Modellbaserad testning

Ville-Markus Yli-Suutala

Innehåll

1 Inledning

Då det hela tiden skrivs mera program och allt fler uppgifter i samhället sköts av program, blir det allt viktigare med effektiva sätt att se till att dessa program uppfyller sina krav. Manuell testning av program är en arbetsintensiv kostsam process. Därför är det viktigt att utveckla nya automatiserade sätt att testa program.

2 Testning

Testning av programvara är en aktivitet där man kör program och jämför resultatet av körningen mot något förväntat resultat. Programmet som testas kallas för testsystemet (system under test, SUT) i den här avhandlingen. Målet med testning är att hitta fel i programmet, eller att bekräfta att programmet fyller de krav som är bestämda för programmet. Kraven behöver inte vara formellt dokumenterade för att det ska gå att testa ett program, men de behöver vara kända.

Testning kan utföras manuellt eller automatiskt. Vid manuell testning försöker testare hitta fel i testsystemet genom att använda det för hand som en vanlig användare. Manuell testning fungerar bra för att hitta fel, men är arbetsintensivt. Automatisk testning innebär användning av program för att automatisera delar av testningsprocessen.

Att korrigera ett fel kostar mera, desto senare i utvecklingsprocessen det upptäcks [2]. Detta skapar ekonomiska incitament att hitta fel så tidigt som möjligt. Det finns inga exakta regler för hur mycket resurser som borde spenderas på testning i ett programvaruutvecklingsprojekt. Kostnaden för fel i programmet varierar från ingenting till människoliv [6], beroende på programmet och på felet. Rätt mängd resurser att spendera på testning är en balans som måste hittas för varje projekt skillt.

2.1 Testfall

Ett par av input och förväntat resultat kallas för ett testfall. Ett testfall för en funktion som utför multiplikation kan till exempel ha talen 2 och 3 som input, och 6 som förväntat resultat. Eftersom mängden av möjliga tillstånd i alla icke-triviala program är allt för stor för att testas, måste testfallen i praktiken begränsas till någon delmängd av möjliga tillstånd. Om multiplikationsfunktionen använder 32-bitars heltal, så har redan den $2^{32} \times 2^{32}$ möjliga inputvärden.

2.2 Testorakel

Sättet att bedöma om ett test fallerar eller passerar kallas för testorakel.

2.3 Täckningskriterium

Eftersom alla input inte går att testa, behövs någon strategi, för att välja en bra begränsad mängd testfall och för att sätta mål för testningsverksamheten. Täckningskriterium är en sådan strategi.

Täckningskriterium är regler för att placera testkrav på testsystemet [1]. Testkrav är programdelar som ska täckas av testfall. Till exempel kan ett täckningskriterium vara att täcka alla funktioner i testsystemet. Då är funktionerna testkrav. Då ett täckningskriterium är bestämt och testkraven är kända, har man ett mål för testningsverksamheten i att täcka alla testkrav, och en mätare för hur testningsverksamheten fortskrider i andelen av testkrav som är täckta.

2.4 White-box och black-box testning

Testningsmetoder har länge klassats som white-box eller black-box beroende på om källkoden för testsystemet används för att skapa testfall eller inte [1]. Till white-box metoder räknas de metoder som använder källkoden, och metoder som inte använder källkoden klassas som black-box.

I modellbaserad testning genereras testfallen från modellen på testsystemet. Modellbaserad testning är alltså en black-box testningsteknik.

2.5 Verifiering och validering

Målet med verifiering är att avgöra om egenskaperna av ett program uppfyller kraven man har bestämt för programmet. Är systemet byggt rätt och gör det allt som det enligt specifikationen borde göra. Till verifiering hör också att bekräfta att programmet inte gör sådant som det enligt specifikationen inte ska göra, till exempel kraschar.

Validering tar reda på om systemet fungerar för det planerade ändamålet. Är användaren nöjd? Verifiering hittar avvikelser mellan systemet man har

byggt och systemet man försökte bygga. Validering frågar om systemet man byggde och specifierade är sådant som det borde vara. Genom validering kan man alltså hitta fel i specifikationen.

3 Modellbaserad testning

Modellbaserad testning är en testningsteknik där man bygger en abstrakt modell av testsystemet och använder den för att automatiskt generera och köra testfall. Det finns också flera andra typer av modellbaserad testning. Till exempel att härleda testfall för manuell testning från en modell räknas också som en typ av modellbaserad testning, men den här avhandlingen behandlar endast den typen av modellbaserad testning där modellen används för att automatiskt generera körbara testfall med orakel.

Utting, Pretschner och Legeard delar upp den typiska processen för tillämpning av modellbaserad testning till fem olika steg [5]:

1. Skapandet av modellen för testsystemet.
2. Fastställande av kriterium för hur testfallen väljs.
3. Omvandling av kriterium för testfallen till testfallsspecifikationer.
4. Testgenerering.
5. Testkörning.

I det här kapitlet gås stegen igenom och processen för modellbaserad testning förklaras i mera detalj.

3.1 Modeller

En modell av ett program är en förenklad beskrivning av programmets beteende. Med förenklad menas att modellen inte innefattar alla detaljer som ingår i programmet. Modellen kan vara på en högre abstraktionsnivå. För att betona skillnaden i abstraktionsnivå mellan programmet och modellen används också uttrycket abstakt modell.

För

3.2 Kriterium

Eftersom mängden möjliga testfall som kan genereras från modellen ofta också är oändlig, behövs ett kriterium för hur en ändlig mängd testfall ska genereras.

Det här kriteriet definierar något som ska täckas av de genererade testen. Om testsystemet är modellerat som en graf, så kan ett kriterium för val av testfall vara till exempel att besöka alla noder eller kanter. För komplexa system kan dock besök alla kantersom kriterium leda till för många testfall för att vara praktiskt.

3.3 Testfallsspecifikationer

3.4 Testgenerering

3.5 Testkörning

Eftersom testen som genereras från modellen är på en högre abstraktionsnivå än testsystemet, måste testen översättas för att de ska bli körbara. Det finns två huvudsakliga sätt som används för att göra abstrakta test körbara.

Första sättet är att programmera ett adaptor-program som sitter mellan MBT-verktyget och testsystemet [4]. Den här metoden används för on-line testgenerering. Testfallen genereras på samma gång som de körs. Då kan resultat från testsystemet påverka testgenereringen, vilket är till nytta vid testning av icke-deterministiska program.

Andra sättet är att generera direkt körbara testscript. Den här metoden används för off-line testgenerering. Testgenerering och körning sker här vid olika tidpunkter och testfallen kan inte anpassas efter testsystemets beteende under körning.

4 Tillfällig

Harmony är ett MBT-verktyg tillverkat av 4TestDev GmbH, som använder testverktyget Cypress för att köra genererade test. För modeller används i Harmony något som de kallar äction-state technique", en notation baserad på Gherkin. Zoltán Micskei upprätthåller en lista på MBT-verktyg [3].

Det finns två typer av testgenerering i modellbaserad testning, on-line och off-line. I on-line testgenerering genereras testen på samma gång som de körs,

medan testgenerering och körning av testen är separata steg i off-line testgenerering.

Hela systemet behöver inte modelleras och testas på en gång. Det går att spjälka upp specifikationen för programmet som testas till mindre delar och bygga skillda modeller för delarna [5].

5 Slutsats

Referenser

[1] Paul Ammann och Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[2] Bill Haskins m. fl. "8.4. 2 error cost escalation through the project life cycle". I: *INCOSE International Symposium*. Vol. 14. 1. Wiley Online Library. 2004, s. 1723–1737.

[3] Zoltán Micskei. *Model-based testing overview, tools and projects*. 2020. URL: `http://home.mit.bme.hu/~micskeiz/pages/mbt.html`.

[4] Mark Utting och Bruno Legeard. *Practical model-based testing: a tools approach*. Elsevier, 2006.

[5] Mark Utting, Alexander Pretschner och Bruno Legeard. "A taxonomy of model-based testing approaches". I: *Software testing, verification and reliability* 22.5 (2012), s. 297–312.

[6] Wikipedia. *Therac-25*. 2022. URL: `https://sv.wikipedia.org/wiki/Therac-25`.