DEGREE PROJECT, IN COMPUTER SCIENCE , SECOND LEVEL
STOCKHOLM, SWEDEN 2014

Design and realisation of an automated software testing system utilizing virtual machines

DESIGN OCH REALISATION AV ETT AUTOMATISERAT MJUKVARUTESTNINGSSYSTEM MED VIRTUELLA MASKINER

MARKUS BYSTRÖM

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

DESIGN OCH REALISATION AV ETT AUTOMATISERAT MJUKVARUTESTNINGSSYSTEM MED VIRTUELLA MASKINER

22 mars 2014

Författare:
Markus Byström
maby@kth.se

Handledare, KTH:
Olov Engwall

Examinator KTH:
Olle Bälter

Uppdragsgivare:
Scania

Handledare Scania:
Carl Blumenthal

English Title:
Design and realisation of an automated software testing system utilizing virtual machines

Sammanfattning

Modern mjukvara körs ofta i många olika miljöer vilket ställer höga krav på testning och kvalitetssäkring. Mjukvara som löpande förvaltas måste testas regelbundet för att säkerställa kompatibiliteten till de miljöer eller plattformar som den används i. Detta kan knyta upp stora mängder resurser i form av mantimmar för testare och hårdvarutillgänglighet. Genom att testa virtuellt är det möjligt att automatisera stora delar av processen på ett enkelt sätt och i och med det effektivisera testningen.

I det här examensarbetet designades, implementerades och utvärderades ett automatiserat testsystem som utnyttjar virtuella maskiner åt Scania på avdelningen för *Diagnostic Communication and Software Download*, RESC.

Det implementerade testsystemet möjliggjorde dagliga regressions- och installationstester på alla de plattformar som mjukvarukomponenten SCOMM, *Scania Communication Module*, används i. Vissa smärre svårigheter märktes av som att några Windows-versioner uppförde sig på lite olika sätt angående rättigheter och beteende samt att det trots den låga overheaden i de virtuella maskinerna kunde uppstå timingproblem i ett fåtal testfall, vilket ledde till att de stundtals kunde misslyckas.

Genom att parallellt köra tester i olika operativsystem kunde flera tester utföras på kortare tid än förut. Testtillförlitligheten ökade också i och med att alla testkörningar varje gång kunde utgå från samma tillstånd av de virtuella maskinerna. Arbetstiden för installation och underhåll av testmiljön kan minskas i och med att många virtuella maskiner kan samexistera på en fysisk maskin.

Design and realisation of an automated software testing system utilizing virtual machines

Abstract

Modern software is often run in many different environments which puts high demands on testing and quality assurance. Continuous testing of software during the software development cycle is necessary in order to ensure the compatibility between the software and the different environments or platforms in which the software is used. This may require significant resources in the form of man hours for testers and hardware availability. By testing in virtual environments it is possible to automate most of this process in an easy way and thus make testing more efficient.

In this master thesis an automated test system utilizing virtual machines was designed, implemented and evaluated for Scania at its department for *Diagnostic Communication and Software Download*, RESC.

The implemented test system enabled regression and installation testing of the software component SCOMM, *Scania Communication Module*, to be performed on all the supported platforms on a daily basis. Some minor difficulties were experienced such as some versions of the Windows operating system behaving differently regarding to permissions and operation and also that despite the low overhead of the virtual machine some timing issues were noticed in a few test cases which led them to intermittently fail.

By testing software in different operating systems in parallel, it was possible to do more testing in less time than before. Testing reliability was increased due to every test starting from a known state of the virtual machines. The time spent on setup and maintenance of the testing environment can be decreased since multiple virtual machines can co-exist on one physical machine.

Förord

Detta examensarbete har utförts inom Civilingenjörsutbildningen i Datateknik och som del i Masterexamen i Datalogi vid Kungliga Tekniska Högskolan i Stockholm. Arbetet har utförts åt Scania CV AB i Södertälje.

Jag vill tacka min handledare Carl Blumenthal på Scania för alla konstruktiva diskussioner och all hjälp med korrekturläsning av denna rapport. Jag vill även tacka min handledare Olov Engwall på KTH för allt stöd jag fått under arbetets gång.

Innehåll

1 Inledning **1**
- 1.1 Bakgrund . . . 1
- 1.2 SCOMM . . . 1
  - 1.2.1 SCOMMs Testramverk . . . 2
  - 1.2.2 Teststrategi av SCOMM . . . 2
- 1.3 Nya användningsmöjligheter . . . 2
- 1.4 Problemställning och mål . . . 3
- 1.5 Kravställning . . . 3
  - 1.5.1 Funktionskrav hos virtualiseringsmjukvara . . . 3

2 Metod **5**
- 2.1 Inledande undersökning . . . 5
  - 2.1.1 Litteraturstudie . . . 5
  - 2.1.2 Nuvarande testramverk för SCOMM . . . 5
  - 2.1.3 Virtualiseringsmjukvaror . . . 5
- 2.2 Systemdesign . . . 5
- 2.3 Implementation . . . 6
- 2.4 Utvärdering . . . 6
  - 2.4.1 Prestanda . . . 6
  - 2.4.2 Hårdvaruutnyttjande . . . 6
  - 2.4.3 Tid . . . 6
  - 2.4.4 Användarvänlighet . . . 7
  - 2.4.5 Automatisering . . . 7

3 Teoretisk Bakgrund **8**
- 3.1 Mjukvarutestning . . . 8
- 3.2 Testnivåer . . . 8
  - 3.2.1 Enhetstestning . . . 8
  - 3.2.2 Integrationstestning . . . 9
  - 3.2.3 Systemtestning . . . 9
- 3.3 Mål med mjukvarutestning . . . 9
  - 3.3.1 Acceptanstestning . . . 9
  - 3.3.2 Installationstestning . . . 9
  - 3.3.3 Kapacitetstest . . . 10
  - 3.3.4 Regressionstestning . . . 10
- 3.4 Virtuella Maskiner . . . 10
  - 3.4.1 Introduktion . . . 10
  - 3.4.2 Arkitektur . . . 11
  - 3.4.3 Korrekthet . . . 12
  - 3.4.4 Ögonblicksbilder . . . 12
  - 3.4.5 Prestanda . . . 13
  - 3.4.6 VT-x . . . 13
- 3.5 Mjukvarugränssnitt, API, CLI, GUI . . . 13
- 3.6 Nätverkskommunikation med WCF . . . 14

4 SCOMMs Testramverk **15**

5 Design och implementation **16**
- 5.1 Utvidgning av testhierarkistrukturen i SCOMMs nuvarande testramverk . . . 16
  - 5.1.1 Ny nivå i testhierarkistrukturen inom testramverket . . . 16
  - 5.1.2 Hårdvaruresurser . . . 17
- 5.2 Virtualiseringsmjukvaror . . . 18

5.3 Styrning av virtuella maskiner 20
5.4 Testkoordinatorn 21
5.5 Testklienten 22
5.6 Kommunikation och dataöverföring 22
5.7 Rapportgenerering 23

6 Resultat **25**
6.1 Kravuppfyllnad 25
6.2 Testhårdvara och mjukvara 25
6.3 Användarvänlighetsaspekten 25
6.4 Tidsaspekten 26
6.5 Prestandaaspekten 27
6.5.1 Virtuella maskiner på mekanisk disk 28
6.5.2 Virtuella maskiner på SSD 28
6.5.3 Genomsnittlig testtid per *testjobb* 28
6.5.4 Tid för start och installation i virtuell maskin 29
6.5.5 Undersökning av overhead 30
6.5.6 USB-prestanda i virtuell maskin 31
6.6 Hårdvaruutnyttjandeaspekten 32
6.6.1 Hårdvaruutnyttjande i ursprungligt testramverk 32
6.6.2 Hårdvaruutnyttjande i utvidgat testramverk 32
6.7 Automatiseringsaspekten 33

7 Analys av resultat **34**
7.1 Prestandaaspekten 34
7.1.1 Testkapacitet 34
7.1.2 Testtider för testjobb 34
7.1.3 Märkbara symptom till följd av overhead 35
7.1.4 Kommunikation mot fysisk CAN-buss 35
7.2 Hårdvaruutnyttjandeaspekten 35
7.2.1 Parallellisering av *testjobb* 35
7.2.2 Parallellisering inom testuppsättning 36
7.3 Tidsaspekten 36
7.4 Användarvänlighetsaspekten 36
7.5 Automatiseringsaspekten 36

8 Analys av implementation **38**
8.1 Styrning av virtuella maskiner 38
8.1.1 CLI 38
8.1.2 API 38
8.2 Intermittenta problem vid parallellisering av *testjobb* 38

9 Slutsats **40**

10 Diskussion **41**
10.1 Underhåll av de virtuella maskinerna 41
10.2 Automation med Jenkins 42
10.3 Vilka fel kan upptäckas? 42
10.4 Virtuella maskiners korrekthet 42

11 Hållbarhet och Etik **43**

Referenser **44**

Figurer

1 Illustration av hur testramverket är uppbyggt och används. . . . 15
2 Illustration av TestJobb och dess kapsling 17
3 Illustration av strukturen på resursfilen 17
4 Illustration av strukturen på ett TestJobb 18
5 Illustration av översättningen av kommandon mellan olika CLI . 21
6 Illustration för hur det utvidgade testramverket är uppbyggt och används. 22
7 Illustration för hur kommunikationen går till i det utvidgade testramverket 23
8 XML-kod för beskrivning av ett *testjobb* 25
9 XML-kod för beskrivning av testresurser 26
10 Tidsjämförelse mellan olika nivåer av parallellisering av *testjobb* . 29
11 Genomsnittlig tidsåtgång per *testjobb* för uppsättning av den virtuella testmiljön 30
12 Testtidsjämförelse av *testuppsättning* med och utan USB-beroenden på både fysisk- och virtuell maskin. 31
13 Processoranvändning under körning av en *testuppsättning* utan virtualisering, med USB-licensnyckel 32
14 Processorutnyttjande vid olika nivåer av parallellisering och lagringsmedium för virtuella maskiner 33

Tabeller

1 Listning av virtualiseringsmjukvaror som kan användas i Windowsmiljö och som kan virtualisera Windows-operativsystem . . 19
2 Tidsmätning uppsättning av en virtuell maskin 27
3 Utläsning av data 60 gånger i rad från USB-licensnyckel 31

Förklaringar och nomenklatur

Förkortning	**Betydelse**
Virtuell maskin	En virtuell maskin, VM, kan ses som ett mjukvarulager som körs på en fysiskt maskin. Den virtuella maskinen ger en bild, eller illusion, av en dedikerad fysisk maskin till programvara som körs inuti den. Detta innebär möjlighet att köra ett helt datorsystem i mjukvara.
Värdbaserad virtuell maskin	Eng. *Hosted Virtual Machine*. Virtuell maskin som körs inuti ett existerande operativsystem. Virtualiseringsmjukvaran (VMM) installeras då som en helt vanlig applikation.
Fysisk och Virtuell testning	Uttrycken fysisk- och virtuell testning förekommer löpande i denna rapport. Fysisk testning definieras som att mjukvara testas direkt på fysisk hårdvara och alltså utan någon form av virtualisering inblandad. Virtuell testning definieras som att mjukvara testas i en virtualiserad maskin.
CAN	CAN, *Controller Area Network* eller CAN-bus är en kommunikationsbuss som finns i nästan alla fordon nyare än år 2000. CAN möjliggör informationsutbyte mellan olika styrenheter i och protokollet kännetecknas av dess höga tillförlitlighet.
SCOMM	SCOMM, eller *Scania Communication Module*, är den mjukvarukomponent som gör det möjligt att med en PC kommunicera med styrenheter i Scanias produkter. Se avsnitt 1.2 för ytterligare information om SCOMM.
NAT	NAT eller *Network Address Translation* möjliggör att nätverkstrafik från ett subnät i en privat adressrymd kan dirigeras (eng. *routing*) via ett IP-nummer i en publik adressrymd till andra publika nät.

XSLT	*XSLT, eller Extensible Stylesheet Language Transformations*, är ett språk för att transformera XML-dokument till andra dokument som exempelvis HTML.
HDD	Hard Disk Drive, eller mekanisk hårddisk. Möjliggör datalagring på roterande magnetiska skivor.
SSD	Solid-state drive, datalagringsdisk utan några rörliga delar. Istället för magnetiska skivor använder en SSD flashminnesteknik för att lagra information.

Nomenklatur

I rapporten förkommer löpande hänvisningar till data-objekt och data-klasser. För att förtydliga vilken instans av detta som åsyftas beskrivs data-objekten i *kursiv* stil och data-klasserna utan kursiv stil.

1 Inledning

Då många mjukvaruprojekt har en tendens att växa kontinuerligt med införande av ny funktionalitet och rättningar ställs höga krav på löpande testning för att säkerställa krav och kvalité på mjukvaran. En vanlig metod för att testa att ändringar och nya funktioner inte inför buggar är regressionstestning. Flera testmetoder beskrivs i avsnitt 3.3.4. I många fall är dessa tester specificerade att köras efter kompilering för att snabbt hitta avvikelser i mjukvarans förväntade beteende. Ett problem är dock att mjukvara ofta körs i många olika miljöer såsom olika operativsystem vilket ställer hårdare krav på testningen. Det är t.ex. sällan som slutanvändaren har exakt samma mjukvara installerad på sin dator som utvecklaren. I det här examensarbetet utvecklas och utvärderas ett system som enkelt kan testa mjukvara i olika operativsystem och med olika konfigurationer på en enda dator med hjälp av virtuella maskiner.

1.1 Bakgrund

I moderna fordon har elektroniken en central roll. De flesta av reglersystemen använder någon form av mjukvara för att styra olika funktioner och är i sig beroende av olika sensorer och givare. Vid fel på exempelvis en givare kan styrdonen ofta upptäcka detta och sätta en felkod. Dessa felkoder ger en ledtråd till var felen finns så att de kan åtgärdas.

Scania har behov av att kunna kommunicera med lastbilarna från en PC för att kunna läsa ut driftsdata, felsöka och uppdatera fordonens programvara. För att lösa detta sker löpande utveckling av en kommunikationsmodul kallad *Scania Communication Module*, SCOMM, som fungerar som ett lager mellan fordonets kommunikationsbuss och en PC. Denna modul används sedan både internt på många avdelningar inom Scania och externt av Scanias återförsäljare där det finns behov av att kommunicera med fordonen. Några exempel på användningsområden är t.ex. i verkstadsprogram som mekaniker använder för att felsöka och uppdatera fordonens programvara och i produktionsverktyg för att kunna kommunicera med styrenheter både i testfordon och på bänk. Bänktestning syftar på att enskilda styrenheter testas utanför ett fordon.

Då denna kommunikationsmodul är central för kommunikationen med fordonen är det viktigt att den fungerar som tänkt och uppfyller kraven som ställs på den. Därför är testning av denna modul av stor betydelse. Scania vill utvidga testningen med möjlighet att genomföra mjukvarutestning i virtuella miljöer, d.v.s. möjlighet att köra tester i virtuella maskiner, och på så sätt kunna automatisera testningen av mjukvara på olika operativsystem och med varierande konfigurationer. Detta förväntas leda till bättre kvalitetssäkring av kommunikationsmodulen.

1.2 SCOMM

Kommunikationsmodulen SCOMM är skriven i C++ och C# och används som tidigare nämnts vid kommunikation med Scanias styrenheter och fordon. För att kunna använda all funktionalitet i SCOMM behöver två USB-enheter vara inkopplade i den PC som ska kommunicera med styrenheter i någon av Scanias produkter. Den ena USB-enheten är ett diagnoskommunikationsgränssnitt som fysiskt kopplar ihop en PC med en eller flera styrenheters CAN-bus och den

andra USB-enheten är en licensnyckel som används för att förhindra obehörig manipulation av styrenheterna och även möjliggör spårning av vilken verkstad som gjort vad med en viss styrenhet.

KWP2000 (Keyword Protocol 2000, ISO 14230) och UDS (Unified Diagnostic Services, ISO 14229-1) är standarder för funktionalitet för diagnoskommunikation som kan implementeras i styrenheter. Syftet med dessa standarder är att få ett tydligt gränssnitt för hur diagnoskommunikationen går till på CAN-bussen. Standarderna implementeras i styrenheterna som ett antal mjukvarutjänster som det är möjligt att komma åt via ett diagnoskommunikationsgränssnitt. UDS är en vidareutveckling av KWP2000 och målet på Scania är att ersätta det äldre KWP2000 med UDS i nya system.

SCOMMs uppgift är att skapa en högre abstraktionsnivå för dessa protokoll på CAN vilket gör det möjligt för annan mjukvara genom SCOMMs API att läsa och skriva data till styrenheter på CAN-bussen utan att behöva hantera CAN-kommunikationen på låg nivå. SCOMM omvandlar även data från CAN-bussen till förståeliga enheter upp till övre lager / applikationer. Dessa dataomvandlingar finns specificerade i XML-filer där specifika styrenheter har definierade skalningar för dess parametrar och signaler.

1.2.1 SCOMMs Testramverk

Ett testramverk finns i dagsläget och används för kontinuerlig testning av SCOMM. Testramverket är skrivet i C# och anropar SCOMM genom dess .Net API. Testramverket gör det möjligt att köra olika funktioner i SCOMM och jämföra returnerade resultat mot förväntade. Det finns även möjlighet att använda inspelad diagnoskommunikation för att kunna testa SCOMM mot styrenheter som inte är fysiskt tillgängliga. Den högsta nivån på indata till testramverket kallas för *testuppsättning* och innehåller enkelt uttryckt alla tester som körs vid regelbunden testning av SCOMM. Se kapitel 4 för en genomgång av testramverket.

1.2.2 Teststrategi av SCOMM

Teststrategin av SCOMM är att främst med automatiserade tester kunna testa maximalt för de resurser som läggs ner på testning. Det förekommer även manuella tester i t.ex. helbil för att kvalitetssäkra nya funktioner. Regressionstestning ska säkerställa att gammal historisk funktionalitet bibehålls. Tester ska i normala fall alltid automatiseras i SCOMMs testramverk och regressionstestet på systemnivå ska köras varje natt på SCOMMs huvudspår i versionshanteringssystemet Perforce. Kontinuerlig kvalitetssäkring ska genomföras och strävan ska vara att de automatiserade testen täcker in alla nya förändringar av produkten.

1.3 Nya användningsmöjligheter

Systemet som utvecklas i detta examensarbete är tänkt att utvidga nuvarande regressionstest med kontinuerlig testning i de olika operativsystem där SCOMM ska kunna användas. Ett möjligt tillvägagångssätt är att över natten köra igenom testerna på de berörda operativsystemen för att säkerställa att inga

ändringar i kod eller drivrutiner påverkat funktionaliteten och stabiliteten av SCOMM.

1.4 Problemställning och mål

Rapporten ämnar utreda olika lösningsalternativ för att utvidga SCOMMs testramverk med virtuell testning och därefter utvärdera en vald och implementerad lösning med avseende på nedanstående punkter.

Utvärderingen av den valda lösningen ska ta hänsyn till följande aspekter:

- Tid: Är det möjligt att vinna tid på att testa virtuellt?
- Prestanda: Hur mycket skiljer prestandan mellan fysisk och virtuell testning? Är det möjligt att öka prestandan genom parallellisering?
- Hårdvaruutnyttjande: Är det möjligt att öka utnyttjandet av hårdvaran?
- Automatisering: Till vilken nivå är det möjligt att automatisera testningen?
- Användarvänlighet: Är det enkelt att underhålla testsystemet?
- Kravuppfyllnad: Uppfyller lösningen alla krav i avsnitt 1.5?

1.5 Kravställning

Den valda lösningen för virtuell testning av SCOMM behöver uppfylla följande krav. Det ska gå att:

- i konfigurationsfil specificera vilka *testuppsättningar* som ska köras i olika virtuella maskiner.
- i konfigurationsfil specificera hur SCOMM ska installeras i virtuell maskin samt förväntat resultat av installationen.
- i konfigurationsfil specificera vilken USB-ansluten hårdvara som ska vara tillgänglig i en virtuell maskin innan test påbörjas.
- från kommandorad starta test på flera virtuella maskiner med olika SCOMM-installationer och ansluten hårdvara.
- få testrapporter sammanställda med testdata och resultat.
- integrera den virtuella testningen till nuvarande testramverk.
- ha all programkod implementerad i C#.

1.5.1 Funktionskrav hos virtualiseringsmjukvara

Det första kravet är att virtualiseringsmjukvaran ska gå att köra i Windows på x86- och x64-plattformen och ska klara av att virtualisera 32- och 64-bitars Windows-operativsystem från Windows XP och framåt. Anledningen till att

just dessa Windows-versioner ska stödjas är att det är de som används av slutanvändarna av SCOMM.

Det andra kravet är att det ska finnas något slags API från vilket det är möjligt att programmeringsmässigt styra de virtuella maskinerna från ett fristående program. Detta är nödvändigt eftersom de virtuella maskinerna måste kunna startas, stoppas, återställas, med mera, per automatik.

Det tredje kravet är att det ska finnas stöd för att komma åt USB-enheter, som är monterade på värden, i de virtuella maskinerna. Det ska också finnas möjlighet att programmeringsmässigt montera och demontera dessa USB-enheter i de virtuella maskinerna. Anledningen till att det ska vara möjligt att från mjukvaran montera och demontera USB-enheterna är för att kunna automatisera testningen av olika hårdvarukonfigurationer. Denna automation innebär att de USB-enheter som en virtuell maskin ska ha tillgång till ska kunna specificeras i en konfigurationsfil. USB-enheternas funktion förklaras i avsnitt 1.2.

Det fjärde kravet är att det ska finnas stöd för så kallade ögonblicksbilder (eng. *snapshots*, beskrivs närmare i avsnitt 3.4.4) eftersom det gör det möjligt att spara olika tillstånd på en och samma virtuella maskin. Detta innebär att det går att återställa maskinen till ett känt tillstånd vilket är det utgångsläge som är bäst lämpat för testning.

2 Metod

Metoden som det här examensarbetet har utgått ifrån består av fyra delar. Först gjordes en inledande undersökning för att samla in nödvändig information om nuvarande lösningar och möjligheten som virtuella maskiner innebär. En designdel omfattade framtagandet av hur systemet skulle se ut och fungera och implementeringsdelen realiserade den fastslagna designen. Slutligen utvärderades systemet mot den fastställda problemformuleringen.

2.1 Inledande undersökning

Innan design- och implementationsarbetet inleddes, genomfördes en inledande undersökning som innefattade litteraturstudier, studie av SCOMMs nuvarande testramverk samt utvärdering av lämpliga virtualiseringsmjukvaror.

2.1.1 Litteraturstudie

En litteraturstudie genomfördes i början av arbetet vilken behandlade mjukvarutestning och virtualisering. Litteraturstudien finns dokumenterad i kapitel 3 och gav en introduktion till mjukvarutestning och virtualisering samt en förståelse för hur teknikerna fungerar och kan användas.

2.1.2 Nuvarande testramverk för SCOMM

En undersökning gjordes av det nuvarande testramverket för att klargöra dess funktionalitet och arkitektur. Undersökningen genomfördes dels genom källkodsgranskning och dels genom workshops där handledaren för examensarbetet på Scania gick igenom hur de olika delarna fungerade. Det nuvarande testramverket för SCOMM, vilket används för systemtestning och regressionstestning, är en central del i examensarbetet och det var viktigt att förstå hur det var uppbyggt. I kapitel 4 görs en genomgång av det nuvarande testramverket.

2.1.3 Virtualiseringsmjukvaror

En utvärdering av lämpliga virtualiseringsmjukvaror gjordes i ett tidigt skede för att undersöka vilka möjligheter som fanns för att testa virtuellt utifrån kraven som specificerats i början av arbetet. De virtualiseringsmjukvaror som var mest lämpade testades även praktiskt för att säkerställa deras funktionalitet.

Utvärderingen återfinns i avsnitt 5.2.

2.2 Systemdesign

Systemet designades utifrån erfarenheterna från litteraturstudien och studierna i början av projektet. En arkitektur togs fram som skulle uppfylla målen som satts upp i början av arbetet. Som verktyg användes bland annat design- och modelleringsverktyget Enterprise Architect, i vilket det är möjligt att

skapa klassdiagram och grafer som illustrerar klassberoenden och kopplingar. Workshops hölls även för att diskutera idéer utifrån punkter som arkitekturval och användbarhet. Detaljer kring systemdesignen presenteras i kapitel 5.

2.3 Implementation

Efter designfasen implementerades en prototyp av systemet. Utvecklingsmiljön som användes för detta var Visual Studio 2010. Med VirtualBox vald som första virtualiseringsmjukvara inleddes arbetet med att implementera styrningen av de virtuella maskinerna. Arbetet var iterativt och implementerad funktionalitet utvärderades genom workshops och kunde därefter förbättras för att skapa en tydlig arkitektur och en bra användbarhet. Loggfunktionalitet till fil implementerades för att underlätta utvärderingen av systemet. Loggarna innebar att många testkörningar kunde schemaläggas nattetid och att prestandamätningar kunde göras helt automatiskt. Prototypen som implementerats användes senare i utvärderingssyfte för att jämföra virtuell testning mot fysisk. Detaljer kring implementationen presenteras i kapitel 5.

2.4 Utvärdering

Nedan följer de metoder som använts för att utvärdera aspekterna i avsnitt 1.5.

Den mätdata som tagits fram är baserad på ett medelvärde av minst tre repeterade körningar av det som skulle mätas. Mätningar av körtid gjordes automatiserat över natten då datorn som användes i arbetet inte utförde några andra uppgifter.

2.4.1 Prestanda

För att utvärdera prestandan mättes tiden som ett virtuellt test tog jämfört med ett fysiskt test. Dessutom mättes den genomsnittliga tiden för en testuppsättning vid parallellisering av många testuppsättningar för att utvärdera prestandavinsten vid parallellisering.

2.4.2 Hårdvaruutnyttjande

SCOMMs nuvarande testramverk utnyttjar för närvarande endast ca 20% av processorns kapacitet vid körning av regressionstestet. Detta var den största anledningen till att parallellisering var intressant. För att undersöka hur parallelliseringen skalade med processoranvändningen loggades och sammanställdes den genomsnittliga processoranvändningen under alla testkörningar.

2.4.3 Tid

För att undersöka tidsåtgången för uppsättning av testsystemet mättes tiden det tog att installera och konfigurera en virtuell maskin och att göra nödvändiga ändringar i testsystemets konfigurationsfiler. Denna tid användes sedan som mått vid resonemang kring alternativa lösningar som gav liknande testteckning.

2.4.4 Användarvänlighet

En av utvärderingspunkterna gjordes utifrån ett användarperspektiv där användarens hantering av det nya testsystemet försökte förutses. Användarvänlighet har en tendens att vara subjektiv men har ändå försökts utvärderas genom resonemang kring interaktionen med systemet och komplexiteten i konfigurationsfilerna.

2.4.5 Automatisering

Då regressionstestning av SCOMM startas automatiskt via byggskript utvärderades möjligheten till att starta virtuella tester genom nuvarande byggskript. Detta gjordes genom att testa att ingen användarinteraktion behövdes efter att testet startats.

3 Teoretisk Bakgrund

I det här avsnittet förklaras innebörden av mjukvarutestning där de vanligaste metoderna beskrivs samt att en introduktion till virtuella maskiner, dess arkitektur och funktioner ges.

3.1 Mjukvarutestning

Mjukvarutestning innebär dynamisk utvärdering av skriven programkod genom att undersöka hur mjukvara beter sig vid exekvering när den får specifik indata. Målet med testningen är att hitta defekter och problem så att mjukvaran kan förbättras och kvalitetssäkras [1, s. 5-1].

Utvärderingen görs genom att ett begränsat antal testfall selektivt väljs utifrån en vanligtvis oändlig domän av möjliga testfall. På grund av domänens storlek är det i de allra flesta fall inte möjligt att genomföra uttömmande testning eftersom det skulle ta alldeles för lång tid. Detta gör att valet av testfall kräver omsorg för att uppnå en hög effektivitet i testningen. Ofta ligger någon form av riskanalys som grund för vilka delar av mjukvaran som ska testas och därigenom vilka testfall som väljs [1, s. 5-1].

För varje testfall måste det finnas ett förväntat resultat för att det ska gå att avgöra om testet har lyckats eller misslyckats. Om ett testfall misslyckas är det viktigt att ta reda på, och förstå, varför felet uppkommer så att eventuellt underliggande problem kan identifieras och åtgärdas. Det är orsaken till felet som ska åtgärdas och inte själva symptomet [1, s. 5-1].

Synen på mjukvarutestning har förändrats till att vara mer konstruktiv. Testning ses inte längre som något som görs efter att t.ex. ett system blivit implementerat, utan testningen ses som en pågående process som utförs parallellt med utvecklingsarbetet. Genom att arbeta med testningen samtidigt som mjukvaran utvecklas kan potentiellt dåliga designval undvikas i ett tidigt skede [1, s. 5-1].

3.2 Testnivåer

Det går att testa mjukvara på många olika nivåer. Nedan beskrivs några av de vanligaste. De olika testnivåerna används under olika faser i utvecklings- och underhållsarbetet. Tre testnivåer kan särskilt urskiljas - enhetstestning, integrationstestning och systemtestning. Nedanstående beskrivning av testnivåerna baseras på [1, kap, 5.2].

3.2.1 Enhetstestning

Enhetstestning är testandet av en enskild komponent eller metod isolerat från resten av systemet. Denna testprocess sker vanligtvis med tillgång till utvecklingsverktygen, som t.ex. den integrerade utvecklingsmiljön, och kan utföras av de programmerare som skrivit koden.

3.2.2 Integrationstestning

Integrationstestning innebär testning av det resulterade systemet efter att delar av systemets komponenter har integrerats, det vill säga satts ihop. Det finns tre olika metoder att göra detta på, Top-Down testning, Bottom-Up testning samt Big-Bang testning.

Top-Down och Bottom-Up beskriver i vilken ordning integrationstester genomförs. Top-Down innebär att integrationstestningen börjar med komponenterna högst upp i komponenthierarkin och sedan fortsätter nedåt. I de fall som komponenter i hierarkin har beroenden till komponenter längre ner måste de lägre komponenterna simuleras genom att de byts ut mot ett skelett eller en specialversion för teständamål.

Bottom-Up innebär att integrationstestningen börjar med komponenterna längst ner i komponenthierarkin och sedan fortsätter uppåt.

I Big-Bang monteras alla komponenter ihop på en gång och därefter testas hela systemet direkt. Big-Bang testning kan spara tid men är ofta inte att föredra då färre buggar hittas med denna metod [2].

3.2.3 Systemtestning

Vid systemtestning testas funktionaliteten och beteendet av hela systemet. När systemtestning genomförs bör de flesta problemen redan ha upptäckts under enhets- och integrationstestningen. Systemtestningen används ofta för att testa systemet med avseende på säkerhet, precision, prestanda och tillförlitlighet.

3.3 Mål med mjukvarutestning

Genom att bestämma ett kvantitativt mål med mjukvarutestningen är det möjligt att få bättre kontroll över testprocessen. Testningen kan utifrån målen inriktas mot att verifiera specifika aspekter av mjukvaran. Nedan beskrivs några teman som är vanligt förekommande i litteratur om mjukvarutestning [1][3].

3.3.1 Acceptanstestning

Denna typ av testning har som mål att testa om mjukvaran uppfyller beställarens kravspecifikation. Antingen testar beställaren mjukvaran själv eller så specificerar denne vilka scenarion som ska testas och testningen genomförs av utvecklaren eller annan part.

3.3.2 Installationstestning

Efter acceptanstestningen görs ofta tester på installationsmiljön i de olika miljöerna (till exempel olika operativsystem) där mjukvaran ska köras. Målet med detta test är att verifiera att installationsprogrammet fungerar korrekt och att alla filer placeras på förväntad plats på lagringsmediet etc.

3.3.3 Kapacitetstest

Ett kapacitetstest innebär testning av mjukvara med avseendet att hitta den maximala belastningen som mjukvaran klarar av.

3.3.4 Regressionstestning

Regressionstestning innebär selektivt testande av ett system eller en komponent efter att någon av systemets komponenter har ändrats. Detta görs för att kontrollera att ändringarna inte medfört några oväntade sidoeffekter och att systemet fortfarande beter sig som förväntat.

3.4 Virtuella Maskiner

Virtuella maskiner är inget nytt utan de har funnits sedan 60-talet. Då användes de bland annat till att ge användare tillgång till ”egna maskiner” genom virtualiserade system som kördes centralt på stordatorer. De var populära inom både forsknings- och affärsvärlden [4].

I takt med att datorhårdvara blev billigare och att persondatorn utvecklades minskade behoven av virtuella maskiner och de användes allt mindre [4].

Sedan slutet på 90-talet har intresset ökat igen, inte bara inom server-marknaden utan även på persondatorsidan [4]. Anledningen till detta är att gamla stordatorsystem byts ut mot virtualiserade serverkluster som delas av olika användare och grupper samt att ett viktigt användningsområde av virtualiseringsteknologin är isolationsaspekten som möjliggör att många operativsystem kan köras på samma fysiska hårdvara isolerade från varandra. Detta innebär att hårdvaran kan utnyttjas mer effektivt och i händelse av att ett virtualiserat system slutar fungera kommer resterande system inte att påverkas [6].

3.4.1 Introduktion

En virtuell maskin, VM, är enkelt förklarat ett mjukvarulager som körs på en fysisk maskin. Den virtuella maskinen ger en bild av en dedikerad fysisk maskin till programvara som körs i den. Det innebär att mjukvara som körs i den virtuella maskinen inte ser den underliggande fysiska hårdvaran utan bara den virtualiserade hårdvaran [4].

Det finns flera typer av virtuella maskiner och två vanliga exempel är de som virtualiserar en process och de som virtualiserar ett helt system. Ett exempel på en virtuell maskin i vilken en enskild process körs är Java Virtual Machine. Värdbaserade virtuella maskiner tillhandahåller en komplett miljö som gör att ett operativsystem och många processer kan samexistera. Genom att använda en *Virtual Machine Monitor*, *VMM*, kan flera operativsystem köras samtidigt och vara isolerade från varandra på en och samma fysiska maskin [6].

Innan vi går in på detaljer kan det vara bra att förklara några av begreppen inom virtualisering. Den underliggande plattformen som stödjer en virtuell maskinen

kallas för värd och det som körs i en virtuell maskin kallas för gäst. Den fysiska maskinen som kör virtualiseringsmjukvaran kallas alltså för värd och ett operativsystem som körs virtuellt kallas för gäst [5][6].

3.4.2 Arkitektur

En virtuell maskin fungerar så att den kod som exekveras i den virtuella maskinen, d.v.s. i gästen, exekveras i en inkapslad mjukvarumiljö på värden. Gästen tror att den har exklusiv åtkomst till hårdvaran och kan alltså köras precis som om den körts direkt på fysisk hårdvara. Gästen kan också flyttas mellan olika värdar som använder samma virtualiseringsmjukvara i och med att bilden, eller illusionen, av den dedikerade hårdvaran som gästen ser kommer vara likadan även om den fysiska hårdvaran skiljer mellan de olika värdarna [5][4].

När en gäst exekverar en instruktion fångas den upp i VMM där den tolkas och exekveras av den fysiska processorn. Det enkla sättet att göra detta på är genom så kallad *interpretation*, eller översättning. Varje instruktion översätts till korrekta instruktioner för värd-hårdvarans arkitektur och exekveras därefter. Detta medför dock en stor overhead då det kan gå åt 10 riktiga hårdvaruinstruktioner för varje virtuell instruktion. Ett bättre sätt är att dynamisk översätta block av instruktioner som sedan kan återanvändas om kod exekveras flera gånger i följd. Denna metod kallas *Dynamic binary translation*, dynamisk binär översättning på svenska, och har en hög overhead första gången ett kodstycke översätts men som övervägs av den goda prestandan vid repeterad körning av den översatta koden [6].

En annan metod som är mycket snabb men som ställer vissa krav på hårdvaruarkitekturen är *Trap-and-emulate* metoden. För att förklara dessa krav måste vi gå in lite djupare på en vanligt förekommande hårdvaruskyddsmekanism. En hårdvaruarkitektur har ofta olika skyddsnivåer som programkod körs i. På så sätt är det, enkelt förklarat, möjligt att hindra att ett program som körs på en mindre skyddad nivå kommer åt minne i en högre skyddad nivå på ett otillåtet sätt. Varje processor har också ett antal instruktioner som endast kan exekveras i de mest skyddade nivåerna för att komma åt I/O och ändra inställningar för minneshanteraren etc. Popek och Goldberg (1974) kallade dessa instruktioner för *sensitive instructions*, känsliga instruktioner. Det finns dessutom ett antal instruktioner som ska ge kontrollen till operativsystemet (trap) när dessa exekveras från en lägre skyddad nivå. Popek och Goldberg kallade dessa för *privileged instructions* [7].

Popek och Goldberg försökte 1974 formellt definiera de krav som en hårdvaruarkitektur måste uppfylla för att vara *strikt virtualiserbar*, och på så sätt kunna använda *trap-and-emulate* metoden, och det kan sammanfattas till följande sats: *"For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions."* [8]

En VMM byggd för en strikt virtualiserbar hårdvaruarkitektur kan med *trap-and-emulate* exekvera gästinstruktioner säkert och direkt i hårdvara. Det fungerar så att alla *icke-känsliga* instruktioner exekveras direkt i hårdvara men *känsliga* instruktioner som systemanrop och I/O måste emuleras [9].

x86-arkitekturen, som är den arkitektur som i stort sett används i alla

persondatorer då detta skrivs, uppfyller inte kraven för strikt virtualisering i och med att det finns virtualiseringskänsliga instruktioner som inte orsakar en trap till OS. Om en VM har möjlighet att köra en virtualiseringskänslig instruktion direkt i hårdvara är den inte längre isolerad då den kan exekvera på samma säkerhetsnivå som själva VMM. Då kan t.ex. en VM komma åt hårdvaran direkt vilket inte uppfyller kravet på isolering som Popek och Goldberg kom fram till [9].

3.4.3 Korrekthet

I en korrekthetsstudie av virtualiseringsmjukvarorna QEMU, VirtualBox, VMware och BOCHS, som gjordes av forskare från *University of Udine* och *University of Milano* i Italien upptäcktes att alla virtualiseringsmjukvaror som testades innehöll defekter. Testet genomfördes genom att forskarna körde testfall först i en virtuell maskin och sedan körde samma testfall direkt i hårdvara på samma fysiska dator [10], därefter jämfördes maskinernas tillstånd.

Forskarna tog fram en egenutvecklad linuxkärna (eng. *kernel*) så att det var möjligt att spara tillstånd på alla register samt det fysiska minnet innan och efter att testfallet kördes. De kunde då ladda in det sparade tillståndet som tagits innan testet i den virtuella maskinen, och sedan köra det fysiska testet utifrån detta tillstånd. Slutligen jämfördes de två tillstånden från den virtuella maskinen som tagits efter det virtuella testet med det tillstånd som den fysiska maskinen hade efter det fysiska testet. Om det fanns några skillnader där innebar det att den virtuella maskinen inte emulerade alla instruktioner korrekt [10].

Defekter som hittades i QEMU gjorde att den virtuella maskinen kunde krascha eller att emulatorn fastnade i en oändlig loop[1]. VirtualBox emuleringsmodul bygger på samma modul som i QEMU och forskarna kom fram till att felen som hittades i VirtualBox är en delmängd av dem som hittades i QEMU. Även dessa fel gjorde att VirtualBox kunde krascha [10].

I BOCHS och VMware hittades ett fåtal defekter som medförde att några instruktioner inte emulerades korrekt. Inga av dessa defekter var särskilt allvarliga och problemet som upptäcktes i BOCHS hade redan åtgärdats när artikeln publicerades [10].

3.4.4 Ögonblicksbilder

En ögonblicksbild i t.ex. VMware Workstation sparar nuvarande tillstånd av en virtuell maskin vilket gör det möjligt att gå tillbaka till det tillståndet vid ett senare tillfälle. Detta inkluderar diskens tillstånd, ramminnets tillstånd samt konfigurationsinställningar för den virtuella maskinen som MAC-address och antalet processorkärnor med mera. Ögonblicksbilder är väldigt användbara när det finns behov av att kunna gå tillbaka till ett för maskinen känt tillstånd. Det är möjligt att ta flera ögonblicksbilder av varje virtuell maskin vilket möjliggör sparandet av flera olika tillstånd och konfigurationer per virtuell maskin [11].

Ögonblicksbilder i VMware Workstation ärver av varandra i en linjär process. Det är alltså möjligt att ta en ögonblicksbild och sedan fortsätta använda den

[1] En oändlig loop innebär att viss programkod upprepas hela tiden och programmet ser då ut att ha "stannat". `http://en.wikipedia.org/wiki/Infinite_loop`, avläst: 2013-10-26

virtuella maskinen och därefter ta ännu en ögonblicksbild som då har den första ögonblicksbilden som förälder. På detta sätt går det att från en bas bygga vidare med diverse grenar med olika konfiguration och tillstånd [11].

3.4.5 Prestanda

Då känsliga instruktioner i den virtuella maskinen måste emuleras är det oundvikligt att detta medför en viss prestandaoverhead jämfört med kod som körs direkt i hårdvara. Enligt VMware innebar användandet av en hybridlösning mellan direktexekvering och dynamisk binär översättning att kod som kördes på systemnivå i den virtuella maskinen kunde köras med *prestanda i nivå med direkt exekvering i hårdvara*. Den större delen av koden som körs i användarrymden (eng. *user space*), vilket är det minnessegment där icke-betrodd kod körs, kan exekveras direkt i processorn vilket inte ger någon direkt overhead [9].

3.4.6 VT-x

VT-x är Intels namn på deras hårdvaruassisterade virtualisering. Genom att lägga till stöd för hanteringen av de känsliga instruktionerna i hårdvara kan den virtuella maskinen göras mycket mindre komplex och overheaden som virtualisering normalt innebär minskar. Den virtuella maskinen blir mindre komplex eftersom det inte längre är nödvändigt att i mjukvara hantera de känsliga instruktionerna utan det kan ske direkt i hårdvara [9].

3.5 Mjukvarugränssnitt, API, CLI, GUI

Ett API, eller *Application Programming Interface*, specificerar hur mjukvarukomponenter kan interagera med varandra. Detta kan innebära att funktionalitet från en programkomponent kan användas i en annan programkomponent. Ett API är i praktiken ofta ett bibliotek bestående av rutiner och datastrukturer vilka kan exponeras för andra programkomponenter[2].

Ett CLI, eller *Command Line Interface*, är ett gränssnitt där användaren kan interagera med ett program genom att skicka kommandon till det, ofta i form av text. Det var det primära sättet att interagera med de mest populära operativsystemen på 70- och 80-talet som t.ex. MS-DOS och Unix. En applikation kan efter att den startats låta en användare köra kommandon mot mjukvaran via ett CLI[3].

Ett GUI, eller *Graphical User Interface*, är ett gränssnitt där användaren kan interagera med ett program genom grafiska ikoner och visuella indikatorer. GUI introducerades efter reaktioner angående att inlärningskurvan för CLI var för brant[4].

[2] `http://en.wikipedia.org/wiki/Application_programming_interface`, avläst 2013-09-25

[3] `http://en.wikipedia.org/wiki/Command-line_interface`, avläst: 2013-09-25

[4] `http://en.wikipedia.org/wiki/GUI`, avläst: 2013-09-25

3.6 Nätverkskommunikation med WCF

Windows Communication Foundation, förkortat WCF, är ett ramverk utvecklat av Microsoft och ingår i .Net-ramverket. WCF är tänkt att underlätta skapandet av flexibla server-klient-applikationer samt göra det enkelt att konvertera nuvarande nätverkstjänstslösningar till WCF med bibehållen kompatibilitet [13].

För att komma åt en WCF-nätverkstjänst måste en *slutpunkt* (eng. *endpoint*) definieras. Den består av tre komponenter, en adress, en bindning och ett kontrakt. Dessa tre komponenter definierar var en nätverkstjänst finns, vilken funktionalitet den tillhandahåller samt vilket protokoll som används för att kommunicera med den. Adresskomponenten specificerar var en nätverkstjänst finns och bindningen specificerar vilket protokoll det är som används. Några exempel på bindningar är över *HTTP* och *HTTPS* med eller utan *SOAP*, via *TCP* och över *Pipes*. Detta urval av protokoll gör att det är möjligt att publicera en och samma nätverkstjänst på flera olika protokoll samtidigt vilket är användbart om många olika applikationer med varierande protokollstöd ska använda samma nätverkstjänst [13].

Kontrakten i WCF specificerar vilken funktionalitet en nätverkstjänst tillhandahåller. I tjänstens kontrakt listas de metoder som klienten ska kunna komma åt. I de fall metoderna använder komplexa typer som t.ex. objekt måste ett datakontrakt specificeras för typerna. Detta görs för att WCF ska kunna serialisera den data som ska skickas mellan klient och server. Programmeraren behöver vid användande av WCF inte tänka på hur datan som skickas ska serialiseras eller hur själva kommunikationen mellan server och klient sker utan detta hanteras automatiskt av ramverket [13].

Genom att ha flera kontrakt för en nätverkstjänst är det möjligt att tillgängliggöra olika nivåer av funktionalitet för olika klienter via en och samma nätverkstjänst. En publik webbapplikation ska kanske inte ha tillgång till viss administrativ funktionalitet som en back-end-applikation behöver ha från en nätverkstjänst. För att tillgodose båda applikationernas behov går det att implementera två olika kontrakt till nätverkstjänsten som då tillgängliggör olika gränssnitt till applikationerna [13].

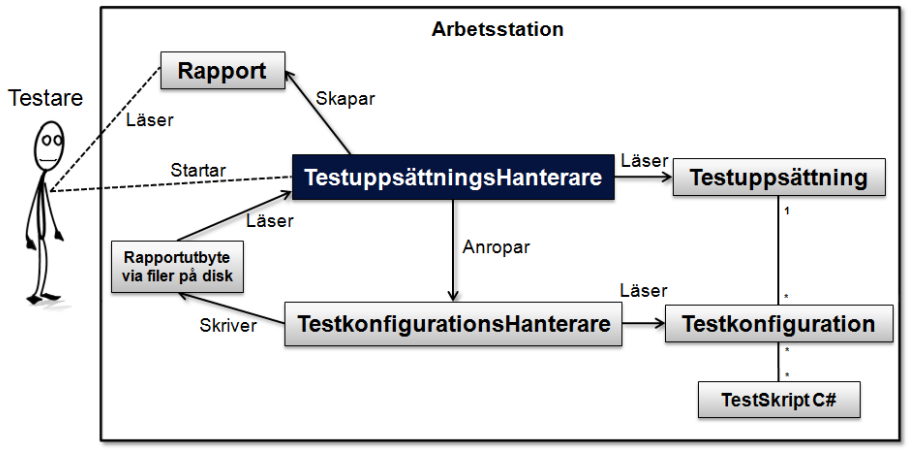
4 SCOMMs Testramverk

SCOMMs nuvarande testramverk använder SCOMM genom dess .Net API och kan då anropa olika funktioner i SCOMM och jämföra returnerade resultat mot förväntade. Det finns möjlighet att använda inspelad diagnoskommunikation från riktiga styrenheter för att kunna testa funktioner i SCOMM utan att ha en fysisk styrenhet tillgänglig.

Testramverket är uppbyggt i olika nivåer i testhierarkistrukturen för att förenkla skapande och återanvändning av tester. Den grundläggande nivån är ett TestSkript vilket är en C#-fil där funktionsanrop mot SCOMM görs och där resultaten av dessa jämförs med med förväntade värden. Inuti TestSkript-filen finns ytterligare nivåer men dessa nämns inte här för enkelhets skull. På nivån över TestSkripten kommer Testkonfiguration vilken kan innehålla ett eller flera TestSkript. En Testkonfiguration kan till exempel innehålla alla regressionstester för kommunikation med en viss version av styrenhet eller testa en speciell del av SCOMM. Översta nivån är Testuppsättning, som innehåller en eller flera Testkonfigurationer. I en Testuppsättning läggs t.ex. alla Testkonfigurationer som i dagsläget används för regressionstestningen av SCOMM. Se figur 1 för en illustration av hur testramverket är uppbyggt. Till höger i figuren illustreras de testhierarkinivåer som är beskrivna ovan. All indata till testramverket, utom TestSkripten, sparas i XML-format på disk.

När testramverket körs genereras en rapport. I denna rapport visas resultaten och de förväntade värdena för funktionsanropen som gjorts mot SCOMM. Om avvikelser finns markeras testet för att visa vad som gått fel. Vid körning av testramverket skrivs rapportfiler i XML-format till disk vid varje avklarat test i de olika nivåerna i testhierarkin. När delar av rapporten ska genereras läses filerna in från disk igen för att sammanställas.

Körningen av en *testuppsättning* kan parallelliseras så att flera *testkonfigurationer* körs samtidigt i separata processer, en per processorkärna.



Figur 1: Illustration av hur testramverket är uppbyggt och används.

5 Design och implementation

I det här kapitlet beskrivs systemdesignen, valet av virtualiseringsmjukvara samt de viktigaste klasserna för det utvidgade testramverket tillsammans med de bakomliggande tankarna och motiveringarna.

5.1 Utvidgning av testhierarkistrukturen i SCOMMs nuvarande testramverk

För att utvidga SCOMMs testramverk till att innefatta funktionalitet för virtuell testning togs en ny nivå i testhierarkistrukturen fram i vilken de virtuella testerna och funktionerna relaterade till detta kunde representeras. Ett av målen med designen var att kunna skriva nya tester och sätta upp nya testmiljöer för virtuell testning utan att behöva ändra i testramverkets källkod. För att uppnå detta mål användes XML som lagringsformat för indata och konfiguration.

5.1.1 Ny nivå i testhierarkistrukturen inom testramverket

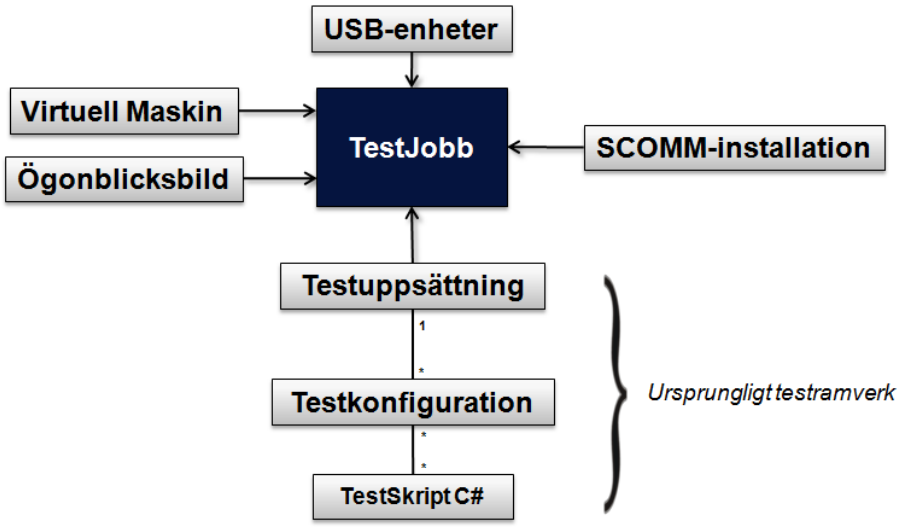
En ny testhierarkinivå skapades inom testramverket för att kapsla in den funktionalitet som tillkommer i det utvidgade testramverket samt länka in den befintliga funktionaliteten. Testhierarkinivån benämns TestJobb och i den specificeras förutsättningarna för ett virtuellt test. Det går att specificera:

- i vilken virtuell maskin ett test ska köras
- vilken ögonblicksbild av den virtuella maskinen som ska användas
- vilka USB-enheter som ska vara tillgängliga i den virtuella maskinen
- vilken SCOMM-installation som ska användas
- vilken testuppsättning som ska köras i den virtuella maskinen

Se figur 2 för en illustration av strukturen för ett TestJobb.

Den virtuella maskinen och USB-enheterna ses som *resurser* på TestJobb-nivån. Alla *testjobb* beskrivs i XML vilket valdes på grund av att det dels är ett enkelt format att läsa, tolka och skriva programmeringsmässigt och dels för att resten av testramverkets testhierarki också beskrivs i XML.

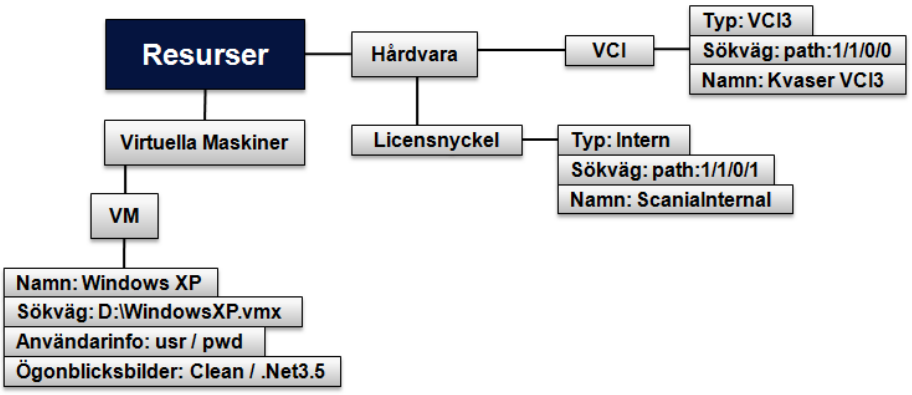
I och med att en ny testhierarkinivå införts måste även en ny rapportnivå införas för att beskriva och sammanfatta tester i den nya testhierarkinivån. Detta beskrivs utförligare i avsnitt 5.7.



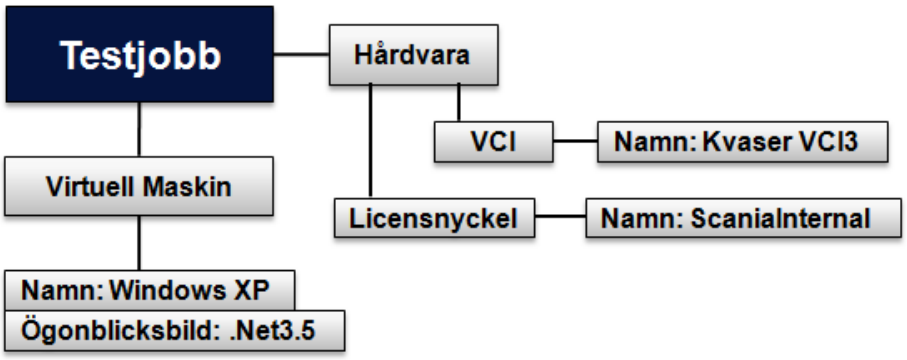
Figur 2: Illustration av TestJobb och dess kapsling

5.1.2 Hårdvaruresurser

För att förenkla användandet av resurser i TestJobb gjordes en separation mellan tillgänglig hårdvara på PCn och de beroenden som ett TestJobb har till tillgänglig hårdvara. Denna separation innebar att mindre information behövde sparas för varje *testjobb* för att beskriva dess resursberoenden. Information som t.ex. sökvägar till filer och hårdvaruadresser till USB-enheter sparas i en resursfil i XML-format. Detta innebär att det, från XML-filen som beskriver ett *testjobb*, är möjligt att referera till resurser i resursfilen med ett namn eller alias. Se figur 3 och 4 för en illustration av hur detta ser ut.



Figur 3: Illustration av strukturen på resursfilen



Figur 4: Illustration av strukturen på ett TestJobb

I och med att det kan finnas många *testjobb* som använder samma resurser minskar denna separation *testjobbens* storlek och komplexitet. Det är även möjligt att återanvända samma *testjobb* på olika PC genom att endast justera den lokala resursfilen. Genom att bara ändra sökvägar och hårdvaruadresser specifika för PCn och använda samma namn eller alias på resurserna i resursfilen kan de ursprungliga referenserna till resurserna i *testjobben* användas. Då denna information finns i en fil på disk är det möjligt att justera resurserna för varje PC som ska använda virtuella tester utan att behöva ändra i själva källkoden.

5.2 Virtualiseringsmjukvaror

Virtualiseringsmjukvara valdes utifrån kraven som specificerats i kapitel 1.5.1.

I tabell 1 listas de virtualiseringsmjukvaror som går att använda under ett Windows-operativsystem och som klarar av Windows-operativsystem som gäst-operativsystem. En kommentar finns om varje virtualiseringsmjukvaras lämplighet för systemet som utvecklats i detta examensarbete. Underlaget till tabellen kommer från Wikipedias sammanställning av virtualiseringsmjukvaror[5].

[5] `http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines`, avläst: 2013-03-21

Virtualiseringsmjukvara	**Kommentar**
Bochs	Endast emulering av hårdvara
Hyper-V Server 2008 R2	Körs direkt på hårdvara vilket är överkurs för det här projektet
Hyper-V Server 2012	Körs direkt på hårdvara vilket är överkurs för det här projektet
iCore Virtual Accounts	Endast stöd för Windows XP
Parallells Workstation	Uppfyller kraven enligt produkt-specifikationen
QEMU	Stödet för många gästoperativsystem osäkert
Simics	Mest lämpad för verifiering av OS eller inbyggda system
VirtualBox	Uppfyller kraven för detta projekt
Virtual PC 2007	Saknar USB-stöd
Windows Virtual PC	Saknar stöd för ögonblicksbilder
Virtuozzo	Lämpad för serverkonsolidering där endast en version av ett OS kan köras
VMware Server 2	Support har upphört på denna produkt
VMware Workstation	Uppfyller kraven för detta projekt

Tabell 1: Listning av virtualiseringsmjukvaror som kan användas i Windowsmiljö och som kan virtualisera Windows-operativsystem

De virtualiseringsmjukvaror som enligt specifikationerna uppfyllde kraven var alltså VirtualBox, VMware Workstation och Parallells Workstation. Av dessa har Virtualbox och VMware Workstation använts i det virtuella testsystemet och dessa virtualiseringsmjukvaror har därför kunnat testas ordentligt. Att VMware Workstation finns tillgängligt hos Scanias IT-leverantör och att VirtualBox är öppen källkod under GPL[6] (eng. *GNU General Public License*) version 2 ledde till att det är dessa som valts för detta projekt.

Det är möjligt att konfigurera de virtuella maskinernas hårdvara i både VirtualBox och VMware Workstation. Antalet processorkärnor och mängden RAM-minne som ska vara tillgängligt i maskinerna går att ställa in. Det går också att lägga till extra enheter som nätverkskort och hårddiskar. Tre vanliga sätt att konfigurera nätverkskorten på är att brygga värdens och gästens nätverkskort, att använda NAT mellan värdens och gästens nätverkskort eller att endast ha ett privat nätverk mellan värd och gäst. Det sista valet innebär att den virtuella maskinen inte kan komma åt företagsnätverket vilket är lämpligt i det här fallet då de virtuella maskinerna som använts i det här arbetet inte är anpassade för Scanias IT-miljö.

Det är dock möjligt att få virtuella maskiner anpassade till Scanias IT-miljö men det bör poängteras att en av anledningarna till att testa på icke-anpassade maskiner är att de bättre representerar hur miljön ser ut hos många av

[6] `https://www.virtualbox.org/`, avläst 2013-10-31

slutanvändarna.

Mängden funktioner som stöds i respektive virtualiseringsmjukvaras *Command-Line Interface*, CLI, varierar mellan de olika virtualiseringsmjukvarorna. I VirtualBox CLI i version 4.2.12 finns i princip all funktionalitet som finns i det grafiska användargränssnitt, vilket är utmärkt.

I VMware Workstation 9 CLI är funktionaliteten inte i nivå med VirtualBox CLI. Vanligt förekommande funktioner som start och stopp av virtuell maskin, skapande och återställande av ögonblicksbilder och filkopiering mellan värd och gäst med mera finns i VMware Workstation CLI. Den funktion som saknades mest för det här arbetet var möjligheten att montera USB-enheter i en virtuell maskin via CLI. Då det var nödvändigt att automatiskt kunna montera USB-enheter i de virtuella maskinerna fick istället konfigurationsfilen för respektive virtuella maskin editeras programmeringsmässigt när VMware användes som virtualiseringsmjukvara. En textrad per USB-enhet behövde läggas till i konfigurationsfilen för att USB-enheterna skulle monteras automatiskt när den virtuella maskinen startades.

De USB-licensnycklar som används för SCOMM har alla samma Vendor ID (VID) och Product ID (PID) och detta skapade problem i VirtualBox när flera likadana USB-licensnycklar var inkopplade till datorn. Problemet som noterades var att det inte gick att montera mer än en av dessa likadana licensnycklar i någon virtuell maskin. Vid försök att montera mer än en licensnyckel gav CLI eller GUI ett felmeddelande om att enheten var upptagen. Detta problem fanns inte med VMware Workstation vilket ledde till att VMware Workstation i fortsättningen användes som primär virtualiseringsmjukvara.

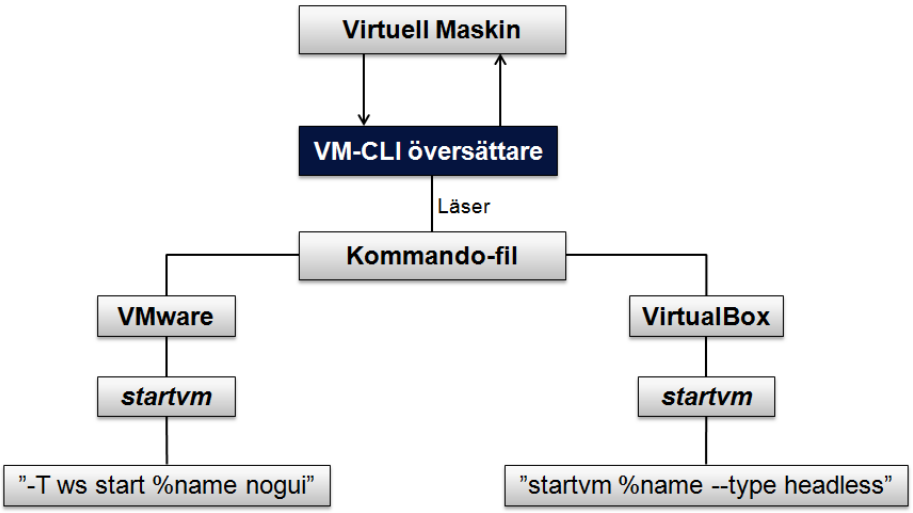
5.3 Styrning av virtuella maskiner

För att kunna automatisera testsystemet måste de virtuella maskinerna kunna styras automatiskt från programkod. Det finns flera olika alternativ på hur detta kan göras, dels grafiskt via ett GUI, dels med text-kommandon via ett CLI och dels direkt från programkod via ett API. Det grafiska gränssnittet faller bort i och med att det är svårare att automatisera än CLI och API. För att minska beroenden till en viss virtualiseringsmjukvara och få en mer allmän lösning valdes en styrningslösning med CLI.

Fördelen med att använda CLI var att det var möjligt att göra en abstraktion mellan olika virtualiseringsmjukvarors CLI och hur dessa anropas ifrån koden. Abstraktionen innebar möjlighet att utifrån ett generiskt kommando i källkoden skapa ett kommando specifikt för en viss virtualiseringsmjukvara. För att göra detta användes det generiska kommando-namnet samt information om i vilken virtualiseringsmjukvara kommandot skulle köras. Det specifika kommandot för varje virtualiseringsmjukvara kunde sedan skapas med hjälp av denna information och en översättningsfil. Fördelen är att källkoden för styrning av de virtuella maskinerna är samma för alla virtualiseringsmjukvaror, och att skillnaden istället hanteras i en översättningsfil som kan editeras utanför utvecklingsmiljön. Om olika virtualiseringsmjukvaror har liknande indata-parametrar till respektive CLI-kommandon är det alltså möjligt att lägga in stöd för flera virtualiseringsmjukvaror utan att göra ändringar i källkoden. Under arbetets gång märktes dock vissa svårigheter med exempelvis hanteringen av USB-enheter vilket innebar att en del av denna varianthantering blev tvungen

att göras i källkoden.

I figur 5 visas en illustration av översättningsstrukturen. Kommandot ***startvm*** är det generiska kommandot som finns implementerat i källkoden för styrningen av de virtuella maskinerna. Detta kommando producerar en textsträng som syns nedanför respektive *startvm*-box i figuren beroende på vilken virtualiseringsmjukvara som används. Textsträngen är argumentet som ska användas till CLI-anropen för att styra den virtuella maskinen.



Figur 5: Illustration av översättningen av kommandon mellan olika CLI

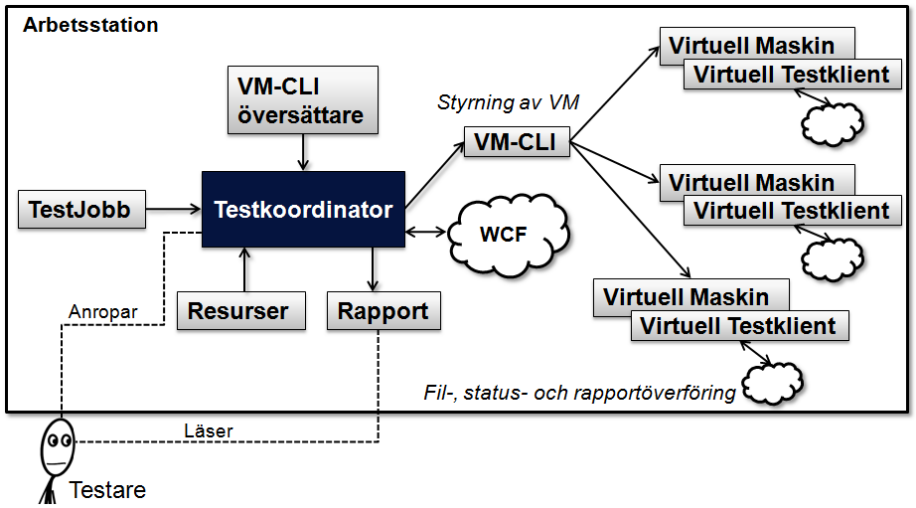
5.4 Testkoordinatorn

Det virtuella testsystemet (illustrerat i figur 6) är uppbyggt kring en server-klient-modell och är skrivet i C# och .Net 3.5. Serverdelen kallas *testkoordinatorn* och det är den som användaren anropar när ett *testjobb* ska köras. Testkoordinatorn körs på den fysiska maskinen och startar, stoppar och återställer de virtuella maskinerna samt monterar och avmonterar USB-enheter.

Då *testjobb* har ett beroende till en viss virtuell maskin och även kan ha hårdvaruberoenden som USB-enheter måste koordinatorn se till att flera *testjobb* som kräver samma resurser inte körs samtidigt. En algoritm går igenom alla *testjobb* och väljer att köra det som har flest beroenden till tillgängliga hårdvaruresurser. När det inte längre går att starta nya *testjobb* får resterande vänta tills resurserna blir lediga.

De virtuella maskinerna styrs genom virtualiseringsmjukvarans CLI, via testkoordinatorn. En separat tråd skapas för varje *testjobb* i testkoordinatorn och dessa trådar kör i sin tur kommandon i virtualiseringsmjukvarans CLI i separata processer. Vid körning av ett kommando väntar *testjobb*-tråden på att CLI-processen ska avslutas för att kunna kontrollera processens returkod som visar om kommandot lyckats eller inte. Om kommandot misslyckats noteras detta i rapporten.

Testkoordinatorn sätter också upp en nätverkstjänst för kommunikation och datautbyte med klienter. Denna tjänst beskrivs i kapitel 5.6.



Figur 6: Illustration för hur det utvidgade testramverket är uppbyggt och används.

5.5 Testklienten

Den virtuella testklienten är den applikation som körs på de virtuella maskinerna. Klienten sköter datakommunikationen mellan Testkoordinatorn och den virtuella maskinen. När testklienten har förberett testmiljön anropas funktionalitet från det ursprungliga testramverket med testuppsättningen, som finns specificerad i *testjobbet*, som indata.

I och med att underhåll av testklienten förväntas ske ingår testklienten inte i den virtuella maskinens ögonblicksbild utan istället flyttas klienten till den virtuella maskinen i början av varje *testjobb*. Om testklienten hade ingått i ögonblicksbilden för de virtuella maskinerna skulle det innebära att ögonblicksbilderna skulle behöva göras om varje gång en ändring i testklienten görs. För att flytta testklienten används virtualiseringsmjukvarans CLI-funktion för att kopiera filer från värd till gäst.

5.6 Kommunikation och dataöverföring

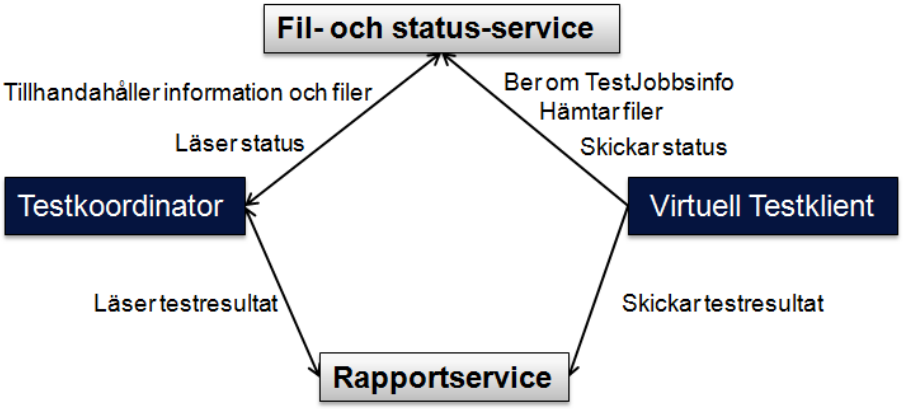
Datautbytet som måste ske mellan testkoordinatorn på den fysiska maskinen och testklienten i den virtuella maskinen under en TestJobb-körning innebar att en överföringsmetod behövde väljas. En nätverksbaserad lösning var önskvärd då det skulle ge flexibilitet i hur data kan utbytas mellan olika komponenter.

För att sköta informationsutbytet mellan testkoordinatorn och testklienten valdes WCF som ramverk för kommunikationen. WCF är ett ramverk för att sköta kommunikation genom nätverkstjänster som tidigare presenterats i avsnitt 3.6. Ramverket valdes främst på grund av enkelheten att skicka över filer och objekt. En annan fördel med WCF var att det var relativt enkelt att uppdatera befintliga komponenter till att använda nätverkstjänster.

Två typer av dataöverföring behövdes för att uppnå kraven med den virtuella testningen. Den ena var möjligheten att överföra information och filer mellan

testkoordinatorn i värden och testklienten i gästen och den andra var möjligheten att överföra rapportobjekt dels mellan olika komponenter i gästen och dels mellan gästen och värden. Se figur 7 för en illustration av hur kommunikationen med de två nätverkstjänsterna går till. Två nätverkstjänster sattes upp i testkoordinatorn och de implementerades som TCP/IP bindningar. Porten som används tilldelas beroende på vilka portar som är lediga på maskinen utifrån ett specificerat intervall. Tilldelningen av kommunikationsport gjordes för att det skulle vara möjligt att köra flera instanser av applikationer som sätter upp nätverkstjänster samtidigt. Adress och port till nätverkstjänsterna skickas sedan med som inparameter till testklienten när den startas från testkoordinatorn.

Då testklienten i den virtuella maskinen startats anropar den testkoordinatorns dataöverföringstjänst för att överföra de program som måste köras för att förbereda testmiljön innan ett test kan startas. Dessa program är t.ex. installationsprogrammet för SCOMM och installationsprogrammet för SCOMMs testramverk. Installationsprogrammet för SCOMM innehåller förutom själva API dll-filerna även drivrutiner för licensnycklar och CAN-hårdvarugränssnitt. Installationsprogrammet för testramverket innehåller testapplikationerna, alla testskript och mallar för rapporterna.



Figur 7: Illustration för hur kommunikationen går till i det utvidgade testramverket

5.7 Rapportgenerering

En ny nivå i testrapporten behövde designas för att kunna beskriva de virtuella testerna och resultaten som de genererat i rapporten. Testrapportdelen för ett *testjobb* har som syfte att beskriva miljön som en *testuppsättning* har körts i. Därför finns följande information med i *testjobb*-rapporten:

- Vilken virtuell maskin och vilken ögonblicksbild som använts.
- Vilka USB-enheter som funnits tillgängliga i den virtuella maskinen.
- Vilka installationer som anropats och resultatet av dem.
- Vilken *testuppsättning* som använts och resultatet av den.

Samtidigt som ett test körs genereras en testrapport. Den är uppbyggd i olika lager där varje lager representerar resultaten för en nivå i testhierarkin.

På högsta nivån sammanfattas testmiljön och de övergripande testresultaten tillsammans med länkar vidare till de lägre nivåerna där alla detaljresultat finns. På så sätt får testaren en snabb överblick av testutfallet och kan sedan välja att visa detaljer för ett specifikt testfall som är av intresse.

I SCOMMs ursprungliga testramverk skrevs XML-filer till disk vid varje avklarat test i de olika nivåerna i testhierarkin. När delar av rapporten sedan skulle genereras lästes filerna in från disk igen för att transformeras till html-filer med hjälp av XSLT. Detta innebar mycket läsning och skrivning till disk. I det utvidgade testramverket skulle det bli omständigt att använda detta lagringssätt eftersom all data måste flyttas ut från gästen till värden senast när ett TestJobb är färdigt i och med att gästens tillstånd kan komma att återställas inför ett nytt TestJobb efter detta. Antingen skulle alla enskilda rapportfiler behöva komprimeras till en enda fil för att sedan föras över till värden via virtualiseringsmjukvarans CLI eller så måste rapporterna läsas in från disk ännu en gång för att sedan serialiseras och skickas via nätverket eller liknande.

Efter initiala tester av WCF beslutades att även implementera WCF-nätverks-tjänster för rapporthanteringen i gästen. Detta innebar att rapportobjekten kunde rapporteras direkt till en nätverksbaserad rapport-tjänst och det blev möjligt att minska läsning och skrivning till disk. I och med att rapport-tjänsten implementerades var det möjligt att skicka rapportobjekt från gästen till värden medan testet pågick vilket innebar att data inte behövde mellanlagras på gästen utan kunde skickas upp till övre lager och hanteras direkt.

6 Resultat

I det här avsnittet presenteras resultaten från utvärderingen av det implementerade systemet. Utvärderingen genomfördes med avseende på aspekterna i problemformuleringen i kapitel 1.4.

6.1 Kravuppfyllnad

Det utvidgade testramverket för SCOMM uppfyller alla de funktionskrav som ställdes upp i projektets början i kapitel 1.5.

6.2 Testhårdvara och mjukvara

Den hårdvara som användes vid utvärderingen var en Dell Optiplex 990 med en fyrkärnig Core i7-2600 på 3.4 GHz med hyperthreading (åtta trådar) samt 16 GB internminne. Lagringsenheterna bestod av en 500 GB mekanisk disk och en 120 GB SSD. Operativsystemet på datorn var Windows 7 x64 Enterprise.

6.3 Användarvänlighetsaspekten

Utvidgningen av testramverket designades med avsikten att det skulle vara enkelt att sätta upp och underhålla testsystemet. Konfigurationsfilerna är överskådliga och ett normalt *testjobb* beskrivs med 9 XML-taggar, se figur 8. En virtuell maskin kan beskrivas med tre XML-taggar och en USB-enhet beskrivs i en XML-tagg, se figur 9.

```
<TestJob name="Regression test on Win 7 x64">
    <VM name="Windows 7 x64" snapshot="Clean"/>
    <TestSetup path="RegressionTestSetup.xml"/>
    <HwDependencies>
        <LicenseContainer name="HASPHLScaniaInternal"/>
        <VCI name="Scania VCI2"/>
    </HwDependencies>
    <ExecutionList>
        <Execution name="ScommInstaller" arguments="" expectedResult="-3"/>
        <Execution name="ScommTestFrameworkInstaller" arguments="" expectedResult="0"/>
    </ExecutionList>
</TestJob>
```

Figur 8: XML-kod för beskrivning av ett *testjobb*

```
<Resources>
    <VMs>
        <VM name="Windows 7 x64" path="D:\VM\Windows 7 x64.vmx" vmm="vmware workstation">
            <logincredentials username="admin" password="admin"/>
            <snapshot name="Clean"/>
        </VM>
    </VMs>
    <Hardware>
        <LicenseContainer ContainerType="HASPHL" USBKeyType="ScaniaInternal"
            name="HASPHLScaniaInternal" identifier="path:1/1/0/6"/>

        <VCI type="VCI2" name="Scania VCI2" identifier="path:1/1/0/0"/>
    </Hardware>
</Resources>
```

Figur 9: XML-kod för beskrivning av testresurser

Användarnamn och lösenord till de virtuella maskinerna valdes att sparas i klartext på grund av att de virtuella maskinerna endast används internt på Scania, att de är isolerade från resten av nätverket och för att detta förenklar hanteringen av konfigurationen.

I och med uppdelningen av testresurser och *testjobbens* resursberoenden till två XML-filer är det väldigt enkelt att sätta upp testmiljön på nya fysiska maskiner genom att återanvända befintlig konfiguration med endast ett fåtal konfigurationsändringar. Redan installerade och konfigurerade virtuella maskiner kan klonas och läggas upp på exempelvis en nätverksdisk och sedan laddas ner när testning i ett visst operativsystem ska göras eller för att snabbt sätta upp virtuell testning på en ny PC.

Den process som upplevdes som minst användarvänlig under utvecklingen och vid konfigurering på andra datorer var den att hämta ut den unika identifikationssträng som USB-enheterna har i de olika virtualiseringsmjukvarorna. Identifikationssträngen används för att specificera vilken USB-enhet som ska monteras i en virtuell maskin. I VMware måste användaren titta i loggfilen till en virtuell maskin för att hitta identifikationssträngen. För att underlätta den processen skapades en liten applikation som utifrån kriterierna på hur identifikationssträngen ser ut kontrollerar vald logg-fil och visar de specifika raderna för användaren. Den virtuella maskinen måste dock ha startats en gång sedan USB-enheten monterats för att loggfilen ska vara uppdaterad med alla tillgängliga USB-enheter.

6.4 Tidsaspekten

För att svara på frågan i problemformuleringen om det är möjligt att spara tid (mantimmar) genom att testa virtuellt, mättes tiden det tog att installera och konfigurera en virtuell maskin. Mätningen genomfördes på en virtuell maskin där Windows Vista installerades på mekanisk disk med VMware Workstation-funktionen Easy Install vilket är en funktion som installerar operativsystemet helt automatiskt utan någon nödvändig interaktion från användaren. Installationen av Windows Vista tog 21 minuter och konfigureringen där automatisk inloggning aktiverades, UAC (User Account Control) stängdes av och där en register-ändring gjordes tog 2 minuter. Därefter stängdes maskinen av och en ögonblicksbild skapades på ett rent system vilket tog 1 minut sammanlagt.

På totalt 24 minuter var det alltså möjligt att installera och konfigurera en ny virtuell maskin för testning. Av dessa 24 minuter behövdes endast 3 minuter av användarinteraktion. Se tabell 2.

Uppgift	**Resultat (minuter)**
Installation av Windows Vista med Easy Install	21 min
Konfiguration av Windows Vista	2 min
Avstängning och skapande av ögonblicksbild	1 min
Skapande av ögonblicksbild (vid avstängd VM)	$<$ 1 min
Återställning till specifik ögonblicksbild (vid avstängd VM)	$<$ 1 min

Tabell 2: Tidsmätning uppsättning av en virtuell maskin

När den virtuella maskinen är installerad och konfigurerad kan testning av mjukvara göras i den. Om mjukvaran som ska testas har beroenden till annan mjukvara än den som följer med operativsystemet kan ytterligare installation och skapande av ögonblicksbilder behöva göras innan maskinen kan börja användas för testning.

Den största tidsvinsten märks dock vid repeterad testning, alltså efter att maskinen är installerad och konfigurerad. I och med att återgången till ett sparat tillstånd med hjälp av ögonblicksbilder endast tar några få sekunder kan ett nytt test startas nästan omedelbart. Jämförelsen med att manuellt installera om en fysisk maskin från grunden eller att använda förberedda disk-avbildningar visar en av de största fördelarna med virtuella maskiner.

För att vid testning alltid utgå från ett känt tillstånd ska maskinen återställas innan varje test. Detta innebär att en fysisk maskin då måste installeras om eller att en disk-avbildning måste användas för att återställa den. Tiden för detta är långt mycket mer än några sekunder som i fallet med virtuella maskiner. Cirka 30 minuter upp till en timme är rimligt för återställning av en fysisk maskin och då ska det poängteras att detta ska göras mellan varje test på maskinen i fråga. Återställningstiden skulle då bli större än själva testtiden. Med virtuella maskiner kan återställning göras på några sekunder och maskinen är sedan redo att användas i nästa test.

6.5 Prestandaaspekten

Prestandaresultaten har tagits fram genom mätning av exekveringstid för tester i det ursprungliga och det utvidgade testramverket. Operativsystemet som användes i de virtuella maskinerna för utvärderingen var Windows 7 x64 Professional. Syftet med mätningarna var att ta reda på hur körtiden för regressionstesterna i testramverket påverkades av att köras virtuellt jämfört med fysiskt.

6.5.1 Virtuella maskiner på mekanisk disk

Tiden för att köra *testjobb* sekventiellt respektive med olika nivåer av parallellisering mättes inledningsvis när de virtuella maskinerna kördes från mekanisk disk. *Testjobben* innehöll testuppsättningen som används för regressionstestning av SCOMM samt att en USB-licensnyckel var tillgänglig i varje virtuell maskin. Tiden mättes från start av testkoordinator-applikationen tills dess att alla *testjobb* har blivit klara, vilket innebar att all tid för uppsättning av de virtuella maskinerna och installation av SCOMM togs med i mätningen.

Den uppmätta tiden för ett sekventiellt *testjobb* var 2155 sekunder och körtiden för två parallella *testjobb* var 3106 sekunder. Detta innebär en tidsökning med 44% för dubbelt så mycket testning när de virtuella maskinerna körs från en mekanisk hårddisk. Tre parallella *testjobb* tog 82% längre tid än ett sekventiellt *testjobb* och fyra parallella *testjobb* tog 125% längre tid än ett sekventiellt *testjobb*. I figur 10 finns en grafisk presentation av resultaten. Tiderna är ett medelvärde baserat på tre repeterade körningar.

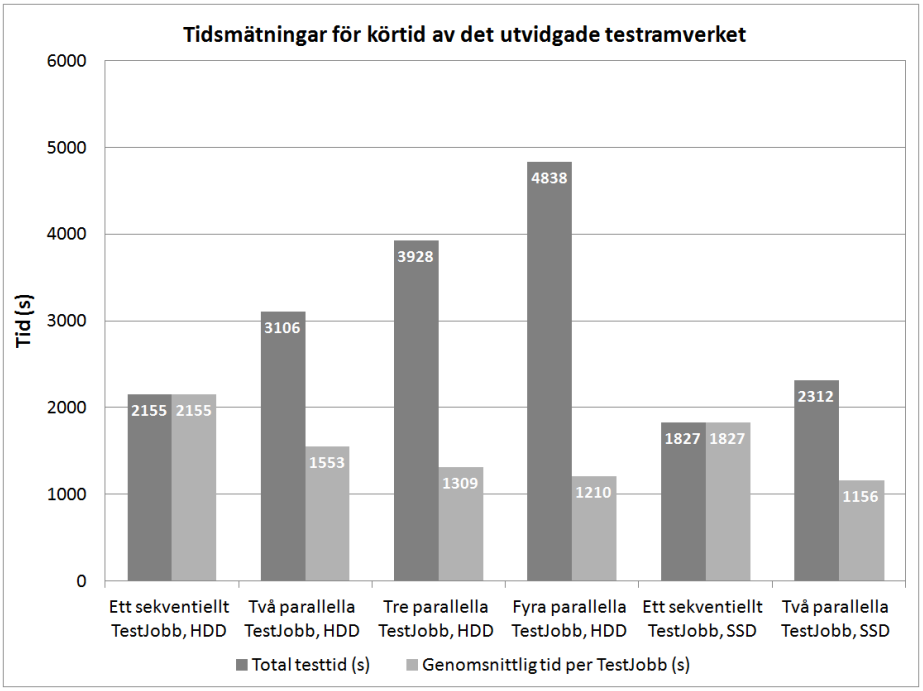
6.5.2 Virtuella maskiner på SSD

Vid körning av parallella *testjobb* i virtuella maskiner från mekanisk disk upptäcktes att disken var väldigt belastad när de virtuella maskinerna startades. Utifrån denna iakttagelse beslutades att även göra tester där de virtuella maskinerna kördes från en SSD (Solid-state drive). På grund av utrymmesbegränsning kunde max två virtuella maskiner köras på den 120 GB Adata S510 SSD som fanns tillgänglig i och med att Windows-installation samt andra applikationer också fanns på disken. Samma *testjobb* användes i testet på mekanisk hårddisk och i testet på SSD. Testtiden för ett sekventiellt *testjobb* uppmättes till 1827 sekunder och testtiden för två parallella *testjobb* var 2312 sekunder, båda fallen med de virtuella maskinerna på SSD. Tidsökningen blev i det fallet 27%. Tiden för både det sekventiella och det parallella *testjobbet* är ett medelvärde av tre repeterade körningar. I figur 10 finns en grafisk presentation av resultaten.

6.5.3 Genomsnittlig testtid per *testjobb*

Utifrån mätresultaten ovan togs även den genomsnittliga tiden per *testjobb* fram för att kunna illustrera prestandavinsten med parallellisering på ett mer konkret sätt. Den genomsnittliga testtiden per *testjobb* sjönk för varje nivå av ökad parallellisering med virtuella maskiner körandes från både mekanisk disk och SSD. I fallet där de virtuella maskinerna kördes på mekanisk disk noterades dock en avtagande vinst i testtid ju fler *testjobb* som kördes parallellt.

Två parallelliserade *testjobb* på SSD fick en genomsnittlig testtid på 1156 sekunder per *testjobb*, jämfört med fyra parallelliserade *testjobb* på mekanisk disk där den genomsnittliga testtiden var 1210 sekunder per *testjobb*. Detta är en skillnad på 671 sekunder respektive 602 sekunder per *testjobb* jämfört med mätningarna av de sekventiella körningarna. Det framgår alltså att två parallella *testjobb* i virtuella maskiner på SSD ger bättre genomsnittlig prestanda än fyra parallella *testjobb* i virtuella maskiner på mekanisk disk. Se figur 10 för en grafisk presentation av resultaten.

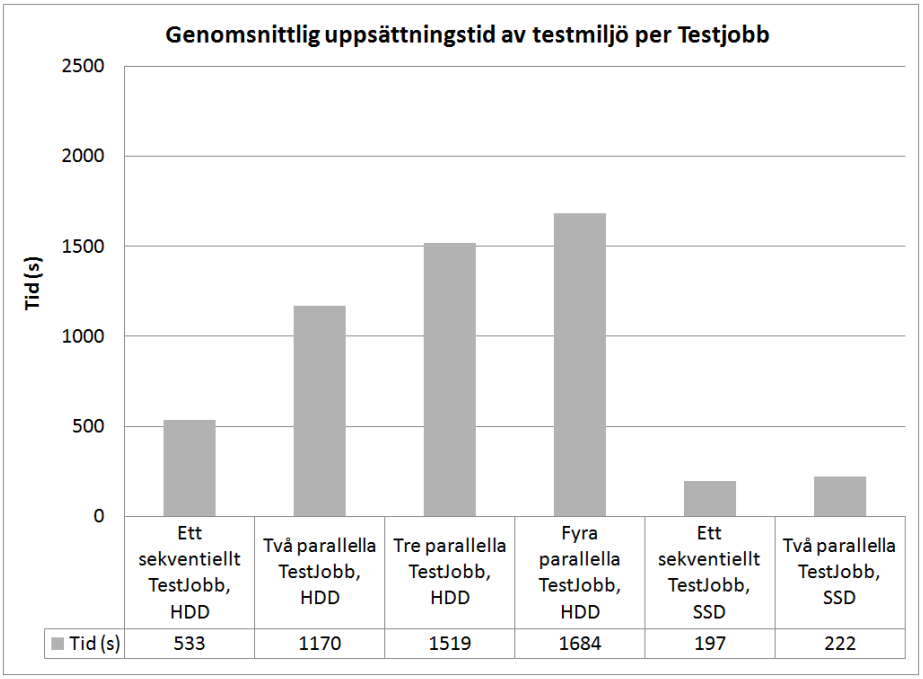


Figur 10: Tidsjämförelse mellan olika nivåer av parallellisering av *testjobb*

6.5.4 Tid för start och installation i virtuell maskin

Genom att subtrahera tidsåtgången för *testuppsättningen* med tidsåtgången för hela *testjobbet* kan uppstartstiden av den virtuella miljön samt tiden för installationen av SCOMM och SCOMMs testramverk i den virtuella maskinen beräknas. Dessa data visade också hur uppsättningstiden för de virtuella miljöerna påverkades beroende på om *testjobben* kördes sekventiellt eller parallellt samt om de virtuella maskinerna kördes från mekanisk disk eller SSD. Med de virtuella maskinerna på mekanisk disk uppmättes en genomsnittlig förlängning av uppsättningstiden per *testjobb* med 120% (533s → 1170s) för två parallella *testjobb* jämfört med ett sekventiellt. Skillnaden mellan uppsättningstiden per *testjobb* för tre parallella *testjobb* och ett sekventiellt *testjobb* var 185% (533s → 1519s) och skillnaden mellan fyra parallella och ett sekventiellt *testjobb* var 216% (533s → 1684s). När de virtuella maskinerna istället låg på SSD uppmättes förlängningen av uppsättningstiden med 13% (197s → 222s) för två parallella *testjobb* jämfört mot ett sekventiellt.

Det mest intressanta är den uppmätta skillnaden mellan uppsättningstiden för *testjobb* vars virtuella maskiner körts från mekanisk disk och från SSD. Uppsättningstiden uppmättes till 170% längre för ett sekventiellt *testjobb* på mekanisk disk jämfört med ett på SSD och den uppmättes till 427% längre för två parallella *testjobb* på mekanisk disk jämfört med SSD. Se figur 11.

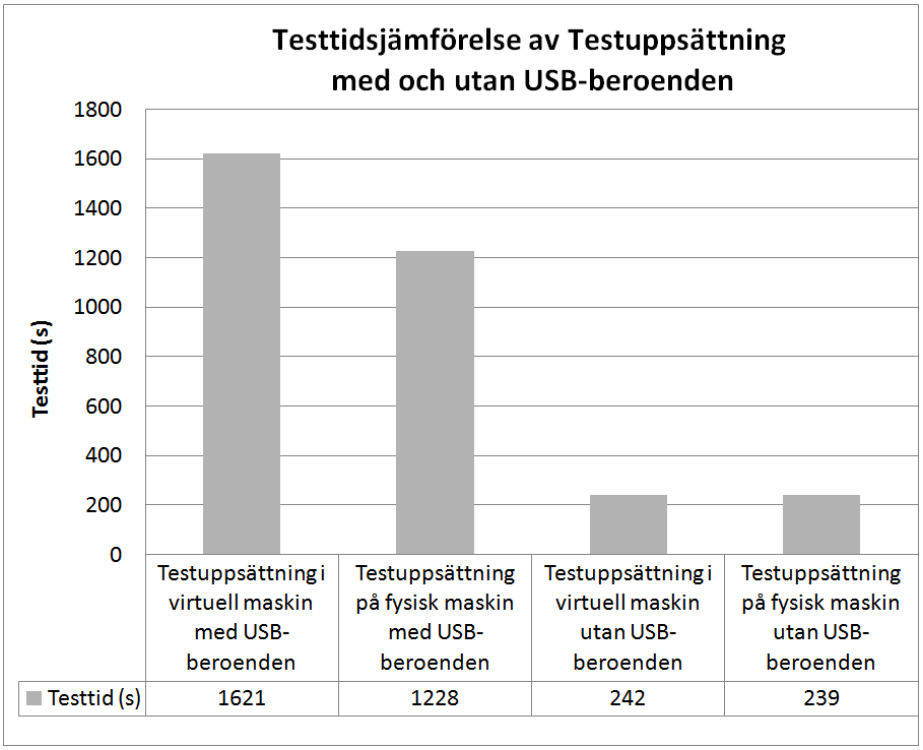


Figur 11: Genomsnittlig tidsåtgång per *testjobb* för uppsättning av den virtuella testmiljön

6.5.5 Undersökning av overhead

För att undersöka hur körningen av en *testuppsättning* påverkades tidsmässigt genom att köras i en virtuell maskin (overhead) gjordes tidsmätningar av dels hur lång tid det tar att köra en *testuppsättning* på en fysisk maskin och dels hur lång tid *testuppsättningen* tar att köra i en virtuell maskin. För att i den virtuella maskinen inte räkna in tiden för uppstart och installation gjordes mätningen från det att *testuppsättningen* startades till och med att den var klar. *Testuppsättningen* som användes innehöll ett komplett regressionstest vilket inkluderade tillgång till en USB-licensnyckel. Samma *testuppsättning* kördes både fysiskt och virtuellt och båda körningarna var från SSD.

Resultatet var något förvånande då körningen i den virtuella maskinen tog 32% längre tid än på den fysiska. I och med att VMware Workstation enligt uppgifter skulle ha väldigt låg overhead (nära nativ exekvering) blev USB-licensnyckeln misstänkt för den stora prestandaskillnaden. Ytterligare ett test gjordes där alla *testkonfigurationer* som krävde USB-licensnyckel plockades bort från *testuppsättningen*. Denna kortare *testuppsättning* kördes sedan i både den virtuella maskinen och på den fysiska. Resultatet visade att testtiderna utan någon USB-licensnyckel var väldigt lika vilket innebär att det var den USB-anslutna licensnyckeln som var orsaken till prestandaförsämringen. Se figur 12 för grafisk presentation av skillnaderna i körtid.



Figur 12: Testtidsjämförelse av *testuppsättning* med och utan USB-beroenden på både fysisk- och virtuell maskin.

6.5.6 USB-prestanda i virtuell maskin

För att mäta hur stor prestandaskillnaden var vid läsning från USB-licensnyckel i en virtuell maskin jämfört med på en fysisk maskin gjordes ett test där informationen från licensnyckeln lästes 60 gånger i rad över sex repeterade körningar från vilket ett medelvärde togs. Testet gjordes för både VMware Workstation och VirtualBox. Körningarna resulterade i tabell 3 nedan.

Produkt	**Läsningstid**	**Förlängning gentemot utan virtualisering**
Utan virtualisering	46 s	-
VirtualBox 4.2.18	62 s	35%
VMware Workstation 9.0.2	69 s	50%

Tabell 3: Utläsning av data 60 gånger i rad från USB-licensnyckel

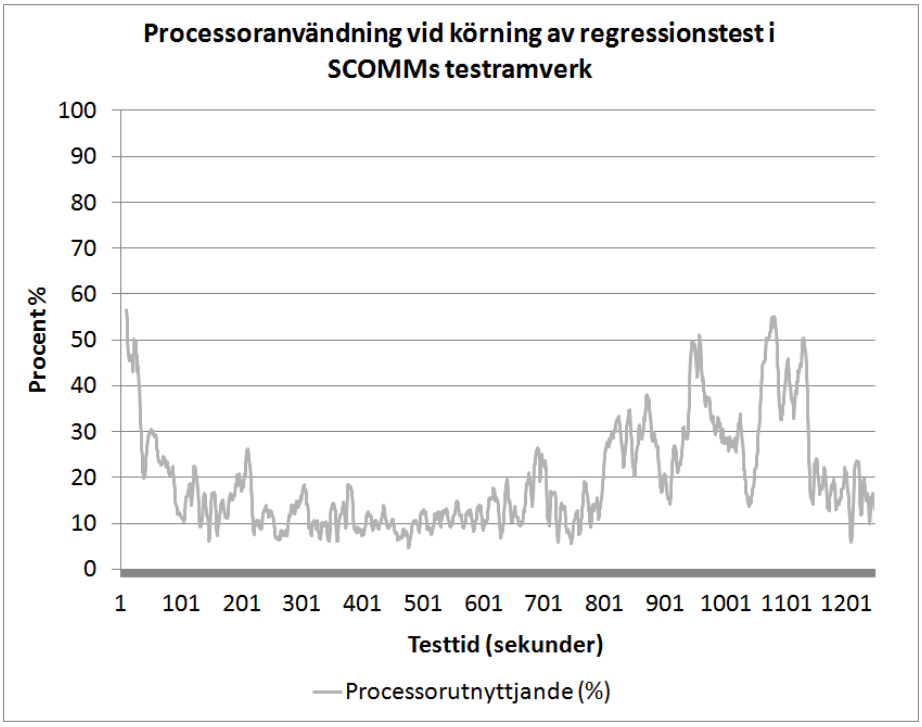
Resultaten visar att det är en markant overhead på läsning från USB-enheter när de är monterade i virtuella maskiner både i VMware och i VirtualBox. Detta kan vara bra att ha i åtanke om USB-prestanda är viktigt för applikationen av virtuella maskiner.

6.6 Hårdvaruutnyttjandeaspekten

För att kunna undersöka hur processoranvändningen påverkas dels vid virtuella och fysiska tester och dels vid olika nivåer av parallellisering loggades processoranvändningen vid körning av *testjobb* och vid körning av SCOMMs ursprungliga testramverk. Den fysiska datorns totala processoranvändning mättes en gång per sekund. Program som inte var relevanta för testkörningarna stängdes av för att inte påverka resultatet.

6.6.1 Hårdvaruutnyttjande i ursprungligt testramverk

Figur 13 nedan visar processorutnyttjandet vid körning av en testuppsättning med SCOMMs regressionstest med USB-licensnyckel på en fysisk maskin med SSD. Som synes används inte mycket av den totala processorkapaciteten utan den genomsnittliga nyttjandegraden är endast 20% vid beräkning av arean under grafen med trapetsregeln. Trapetsregeln användes för att kunna approximera processorutnyttjandet då mätvärde för detta saknades vid enstaka tillfällen på tidsaxeln.



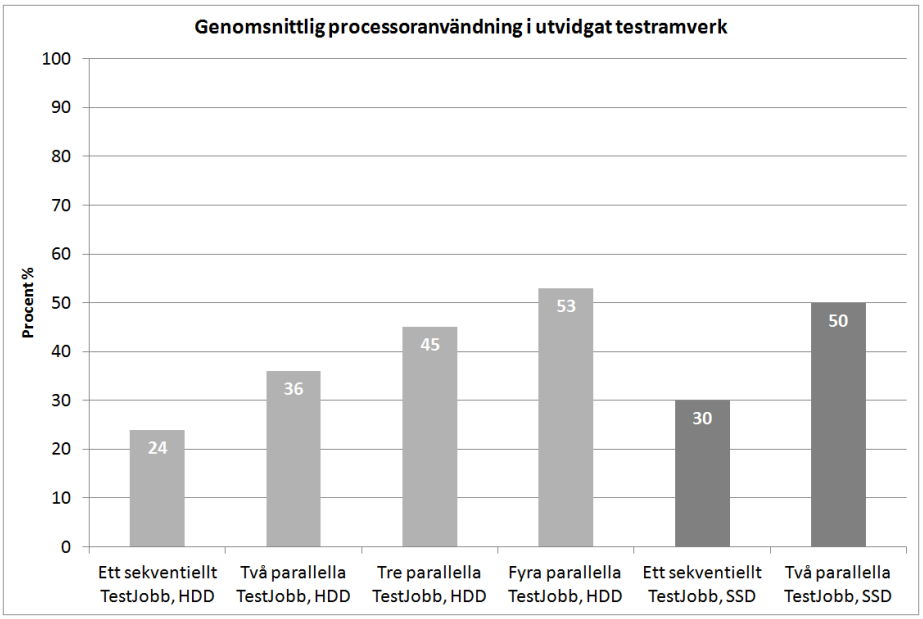
Figur 13: Processoranvändning under körning av en *testuppsättning* utan virtualisering, med USB-licensnyckel

6.6.2 Hårdvaruutnyttjande i utvidgat testramverk

En undersökning av hur processorutnyttjandet ser ut vid körning av parallella *testjobb* är intressant eftersom det klargör vilken nivå som är lämplig för antalet *testjobb* som körs parallellt. Undersökningen visade att processorutnyttjandet ökade med antalet parallella *testjobb*, men ökningen var inte linjär utan

avtagande desto fler *testjobb* som kördes parallellt på mekanisk disk. Totalt jämfördes fyra nivåer av parallelliserade *testjobb* på mekanisk disk och två nivåer av parallelliserade *testjobb* på SSD.

Den största ökningen av processorutnyttjande mellan två parallelliseringsnivåer skedde mellan körning av ett sekventiellt *testjobb* på SSD och två parallella *testjobb* på SSD. Där ökade processorutnyttjandet från 30% till 50%. Se figur 14. Processorutnyttjandet för ett sekventiellt *testjobb* på mekanisk disk och för fyra parallella *testjobb* på mekanisk disk var 24% respektive 53%. Av jämförelsen framgår att processorutnyttjandet per *testjobb* är som störst vid körning av de virtuella maskinerna på SSD.



Figur 14: Processorutnyttjande vid olika nivåer av parallellisering och lagringsmedium för virtuella maskiner

6.7 Automatiseringsaspekten

Det är möjligt att specificera de förutsättningar som ska gälla i den virtuella maskinen, såsom tillgänglig USB-ansluten hårdvara, vilket gör att denna kan monteras automatiskt vid start av en virtuell maskin. Dessa funktioner gör att systemet kan schemaläggas och köras automatiskt efter att en resursfil en gång initialt konfigurerats på varje PC. Testprogrammet kan startas från kommandorad och ger ett returvärde som indikerar om de test som körts har lyckats eller inte.

7 Analys av resultat

I det här kapitlet analyseras resultaten från kapitel 6.

7.1 Prestandaaspekten

I delavsnitten nedan analyseras prestandaresultaten samt vissa fenomen som uppmärksammats vid resultatens sammanställning.

7.1.1 Testkapacitet

Genom mätningarna från prestandaresultaten går det att räkna ut hur många tester det till exempel är möjligt att köra per natt. Avsikten är i skrivande stund att kunna köra en stor mängd regressionstester virtuellt (d.v.s. många *testjobb*) på alla operativsystem varje natt. Testtiden för natt-tester definieras till mellan klockan 18-06, d.v.s. 12 timmar. Över helger skulle det vara möjligt att genomföra utökad testning.

Utifrån de genomsnittliga tiderna för *testjobb* i 6.5.3 är det möjligt att ta reda på den kapacitet för testning som är tillgänglig varje natt. Genom att köra alla *testjobb* sekventiellt på mekanisk disk kan en lägsta kapacitet räknas ut till 20 *testjobb* per 12-timmars-period ($\frac{12\times3600}{2155} \approx 20$). I skrivande stund är det ur testsynpunkt intressant att testa kommunikationskomponenten på 8 olika Windows-versioner (8 *testjobb*) vilket innebär att det finns gott om kapacitet över till nya testmöjligheter. Den högsta *uppmätta*[7] kapaciteten skulle enligt tidigare resultat bli 37 *testjobb* under en 12-timmars-period.

7.1.2 Testtider för testjobb

Vid körning av tester med två parallelliserade *testjobb*, benämnda *Testjobb 1* och *Testjobb 2*, under utvärderingen av systemets prestanda uppmärksammades en avvikelse i den förväntade tidsåtgången. När båda *testjobben* testade samma testuppsättning blev alltid *Testjobb 1* klart innan *Testjobb 2* även om *Testjobb 2* startades först. Analysen av detta visade att det var USB-licensnycklarna som användes i dessa två *testjobb* som var orsaken skillnaden. *Testjobb 2* använde en nyare version av USB-licensnyckeln än *Testjobb 1* och den nyare visade sig vara 25% långsammare än den äldre versionen av USB-licensnyckeln. I virtuella maskiner på SSD tog det 14% längre tid att köra den testuppsättning som använde den långsammare nyckeln. I virtuella maskiner på mekanisk disk tog det ca 25% längre tid.

Notera att nycklarna som användes i de *testjobb* som kördes i utvärderingarna alltid var desamma. Det vill säga vid två parallelliserade *testjobb* hade *Testjobb*1 alltid samma nyckel oavsett om de virtuella maskinerna kördes på mekanisk disk eller SSD etc. Detta har lett till att tidsskillnaderna mellan ett sekventiellt *testjobb* och flera parallelliserade *testjobb* blivit något längre än om alla *testjobb* haft den äldre USB-licensnyckeln. I analysen jämfördes även prestandan på

[7] Om flera parallelliserade *testjobb* på SSD kan köras skulle kapaciteten troligen bli ännu högre om de följde samma trend

USB-nyckeln när läsningar från olika nycklar gjordes samtidigt i olika virtuella maskiner, men ingen prestandaskillnad kunde uppmätas i det fallet.

En kontrollmätning gjordes av två parallella *testjobb* på SSD vilket visade att båda *testjobben* tog lika lång tid när båda använde nycklar av den långsammare versionen. Tiden för den totala körningen var i linje med den som uppmättes för två parallella *testjobb* på SSD, se figur 10. Att tiderna är lika beror på att den långsammare nyckeln var den begränsande faktorn i det ursprungliga testet.

7.1.3 Märkbara symptom till följd av overhead

Vissa symptom har märkts på grund av den overhead som de virtuella maskinerna medför. I några av testerna som ingår i regressionstestet mäts den tiden som en funktion ska vänta innan den börjar läsa eller skriva data till CAN-bussen. Väntetiden kan i vissa fall vara något längre än förväntat på grund av den overhead som finns. Ibland misslyckas därför dessa testfall i den virtuella testningen. Det ska dock poängteras att detta ibland inträffar även när tester körs på fysiska maskiner och beror på datorns belastning vid testet.

7.1.4 Kommunikation mot fysisk CAN-buss

I och med den uppmätta USB-prestandan finns vissa farhågor angående kommunikation mot fysisk CAN-buss med det USB-anslutna diagnoskommunikationsgränssnittet. Ett enkelt test har genomförts som visade att kommunikationen mot en CAN-buss med tre styrenheter på bänk fungerade. I testet lästes en stor konfigurationsfil ut från en uppkopplad styrenhet vilket gick bra även om utläsningen tog aningen längre tid än när den gjordes från en fysisk maskin. Mera omfattande tester skulle behövas på det här området för att utvärdera prestandan mot dels en fysisk rigg, d.v.s. mot flertalet styrenheter ihopkopplade med varandra (ofta monterade på en ställning), och dels mot helbil, dvs ett komplett fordon.

7.2 Hårdvaruutnyttjandeaspekten

I avsnitten nedan analyseras hårdvaruutnyttjandet vid parallellisering av dels *testjobb* och dels *testkonfigurationerna* inuti en *testuppsättning*.

7.2.1 Parallellisering av *testjobb*

Från resultatkapitlet framgick att hårdvaruutnyttjandet kunde ökas mycket vid parallellisering av *testjobb* både om de virtuella maskinerna kördes från HDD eller från SSD. Hårdvaruutnyttjandet kan troligtvis ökas ännu mer med en större SSD så att flera virtuella maskiner kan köras samtidigt. Trots att USB-enheterna var en flaskhals för ett enskilt *testjobb* kunde flera USB-enheter användas parallellt för att öka utnyttjandet av resten av hårdvaran.

7.2.2 Parallellisering inom testuppsättning

Efter att resultaten från processoranvändningen vid parallelliserade *testjobb* hade sammanställts studerades vari begränsningen i processoranvändningen låg för det ursprungliga testramverket, d.v.s. för *testkonfigurationerna* inuti *testuppsättningen*. Teorin var att den låga processoranvändningen på 20% som uppmätts utan virtualisering (med 8 parallella processer) berodde på läsning från licensnyckeln. Om då testerna som var beroende av licensnyckeln plockades bort från *testuppsättningen* borde processoranvändningen kunna uppnå 100% så länge som lämpligt antal processer exekverar parallellt. Testerna utfördes på samma testhårdvara som beskrivs i kapitel 6.2.

Det visade sig emellertid att det inte var möjligt att uppnå en processoranvändning större än ∼60% oavsett hur många processer som exekverade parallellt i det ursprungliga testramverket. Vid fyra parallella processer uppgick processoranvändningen till 27% och vid 8 parallella till 57%. Vid *16 eller fler* parallella processer uppgick processoranvändningen till ∼60%.

Noggrannare studie av de testfall som körs är nödvändigt för att ta reda på var flaskhalsen för processorn finns vid körning utan licensnyckel. Om flaskhalsen kan identifieras är det möjligt att testramverkets prestanda kan förbättras ytterligare, både vid fysiska och virtuella *testjobb*.

7.3 Tidsaspekten

Tiden som testaren behöver lägga på att förbereda testmiljön för att kunna utföra test från ett känt tillstånd kan minskas väsentligt vid användandet av virtuella maskiner. Dessutom innebär möjligheten att automatiskt schemalägga och köra tester av SCOMM i olika operativsystem att testtäckningen kan ökas till en liten initial tidskostnad för uppsättningen av testsystemet.

7.4 Användarvänlighetsaspekten

Även om XML-filerna som används för att konfigurera testsystemet är relativt enkla innebär manuell editering av dem ändå en risk för att fel smyger in. Detta kan leda till att tid behöver läggas på felsökning av testmiljön istället för på själva testen vilket inte är önskvärt. En möjlig förbättring är att göra ett grafiskt gränssnitt som ser till att minimera felen som kan ske vid inmatning och ändring av konfigurationsfilerna. Gränssnittet kan då t.ex. verifiera att sökvägarna till vald virtuell maskin, hårdvarugränssnitt och USB-enheter är korrekta direkt när de matas in av testaren.

7.5 Automatiseringsaspekten

En reflektion som gjorts vid automatiserade körningar av systemet är att alla aspekter som påverkar testresultatens giltighet bör framgå tydligt i rapporten för att minimera onödigt arbete. Om något i förberedelsefasen går fel, som t.ex. installationen av SCOMM eller testramverket, blir ju givetvis resultaten av testkörningarna som är beroende av SCOMM eller testramverket ogiltiga. Att i en sådan situation analysera resultaten av testerna tillför inget direkt värde.

Därför är det väldigt viktigt att kontrollera att alla förutsättningar för att kunna köra testerna är uppfyllda och tydligt visa detta för testaren i rapporten.

8 Analys av implementation

I det här avsnittet analyseras implementationen av testsystemet samt vissa problem som stöttes på vid parallellisering av *testjobb*.

8.1 Styrning av virtuella maskiner

Här nedan analyseras hur styrningen av de virtuella maskinerna fungerat och om det kan finnas fördelar att använda ett API för styrningen istället för CLI.

8.1.1 CLI

I kapitel 5.3 beskrevs hur styrningen av de virtuella maskinerna implementerades och i det här avsnittet görs en reflektion av valet av styrning via CLI och hur styrningen fungerat i praktiken. Generellt sett har lösningen fungerat bra. Det som inneburit lite problem är felhanteringen då anpassade undantag (eng. *exceptions*) inte erhållits från CLI-anropen. Processen som kör ett CLI-anrop returnerar bara en returkod, 0 om allt gått bra och -1 om nåt gått fel. För att hantera fel lästes och tolkades felmeddelanden i form av text från processens *stdout*. En hel del tid gick till att identifiera vissa fel och sedan försöka lösa dem i mjukvara.

Ett problem som märktes vid användningen av CLI:t för VMware Workstation 9.0.2 var att de loggfiler som skapas vid körning av ett anrop med tiden växte, vilket efter några månader resulterade i att CLI-kommandon slumpmässigt returnerade att de misslyckats. Detta fick som följd att tester som kördes vid detta tillfället kunde avbrytas. För att förhindra problemen måste loggfilerna rensas med jämna mellanrum.

8.1.2 API

Vid styrning av de virtuella maskinerna via ett API görs API-anrop direkt i källkoden för de funktionerna som ska köras. En fördel med detta är att koden blir renare än om ett generiskt gränssnitt används som måste starta diverse processer och manipulera strängar för varje anrop. API-anropen har också tydlig undantagshantering vilket förenklar felhanteringen i källkoden. Nackdelen är att källkoden blir specifik för en viss virtualiseringsmjukvara och det blir komplext att använda olika virtualiseringsmjukvaror samtidigt.

8.2 Intermittenta problem vid parallellisering av *testjobb*

Vid testning av att parallellt köra *testjobb* upptäcktes ett problem som ledde till att *testjobb* ibland avbröts innan de ens börjat installera SCOMM i den virtuella maskinen. Det visade sig att detta endast förekom när två eller flera virtuella maskiner startades samtidigt, men felet var ändå intermittent och ibland fungerade det och ibland inte. Efter förbättrad loggning och repeterad körning tills felet uppkom igen noterades att VMware Tools, en mjukvara från VMware som körs som en Windows-tjänst i alla gäst-operativsystem, inte var

igång vid de tillfällen som *testjobb* avbröts för tidigt. VMware Tools måste vara igång i gästen för att denna ska kunna manipuleras via VMware Workstation CLI. När VMware Tools inte startas tror testkoordinatorn att den virtuella maskinen har misslyckats att starta, och testet avbryts. I *Windows EventLog* noterades att VMware Tools hade överskridit den tidsgräns, 30 sekunder, som Windows-tjänster har på sig att starta. Felet berodde på att den mekaniska hårddisken blev en flaskhals vid start av flera virtuella maskiner samtidigt då läsningar och skrivningar från disken tog mycket längre tid än när endast en virtuell maskin startades.

Värt att notera är att detta problem enbart förekom när de virtuella maskinerna var sparade på en mekanisk disk och flera virtuella maskiner startades samtidigt.

Föra att lösa problemet gjordes en registerändring i de virtuella maskinerna där tidsgränsen innan tjänster avbryts om deras start drar ut på tiden ändrades.[8] Tidsgränsen sattes till 600 sekunder.

Detta är långt ifrån en elegant lösning eftersom ändringen behöver göras på varje gäst-operativsystem som installeras, men eftersom det inte är möjligt att ändra i tjänstens uppstartssekvens så att tjänsten ber om mer tid om den inte hinner starta innan tidsgränsen är nådd valdes denna lösning. Genom att lägga in nyckeln och nyckelvärdet i en registerfil (.reg) är det dock möjligt att applicera inställningen betydligt snabbare.

[8] För Windows 7 x64 (och många andra Windows-versioner) skapas eller ändras en nyckel på följande plats: [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control]. Nyckeln ifråga är *ServicesPipeTimeout*. Nyckelvärdet sattes till ett decimaltal på *600000* vilket motsvarar 600 sekunder.

9 Slutsats

De slutsatser som dras utifrån det här examensarbetet är att användningen av virtuella maskiner effektiviserar testningen av mjukvara, dels på grund av att det är möjligt att testa mer på kortare tid, dels på grund av mindre andel manuellt arbete för uppsättning och körning av test. Prestandan är enligt utvärderingen bra och kan ökas mycket genom parallellisering av tester som att t.ex. testa i olika operativsystem samtidigt. Prestandan är fullt tillräcklig för regelbunden testning i virtuella maskiner.

Det var förvånande att det var så pass få av de undersökta virtualiseringsmjukvarorna som uppfyllde samtliga krav som satts upp för det här arbetet. Det var ofta något enstaka krav som inte uppfylldes. Det sammanlagda intrycket av VMware Workstation var mycket positivt. En negativt aspekt som kunde konstateras var avsaknaden av en funktion direkt i CLI för att kunna montera USB-enheter i de virtuella maskinerna. VirtualBox upplevdes som en kompetent virtualiseringsmjukvara med många funktioner. Det största problemet som uppmärksammades och som också innebar en övergång till VMware Workstation var att de USB-enheter som används av SCOMM inte gick att montera i de virtuella maskinerna om USB-enheterna skulle användas samtidigt. I båda virtualiseringsmjukvarorna uppmättes en reducerad USB-prestanda vilket kan vara bra att känna till om användningsområdet av den virtuella maskinen har höga krav på överföringshastighet i det avseendet.

Känslan att virtualiseringsmjukvaror kan ha vissa egenheter som kan påverka pålitligheten har gjort sig påmind under arbetets gång. Det kan vara bra att ha i åtanke att problem kan förekomma beroende på hur produkten används, som t.ex. problemet med tjänstestarten vid parallellisering och problemet med loggfilerna i VMware Workstation som redan nämnts.

Tester i virtuella maskiner innebär nya möjligheter för test av hårdvaruberoende funktioner som inte varit praktiskt möjligt tidigare. Ett exempel på detta är att det nu skulle vara möjligt att automatiskt testa nedladdning av mjukvara till styrenheter monterade i rigg från olika operativsystem utan att t.ex. behöva flytta på fysiska kablar mellan varje test. Om dessa nya möjligheter utnyttjas innebär det att större testtäckning kan uppnås och att fel kan hittas i ett tidigare skede än de kan idag.

Slutligen är virtuella maskiner ett mycket bra verktyg för alla mjukvaruutvecklare som snabbt vill kunna göra grundläggande undersökande testning under utvecklingsfasen av mjukvara i olika miljöer. Tiden det tar att sätta upp några virtuella maskiner är liten i relation till nyttan.

10 Diskussion

I det här avsnittet diskuteras underhåll av virtuella maskiner, hur schemaläggning av testsystemet kan införas, vilka fel som kan upptäckas genom virtuell testning och även virtuella maskiners korrekthet.

10.1 Underhåll av de virtuella maskinerna

För att kunna testa med samma förutsättningar i testmiljön som slutanvändarna har bör de virtuella maskinerna uppdateras med de senaste mjukvaru- och säkerhetsuppdateringarna. Microsoft släpper t.ex. säkerhetsuppdateringar på den andra tisdagen varje månad[9] vilket innebär uppdateringar som potentiellt kan innebära inkompatibilitet hos mjukvaran som testas. Här behöver en avvägning göras mellan att ha en verklighetsanpassad testmiljö och att minimera tiden för underhåll. Att arbeta med många virtuella maskiner medför naturligtvis ett visst underhållsarbete, särskilt om det finns många olika konfigurationer i olika ögonblicksbilder för varje virtuell maskin.

Säg att vi har ett tillstånd X i en virtuell maskin. Utifrån detta tillstånd har vi två tillstånd som utgår från X, benämnda Y och Z. Dessa två tillstånd har olika konfigurationer av maskinen, men de har samma mjukvara med avseende på säkerhetsuppdateringar som X. Om vi nu vill uppdatera Y och Z med nya uppdateringar måste uppdateringarna göras två gånger, en gång för Y och en gång för Z. Detta innebär att dubbla arbetet måste utföras jämfört med om vi bara haft ett tillstånd. Efter uppdateringarna har vi två nya tillstånd Y^1 och Z^1.

En annan metod är att istället utgå från X och där uppdatera systemet till X^1. Då behöver själva uppdateringen endast göras en gång, men sedan tillkommer tid för att från tillståndet X^1 göra om de konfigurationsinställningar som Y och Z hade jämfört med X. Totalt sett har vi då fått tre nya tillstånd, X^1, Y^1 och Z^1. Om de gamla tillstånden (ögonblicksbilderna) dessutom ska sparas för att kunna genomföra regressionstestning mot både gamla och nya tillstånd av operativsystemet kommer det i längden innebära att väldigt många ögonblicksbilder finns och strukturen snabbt blir komplex. Samtidigt kommer det så småningom bli problem med att testtiden blir orimligt lång i och med det ökade antalet testfall.

En medelväg är att göra en analys av hur stor förväntad risk nya säkerhetsuppdateringar till operativsystemet utgör för den mjukvara som ska testas. Om risken inte bedöms vara väldigt stor är en möjlighet istället att uppdatera testmiljön när nya *Service Packs* lanseras. Då sätts en helt ny virtuell maskin upp med det nya service-paketet och i denna görs ögonblicksbilder för de konfigurationer som ska testas. Detta minskar arbetet med underhåll av testmiljön och begränsar antalet testkombinationer. En följd av att en ny virtuell maskin sätts upp är möjlighet till ytterligare parallellisering.

[9] `http://en.wikipedia.org/wiki/Patch_Tuesday`, avläst: 2013-10-31

10.2 Automation med Jenkins

Jenkins är en mjukvara som schemalägger, startar och övervakar repeterade körningar av jobb som t.ex. mjukvarubyggen och mjukvarutester. Scania använder för närvarande Jenkins som verktyg för att både bygga och köra regressionstester på mjukvara som utvecklas. Att anropa systemet som utvecklats i det här arbetet från Jenkins vore en bra lösning för att kunna schemalägga testerna och det finns även möjlighet att automatiskt starta tester när ändringar i källkoden har skickats in till källkodshanteringssystemet.

10.3 Vilka fel kan upptäckas?

Förutom de tekniska lösningarna på styrningen av virtuella miljöer för mjukvarutester är det väldigt intressant att diskutera vad det egentligen är för fel som kan uppstå när mjukvara körs i andra operativsystem än det som utvecklingen görs i. Då kan det finnas problem som kan göra att mjukvaran inte startar över huvud taget på grund av inkompatibilitet med något bibliotek eller någon arkitekturell skillnad mellan operativsystemen.

Andra problem som kan uppstå, vilka är mer specifika för SCOMM, är inkompatibilitet mellan mjukvaran och hårdvaruresurser som licensnycklar och hårdvarugränssnitt som SCOMM använder sig av. Drivrutinspaket från tredjepartstillverkare används i dagsläget i SCOMM och dessa stödjer de vanligaste operativsystemen i Windows-familjen.

Trots att det är tredje part som levererar delar av mjukvaran är det viktigt att kunna testa att det kompletta systemet verkligen fungerar i alla miljöer med berörda drivrutiner för att det ska vara möjligt att hålla en hög kvalité på leveranserna av mjukvaran. Säg att du tillhandahåller en mjukvara som använder någon komponent från tredje part. Om denna tredjepartsmjukvara gör att din produkt inte fungerar som den ska för att du inte testat din produkt tillsammans med tredjepartsmjukvaran är det fortfarande *din* produkt som inte fungerar och *dina* kunder som kommer i kläm.

10.4 Virtuella maskiners korrekthet

I avsnitt 3.4.3 avhandlades korrektheten av vanligt förekommande virtuella maskiner. Då virtuella tester genomförs istället för fysiska förutsätts det ju att de virtuella maskinerna beter sig på samma sätt som de fysiska. Om de virtuella maskinerna inte virtualiserar systemarkitekturen korrekt kan det leda till bortkastad tid genom sökning efter fel i den testade mjukvaran som i själva verket fanns i själva virtualiseringsmjukvaran. Då fel upptäcktes i samtliga testade virtualiseringsmjukvaror, vissa allvarliga, andra mindre allvarliga, måste slutsatsen dras att det inte går att vara säker på att virtualiseringsmjukvarorna är helt transparenta mot gästen.

Resultaten av studien i kapitel 3.4.3 visade att VirtualBox hade fler defekter som ledde till transparensproblem än VMware. Värt att tänka på är att studien gjordes 2010 och både VMware och VirtualBox kan ha ändrats mycket sedan dess. Det var dessutom endast emuleringsmodulerna som testades. Korrektheten av hårdvaruassisterad virtualisering testades ej. Virtualiseringsmjukvarorna som

utvärderats i detta arbete har alla använt hårdvaruassisterad virtualisering.

11 Hållbarhet och Etik

Systemet som utvecklats i det här examensarbetet innebär att det är möjligt att utnyttja den befintliga hårdvaran mer effektivt för att utöka testningen utan att behöva göra direkta investeringar i ny hårdvara. Detta leder både till att belastningen på miljön inte ökar och att, om testerna automatiseras, arbetstiden kan användas till mer kvalificerat arbete.

Referenser

[1] Abran, A., Moore, J. m.fl. (2004): *Software Engineering Body of Knowledge*
ISBN: 0769523307

[2] *Systems and software engineering —Vocabulary*, 2010
ISO/IEC/IEEE 24765

[3] Veenendaal, E. (2012): *Standard glossary of terms used in Software Testing, version 2.2*
International Software Testing Qualifications Board

[4] Rosenblum, M. (2004): *The Reincarnation of Virtual Machines*
Magazine, Queue - Virtual Machines, Volym 2, Nr 5

[5] Figueiredo, R J (2003): *A Case For Grid Computing On Virtual Machines*
23rd International Conference on Distributed Computing Systems, Proceedings. s. 550-559

[6] Smith, J, Nair, J. (2005): *The architecture of virtual machines*
Magazine, Computer, Volym 38, Nr 5

[7] Tanenbaum A. S. (2009): *Modern Operating Systems*
Pearson Education, Upplaga 3

[8] Popek G. J., Goldberg R. P., (1974): *Formal requirements for virtualizable third generation architectures*
Communications of the ACM, Volym 17, Nr 7

[9] Bugnion, E., Devine, S., Rosenblum, M. m fl (2012): *Bringing Virtualization to the x86 Architecture with the Original VMware Workstation*
ACM Transactions on Computer Systems (TOCS), Volym 30, Nr 4

[10] Martignoni, L., Paleari R., Roglia, G. m.fl (2010): *Testing System Virtual Machines*
ISSTA '10 Proceedings of the 19th international symposium on software testing and analysis, s. 171-182

[11] VMware Workstation 9 Documentation Center
`http://pubs.vmware.com/workstation-9/index.jsp?topic=\%2Fcom.vmware.ws.using.doc\%2FGUID-7CF19099-D200-4972-B6A2-48DCDE1F3B15.html`
Avläst: 2013-05-09

[12] *Certified Tester, Foundation Level Syllabus, Version 2011*
International Software Testing Qualifications Board

[13] David Chappell, Chappell & Associates (2010):
Introducing Windows Communication Foundation
Microsoft Corporation