar acceptanskriterierna och kan vara manuella eller automatiska. De skapas ofta medan användarberättelserna utvecklas och förbättrar därmed kvaliteten i berättelserna. De är användbara när automatiserade regressionstestsviter skapas.
- Kvadrant Q3 är på system- eller användaracceptanstestnivå och kundorienterade, och innehåller tester som utvärderar produkten med realistiska scenarier och data. Den här kvadranten innehåller utforskande testning, scenarier, processflöden, användbarhetstestning, användaracceptanstestning, alfatestning och betatestning. Testerna är användarorienterade och ofta manuella.
- Kvadrant Q4 är på system- eller driftacceptanstestnivå och teknikorienterade. Den innehåller tester som utvärderar produkten. Den här kvadranten innehåller prestanda-, last- och stresstester, skalbarhetstester, säkerhetstester, underhållbarhets- och minneshanteringstester, testning av kompatibilitet, interoperabilitet, datamigration och infrastruktur samt återhämtningstestning. Testerna är ofta automatiserade.

Under varje given iteration kan tester från någon eller alla kvadranter behövas. Testkvadranterna gäller dynamisk testning snarare än statisk testning.

3.1.4 Testarens roll

I den här kursplanen har det talats generellt om agila metoder och tekniker och testarens roll inom olika agila livscykler. I det här underavsnittet tittar vi närmare på testarens roll i ett projekt som följer en Scrum-livscykel [Aalst13].

Teamwork
Teamwork är en grundläggande princip inom agil utveckling. Den agila metoden lägger vikt vid ett helt team som består av utvecklare, testare och verksamhetsrepresentanter som samarbetar. Här är de bästa arbetssätten för teamorganisation och beteende i Scrum-team:

- Tvärfunktionellt: Varje teammedlem bidrar med sin unika kompetenssammansättning till teamet. Teamet samarbetar kring teststrategi, testplanering, testspecificering, testexekvering, testutvärdering och rapportering av testresultat.
- Självorganiserande: Teamet kan bestå av bara utvecklare, men enligt vad som noteras i avsnitt 2.1.5 bör det helst ingå minst en testare.
- Samlokalisering: Testarna sitter tillsammans med utvecklarna och produktägaren.
- Samarbetsinriktat: Testarna samarbetar med teammedlemmarna, andra team, intressenterna, produktägaren och Scrum Master.
- Delaktigt: Tekniska beslut om design och testning fattas av teamet som helhet (utvecklare, testare och Scrum Master), i samarbete med produktägaren och andra team om det behövs.
- Engagerat: Testaren engagerar sig i att ifrågasätta och utvärdera produktens beteende och egenskaper med hänsyn till kundernas och användarnas förväntningar och behov.
- Transparent: Utvecklingens och testningens förlopp visas på aktivitetstavlan (se avsnitt 2.2.1).
- Trovärdigt: Testaren måste se till att teststrategin och dess implementation och exekvering är trovärdiga. Annars litar inte intressenterna på testresultaten. Detta sker ofta genom att intressenterna informeras om testprocessen.
- Öppet för återkoppling: Återkoppling är en viktig del av att lyckas i alla projekt, i synnerhet i agila projekt. Med retrospektiv kan teamet lära sig av både framgångar och misslyckanden.
- Följsamt: Testningen måste kunna reagera på förändringar, liksom alla andra aktiviteter i agila projekt.

Med dessa metoder maximeras sannolikheten för lyckad testning i Scrum-projekt.

Sprint zero
Sprint zero är den första iterationen i projektet där flera förberedande aktiviteter äger rum (se avsnitt 1.2.5). Under denna iteration samarbetar testaren med teamet för att utföra följande uppgifter:

- Identifiera projektets omfattning (dvs. produktbackloggen)
- Skapa en inledande systemarkitektur och prototyper på hög nivå
- Planera, införskaffa och installera de verktyg som behövs (t.ex. för testhantering, felhantering, testautomatisering och kontinuerlig integration)
- Skapa en inledande teststrategi för alla testnivåer med beaktande av (bland annat) testningens omfattning, tekniska risker, testtyper (se avsnitt 3.1.3) och mål för täckningen
- Utföra en inledande kvalitetsriskanalys (se avsnitt 3.2.1)
- Definiera testmätetal för att mäta testprocessen, testprogressen inom projektet och produktkvaliteten
- Specificera Definition of Done
- Skapa en aktivitetstavla (se avsnitt 2.2.1)
- Definiera i vilka situationer som testning ska fortsätta respektive avbrytas innan systemet levereras till kunden

Under sprint zero anges inriktningen för vad testningen ska uppnå, och på vilket sätt testningen ska utföras under sprintarna för att åstadkomma detta.

Integration
I agila projekt är målet att regelbundet (företrädesvis efter varje sprint) leverera mervärde för kunden. För att detta ska bli möjligt måste både design och testning beaktas i integrationsstrategin. För att möjliggöra en strategi för kontinuerlig testning av de funktioner och egenskaper som ska levereras är det viktigt att identifiera alla beroenden mellan bakomliggande funktioner.

Testplanering
Eftersom testning är en helt integrerad aktivitet i det agila teamet bör testplaneringen inledas under leveransplaneringen och uppdateras under varje sprint. Vid testplanering för såväl leverans som för varje sprint ska de frågor som diskuteras i avsnitt 1.2.5 beaktas.

Sprintplaneringen leder fram till en uppsättning uppgifter som kan placeras på aktivitetstavlan. Varje uppgifts längd bör vara en eller två arbetsdagar. Eventuella testproblem bör spåras så att takten i testförloppet hålls uppe.

Arbetssätt för agil testning
Det finns flera användbara arbetssätt för testare i ett Scrum-team. Här är några exempel:

- Partestning: Två teammedlemmar (t.ex. en testare och en utvecklare, två testare eller en testare och produktägaren) sitter tillsammans vid en arbetsstation för att utföra en testuppgift eller någon annan uppgift inom sprinten.
- Inkrementell testdesign: Testfall och -charters byggs gradvis upp utifrån användarberättelser och andra testbaser, först i form av enkla tester och sedan med ökande komplexitet.
- Mindmapping: Mindmapping är ett användbart arbetssätt vid testning [Crispin08]. Testare kan till exempel använda mindmapping för att identifiera vilka testsessioner som ska utföras, för att visa teststrategier och för att beskriva testdata.

De här arbetssätten ska användas utöver de övriga arbetssätt som tas upp i den här kursplanen och i kapitel 4 i kursplanen för grundnivå [ISTQB_FL_SYL].

3.2 Utvärdering av kvalitetsrisker och testuppskattning

Ett vanligt mål för testning inom alla typer av projekt (såväl agila som traditionella) är att minska risken för produktkvalitetsproblem till en acceptabel nivå före leveransen. Testare i agila projekt kan använda samma metoder som används i traditionella projekt för att identifiera kvalitetsrisker (eller produktrisker), utvärdera risknivån, uppskatta hur mycket arbete som krävs för att minska risken tillräckligt mycket och därefter mildra riskerna genom att designa, implementera och exekvera tester. Eftersom agila projekt kännetecknas av korta iterationer och snabba förändringar kan de här metoderna dock behöva anpassas.

3.2.1 Utvärdering av kvalitetsrisker i agila projekt

En av många utmaningar för testare är att välja, allokera och prioritera rätt bland testvillkoren. Bland annat måste testaren utreda hur mycket arbete som behöver allokeras för att varje villkor ska kunna testas och därefter placera de planerade testerna i en ordningsföljd som ger ett så effektivt testarbete som möjligt. Testare i agila team kan använda strategier för att identifiera, analysera och motverka risker för att komma fram till ett acceptabelt antal testfall som ska exekveras. Om det finns många interagerande begränsningar och variabler kan det dock krävas en del kompromisser.

En risk är lika med möjligheten för negativa eller oönskade utfall eller händelser. Risknivån fastställs genom att man utvärderar sannolikheten för att den händelse som risken avser inträffar, och vilken konsekvens detta har. Om ett potentiellt problem framför allt påverkar produktkvaliteten kallas det potentiella problemet för en kvalitetsrisk eller produktrisk. Om ett potentiellt problem framför allt påverkar sannolikheten för framgång för projektet kallas det potentiella problemet för en projektrisk eller planeringsrisk [Black07] [vanVeenendaal12].

Inom agila projekt sker kvalitetsriskanalys vid två tillfällen.

- Leveransplanering: verksamhetsrepresentanter som känner till funktionerna i leveransen ger en överblick på hög nivå över riskerna, och hela teamet (inklusive testarna) kan bidra till att identifiera och utvärdera risker.
- Iterationsplanering: hela teamet identifierar och utvärderar kvalitetsrisker.

Här är några exempel på kvalitetsrisker för ett system:

- Felaktiga beräkningar i rapporter (en funktionell risk kopplad till exakthet)
- Långsam respons på användarindata (en icke-funktionell risk kopplad till effektivitet och responstid)

- Svårbegripliga skärmbilder och fält (en icke-funktionell risk kopplad till användbarhet och förståelighet)

Som nämnts tidigare inleds en iteration med iterationsplanering, vars resultat är att uppskattade aktiviteter placeras på en aktivitetstavla. Uppgifterna kan prioriteras delvis utifrån vilken kvalitetsrisknivå de är kopplade till. De uppgifter som är kopplade till större risker bör inledas tidigare och testningen bör vara mer utförlig. De uppgifter som är kopplade till mindre risker kan inledas senare och testas i mindre omfattning.

Här beskrivs ett exempel på hur processen för kvalitetsriskanalys i ett agilt projekt kan gå till under iterationsplaneringen:

1. Samla alla medlemmar i det agila teamet, inklusive testaren/testarna.
2. Skapa en lista med alla backloggobjekt för den aktuella iterationen (t.ex. på en aktivitetstavla).
3. Identifiera de kvalitetsrisker som är associerade med varje objekt, med beaktande av alla relevanta kvalitetsegenskaper.
4. Utvärdera varje identifierad risk. Detta innefattar två uppgifter: att kategorisera risken, och att avgöra risknivån utifrån konsekvens och sannolikhet för defekter.
5. Fastställ hur omfattande testningen ska vara (proportionellt i relation till risknivån).
6. Välj en eller flera lämpliga testmetoder för att motverka varje risk utifrån risknivån och den relevanta kvalitetsegenskapen.

Därefter designar, implementerar och exekverar testaren tester för att mildra riskerna. Testerna ska innefatta samtliga funktioner, beteenden, kvalitetsegenskaper och attribut som påverkar nöjdheten bland kunder, användare och intressenter.

Under hela projektet ska hela teamet vara medvetna om att ytterligare information när som helst kan förändra riskbilden och/eller risknivån för de kända kvalitetsriskerna. Därför bör kvalitetsriskanalysen justeras regelbundet, något som kan leda till att även testerna justeras. Det kan till exempel bli aktuellt att lägga till nya risker, att omvärdera nivån för befintliga risker och att utvärdera effekten av de befintliga åtgärderna för att motverka risker.

Kvalitetsrisker kan också motverkas innan testexekveringen inleds. Om man till exempel upptäcker problem med användarberättelser under riskidentifieringen kan projektteamet göra en grundlig granskning av användarberättelserna för att mildra dessa problem.

3.2.2 Uppskatta mängden testarbete utifrån innehåll och risk

Under leveransplaneringen uppskattar det agila teamet hur mycket arbete som krävs för att slutföra leveransen. Denna uppskattning innefattar också mängden testarbete. En vanlig, samsynsbaserad uppskattningsteknik som används inom agila projekt är ”planning poker”. Produktägaren eller kunden läser en användarberättelse för dem som ska uppskatta arbetsmängden. Varje teammedlem har en kortlek med kort vars värden motsvarar t.ex. Fibonaccis talföljd (dvs. 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 …) eller någon annan sekvens (t.ex. klädstorlekar från XS till XXL). Värdena motsvarar antalet poäng, arbetsdagar eller andra enheter som används för uppskattningen. Fibonaccis talföljd rekommenderas eftersom talen i denna sekvens åskådliggör att osäkerheten ökar proportionellt med användarberättelsens storlek. Ett högt värde vid uppskattningen innebär vanligtvis att användarberättelsen är svår att förstå eller att den bör delas upp i flera mindre berättelser.
Teammedlemmar diskuterar funktionen och ställer frågor till produktägaren om det behövs. Vid uppskattningen beaktas olika aspekter, som mängden utvecklings- och testarbete, användarberättelsens komplexitet och testningens omfattning. Därför bör alltid risknivån för backloggobjektet inkluderas, utöver den prioritet som produktägaren anger, innan ”planning poker”-sessionen inleds. När funktionen har diskuterats väljer varje teammedlem individuellt (och dolt) ett kort som motsvarar hans eller hennes uppskattning. Därefter läggs alla kort fram samtidigt. Om alla teammedlemmar har valt samma värde används detta värde som uppskattning. Om teammedlemmar har valt olika värden diskuterar de kring skillnaderna, och därefter upprepas kortläggningsproceduren tills man har nått ett beslut, antingen genom kompromisser eller genom att man använder regler (t.ex. att använda medianvärdet eller det högsta värdet) för att begränsa proceduren. Dessa diskussioner leder fram till en rimlig och realistisk uppskattning av mängden arbete som krävs för att slutföra alla objekt i produktbackloggen enligt produktägarens önskemål, och förbättrar också hela teamets förståelse för det arbete som ska utföras [Cohn04].

3.3 Tekniker i agila projekt

Många av de testmetoder och testnivåer som används i traditionella projekt kan även användas i agila projekt. I agila projekt kan det dock behövas en del anpassning av testtekniker, terminologi och dokumentation.

3.3.1 Acceptanskriterier, tillräcklig täckningsgrad och annan information för testning

I agila projekt presenteras de inledande kraven som användarberättelser i en prioriterad backlogg i början av projektet. Dessa inledande krav är ofta kortfattade och anges vanligtvis i ett fördefinierat format (se avsnitt 1.2.2). Icke-funktionella krav, som användbarhet och prestanda, är också viktiga och kan presenteras i form av unika användarberättelser eller kopplas till andra, funktionella användarberättelser. Icke-funktionella krav kan anges i ett fördefinierat format eller enligt en standard, till exempel [ISO25000] eller en branschspecifik standard.

Användarberättelserna är en viktig testbas. Här är några andra tänkbara testbaser:

- Erfarenheter från tidigare projekt
- Befintliga funktioner, features och kvalitetsegenskaper i systemet
- Kod, arkitektur och design
- Användarprofiler (kontext, systemkonfigurationer och användarbeteende)
- Information om defekter från befintliga och tidigare projekt
- Kategoriserade defekter i en defekttaxonomi
- Tillämpliga standarder (t.ex. [DO-178B] för programvara för flygelektronik)
- Kvalitetsrisker (se avsnitt 3.2.1)

Under varje iteration skapar utvecklarna kod för att åstadkomma de funktioner och features som beskrivs i användarberättelserna med alla relevanta kvalitetsegenskaper. Därefter verifieras och valideras koden genom acceptanstestning. För att acceptanskriterier ska vara testbara ska de innefatta följande faktorer (där det är relevant) [Wiegers13]:

- Funktionellt beteende: Det externt observerbara beteendet när användaråtgärder används som indata inom bestämda konfigurationer.
- Kvalitetsegenskaper: Hur systemet utför ett angivet beteende. Egenskaperna kan också kallas för kvalitetsattribut eller icke-funktionella krav. Några vanliga exempel på kvalitetsegenskaper är prestanda, tillförlitlighet, användbarhet osv.
- Scenarier (användningsfall): En sekvens av åtgärder som utförs mellan en extern aktör (vanligtvis en användare) och systemet för att åstadkomma ett angivet mål eller en verksamhetsaktivitet.
- Affärsregler: Uppgifter som endast kan utföras i systemet under vissa villkor som definieras av externa procedurer och begränsningar (t.ex. de procedurer som ett försäkringsbolag använder för att hantera anspråk).
- Externa gränssnitt: Beskrivningar av kopplingarna mellan systemet som ska utvecklas och de omgivningar där det ska användas. Externa gränssnitt kan delas in i olika undertyper (användargränssnitt, gränssnitt mot andra system osv).
- Begränsningar: Design- och implementeringsbegränsningar som begränsar utvecklarens möjligheter. På enheter med inbyggd programvara måste man till exempel ofta beakta fysiska begränsningar som storlek, vikt och gränssnittsfaktorer.
- Datadefinitioner: Kunden kan beskriva format, datatyp, tillåtna värden och standardvärden för ett dataobjekt i sammansättningen av en komplex affärsdatastruktur (till exempel postnumret i en postadress).

Utöver användarberättelserna och tillhörande acceptanskriterier finns det även annan information som är relevant för testaren, bland annat:

- Hur systemet är tänkt att fungera och användas
- Vilka systemgränssnitt som kan användas för att testa systemet
- Om det aktuella verktygsstödet är tillräckligt
- Om testaren har tillräckliga kunskaper och färdigheter för att utföra de nödvändiga testerna

Det är vanligt att testare upptäcker att ytterligare information behövs (t.ex. om kodtäckning) under iterationerna, och de bör samarbeta nära med övriga medlemmar i det agila teamet för att inhämta all nödvändig information. Relevant information är viktigt för att det ska gå att bedöma om en viss uppgift

kan anses vara slutförd. Konceptet ”Definition of Done” är avgörande för alla agila projekt och det gäller inom flera olika områden, vilket beskrivs i följande del- och underavsnitt.

Testnivåer
En ”Definition of Done” anges för varje testnivå. I listan nedan finns olika exempel på definitioner som kan användas på olika testnivåer.

- Komponenttest
  - 100 % beslutstäckning där så är möjligt, med noggranna granskningar av eventuella onåbara vägar
  - Statisk analys av all kod har utförts
  - Inga olösta större defekter (rangordning baserat på prioritet och allvarlighet)
  - Ingen känd, oacceptabel teknisk skuld återstår i design eller kod [Jones11]
  - All kod, alla komponenttester och deras resultat har granskats
  - Alla komponenttester har automatiserats
  - Alla viktiga egenskaper är inom de överenskomna gränsvärdena (t.ex. prestanda)
- Integrationstest
  - Alla funktionella krav har testats, inklusive både positiva och negativa tester, med ett lämpligt antal tester utifrån storlek, komplexitet och risker
  - Alla gränssnitt mellan komponenter har testats
  - Alla kvalitetsrisker har testats i enlighet med den definierade testomfattningen
  - Inga olösta större defekter (prioriterade efter risk och vikt)
  - Alla upptäckta defekter har rapporterats
  - Alla regressionstester har automatiserats så långt det är möjligt, och alla automatiserade tester har lagrats på en gemensam plats
- Systemtest
  - ”End-to-end”-testning av användarberättelser, funktioner och features
  - Alla användaridentiteter har testats
  - De viktigaste kvalitetsegenskaperna i systemet har testats (t.ex. prestanda, robusthet, tillförlitlighet)
  - Testning har utförts i en eller flera produktionsliknande miljö(er) av all hård- och programvara för alla konfigurationer som stöds, så långt det är möjligt
  - Alla kvalitetsrisker har testats i enlighet med den definierade testomfattningen
  - Alla regressionstester har automatiserats så långt det är möjligt, och alla automatiserade tester har lagrats på en gemensam plats
  - Alla upptäckta defekter har rapporterats och troligen åtgärdats
  - Inga olösta större defekter (prioriterade efter risk och viktighetsgrad)

Användarberättelse
”Definition of Done” för användarberättelser kan fastställas utifrån följande kriterier:

- De användarberättelser som valts ut för iterationen är fullständiga, hela teamet förstår dem och de innehåller detaljerade, testbara acceptanskriterier
- Alla delar av användarberättelsen har specificerats och granskats, och acceptanstesterna för användarberättelsen har slutförts
- De uppgifter som måste utföras för att implementera och testa de utvalda användarberättelserna har identifierats och arbetsmängden har uppskattats av teamet

Feature
”Definition of Done” för features, som kan omfatta flera användarberättelser eller epics, kan innefatta följande:

- Alla berörda användarberättelser, inklusive acceptanskriterier, har definierats och godkänts av kunden
- Designen är färdig, utan känd teknisk skuld
- Koden är färdig, utan känd teknisk skuld eller oavslutad omstrukturering
- Komponenttester har utförts och den önskade täckningsgraden har uppnåtts
- Integrationstester och systemtester för funktionen har utförts i enlighet med definierade täckningsgradkriterier
- Inga större defekter återstår att åtgärda

- Featuredokumentationen är fullständig (kan innefatta leveransdokument, användarmanualer och onlinehjälp)

Iteration
”Definition of Done” för en iteration kan fastställas utifrån följande kriterier:
- Alla features för iterationen är klara och har testats var för sig i enlighet med nivåkriterierna
- Alla eventuella icke-kritiska defekter som inte har kunnat åtgärdas inom iterationen har lagts till i produktbackloggen och prioriterats
- Integrationen av alla features i iterationen har slutförts och testats
- Dokumentationen har skrivits, granskats och godkänts

I det här läget kan programvaran eventuellt vara klar för leverans eftersom iterationen har slutförts, men vissa iterationer leder inte fram till en leverans.

Leverans
”Definition of Done” för en leverans (som kan innefatta flera iterationer) kan innefatta följande:
- Täckningsgrad: Alla relevanta testbaselement för allt innehåll i leveransen har testats. Täckningsgraden bedöms utifrån vad som är nytt och vad som har förändrats, komplexitet och storlek och de associerade riskerna för felsymptom.
- Kvalitet: Defektintensiteten (t.ex. hur många defekter som upptäcks per dag eller per transaktion), defekttätheten (t.ex. antalet defekter som upptäcks i relation till antalet användarberättelser, arbetsinsats och/eller kvalitetsattribut) och det uppskattade antalet återstående defekter är inom acceptabla gränser, följderna av ej åtgärdade och återstående defekter (med beaktande av deras allvarlighet och prioritet) har definierats och befunnits acceptabla, och den återstående risknivån för varje identifierad kvalitetsrisk har definierats och befunnits acceptabel.
- Tid: Om det fördefinierade leveransdatumet inte har kunnat hållas måste man beakta de affärsmässiga följderna av att genomföra leveransen jämfört med att inte leverera.
- Kostnad: Den uppskattade livscykelkostnaden bör användas för att beräkna avkastningen för det levererade systemet (de beräknade utvecklings- och underhållskostnaderna bör alltså vara betydligt lägre än den förväntade totala försäljningen av produkten). Huvuddelen av livscykelkostnaden härrör ofta från underhåll efter att produkten har levererats, eftersom ett antal defekter vanligtvis följer med till produktionsfasen.

3.3.2 Använda acceptanstestdriven utveckling

Acceptanstestdriven utveckling är ett arbetssätt som utgår från testningen (”test-first”). Testfall skapas innan en användarberättelse implementeras. Testfallen skapas genom ett samarbete mellan utvecklare, testare och verksamhetsrepresentanter inom det agila teamet [Adzic09] och de kan vara manuella eller automatiserade. Det första steget är en workshop för att skapa specifikationen, där användarberättelsen analyseras, diskuteras och skrivs av utvecklare, testare och verksamhetsrepresentanter. Alla eventuella ofullständigheter, tvetydigheter eller fel i användarberättelsen åtgärdas under den här processen.

Nästa steg är att skapa testerna. Detta görs antingen av hela teamet eller av testaren. Oavsett hur testerna skapas ska de sedan valideras av en oberoende person, till exempel en verksamhetsrepresentant. Testerna är exempel som beskriver de specifika egenskaperna från användarberättelsen. Exemplen hjälper teamet att implementera berättelsen på rätt sätt. Begreppen ”exempel” och ”test” används ibland som synonymer, eftersom de avser samma sak. Allt arbete inleds med enkla exempel och öppna frågor.

Normalt utförs de positiva testerna först för att bekräfta att programmets beteende är korrekt, utan undantags- eller feltillstånd som stör den aktivitetssekvens som genomförs om allt går som förväntat. När de positiva testerna har genomförts bör teamet skriva negativa tester som även innefattar icke-funktionella attribut (t.ex. prestanda och användbarhet). Testerna ska formuleras så att alla intressenter förstår dem. Språket ska vara naturligt och lättbegripligt och testet ska innefatta alla eventuella nödvändiga förutsättningar, indata och relaterade utdata.

Exemplen måste innefatta alla egenskaper från användarberättelsen och får inte innehålla något som inte finns i berättelsen. Det innebär att det inte får finnas exempel som beskriver en aspekt av

användarberättelsen som inte finns dokumenterad i själva berättelsen. Dessutom bör det inte finnas mer än ett exempel som beskriver samma egenskap hos en användarberättelse.

3.3.3 Black-box-testdesign för funktionella och icke-funktionella egenskaper

Vid agil testning skapas många tester av testarna samtidigt som utvecklarna arbetar med programmering. Utvecklarna programmerar med utgångspunkt från användarberättelser och acceptanskriterier, och testarna skapar tester som också baseras på användarberättelser och relaterade acceptanskriterier. (Vissa tester, till exempel utforskande tester och andra typer av erfarenhetsbaserade tester, skapas senare, under exekveringsfasen. Mer information finns i avsnitt 3.3.4.) Testarna kan använda traditionella black-box-testdesigntekniker som ekvivalensklassindelning, gränsvärdesanalys, beslutstabeller och tillståndsbaserad testning för att skapa de här testerna. Gränsvärdesanalys kan till exempel användas för att välja testvärden när en kund endast får välja ett begränsat antal objekt att köpa.

I många fall kan icke-funktionella krav beskrivas i form av användarberättelser, men även black-box-testdesigntekniker (till exempel gränsvärdesanalys) kan användas för att skapa tester för icke-funktionella kvalitetsegenskaper. En användarberättelse kan innehålla krav på prestanda eller tillförlitlighet. En viss exekvering kan till exempel förses med en tidsgräns, eller så kan det finnas en gräns för hur många gånger en uppsättning åtgärder får fallera.

Mer information om black-box-testdesigntekniker finns i kursplanen för grundnivå [ISTQB_FL_SYL] och Advanced Level Test Analyst-kursplanen [ISTQB_ALTA_SYL].

3.3.4 Utforskande testning och agil testning

Utforskande testning är viktigt i agila projekt eftersom det finns begränsad tid för testanalys och användarberättelserna inte är särskilt detaljerade. För att uppnå bästa möjliga resultat bör utforskande testning kombineras med andra erfarenhetsbaserade tekniker i en reaktiv teststrategi. Denna bör i sin tur kombineras med andra teststrategier, som analytisk riskbaserad testning, analytisk kravbaserad testning, modellbaserad testning och regressionshämmande testning. Teststrategier och kombinationer av olika teststrategier diskuteras i kursplanen för grundnivå [ISTQB_FL_SYL].

Vid utforskande testning utförs testdesign och testexekvering samtidigt, med vägledning av en förberedd testcharter. I en testcharter definieras de testvillkor som ska behandlas under en timeboxindelad testsession. Vid utforskande testning används resultaten av ett test som utgångspunkt för nästa test. Samma white-box- och black-box-tekniker kan användas för att designa testerna som vid förhandsdesignad testning.

En testcharter kan innehålla följande information:

- Aktör: den tänkta användaren av systemet
- Syfte: temat för chartern, inklusive vilket specifikt mål som aktören vill uppnå, dvs. testvillkoren
- Förberedelser: allt som behöver göras för att testexekveringen ska kunna startas
- Prioritet: den relativa betydelsen för chartern baserat på prioriteten för den associerade användarberättelsen eller risknivån
- Referens: specifikationer (t.ex. användarberättelsen), risker eller andra informationskällor
- Data: alla data som behövs för att genomföra chartern
- Aktiviteter: en lista med uppgifter som aktören kan förväntas vilja utföra i systemet (t.ex. ”Logga in på systemet som super user”) och med faktorer som kan vara intressanta att testa (både positiva och negativa tester)
- Orakelanteckningar: hur produkten ska utvärderas för att säkerställa att resultatet är korrekt (t.ex. att beskriva vad som händer på skärmen och jämföra det med det som står i användarmanualen)
- Variationer: alternativa åtgärder och utvärderingar som kompletterar det som beskrivs under Aktiviteter

Utforskande testning kan hanteras genom så kallad sessionsbaserad teststyrning. En session definieras som en oavbruten period med testning som kan vara i mellan 60 och 120 minuter. Testsessionerna innefattar följande:

- Genomgångssession (lära sig hur det fungerar)

- Analyssession (utvärdering av funktionaliteten eller egenskaperna)
- Djupgående täckning (ytterligheter, scenarier, interaktioner)

Kvaliteten på testerna är beroende av testarnas förmåga att ställa relevanta frågor om vad som ska testas. Här är några exempel:

- Vad är det viktigaste vi bör ta reda på om systemet?
- På vilka sätt kan fel uppstå i systemet?
- Vad händer om...?
- Vad borde hända om...?
- Har kundens behov, krav och förväntningar uppfyllts?
- Kan systemet installeras (och avinstalleras vid behov) på alla uppgraderingsvägar som stöds?

Under testexekveringen använder testaren sin kreativitet, sin intuition, sitt logiska tänkande och andra färdigheter för att hitta eventuella problem med produkten. Dessutom måste testaren ha god kunskap om och förståelse för programvaran som testas, verksamhetsdomänen, hur programmet ska användas och hur man avgör när systemet inte fungerar som det ska.

Heuristik kan användas vid testningen för att vägleda testaren om hur testningen ska utföras och hur resultaten kan utvärderas [Hendrickson]. Här är några exempel:

- Gränsvärden
- CRUD (Create, Read, Update, Delete - skapa, läs, uppdatera, radera)
- Konfigurationsvariationer
- Avbrott (t.ex. utloggning, avstängning eller omstart)

Det är viktigt att testaren dokumenterar processen i så stor utsträckning som möjligt. Annars blir det svårt att gå tillbaka och se hur ett problem i systemet upptäcktes. Här följer några exempel på information som bör dokumenteras:

- Täckningsgrad: Vilka indata har använts? Hur mycket har testats och hur mycket återstår att testa?
- Utvärderingsanteckningar: Vilka observationer har gjorts under testningen? Var systemet och funktionen stabila under testning? Upptäcktes några defekter? Vad planeras som nästa steg utifrån de aktuella observationerna? Eventuella andra idéer?
- Risk-/strategilista: Vilka risker har testats och vilka av de viktigaste riskerna återstår? Kan den ursprungliga strategin följas eller behöver den ändras?
- Problem, frågor och anomalier: Anteckningar om alla oväntade beteenden, frågor eller funderingar om arbetssättets effektivitet, idéer/testförsök, testmiljön, testdata, eventuella missförstånd kring funktionen, testskriptet eller systemet som testas
- Faktiskt beteende: registrering av de aspekter av systemets faktiska beteende som behöver sparas (t.ex. videoklipp, skärmdumpar, utdatafiler)

Den loggade informationen bör samlas och/eller sammanfattas i någon typ av statushanteringsverktyg (t.ex. testlednings- eller ärendehanteringsverktyg eller aktivitetstavlan) på ett sätt som gör det enkelt för alla intressenter att förstå aktuell status för all testning som har utförts.

3.4 Verktyg i agila projekt

De verktyg som beskrivs i kursplanen för grundnivå [ISTQB_FL_SYL] är relevanta för och kan användas av testare i agila team. Olika verktyg används på olika sätt och vissa verktyg är mer relevanta och användbara i agila projekt än i traditionella projekt. Till exempel kan verktyg för testledning, kravhantering och avvikelsehantering (defektspårning) användas av agila team, men i vissa agila team använder man i stället ett heltäckande verktyg (t.ex. för applikationslivscykelhantering eller uppgiftshantering) som innehåller funktioner som är relevanta för agil utveckling, till exempel aktivitetstavlor, burndown charts och användarberättelser. Konfigurationshanteringsverktyg är viktiga för testare i agila team eftersom ett stort antal automatiserade tester används på alla nivåer vilket innebär att de associerade testobjekten behöver lagras och hanteras.

Utöver de verktyg som beskrivs i kursplanen för grundnivå [ISTQB_FL_SYL] kan testare i agila projekt även använda de verktyg som beskrivs i följande underavsnitt. De här verktygen används av hela teamet

för att säkerställa bästa möjliga samarbete och informationsdelning inom teamet, vilket är avgörande för agila projekt.

3.4.1 Verktyg för ärendehantering och spårning

I vissa fall använder agila team fysiska berättelse- eller aktivitetstavlor (t.ex. en whiteboard eller anslagstavla) för att hantera och spåra användarberättelser, tester och andra uppgifter inom varje iteration. Ett agilt team kan också använda programvara för applikationslivscykelhantering (”application lifecycle management”, ALM-verktyg) och ärendehantering, till exempel elektroniska aktivitetstavlor. De här verktygen kan användas för att göra följande:

- Registrera användarberättelser och relaterade utvecklings- och testuppgifter, för att se till att inget glöms bort under iterationen
- Registrera teammedlemmarnas uppskattningar om de uppgifter de ska utföra och automatiskt beräkna hur mycket arbete som krävs för att implementera en berättelse, som stöd för effektiva iterationsplaneringssessioner
- Koppla utvecklings- och testuppgifter till en användarberättelse för att skapa en komplett bild av hur mycket arbete det krävs av teamet för att implementera berättelsen
- Samla utvecklarnas och testarnas uppdateringar av uppgiftsstatus allt eftersom de utför sitt arbete, vilket automatiskt ger en aktuell beräkning av status för varje berättelse, iterationen och den övergripande leveransen
- Skapa en visuell återgivning (genom mätetal, diagram och dashboards) av aktuell status för varje användarberättelse, iterationen och leveransen så att alla intressenter (även teammedlemmar som befinner sig långt bort) snabbt kan få överblick över projektet
- Integration med konfigurationshanteringsverktyg vilket kan möjliggöra automatiserad registrering av kodincheckning och byggen i relation till uppgifter och (i vissa fall) automatiserade statusuppdateringar för uppgifter

3.4.2 Verktyg för kommunikation och informationsdelning

Utöver e-post, dokument och muntlig kommunikation använder agila team ofta tre ytterligare verktygstyper för att kommunicera och dela information: wiki-sidor, snabbmeddelanden och skrivbordsdelning.

Teamet kan använda wiki-sidor för att skapa och dela en kunskapsbas online om olika aspekter av projektet, till exempel:

- Diagram över produktfunktioner, funktionsdiskussioner, prototypdiagram, foton av whiteboard-diskussioner och annan information
- Verktyg och/eller tekniker för utveckling och testning som teammedlemmar vill rekommendera till övriga teamet
- Mätetal, diagram och dashboards om produktstatus. Detta är särskilt användbart om wiki-sidan integreras med andra verktyg, till exempel byggservern och verktyget för aktivitetshantering, eftersom produktstatus kan uppdateras automatiskt
- Samtal mellan teammedlemmar, vilket liknar snabbmeddelanden och e-post men kan delas med alla övriga teammedlemmar

Snabbmeddelanden, telefonkonferenser och videochattverktyg ger följande fördelar:

- Möjliggör kommunikation i realtid inom teamet, vilket är särskilt praktiskt om alla inte jobbar på samma plats
- Gör att geografiskt utspridda team kan delta i dagliga ståmöten
- Sänker telefonkostnaderna genom användning av IP-teknik, vilket förbättrar möjligheterna till kommunikation inom geografiskt utspridda team

Skrivbordsdelning och inspelningsverktyg ger följande fördelar:

- Möjliggör produktdemonstrationer, kodgranskning och till och med partestning inom utspridda team
- Inspelade produktdemonstrationer i slutet av varje iteration kan publiceras på teamets wiki-sida

De här verktygen ska användas som komplement, inte som ersättning, för personliga möten inom agila team.

3.4.3 Verktyg för programvarubyggen och distribution

Som nämnts tidigare i den här kursplanen är dagliga byggen och distributioner av programvara en grundsten inom alla agila projekt. Detta innebär att verktyg för kontinuerlig integration och för distribution av byggen måste användas. Användningsområden, fördelar och risker med dessa verktyg beskrevs i avsnitt 1.2.4.

3.4.4 Konfigurationshanteringsverktyg

I agila team kan konfigurationshanteringsverktyg användas för att lagra kod och automatiserade tester, men dessutom kan även manuella tester och andra testarbetsprodukter lagras på samma plats som projektets källkod. Detta ger spårbarhet så att man kan se vilka versioner av programvaran som har testats med vilka versioner av testerna, och det går att göra snabba förändringar utan att tidigare information går förlorad. De huvudsakliga typerna av versionshanteringssystem är centraliserade källkontrollsystem och distribuerade versionshanteringssystem. Teamets storlek, struktur, placering och behoven av integration med andra verktyg avgör vilket system för versionshantering som passar bäst för ett visst agilt projekt.

3.4.5 Verktyg för design, implementering och exekvering av tester

Vissa verktyg är användbara för agila testare i specifika delar av programtestprocessen. De flesta av de här verktygen är inte nya eller specifika för agila projekt, men de innehåller viktiga funktioner som är användbara för att klara de snabba förändringarna i agila projekt.

- Testdesignverktyg: Användning av exempelvis mindmappingverktyg har blivit allt vanligare för att snabbt designa och definiera tester för en ny funktion.
- Verktyg för testfallshantering: Den typ av verktyg för testfallshantering som används i agila projekt kan vara en del av teamets verktyg för applikationslivscykelhantering (”application lifecycle management”, ALM-verktyg) och ärendehantering.
- Verktyg för att förbereda och generera testdata: Verktyg som genererar data för att fylla databasen i en applikation kan vara mycket användbara om det behövs stora mängder data och olika datakombinationer för att testa applikationen. Sådana verktyg kan också användas för att omdefiniera databasstrukturen när produkten förändras under det agila projektet, och för att omstrukturera skripten som genererar data. På så sätt kan testdata uppdateras snabbt när förändringar sker. I vissa verktyg för att förbereda testdata används produktionsdatakällor som råmaterial och sedan används skript för att ta bort eller avidentifiera känsliga data. Andra verktyg av det här slaget kan användas för att validera stora mängder indata eller utdata.
- Verktyg för att läsa in testdata: När data har genererats för testning måste de läsas in i applikationen. Manuell datainmatning tar ofta lång tid och risken för misstag är stor. I stället kan man använda inläsningsverktyg som gör processen säkrare och mer effektiv. Många datagenereringsverktyg har en inbyggd komponent för datainläsning. Det kan också finnas möjlighet att göra massinläsningar av data via databashanteringssystemet.
- Verktyg för automatiserad testexekvering: Vissa testexekveringsverktyg passar bättre för agil testning än andra. Flera olika verktyg är tillgängliga, både kommersiella och open source-verktyg, som stöd för ”test first”-arbetssätt som beteendedriven utveckling (BDD), testdriven utveckling och acceptanstestdriven utveckling. Med hjälp av de här verktygen kan testare och verksamhetsrepresentanter uttrycka det förväntade systembeteendet i tabeller eller i fritext med nyckelord.
- Verktyg för utforskande testning: Verktyg som registrerar och loggar aktiviteter som utförs i en applikation under utforskande testning kan vara användbara för både testare och utvecklare, eftersom alla åtgärder registreras. När en defekt upptäcks kan testaren alltså gå tillbaka och undersöka vilka åtgärder som vidtogs precis innan felet uppstod, så att defekten kan rapporteras på rätt sätt till utvecklarna. Den loggning som sker under utforskande testning kan vara värdefull om testet så småningom inkluderas i uppsättningen med automatiserade regressionstester.

3.4.6 Verktyg för molntjänster och virtualisering

Genom virtualisering kan en enda fysisk resurs (server) fungera som flera separata, mindre resurser. Genom att använda virtuella datorer eller molninstanser som flera olika servrar får teamet bättre möjligheter till utveckling och test, samtidigt som man slipper fördröjningar och väntan på fysiska servrar. I de flesta virtualiseringsverktyg finns en inbyggd funktion för ögonblicksbilder (snapshots), som gör att det går snabbt att etablera en ny server eller återställa en server. I vissa teststyrningsverktyg används virtualiseringsteknik för att skapa en ögonblicksbild av servern vid den tidpunkt då ett fel upptäcks, så att testare kan visa bilden för de utvecklare som undersöker felet.

Referenser

Standarder

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.
- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

Dokument från ISTQB

- [ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, version 2012
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, version 2012
- [ISTQB_FA_OVIEW] ISTQBs översiktsdokument för agil testning på grundnivå, version 1.0
- [ISTQB_FL_SYL] ISTQBs kursplan för grundnivå, version 2011

Böcker

[Aalst13] Leo van der Aalst and Cecile Davis, “TMap NEXT® in Scrum,” ICT-Books.com, 2013.

[Adzic09] Gojko Adzic, “Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing,” Neuri Limited, 2009.

[Anderson13] David Anderson, “Kanban: Successful Evolutionary Change for Your Technology Business,” Blue Hole Press, 2010.

[Beck02] Kent Beck, “Test-driven Development: By Example,” Addison-Wesley Professional, 2002.

[Beck04] Kent Beck and Cynthia Andres, “Extreme Programming Explained: Embrace Change, 2e” Addison-Wesley Professional, 2004.

[Black07] Rex Black, “Pragmatic Software Testing,” John Wiley and Sons, 2007.

[Black09] Rex Black, “Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e,” Wiley, 2009.

[Chelimsky10] David Chelimsky et al, “The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends,” Pragmatic Bookshelf, 2010.

[Cohn04] Mike Cohn, “User Stories Applied: For Agile Software Development,” Addison-Wesley Professional, 2004.

[Crispin08] Lisa Crispin and Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams,” Addison-Wesley Professional, 2008.

[Goucher09] Adam Goucher and Tim Reilly, editors, “Beautiful Testing: Leading Professionals Reveal How They Improve Software,” O'Reilly Media, 2009.

[Jeffries00] Ron Jeffries, Ann Anderson, and Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.

[Jones11] Capers Jones and Olivier Bonsignour, “The Economics of Software Quality,” Addison-Wesley Professional, 2011.

[Linz14] Tilo Linz, “Testing in Scrum: A Guide for Software Quality Assurance in the Agile World,” Rocky Nook, 2014.

[Schwaber01] Ken Schwaber and Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.

[vanVeenendaal12] Erik van Veenendaal, “The PRISMA approach”, Uitgeverij Tutein Nolthenius, 2012.

[Wiegers13] Karl Wiegers and Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013.

Agila termer

De nyckelord som finns i ISTQBs ordlista listas i början av varje kapitel. Vad gäller vanliga termer för agila projekt har vi utgått från definitionerna i nedanstående väletablerade webbresurser.

Vi uppmuntrar läsaren att läsa mer om olika agila termer på de här webbplatserna. Länkarna var aktiva vid den tidpunkt då det här dokumentet publicerades.

Övriga referenser

Följande referenser leder till olika typer av onlineinformation. Referenserna kontrollerades vid den tidpunkt då den här kursplanen publicerades, men ISTQB kan inte hållas ansvarigt om någon av referenserna inte längre är tillgänglig.

- [Agile Alliance Guide] Flera författare. http://guide.Agilealliance.org/.
- [Agilemanifesto] Flera författare. www.agilemanifesto.org.

Index

acceptanskriterier, 12, 19, 20, 22, 23, 24, 26, 27, 28, 32, 33, 35
acceptanstestdriven utveckling, 27, 34
acceptanstester, 9, 14, 15, 20, 23, 33
agil aktivitetstavla, 21
agil programvaruutveckling, 7, 10
agila aktivitetstavlor, 22
agila manifestet, 7, 8
angreppssätt för test, 14, 26
anpassning till förändring, 8
användarberättelse (user story), 7, 12, 13, 14, 18, 19, 20, 22, 23, 24, 27, 28, 31, 32, 33, 34, 35, 36, 37
användbarhetstestning, 28
automatiserad testexekvering, 26
automatiserade tester, 7, 9, 19, 21, 22, 23
berättelsekort, 12, 22
beteendedriven utveckling, 27, 28
burndown charts, 21, 22, 36
code churn, 22
dagligt ståmöte, 9, 11, 21, 22, 37
datagenereringsverktyg, 38
defekttaxonomi, 32
epic, 19, 33
extrem programmering, 10, 18, 27
fungerande programvara, 8
givet-när-så, 27
hastighet, 15, 22
hela teamets ansvar, 8, 9, 10
individer och interaktioner, 8
inkrement, 10
integration, 29
interoperabilitet, 12, 28
intressenter, 9, 18, 20, 21, 24, 27, 28, 31, 34, 36, 37
INVEST, 12
iteration, 9, 10, 12, 13, 14, 21, 37
iterationsplanering, 14, 15, 18, 20, 22, 24, 30, 31, 37
Kanban, 11
Kanbantavla, 11
konceptet 3C, 12
konfigurationselement, 17
konfigurationshantering, 17, 22, 23, 37, 38
kontinuerlig integration, 9, 10, 13, 20, 23, 27, 28, 29, 38
kundsamarbete, 8
kvalitetsrisk, 15, 19, 26, 31
kvalitetsriskanalys, 29, 30
leveransplanering, 12, 14, 15, 18, 23, 30
livscykel för programvara, 7
modell för inkrementell utveckling, 7
modell för iterativ utveckling, 7
partestning, 18, 30, 37
Power of Three, 9
prestandatestning, 26
processförbättring, 13, 22
produktbacklogg, 11, 12, 14, 15, 29, 31, 34
produktrisk, 26, 31
produktägare, 11
projektarbetsprodukter, 19
ramverk för komponenttest, 26
regressionstestning, 13, 19, 20, 23, 26, 27
retrospektivmöten, 13
rotorsaksanalys, 13
Scrum, 10, 20, 29
Scrum Master, 11, 29
sprint, 10, 12, 29
sprint zero, 29
sprintbacklogg, 11
säkerhetstestning, 28
teamwork, 29
teknisk skuld, 18, 22
testautomatisering, 23, 24, 29
testbas, 7, 15, 32
testcharter, 26, 35
testdriven utveckling, 7, 10, 19, 26, 27
testkvadranter, 28
testkvadrantmodellen, 28
testorakel, 7, 15
testplanering, 30
testpyramiden, 28
teststrategi, 24, 26, 29
testuppskattning, 26
timeboxing, 11, 12
tolv principer, 8
transparens, 11
utforskande testning, 19, 26, 28, 35, 36
verifiering av bygge, 17, 23
verktyg för att förbereda testdata, 38
versionshantering, 38
XP. *Se* extrem programmering