MALMÖ UNIVERSITET

Teknik och samhälle
Datavetenskap och medieteknik

Examensarbete

15 högskolepoäng, grundnivå

Tillämpningar av modellbaserad testning i industrin - exempel på användning och hinder

Applications of model-based testing in the industry - examples of use and challenges

Oscar Norling

Karl-Olof Welin

Examen: Kandidatexamen 180 hp
Huvudområde: Datavetenskap
Program: Systemutvecklare, applikationsutveckling
Datum för slutseminarium: 2020-06-03

Handledare: Magnus Krampell
Examinator: Jesper Larsson

Sammanfattning

Det finns en omfattande litteratur kring modellbaserad testning (MBT) men med få tecken på att metoden har fått något större genomslag i industrin. Målet med studien är att identifiera exempel där MBT används som testmetod inom industrin och eventuella hinder som finns i MBT-processen. För att genomföra detta används en mixed-methods ansats bestående av en systematisk litteraturstudie följt av en utforskande fallstudie. I fallstudien tillämpas MBT med hjälp av verktyget Modbat på ett mjukvarusystem.

Endast ett fåtal industriella tillämpningar av MBT identifieras i litteraturstudien. Totalt sju studier kvarstår efter fulltextgranskningen. Studierna finns primärt inom mjukvaruindustrin och flygindustrin men innehåller även exempel från hälso- sjukvård och bilindustrin. Den utforskande fallstudien indikerar tre typer av hinder. Det första är mängden arbete med, samt bristande användarvänlighet hos verktygen. Den andra är svårigheten med att skriva ett adaptionslager som integrerar systemet med verktyget och modellen för att göra testfallen körbara. Det sista hindret är det kraftiga beroendet på att modellen utformas korrekt och stämmer med systemets tilltänkta beteende. Dessa tre hinder pekas även ut i verken från litteraturstudien. Vidare pekas bland annat även icke-tekniska svårigheter ut under litteraturstudien i form av att hela arbetsgruppen och ledningen behöver engageras för att införa ett nytt arbetssätt.

Med en begränsad fallstudie och ett enkelt system bekräftas tre hinder i MBT-processen som även identifieras i litteraturgenomgången. MBT framstår som ett primärt akademiskt område med ett fåtal exempel på användning inom industrin.

Abstract

There is extensive literature concerning model-based testing (MBT) but few signs that the method have had any major breakthrough in the industry. The goal of this study is to identify examples of MBT being used in the industry and any challenges faced during the MBT-process. The study is conducted using a mixed methods approach, consisting of a systematic literature review followed by an exploratory case-study. The case-study applies MBT to a software system using the MBT-tool Modbat.

Only seven studies remain after the fulltext review is performed. The studies are primarily from the software and aerospace industries but also include examples from the healthcare and automotive industries. The exploratory case-study identifies three challenges. The first one is the amount of work and lacking usability related to the MBT-tools. The second challenge is implementing the adaption layer, integrating the system under test with the tool and model to make test cases executable. The final challenge is the dependency on a correct model representing the systems expected behaviour. These three challenges are also identified in the systematic literature review. Other challenges from the literature review include non-technical difficulties concerning training and the need to motivate staff and management.

Using a limited case-study and a simple system three challenges, which are also identified in the literature review, throughout the MBT process are confirmed. MBT appears primarily as an academic subject with some examples of use in the industry.

Tack till vår handledare Magnus Krampell för trevliga diskussioner, fint stöd och feedback under uppsatsarbetet.

Ordlista och förkortningar

Adaptionslager, adaption layer - Det lager av kod som kopplar ihop systemet och modellen. Utgör ett gränssnitt mot systemet som möjliggör kommunikation med modellverktyget och modellen.

Avbrottssannolikhet - Innebär att det finns en användardefinierad sannolikhet för att exekveringen av testfallet ska avbrytas vid varje steg i modellen.

MBT - Modellbaserad testning.

Mutationsbaserad testning - Testmetod som innebär att fel introduceras slumpmässigt i modellen för att utvärdera om det specifika felet kan hittas i SUT.

Orakel, testorakel - Uppsättning information som kan svara på om utfallet från ett test motsvarar systemets förväntade beteende.

SUT, systemet - System under test, mjukvarusystemet som testas.

Innehåll

1 Inledning

1.1 Bakgrund

Mjukvarutestning kan definieras som att verifiera att ett system uppvisar det förväntade beteendet för en mängd testfall [1]. Automatiserad testning är på många sätt den moderna standarden för att verifiera mjukvarusystem. Continuous integration och att snabbt kunna lansera verifierad kod är av stor vikt och driver på automatiseringen av tester samtidigt som kraven på dessa skärps [2]. Även om exekveringen ofta är automatiserad är det vanligt att testfall skapas manuellt, antingen av specialiserade testare eller systemutvecklare.

Modellbaserad testning (MBT) är en arbetsmetod som ämnar underlätta testningsprocessen genom att automatisera både generering och exekvering av testfall samt utvärdera om utfallet från testerna stämmer med förväntat beteende. Detta genom att en modell som beskriver systemet och dess förväntade beteende formaliseras genom exempelvis UML eller någon liknande notation. Modellen används sedan för att generera både testfall och förväntat beteende för det faktiska systemet (System Under Test, SUT) [3].

MBT eliminerar inte arbetet kring test utan skiftar istället arbetet från att manuellt skriva testfall till att definiera modellen och koppla ihop denna med SUT. Detta gör att modellerings-kunskaper krävs men även att arbetet sker vid andra tidpunkter i utvecklingsprocessen. Detta hjälper till att integrera testning tidigt i utvecklingsprocessen genom att modellerna utvecklas parallellt med systemet. Utvecklingen av modellerna vid den här punkten hjälper till genom att identifiera motsägelsefulla och saknade krav [4]. Vidare krävs arbete med det så kallade adaptionslagret (adaption layer), alltså att faktiskt koppla ihop SUT så att test-fallen kan exekveras automatiskt [3].

1.2 Tidigare forskning

Modellbaserad testning har diskuterats i olika former sedan 1970-talet. Under de senaste 20 åren har ett uppsving skett både inom akademiska sammanhang och inom industrin [5]. I en survey från 2007 tittar Dias Neto m.fl. närmare på 271 artiklar relaterade till MBT och nästan lika många varianter och angreppssätt, ofta med tillhörande specialiserade verktyg [6]. Villalobos-Arias m.fl. utförde 2019 en litteraturstudie på tertiär nivå där de undersökte 22 litteraturstudier, varav 12 systematiska, publicerade mellan 1996 och 2016 [7]. I studien identifieras 98 verktyg för MBT. Trots den stora mängden verktyg så fanns det gap som verktygen inte täckte, exempelvis test av icke-funktionella krav. Vidare identifieras och kategoriseras de utmaningar som kvarstår inom MBT, effektivitet, tillgänglighet, komplexitet, utövarens färdigheter, kostnad och insats. Tillgänglighet identifierades som den största utmaningen.

Även om mycket av forskningen skett i akademisk miljö så har vissa studier lyckats påvisa praktiska resultat som fått genomslag även utanför akademin. En grupp forskare lyckades till exempel att med ett egenutvecklat verktyg (OAuthTester) hitta tre tidigare oupptäckta säkerhetsluckor i protokollet OAuth 2.0 [8]. Svagheter som enligt forskarna hade fått avsevärda konsekvenser om de utnyttjats. Bringmann och Krämer rapporterar om användning av MBT i kombination med modellbaserad utveckling inom bilindustrin

[9]. Marijan m.fl genomför två fallstudier tillsammans med grupper av testingenjörer som får generera testfall med MBT [10]. Sarma m.fl. utvärderar MBT-verktyg för verifiering av styrsystem för röntgenmaskiner och rapporterar sina erfarenheter [11].

Khan m.fl. har genomfört en litteraturstudie av empirisk forskning inom MBT som undersöker verkens rapporteringskvalité utifrån en uppsättning kriterier definierade av författarna [12]. Slutsatsen är att kvalitén generellt är låg, ungefär hälften av verken uppfyller mer än 50% av kriterierna. Khan m.fl. påpekar samtidigt att kvalitén ökat med tiden och senare verk innehåller tydligare beskrivningar av genomförandet. Vidare saknas ofta detaljer kring processen och analysen vilket leder till att det blir svårt återskapa metoderna samt inte går att hämta någon praktisk hjälp ur beskrivningarna. Slutligen påpekar Khan m.fl. att få studier har genomförts i en faktisk industriell miljö. I flera fall där studier utförts på eller i samarbete med industrin är det många gånger forskare som gör en fallstudie under kontrollerade former genom att låta en avdelning prova att arbeta med MBT som kompletterande testmetod.

Santos m.fl. undersöker om testingenjörer i industrin och forskare inom mjukvarutestning har olika syn på vad som är av vikt för att förbättra testprocessen och testverktygen. Slutsatsen är att akademiska forskare är mer fokuserade på nya tekniker och verktyg medan tillämpande testare i industrin är mer intresserade av att förbättra existerande metoder och verktyg. Grupperna möts däremot i intresset för testautomation [13]. Tillgänglighetsproblemet som identifieras av Villalobos m.fl. kan delvis ha en förklaring i slutsatserna som Santos m.fl. drar. Akademins strävan efter nya tekniker och verktyg gör att processerna och verktygen inte förbättras i den utsträckning som industrin efterfrågar. Även Weißleder och Schlingloff menar att det inte är självklart att många av verktygen för MBT fungerar vid industriella tillämpningar. Vidare pekar de på att det inte är den enda förklaringen utan att det även är svårt att i pågående projekt hitta tid och finansiering för att införa nya arbetssätt [4].

1.3 Problemformulering och forskningsfrågor

MBT har utvecklats i forskarvärlden under många år och det finns en omfattande litteratur. Samtidigt finns det få exempel på praktisk användning av MBT som testmetod utanför forskningsvärlden. Detta indikerar att MBT kan vara ett område som är akademiskt och teoretiskt relevant men som har begränsad spridning utanför akademin. Exempelvis drog Utting m.fl. redan 2012 slutsatsen att MBT är redo för storskalig lansering, något den inledande litteraturgenomgången visar få tecken på [5]. Detta leder till följande hypotes:

> *MBT har utvecklats inom forskarvärlden under en längre tid, men har begränsningar eller saknar tillräcklig mognad för att användas inom mjukvaruindustrin.*

För att verifiera eller falsifiera hypotesen formuleras följande två forskningsfrågor.

RQ 1: Vilka tecken på användning av MBT inom industrin kan utläsas av publicerade forskningsrapporter? Med ”användning” avses genomförande av testning baserat på MBT som drivs av företag inom industrin, inte av akademiska forskare, som använder industrin som exempel.

RQ 2: Vilka svårigheter och hinder för att använda MBT inom industrin finns beskrivna i forskningsrapporterna från RQ1 och vilka svårigheter och hinder kan hittas i en utforskande fallstudie?

1.3.1 Förtydligande och avgränsningar

RQ 1: Besvaras genom en litteraturstudie samt jämförelse av utvalda publikationer. Litteraturstudien är begränsad till den akademiska litteraturen och inkluderar därför inte publikationer i andra kanaler. Tillämpningar i industrin definieras som fall där någon aktör använder MBT utanför ramarna för ett forskningsprojekt. Fall där en forskargrupp utför MBT på befintlig mjukvara med forskning som enda syfte inkluderas inte.

RQ 2: Besvaras delvis utifrån litteraturstudien genom att lista identifierade hinder och svårigheter. Detta svar kompletteras genom att utföra en utforskande fallstudie där en modell konstrueras och ett MBT-verktyg används för att verifiera ett mjukvarusystem. Detta ämnar påvisa praktiska svårigheter och hinder under arbetet med MBT.

2 Modellbaserad testning

Utting m.fl. klassar MBT som funktionella tester utförda med ett black-box angreppssätt [3]. Funktionella tester beskrivs som test av vad systemet kan utföra, vilka funktioner det har och att deras beteende stämmer överens med krav och förväntningar. Funktionella tester kan därmed inte testa exempelvis användbarhet, prestanda eller andra icke-funktionella egenskaper. Utting m.fl. påpekar dock att MBT i viss utsträckning kan användas för att testa hur robust ett system är [3].

Att MBT utgår från ett black-box angreppssätt innebär att designen av testfallen inte använder information kring hur systemets kod är strukturerad [3]. Eftersom testfallen i MBT utgår från en modell baserad på krav och artefakter från utvecklingsprocessen som beskriver det förväntade beteendet av systemet blir det naturligt att metoden klassas som black-box.

MBT kan utföras på alla olika nivåer, från enhetstest till systemtest. Det är möjligt att utföra både test på individuella moduler, integrationstest, där flera moduler testas tillsammans, eller fullständiga systemtest [3].

2.1 Vad är modellbaserad testning?

Den ytliga beskrivningen av modellbaserad testning i inledningen är inte en fullständig beskrivning av MBT. I denna del beskrivs MBT utförligare samtidigt som begreppet definieras inom uppsatsens ramar.

Det finns fyra övergripande metoder som kan klassas som MBT [3].

1. Generera indata för testfall utifrån en domänmodell.
2. Generera testfall utifrån en modell av systemets miljö/beteende.
3. Generera testfall med tillhörande orakel utifrån en beteendemodell.
4. Generera testskript utifrån abstrakta tester.

I den första varianten genereras indata utifrån en domän av möjliga värden på indata. Utting m.fl. menar att det löser ett tydligt praktiskt problem med att exempelvis minimera antalet testfall men är samtidigt inte en fullständig lösning på problemet med testdesign.

Den andra varianten beskriver den förväntade miljön för systemet, exempelvis genom en statistisk modell över systemets användning. Denna variant kan generera serier av anrop som simulerar användning men det saknas fortfarande ett automatiserat sätt att utvärdera om testerna uppfyller det förväntade beteendet eller ej.

Den tredje varianten är utförligare och innefattar både exekverbara testfall och orakel innehållande information kring vilka utfall testerna förväntas ha. Detta är en mer komplicerad modell än de två första. Metoden är svårare eftersom modellen behöver ha mer nyanserad information som inkluderar förväntat beteende hos systemet för att kunna avgöra om systemets beteende motsvarar förväntningen på systemet.

Den fjärde varianten är något annorlunda och innebär att transformera abstrakta testfall till körbara test. Detta genom att exempelvis använda information rörande systemets APIer och struktur för att göra testet körbart.

När vi beskriver MBT är det framförallt den tredje definitionen som avses.

2.2 Testprocessen

Processen för MBT finns beskriven på flera ställen [3], [5]. Följande process är en sammanfattning av stegen som beskrivs i [3] och [5].

1. Konstruera en modell över systemet utifrån formella och informella krav samt tillgängliga artefakter.
2. Definiera kriterier för testfallen - målet med testerna.
3. Transformera testfalls-kriterium till en specifikation för testfallen - översätt kriterierna till ett set av relevanta specifikationer.
4. Generera testfall utifrån specifikationen.
5. Exekvera testfallen.

1) Att definiera en modell över systemet är inte en trivial uppgift. Målet med modelleringen kan sammanfattas som att på ett logiskt sätt beskriva systemets förväntade beteende utan onödiga detaljer. Detta kräver att rätt abstraktionsnivå för modellen identiferas. Modellen behöver beskriva systemet tillräckligt detaljerat för att kunna dra slutsatser om förväntat beteende samtidigt som så lite av systemet som möjligt ska modelleras [3]. Vidare behöver modellen utformas efter det MBT-verktyg som används så att modellen blir tolkningsbar för verktyget. MBT kan användas på enskilda moduler eller för att testa delar av system. Om hela systemet inte ska testas modelleras endast de delar som är relevanta för målet med testningen.

Modelleringen utförs ofta så att modellen beskriver systemet som en finit tillståndsmaskin. Detta är däremot inget krav, en modell kan definieras på flera andra sätt, exempelvis genom dataflöden [5]. Modellerna beskriver abstraktionen av systemet på ett logiskt sätt enligt en specifik notation. Det vanligaste exemplet på notation är UML men det finns en stor mängd alternativ så som Z, SysML, Gherkin eller Markov kedjor [14]. Det är många gånger kompatibilitet med MBT-verktyget som styr vilken notation som används. Kopplingen mellan verktygen och notationen är stark och Sarma m.fl. påpekar att verktygen skulle behöva stödja flera typer av modeller för att förenkla byte av verktyg eller notation [11].

2) Kriterium för testfallen kan för en tillståndsmaskin vara exempelvis att täcka alla tillstånd, att exekvera alla möjliga tillståndsövergångar eller att mata modellen med slumpmässig indata [5].

3) Översättning av kriterierna till modellen, exempelvis den uppsättning tillstånd som behöver nås för att alla tillstånd i modellen ska täckas in. Steget är en konkretisering av målet för testerna till den givna modellen [5].

4) Testfall genereras så att kriterierna i *3)* kan utvärderas. Genererandet av testfallen sker med hjälp av det MBT-verktyg som används. Det finns flera tekniker för att generera testfallen och tillgängligheten varierar med verktygen. Utting m.fl grupperar dessa som:

i) Stokastiska

ii) Söknings-baserade

iii) Bounded model checking

iv) Symbolic execution

v) Deductive theorem proving

vi) Constraint solving

Teknikerna passar olika väl beroende på vilken typ av system det är och vad testningen avser [5].

5) Slutligen exekveras testfallen. För att automatiskt kunna exekvera testfallen behöver testverktyget anslutas till SUT. För att integrera verktyget och systemet skrivs ett adaptionslager som utgör ett gränssnitt mellan systemet och testverktyget. Det finns tre övergripande sätt att skriva adaptionslagret beskrivna av Utting m.fl. [3]:

i) adaption innebär att skriva en abstraktion runt systemet. Detta innebär att det går att skriva funktioner som utför flera steg på systemet för att exempelvis utföra pre-conditions för testerna. Generellt ska abstraktionsnivån matcha den i modellen så att stegen från modellen kan kopplas direkt till systemet.

ii) transformation innebär motsatsen, alltså att adaptionslagret ska generera testskript som kan köras direkt på systemet.

iii) mixad innebär en kombination där testverktyget genererar testskript som är körbara på en abstraktion av systemet. Denna kombination ämnar vara mer användbar då det test-skripten inte behöver vara lika detaljerade samtidigt som abstraktionen av systemet inte behöver gå lika långt. Detta gör att abstraktionen av systemet kan återanvändas i större utsträckning till exempelvis flera olika modeller för olika delar av systemet.

När testerna slutligen körs ska, enligt MBT definitionen som används, även oraklen besvara om varje testfall lyckats eller misslyckats. Sammanfattning av resultaten är i stort beroende av det använda testverktyget samt implementationen.

3 Metod

Forskningsfrågorna besvaras med hjälp av så kallade mixed methods bestående av två forskningsmetoder. En systematisk litteraturstudie genomförs för att identifiera exempel på reella tillämpningar av MBT. Litteraturstudien följs av en utforskande fallstudie där MBT tillämpas på ett befintligt system med hjälp av ett MBT-verktyg.

En utforskande fallstudie är ett sätt att synliggöra och ge bättre förståelse för ett forskningsområde genom att ett praktiskt exempel undersöks. Metoden kan inte ge generaliserbara svar utan används istället för att hitta hypoteser och specificera frågeställningar för vidare studier [15].

Sammansättningen av de två metoderna passar väl för hypotesen och forskningsfrågorna som beskrivs i avsnitt 1.3 då litteraturstudien kan identifiera praktiskt användande av MBT och eventuella hinder. Samtidigt gör fallstudien att eventuella hinder undersöks praktiskt och kan jämföras med hinder från litteraturen för att ge ett något mer generaliserbart resultat.

3.1 Litteraturstudie

Litteraturstudien utgörs av en systematisk litteraturgenomgång av praktiska tillämpningar av MBT. Sökningarna görs via databaserna hos Association for Computing Machinery (ACM) och Institute for Electrical and Electronics Enginers (IEEE). Litteraturstudien skulle kunna omfatta flera databaser för att öka omfattningen av träffar som i exempelvis [12]. Uppsatsens omfattning gör att fler databaser inte inkluderas. Vidare indikerar pilot-sökningar att möjligheten och utformningen av avancerade sökningar varierar mellan databaserna. Detta medför att desto fler databaser som inkluderas desto större skillnader blir det mellan sökningarna och resultaten blir för snäva eller för breda för att vara hanterbara. Granskningen sker i tre steg. Inledningsvis sorteras verk bort baserat på titel. För kvarstående verk granskas sammanfattningen och slutligen granskas fulltext för de kvarstående.

3.1.1 Sökningar

Följande sökning utförs via IEEE. Sökningen gäller till och med mars 2020.

> (“Document Title”: “Model-based test*” AND (“Document Title”: empirical OR “Document Title”: “case-study” OR “Document Title”: industr* OR “Document Title”: experience OR “Document Title”: experiment OR “Document Title”: practic* OR “Document Title”: application)
> OR
> (“Document Tite”: “Model-based test*” AND (“Abstract”: empirical OR “Abstract”: “case-study” OR “Abstract”: industr* OR “Abstract”: experience OR “Abstract”: experiment OR “Abstract”: practic* OR “Abstract”: application))

Sökningen utvärderas även med att kräva MBT i sammanfattningen istället för titeln. Detta försök överges eftersom en markant andel av träffarna snabbt kan konstateras sakna relevans för forskningsfrågorna.

Motsvarande sökning för ACM används:

> Title:((“model-based testing” OR “model-based test”) AND (empirical OR “case-study” OR industr* OR experience OR experiment OR practic* OR application))

3.1.2 Inklusionskriterier

Studier av praktiska tillämpningar av MBT inkluderas, detta innebär fall där MBT används i syfte att verifiera någon mjukvara.

Följande typer av studier inkluderas ej:

- Studier där ett ramverk eller arbetsprocess för MBT föreslås men utan praktisk tillämpning bortom att bekräfta det föreslagna artefaktet.
- Studier som fokuserar på mjukvarumässiga förbättringar för MBT, exempelvis algoritmutveckling för urval av testfall.
- Studier där forskare använder MBT på en mjukvara med målet att verifiera att MBT fungerar istället för att MBT används för att verifiera mjukvaran.

3.2 Utforskande fallstudie

Fallstudien består av fyra steg som beskrivs nedan.

3.2.1 Val av system att testa

Målet för testsystemet är att det ska vara skrivet av en tredje part. Detta för att det inte ska finnas en mental bild av systemets uppbyggnad då det kan påverka utformningen av modellen. På så sätt följs en så realistisk arbetsprocess som möjligt via ett black-box angreppssätt och eventuella svårigheter med att genomföra modellering och testning synliggörs. Systemet ska vara skrivet utan omfattande tredjepartsbibliotek eller ramverk för att säkerställa att testfall och eventuella defekter beror på systemet och inte följer av externa beroenden.

3.2.2 Val och utvärdering av MBT-verktyg

Tillgängliga MBT-verktyg identifieras i den inledande litteraturgenomgången och litteraturstudien. Val av MBT-verktyg görs enligt tre kriterier: verktygen ska vara fritt tillgängliga, exempelvis släppta med öppen källkod, ha tillhörande dokumentation och aktivt underhållas. Slutligen körs inledande test för att säkerställa att verktyget fungerar korrekt.

3.2.3 Modellkonstruktion

Modelleringen sker i tre steg. I det första steget identifieras övergripande programflöde genom att använda systemet samt tillhörande dokumentation och artefakter. I det andra steget definieras modellens logik samt vilken data den behöver hålla reda på för att verifiera

beteendet hos SUT. Det sista steget är implementation av adaptionslagret och innebär att modellen kopplas ihop med SUT så att testfallen kan exekveras.

3.2.4 Genomförande av tester

För att undvika osystematisk testning av SUT och för att ha en tydlig process att testa efter används en testplan. Målet med testplanen är att på ett strukturerat sätt visa vilka egenskaper och fel som kan verifieras i systemet.

Den första delen av testplanen innehåller steg som inte introducerar några fel utan istället verifierar att ett korrekt system passerar testerna utan att något flaggas som fel. I den andra delen av testplanen introduceras fel i SUT manuellt, följt av exekvering av genererade testfall.

4 Resultat och analys

4.1 Litteraturstudie

Stegen i den systematiska litteraturgenomgången av den slutgiltiga sökningen sammanfattas i tabell 1 nedan.

Tabell 1: Resultat från systematisk litteraturgenomgång

Steg	**Antal**
Inledande sökning	143
Efter dubbletter och felaktiga resultat	119
Efter granskning av titel	69
Efter granskning av sammanfattning	45*
Efter granskning i fulltext	7
Totalt	7

*Tre studier exkluderas eftersom de inte finns tillgängliga i fulltext.

En lista över verken som granskas i fulltext finns i appendix A. En sammanfattning över skälen att de exkluderas finns i tabell 2.

Tabell 2: Skäl till exklusion vid fulltextgranskning

Skäl	**Antal**
MBT valideras genom tillämpning i industrin	24
Utveckling av MBT-metod	10
Utvärdering av verktyg	1
Berör endast MBT inom forskning	1
Berör endast inställning till MBT	1
Berör inte MBT empiriskt	1
Kvar i fulltext	7
Totalt	45

Studierna som granskas i fulltext kan klassas i två huvudsakliga kategorier. Den första och mest prevalenta är fallstudier där MBT används på ett system ur industrin. Denna grupp fångas upp i sökningen till följd av det tydliga empiriska arbetet men är i de flesta fall inte relevant för frågeställningarna. Ansatsen kan sammanfattas som att validera att någon tillämpning av MBT fungerar. Dessa studier har i de flesta fall inga konkreta exempel på att MBT används vid mjukvaruutveckling i industrin utan använder istället industrin för att validera MBT.

Den näst största kategorin är så kallade experience reports, alltså rapporter från användning av MBT. De flesta studier som inkluderas i den slutgiltiga granskningen kommer från denna kategori och inkluderar praktiska exempel på användning av MBT. Gruppen innehåller exempel både från akademiska tillämpningar och reella exempel från industrin.

Resterande forskningsrapporter består av olika typer av studier, exempelvis surveys, experiment och empiriskt baserade jämförelser mellan MBT och någon form av traditionell

testning. Den största andelen studier från dessa grupper som exkluderas är till följd av de inte är relevanta för frågeställningen vid fulltextgranskning. Nedan följer sammanfattningar av de forskningsrapporter som inkluderas efter fulltextgranskningen.

4.1.1 Sammanfattning av inkluderade studier

Altinger, Wotawa och Schurius genomför en survey av testmetoder i bilindustrin [16]. Urvalet består av 45 personer som identifierats arbeta med testning. De tillfrågade uppmuntrades även skicka frågorna vidare till relevanta kollegor och avdelningar. Totalt 64 svar mottogs, varav 90% från industrin. I svaren uppger 32-38% att de använder MBT. Den högsta andelen är i gruppen som arbetar på testavdelningar (38.52%), medan de lägre värdena gäller för forskning(32.26%), för-produktion(32.58%) och serie-produktion (35.96%). Studien visar att MBT används inom bilindustrin, men det är svårt att avgöra var till följd av grupperingen efter testavdelningar, som kan utföra arbete under flera olika faser av utveckling och produktion.

Stobie rapporterar om hur MBT används på Microsoft. Två MBT-metoder som har utvecklats och används av Microsoft beskrivs. Den första är ett verktyg som modellerar tillståndsmaskiner. Den andra är ett exekverbart språk för abstrakta tillståndsmaskiner samt tillhörande verktyg. Stobie går igenom några exempel på fel som identifierats med hjälp av verktygen samt var de används [17].

Entin m.fl. rapporterar sina erfarenheter från att börja arbeta med MBT i ett scrumprojekt rörande diagnostik och testverktyg inom el-industrin [18]. Författarna rapporterar utförligt om sitt införande, både organisatoriskt och arbete med testerna. Slutsatserna rör primärt att olika verktyg inte är kompatibla med varandra samt att en stor del av det inledande arbetet inte är tekniskt, utan istället handlar om att förklara, lära upp och få med teamet i arbetet. På samma sätt är det inte självklart att ledningen är övertygad om huruvida MBT är en lämplig testmetod. Underhåll av modellen diskuteras även samt att det krävs versionering av tillhörande artefakter för att kunna utföra tester med äldre versioner av modellen. Entin m.fl. beskriver det vanligaste modelleringsfelet som att modellen fastnar i en del av tillståndsmaskinen och inte kan lämna den. Slutligen konstateras det delvis att kommersiella modelleringsverktyg är dyra samt att verktyg som använder samma modelleringssätt inte är kompatibla med varandra och att det därför är svårt att byta ut verktyg och modeller [18].

Ganesan m.fl använder MBT på ett av NASAs system som komplement till enhetstester. Enhetstesterna har nästan 100% line coverage men tar inte samtidighet (concurrency) i beaktning vilket MBT-tillämpningen ämnar göra. De använder sig av Microsoft SpecExplorer för att modellera systemet och generera testfall. Testerna sker på en nivå där testfallen genereras i körbar form via en adapter men resultaten måste granskas av en testingenjör. De hinder som Ganesan m.fl. beskriver rör modelleringsfel till följd av feltolkade krav och felaktigheter i adaptern. Vidare beskrivs risken att en ohanterbar mängd testfall genereras. Ganesan m.fl. beskriver även svårigheten som uppstår med att modellera ett system med odokumenterade krav eftersom det förväntade beteende inte går att identifiera vilket resulterar i felaktiga modeller [19].

Wendland m.fl. använder MBT inom ramen för mjukvaruutveckling och förnyelse av ett

äldre system för reseplanering. Modellen byggs för att beskriva funktionaliteten hos det tidigare systemet och sedan genereras testfallen och körs på det nya systemet för verifiera att funktionaliteten är på plats. Författarna rapporterar om den använda designen och arkitekturen samt val och svårigheter från processen. Slutsatserna är att tidigare erfarenhet av modellering, MBT och verktygen var en stor hjälp samt att inga större förändringar i metodologi behövdes för förnyelsearbetet. Svårigheterna sammanfattas som brist på detaljerade funktionella krav samt tiden det tar att konstruera adaptionslagret och göra testfallen exekverbara [20].

Vieira m.fl. använder MBT för att verifiera mjukvarusystem för sjukvården, men använder domänexperter som inte vanligtvis arbetar med mjukvaruutveckling som konstruerar modellerna. Resultaten av detta är blandade. Arbetssättet beskrivs fungera och att det finns fördelar med domänexperter som tar fram modellerna. Samtidigt påverkas kvalitén i modellerna negativt ur ett mjukvaruperspektiv genom att exempelvis modularitet och integrationsmöjligeter inte beaktas. Detta medför att stöd krävs från deltagare med mer erfarenhet av mjukvaruutveckling för att göra modellerna bättre strukturerade. Det mest tydliga hinder som Viera m.fl. diskuterar rör att sätta upp SUT så att det innehåller relevant data för det specifika testfallet. Detta föreslår de en lösning på genom en testdatabas där relevant testdata kan hämtas beroende på testfallet [21].

Lindvall m.fl. använder MBT i kombination med metamorfisk testning för att testa ett gränssnitt till ett omfattande databassystem för telemetrisk data under utveckling hos NASA. Då användaren ej på förhand känner till resultatet av ett anrop till systemet så kan ej något testorakel genereras, ett problem de angriper och levererar en tänkbar lösning på. För att generera testfall ur modellen använde de ett egenutvecklat verktyg, the FAST tool. De anser resultatet lovande och då mjukvarusystemet de testar är under fortsatt utveckling kommer testmetoden fortsätta att användas [22].

4.1.2 Sammanfattning av identifierade hinder för MBT-processen ur litteraturstudien

Till följd av studietypen innehåller varken Stobie eller Altinger någon direkt information kring hinder i MBT-processen.

Ganesan m.fl. beskriver hinder i form av: fel i modellen, problem till följd av en felaktig adapter samt svårigheten att modellera förväntat beteende när det finns odokumenterade krav. Slutligen nämns problem som uppstår när mängden testfall exploderar och exekveringen blir ohanterbar [19].

Hindren och svårigheterna som Entin m.fl. rapporterar på den tekniska sidan är att olika verktyg inte är kompatibla med varandra även om samma modelleringssätt används. Vidare noterar de att det generellt tar tre iterationer innan en modell fungerade och att ett av de vanligaste modelleringsfelen är fall där modellen fastnar i en del av tillståndsmaskinen. Trots detta menar Entin m.fl. att ett av de största hindren inte är tekniskt utan istället handlar om att engagera och få med sig medarbetarna och ledningen [18].

Wendland m.fl. beskriver bekymmer med bristande funktionella krav och bristande detaljer i kraven. Vidare argumenterar de för att adaptionslagret och arbetet med att göra testfallen exekverbara tar en stor mängd tid i anspråk [20].

Vieira m.fl. beskriver endast ett hinder som kan relateras till MBT, att relevant testdata måste finnas tillgänglig i datadrivna system för att kunna exekvera testfall som är beroende av data. Ett förslag på en lösning är att konstruera en databas innehållande nödvändig data som kan hämtas för respektive testfall, men detta kräver utveckling och generering av databasen [21].

Lindvall m.fl. beskriver och hanterar ett tangerande problem där testorakel inte kan genereras för ett databassystem då resultatet från databasen inte går att förutsäga [22]. Detta är inte ett generellt hinder för MBT men visar att det inte nödvändigtvis alltid går att definiera orakel vilket i vissa fall påverkar användningen av MBT.

4.2 Utforskande fallstudie

Resultatet från fallstudien är uppdelat enligt stegen som beskrivs i metodavsnittet. Först beskrivs SUT, följt av testverktyget och modellen. Därefter presenteras testplanen och slutligen rapporteras genomförandet samt de hinder som påträffas på under arbetet.

4.2.1 Testsystemet (SUT)

Testsystemet består av mjukvara som emulerar en bankomat, skriven i Java. Källkoden är hämtad ur en lärobok i Java och finns tillgänglig i sin helhet i boken [23].

Systemet innehåller konton, pinkoder och saldon i en Java-klass som emulerar en bankdatabas. Operationer på bankomaten innefattar auktorisation, saldokontroll, uttag och insättning av pengar. Systemet emulerar även skärm och tangentbord för kommunikation med användaren via terminalen.

Systemet väljs för att det erbjuder viss komplexitet utan att vara för omfattande för sammanhanget. Vidare är det skrivet med endast Javas standardbibliotek vilket underlättar testprocessen och minskar osäkerheten kring resultaten. Detta eftersom eventuella kompabilitetsproblem med exempelvis externa ramverk undviks och fel som upptäcks sannolikt är logikfel eller modelleringsfel.

För att kunna genomföra testerna krävs vissa ändringar i systemet. Ändringarna avser att ge verktyget, och därigenom modellen, kontroll över flödet av indata till systemet. Ändringarna utförs i samband med utveckling av modellens adaptionslager.

4.2.2 Testverktyg

Verktygen kan förenklat delas upp i två kategorier, akademiska eller öppen källkod och kommersiella. En sammanställning av MBT-verktyg finns i [14]. Kvalitén på de fritt tillgängliga verktygen varierar kraftigt, medan de kommersiella är svårutvärderade till följd av bristande tillgång.

Underlaget för vårt val av verktyg baseras på litteraturstudien samt Micskeis sammanställning av tillgängliga MBT-verktyg [14]. Totalt identifieras 27 verktyg varav tio ickekommersiella. Efter att i en första genomgång sorterat bort verktyg som ej själv kan exekvera testfallen står valet mellan fem verktyg, GraphWalker, MoMuT-UML, ModelJUnit, OSMO och Modbat.

GraphWalker och MoMuT-UML är verktyg som nämns vid flera tillfällen i litteraturen. Både verktygen är mutationsbaserade. Detta innebär att fel introduceras slumpmässigt i modellen och för att utvärdera om det specifika felet kan hittas i SUT. Då mutationsbaserad testning är en särskild inriktning av MBT väljs dessa verktyg bort eftersom MBT-processen annars blir specifik för denna typ av testning.

Den senaste versionen av ModelJUnit är från 2014. Vi väljer att fokusera på nyare verktyg för att både fånga upp eventuella förändringar inom MBT-utvecklingen, men även då det sannolikt är färre bekymmer med hantering av äldre beroenden och versioner av tillhörande mjukvara.

OSMO och Modbat är på många sätt likvärdiga med varandra enligt kriterierna definierade i metodavsnittet. De underhålls aktivt och har båda god dokumentation. En stor del av utvecklingsarbetet med OSMO är däremot äldre. Detta leder till att vi väljer Modbat då det är väl dokumenterat med guider för installation, modellering och testexekvering, samt underhålls aktivt.

4.2.3 Modbat

Modbat använder modeller utformade som extended finite state machines (EFSM) skrivna i programmeringsspråket Scala. Modbat finns tillgängligt som .jar-fil eller via källkoden på Github [24]. Modbat läggs till som en modul i utvecklingsmiljön eller körs direkt från terminalen och innehåller alla nödvändiga funktioner för att modellera och köra tester.

Kortfattat traverserar Modbat modellens tillstånd och väljer slumpmässigt en exekveringsväg utifrån modellens nuvarande tillstånd. Om inga exekveringsvägar i nuvarande tillstånd finns så avslutas testfallet. Den som modellerar har därför kontroll över möjliga exekveringsvägar och sekvenser. Verktyget kan hantera både ändliga och oändliga modeller då Modbat innehåller mekanismer för att undvika oändliga körningar. Mekanismerna är en avbrottssannolikhet samt kontroll av upprepade loopar. Avbrottssannolikheten innebär att det finns en användardefinierad sannolikhet för att exekveringen av testfallet ska avbrytas vid varje steg i modellen. Testfallet avbryts när någon av mekanismerna aktiveras eller när ett fel påträffas, vilket innebär att systemets tillstånd inte matchar modellens efter ett tillståndsbyte. Då fel påträffats loggar Modbat stegen som ledde fram till felet i en separat fil för just det testfallet.

Modeller för Modbat

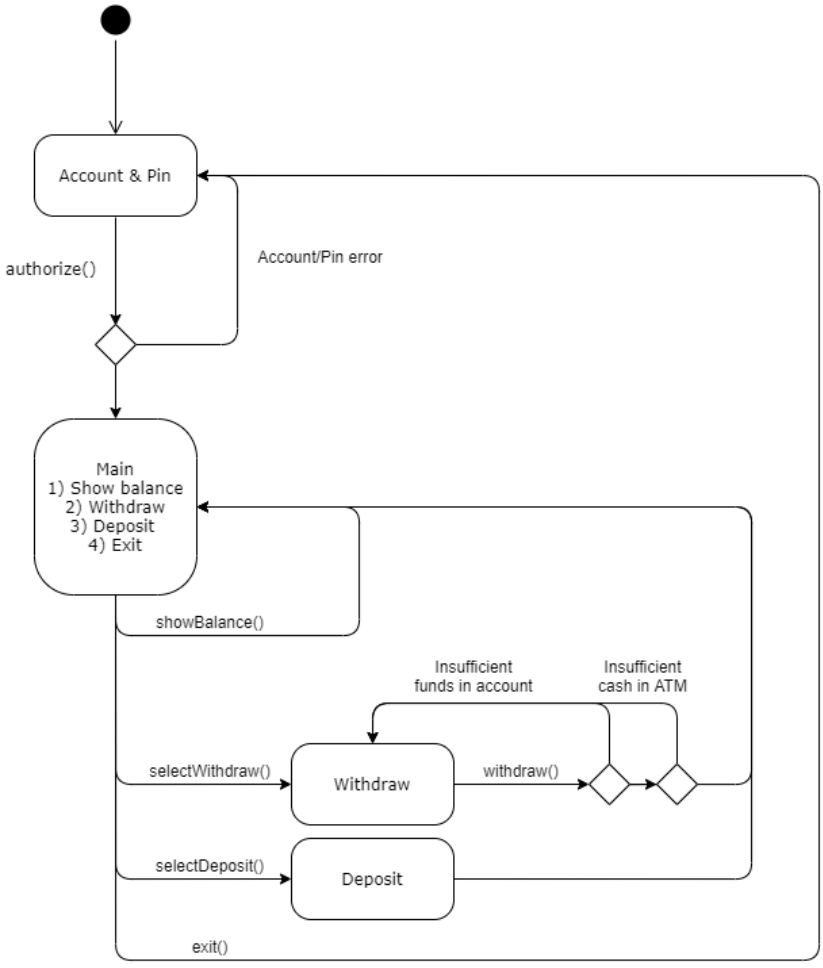
Kodexemplet nedan är hämtat från ett exempel på en Modbatmodell som modellerar en enkel räknare i Java [24]. När ett testfall körs startar Modbat i det första tillståndet, i detta fall “reset”, väljs slumpmässigt mellan möjliga övergångar och kör anropen inom klammerparanteserna. När Modbat når ett tillstånd utan övergångar avslutas testet. Modbat innehåller även mekanismer som preconditions, guards med mera för att i detalj kunna modellera systemet. För ytterligare detaljer kring modelleringsprocessen med Modbat se [24][25].

```
'reset' -> 'zero' := { counter = new SimpleCounter() }
'zero' -> 'one' := { counter.inc }
'one' -> 'two' := { counter.inc }
```

```
'zero' -> 'two' := { counter.inc2 }
'two' -> 'end' := { assert (counter.value == 2) }
```

4.2.4 Modell över bankomatsystemet

Modellen som används för att testa systemet illustreras i figur 1. Figuren är en översikt av modellen som beskriver flödet och de huvudsakliga tillstånden. Arbetet med modelleringen presenteras tillsammans med erfarenheterna från MBT-processen i sektion 4.2.6.



Figur 1: Modellöversikt

Modellens olika tillstånd och övergångar representerar testsystemets olika menyer och vägar in/ut ur dem. Efter att en övergång traverserats bekräftar modellen en uppsättning villkor för att se att testsystemet befinner sig i ett förväntat giltigt tillstånd. Hela modellen så som den används vid testningen finns i appendix B.

4.2.5 Testplan

Följande punkter valideras under testningen. Först presenteras steg som verifierar att modellen inte indikerar korrekta körningar som fel.

KF-1 Korrekt kombination av pinkod och kontonummer verifieras och systemet går vidare till nästa tillstånd (main).

KF-2 Inkorrekt kombination av pinkod och kontonummer accepteras ej och systemet står kvar vid inloggning.

KF-3 Verifiera att korrekta uttag utförs. Aktuella saldon verifieras efter uttag med förväntade uppdaterade värden samt att systemet befinner sig i förväntat tillstånd.

KF-4 Verifiera att insättningar utförs korrekt. Aktuella saldon verifieras med förväntade nya värden samt att systemet befinner sig i förväntat tillstånd.

KF-5 Verifiera utloggning. Systemet ska återgå till starttillstånd och eventuella kontouppgifter ska vara rensade.

KF-6 Verifiera att utförda ändringar lagras mellan ut- och inloggning.

Följande fel introduceras sedan ett i taget:

IF-1 Tillgängligt saldo ändras istället för totalt saldo.

IF-2 Vid uttag ur bankomaten räknas antalet sedlar inte ner.

IF-3 Omräkningen från cent till dollar utförs felaktigt.

IF-1 innebär att bekräftelse av mängden inmatade pengar inte utförs, och istället ändras tillgängligt saldo på det inloggade kontot direkt. Fallet IF-2 identifieras oavsiktligt under utvärderingen av systemet och innebär att modulen som matar ut sedlar inte räknar ner antalet sedlar vid uttag ur bankomaten. Felet inkluderas då det är ett reellt exempel ur SUT. IF-3 motsvarar ett enkelt räknefel där konverteringen från cent till dollar sker genom att dividera med 10 istället för 100.

4.2.6 Erfarenheter från MBT-verktyg, modellering samt implementation

Testverktyg och SUT
Processen börjar med att både Modbat och testsystemet provkörs för att bekräfta funktionen oberoende av varandra. Modbat används inledningsvis tillsammans med exemplen ur tillhörande dokumentation [24]. En del problem med mjukvaruversioner och kompatibilitet upplevs här men går att lösa med hjälp av tillhörande dokumentation. Nästa steg är att validera ett väldigt enkelt egenkonstruerat system och tillhörande modell. I det här steget upplever vi flera olika typer av problem som inledningsvis är svåra att separera från varandra.

Det första problemet gäller att kompilera Scala-modellen tillsammans med systemet. Eftersom Modbat körs från terminalen kompileras även Java och Scala programmen manuellt där. Detta leder till att modellen inte hade tillgång till systemet. Lösningen blir istället att använda en integrerad utvecklingsmiljö (IDE) för att kompilera modellen och systemet tillsammans och sedan utföra testerna via terminalen och den kompilerade modellen.

Det andra problemet rör classpath och inställningarna som behöver göras för att exekvera modellen via Modbat. Det här steget kräver mindre felsökning men upplevs trots det inte som trivialt. Sammanfattningsvis går problemen att lösa med hjälp av dokumentationen men det krävs tid för varje enskild del.

Provkörningen av SUT var i stort oproblematiskt. Detta beror till största del på att systemet är avskalat och enkelt. Eftersom bankomatsystemet är skrivet helt med Javas standardbibliotek går det att exekvera redan vid första försöket. Inga problem med kompabilitet identifierades.

Modellering
Modelleringen sker i tre steg. Först identifierar vi ett övergripande programflöde genom att utforska systemet som användare. Under en normal process innehåller detta steg granskning av artefakter och krav för att fånga förväntat beteende hos SUT. I detta fall finns inget kravarbete och få detaljer kring hur och vad systemet ska göra vilket gör det svårare att få till rätt detaljnivå i modellen. Bristen på artefakter leder till osäkerhet kring om hela systemets förväntade beteende faktiskt modelleras. Vi översätter det övergripande programflödet till en enkel tillståndsmaskin. Tillståndsmaskinen beskriver systemet på en hög nivå motsvarande beskrivningen i figur 1. I det andra steget definierar vi modellens övergångar samt vilken data den behöver hålla reda på för att verifiera motsvarande värden i SUT. Exempelvis bekräftas vissa tillstånd genom de värden SUT skickar till bankomatens skärm. I detta steg arbetar vi mer aktivt mot systemets källkod då det saknas artefakter från kravprocessen.

Adaptionslager
Det sista steget i modelleringsprocessen innan Modbat kan användas för att verifiera SUT med hjälp av modellen är att skriva ett adaptionslager, alltså att koppla samman modellen och SUT. Detta steg beskrivs övergripande i de flesta verk som förklarar MBT processen men utan närmare detaljer. Steget visar sig vara ett av de mest komplicerade att genomföra i praktiken. I stort behöver verktyget kunna kontrollera tre saker, vad som visas på bankomatens skärm, indata till systemet via bankomatens emulerade tangentbord och systemets interna värden för bland annat. saldon och sedelräkning.

Adaptionslagret implementeras huvudsakligen enligt adaptions-metoden som beskrivs i avsnitt 2.2, alltså att en abstraktion skrivs kring systemet för att matcha modellens abstraktionsnivå. Istället för att kapsla in hela systemet implementeras en klass i bankomaten som får tillgång till alla övriga klasser. På detta sätt kan adaptionsklassen nå relevanta data samtidigt som SUT fungerar som tidigare. En del ändringar i SUT krävs för att få adaptionsklassen att fungera. Det handlar till största del om att ändra tillgång och sprida referenser till adaptions-klassen. Detta löser kontroll av systemets värden samt vad som visas på skärmen, men inte kontroll över indata.

Då systemet emulerar input från ett numeriskt tangentbord via inmatning av siffror i terminalen behöver modellen få kontroll över inputströmmen. Inledningsvis löses detta genom att byta ut inputströmmen från terminalen mot en ström innehållande en kontrollerad serie indata. Detta fungerar vid inledande försök men misslyckas vid exekvering av testfallen då det inte tillåter att dynamisk indata genereras beroende på modellens tillstånd.

Den lösning som istället används är att koppla bort tangentbords-avläsningen från ter-

minalen och ersätta den med indata som kommer från adaptions-klassen. Detta gör att motsvarande indata som en användare matat in via terminalen kan skickas från modellen via adaptionsklassen.

Genomförande av tester
För att integrera Modbat, SUT och tillhörande modell används samma angrepssätt som vid provkörning av Modbat. Systemet och modellen kompileras tillsammans via en IDE och exekveras sedan från terminalen tillsammans med Modbat för att genomföra testerna.

Modbat kan effektivt utföra tester och lyckas framförallt utföra serier av tester som annars skulle vara osannolika. Eftersom de utförda stegen slumpmässigt väljs ut från möjliga övergångar i modellens nuvarande tillstånd testas även osannolika serier av inputs, exempelvis att saldot kontrolleras upprepade gånger i rad eller att felaktiga uttagsvärden upprepas gång på gång. Den stokastiska delen gör även att det finns en osäkerhet om hela systemet exekveras i varje uppsättning testfall. Verktyget hanterar detta genom att rapportera andelen övergångar och tillstånd som besökts efter varje exekvering. Ett exempel på hur rapporten för en lyckad körning ser ut finns i figur 2.

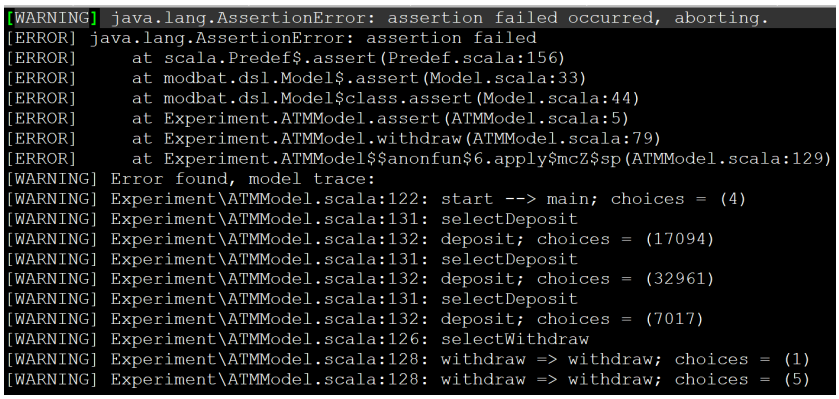
```
[INFO] 200 tests executed, 200 ok, 0 failed.
[INFO] 4 states covered (100 % out of 4),
[INFO] 9 transitions covered (100 % out of 9).
[INFO] Random seed for next test would be: 8cf406f06ed744c
```

Figur 2: Körning utan identifierade fel

```
[INFO] 25 tests executed, 10 ok, 15 failed.
[INFO] One type of test failure:
[INFO] 1) java.lang.AssertionError: assertion failed at withdraw => withdraw:
[INFO]    d0b7c238d65d5a 67a87f6219a400f6 47dbc6da0043aa84 702c77452ed56183 1886
27e921720658 4c346ef20c58395e 3e95ed760fc1adb8 23f57e72e7cda06 906810c3431dc0a1
975206334e87b6d 129ead528ea7f66 137a0d6e3ca45e1d 793759819d993b4 38d13d6c0a8aa5d
9 22a62a1d2d8d345a
[INFO] 4 states covered (100 % out of 4),
[INFO] 8 transitions covered (88 % out of 9).
[INFO] Random seed for next test would be: 6819f8c40a592271
```

Figur 3: Körning med identifierade fel

I fall där något testfall misslyckas genereras en logg-fil innehållande anropen som leder till felet. Detta gör att det går att identifiera var ett fel inträffat. Rapporten från en körning där fel identiferas visas i figur 3. Ett exempel på tillhörande logg-fil finns i figur 4.



Figur 4: Exempel på loggfil vid identifierat fel

4.2.7 Genomförande och utfall enligt testplan

Modbat körs i sviter om 200 test per körning med en avbrottssannolikhet på 2%. Över en uppsättning av 30 testomgångar ger detta ett medelvärde på 11.952,5 inputs per omgång (min: 3.894 max: 32.039 sd: 7.403,6). Utfallen enligt testplanen sammanfattas i tabell 3.

Tabell 3: Utfall från modellbaserad testning

ID	Beskrivning	Resultat
KF-1	Auktorisera vid korrekt inloggning	Inga fel identifieras
KF-2	Auktorisera ej vid inkorrekt inloggning	Inga fel identifieras
KF-3	Verifiera att korrekta uttag utförs.	Inga fel identifieras
KF-4	Verifiera att insättningar utförs korrekt.	Inga fel identifieras
KF-5	Verifiera utloggning ur systemet.	Inga fel identifieras
KF-6	Verifiera datalagring	Inga fel identifieras
IF-1	Insättning uppdaterar fel saldo	Infört fel identifieras
IF-2	Fel i sedelräknaren	Infört fel identifieras ej
IF-3	Fel i konvertering mellan cent/dollar	Infört fel identifieras

Alla steg för att verifiera att SUT och Modbat är integrerade och fungerar tillsammans (KF-1 - KF-6) genomförs utan identifierade fel. När ett fel introduceras som leder till att fel saldo uppdateras (IF-1) lyckas Modbat identifiera detta med hjälp av modellen då den parallella kontrollräkningen av saldona i modellen inte överensstämmer med saldot i SUT. På samma sätt identifieras fel i konvertering mellan dollar och cent vid inmatning av sedlar (IF-3). Fel i nedräkning av antal sedlar (IF-2) identifieras däremot inte. Anledningen att IF-2 inte identifieras är att modellen inte innefattar modulen som hanterar sedelmängden.

Inledningsvis rapporterar Modbat fel redan innan något avsiktligt fel från testplanen införts i SUT. Verktyget och tillhörande rapporter används då för att identifiera en bug vi råkat införa vid anpassningen av systemet för att implementera adaptionslagret. Felet är

en specifik uppsättning inputs som under vissa förutsättningar leder till att systemet tolkar indata på ett felaktigt sätt. Med hjälp av fel-loggen kan de specifika förutsättningarna identifieras och adaptionslagret korrigeras.

5 Diskussion

5.1 Svar på forskningsfrågor

RQ1 - Vilka tecken på användning av MBT inom industrin kan utläsas av publicerade forskningsrapporter?
Endast ett fåtal praktiska exempel på MBT som testmetod i industrin identifieras. Totalt sju forskningsrapporter kvarstår efter fulltextgranskningen. Två av dessa är första- och andrahands rapporter på att MBT används som testmetod [16][17]. Resterande är studier där författarna varit med och arbetat med MBT [18][19][20][21][22].

Antalet exempel tyder på att MBT inte har någon större spridning i mjukvaruindustrin.

RQ2 - Vilka svårigheter och hinder för att använda MBT inom industrin finns beskrivna i forskningsrapporterna från RQ1 och vilka svårigheter och hinder kan hittas i en utforskande fallstudie?
De hinder som identifieras i litteraturstudien är: modelleringsfel och kravet på att modellen ska vara korrekt för att testerna faktiska ska verifiera SUT [18][19], tidsåtgång och svårigheter med implementation av adaptionslagret [19][20] samt verktygsproblem i form av dyra proprietära lösningar eller verktyg där modellerna saknar kompatibilitet med andra verktyg [18].

Även risken för tillståndsexplosioner där mängden testfall blir ohanterbar pekas ut [19]. Svårigheter med hantering av dataunderlag för datadrivna testfall beskrivs av Vieira m.fl. [21].

Ett icke-tekniskt hinder som lyfts fram av Entin m.fl. är svårigheten att få med kollegor och ledning vid införande av MBT. De lyfter vidare fram versionshantering av modeller som ett eventuellt problem då rätt version av modellen behöver matchas med systemet och övriga artefakter för att kunna reproducera fel [18].

Den utforskande fallstudien bekräftar tre huvudsakliga utmaningar som identifieras i litteraturstudien. Den första utmaningen rör att konstruera modellen så att den korrekt representerar systemets tilltänkta beteende.

Den andra utmaningen är verktygen som används för MBT. Upplevelsen av verktyget inför och under utförandet av fallstudien är att det fungerar för att testa men kräver inledande arbete för att både utvärdera och nå stadiet där de första testerna kan utföras. Detta kan sammanfattas som att användbarheten upplevs som förhållandevis låg vilket gör startsträckan onödigt lång.

Den sista utmaningen är att skriva adaptionslagret så att testfallen kan exekveras direkt via modellen. Detta steg är tidskrävande eftersom alla relevanta delar av systemet behöver exponeras för verktyget och kan bli väldigt komplicerat. Det finns även en risk att SUT påverkas under implementation av adaptionslagret vilket i sin tur kan påverka utfallen av testerna.

Hypotes
Utifrån svaren på RQ 1 och RQ 2 kan hypotesen att MBT primärt är av akademiskt intresse, och att en uppsättning hinder för bredare användning kvarstår inte anses falsifierad. Hypotesen stämmer i bemärkelsen att det inte kan påvisas någon bredare användning av MBT inom industrin men det finns enstaka exempel.

5.2 Resultat i relation till tidigare forskning

Utting m.fl. drog 2012 slutsatsen att MBT är redo för storskalig lansering vilket vi inte hittar några tecken på att ha hänt [5]. Fåtalet reella tillämpningar av MBT som identifieras tyder på att MBT inte används i den utsträckning som motsvarar den akademiska litteraturen inom området. Precis som Khan m.fl. påpekar finns det många exempel på fallstudier där MBT ser ut att användas i ett verkligt sammanhang [12]. Dessa är i de flesta fall exempel där författarna har en koppling till organisationen som äger systemet, alternativt lånar ett system, och innebär utvärdering av MBT snarare än tillämpning av MBT som testmetod.

Detta leder till frågor kring varför inte fler exempel på där MBT tillämpas som testmetod kunde identifieras. Ett skäl kan vara att industrin saknar tillräcklig anledning att byta angreppssätt och hantera den omställning och barnsjukdomar som medföljer. Santos m.fl. drar slutsatsen att forskare är mer intresserade av nya metoder och verktyg medan verksamma inom industrin är mer intresserade av förbättring av befintliga arbetssätt [13]. Detta stämmer med slutsatserna kring människornas roll vid införandet av MBT och vikten av att kollegor och ledning är med, som Entin m.fl. tar upp. Om intresset är lågt är det svårt för en metod att få genomslag [18]. Även Weißleder och Schlingloff konstaterar att det är svårt att hitta tid och pengar för nya arbetssätt [4]. Villalobos-Arias m.fl. drar slutsatsen att bristande tillgänglighet hos MBT är det största hindret [7].

De tre eventuella problemområden vi identifierar i den utforskande fallstudien är i stort i linje med tidigare forskning. MBT-verktyget fungerar men är svåra att komma igång med. Detta påpekar även Marijan m.fl. och argumenterar för att verktygen behöver bli mer användarvänliga och tillåtande [10]. Weißleder och Schlingloff tar upp osäkerheten kring hur väl verktygen fungerar vid större industriella tillämpningar [4]. Entin m.fl instämmer i kritiken av verktygen och menar att kommersiella verktyg är dyra och att befintliga verktyg inte är tillräckligt kompatibla med varandra i fall där modellen eller verktyget behöver bytas ut [18].

Däremot anser vi inte att detta är det största problemet i ett längre perspektiv och kan variera beroende på verktyg. När verktyget väl är igång fungerar det utan problem. Istället upplever vi att integrationen mellan modellen och SUT som ett större problem. Ganesan m.fl. beskriver fel vid implementation av adaptionslagret som ett hinder [19]. Wendland m.fl. pekar ut tidsåtgången för att få adaptionslagret att fungera som ett problem [20]. Vidare riskerar ändringar i SUT att introducera nya buggar och kan därmed påverka de delar som faktiskt ska verifieras.

Det sista och svåraste problemområdet vi identifierar gäller abstraktionen som sker via modellen. Det faktum att flera av testerna utförs utan att felet med sedelräknaren identifieras visar tydligt hur känsligt MBT är för felaktiga modeller. I vårt fall kan utelämnandet

delvis förklaras av avsaknaden på en kravspecifikation och övriga artefakter som vanligtvis används för att konstruera modellen. Detta stämmer med vad Ganesan m.fl. beskriver som problem gällande modellering av system med odokumenterade krav [19]. Både Entin m.fl. och Ganesan m.fl. rapporterar exempel på modelleringsfel under MBT-arbetet [18][19].

Enkelheten att utföra en stor mängd tester när modellen och systemet var integrerade indikerar fördelar med MBT som testmetod. Det är även enkelt att exekvera en stor mängd automatiskt genererade testfall, frågan hur stor andel av dessa som faktiskt är värdefulla är däremot inte besvarad.

Under litteraturgenomgången observerade vi även att några författargrupper förekommer upprepade gånger i olika konstellationer. Dessa grupper står bakom en stor andel av de publicerade verken vilket väcker frågor kring huruvida intresset för MBT är särskilt utspritt utanför kretsen som forskar på utveckling av metoden.

5.3 Metoddiskussion

Metoden att leta i akademisk litteratur efter icke-akademiska tillämpningar går att ifrågasätta. Det är möjligt att det finns exempel på tillämpning av MBT i industrin som är publicerad i andra kanaler. Alternativt kan användningen vara opublicerad. Vidare är det i vissa forskningsrapporter inte självklart om tillämpningen är primärt fokuserad på att testa mjukvara eller validera MBT. Till följd av detta påverkar tolkningen av inklusionskriterierna vilka studier som inkluderas.

Fallstudien är begränsad och är inte tagen från ett reellt mjukvaruprojekt i produktion. Detta gör att MBT processen inte är fullständig. Exempelvis finns inga funktionella krav eller andra artefakter som utgör grunden för modellen. Detta gör att modellen inte blir lika tydligt kravstyrd och kan innehålla fel som exempelvis fallet när sedelräknaren inte modelleras. Vidare går det inte heller att stöta på problem som större system kan ha, exempelvis en ohanterbar mängd testfall som beskrivs av Ganesan m.fl. [19]. Trots studiens omfattning samt vår begränsade erfarenhet hittar vi flera hinder som beskrivs i litteraturen.

Generaliserbarheten av resultatet från den utforskande fallstudien är i sig låg. Det är exempelvis inte säkert att de hinder som identifieras hade varit de samma om exempelvis ett annat verktyg eller SUT använts. Vidare är kommersiella och proprietära verktyg inte inkluderade i studien vilket gör att kritiken av verktyg inte nödvändigtvis gäller för dessa.

5.4 Förslag på framtida forskning

Baserat på den identifierade litteraturen råder det tydlig brist på reella exempel där MBT används i industrin. Frågan är om det är brist på studier eller om det är så att MBT är en relativt ovanlig testmetod. Det mest relevanta skulle vara mer omfattande studier som på ett systematiskt och bredare sätt tar reda på i vilken utsträckning MBT används i praktiken. En naturlig följdfråga är varför eller varför inte metoden används.

En möjlig breddning och utveckling av nuvarande studie skulle kunna vara att testa samma

SUT med flera olika modeller eller flera olika verktyg. Alternativt att flera system testas med samma verktyg för att identifiera de hinder som är gemensamma vid upprepade tillämpningar och de som beror på systemen.

Om MBT används och passar under särskilda omständigheter såsom vid testning av säkerhetskritiska system finns det anledning att undersöka vilka detta är samt vilka egenskaper som gör MBT lämplig som testmetod. Detta skulle kunna förtydliga MBTs roll bland andra testmetoder.

MBT har diskuterats flitigt i litteraturen i åtminstone 20 år utan att få något synligt bredare genomslag i praktiken. För att motivera fortsatt forskning rörande MBT behövs reella exempel på att MBT faktiskt används annars är det svårt att motivera vidare utveckling av metoderna.

6 Slutsats

Under en längre period har en stor mängd litteratur kring modellbaserad testning publicerats. Litteraturen är till största del akademiskt inriktad och MBT som testmetod i industrin verkar inte vara särskilt vanlig. Målet med studien är att identifiera exempel där MBT används som testmetod i industrin samt vilka hinder upplevs under MBT-processen. Detta utforskas med hjälp av ett mixed methods angrepp bestående av en systematisk litteraturstudie fokuserad på industriella tillämpningar av MBT samt en utforskande fallstudie där verktyget Modbat används.

Endast ett fåtal industriella tillämpningar av MBT identifieras i litteraturstudien. Totalt inkluderas sju studier efter fulltextgranskningen. Studierna finns primärt inom mjukvaruindustrin och flygindustrin men innehåller även exempel från hälso- sjukvård och bilindustrin. Den utforskande fallstudien indikerar tre typer av hinder. Det första är mängden arbete med, samt bristande användarvänlighet hos verktygen. Den andra är svårigheten med att skriva ett adaptionslager som integrerar systemet med verktyget och modellen för att göra testfallen körbara. Det sista hindret är det kraftiga beroendet på att modellen utformas korrekt och stämmer med systemets tilltänkta beteende. Dessa tre hinder pekas även ut i verken från litteraturstudien. Vidare pekas bland annat även icke-tekniska svårigheter ut under litteraturstudien i form av att hela arbetsgruppen och ledningen behöver engageras för att införa ett nytt arbetssätt.

Sammanfattningsvis identifieras få exempel där MBT faktiskt används som testmetod i industrin. Vi kan med en begränsad fallstudie och ett enkelt system bekräfta hinder i MBT-processen som även beskrivs i den systematiska litteraturgenomgången. Att påstå att metoden inte används inom industrin är en överdrift men utsträckningen är begränsad. MBT är framförallt ett akademiskt område. Däremot tyder existensen av kommersiella verktyg på att metoden används i någon utsträckning, men avslöjar inte var eller varför.

Referenser

[1] P. Bourque, R. E. Fairley och I. C. Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd. Washington, DC, USA: IEEE Computer Society Press, 2014, ISBN: 0769551661.

[2] B. Choi, M.-J. Escalona och K. Herzig, "Summary of the 14th Edition of the IEEE/ACM Workshop on Automation of Software Test (AST)", *SIGSOFT Softw. Eng. Notes*, årg. 44, nr 3, s. 53, nov. 2019, ISSN: 0163-5948. URL: https://doi.org/10.1145/3356773.3356808.

[3] M. Utting och B. Legeard, *Practical model-based testing. [electronic resource] : a tools approach.* Morgan Kaufmann Publishers Inc., ISBN: 0080466486. URL: https://proxy.mau.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=cat05074a&AN=malmo.b1943148&lang=sv&site=eds-live.

[4] S. Weißleder och H. Schlingloff, "An evaluation of model-based testing in embedded applications", i *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, IEEE, 2014, s. 223–232.

[5] M. Utting, A. Pretschner och B. Legeard, "A taxonomy of model-based testing approaches", *Software Testing, Verification and Reliability*, årg. 22, nr 5, s. 297–312, 2012. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.456. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.456.

[6] A. C. Dias Neto, R. Subramanyan, M. Vieira och G. H. Travassos, "A Survey on Model-Based Testing Approaches: A Systematic Review", i *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, ser. WEASELTech '07, Atlanta, Georgia: Association for Computing Machinery, 2007, 31–36, ISBN: 9781595938800. URL: https://doi.org/10.1145/1353673.1353681.

[7] V.-A. Leonardo, Q.-L. Christan, M. Alexandra och J. Marcelo, "Model-based testing areas, tools and challenges: A tertiary study.", *CLEI Electronic Journal*, nr 1, 2019, ISSN: 0717-5000. URL: https://proxy.mau.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsdoj&AN=edsdoj.28d8470f48fc47dfa610b5ada8005c11&lang=sv&site=eds-live.

[8] R. Yang, G. Li, W. C. Lau, K. Zhang och P. Hu, "Model-Based Security Testing: An Empirical Study on OAuth 2.0 Implementations", i *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16, New York, NY, USA: Association for Computing Machinery, 2016, 651–662, ISBN: 9781450342339. URL: https://doi.org/10.1145/2897845.2897874.

[9] E. Bringmann och A. Krämer, "Model-Based Testing of Automotive Systems", i *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*, ser. ICST '08, USA: IEEE Computer Society, 2008, 485–493, ISBN: 9780769531274. URL: https://doi.org/10.1109/ICST.2008.45.

[10] D. Marijan, "A Review of Two Experiences from Applying Model Based Testing in Practice", i *Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, ser. ISSREW '12, USA: IEEE Computer Society, 2012, 231–236, ISBN: 9780769549286. URL: https://doi.org/10.1109/ISSREW.2012.78.

[11] M. Sarma, P. V. R. Murthy, S. Jell och A. Ulrich, "Model-Based Testing in Industry: A Case Study with Two MBT Tools", i *Proceedings of the 5th Workshop on Automation of Software Test*, ser. AST '10, Cape Town, South Africa: Association for Computing Machinery, 2010, 87–90, ISBN: 9781605589701. URL: https://doi.org/10.1145/1808266.1808279.

[12] M. U. Khan, S. Iftikhar, M. Z. Iqbal och S. Sherin, "Empirical studies omit reporting necessary details: A systematic literature review of reporting quality in model based testing", *Computer Standards & Interfaces*, årg. 55, s. 156 –170, 2018, ISSN: 0920-5489. URL: http://www.sciencedirect.com/science/article/pii/S0920548916302112.

[13] R. E. S. Santos, A. B. Bener, M. T. Baldassarre, C. V. C. Magalhães, J. S. Correia-Neto och F. Q. B. d. Silva, "Mind the Gap: Are Practitioners and Researchers in Software Testing Speaking the Same Language?", i *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*, ser. CESSER-IP '19, Montreal, Quebec, Canada: IEEE Press, 2019, 10–17. URL: https://doi.org/10.1109/CESSER-IP.2019.00010.

[14] Z. Micskei. (2017). Model-based testing (MBT), URL: http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html (hämtad 2020-03-02).

[15] B. J. Oates, *Researching Information Systems and Computing*. Sage Publications Ltd., 2006.

[16] H. Altinger, F. Wotawa och M. Schurius, "Testing Methods Used in the Automotive Industry: Results from a Survey", i *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, ser. JAMAICA 2014, San Jose, CA, USA: Association for Computing Machinery, 2014, 1–6, ISBN: 9781450329330. URL: https://doi.org/10.1145/2631890.2631891.

[17] K. Stobie, "Model Based Testing in Practice at Microsoft", *Electron. Notes Theor. Comput. Sci.*, årg. 111, nr C, 5–12, jan. 2005, ISSN: 1571-0661.

[18] V. Entin, M. Winder, B. Zhang och S. Christmann, "Introducing Model-Based Testing in an Industrial Scrum Project", i *Proceedings of the 7th International Workshop on Automation of Software Test*, ser. AST '12, Zurich, Switzerland: IEEE Press, 2012, 43–49, ISBN: 9781467318228.

[19] D. Ganesan, M. Lindvall, S. Hafsteinsson, R. Cleaveland, S. L. Strege och W. Moleski, "Experience Report: Model-Based Test Automation of a Concurrent Flight

Software Bus", i *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, s. 445–454.

[20] M.-F. Wendland, M. Kranz, C. Hein, T. Ritter och A. García Flaquer, "Model-Based Testing in Legacy Software Modernization: An Experience Report", i *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to Testing Automation*, ser. JAMAICA 2013, Lugano, Switzerland: Association for Computing Machinery, 2013, 35–40, ISBN: 9781450321617. URL: `https://doi.org/10.1145/2489280.2489291`.

[21] M. Vieira, X. Song, G. Matos, S. Storck, R. Tanikella och B. Hasling, "Applying Model-Based Testing to Healthcare Products: Preliminary Experiences", i *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08, Leipzig, Germany: Association for Computing Machinery, 2008, 669–672, ISBN: 9781605580791. URL: `https://doi.org/10.1145/1368088.1368183`.

[22] M. Lindvall, D. Ganesan, R. Árdal och R. E. Wiegand, "Metamorphic Model-Based Testing Applied on NASA DAT: An Experience Report", i *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15, Florence, Italy: IEEE Press, 2015, 129–138.

[23] P. Deitel och H. Deitel, *Java How to Program*, 9th. USA: Prentice Hall Press, 2012, ISBN: 0132575663.

[24] C. Artho och A. Biere, *Modbat*, `https://github.com/cyrille-artho/modbat`.

[25] C. Artho, M. Seidl, Q. Gros, E.-H. Choi, T. Kitamura, A. Mori, R. Ramler och Y. Yamagata, "Model-based testing of stateful APIs with Modbat", i *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2015, s. 858–863.

Appendix

Appendix A: Litteratur som granskas i fulltext

1. S. Mohacsi, M. Felderer and A. Beer, Estimating the Cost and Benefit of Model-Based Testing: A Decision Support Procedure for the Application of Model-Based Testing in Industry, 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, Funchal, 2015, pp. 382-389

2. H. Samih, H. L. Guen, R. Bogusch, M. Acher and B. Baudry, Deriving Usage Model Variants for Model-Based Testing: An Industrial Case Study, 2014 19th International Conference on Engineering of Complex Computer Systems, Tianjin, 2014, pp. 77-80

3. M. Dhingra, A. Arora and R. Ghayal, Achievements and challenges of Model Based Testing in industry, 2012 CSI Sixth International Conference on Software Engineering (CONSEG), Indore, 2012, pp. 1-5

4. C. Rütz and J. Schmaltz, An Experience Report on an Industrial Case-Study about Timed Model-Based Testing with UPPAAL-TRON, 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, Berlin, 2011, pp. 39-46

5. D. Marijan, A Review of Two Experiences from Applying Model Based Testing in Practice, 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops, Dallas, TX, 2012, pp. 231-236

6. A. Kirkici, C. Sahin Gebizli and H. Sözer, Risk-Driven Model-Based Testing of Washing Machine Software: An Industrial Case Study, 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Vasteras, 2018, pp. 398-403

7. S. Wang, S. Ali, T. Yue and M. Liaaen, Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study, 2013 13th International Conference on Quality Software, Najing, 2013, pp. 75-84,

8. A. Marques, F. Ramalho and W. L. Andrade, Comparing Model-Based Testing with Traditional Testing Strategies: An Empirical Study, 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops, Cleveland, OH, 2014, pp. 264-273

9. J. Blom, B. Jonsson and S. Nyström, Industrial Evaluation of Test Suite Generation Strategies for Model-Based Testing, 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Chicago, IL, 2016, pp. 209-218

10. C. Schulze, M. Lindvall, S. Bjorgvinsson and R. Wiegand, Model generation to support model-based testing applied on the NASA DAT Web-application - An experience report, 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), Gaithersbury, MD, 2015, pp. 77-87

11. J. Peltola, S. Sierla, P. Aarnio and K. Koskinen, Industrial evaluation of functional

Model-Based Testing for process control applications using CAEX, 2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA), Cagliari, 2013, pp. 1-8

12. D. Ganesan, M. Lindvall, S. Hafsteinsson, R. Cleaveland, S. L. Strege and W. Moleski, Experience Report: Model-Based Test Automation of a Concurrent Flight Software Bus, 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, 2016, pp. 445-454
13. J. Botella, F. Bouquet, J. Capuron, F. Lebeau, B. Legeard and F. Schadle, Model-Based Testing of Cryptographic Components – Lessons Learned from Experience, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luxembourg, 2013, pp. 192-201
14. T. Kanstrén, O. Puolitaival, V. Rytky, A. Saarela and J. S. Keränen, Experiences in setting up domain-specific model-based testing, 2012 IEEE International Conference on Industrial Technology, Athens, 2012, pp. 319-324
15. M. Lindvall, D. Ganesan, R. Árdal and R. E. Wiegand, Metamorphic Model-Based Testing Applied on NASA DAT – An Experience Report, 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015, pp. 129-138
16. C. Schulze, D. Ganesan, M. Lindvall, D. Mcf Omas and A. Cudmore, Model-based testing of NASA's OSAL API — An experience report, 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), Pasadena, CA, 2013, pp. 300-309
17. S. Wieczorek, V. Kozyura, M. Schur and A. Roth, Practical model-based testing of user scenarios, 2012 IEEE International Conference on Industrial Technology, Athens, 2012, pp. 306-311
18. H. J. Herpel et al., Model based testing of satellite on-board software — An industrial use case, 2016 IEEE Aerospace Conference, Big Sky, MT, 2016, pp. 1-9
19. V. Entin, M. Winder, B. Zhang and S. Christmann, Introducing model-based testing in an industrial scrum project, 2012 7th International Workshop on Automation of Software Test (AST), Zurich, 2012, pp. 43-49
20. S. Weißleder and H. Schlingloff, An Evaluation of Model-Based Testing in Embedded Applications, 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation, Cleveland, OH, 2014, pp. 223-232
21. A. T. Endo and A. Simao, Model-Based Testing of Service-Oriented Applications via State Models, 2011 IEEE International Conference on Services Computing, Washington, DC, 2011, pp. 432-439
22. J. R. Calamé, and J. van de Pol. Applying model-based testing to html rendering engines–a case study. Testing of Software and Communicating Systems. 2008, pp. 250-265.

23. M. Sarma, P. V. R. Murthy, S. Jell, and A. Ulrich. Model-based testing in industry: a case study with two MBT tools. In Proceedings of the 5th Workshop on Automation of Software Test (AST '10). Association for Computing Machinery, New York, NY, USE, 2010, pp. 87–90

24. H. Anders and P. Pettersson. Model-based testing of a wap gateway: an industrial case-study. International Workshop on Parallel and Distributed Methods in Verification. Springer, Berlin, Heidelberg, 2006, pp. 116-131

25. H. Hemmati, L. Briand, A. Arcuri, and S. Ali. An enhanced test case selection approach for model-based testing: an industrial case study. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (FSE '10). Association for Computing Machinery, New York, NY, USA, 2010, pp. 267–276

26. S. Ali, T. Yue, L. Briand, and S. Walawege. A product line modeling and configuration methodology to support model-based testing: an industrial case study. In Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems (MODELS'12). Springer-Verlag, Berlin, Heidelberg, 2012, pp. 726–742

27. T. Ayav, T. Tuglular and F. Belli, Model Based Testing of VHDL Programs, 2015 IEEE 39th Annual Computer Software and Applications Conference, Taichung, 2015, pp. 427-432

28. Marlon Vieira, Xiping Song, Gilberto Matos, Stephan Storck, Rajanikanth Tanikella, and Bill Hasling. 2008. Applying model-based testing to healthcare products: preliminary experiences. In Proceedings of the 30th international conference on Software engineering (ICSE '08). Association for Computing Machinery, New York, NY, USA, 669–672

29. V. Chinnapongse, I. Lee, O. Sokolsky, S. Wang, and P. L. Jones. Model-Based Testing of GUI-Driven Applications. In Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS '09). Springer-Verlag, Berlin, Heidelberg, 2009, pp. 203–214

30. P. Olsen, J. Foederer, and J. Tretmans. . Model-based testing of industrial transformational systems. In Proceedings of the 23rd IFIP WG 6.1 international conference on Testing software and systems (ICTSS'11). Springer-Verlag, Berlin, Heidelberg, 2011, pp. 131–145.

31. Keith Stobie. 2005. Model Based Testing in Practice at Microsoft. Electron. Notes Theor. Comput. Sci. 111, 2005, pp. 5–12.

32. A.C. Dias Neto and G. H. Travassos. Surveying model based testing approaches characterization attributes. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '08). Association for Computing Machinery, New York, NY, USA, 2008, pp. 324–326

33. J. Ernits, R. Roo, J. Jacky, and M. Veanes. Model-Based Testing of Web Applications Using NModel. In Proceedings of the 21st IFIP WG 6.1 International Conference

on Testing of Software and Communication Systems and 9th International FATES Workshop (TESTCOM '09/FATES '09). Springer-Verlag, Berlin, Heidelberg, 2009, pp. 211–216

34. H. Sözer and C. zAhin Gebizli. Model-based testing of digital TVs: an industry-as-laboratory approach. Software Quality Journal 25, 4, 2017, 1185–1202

35. C. Poncelet and F. Jacquemard. Model based testing of an interactive music system. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). Association for Computing Machinery, New York, NY, USA, 2015, pp. 1759–1764

36. M. Markthaler et al., Improving Model-Based Testing in Automotive Software Engineering, 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), Gothenburg, 2018, pp. 172-180.

37. C. Schulze, D. Ganesan, M. Lindvall, R. Cleaveland, and D. Goldman. Assessing model-based testing: an empirical study conducted in industry. In Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014). Association for Computing Machinery, New York, NY, USA, 2014, pp. 135–144

38. M-F Wendland, M. Kranz, C. Hein, T. Ritter, and A. García Flaquer. Model-based testing in legacy software modernization: an experience report. In Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA 2013). Association for Computing Machinery, New York, NY, USA, 2013, pp. 35–40

39. C. Mühlbacher, G. Steinbauer, S. Gspandl, and M. Reip. Model-Based Testing of an Industrial Multi-Robot Navigation System. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS '17). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2017, pp. 1652–1654

40. S. Weißleder. Influencing Factors in Model-Based Testing with UML State Machines: Report on an Industrial Cooperation. In Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS '09). Springer-Verlag, Berlin, Heidelberg, 2009, pp. 211–225

41. V. A. De Santiago Júnior and N. L. Vijaykumar. Generating model-based test cases from natural language requirements for space application software. Software Quality Journal 20, 1 (March 2012), 2012, pp. 77–143

42. S. Herbold, P. Harms, and J. Grabowski. Combining usage-based and model-based testing for service-oriented architectures in the industrial practice. Int. J. Softw. Tools Technol. Transf. 19, 3 (June 2017), 2017, pp. 309–324

43. P. B. Lakey. Model-based specification and testing applied to the Ground-Based Midcourse Defense (GMD) system: an industry report. In Proceedings of the 1st

international workshop on Advances in model-based testing (A-MOST '05). Association for Computing Machinery, New York, NY, USA, 2005, pp. 1–7

44. F. Bouquet, E. Jaffuel, B. Legeard, F. Peureux, and M. Utting. Requirements traceability in automated test generation: application to smart card software validation. In Proceedings of the 1st international workshop on Advances in model-based testing (A-MOST '05). Association for Computing Machinery, New York, NY, USA, 2005, pp. 1–7

45. H. Altinger, F. Wotawa, and M. Schurius. Testing methods used in the automotive industry: results from a survey. In Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing (JAMAICA 2014). Association for Computing Machinery, New York, NY, USA, 2014, pp. 1–6

Appendix B: Scala-modell över SUT

Modelldefinitionen finns på de sista tio raderna. Resterande är kontroller av tillstånd och övergångar samt kommunikation via adaptionslagret.

```
package Experiment
import modbat.dsl._

class ATMModel extends Model {
    val a = new Adapter
    a.printConsole = false
    val theATM = new ATM(a)

    var currentAccountNbr = 0
    var modelAvailableBalance = 1000.0
    var modelTotalBalance = 1200.0

    val msgEnterAccountNbr = "Please enter your account number: \n"
    val msgEnterPin = "Enter your PIN:\n"
    val msgMain = "Main menu:\n1 - View my balance\n2 - Withdraw cash\n3 -
        Deposit funds\n4 - Exit\nEnter a choice:\n"
    val msgWithdraw = "Withdrawal Menu:" + "\n1 - $20" + "\n2 - $40" + "\n3 -
        $60" + "\n4 - $100" + "\n5 - $200" + "\n6 - Cancel transaction" +
        "\nChoose a withdrawal amount:\n"
    val msgInvalidWithdrawalSelection = "Invalid selection. Try again.\n"
    val msgDeposit = "Please enter a deposit amount in " + "CENTS (or 0 to
        cancel): \n"

    def authorize(): Boolean = {
        val inp = choose(0,10)/9
        val accountNum = 12345
        val pin = 54321

        theATM.simulateInput(accountNum + inp)
        assert(a.IOLog.getFirst() == msgEnterPin)
        theATM.simulateInput(pin)
        if(inp == 1){
            assert(a.IOLog.getFirst() == msgEnterAccountNbr)
            return false
        } else {
            assert(a.IOLog.getFirst() == msgMain)
            currentAccountNbr = accountNum
            return true
        }
    }

    def checkBalance() {
        theATM.simulateInput(1)
        var tmpAccount = theATM.bankDatabase.getAccount(currentAccountNbr)

        assert(tmpAccount.availableBalance == modelAvailableBalance)
```

```
    assert(tmpAccount.totalBalance == modelTotalBalance)
}

def selectWithdraw() {
    theATM.simulateInput(2)
    assert(a.IOLog.getFirst() == msgWithdraw)
}

def withdraw() : Boolean = {
    var selectAmount = choose(0,10)
    var amount = 0

    selectAmount match {
    case 1 => amount = 20
    case 2 => amount = 40
    case 3 => amount = 60
    case 4 => amount = 100
    case 5 => amount = 200
    case _ => amount = 0
}

theATM.simulateInput(selectAmount)
var tmpAccount = theATM.bankDatabase.getAccount(currentAccountNbr)

if((selectAmount < 7) && (selectAmount > 0)) { //Valid selection
    if(amount <= modelAvailableBalance) { //Funds ok
        modelAvailableBalance -= amount
        modelTotalBalance -= amount

        assert(tmpAccount.availableBalance == modelAvailableBalance)
        assert(tmpAccount.totalBalance == modelTotalBalance)

        return true
    } else { //Insufficient funds
        assert(tmpAccount.availableBalance < amount)
        return false
    }
    return true;
} else { //Invalid selection
    assert(a.IOLog.getFirst() == msgWithdraw)
    assert(a.IOLog.get(1) == msgInvalidWithdrawalSelection)
    return false
    }
}

def selectDeposit(): Unit = {
    theATM.simulateInput(3)
    assert(a.IOLog.getFirst() == msgDeposit)
}

def deposit(): Unit = {
    val amount = choose(0, 100000)
```

```
        theATM.simulateInput(amount)

        if(amount > 0) {
            modelTotalBalance += amount/100
        }

        var tmpAccount = theATM.bankDatabase.getAccount(currentAccountNbr)
        assert(tmpAccount.availableBalance == modelAvailableBalance)
        assert(tmpAccount.totalBalance == modelTotalBalance)
    }

    def exit(): Unit = {
        theATM.simulateInput(4)

        assert(a.IOLog.getFirst == msgEnterAccountNbr)
        assert(theATM.currentAccountNumber == 0)
    }

    "start" -> "start" := {
    } nextIf({() => authorize() } -> "main")

    "main" -> "main" := checkBalance()
    "main" -> "withdraw" := selectWithdraw()

    "withdraw" -> "withdraw" := {
    } nextIf({() => withdraw() } -> "main")

    "main" -> "deposit" := selectDeposit()
    "deposit" -> "main" := deposit()

    "main" -> "start" := exit()
}
```