Fjärrextrahering av komprimerade filarkiv

En undersökning av komprimerade filarkiv och utökning av *Custom Tools Plugin* – en insticksmodul till Jenkins

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Martin Hjelmqvist martin@hjelmqvist.eu

© Copyright Hjelmqvist, Martin

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2016

Sammanfattning

Continuous Integration, eller kontinuerlig integration är ett sätt att hantera komplexiteten i moderna projekt inom mjukvaruutveckling. Istället för att separat utveckla moduler och därefter utföra en tidsödande integrationsfas i slutet av varje projekt så är det möjligt att från början testa varje modul var för sig och se till att de kan samarbeta så tidigt som möjligt i utvecklingsarbetet.

Företaget *HMS Industrial Networks* är en oberoende leverantör av produkter för industriell kommunikation och använder sig av systemet Jenkins, tidigare kallat Hudson, för integration av sina moduler. Jenkins automatiserar och följer upp integration och testning, men i en av de insticksmoduler företaget använder saknas viss funktionalitet.

Insticksmodulen *Custom Tools Plugin* (CTP) till Jenkins kan använda *tool installers* för att extrahera komprimerade filarkiv i både ZIP- och TAR.GZ-format på datorer som är anslutna till Jenkins-servern. Dessa två filformat kan dock maximalt vara 4 respektive 8 gigabyte stora. Syftet med det här examensarbetet är att undersöka olika komprimerade arkivfilformat och implementera stöd för ett tredje filformat med stöd för en filstorlek på upp till och med 100 Gigabyte.

Examensarbetet utmynnade i utveckling av en helt ny, fristående tool installer som kan extrahera komprimerade filarkiv i RAR-format. Tool installern integrerades i *Extra Tool Installers Plugin* (ETI), som är en insticksmodul till Jenkins och nyttjas av CTP. Filarkiv med RAR-formatet kan ha en maximal filstorlek på 9 miljarder gigabytes vilket innebär att kravet avseende filstorlek är uppfyllt.

Slutresultatet är därmed att Jenkins-användare genom CTP kan installera program som överstiger den maximala filstorleken på 4 och 8 gigabyte hos formaten ZIP och TAR.GZ.

Nyckelord: Jenkins, insticksmodul, Java, komprimering, filarkiv, filformat

Abstract

Continuous Integration is an approach to handle the complexity in modern software development projects. Instead of developing modules separately and then performing a time consuming phase of integrating all modules in the end of each project, it is possible to continuously test the compatibility of a single module as they are developed.

The company *HMS Industrial Networks* is an independent distributor of products for industrial communication. They use the Jenkins system, previously named Hudson, to continuously integrate their modules. Jenkins automate and monitors integration and testing, but in one of the Jenkins plugins they use there is a flaw.

The Jenkins plugin *Custom Tools Plugin* (CTP) can use *tool installers* to extract compressed file archives in two file formats, ZIP and TAR.GZ, to computers connected to the Jenkins server. These two file formats can only have a maximum file size of 4 and 8 Gigabyte respectively. The purpose of this thesis is to examine different file formats and implement support for a third format with a maximum file size of up to 100 Gigabyte.

The thesis culminated in the development of a new independent tool installer capable of extracting compressed RAR archives. The tool installer was integrated in the Jenkins plugin *Extra Tool Installers Plugin* (ETI) which can be used by CTP. RAR-archives can have a maximum file size of 9 billions Gigabyte which fulfils the requirement on the implemented solution.

The end result is therefore that Jenkins users through CTP are able to install programs that exceeds the maximum file size limit on 4 and 8 gigabyte of the file formats ZIP and TAR.GZ.

Keywords: Jenkins, java, plugin, compression, archive, file format

Förord

Jag vill tacka HMS Industrial Networks och mina handledare, Joakim Wiberg och Timmy Brolin, för möjligheten att få arbeta med detta uppdrag och stifta bekantskap med kontinuerlig integration i en industriell verksamhet. Ett stort tack riktas även till Christian Nyberg och Christin Lindholm för den hjälp och support som de bidragit med i deras roller som handledare och examinator.

Innehållsförteckning

1 Inledning 1
1.1 Bakgrund 1
1.2 Syfte 2
1.3 Målsättning 2
1.4 Frågeställningar 2
1.5 Avgränsningar 3

2 Teknisk bakgrund 5
2.1 Jenkins 5
2.2 Insticksmoduler till Jenkins 9
2.2.1 Custom Tools Plugin 9
2.2.2 Extra Tool Installer Plugin 12
2.3 Utvecklingsmiljöer 13
2.3.1 Eclipse Kepler 14
2.3.2 Maven 14
2.3.3 Netbeans 8.1 15
2.4 GitHub 16
2.5 Komprimerade filformat 17
2.5.1 Filformat: 7z 17
2.5.2 Filformat: RAR 18

3 Metod 21
3.1 Arbetsprocess 21
3.1.1 Informationsinhämtning 21
3.1.2 Undersökning av lämpliga filformat 21
3.1.3 Utveckling av uppackningsprogram till utvalda filformat 22
3.1.4 Mätningar på uppackning av komprimerade filer 22
3.1.5 Val av filformat 25
3.1.6 Integrering av utvalt uppackningsprogram 27
3.1.7 Tester och mätningar som bekräftar önskad funktionalitet 27
3.1.8 Integrering av resultatet till GitHub 28
3.2 Felkällor 28
3.3 Validitet och Reliabilitet 29
3.3.1 Examensarbetets validitet 29
3.3.2 Examensarbetets reliabilitet 29
3.4 Källkritik 30

4 Analys 33
4.1 Val av filformat 33
4.2 Val av implementation 34

5 Resultat.. 37
5.1 Mätresultat före implementationen 37
5.2 Implementation... 40
5.3 Mätresultat efter implementationen 43
5.4 Begränsningar 44
6 Slutsats.. 47
6.1 Framtida utvecklingsmöjligheter.. 51
7 Terminologi 53
Referenser 57
Appendix 63
A Källkod: RarExtractionInstaller.java 63
B Källkod: RarFilePath.java 67
C pom.xml 73

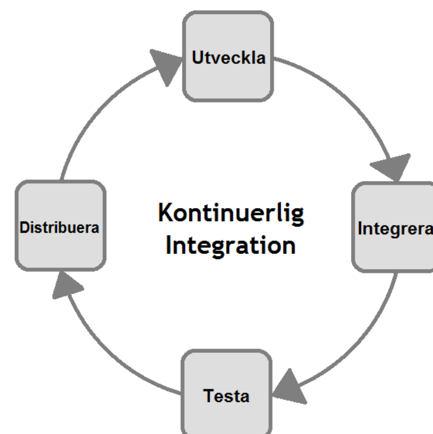
1 Inledning

Detta är ett examensarbete för högskoleingenjörsexamen på Lunds Tekniska Högskola och har utförts på HMS Industrial Networks i Halmstad. Examensarbetet undersöker komprimerade arkivfilformat och utökar en insticksmodul till Jenkins med funktionalitet att kunna fjärrextrahera ett komprimerat arkiv på en dator ansluten till en Jenkins-server.

1.1 Bakgrund

I moderna mjukvaruprojekt kan en programvara utvecklas på två sätt med avseende på integration av ny funktionalitet till den blivande slutprodukten. Det ena sättet innebär att först utveckla alla funktioner och komponenter, integrera dessa samtidigt, testa programvaran och sedan distribuera produkten. Att integrera alla delar i en och samma fas i hopp om att de ska passa som pusselbitar kan ta lång tid att utföra. Det finns därför ett annat tillvägagångssätt för att undvika en tidsödande integrationsfas med potentiellt misslyckade tester. Metoden innebär att utveckla en komponent eller funktion i taget, integrera i den blivande slutprodukten, testa och bekräfta önskad funktionalitet. Därmed kan en alltid körbar mjukvara finnas tillgänglig. Detta kallas för *kontinuerlig integration*.

Kontinuerlig integration innebär att när en mindre modul är klar så integreras den i den blivande slutprodukten, testas och distribueras. Därefter påbörjas nästa modul som utvecklas, integreras, testas och distribueras (se figur 1.1). Syftet är att inte behöva genomgå en lång integrationsfas i den avslutande delen av ett projekt. [1]



Figur 1.1: Cykeln i kontinuerlig integration (Utveckla – Integrera – Testa - Distribuera)

Ett system som möjliggör kontinuerlig integration är Jenkins. Jenkins ägdes först av Sun Microsystems Inc. och senare av Oracle under namnet Hudson. Precis som Jenkins var Hudson ett system för kontinuerlig integration. Arbetet med Hudson leddes av mjukvaruutvecklaren Kohsuke Kawaguchi. Senare bytte Hudson namn till Jenkins. På grund av meningsskiljaktigheter i organisationen splittrades projektet i två delar. De två projekten, Hudson och Jenkins, kan ses som två grenar, där Oracle behöll det kommersiella Hudson och Kohsuke Kawaguchi vidareutvecklade Jenkins öppna källkod. [2]

Jenkins-CI [3] är ett system för kontinuerlig integration som används till att kontinuerligt bygga och testa mjukvaruprojekt i syfte att göra det lättare för utvecklare

att integrera ändringar i olika projekt. Ett företag som använder Jenkins är *HMS Industrial Networks* [4] som är en oberoende leverantör av hårdvaruprodukter för industriell kommunikation. De är i behov av att kunna installera större program i sina utvecklingsprojekt, men *Custom Tools Plugin* [5] (CTP), en insticksmodul till Jenkins, som automatiskt installerar olika program saknar stöd för komprimerade installationsfiler som är större än 8 GB. Jenkins kan utökas med insticksmoduler [6] för ökad funktionalitet. En av dessa insticksmoduler är Custom Tools Plugin. Denna modul kan installera olika program på datorer i ett datorkluster och består av olika installationsprogram (*tool installers*) som kan göra detta. [5] En av dessa tool installers kan hämta ett komprimerat filarkiv via en specificerad länk och överföra arkivet till datorklustret tillsammans med kod som kan utföra uppackning av arkivet på plats hos respektive dator i klustret. De format som tool installern kan hantera är ZIP och TAR.GZ och dessa kan med den nuvarande koden i Custom Tools Plugin ha en maximal filstorlek på 4 respektive 8 GB, men HMS är i behov av att kunna installera program som är upp till och med 100 GB stora. [5]

1.2 Syfte

Syftet med examensarbetet är att utveckla stöd för större komprimerade installationsfiler hos insticksmodulen *Custom Tools Plugin*. Insticksmodulen ska kunna genomföra en snabb installation av komprimerade filer med storlekar upp till och med 100 GB.

1.3 Målsättning

Målet med examensarbetet kan i huvudsak sammanfattas i nedanstående tre punkter:

1. Den utökade insticksmodulen ska kunna genomföra installationer med komprimerade installationsfiler med storlekar upp till och med 100 GB, på datorer med operativsystemet Windows eller Linux.
2. En installation ska ta mindre tid än vad den gör idag. Mätningar ska genomföras under en installation av programmet Vivado (som har en ungefärlig filstorlek på 4 GB) i syfte att få veta hur lång tid det tar i nuläget.
3. Utökningen ska integreras med insticksmodulens ursprungliga ”repository” på GitHub (se kapitel 2.4).

1.4 Frågeställningar

Utifrån syftet och målsättningarna kunde följande frågeställningar formuleras:

1. Vilket filformat som stödjer komprimerade installationsfiler med storlekar upp till och med 100 GB är bäst lämpat att implementera stöd för i Custom Tools Plugin?
2. Installationen ska ta mindre tid än i nuläget. Hur lång tid tar den nu?
3. Vilka faktorer påverkar tiden för en installation?
4. Hur kan man påverka dessa faktorer för att minska tiden för en installation?
5. Hur lång tid tar installationen med det nya filformatet?
6. Vad krävs för att resultatet ska bli integrerat i GitHub?
7. Hur fungerar kontinuerlig integration?

1.5 Avgränsningar

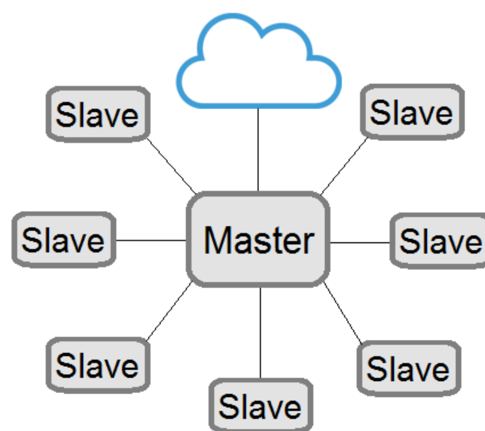
Insticksprogrammet ska kunna installera på datorer med antingen operativsystemet Windows eller Linux-distributioner. Utveckling skedde enbart i programmeringsspråket Java eftersom det existerande insticksprogrammet är programmerat i Java.

2 Teknisk bakgrund

I detta kapitel om teknisk bakgrund ges en presentation av olika arkivfilformat med tillhörande komprimeringsprogram och komprimeringstekniker. Jenkins-CI och insticksmodulerna Custom Tools Plugin (CTP) och Extra Tool Intallers Plugin (ETI) till Jenkins presenteras liksom hur källkoden till dessa moduler versionshanteras på GitHub. Även skillnaderna mellan de olika utvecklingsmiljöerna som användes under examensarbetet belyses. CTP och ETI var två av de insticksmoduler till Jenkins som examensarbetet berörde, då filformaten 7z och RAR undersöktes. Utveckling skedde i olika utvecklingsmiljöer, Eclipse och Netbeans samt byggsystemet Maven. Källkod laddades upp till GitHub som tillhandahåller ett versionshanteringssystem.

2.1 Jenkins

Jenkins är ett system för automatisering och uppföljning av integration och testning [7]. Systemet möjliggör kontinuerlig integration i moderna utvecklingsprojekt istället för att i slutet av ett projekt utföra en lång integrationsfas.

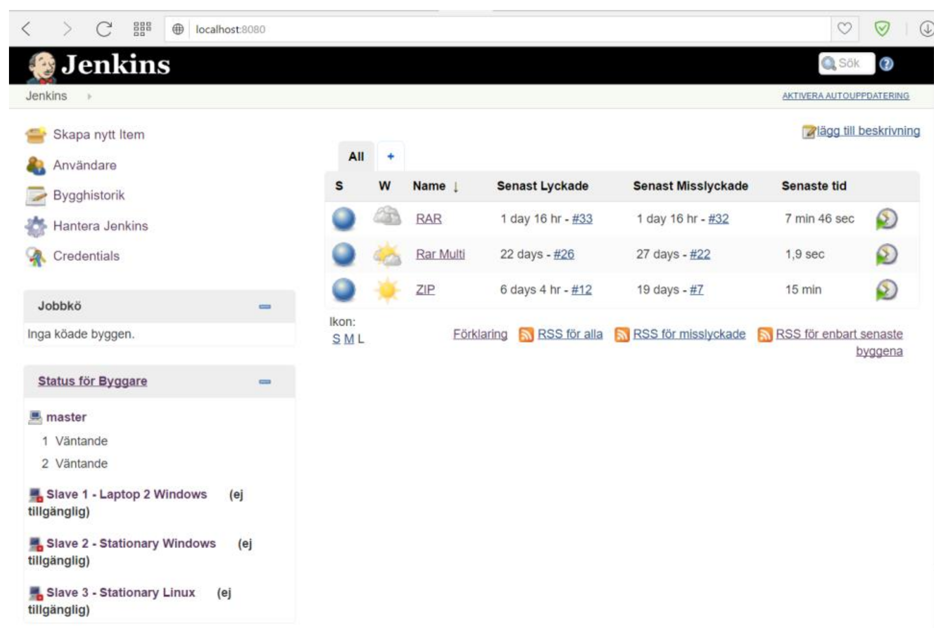


Figur 2.1: Ett datorkluster med noder (slaves) anslutna till Jenkins-servern (master). Servern betjänar HTTP-förfrågningar från noderna.

För att använda Jenkins måste det installeras på den server som kommer att köra *jobb* på anslutna *noder*. Noder, datorer anslutna till en Jenkins-server (se figur 2.1), kan utföra ett visst antal specifika jobb parallellt med varandra, beroende på hur många *executors* respektive nod är inställd att ha. En executor kan liknas vid en processorkärna som kan utföra en uppgift i taget. Dessa jobb kan konfigureras att utföra vissa uppgifter. Det finns två typer av noder i Jenkins. Den ena kallas för *Master* och är belägen på den server där Jenkins är installerat. Mastern kan utföra jobb och betjäna förfrågningar av HTTP-protokollet från anslutna noder (se figur 2.1). Om risk finns att belastningen blir för stor på mastern kan användaren skapa och konfigurera slavar på noderna som är anslutna till servern innan jobb körs för att avlasta mastern genom att fördela belastningen över slavarna. Mastern fördelar automatiskt jobb som ska köras på de anslutna nodernas executors för att själv inte bli överbelastad. En användare kan starta en slav-instans på en nod genom att köra ett program som kallas för *Slave Agent.* Det behövs alltså ingen grundinstallation av Jenkins på någon slavnod för att utföra olika jobb. Det är möjligt att köra slavar på samma dator som mastern. [8]

När Jenkins-servern är igång finns det möjlighet att gå in i Jenkins genom ett webbläsargränssnitt där Jenkins kan konfigureras (se figur 2.2).

Gränssnittet nås genom adressen `http://server-name:port/` där `server-name` är namnet på den dator som kör Jenkins-servern och `port` är den port på datorn som används av Jenkins. I gränssnittet går det att konfigurera Jenkins, användare, noder och jobb.



Figur 2.2: Skärmdump av webbläsargränssnittet i Jenkins.

Det finns olika sätt att starta en Slave Agent beroende på vilket operativsystem som används på datorn. Följande metoder kan användas för att starta en Slave Agent:

1. Master startar Slave Agent via SSH [9]
2. Master startar Slave Agent i Windows [10]
3. Ett skript som startar en Slave Agent exekveras [11]
4. Java Web Start [12]
5. Exekvera *slave.jar* i en kommandoterminal [13]

I examensarbetet användes Java Web Start [12] på datorer med operativsystemet Windows och för Linux-distributioner exekverades slave.jar i kommandoterminalen. Med Java Web Start så behöver användaren öppna Jenkins-serverns gränssnitt i en webbläsare på den blivande slavnoden, välja den slav i nodlistan i figur 2.3 som den vill agera som och sedan klicka på ikonen ”Launch” (se figur 2.4). Då öppnas ett fönster som visar status på uppkopplingen mellan master och slave (se figur 2.4).

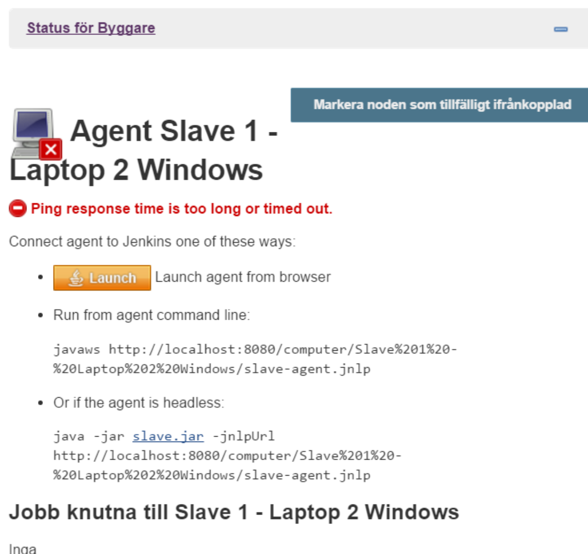
På Linux-distributioner fungerar inte ovanstående metod på grund av komplikationer med Java. Istället måste användaren hämta en JAR-fil från serverns webbläsargränssnitt till den blivande slavnoden och exekvera följande kommando i terminalen [13]:

```
$ java -jar slave.jar -jnlpUrl http://yourserver:port/computer/slave-
  name/slave-agent.jnlp
```

I http-adressen måste `yourserver` ersättas med namnet på datorn som servern körs på och `slave-name` måste ersättas med det namn slaven fick när den konfigurerades i Jenkins.

Figur 2.3: En lista med förkonfigurerade noder i Jenkins webbgränssnitt. Listan finns även med i figur 2.2.

När konfigurationen av noder är klar går det att konfigurera de jobb som ska köras. Det finns flera olika typer av jobb som använder olika metoder för att utföra specifika uppgifter. Dessa jobb kan både köras en efter en eller kopplas samman till en större kedja av uppgifter. Det finns många olika typer av jobb, men alla är inte inbyggda i Jenkins. För att få tillgång till ett jobb som ännu inte finns i Jenkins måste dess insticksmodul installeras. I tabell 2.1 listas tre olika typer av jobb som användes i examensarbetet för att testa önskad funktionalitet i lösningen som utvecklades.



Figur 2.4: Olika alternativ för att ansluta till den förkonfigurerade slavnoden "Laptop 2 Windows" från figur 2.3.

Freestyle Project	Jenkins bygger jobbet med valfri versionshanterare och valfritt byggsystem. Jobbet kan utföra många av de olika typer av uppgifter som vanligtvis behöver göras.
Pipeline	Jenkins exekverar ett Groovy-skript [14] som kan utföra en serie med uppgifter som sträcker sig över flera slavnoder.
Multi-configuration Project	Lämplig för projekt som behöver ett stort antal olika konfigurationer. Projektet har en så kallad konfigurationsmatris där noder anslutna till Mastern kan markeras för utföra markerade jobb. Till exempel, ett mjukvaruprojekt måste testas på flera olika operativsystem. I matrisen markeras tre noder med operativsystemen Windows, Linux respektive MacOS X att köra ett bygge som testar projektets funktionalitet.

Tabell 2.1: Tre olika jobb i Jenkins som användes i examensarbetet. Genom att installera fler insticksmoduler i Jenkins kan andra jobb skapas utöver dessa.

Ett konkret exempel på hur ett av dessa jobb kan användas presenteras i figur 2.5 i form av ett Groovy-skript för en Pipeline**.** Exemplet förevisar hur en Pipeline i Jenkins kan användas i den kontinuerliga integrationen för att testa om en modul som har integrerats i ett mjukvaruprojekt fungerar. Mastern påbörjar exekvering av skriptet där första raden är ett node-block. Det innebär att koden i detta block ska köras på en nod som Mastern tilldelar. Mastern kan välja att tilldela uppgiften till en slavnod i syfte att fördela belastningen. I skriptet görs en beskrivande `echo`-utskrift för varje steg. Det första steget (`git url: '…'`) hämtar det repository från GitHub där mjukvaruprojektet är beläget. Det andra steget (`tool`) använder sig av CTP (se kapitel 2.2.1) för att installera ett program `Maven` på noden, där det kommer att nyttjas i kommande steg för att bygga projektet. Nästa steg (`def`) är att definiera en variabel för `Maven` för att veta vart programmet installerades. Det värde som returneras av variabeln är en sökväg till platsen där programmet installerades. I det sista steget (`bat`) bygger Maven projektet i det repository som hämtades i första steget. Om något fel uppstår i bygget av projektet skrivs ett felmeddelande ut i loggen för den Pipeline som kördes. Det innebär att integrationen inte godkändes, måste åtgärdas och sedan testas om igen. Om bygget går igenom betyder det att integrationen lyckades och den kontinuerliga integrationen kan fortsätta.

Pipeline

Definition: Pipeline script

Script

```
node {
    echo 'Hämtar projektet Simple-Maven-Project från GitHub.'
    git url: 'https://github.com/user/simple-maven-project.git'

    echo 'Tillser att Maven är installerat på noden.'
    tool name: 'Maven', type: 'com.cloudbees.jenkins.plugins.customtools.CustomTool'

    echo 'Definierar en variabel som representerar Maven.'
    def mvnHome = tool  'Maven'

    echo 'Ett batch-kommando exekveras som kör Maven-kommandot "verify" på GitHub-projektet.'
    bat "${mvnHome}\\bin\\mvn -B verify"
}
```

☑ Use Groovy Sandbox

Pipeline Syntax

Figur 2.5: Ett exempel på ett Groovy-skript i en Pipeline. Skriptet hämtar ett repository, installerar Maven (se kapitel 2.3.2) och exekverar sedan ett Maven-kommando som bygger projektet från det repository som hämtades.

2.2 Insticksmoduler till Jenkins

En insticksmodul till Jenkins är mjukvara som kan installeras i Jenkins för utökad funktionalitet i de jobb som körs. Insticksmodulerna kan antingen hämtas och installeras genom Jenkins inbyggda insticksmodulshanterare eller uppladdning av en HPI-fil som innehåller modulen. HPI-filen innehåller en mappstruktur med filer som definierar insticksmodulen och skapas när insticksmodulen byggs i Maven (se kapitel 2.3.2).

2.2.1 Custom Tools Plugin

Custom Tools Plugin [5] (CTP) är en insticksmodul till Jenkins som kan installera *custom tools* på noder anslutna till en Jenkins-server med hjälp av så kallade *Tool Installers*. Ett Custom Tool är en konfiguration som kan genomföra installation av ett eller flera program till en förutbestämd plats hos respektive ansluten nod. Varje enskilt program som ett custom tool är konfigurerat att installera, installeras av en tool installer. En tool installer är en Java-klass som utför en installation av ett program. *ZipExtractionInstaller* är en tool installer som hämtar ett ZIP-arkiv från en angiven adress och extraherar arkivet hos respektive ansluten nod (se streckat område i figur 2.6).

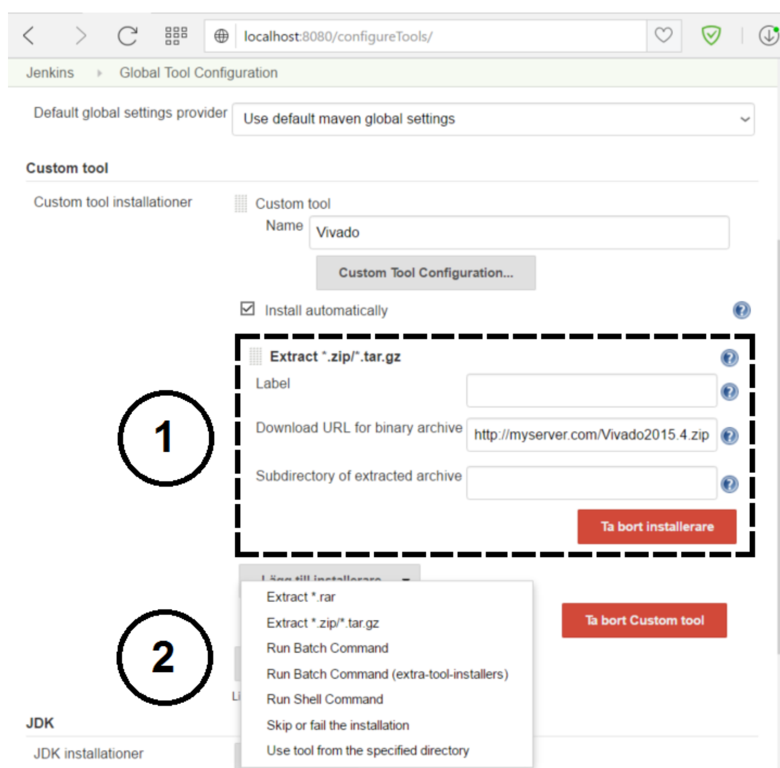
CTP är utvecklat i Java och källkoden finns tillgängligt som open source i dess repository på GitHub (för beskrivning av GitHub, se kapitel 2.4). [15]

CTP kan utföra installation av program genom följande tool installers i tabell 2.2:

BatchCommandInstaller	Exekvering av Batch-kommandon
CommandInstaller	Exekvering av Shell-kommandon
ZipExtractionInstaller	Extrahering av .zip- eller .tar.gz-filer

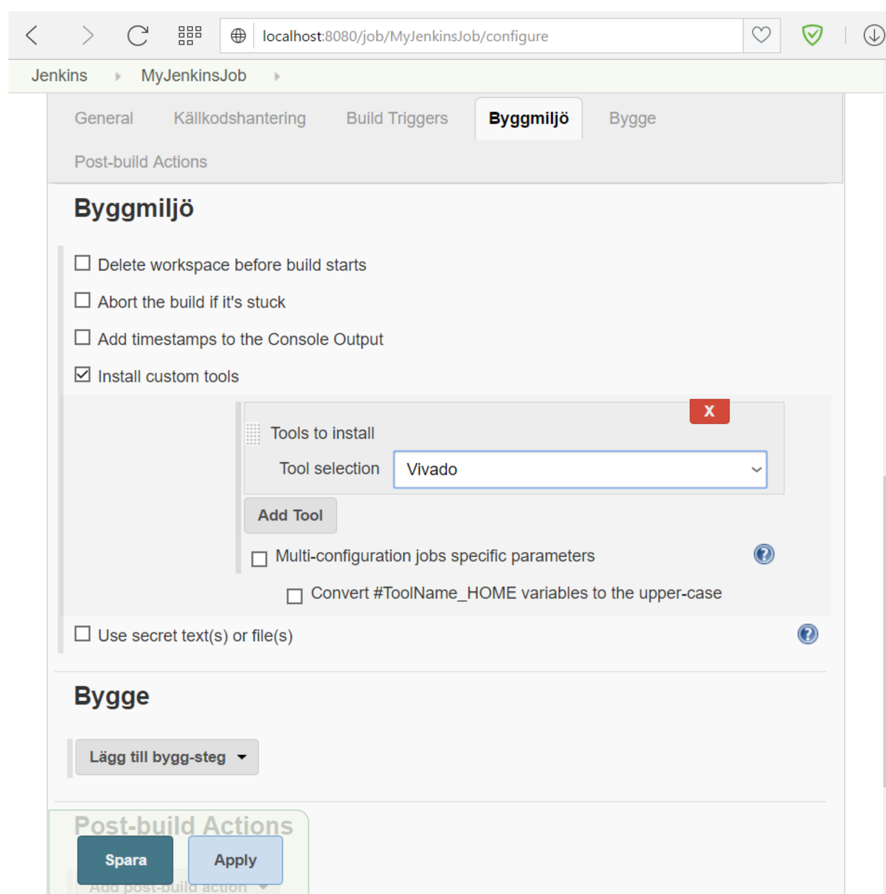
Tabell 2.2: Lista med tool installers som följer med CTP och respektive tool installers beskrivning. [5]

Figur 2.6 visar en skärmdump från Jenkins när ett custom tool konfigureras. Detta custom tool benämns ”Vivado” och har en tool installer (streckat område (1)) som är konfigurerad att hämta ett ZIP-arkiv från den adress som är specificed i ”*Download URL for binary archive*”, även kallat ”*Tool Home*”. ZIP-arkivet innehåller i detta fallet ett program kallat Vivado. Det går att lägga till fler tool installers (se figur 2.6) för att installera ytterligare program, genom att klicka på en knapp ”*Lägg till installerare*” och sedan i listan (2) välja en önskad metod att installera programmet på.



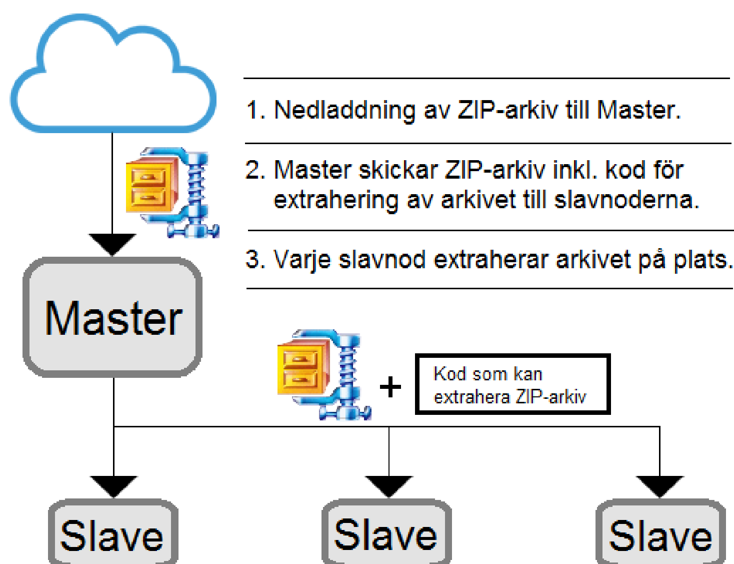
Figur 2.6: Skärmdump av när ett custom tool konfigureras i Jenkins. Det streckade området (1) är den grafiska vyn som en tool installer representerar. (2) är en lista med tillgängliga tool installers som kan läggas till för att installera fler program.

Figur 2.7 visar en skärmdump när ett jobb i Jenkins konfigureras att installera programmet Vivado som konfigurerades i figur 2.6.



Figur 2.7: Skärmdump av när ett jobb konfigureras i Jenkins. Jobbet är här konfigurerat att låta insticksmodulen Custom Tools Plugin installera ett komprimerat program ”Vivado” på anslutna noder.

Diagrammet i figur 2.8 visar en installation av ett program komprimerat i ett ZIP-arkiv som hämtas från sökvägen i *Tool Home* till noden Master. Sökvägen specificeras när tool installern konfigureras i Jenkins. Master överför sedan arkivet och en Java-klass som heter FileCallable [16] till varje ansluten nod. Den uppgift som klassen FileCallable har är att installera programmet som finns i det komprimerade arkivet på noden. Det innebär bland annat att FileCallable måste packa upp det komprimerade arkivet och lägga filerna på rätt ställe i noden. Syftet med det upplägget är att istället för att låta mastern utföra uppackningen och överföra ett stort uppackat program till de anslutna noderna, så är det effektivare att överföra ett mindre komprimerat arkiv med programmet i och en FileCallable som utför uppackningen på respektive nod. Installationstiden blir därför kortare då mängden data som behöver överföras är mindre. Mastern ska i största utsträckning hålla sig fri från uppgifter och låta slavnoderna utföra allt arbete, därför utförs extraheringen på plats hos respektive nod.



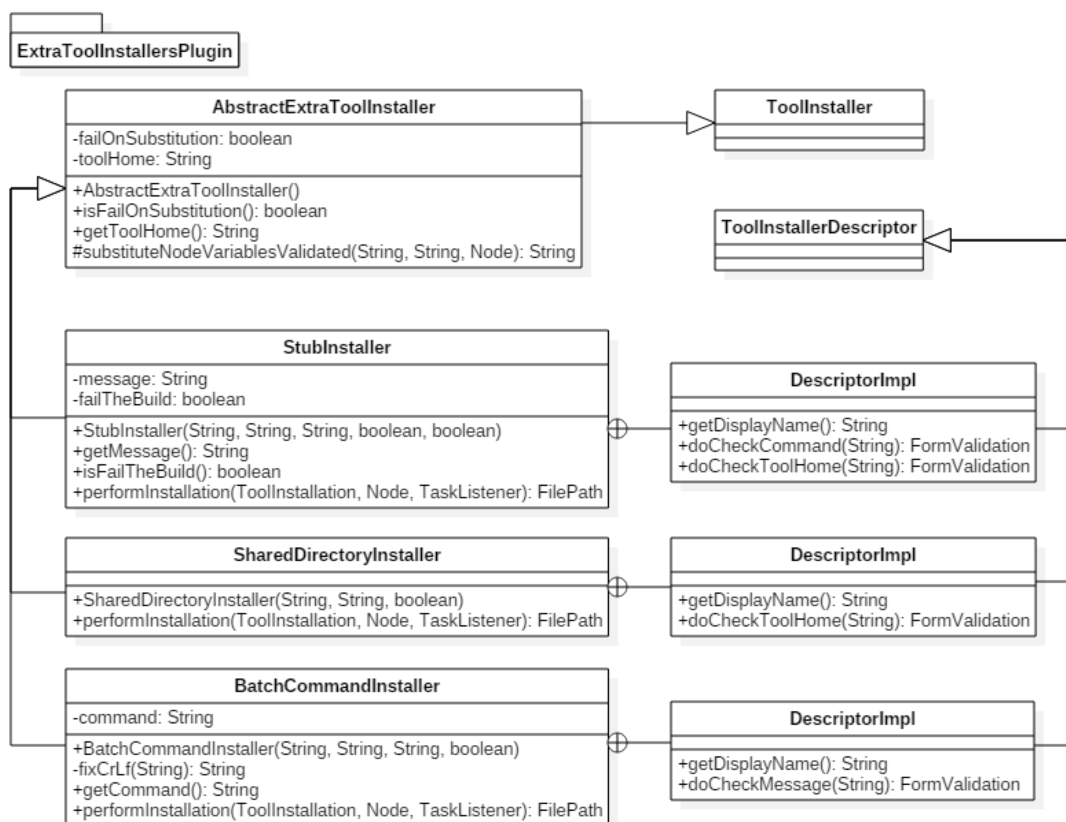
Figur 2.8: Flödesdiagram över en installation av ett ZIP-arkiv på flera anslutna noder.

2.2.2 Extra Tool Installer Plugin

CTP har inledningsvis tre tool installers till förfogande (se tabell 2.2), men om dessa tool installers inte kan användas då de installerar på ett sätt som inte fungerar för användaren så kan Extra Tool Installers Plugin [17] (ETI) tillhandahålla andra tool installers som CTP kan använda.

ETI är en insticksmodul till Jenkins och expanderar CTP genom att tillhandahålla fler tool installers likt de som tillkommer med CTP (se tabell 2.2). Det är huvudsakligen ett paket med tool installers som CTP kan nyttja för att installera program. ETI utför därmed inte själv några installationer.

Insticksmodulen är utvecklad i programmeringsspråket Java.



Figur 2.9: UML-diagram av strukturen i ETI innan integration av RAR-filsuppackaren.

Programmet som har utvecklats under examensarbetet integrerades i ETI och UML-diagrammet i figur 2.9 visar modulens struktur innan integrationen. Inledningsvis innehöll ETI enbart tre tool installers, *StubInstaller, SharedDirectoryInstaller* och *BatchCommandInstaller*. De ärver funktionalitet från den abstrakta klassen *AbstractExtraToolInstaller* [18]. Varje tool installer har en egen inre klass *DescriptorImpl* (se figur 2.9) som definierar en del av den grafiska vyn i Jenkins som en tool installer representerar (se streckat område i figur 2.6). Klassen definierar vyn genom att förse webbgränssnittet med ett beskrivande namn på tool installern och eventuella felmeddelanden i form av String-variabler, en typ av textsträngar.

Repository för ETI finns tillgängligt som open source på GitHub (för beskrivning av GitHub, se kapitel 2.4). [19]

2.3 Utvecklingsmiljöer

I examensarbetet utfördes vidareutveckling av CTP och ETI i två olika utvecklingsmiljöer, Eclipse Kepler och Netbeans. För att bygga och paketera ett Java-projekt användes byggsystemet Maven som producerar filer med ett projekts funktionalitet. Dessa filer användes för att testa den utvecklade mjukvaran i Jenkins.

2.3.1 Eclipse Kepler

Eclipse [20] är en integrerad utvecklingsmiljö (IDE) som modulärt kan utvidgas med insticksmoduler för att utöka funktionaliteten. I examensarbetet användes versionen Kepler. [21] Programmet användes till utveckling av Java-klasser som testade funktionaliteten hos diverse Java-bibliotek som tillhandahåller uppackning av komprimerade arkiv med olika filformat.

2.3.2 Maven

Maven [22] är ett system som förenklar byggen av Java-baserade projekt genom automatisk hantering av beroenden (*dependencies*). När ett projekt måste nyttja någonting från ett bibliotek, t.ex. klasser eller metoder, så kallas det för ett beroende. Istället för att manuellt hitta en lämplig JAR-fil som innehåller de bibliotek ett projekt efterfrågar och sedan importera filen till projektet, så kan Maven detektera, lista och automatiskt importera lämpliga JAR-filer. I följande kodexempel skapas ett beroende av Junrar-biblioteket i den kodrad som är understruken:

```
import com.github.junrar.Archive;
public class ExampleDependencies{
    private Archive myJunrarArchive;

    public ExampleDependencies(File theRarArchive){
        myJunrarArchive = theRarArchive;
    }
}
```

När ett objekt av klassen `Archive` från Junrar-biblioteket deklareras behöver projektet ha tillgång till dess egenskaper och funktionalitet. Maven hämtar då en JAR-fil från adressen `com.github.junrar` där `Archive` finns. JAR-filen innehåller all efterfrågad funktionalitet kring det biblioteket som klassen finns i. Hur Maven ska bygga ett projekt står specificerat i en XML-fil, *pom.xml*, som ska finnas direkt i en projektmapp för att den ska hittas. Se appendix C för examensarbetets pom.xml. En XML-fil är en textfil som enbart beskriver en struktur eller lagring av data. Följande utdrag från XML-filen i appendix C visar hur Maven hanterar ett beroende av biblioteket JUnrar.

```
<dependencies>
    <dependency>
        <groupId>com.github.junrar</groupId>
        <artifactId>junrar</artifactId>
        <version>0.7</version>
    </dependency>
    .........
</dependencies>
```

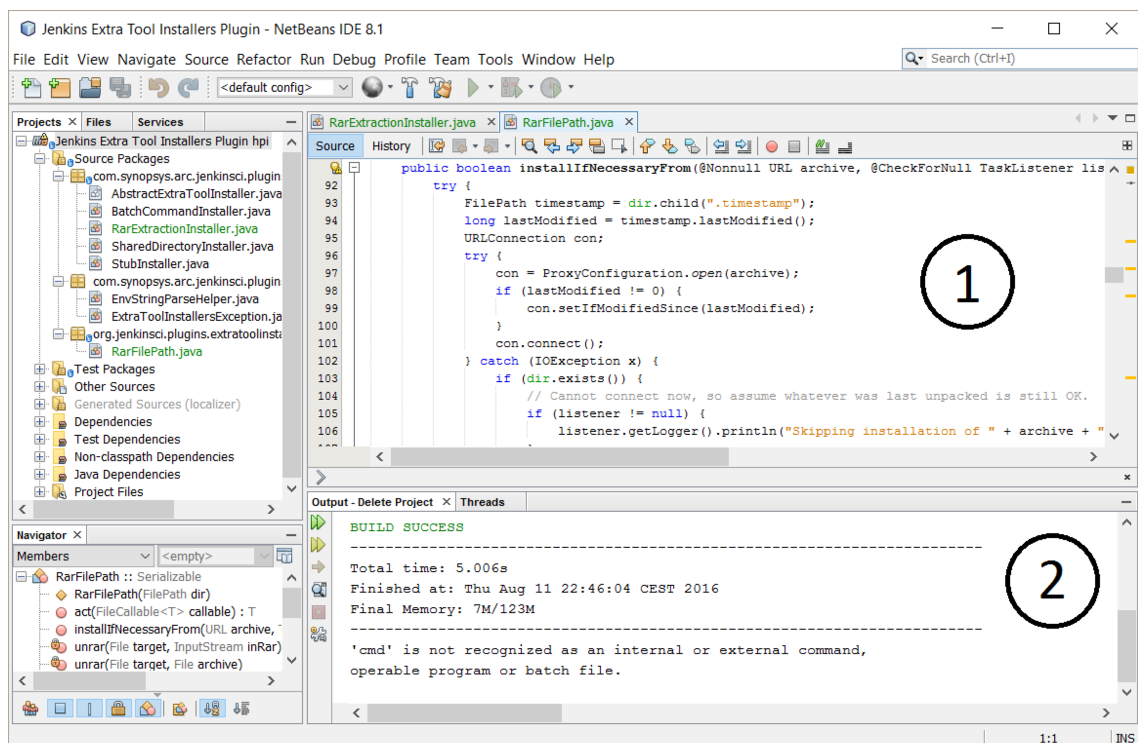
Taggen `groupId` kallas för en gruppidentifierare och specificerar vilken grupp, organisation, företag eller projekt som ett bibliotek tillhör, till exempel `com.github.junrar`. Konventionen för gruppidentifierare är organisationens

omvända domännamn. Gruppidentifieraren `com.github.junrar` betyder att biblioteket finns i ett repository som heter `junrar` och är beläget på `github.com`. Taggen `artifactId` identifierar vilket projekt under gruppidentifierarens repository som beroendet behöver. Taggen `version` specificerar vilken version av projektet som avses då det kan finnas flera versioner tillgängliga. [23]

När ett projekt byggdes med Maven skapades bland annat en HPI-fil (Hemera Photo Image) som innehöll en färdigbyggd insticksmodul av det projekt som byggdes. Den filen användes sedan för att ladda upp och installera insticksmodulen i Jenkins så att funktionaliteten kunde testas. I examensarbetet användes inledningsvis det terminalbaserade byggsystemet Maven där kommandot *mvn package* exekverades i projektmappen för att påbörja ett bygge av projektet. Orsaken till att en terminalbaserad version av Maven användes var att Maven inte var integrerat i Eclipse.

2.3.3 Netbeans 8.1

Under andra hälften av projektet byttes IDE från Eclipse Kepler till Netbeans 8.1 [24]. Anledningen var att förenkla utvecklingsprocessen och frångå det terminalbaserade byggsystemet Maven. Netbeans 8.1 är en av utvecklingsmiljöerna som rekommenderades i Jenkins guide för insticksmodulsutveckling [25] och har byggsystemet Maven integrerat. Figur 2.10 visar en skärmdump av ETI under utveckling i Netbeans. Den vy som är markerad med en etta är kodeditorn och nedanför markerat med en tvåa är en logg från när projektet byggs med Maven.



Figur 2.10: Skärmdump av ETI under utveckling i Netbeans 8.1.

De mest utmärkande skillnaderna mellan Eclipse och Netbeans för examensarbetets vidkommande var att Eclipse kan nyttja moduler som ger tillgång till mer funktionalitet, medan Netbeans inte är lika modulärt men har Maven inbyggt i programmet, vilket var det viktigaste gällande val av IDE i examensarbetet.

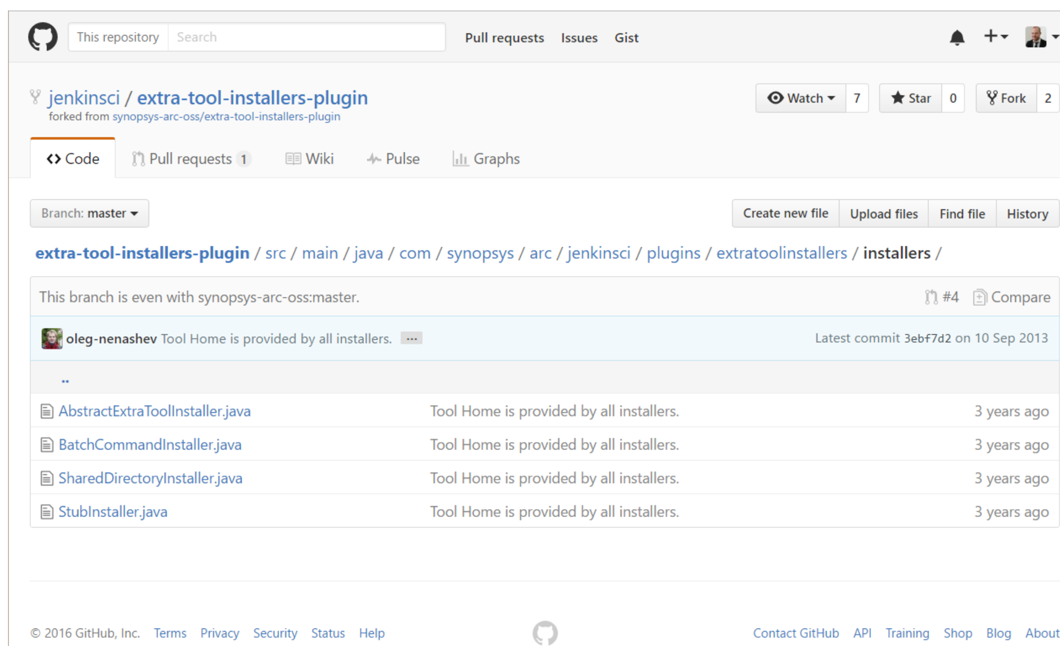
2.4 GitHub

GitHub [26] är ett webbhotell för programmeringsprojekt och använder versionshanteringssystemet Git. Källkod kan delas och lagras öppet i *repositories* [27] i GitHub. Ett repository fungerar som en mapp på GitHub för ett specifikt projekt där projektfiler lagras och varje fils revisionshistorik finns tillgänglig. Repositories kan vara publika eller privata. Under projektet laddades programkod upp till GitHub för att dels kunna finnas tillgänglig för andra utvecklare och dels vara säkerhetskopia.

CTP och ETI finns tillgängliga på GitHub i två separata repository som det går att utföra en ”fork” [28] på, där ett repository är en plats för ett specifikt projekts källkod. Fork innebär att en legal kopia av källkoden skapas i ett eget repository [28] i GitHub som sedan kan vidareutvecklas avskilt från ursprungskoden. Skaparen av en fork har själv kontroll över källkodskopian som kan hanteras hur som helst utan att påverka ursprungskoden. Källkodskopian kan sedan exporteras från GitHub och importeras av det IDE som används för utvecklingen. Därefter kan en vidareutvecklad version av källkodskopian genomgå en granskning innan integrering med ursprungskoden. Granskningen initieras av en ansökan (*Pull Request*) [28] om att få integrera ändringar på ett repository. I granskningen diskuteras eventuella förbättringspunkter innan koden integreras. Under examensarbetet gjordes en fork av ETI som genomgick en Pull Request där koden granskades tillsammans med förvaltaren av CTP och ETI.

Förvaltare av CTP och ETI är en rysk mjukvaruutvecklare som heter Oleg Nenashev. Det innebär att Oleg Nenashev kan ställa krav på den kod som ska integreras innan koden godkänns. Han har även bidragit med handledning genom e-mailkorrespondens och diskussioner i pull requests på GitHub under utvecklingen i examensarbetet.

I figur 2.11 visas en skärmdump av ETIs repository på GitHub tillsammans med Java-klasser som är insticksmodulens olika tool installers.



Figur 2.11: Skärmdump av ETIs repository på GitHub

2.5 Komprimerade filformat

Komprimerade filformat används för att minska filstorleken på datafiler och det finns en mängd av dem med olika funktioner. Vid en installation i Jenkins kan ett filarkiv överföras till noder för att sedan extraheras på plats med hjälp av en uppackningsalgoritm. Nedan beskrivs filformaten 7z och RAR, som undersöktes i examensarbetet.

2.5.1 Filformat: 7z

Formatet

7z-formatet [29] är en öppen arkitektur som kan utökas med ny funktionalitet om behovet uppstår. Ett filarkiv med 7z-formatet innehåller även en uppackare, vilket innebär att om arkivet exekveras så startas en algoritm som extraherar arkivet.

Formatet kan ha en maximal filstorlek på 16 exabyte (se tabell 5.1). 7z är *lossless*, vilket innebär att algoritmer som komprimerar data gör det på ett sätt som inte förstör någon information. All ursprunglig data går att återskapa från ett arkiv som är lossless. [30]

Komprimeringsprogram
7-Zip heter det huvudsakliga komprimeringsprogrammet som hanterar 7z-formatet och är inbyggt i Windows. [30] Eftersom 7z-formatet är en öppen arkitektur finns dess källkod som open source i olika varianter. Källkod för att både komprimera och extrahera filer är open source och finns tillgänglig bland annat i form av JAR-filsbibliotek [31] och GitHub-repository. [32]

7-Zip kan kryptera 7z-arkiv med 256-bitars *Advanced Encryption Standard (AES-256).*

Komprimeringstekniker
7z kan komprimeras med följande komprimeringstekniker: [30]

1. LZMA – LZMA är standardalgoritmen för 7z och är en utökad version av *LZ77* [33].
2. LZMA2 – En utökad version av LZMA. [29]
3. PPMD – En variant av PPMdH, av Dmitry Shkarin.
4. BCJ – En konverterare för 32-bitars körbara x86-filer.
5. BCJ2 – En konverterare likt BCJ.
6. BZip2 – En implementation av Burrows-Wheeler-metoden. [34]
7. Deflate – En algoritm baserad på LZ77.

LZMA-algoritmen står för *Lempel-Ziv-Markov chain-Algorithm* och är standardalgoritmen för 7z-formatet. LZMA är även en utökad version av LZ77-algoritmen som utvecklades på 1970-talet för ordboksbaserad komprimering. Ordboksbaserad komprimering innebär att varje ord i en textsträng ersätts med två koordinater i en ordbok. Dessa koordinaterna har formatet *A/B* där *A* anger sidan i ordboken och *B* anger indexet för ordet på sidan *B*. Tabell 2.3 visar hur en textsträng ser ut före och efter ordboksbaserad komprimering.

Textsträng	Ett	Exempel	På	ordboksbaserad	komprimering
Komprimerad	1/78	6/289	2/85	478/63	385/485

Tabell 2.3: Hur en textsträng komprimeras enligt en ordboksbaserad komprimeringsalgoritm.

De 7z-arkiv som användes i examensarbetet var komprimerade med LZMA-algoritmen. [30]

2.5.2 Filformat: RAR

Formatet
Eugene Roshal är en rysk mjukvaruutvecklare som bland annat utvecklade RAR-formatet. RAR är en akronym som står för Roshal ARchive. [35] RAR-formatet är ett delvis proprietärt filformat, vilket i detta fallet innebär att komprimeringsalgoritmen inte finns tillgänglig som open source. Extraheringsalgoritmen är open source, med villkoret

att komprimeringsalgoritmen inte får återskapas med hjälp av extraheringsalgoritmen. [36]

Formatet kan ha en maximal filstorlek på 9 exabytes (9 miljarder GB) och en minimal filstorlek på 20 bytes (se tabell 5.1). RAR är även lossless. [37]

Komprimeringsprogram

WinRAR är ett program som hanterar både komprimering och extrahering av filer med filformatet RAR och ZIP. Maximal filstorlek som kan komprimeras av WinRAR är ungefär $9 * 10^{18}$ Gigabyte. Utöver RAR och ZIP kan WinRAR utföra extrahering av arkiv med något av följande arkivfilformat: [35]

CAB, ARJ, LZH, TAR, GZ, ACE, UUE, BZ2, JAR, ISO, 7Z eller Z.

Om det finns behov av att komprimera och arkivera stora filer är det möjligt att dela upp arkivet i flera volymer med storlekar som går att specificera. [38] Det beror på att det finns och har funnits begränsningar på filstorlekar som kan kringgås genom att dela upp en fil i mindre volymer. Dessa begränsningar uppkom först på disketter och CD-skivor, där den maximala filstorleken på en CD-skiva inte fick överstiga 2 GB. Begränsningarna lever vidare på fildelningshemsidor än idag.

WinRAR kan kryptera RAR-arkiv med 128-bitars *Advanced Encryption Standard* (AES-128).

Komprimeringstekniker

RAR har två inställningar, *general* och *special*, att välja mellan vid komprimering av filer. Den förstnämnda använder sig av en algoritm baserat på LZSS-algoritmen som kan liknas vid DEFLATE-algoritmen som ZIP nyttjar. Den andra använder en komprimeringsteknik baserad på PPMD-algoritmen, av Dmitry Shkarin, vid komprimering av ”text-like data”. [35]

3 Metod

Kapitlet beskriver examensarbetets arbetsprocess, diskuterar referenser och felkällor samt redogör för validiteten och reliabiliteten hos mätningar utförda i examensarbetet.

3.1 Arbetsprocess

Arbetsprocessen genomfördes enligt en kombination av vattenfallsmodellen [39] och agil utveckling [40]. För att uppnå målen delades processen upp i följande moment:

1. Informationsinhämtning
2. Undersökning av lämpliga filformat
3. Utveckling av uppackningsprogram till utvalda filformat*
4. Mätningar på uppackning av komprimerade filer
5. Val av filformat
6. Integrering av utvalt uppackningsprogram till ETI*
7. Tester och mätningar som bekräftar önskad funktionalitet*
8. Integrering av resultatet till GitHub*

Vid planering av arbetsprocess valdes en sekventiell struktur. Varje delmoment var beroende av resultatet på föregående, vilket resulterade i en sekventiell arbetsmetod. Det gick dock inte att detaljplanera innehållet i varje moment på förhand, därför krävdes flexibilitet för att uppnå önskat resultat inom varje delmoment.

För att kunna bearbeta och hitta en lämplig lösning på de delmoment markerade med en asterisk (*) användes en agil metod som innebär att den mjukvara som skulle utvecklas delades upp i mindre delar. Till exempel kunde först en mjukvaras grafiska utseende utvecklas och testas, därefter kunde fokus sedan övergå till mjukvarans funktionalitet som också utvecklades och testades.

3.1.1 Informationsinhämtning

Inledningsvis behövdes mer information om Jenkins och Custom Tools Plugins struktur och funktionalitet erhållas. Programmet och insticksmodulen installerades på en privat bärbar dator (se tabell 3.2 för systemspecifikation) som var tillgänglig under examensarbetet och sedan genomfördes enklare tester av systemets funktionalitet.

Informationsinhämtning skedde löpande under examensarbetet parallellt med kommande moment.

3.1.2 Undersökning av lämpliga filformat

Därefter undersöktes olika format för komprimerade arkivfiler som kunde vara lämpliga att utveckla stöd för i insticksmodulen Custom Tools Plugin. Undersökningen var uppdelad i två urval. Syftet med uppdelningen var att i det första urvalet välja ut

filformat som gör att målsättning 1 och 3 kan uppnås. Det andra urvalets syfte var att skapa förutsättningar för att kunna uppnå målsättning 2 eftersom det inte var möjligt att veta om installationen skulle ta kortare tid med det nya formatet. Det fanns många filformat att välja mellan i det första urvalet som gjordes i en lista på File-Extensions.org [41]. I det första urvalet valdes filformat ut som uppfyllde följande krav:

1. Kompatibelt med operativsystemen Windows och Linux.
2. Kan ha en maximal komprimerad filstorlek på upp till och med 100 GB .
3. Uppackningsalgoritmen är open source.

Relativt snabbt föll ögonen på RAR [35], 7z [30] och GZIP [42] som undersöktes. Vid närmare undersökning av GZIP visade det sig att enbart enstaka filer kunde komprimeras av GZIP då den inte har möjlighet att arkivera flera filer. Av de filformat i första urvalet skulle de som var mest lämpliga väljas ut. Följande meriterande kriterier användes:

1. Högst populäritet
2. Högst komprimeringsgrad

Efter dessa två urval hade filformaten 7z och RAR valts ut för fortsatta undersökningar och mätningar. Urvalen analyseras i kapitel 4.1.

3.1.3 Utveckling av uppackningsprogram till utvalda filformat

När två lämpliga filformat hade valts ut för fortsatta undersökningar utvecklades två Java-klasser som kunde utföra uppackning av respektive filformat. Syftet var att kunna genomföra mätningar under uppackningen för de två filformaten och även ha en färdig uppackningsmodul som kunde integreras i slutändan. En av funktionerna hos de båda klasserna var att definiera två sökvägar, en till det arkiv som skulle packas upp och en till den plats som arkivets innehåll skulle packas upp till. När sökvägarna var definierade anropades ett Java-bibliotek för respektive filformat som extraherade ett filarkiv utifrån de två sökvägarna. För 7z-formatet användes biblioteket SevenZ [43] från Apache och för RAR-formatet användes JUnrar [44] utvecklat av Edmund Wagner. Apaches bibliotek SevenZ kan hantera 7z-arkiv och JUnrar kan hantera RAR-arkiv. Källkoden för båda biblioteken är open source, med ett undantag för RAR där källkoden finns tillgänglig för uppackning men inte komprimering.

3.1.4 Mätningar på uppackning av komprimerade filer

När klasserna var färdigställda påbörjades mätningarna. De utfördes på uppackning av ett program vid namn Vivado som används på HMS. Programmet var komprimerat och arkiverat i respektive filformat som listas i tabell 3.1. Varje arkiv komprimerades av olika program, men komprimeringsnivån för alla filformat var inställda på ”Normal” för att kunna se vilket format som kunde komprimera mest. Arkivets filstorlek, både okomprimerad och komprimerad samt komprimeringsgraden, visas i tabell 3.1.

Komprimeringsgraden är kvoten mellan den komprimerade och okomprimerade filstorleken.

Arkiv	Vivado2015.4.7z	Vivado2015.4.rar	Vivado2015.4.zip
Okomprimerad filstorlek	6.55 GB	6.55 GB	6.55 GB
Komprimerad filstorlek	2.33 GB	2.74 GB	3.13 GB
Format	7z	RAR	ZIP
Komprimeringsprogram	7-Zip	WinRAR	WinZip
Komprimeringsnivå	Normal	Normal	Normal
Komprimeringsgrad	0.3557	0.4183	0.4778
Innehåll	Programfiler	Programfiler	Programfiler

Tabell 3.1: Beskrivning av arkiven i formaten 7z, RAR och ZIP.

Det som skulle mätas var hur lång tid det tar att packa upp en komprimerad fil som befinner sig lokalt på datorns systemhårddisk. Teknisk systemspecifikation för den dator som utförde uppackningarna finns i tabell 3.2. Motiveringen till varför filen skulle befinna sig lokalt på datorn istället för att hämtas från en filserver i nätverket eller på internet var att man då kan bortse från en eventuellt varierande hastighet på internetuppkopplingen.

Operativsystem:	Windows 10 Pro 64 bitar (10.0, build 10586)
Systemtillverkare:	SAMSUNG ELECTRONICS CO., LTD:
Processor:	Intel Core i7-3635QM CPU @ 2.40 GHz (8 CPUs)
Minne:	8192 MB RAM
Grafikkort:	AMD Radeon HD 8800M Series
Systemhårddisk:	Samsung SSD 840 PRO Series 250 GB

Tabell 3.2: Teknisk systemspecifikation på den bärbara dator som användes under mätningarna i kapitel 3.1.4.

Mätningarna utfördes enligt den mätmetod som presenteras i figur 3.1. Enligt mätmetoden startas ett tidtagarur, extrahering av ett filarkiv utförs, tidtagaruret stoppas och sedan görs en utskrift av mätresultatet. Mätresultatet presenteras i tabell 5.2.

Java-klassen utför följande steg:

1. Skapa ett tidtagarur från ett bibliotek av Apache.
2. Starta tidtagning.
3. Starta extrahering av ett filarkiv. Beroende på filarkivets format används anpassad extraheringskod för respektive format. Tidtagningen sker parallellt med extraheringen.
4. Stoppa tidtagning när extraheringen är utförd.
5. Tidtagaruret visar förfluten tid i millisekunder, därför behöver tiden konverteras till minuter och sekunder. Beräkna förfluten tid i minuter och sekunder.
6. Skriv ut förfluten tid i minuter och sekunder till utskriftskonsolen i den IDE som används.

```
import org.apache.commons.lang.time.StopWatch;
public class MeasureExtraction {
    public static void main(String[] args) {
        StopWatch watch = new StopWatch();// Skapa ett tidtagningsur
        watch.start();                    // Starta tidtagningen

        /*
           Kod som utför uppackning       // Utför extrahering
           av antingen ett 7z-arkiv
           eller RAR-arkiv.
        */

        watch.stop();                     // Stoppa tidtagningen
        long time = watch.getTime();      // Hämta förfluten tid i
                                          //   millisekunder
        long min = time / (1000 * 60);    // Beräkna antal minuter
        long sec = (time / 1000) % 60;    // Beräkna antal sekunder

        System.out.println(               // Utskrift av
                "Förfluten tid: " +       //   förfluten tid
                min + " minuter " +
                sec + " seconds.");
    }
}
```

Figur 3.1: Mätmetod för tidmätning i en Java-klass som extraherar ett filarkiv och sedan skriver ut förfluten tid till utskriftskonsolen i den IDE som används.

Efter att mätningarna hade genomförts upptäcktes dock ett problem med 7z-formatet. Enligt ett dokument [45] om klassen SevenZFile, som användes under mätningarna, så var formatet anpassat för byteordningen Little Endian och inte Big Endian. Little och Big Endian är olika sätt att lagra data i ett minne. Minnet är en samling bytes där en byte är åtta bitar. Om någonting i minnet representeras av 32 bitar, eller fyra bytes, så beror avläsningen av dessa bytes på vilken byteordning som systemet använder. Om dessa bytes har ett hexadecimalt värde på $90AB12CD_{16}$, så kan avläsning enligt Big Endian ses i tabell 3.3 och avläsning enligt Little Endian ses i tabell 3.4.

Adress	1000	1001	1002	1003
Värde	90	AB	12	CD

Tabell 3.3: Minnesrepresentation av det hexadecimala värdet $90AB12CD_{16}$ enligt byteordningen Big Endian.

Adress	1000	1001	1002	1003
Värde	CD	12	AB	90

Tabell 3.4: Minnesrepresentation av det hexadecimala värdet $90AB12CD_{16}$ enligt byteordningen Little Endian.

Det betyder att om ett bibliotek läser av bytes enligt Little Endian i ett system som använder Big Endian, så kommer uppackningen att misslyckas då avläsningen sker i fel ordning.

Det går att packa upp 7z-arkiv i båda byteordningarna om programmet som utför uppackningen tar hänsyn till den byteordning som gäller. Dock har SevenZ-biblioteket inte stöd för Big Endian [45], vilket är nödvändigt då en del *Nix-distributioner använder den byteordningen. 7z-formatet lagrar inte heller behörighetsbitar från filerna som komprimeras. Information om vilka som får använda eller redigera en fil försvinner.

SevenZ-biblioteket var därför inte längre ett alternativ eftersom det inte hanterade Big Endian. Ett bibliotek som hanterade båda byteordningarna var Sevenzip-Jbinding [46]. Det är ett C++-bibliotek som med hjälp av en Java-wrapper [47] kan hantera uppackning av 7z-filer i båda Little och Big Endian.

3.1.5 Val av filformat

Efter mätningarna gjordes teoretiska beräkningar för att undersöka om en kortare extraheringstid är viktigare än en kortare överföringstid i val av filformat. Enligt tabell 3.1 har 7z-formatet en högre komprimeringsgrad och enligt tabell 5.2 har RAR-formatet en kortare extraheringstid. För att undersöka vilken effekt dessa parametrar skulle ha i ett teoretiskt fall, antas att ett 100 GB filarkiv ska skickas över ett nätverk med 100 Mb/s och sedan extraheras. Nedan följer en överslagsberäkning av erfaren överföringshastighet för en arkivstorlek på 100 GB över en anslutning på 100 Mb/s. Antag att vi har en 100 Mbit/s nätverksanslutning som kan utnyttjas till 80%. 80 Mbit/s motsvarar 80/8=10 Mb/s i överföringshastighet. Enligt tabell 3.1 har följande komprimeringsgrader på det komprimerade programmet Vivado erhållits för respektive filformat:

7z: 0,3557
RAR: 0,4183

Uppackningshastigheten kan beräknas med hjälp av tabell 5.2:

7z: 6,55 GB extraherades i genomsnitt på 7 min 53,9 sek = 473,9 sek
RAR: 6,55 GB extraherades i genomsnitt på 7 min 3,6 sek = 423,6 sek

	7z	**RAR**
Arkivstorlek (GB) före komprimering.	100	100
Komprimeringsgrad (enligt tabell 2.3)	0,3557	0,4183
Arkivstorlek (GB) efter komprimering.	35,57	41,83
Effektiv överföringshastighet (Mb/s)	10	10
Resultat – Överföringstid (sek)	3557	4183

Tabell 3.5: Beräknad överföringstid för ett 100 GB stort filarkiv i nätverk med 100 Mbit/s för filformaten 7z och RAR.

	7z	**RAR**
Arkivstorlek test (GB)	6,55	6,55
Uppackningstid (sek) enligt tabell 5.2	473,9	423,6
Uppackningshastighet (Mb/s)	13,82	15,46
Arkivstorlek (GB)	100	100
Uppackningshastighet (Mb/s)	13,82	15,46
Resultat – Uppackningstid (sek)	7235	6467

Tabell 3.6: Beräknad extraheringstid för ett 100 GB stort filarkiv för filformaten 7z och RAR.

	7z	**RAR**
Resultat – Överföringstid (sek)	3557	4183
Resultat – Uppackningstid (sek)	7235	6467
Resultat – Totaltid (sek)	10792	10650

Tabell 3.7: Resultat av tidsåtgång för en installation av ett 100 GB stort filarkiv för respektive filformat i ett nätverk med en effektiv överföringshastighet på 80 Mbit/s.

Tidsskillnaden är förhållandevis liten i denna teoretiska beräkning:

Tidsskillnad: 10792 – 10650 = 142 sek

Det tar 142 sek (2 min 22 sek) kortare tid för det 100 GB stora RAR-arkivet att överföras och extraheras på en nod. Ett snabbare nätverk hade gjort skillnaden större och vice versa med ett långsammare nätverk. Om överföringshastigheten är högre än 100 Mbit/s som antogs i beräkningarna, så har en kortare extraheringstid större inverkan på den totala installationstiden än överföringstiden.

På ett forum för utveckling av Jenkins och dess insticksmoduler fördes diskussioner med Oleg Nenashev om vilket filformat som skulle vara mest optimalt att implementera

stöd för. Han hävdade att RAR-formatet var ett bättre val på grund av att fler program levereras i RAR-formatet (se kapitel 4.1).

Beslutet föll på RAR-formatet på grund av högre populäritet och kortare extraheringstid (för vidare motivering, se kapitel 4.1).

3.1.6 Integrering av utvalt uppackningsprogram

Därefter undersöktes hur den färdiga uppackningsmodulen skulle kunna integreras i den öppna källkoden i Custom Tools Plugin. Oleg Nenashev ansåg att uppackningsmodulen inte var lämplig att placera i koden för CTP, då CTP enbart sköter hanteringen av tool installers som installerar program. Istället skulle en självständig tool installer som använder uppackningsmodulen utvecklas. Den självständiga tool installern skulle sedan integreras med ETI som tillhandahåller fler och andra sorters installationsmoduler till CTP än de som vanligtvis ingår. Motivering till val av implementation presenteras i kapitel 4.2.

Då uppackningsprogrammet inte skulle integreras i CTP genomfördes ytterligare informationsinhämtning kring ETI och hur det skulle kunna utvidgas med det utvecklade uppackningsprogrammet. Med kontinuerlig, men bristfällig och oregelbunden, kontakt med Oleg togs en självständig tool installer till ETI fram som kunde installera program komprimerade i RAR-formatet på noder anslutna till Jenkins-servern.

3.1.7 Tester och mätningar som bekräftar önskad funktionalitet

Nästa moment var att testa koden genom att skapa och konfigurera ett jobb i Jenkins som utförde installation av Vivado-arkivet på vissa noder. Följande noder var anslutna till servern när testet genomfördes:

1. Master (Noden med Jenkins-servern)
2. Slave 1 (Stationär dator med Windows)
3. Slave 2 (Virtuell Ubuntu-distribution på föregående stationära dator)
4. Slave 3 (Stationär dator med Ubuntu-distribution som OS)

Syftet med testet var att utföra en framgångsrik installation av ett RAR-arkiv på noder som har antingen operativsystemet Windows eller en Linux-distribution. Därefter testades uppackning av ett RAR-arkiv med en filstorlek på mer än 100 GB i syfte att kontrollera dess kapacitet. Det stora arkivet bestod av programmet Vivado som hade kopierats flera gånger i syfte att fortfarande ha programfiler som innehåll.

Efter testerna utfördes avslutande mätningar i syfte att jämföra installationstiden mellan den egenutvecklade tool installern, som extraherar RAR-arkiv, och den redan existerande tool installern som extraherar ZIP-arkiv. Resultatet presenteras i tabell 5.3. Likt mätningarna i kapitel 3.1.4 användes programmet Vivado komprimerat i formaten

RAR och ZIP under mätningarna. Båda arkiven var belägna lokalt på systemhårddisken, i syfte att inte behöva ta hänsyn till eventuell nätverkstrafik, och extraherades av respektive tool installer i Jenkins.

3.1.8 Integrering av resultatet till GitHub

Efter testerna skulle koden granskas och förberedas inför en integration med ursprungskoden för ETI på GitHub. Det gjordes genom att skapa en ansökan i form av en Pull Request [28] (se kapitel 2.4) som initierar en granskningsprocess där eventuella förbättringspunkter i koden diskuteras. Oleg Nenashev är förvaltare av ETI och kan därför ställa krav på koden som ska integreras innan en pull request kan godkännas. De krav [48] som Oleg Nenashev ställde på koden för att ansökan skulle godkännas var:

1. Implementationen ska framgångsrikt installera programverktyg komprimerade och arkiverade i filformatet *.rar på anslutna noder.
2. Implementationen ska automatiskt kunna testas med JUnit-tester [49]. Testerna ska framgångsrikt packa upp ett RAR-arkiv på en master-nod och en slave-nod.
3. I Jenkins-vyn för konfigurering av RarExtractionInstaller ska en *configuration round-trip* [50] genomföras för att säkerställa att de inställningar som gjordes, återspeglades korrekt på det konstruerade mobellobjektet.
4. Om fel under uppackning uppstår, ska inte programmet krascha utan istället avge ett felmeddelande som beskriver anledningen till kraschen.

På grund av långa responstider under granskningsprocessen i kapitel 3.1.8 med Oleg Nenashev så har examensarbetet följande status vid inlämning:

1. Fungerande tool installer som kan extrahera RAR-arkiv med storlekar upp till och med 100 GB.
2. Ej fullständiga enhetstester som fortfarande är under utveckling.
3. Källkoden är ej integrerad med ursprungskoden på GitHub.

3.2 Felkällor

Under mätningarna fanns det felkällor som till viss del kan ha påverkat mätresultatet.

Mätningarna i kapitel 3.1.4 och 3.1.7 utfördes lokalt på en dator som innehöll tillräcklig extraheringskod och filarkiv till extraheringen. Orsaken till detta val var att inte behöva hämta arkivet över ett nätverk då nätverkstrafik kan ha stor inverkan på denna typ av mätresultat.

Under alla mätningar var förutsättningarna i övrigt exakt detsamma med avseende på uppstartade program. Likväl kan den lokala datorns interna bakgrundsprocesser ha påverkat mätningarna.

Även gällande komprimeringsgraden på arkiven i tabell 3.1 har målet varit att skapa samma förutsättningar för respektive format. Däremot finns det inställningsparametrar i

komprimeringsprogrammen WinRAR och 7-Zip (se kapitel 2.5.1 och 2.5.2) som kan påverka resultatet. Till exempel inställning av komprimeringsgraden på arkiven i tabell 3.1 sattes värdet till ”normal” på en femgradig skala. Valet av ”normal” komprimeringsgrad i WinRAR kan skilja sig från ”normal” i 7-Zip och kan därmed betraktas som en felkälla.

En annan inställningsparameter som kan påverka mätresultatet är inställning av komprimeringsteknik. I WinRAR finns inget val för detta, men i 7-Zip finns det flera olika komprimeringstekniker att använda. I examensarbetet användes komprimeringstekniken LZMA som är 7z-formatets standardteknik.

Sammanfattning av felkällor:

1. Belastning på datorn som extraherade filarkiv under mätningarna.
2. Val av komprimeringsgrad
3. Val av komprimeringsteknik

3.3 Validitet och Reliabilitet

Kapitlets syfte är att diskutera validiteten och reliabiliteten hos de mätningar som gjordes i kapitel 3.1.4 och 3.1.7.

Validitet och reliabilitet är termer som används i kvantitativ forskning.Validitet innebär att en mätning verkligen mäter det som man avser att mäta. En mätning får inte påverkas av faktorer som kan skada dess pålitlighet. [51] Reliabiliteten anger tillförlitligheten i en mätning. Oberoende på vem som utför mätningarna och hur många gånger mätningarna utförs ska resultatet vara ungefär detsamma. [51]

- Hög reliabilitet garanterar inte hög validitet.
- Hög validitet förutsätter hög reliabilitet.

3.3.1 Examensarbetets validitet

För att uppnå en hög validitet hos mätningarna i kapitel 3.1.4 och 3.1.7 var de arkiv som skulle extraheras belägna lokalt på systemhårddisken på datorn där mätningarna gjordes. Om ett arkiv hade funnits på en filserver skulle arkivet behöva överföras över nätverket till datorn för att sedan extraheras. Då hade inte tiden för extrahering uppmätts, istället hade mätresultatet visat tiden det tar för arkivet att överföras och extraheras. Därmed hade validiteten varit låg då man inte mäter det man avser att mäta.

3.3.2 Examensarbetets reliabilitet

Genom upprepade mätningar kunde en hög reliabilitet säkerställas då mätresultaten inte visade en större variation (se figur 5.1 och figur 5.6).

De arkiv som användes under mätningarna innehöll programfiler (se tabell 3.1) till programmet Vivado som HMS använder i sin verksamhet. Om arkiven hade innehållit bilder i JPEG-format som redan är förkomprimerade [52] hade komprimeringsgraden varit lägre och extraheringstiden längre. Att mäta tiden för extrahering av arkiv som innehåller bilder är inte relevant då det är program som ska installeras och sedan nyttjas i Jenkins.

3.4 Källkritik

De källor som hänvisar till Jenkins egna Wiki-sidor är nödvändiga då Jenkins egen tekniska dokumention [3] inte omfattar den information som berör vidareutveckling av Jenkins och dess insticksmoduler. Utvecklare som läser den tekniska dokumentationen hänvisas även till dessa sidor för information om vidareutveckling. De källor som Jenkins egen tekniska dokumentation faktiskt täcker är officiella Java-dokument [16] [53] om den funktionalitet som redan finns och anses vara trovärdiga.

Oleg Nenashev är en ingenjör inom inbyggda system och även ansedd inom utveckling av Jenkins-CI. Han är tillika förvaltare av källkoden för Custom Tools Plugin och Extra Tool Installers Plugin. [48] [54]

GitHub [26] (se kapitel 2.4) kan versionshantera källkod som är open source. De källor som hänvisar till GitHub är trovärdiga då dess funktionalitet har nyttjats i examensarbetet och fungerar så som källorna beskriver. Följande GitHub-källor användes:

- Custom Tools Plugin [5]
- Extra Tool Installers Plugin [17] [18]
- JUnrar [44]
- SevenZip-JBinding [32] [47]
- Pull Request #4 [48]
- GitHubs terminologi [28]

Angående källor som refererar till en mjukvaras egen hemsida har det varit viktigt att skilja på objektiv fakta och säljinriktad information. Hänsyn har tagits till vad som är vinklad information och vad som är objektiv fakta av det som har hämtats från dessa källor. Dessa källor är följande:

- Eclipse [20] [21]
- Maven [22] [23]
- Netbeans [24]
- SevenZ [31] [45]
- SevenZip-Jbinding [32] [46] [47]
- 7z [29]
- RAR [55] [36]

David Salomon, som har skrivit boken *Data Compression – The Complete Reference*, är en professor på avdelningen för datavetenskap på *California State University*. Det är en omfattande bok med vetenskaplig tyngd som gör boken trovärdig. [30] [33] [34] [35]

Martin Fowler [56] är författare inom området mjukvaruarkitektur med inriktning på bland annat kontinuerlig integration. [1]

The Data Compression Boo, skriven av *Mark Nelson* och *Jean-Loup Gailly*, behandlar olika typer av datakomprimering som vanligtvis används på persondatorer eller medelstora system, inkluderat komprimering av binära program, data, ljud och grafik. Jean-Loup Gailly har även tillsammans med Mark Adler utvecklat *zlib*, en implementation av Deflate. [42]

John Ferguson Smart har inte bidragit med kod till vidareutveckling av Jenkins, utan har istället skrivit boken *Jenkins: The Definitive Guide* [2] för att underlätta för nya användare som vill lära sig mer om Jenkins-CI. Skaparen av Jenkins, Kohsuke Kawaguchi, hänvisar till just Jenkins när han blir tillfrågad om det finns en bok om Jenkins. [57]

Artikeln ”*What exactly is GitHub anyway?*” av *Klint Finley* beskriver bakgrunden till GitHub, dess funktioner och struktur. Den information som hämtades från artikeln anses vara trovärdig då GitHub användes under examensarbetet för säkerhetskopiering och integration till källkod på GitHub. Informationen bekräftades under examensarbetet.

Undersökningen i kapitel 3.1.2 baserades på listan med komprimerade arkivfilformat på *File-Extensions.org*. [41] Listan användes för att veta vilka format som fanns att undersöka, men information om respektive format hämtades inte från File-Extensions.org.

Craig Larman är en specialist inom bland annat iterativ och inkrementerande utveckling. Han har skrivit ett flertal böcker om agil mjukvaruutveckling, däribland *Agile and Iterative Development: A Manager's Guide.* [40]

Conrad Weisert har bland annat undervisat i datavetenskap vid *Loyola University*. I sin artikel *”There's no such thing as the Waterfall approach!”* är han inte positivt inställd till vattenfallsmodellen, men beskriver dess struktur på ett sätt som är lätt att förstå. [39]

4 Analys

I detta kapitel analyseras de krav, urval och resultat som slutade i att ett stöd för RAR-formatet utvecklades. Kapitlet ger en insikt i hur processen var strukturerad och varför samt lyfter fram de val som gjordes som påverkade resultatet.

4.1 Val av filformat

Utifrån målsättning 1 och 3 (se kapitel 1.3) hade en lista med krav formulerats inför det första urvalet i kapitel 3.1.2. Följande krav med respektive motivering togs fram inför det första urvalet av filformat:

1. *Kompatibelt med operativsystemen Windows och Linux.*

   De datorer som används i datorklustret hos HMS har antingen Windows eller Linux som operativsystem. Att filformatet är kompatibelt med både dessa operativsystem är obligatoriskt, annars fungerar inte extraheringen fullt ut.

2. *Kan ha en maximal komprimerad filstorlek på upp till och med 100 GB* .

   Detta krav har sitt ursprung i ett behov hos HMS. Den ursprungliga orsaken till examensarbetet är att de inte har kunnat installera program i arkiv som är större än 8 GB. För att kunna täcka det behovet behövs ett tredje filformat som kan installera program komprimerade i arkiv som är upp till och med 100 GB.

3. *Uppackningsalgoritmen är open source.*

   Uppackningsalgoritmen måste vara open source då källkod för att extrahera ett arkiv måste kunna integreras i tool installern. Slutprodukten ska sedan laddas upp till GitHub som open source.

Dessa krav är nödvändiga för att kunna uppnå målsättning 1 och 3 och användes för att kunna göra det första urvalet av arkivfilformat.

Det andra urvalet syftade till att uppfylla målsättning 2 och skulle skapa förutsättningarna inför mätningarna i kapitel 3.1.4. Syftet med mätningarna är att säkerställa att installation med den i examensarbetet utvecklade tool installern tar kortare tid än med redan existerande tool installers. De filformat som undersöktes i första urvalet rangordnades efter två krav:

1. *Högst komprimeringsgrad*

   En hög komprimeringsgrad medför en mindre filstorlek och därmed en kortare överföringstid när ett arkiv ska hämtas över ett nätverk till anslutna noder. En kortare överföringstid innebär en kortare installationstid, men inte nödvändigtvis en kortare uppackningstid.

2. *Högst populäritet*

   En hög populäritet innebär att filformatet används i stor utsträckning för att komprimera data. Det är då större sannolikhet att Jenkins-användare har stött på filformatet, kan dess funktioner och har ett tillhörande komprimeringsprogram.

De två filformaten som valdes ut för fortsatt undersökning, 7z och RAR, var båda lämpliga kandidater till viss del. De är kompatibla både med Windows och Linux (se tabell 5.1), om det bibliotek som utför uppackningen har stöd för respektive OS. Oleg Nenashev ansåg att RAR var det mest lämpliga filformatet för ändamålet på grund av den populäritet formatet har. Han menade att det finns ett större antal Windows-applikationer som levereras komprimerade i formatet RAR jämfört med 7z.

> ”*The advantage is that there is a big number of Windows tools being delivered by RAR. Hence such installer provides much more added value than 7z.*” – Oleg Nenashev [54]

Valet föll på filformatet RAR dels på grund av kortare extraheringstid (se tabell 5.2) och dels högre populäritet. I detta fallet ansågs en kortare extraheringstid vara viktigare än en högre komprimeringsgrad. Den högre komprimeringsgraden hos 7z medför mindre filer och därmed en kortare överföringstid när filarkivet ska hämtas till mastern och överföras till anslutna noder.

Efter implementationen i kapitel 5.2 genomfördes avslutande mätningar på uppackning av programmet Vivado komprimerat i formaten RAR och ZIP. Mätresultatet i tabell 5.3 visar att den redan existerande ZipExtractionInstaller tar kortare tid än RarExtractionInstaller som utvecklades i examensarbetet. Installation av ett ZIP-arkiv tar ungefär 16 minuter medan en installation av ett RAR-arkiv tar ungefär 19 minuter. Det innebär att en installation med den nya tool installern inte tar kortare tid än den existerande toolinstallation och att målsättning 2 inte är uppfyllt.

4.2 Val av implementation

Den lösning som implementerades i ETI är en självständig tool installer. I figur 5.4 visas hur implementationen ser ut tillsammans med övriga tool installers i ETI. Att den är självständig betyder att den har en egen Java-klass. Den första idén om hur det skulle implementeras var att inte skapa en självständig tool installer, utan att istället integrera uppackaren i den serialiserbara klassen FilePath som redan utför uppackning av ZIP- och TAR.GZ-filer. Genomsökning av existerande kod i syfte att hitta uppackningsalgoritmen ledde till att följande kodutdrag hittades i klassen FilePath:

```
if(archive.toExternalForm().endsWith(".zip")){
    unzipFrom(cis);
}else{
    untarFrom(cis,GZIP);
}
```

Det koden gör är att filändelsen på det specificerade arkivet kontrolleras i syfte att välja ut korrekt tool installer som kan utföra installationen. Om filändelsen slutar på ”.zip” anropas en metod `unzipFrom(cis)` som initierar en uppackning av ZIP-arkiv som variabeln `cis` representerar. Om det inte slutar på ”.zip” så antas det att filändelsen är ”.tar.gz” och en metod `untarFrom(cis,GZIP)` som initierar en uppackning av TAR.GZ-arkiv anropas. Tanken var då att utöka det här kodutdraget med ytterligare en kontroll om filändelsen slutar på ”.rar”. Likt övriga fall så skulle en metod `unrarFrom(cis)` anropas som initierar uppackning av det specificerade RAR-arkivet. Programmeringstekniskt är det en enkel lösning, men rent praktiskt innebär det en stor versionsuppgradering eftersom klassen FilePath berör många insticksmoduler och är även inbäddad i själva Jenkins.

Istället för att göra en versionsuppgradering på hela Jenkins utvecklades istället den lösning som innebar en självständig tool installer. Det finns två fördelar med den varianten. Den ena är att man gör en versionsuppgradering på en enskild insticksmodul och inte hela Jenkins-systemet. Den andra är att det öppnar upp en möjlighet att vidareutveckla ETI ytterligare genom att det blir programmeringstekniskt enklare att utveckla fler tool installers som kan packa upp andra filformat, exempelvis 7z-formatet. I princip så kan det räcka med att kopiera klasserna RarExtractionInstaller och RarFilePath, ändra klassnamn och variabel- och metodbeteckningar som passar det nya formatet och sedan ersätta den kodrad i RarFilePath som anropar Junrars metod `extractArchive(File target, File archive)` med en liknande metod i ett annat bibliotek som utför uppackning av `archive` till `target`.

5 Resultat

Kapitlets syfte är att i kronologisk ordning presentera de resultat som examensarbetet producerat i form av mätresultat och implementation. Inledningsvis presenteras de mätresultat som erhölls före implementationen med syftet att välja ut ett lämpligt filformat. Därefter beskrivs implementationens struktur och funktionalitet. Den avslutande delen lägger fram resultatet från de mätningar som utfördes efter implementationen. Även dessa mätningar mätte förfluten extraheringstid, fast extrahering gjordes med respektive filformats tool installer i Jenkins på Master-noden. Kapitlet har följande översikt:

1. Mätresultat före implementationen
2. Implementation
3. Mätresultat efter implementationen
4. Begränsningar

5.1 Mätresultat före implementationen

Filnamn	Filändelse	Maximal storlek	Minimal storlek	Kompatibel med följande OS (Om stöd finns hos uppackningsbiblioteket som används):	
				Windows	*Nix
RAR [35]	.rar	9 exabytes / ~8 exbibytes	20 bytes	Ja	Ja
7z	.7z	16 exabytes / ~14 exbibytes	Ej specificerad	Ja	Ja
ZIP	.zip	~4 GB	22 bytes	Ja	Ja
TAR.GZ	.tar.gz	~8 GB	Ej specificerad	Ja	Ja

Tabell 5.1: Allmän översikt över formaten RAR, 7z, ZIP och TAR.GZ.

I tabell 5.1 visas en allmän översikt över de två utvalda filformaten, RAR och 7z, som även jämförs med ZIP och TAR.GZ som redan används i Custom Tools Plugin. Uppdraget var att implementera stöd för ett filformat som kunde hantera filer med filstorlekar upp till och med 100 GB eftersom ZIP och TAR.GZ endast kunde arkivera och komprimera filstorlekar på maximalt 4 respektive 8 GB. I tabell 5.1 kan det utläsas att RAR kan hantera en maximal filstorlek på 9 exabytes och 7z kan hantera upp till och med 16 exabytes (1 exabyte = 10^9 Gigabyte).

Mätningarna som redovisas i tabell 5.2 är utförda på den privata bärbara datorn (se tabell 3.2 för systemspecifikation) och varje mätning görs under uppackning av ett program som HMS använder i sin verksamhet, vilket bidrar till relevanta mätresultat. Mätresultatet i tabell 5.2 presenteras även grafiskt i en punktgraf i figur 5.2.

Genomsnittlig uppackningstid beräknades genom summering av respektive uppackningstid i sekunder och därefter dividering med antal genomförda mätningar.

Uppackning av respektive filformat gjordes enligt följande:

Vivado2015.4.7z

- Java-klass som nyttjar Apaches bibliotek SevenZ som kan hantera läsning och uppackning av 7z-filer.

Vivado2015.4.rar

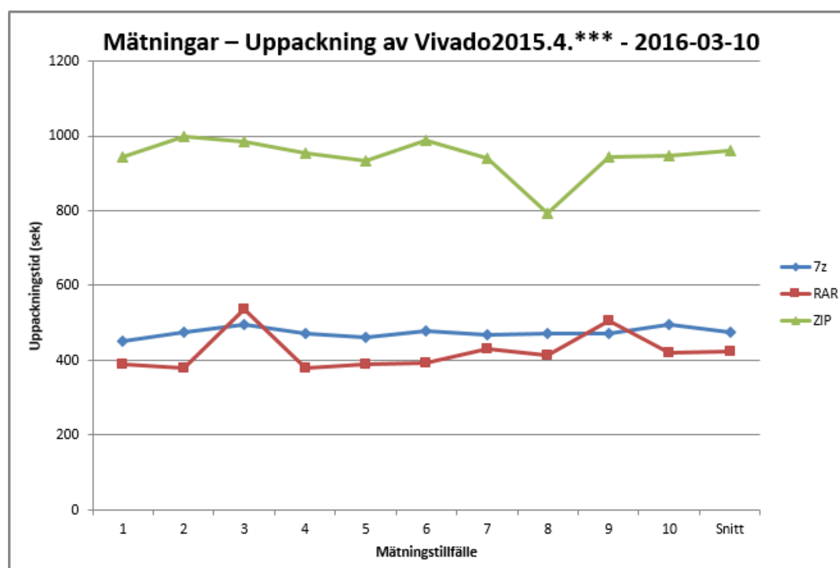
- Java-klass som nyttjar Junrars GitHub-repository med kapacitet att hantera läsning och uppackning av RAR-filer som inte är uppdelade i flera volymer.

Vivado2015.4.zip

- Ett jobb i Jenkins konfigurerat att utföra en installation av ZIP-arkivet på en Master-nod.

Mätningar – Uppackning av Vivado2015.4.*** - 2016-03-10			
Mätning #	**7z**	**RAR**	**ZIP**
1	7 min 31 sek	6 min 29 sek	15 min 42 sek
2	7 min 55 sek	6 min 19 sek	16 min 38 sek
3	8 min 14 sek	8 min 56 sek	16 min 25 sek
4	7 min 51 sek	6 min 19 sek	15 min 54 sek
5	7 min 41 sek	6 min 28 sek	15 min 35 sek
6	7 min 57 sek	6 min 32 sek	16 min 29 sek
7	7 min 49 sek	7 min 12 sek	15 min 39 sek
8	7 min 53 sek	6 min 54 sek	16 min 13 sek
9	7 min 52 sek	8 min 25 sek	15 min 43 sek
10	8 min 16 sek	7 min 02 sek	15 min 46 sek
Genomsnittstid	7 min 53,9 sek	7 min 3,6 sek	16 min 0,4 sek

Tabell 5.2: Uppmätta tider vid uppackning av programmet Vivado2015.4 i formaten 7z, RAR och ZIP (se tabell 3.1). 7z- och RAR-arkivet extraherades med uppackningsprogrammen som utvecklades i kapitel 3.1.3. ZIP-arkivet extraherades med den existerande tool installern som hanterar ZIP-arkiv.

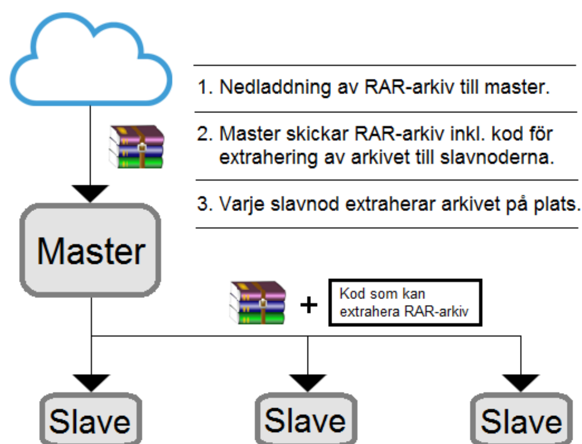


Figur 5.1: Grafisk representation av mätresultatet i tabell 5.2.

5.2 Implementation

Inför implementationen valdes RAR ut som lämpligast filformat. Motiveringen till beslutet beskrivs i kapitel 4.1.

Följande avsnitt refererar till UML-diagrammet i figur 5.2. Resultatet av implementationen är att en ny tool installer som kan packa upp RAR-arkiv har utvecklats. Funktionaliteten att extrahera filarkiv på anslutna noder är densamma som *ZipExtractionInstaller* som utför uppackning ZIP-arkiv, fast med RAR-arkiv istället. Tool installern är uppdelad i två separata Java-klasser, *RarExtractionInstaller* (se Appendix A) och *RarFilePath* (se Appendix B). RarExtractionInstaller ärver av den abstrakta mallen AbstractExtraToolInstaller och ses därför som en Tool Installer som kan hanteras av CTP. Klassen kommer då att fungera som ett gränssnitt i Jenkins där tool installern kan konfigureras inför en installation. När en installation ska påbörjas anropas metoden `performinstallation()` i RarExtractionInstaller med information om vilket program som ska installeras på vilken nod. Metoden kallar sedan på `installIfNecessary()` i klassen RarFilePath som utför själva uppackningen. Uppackningen initieras om verktyget inte redan finns eller om verktyget är installerat men i en äldre version. RarFilePath är en mindre kopia av den redan existerande FilePath [53] som agerar uppackningskod på de noder som behöver extrahera ett RAR-arkiv. Den nya tool installern ska fungera precis som ZipExtractionInstaller förutom att RAR-arkiv ska extraheras istället. Därför behövdes en del av koden från FilePath [58] för att även den nya tool installern ska kunna överföra sin uppackningskod till anslutna noder så att RAR-arkiv kan fjärrextraheras.



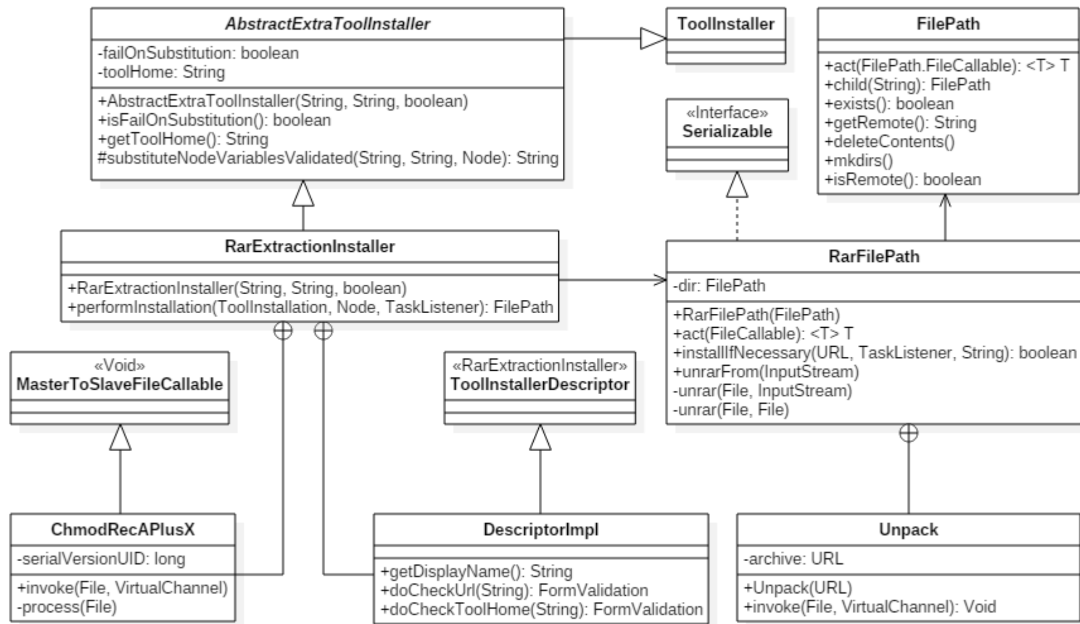
Figur 5.2: Flödesdiagram av examensarbetets lösning.

Syftet med att ha två Java-klasser är att enbart koden för uppackningsprocessen, inte koden som definierar det grafiska utseendet på tool installern, måste kunna överföras till datan. Figur 5.2 visar ett flödesdiagram av hur implementationen fungerar. Konceptet med att flytta uppackningsprocessen ut till respektive nod innebär inte bara att Mastern blir avlastad, men även att ett mindre komprimerat arkiv överförs istället för ett större uppackat program. Överföringstiden blir därför kortare.

För att uppackningen ska kunna utföras fjärrstyrt på andra noder används en implementation av *Filepath.FileCallable* [16] som kan överföras till en nod genom en *VirtualChannel* och exekveras där datan finns. För att en FileCallable ska kunna överföras till en nod måste uppackningsprocessen i RarFilePath vara serialiserbar, därav klassens implementation av *Serializable*. Det är nödvändigt för att uppackningsprocessen ska kunna överföras till andra noder. Klassen

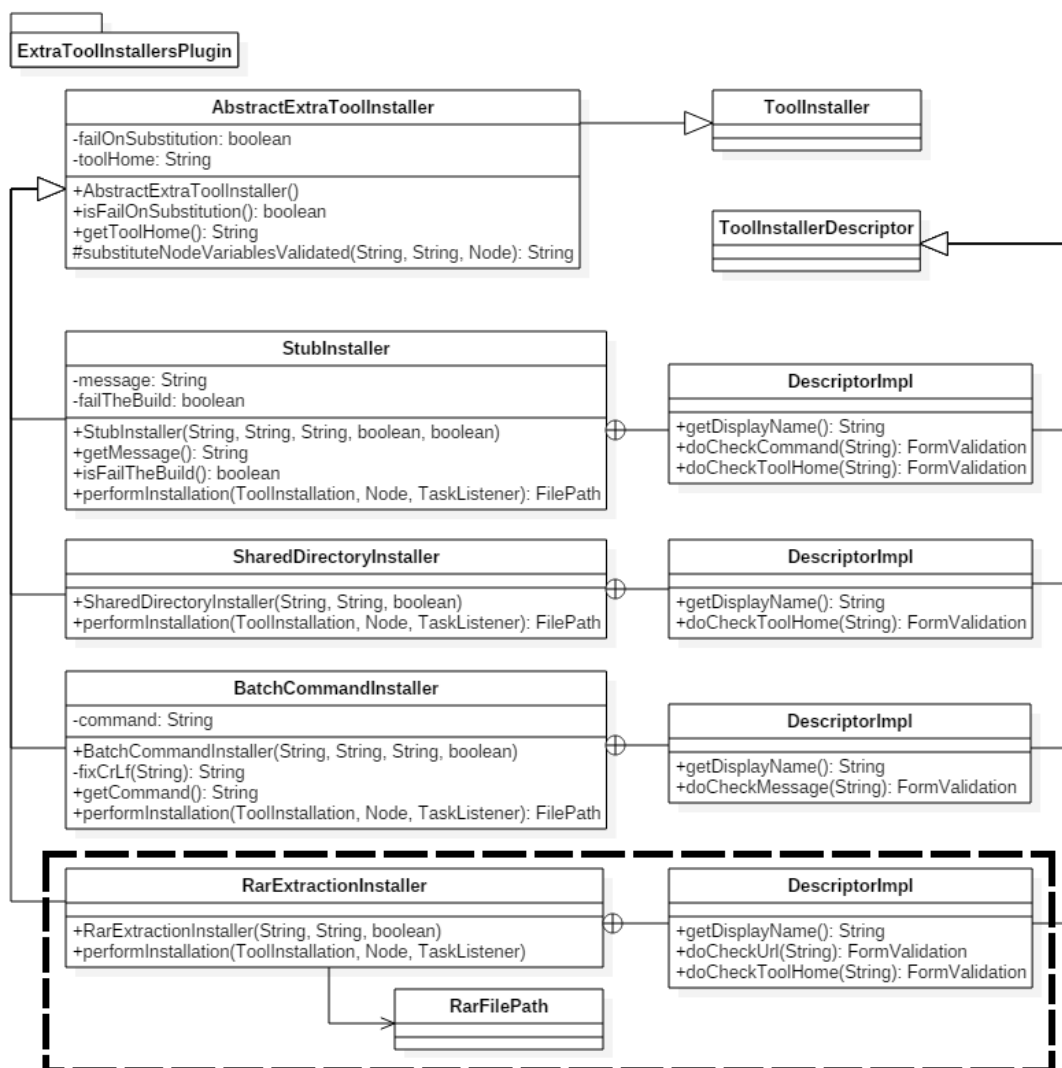
RarExtractionInstaller, gränssnittet för installationsprogrammet, behöver inte överföras och ska därför inte vara seriell.

RarExtractionInstaller har två inre klasser, *DescriptorImpl* och *ChmodRecAPlusX*. DescriptorImpl definierar en del av den grafiska vyn som representerar tool installern i Jenkins. ChmodRecAPlusX ändrar rättigheterna på de uppackade filerna så att de kan exekveras.



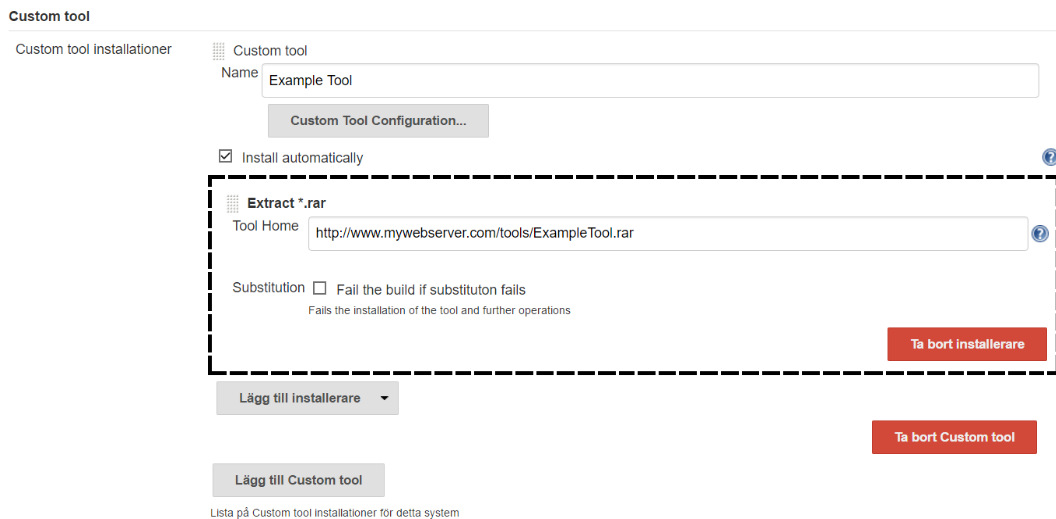
Figur 5.3: UML-diagram över implementationen av RAR-filsuppackaren i ExtraToolInstallersPlugin utan övriga tool installers.

Figur 5.4 visar ett UML-diagram av ETI efter integration med implementationen från figur 5.3. Det streckade området i figur 5.4 innehåller allt som visas i figur 5.3. När ETI installeras i Jenkins finns RarExtractionInstaller med som alternativ och kan konfigureras. Likt övriga tool installers i ETI så ärver RarExtractionInstaller från AbstractExtraToolInstaller och har en inre klass DescriptorImpl som definierar en del av den grafiska vyn.



Figur 5.4: UML-diagram av ETI efter integration av RAR-uppackaren. Det streckade området är en generell representation av implementationen och innehåller allt som visas i UML-diagrammet i figur 5.3.

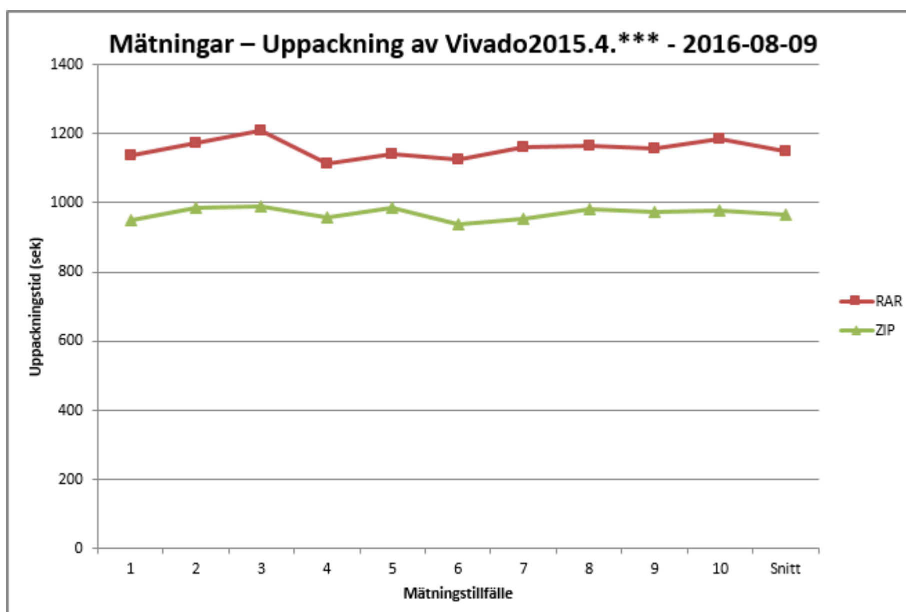
Figur 5.5 visar en skärmdump från när ett Custom Tool konfigureras i vyn Global Tool Configuration i Jenkins. Det streckade området i figur 5.5 visar den grafiska representationen av den implementerade lösningen som har utvecklats. Om ett jobb konfigureras att installera *Example Tool* kommer ett RAR-arkiv hämtas för den URL som har specificerats i fältet *Tool Home*. Arkivet hämtas till Master-noden, sprids ut till övriga anslutna noder och blir därefter uppackad på plats hos respektive nod (se figur 5.2 för flödesdiagram).



Figur 5.5: Skärmdump från vyn Global Tool Configuration där tool installers kan konfigureras. Bilden visar en konfiguration där ett Custom Tool (Example Tool) får i uppgift att installera ett program i form av ett RAR-arkiv från en URL i Tool Home. Den utvecklade tool installern representeras av det streckade området.

5.3 Mätresultat efter implementationen

Efter implementationen i kapitel 5.2 genomfördes avslutande mätningar på uppackning av programmet Vivado komprimerat i formaten RAR och ZIP. Den egenutvecklade tool installern som extraherar RAR-arkiv och den redan existerande tool installern som extraherar ZIP-arkiv användes under mätningarna. Tabell 5.3 och figur 5.6 visar resultatet från mätningarna. Resultatet visar på att den implementerade lösningen inte har en kortare installationstid än den redan existerande tool installern.



Figur 5.6: Grafisk representation av mätresultatet i tabell 5.3.

Mätningar – Uppackning av Vivado2015.4.*** - 2016-08-09		
Mätning #	**RAR**	**ZIP**
1	18 min 56 sek	15 min 49 sek
2	19 min 32 sek	16 min 24 sek
3	20 min 08 sek	16 min 30 sek
4	18 min 35 sek	15 min 59 sek
5	19 min 03 sek	16 min 24 sek
6	18 min 47 sek	15 min 38 sek
7	19 min 22 sek	15 min 52 sek
8	19 min 26 sek	16 min 21 sek
9	19 min 17 sek	16 min 14 sek
10	19 min 47 sek	16 min 18 sek
Genomsnittstid	19 min 9,3 sek	16 min 6,2 sek

Tabell 5.3: Uppmätta tider vid uppackning av programmet Vivado2015.4 i formaten RAR och ZIP. Respektive tool installer i Jenkins användes till uppackningen.

5.4 Begränsningar

De begränsningar som finns i den utvecklade tool installern är att arkiv med flera volymer inte kan packas upp och installeras. Ett stort RAR-arkiv kan delas upp i mindre volymer för lättare hantering. JUnrar-biblioteket kan packa upp ett arkiv som består av flera volymer om de finns tillgängliga i samma mapp, men enbart en fil kan hämtas från sökvägen i *Tool Home* och inte flera filer.

6 Slutsats

Syftet med examensarbetet var att utveckla stöd för större komprimerade installationsfiler hos insticksprogrammet *Custom Tools Plugin*. Insticksprogrammet skulle kunna genomföra en snabb installation av komprimerade filer med storlekar upp till och med 100 GB.

Examensarbetet hade olika målsättningar enligt följande:

1. *”Den utökade insticksmodulen ska kunna genomföra installationer med komprimerade installationsfiler med storlekar upp till och med 100GB, på datorer med operativsystemet Windows eller Linux.”*

   Den egenutvecklade tool installern, RarExtractionInstaller, kan installera program komprimerade i RAR-formatet med filstorlekar på upp till och med 100 GB, vilket uppfyller målet. En uppackningsmodul för RAR-arkiv hade kunnat integreras i ZipExtractionInstaller, men Java-klassen FilePath som utför extrahering av ZIP- och TAR.GZ-arkiv åt ZipExtractionInstaller, är inbyggd i Jenkins källkod. Det hade inneburit en versionsuppgradering av Jenkins. Istället utvecklades en självständig tool installer vars kodstruktur anpassades för att kunna utökas med ytterligare uppackningsmoduler för andra arkivfilformat.

2. *”En installation ska ta mindre tid än vad den gör idag. Genomför mätningar under en installation av verktyget Vivado (som har en ungefärlig filstorlek på 4GB) i syfte att få veta hur lång tid det tar i nuläget.”*

   Enligt mätresultaten i tabell 5.3 kan genomsnittliga installationstider för respektive tool installer jämföras. Resultatet tyder på att en installation med RarExtractionInstaller i kapitel 5.2 inte tar kortare tid än vad den gör med ZipExtractionInstaller som är den existerande tool installern. Därmed är målet inte uppfyllt.

   Orsaken kan vara att uppackningsalgoritmen i JUnrar-biblioteket inte är lika effektiv som uppackningsalgoritmen för ZIP-arkiv. Under mätningarna i kapitel 3.1.4 och 3.1.7 var arkiven belägna lokalt på systemhårddisken för att kunna bortse från någon överföringshastighet. Om arkiven hade befunnit sig på en filserver i det närliggande nätverket hade skillnaden mellan RAR och ZIP gällande överföringshastighet kunnat försummas. Skillnaden i komprimeringsgrad mellan RAR och ZIP är ungefär 6 % (se tabell 3.1)..

3. *”Utökningen ska integreras med insticksmodulens ursprungliga ”repository” på GitHub.”*

   Målet har inte uppnåtts på grund av långsam återkoppling med förvaltaren Oleg Nenashev. Oleg Nenashev är förvaltare av insticksmodulens repository på GitHub och alla ändringar på dess källkod måste genomgå en granskningprocess som initieras av en pull request [28] och behöver således godkännas av honom. Långa svarstider på frågor och funderingar fördröjde integrationen till GitHub. En integration med insticksmodulens källkod på GitHub kan dock ses som ett sätt att sprida resultatet på, och är inte resultatet i sig. Målet är fortfarande aktuellt, men kommer att behöva mer tid för att bli uppnått.

För att kunna uppfylla syftet och de uppsatta målen formulerades frågeställningarna nedan. Följande avsnitt avser att **besvara frågeställningarna** i kapitel 1.4, numrerade efter respektive frågeställning.

1. *”Vilket filformat som stödjer komprimerade installationsfiler med storlekar upp till och med 100 GB är bäst lämpat att implementera stöd för i Custom Tools Plugin?”*

   Filarkiv med 7z-formatet kunde bli komprimerade mer än de med RAR-formatet, vilket medför att tiden för att överföra arkivet till slavnoderna blir kortare. Dock går det att minska den överföringstiden med bättre nätverksutrustning. Det tog längre tid för 7z-arkivet att extraheras, än vad det tog för RAR-arkivet, enligt mätresultaten i tabell 5.2. För att minska uppackningstiden måste algoritmen effektiviseras alternativt bytas ut, men det är för komplext i jämförelse med att investera i bättre nätverksutrustning. Filformatet RAR ansågs vara bäst lämpat att utveckla en ny tool installer för när implementeringen påbörjades på grund av dess populäritet [54] och kortare installationstid (se tabell 5.2). Se kapitel 4.1 *Val av filformat* för motivering.

2. *”Installationen ska ta mindre tid än i nuläget. Hur lång tid tar den nu?”*

   En installation av programmet Vivado tar ungefär 16 minuter (se tabell 5.2 och tabell 5.3) med den redan existerande tool installern ZipExtractionInstaller (se tabell 3.1) som extraherar *.zip-filer. Installationen genomfördes lokalt på Mastern där även arkiven var belägna. Ingen hänsyn behövde därför tas till någon överföringtid.

3. *”Vilka faktorer påverkar tiden för en installation?”*

Följande faktorer påverkar tiden för en installation, rangordnade efter högst inverkan på installationstiden:

1. Uppackningshastigheten hos algoritmen som extraherar arkivet.
2. Arkivets komprimeringsgrad:
   En hög komprimeringsgrad medför en mindre filstorlek som i sin tur medför en kortare överföringstid. Däremot blir uppackningstiden längre då processorn måste utföra fler beräkningar för att återskapa den ursprungliga datan under en extrahering.
3. Processorkapaciteten hos noden som utför extraheringen.
4. Nätverksutrustningens överföringshastighet.

4. *”Hur kan man påverka dessa faktorer för att minska tiden för en installation?”*

Uppackningshastigheten hos den algoritm som extraherar arkivet är svår att påverka. Att effektivisera uppackningsalgoritmen kräver djupare teknisk kunskap inom området för datakompression. Det kan vara möjligt att effektivisera algoritmen, men det är svårt och det finns i slutändan en gräns för hur effektiv en uppackningsalgoritm kan bli.

Det går att konfigurera komprimeringsgraden när programmet ska komprimeras i komprimeringsprogrammet för filformatet. Avvägningar kring komprimeringsgraden måste göras då en hög komprimeringsgrad medför kortare överföringstid och en längre uppackningstid, medan en låg komprimeringsgrad medför en längre överföringstid och en kortare uppackningstid.

En ny och mer kraftfull processor till en nod kan korta ner uppackningstiden. Dock kan det innebära höga kostnader vid en sådan investering.

Uppgradering av nätverksutrustning för en ökad överföringshastighet kan också innebära höga kostnader. Med för stora arkiv kan det korta ner installationstiden avsevärt.

5. *”Hur lång tid tar installationen med det nya filformatet?”*

Installationen med RarExtractionInstaller tar i genomsnitt 19 minuter i jämförelse med ZipExtractionInstaller som i genomsnitt tar 16 minuter (se tabell 5.3). Examensarbetets lösning tar därmed inte kortare tid än den redan existerande tool installern ZipExtractionInstaller. Installationen genomfördes lokalt på Mastern där även arkiven var belägna. Ingen hänsyn behövde därför tas till någon överföringtid. Den längre installationstiden kan bero på att uppackningsalgoritmen i JUnrar-biblioteket inte är lika effektiv som uppackningsalgoritmen för ZIP-arkiv.

6. *”Vad krävs för att resultatet ska bli integrerat i GitHub?”*

   De krav som ställdes för att implementationens pull request (se kapitel 2.4) till ETI skulle godkännas var följande:

   1. *Implementationen skall framgångsrikt installera program komprimerade och arkiverade i filformatet *.rar på anslutna noder.*

      Kravet uppfylldes under testerna i kapitel 3.1.7 där implementationen framgångsrikt installerade ett program komprimerat i ett RAR-arkiv på anslutna noder som hade antingen operativsystemet Windows eller en Linux-distribution.

   2. *Implementationen ska automatiskt kunna testas med JUnit-tester. Testerna ska framgångsrikt packa upp ett RAR-arkiv på en master-nod och en slavnod.*

      Kravet är inte uppfyllt på grund av tidsbrist i integrationen till GitHub. JUnit-testerna var under utveckling i vid inlämning av examensarbetet, men kommer att färdigställas efter inlämning.

   3. *I Jenkins-vyn för konfigurering av RarExtractionInstaller ska en configuration round-trip genomföras för att säkerställa att de inställningar som gjordes, återspeglades korrekt på det konstruerade mobellobjektet.*

      Kravet är inte uppfyllt på grund av tidsbrist i integrationen till GitHub. Ett test av *configuration round-trip* var under utveckling vid inlämning av examensarbetet, men kommer att färdigställas efter inlämning.

   4. *Om fel under uppackning uppstår, ska inte programmet krascha utan stället avge ett felmeddelande som beskriver anledningen till kraschen.*

      Om fel uppstår under installation av ett program skrivs felmeddelandet ut i loggen för jobbet som kör installationen.

7. *”Hur fungerar kontinuerlig integration?”*

   Kontinuerlig integration innebär att när en mindre modul är klar så integreras den i den blivande slutprodukten, testas och distribueras. Därefter påbörjades nästa modul som utvecklas, integreras, testas och distribueras (se figur 1.1). Syftet är att inte behöva genomgå en lång integrationsfas i den avslutande delen av ett mjukvaruprojekt. [1]

Resultatet i kapitel 5.1 består av mätningar och undersökningar som innan implementation indikerade att RAR-formatet var det bättre formatet att implementera stöd för. Utifrån detta resultatet kan vissa slutsatser dras. Eftersom installationstiden inte blev kortare än vad den var förut rekommenderas det att, vid installation av mindre program, använda sig av nuvarande ZipExtractionInstaller (se kapitel 2.2.1 CTP). Den lösning som implementerades i ETI är en tool installer som är kapabel till att genomföra installation av komprimerade RAR-arkiv med filstorlekar på mer än 100 GB, men en installation av ett mindre program i RAR-formatet som hade kunnat hanteras av ZipExtractionInstaller tar längre tid (se tabell 5.3). Implementationen kan nyttjas av användare hos Jenkins, CTP och ETI inom kontinuerlig integration när ett behov uppstår av att installera stora program på noder i ett datorkluster.

Vid en eventuell vidareutveckling av ETI där en tool installer för ytterligare ett format behövs kan examensarbetets arbetsprocess (se kapitel 3.1) underlätta val av filformat.

I sin verksamhet kan HMS installera större program än vad som tidigare varit möjligt i de datorkluster som används i den kontinuerliga integrationen.

6.1 Framtida utvecklingsmöjligheter

Den implementerade lösningen finns tillgänglig för vidareutveckling då källkoden är open source och tillgänglig på GitHub, se kapitel 2.2.2. Eftersom RAR-filsuppackaren är en självständig tool installer och modulärt utvecklad med avseende på var själva uppackningsalgoritmen exekveras, så kan det räcka med att ersätta en kodrad i RarFilePaths metod `unrar(File target, File archive)` med ett annat bibliotek som tar in två File-parametrar och utför uppackning från arkivet `archive` till destinationen `target`. Det innebär att det finns goda möjligheter att integrera stöd för många fler filformat, exempelvis 7z.

7 Terminologi

IDE	Integrated Development Environment. Utvecklingsmiljö som vanligtvis tillhandahåller kodredigerare, kompilator och debugger.
Debugger	Program som underlättar sökning och rättning av fel i programkod.
Bibliotek	I detta fallet ett Java-bibliotek som ger tillgång till Java-klasser med funktionalitet som eftersöks.
Vivado	Ett program som HMS använder i sin verksamhet. I examensarbetet används programmet som en del av de mätningar som genomfördes i form av ett komprimerat filarkiv.
***Nix**	Operativsystem baserade på Unix, till exempel Linux.
Jobb	Projekt eller byggen i Jenkins som kan konfigureras att utföra olika uppgifter på specifika anslutna noder.
Nod	En nod är en klientinstans som, om ansluten till Jenkins-server, kan utföra jobb och avlasta mastern. Datorer kan koppla upp sig mot noder skapade av Jenkins för att kunna konfigureras och ta del av den belastning som finns på servern.
Master	Mastern är en grundinstallation av Jenkins som hanterar alla jobb som körs på Jenkins-servern. Den hanterar alla förfrågningar av http-protokollet och jobb som körs. Om masterns begränsade minnesresurser sinar kan belastningen fördelas på slavnoder.
Slave	En slave/slav/slavnod är en nod i Jenkins med syftet att avlasta mastern från den belastning som jobben skapar.
Tool Installer	En Java-klass som hanteras antingen direkt av Custom Tools Plugin (se kapitel 2.2.1) eller indirekt av Custom Tools Plugin genom Extra Tool Installers Plugin (se kapitel 2.2.2). Dess funktion är att kunna installera program på angivna anslutna noder.
Custom Tool	En konfiguration i Jenkins och hanteras av Custom Tools Plugin (CTP). Ett Custom Tool är en förinställd funktion för att installera ett eller flera program på under byggen av jobb i Jenkins.

OS	Förkortning för ”operativsystem”.
GitHub	Ett webbhotell med stöd för versionshantering där källkod lagras i så kallade repositories.
Repository	Ett mjukvaruprojekts mapp på GitHub som innehåller projektfiler, dokumentation och lagrar varje fils revisionshistoria.
Open Source	Open source betyder att källkoden till en viss mjukvara finns tillgänglig i den mån som mjukvarans licens definierar.
URL	Uniform Resource Locator – En webbadress (länk) som är en sökväg till en resurs på nätet, till exempel en hemsida eller en fil.

Referenser

[1] Fowler, Martin. Continous Integration. *Martin Fowler.* [Online] den 1 Maj 2006. [Citat: den 15 Augusti 2016.] http://martinfowler.com/articles/continuousIntegration.html.

[2] John Ferguson Smart. From Hudson to Jenkins: A short history. *Safari Books Online.* [Online] Juli 2011. [Citat: den 15 Augusti 2016.] https://www.safaribooksonline.com/library/view/jenkins-the-definitive/9781449311155/ch01s04.html.

[3] Dokumentation. *Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://jenkins.io/doc/.

[4] HMS Industrial Networks - Företagsfakta. *HMS Industrial Networks.* [Online] [Citat: den 15 Augusti 2016.] http://www.hms-networks.com/about.

[5] Custom Tools Plugin. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Custom+Tools+Plugin.

[6] Plugins. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Plugins.

[7] Meet Jenkins. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins.

[8] Distributed builds. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#.

[9] Have Master launch slave agent via SSH. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Havemasterlaunchslaveagentviassh.

[10] Have Master launch slave agent on Windows. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-

ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-HavemasterlaunchslaveagentonWindows.

[11] Write your own script to launch Jenkins slaves. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-WriteyourownscripttolaunchJenkinsslaves.

[12] Launch slave agenet via Java Web Start. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-LaunchslaveagentviaJavaWebStart.

[13] Launch slave agent headlessly. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Launchslaveagentheadlessly.

[14] Groovy. *Groovy Lang.* [Online] [Citat: den 23 Augusti 2016.] http://www.groovy-lang.org.

[15] Custom Tools Plugin. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://github.com/jenkinsci/customtools-plugin.

[16] Javadoc - FilePath.FileCallable. *Javadoc - Jenkins.* [Online] [Citat: den 15 Augusti 2016.] http://javadoc.jenkins-ci.org/hudson/FilePath.FileCallable.html.

[17] Extra Tool Installers Plugin. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Extra+Tool+Installers+Plugin.

[18] Extra Tool Installers Plugin - Installers. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://github.com/jenkinsci/extra-tool-installers-plugin/tree/master/src/main/java/com/synopsys/arc/jenkinsci/plugins/extratoolinstallers/installers.

[19] Extra Tool Installers Plugin. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://github.com/jenkinsci/extra-tool-installers-plugin.

[20] Eclipse IDE. *Eclipse.* [Online] [Citat: den 15 Augusti 2016.] https://eclipse.org/ide/.

[21] Eclipse Kepler. *Eclipse.* [Online] [Citat: den 15 Augusti 2016.] https://eclipse.org/kepler/.

[22] What is Maven? *Maven.* [Online] [Citat: den 15 Augusti 2016.] https://maven.apache.org/what-is-maven.html.

[23] Dependencies. *Maven.* [Online] [Citat: den 15 Augusti 2016.] https://maven.apache.org/pom.html#Dependencies.

[24] Netbeans Features. *Netbeans.* [Online] [Citat: den 15 Augusti 2016.] https://netbeans.org/features/index.html.

[25] Plugin Tutorial - Netbeans. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Plugin+tutorial#Plugintutorial-NetBeans.

[26] GitHub. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://github.com.

[27] Finley, Klint. What exactly is GitHub anyway? *Tech Crunch.* [Online] den 14 Juli 2012. [Citat: den 15 Augusti 2016.] https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/.

[28] GitHub Glossary. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://help.github.com/articles/github-glossary.

[29] 7-Zip. [Online] [Citat: den 15 Augusti 2016.] http://www.7-zip.org/7z.html.

[30] LZMA and 7-Zip. [bokförf.] David Salomon. *The.* Fourth Edition. u.o. : Springer, 2007, 3.24.

[31] Apache Commons Compress - SevenZ Jar Library. *Java2s.* [Online] [Citat: den 15 Augusti 2016.] http://www.java2s.com/Code/Jar/c/Downloadcommonscompress15jar.htm.

[32] Sevenzip-JBinding. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://github.com/borisbrodski/sevenzipjbinding.

[33] Dictionary Methods. [bokförf.] David Salomon. *Data Compression - The Complete Reference.* Fourth Edition. u.o. : Springer, 2007, 3.

[34] The Burrows-Wheeler Method. [bokförf.] David Salomon. *Data Compression - The Complete Reference.* Fourth Edition. u.o. : Springer, 8.1.

[35] RAR and WinRAR. [bokförf.] David Salomon. *Data Compression - The Complete Reference.* Northridge, CA : Springer, 2007, Vol. Fourth Edition, 3.20, ss. 226-228.

[36] RAR License. *RARLab.* [Online] [Citat: den 15 Augusti 2016.] http://www.rarlab.com/license.htm.

[37] RAR vs ZIP. *Diffen.* [Online] [Citat: den 23 Augusti 2016.] http://www.diffen.com/difference/RAR_vs_ZIP.

[38] Useful information about multi-volume archives. *File-Extensions.* [Online] [Citat: den 15 Augusti 2016.] http://www.file-extensions.org/article/useful-information-about-multi-volume-archives.

[39] Weisert, Conrad. Waterfall. *Idinews.* [Online] den 8 Februari 2003. [Citat: den 15 Augusti 2016.] http://idinews.com/waterfall.html.

[40] Agile. [bokförf.] Craig Larman. *Agile and Iterative Development: A Manager's Guide.* 3, ss. 25-26.

[41] Common Archives. *File-Extensions.org.* [Online] [Citat: den 30 Augusti 2016.] http://www.file-extensions.org/common/archive.

[42] Deflate: Zip and Gzip. [bokförf.] David Salomon. *Data Compression - The Complete Reference.* Fourth Edition. u.o. : Springer, 2007, 3.23, ss. 230-232.

[43] Commons Compress - SevenZ. *Apache.* [Online] [Citat: den 15 Augusti 2016.] https://commons.apache.org/proper/commons-

compress/apidocs/org/apache/commons/compress/archivers/sevenz/packag e-summary.html.

[44] JUnrar. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://github.com/edmund-wagner/junrar.

[45] Commons Compress - SevenZFile. *Apache.* [Online] [Citat: den 15 Augusti 2016.] https://commons.apache.org/proper/commons-compress/javadocs/api-1.10/org/apache/commons/compress/archivers/sevenz/SevenZFile.html.

[46] Sevenzip-JBind. *Sourceforge - Sevenzip-JBind.* [Online] [Citat: den 15 Augusti 2016.] http://sevenzipjbind.sourceforge.net.

[47] SevenZip-JBinding. *GitHub.* [Online] [Citat: den 15 Augusti 2016.] https://github.com/borisbrodski/sevenzipjbinding/tree/master/jbinding-java/src/net/sf/sevenzipjbinding.

[48] Nenashev, Oleg. Pull Request #4. *GitHub.* [Online] den 15 Juli 2016. [Citat: den 15 Augusti 2016.] https://github.com/jenkinsci/extra-tool-installers-plugin/pull/4#issuecomment-232974228.

[49] Unit Test. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Unit+Test.

[50] Unit Test - Configuration Round-trip Testing. *Wiki - Jenkins-CI.* [Online] [Citat: den 15 Augusti 2016.] https://wiki.jenkins-ci.org/display/JENKINS/Unit+Test#UnitTest-Configurationroundtriptesting.

[51] Validitet och reliabilitet. *Infovoice.* [Online] den 13 03 2002. [Citat: den 22 Augusti 2016.] http://infovoice.se/fou/bok/10000035.shtml.

[52] Nelson, Mark och Gailly, Jean-Loup. The Data Compression Book. *University of Bahrain - College of Applied Studies.* [Online] [Citat: den 23 Augusti 2016.] http://staff.uob.edu.bh/files/781231507_files/The-Data-Compression-Book-2nd-edition.pdf.

[53] Javadoc - FilePath. *Javadoc.* [Online] [Citat: den 15 Augusti 2016.] http://javadoc.jenkins-ci.org/hudson/FilePath.html.

[54] Nenashev, Oleg. Custom-Tools Plugin: Add support for 7z-files. *Google Groups.* [Online] den 30 April 2016. [Citat: den 15 Augusti 2016.] https://groups.google.com/d/msg/jenkinsci-dev/74aZNSlm3Yg/jb4ZaJSKCwAJ.

[55] RAR File. *RARLab.* [Online] [Citat: den 15 Augusti 2016.] http://www.rarlab.com/rar_file.htm.

[56] About Martin Fowler. *MartinFowler.com.* [Online] [Citat: den 30 Augusti 2016.] http://martinfowler.com/aboutMe.html.

[57] [bokförf.] John Ferguson Smart. *Jenkins: The Definitive Guide.*

[58] FilePath.java. *GitHub.* [Online] [Citat: den 30 Augusti 2016.] https://github.com/jenkinsci/jenkins/blob/master/core/src/main/java/hudson/FilePath.java.

[60] Dokumentation. *Jenkins.* [Online] [Citat: den 22 Augusti 2016.] https://jenkins.io/doc/.

Appendix

A Källkod: RarExtractionInstaller.java

```
/*
 * The MIT License
 *
 * Copyright (c) 2016, Martin Hjelmqvist
 * Copyright (c) 2009, Sun Microsystems, Inc.
 * Permission is hereby granted, free of charge, to any person
obtaining a copy
 * of this software and associated documentation files (the
"Software"), to deal
 * in the Software without restriction, including without limitation
the rights
 * to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell
 * copies of the Software, and to permit persons to whom the Software
is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN
 * THE SOFTWARE.
 */
package
com.synopsys.arc.jenkinsci.plugins.extratoolinstallers.installers;

import hudson.Extension;
import hudson.FilePath;
import hudson.ProxyConfiguration;
import hudson.Functions;
import hudson.model.Node;
import hudson.model.TaskListener;
import hudson.remoting.VirtualChannel;
import hudson.tools.ToolInstallation;
import hudson.tools.ToolInstallerDescriptor;
import hudson.util.FormValidation;
import jenkins.MasterToSlaveFileCallable;
import java.io.File;
import java.io.IOException;
import java.net.HttpURLConnection;
```

```java
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import org.apache.commons.lang.StringUtils;
import org.jenkinsci.plugins.extratoolinstallers.util.RarFilePath;
import org.kohsuke.stapler.DataBoundConstructor;
import org.kohsuke.stapler.QueryParameter;

/**
 * Installs a tool by downloading and unpacking a RAR file.
 * The installer is inspired by {@link ZipExtractionInstaller}.
 * Limitation: Extracts only a single RAR file. A tool archive split
into multiple RAR volumes will be not fully extracted.
 *
 * @author Martin Hjelmqvist <martin@hjelmqvist.eu>.
 */
public class RarExtractionInstaller extends AbstractExtraToolInstaller
{

    /**
     * Messages displayed when configuring the tool in Jenkins.
     * TODO: Move this to Messages.java generated by the localizer.
     */
    private static final String RAR_EXTRACTION_INSTALLER_DISPLAY_NAME
= "Extract *.rar";
    private static final String
RAR_EXTRACTION_INSTALLER_BAD_CONNECTION = "Server rejected
connection.";
    private static final String RAR_EXTRACTION_INSTALLER_MALFORMED_URL
= "Malformed URL.";
    private static final String
RAR_EXTRACTION_INSTALLER_COULD_NOT_CONNECT = "Could not connect to
URL.";
    private static final String RAR_EXTRACTION_INSTALLER_NO_TOOLHOME =
"Must provide a tool home directory.";

    /**
     * Constructs a {@link RarExtractionInstaller}.
     *
     * @param label Optional label for this installer.
     * @param toolHome Path to the location of the tool archive.
     * @param failOnSubstitution Fail the build if substitution fails.
     */
    @DataBoundConstructor
    public RarExtractionInstaller(String label, String toolHome,
            boolean failOnSubstitution) {
        super(label, toolHome, failOnSubstitution);
    }

    /**
     * Performs installation of tool on the node.
     * If the tool archive is e.g. corrupted, exception will be thrown
from the JUnrar library.
     */
    @Override
```

```
    public FilePath performInstallation(ToolInstallation tool, Node
node, TaskListener log) throws IOException, InterruptedException {
        FilePath dir = preferredLocation(tool, node);
        RarFilePath rarDir = new RarFilePath(dir);
        String toolHome = getToolHome();

        if (rarDir.installIfNecessaryFrom(new URL(toolHome), log,
"Unpacking " + toolHome + " to " + dir + " on " +
node.getDisplayName())) {
            rarDir.act(new ChmodRecAPlusX());
        }
        return dir;
    }

    @Extension
    public static class DescriptorImpl extends
ToolInstallerDescriptor<RarExtractionInstaller> {

        @Override
        public String getDisplayName() {
            return RAR_EXTRACTION_INSTALLER_DISPLAY_NAME;
        }

        public FormValidation doCheckUrl(@QueryParameter String value)
{
            try {
                URLConnection con = ProxyConfiguration.open(new
URL(value));
                con.connect();
                if (con instanceof HttpURLConnection) {
                    if (((HttpURLConnection) con).getResponseCode() !=
HttpURLConnection.HTTP_OK) {
                        return
FormValidation.error(RAR_EXTRACTION_INSTALLER_BAD_CONNECTION);
                    }
                }
                return FormValidation.ok();
            } catch (MalformedURLException x) {
                return
FormValidation.error(RAR_EXTRACTION_INSTALLER_MALFORMED_URL);
            } catch (IOException x) {
                return FormValidation.error(x,
RAR_EXTRACTION_INSTALLER_COULD_NOT_CONNECT);
            }
        }

        public FormValidation doCheckToolHome(@QueryParameter String
value) {
            if (StringUtils.isBlank(value)) {
                return FormValidation.ok();
            } else {
                return
FormValidation.error(RAR_EXTRACTION_INSTALLER_NO_TOOLHOME);
            }
        }
    }
```

```
    /**
     * Sets execute permission on all files.
     * Inspired by "ZipExtractionInstaller" from the Custom Tools
plugin.
     */
    static class ChmodRecAPlusX extends
MasterToSlaveFileCallable<Void> {

        private static final long serialVersionUID = 1L;

        public Void invoke(File d, VirtualChannel channel) throws
IOException {
            if (!Functions.isWindows()) {
                process(d);
            }
            return null;
        }

        private void process(File f) {
            if (f.isFile()) {
                f.setExecutable(true, false);
            } else {
                File[] kids = f.listFiles();
                if (kids != null) {
                    for (File kid : kids) {
                        process(kid);
                    }
                }
            }
        }
    }
}
```

B Källkod: RarFilePath.java

```
/*
 * The MIT License
 *
 * Copyright (c) 2016, Martin Hjelmqvist
 * Copyright (c) 2004-2010, Sun Microsystems, Inc., Kohsuke Kawaguchi,
 * Eric Lefevre-Ardant, Erik Ramfelt, Michael B. Donohue, Alan Harder,
 * Manufacture Francaise des Pneumatiques Michelin, Romain Seguy
 * Permission is hereby granted, free of charge, to any person
obtaining a copy
 * of this software and associated documentation files (the
"Software"), to deal
 * in the Software without restriction, including without limitation
the rights
 * to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell
 * copies of the Software, and to permit persons to whom the Software
is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN
 * THE SOFTWARE.
 */
package org.jenkinsci.plugins.extratoolinstallers.util;

import com.github.junrar.testutil.ExtractArchive;
import hudson.FilePath;
import hudson.FilePath.FileCallable;
import hudson.ProxyConfiguration;
import hudson.model.TaskListener;
import hudson.remoting.RemoteInputStream;
import hudson.remoting.VirtualChannel;
import hudson.util.IOUtils;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.Serializable;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import javax.annotation.CheckForNull;
```

```java
import javax.annotation.Nonnull;
import jenkins.MasterToSlaveFileCallable;
import jenkins.SlaveToMasterFileCallable;
import org.apache.commons.io.input.CountingInputStream;
import org.kohsuke.accmod.Restricted;
import org.kohsuke.accmod.restrictions.NoExternalUse;

/**
 * A version of {@link FilePath} modified to extract RAR files instead
of ZIP and TAR.GZ.
 * Intended to fit the needs of the {@link RarExtractionInstaller}.
 * Inspired by {@link FilePath}.
 *
 * @author Martin Hjelmqvist <martin@hjelmqvist.eu>.
 */
@Restricted(NoExternalUse.class)
public class RarFilePath implements Serializable {

    private FilePath dir;

    public RarFilePath(FilePath dir) {
        this.dir = dir;
    }

    public <T> T act(final FileCallable<T> callable) throws
IOException, InterruptedException {
        return dir.act(callable);
    }

    /**
     * Given a rar file, extracts it to the given target directory, if
necessary.
     *
     * <p>
     * This method is a convenience method designed for installing a
binary package to a location that supports upgrade and downgrade.
Specifically,
     *
     * <ul>
     * <li>If the target directory doesn't exist {@linkplain #mkdirs()
it will be created}.
     * <li>The timestamp of the archive is left in the installation
directory upon extraction.
     * <li>If the timestamp left in the directory does not match the
timestamp of the current archive file, the directory contents will be
discarded and the archive file will be re-extracted.
     * <li>If the connection is refused but the target directory
already exists, it is left alone.
     * </ul>
     *
     * @param archive The resource that represents the rar file. This
URL must support the {@code Last-Modified} header. (For example, you
could use {@link ClassLoader#getResource}.)
     * @param listener If non-null, a message will be printed to this
listener once this method decides to extract an archive, or if there
is any issue.
```

```
     * @param message a message to be printed in case extraction will
proceed.
     * @return true if the archive was extracted, false if the
extraction was skipped because the target directory was considered up
to date or an error occurred.
     */
    public boolean installIfNecessaryFrom(@Nonnull URL archive,
@CheckForNull TaskListener listener, @Nonnull String message) throws
IOException, InterruptedException {
        try {
            FilePath timestamp = dir.child(".timestamp");
            long lastModified = timestamp.lastModified();
            URLConnection con;
            try {
                con = ProxyConfiguration.open(archive);
                if (lastModified != 0) {
                    con.setIfModifiedSince(lastModified);
                }
                con.connect();
            } catch (IOException x) {
                if (dir.exists()) {
                    // Cannot connect now, so assume whatever was last
unpacked is still OK.
                    if (listener != null) {
                        listener.getLogger().println("Skipping
installation of " + archive + " to " + dir.getRemote() + ": " + x);
                    }
                    return false;
                } else {
                    throw x;
                }
            }

            if (lastModified != 0 && con instanceof HttpURLConnection)
{
                HttpURLConnection httpCon = (HttpURLConnection) con;
                int responseCode = httpCon.getResponseCode();
                if (responseCode ==
HttpURLConnection.HTTP_NOT_MODIFIED) {
                    return false;
                } else if (responseCode != HttpURLConnection.HTTP_OK)
{
                    listener.getLogger().println("Skipping
installation of " + archive + " to " + dir.getRemote() + " due to
server error: " + responseCode + " " + httpCon.getResponseMessage());
                    return false;
                }
            }

            long sourceTimestamp = con.getLastModified();

            if (dir.exists()) {
                if (lastModified != 0 && sourceTimestamp ==
lastModified) {
                    return false;   // Tool is up to date.
                }
```

```
                dir.deleteContents();
            } else {
                dir.mkdirs();
            }

            if (listener != null) {
                listener.getLogger().println(message);
            }

            if (dir.isRemote()) {
                // First try to download from the slave machine.
                try {
                    act(new Unpack(archive));
                    timestamp.touch(sourceTimestamp);
                    return true;
                } catch (IOException x) {
                    if (listener != null) {
                        x.printStackTrace(listener.error("Failed to
download " + archive + " from slave; will retry from master"));
                    }
                }
            }

            // for HTTP downloads, enable automatic retry for added
resilience
            InputStream in = archive.getProtocol().startsWith("http")
? ProxyConfiguration.getInputStream(archive) : con.getInputStream();
            CountingInputStream cis = new CountingInputStream(in);
            try {
                if (archive.toExternalForm().endsWith(".rar")) {
                    unrarFrom(cis);
                }
            } catch (IOException e) {
                throw new IOException(String.format("Failed to unpack
%s (%d bytes read of total %d)",
                        archive, cis.getByteCount(),
con.getContentLength()), e);
            }
            timestamp.touch(sourceTimestamp);
            return true;
        } catch (IOException e) {
            throw new IOException("Failed to install " + archive + "
to " + dir.getRemote(), e);
        }
    }

    // this reads from arbitrary URL
    private final class Unpack extends MasterToSlaveFileCallable<Void>
{

        private final URL archive;

        Unpack(URL archive) {
            this.archive = archive;
        }
```

```java
        @Override
        public Void invoke(File dir, VirtualChannel channel) throws
IOException, InterruptedException {
            InputStream in = archive.openStream();
            try {
                CountingInputStream cis = new CountingInputStream(in);
                try {
                    unrar(dir, cis);
                } catch (IOException x) {
                    throw new IOException(String.format("Failed to
unpack %s (%d bytes read)", archive, cis.getByteCount()), x);
                }
            } finally {
                in.close();
            }
            return null;
        }
    }

    public void unrarFrom(InputStream in) throws IOException {
        final InputStream inRar;
        inRar = new RemoteInputStream(in);
        try {
            act(new SlaveToMasterFileCallable<Void>() {
                public Void invoke(File dir, VirtualChannel channel)
throws IOException {
                    unrar(dir, inRar);
                    return null;
                }
                private static final long serialVersionUID = 1L;
            });
        } catch (InterruptedException e) {
            Thread.interrupted();
        }
    }

    /**
     * Converts the InputStream to a file, then performs the
extraction into the file 'target' using the JUnrar library.
     *
     * @param target Where the tool will be installed.
     * @param inRar RAR file to be extracted.
     * @throws IOException Thrown if an I/O error occurs.
     */
    private void unrar(File target, InputStream inRar)
            throws IOException {
        File archive = File.createTempFile("tempRar", null);

        try {
            IOUtils.copy(inRar, archive);
            unrar(target, archive);
        } finally {
            archive.delete();
        }
    }
```

```
    /**
     * Performs the extraction specified archive file into the file
'target' using the JUnrar library.
     *
     * @param target Where the tool will be installed.
     * @param archive RAR file to be extracted.
     */
    private void unrar(File target, File archive) {
        try{
            ExtractArchive.extractArchive(archive, target);
        }catch(RuntimeException e){

        }

    }
}
```

C pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.jenkins-ci.plugins</groupId>
        <artifactId>plugin</artifactId>
        <version>1.580.3</version>
    </parent>

    <groupId>com.synopsys.arc.jenkinsci</groupId>
    <artifactId>extra-tool-installers</artifactId>
    <version>0.4-SNAPSHOT</version>
    <name>Jenkins Extra Tool Installers Plugin</name>
    <description>Provides additional tool installers</description>
    <packaging>hpi</packaging>
    <url>https://wiki.jenkins-
ci.org/display/JENKINS/Extra+Tool+Installers+Plugin</url>

    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>

    <repositories>
        <repository>
            <id>repo.jenkins-ci.org</id>
            <url>http://repo.jenkins-ci.org/public/</url>
        </repository>
    </repositories>

    <pluginRepositories>
        <pluginRepository>
            <id>repo.jenkins-ci.org</id>
            <url>http://repo.jenkins-ci.org/public/</url>
        </pluginRepository>
    </pluginRepositories>

    <developers>
        <developer>
            <id>oleg_nenashev</id>
            <name>Oleg Nenashev</name>

<email>nenashev@synopsys.com;o.v.nenashev@gmail.com</email>
            <organizationUrl>http://www.synopsys.com</organizationUrl>
            <organization>Synopsys Inc.</organization>
            <roles>
                <role>maintainer</role>
            </roles>
            <timezone>+4</timezone>
        </developer>
    </developers>

    <scm>
```

```
<connection>scm:git:ssh://github.com/jenkinsci/${project.artifactId}-
plugin.git</connection>

<developerConnection>scm:git:ssh://git@github.com/jenkinsci/${project.
artifactId}-plugin.git</developerConnection>
        <url>https://github.com/jenkinsci/${project.artifactId}-
plugin</url>
    </scm>

    <licenses>
        <license>
            <name>MIT License</name>
            <url>http://www.opensource.org/licenses/mit-
license.php</url>
            <distribution>repo</distribution>
        </license>
    </licenses>

    <dependencies>
        <dependency>
            <groupId>com.github.junrar</groupId>
            <artifactId>junrar</artifactId>
            <version>0.7</version>
        </dependency>
        <dependency>
            <groupId>com.cloudbees.jenkins.plugins</groupId>
            <artifactId>custom-tools-plugin</artifactId>
            <version>0.4.4</version>
            <scope>test</scope>
            <type>jar</type>
        </dependency>
        <dependency>
            <groupId>org.kohsuke</groupId>
            <artifactId>access-modifier-annotation</artifactId>
            <version>1.0</version>
            <type>jar</type>
        </dependency>
    </dependencies>
</project>
```