Tallinna Ülikool
Digitehnoloogiate Instituut

Projektide haldus Maveni abil

Seminaritöö

Autor: Madeleine Poogen
Juhendaja: Jaagup Kippar

Tallinn 2016

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise pooltvarem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Sisukord

Sissejuhatus

Käesolev seminaritöö uurib funktsionaalsusi, mida pakub projektihaldusprogramm Apache Maven 3.3, Java projektide haldamiseks. Tutvustamiseks luuakse ülevaade programmist selle paigaldamise ja rakendamise läbi. Näited tehakse läbi kasutades Eclipse Luna’t ja selle liidest M2Eclipse.

Seminaritöö eesmärgiks on anda ülevaade Maveni võimalustest ja eripäradest võrreldes teiste projektihaldusprogrammide võimalustega.

Töö on jagatud kaheks osaks. Esimene osa, annab ülevaate Maveni ajaloost, struktuurist ja eripäradest võrreldes teiste sisuhaldusprogrammidega.

Teises osas rakendab autor projektihaldurit koodihalduses ja loob ülevaate kasutamisprotsessist läbi paigaldamise ja näitporjekti loomise.

Võõrkeelsete lühendite loetelu

Ant – Apache build manager, Apache järguhaldur.

Eclipse – Integrated development eviroment, integreeritud arenduskeskond.

Gump – Build manager, järguhaldur.

IDE – Integrated Development Eviroment, integreeritud arenduskeskond.

JAR – Java Archive, Java arhiiv.

Java – Programming language, programmeerimise keel.

JDK – Java Development Kit, Java arenduskomplekt.

JEdit – Software text editor, tarkvara tekstiredaktor.

JRE – Java Runtime Environment, Java käitlusaja keskkond.

JUnit – Unit testing framework, Üksuse testimise raamistik.

M2Eclipse – Plugin, pistikprogramm.

Maven – Apache build manager, Apache järguhaldur.

NetBeans – Integrated development eviroment, integreeritud arenduskeskond.

POM – Project Object Model, Projekt objekt mudel.

TestNG – Unit testing framework, Üksuse testimise raamistik.

XML – Extensible Markup Language, Laiendatav märgistuskeel.

1. Maven

Apache Maven on projektihalduse tööriist mis põhineb Projekt Objekt Mudel kontseptsioonil. Idee Maven luua tulenes vajadusest projektide standardartsete järkude järele, mille olemasolu annaks võimaluse selgelt projekti defineerida ja erinevate projektide vahel JAR faile jagada. Maveni peamine ülessanne on anda arendajale võimalus hallata Java projekti arenduskäigu kõiki olekuid võimalikult vähese ajakuluga ja muuta arendusprotsess võimalikult lihtsaks (Apache Software Foundation, 2016).

1.1. Ajalugu

Maven sai alguse Jakarta Alexandria projekti all olles, kust arenes välja ka nii mõnigi teine projekt. Maven oli vaid 5 kuud Alexandria projekti all, enne kui ta liikus aastal 2002 Turbine projekti alla. Kuna tol ajal oli Ant kasutusel, siis leidsid Turbine projekti arendajad, et erinevate Ant'i järkude kaustamine iga projekti elemendi jaoks oli väga kasutu ja tülikas (Apache Software Foundation, 2016). Kuigi Ant'i projektide infrastruktuur on tähtis, siis tihti peale lakkas see töötamast just siis, kui projekti peal töötab rohkem kui paar inimest või projekt on vaja redakteetrida. Tähtis oli see, et programmi järk töötaks lihtsalt, kuid Ant'i puhul oli haruldane see, et kõik kohe ideaalselt tööle hakkas. Arendajad, kes kasutasid Ant'i pidid tihti tunde veetma, et aru saada sellest, kus mingi fail on või kust mingi reegel avaldub. Seetõttu toetus Maveni vajadus kahele peamisele puntkile:

- Vajadus luua mudel, kust saab kõik projekti puudutava info kätte.
- Vajadus standartsele kataloogide hierarhiale.

Kõik algas lihtsa XML esitusena, kus oli ära otsustatud, milline võiks kataloogide hierarhilise struktuuri standard. Samal ajal Alexandria all arenev projekt Gump taheti küll algselt inspiratsiooniks võtta, kuid Gump üritas anda kasutajale võimalikult palju struktuurset vabadust ja võimalust kasutada mitmeid JAR faile projekti kohta, mis oli vastupidine sellele, mida Mavenit taheti tegema panna. Maveni arendajad tahtsid, et projekti infrastruktuur näeks ja töötaks samamoodi iga projekti raames, ja et Mavenit kasutades oleks kõik kasutajal kohe teada, kust mida otsida ja mida muuta, kuna kõik oli alati ühes kohas.

Maven 1.x arengu käigus pandi samuti tähele, et paljud JAR failid, millele Turbine projektid toetusid, olid kas duplikaadid või sama JAR faili erinevad järgud, mis olid jaotatud erinevate projektide peale ja võtsid palju ruumi. Lahenduseks võeti mudelisse kasutusele Java-tüüpi päriluse standardid ja leiti viis kuidas luuda hoidla järkude arendamise jaoks.

Maven 1.x kannatas paljude vigade all, kuid nendest vigadest tulenenud tagasiside viis aastal 2005 stabiilse Maven 2.0 arendamiseni, mis oli piisavalt stabiilne, et seda hakkasid ka teised projektid aktiivselt kasutama (Apache Software Foundation, 2016).

1.2. Maven projektihalduses

Maven on projektihalduse tööriist mis haldab endas POM'i, kokkulepitud standardeid, projekti elutsükkleid, sõltuvuste juhtumise süsteemi ja loogikat järkude eesmärkide teostamiseks erinevates arendustsükklite faasides. Mavenit kasutades defineeritakse projekt POM'i abil, mis aitab Mavenil kasutada samu pluginaid üle terve projekti (Apache Software Foundation, 2016). Maveni põhiline eesmärk on aidata arendajal hallata tervet arendusprotsenti võimalikult väheseaja ja vaevaga. Selleni jõudmiseks lähtutakse järgmistest punktidest:

- Järkude loomise protsessi lihtsustamine
- Ühtlase järgusüsteemi tagamine
- Kvaliteetse projekti informatsiooni tagamine
- Juhiste andmine parimate arendustavade jaoks
- Uute vahendite kasutusele võtmiseks läbipaistva migreerumise lubamine

Eeltoodud punktide üks tähtsamaid elemente on üks Maveni tähtsaim põhimõte – konventsioon üle konfiguratsiooni (Apache Software Foundation, 2016). Põhimõtte konseptsioon on lihtne: Süsteemid, teegid ja raamistikud peaksid järgima mingit mõstlikku vaikseadistust ja peaksid ilma eelkonfigureerimata toimima. Selle talitluse jõudlustumiseks on tagatud projektidele mõistlik vaikimisi käitumine:

- Programmi lähtekood on vaikimisi kaustas `${baaskataloog}/src/main/java`.
- Programmi ressursid on kaustas `${baaskataloog}/src/main/resources`.
- Programmi testid on kaustas `${baaskataloog}/src/test`.
- Projekti tulemuseks on JAR fail.

Peale kataloogide haldamise, mõjutab see konseptsioon ka Maveni enda pluginaid, millele on rakendatud tavasid lähtekoodi komplileerimiseks, versioonide pakkimiseks, veebilehtede loomiseks ja paljudeks teisteks protsessideks. Mavenil on olemas defineeritud järgutsükkel ja kolmplekt pluginaid, mis teavad kuidas luua ja koostada tarkvara, mille tõttu on seda lihtne

kasutada. Arendaja, kes tahab muuta vaikevalikuid, saab seda ka teha, kuid soovituslikult mitte täies mahus (O'Brien et al., 2008).

Üks suur osa Mavenist on selle võime tagasiühilduvust rakendada. Näiteks *The Maven Surefire plugin* on plugin, mida kasutatakse komponentide testimise jooksutamiseks. Maven kasutab testimiseks vaikseadena JUnit'it, kuid TestNG võeti vahepeal lisavõimalusena kasutusele. Kui arendaja kasutas Surefire pluginat, et komplieerida ja jooksutada JUnit 3 teste ja uuendas oma Surefire pluginat, siis see ei takistanud tal teste jooksutada, kui andis võimaluse kasutada TestNG pluginat testimiseks. Arendaja jaoks ei muutunud midagi peale ühe versiooninumbri muutumise POM failis.

See Maveni eripära mõjutab muidugi palju rohkemat, kui ühte pluginat, sest ka Ant projekte saab Maveniga kasutada, mille tõttu on Ant'i pealt Maveni peale üle minemine palju lihtsam kui vastupidi. Samuti võimalus uuendada ja installida uusi tööriistu, ilma et nende töölesaamiseks, kas midagi katki läheks või peaks tundide viisi aega kulutama, on tihti see mis tõmbab arendajaid Mavenit kasutama.(O'Brien et al., 2008).

Mavenil on programmi mudel, mis tähendab, et arendaja kood ei muudeta lihtsalt baitkoodiks, vaid, et arenduse käigus luuakse kirjeldus tarkvara projektist ja määratakse sellele unikaalselt koordinaadid, mille läbi saab kirjeldada projekti atribuute. Selline mudeli defineerimine võimaldab järgnevaid funktsioone:

- Sõltuvuste korraldus

Projekte defineeritakse unikaalsete identifikaatorite läbi, mis koosnevad Gruppi idenfikikaatorist, artefakti indentifikaatorist ja versioonist, siis projektid saavad kasutada neid koordinaate, et sõltuvusi deklareerida.

- Kaughoidlad

Kordinaate, mis on defireeritud POM failis, saab kasutada artefaktide hoitlate loomiseks.

- Universaalne järgu taaskasutamise loogika

Pluginad kasutavad loogikat, mis POM failis oleva kirjeldavate andmete ja konfiguratsiooni parameetritega.

- Tööriistade integreerimine

Tööriistad nagu Eclipse, NetBeans ja JEdit suudavad tänu Maveni struktuursusele leida informatsiooni projekti kohta samast kohast. Enne Maveni kasutuselevõttu, oli igal IDE-l oma viis kuidas projekti sisemist struuktuuri hallata. Maven on loonud standartse mudeli, mille järgi saab projekti faile genereerida.

- Lihtne projektide ja artefaktide, otsimine ja filtreerimine

Tööriistad nagu Nexsus aitavad indekseerida ja otsida hoidla sisu, kasutades informatsiooni mida hoitakse POM failis (O'Brien et al., 2008).

1.3. Maven ja Ant

Mavenit võrreldakse tihti Ant'iga, mis on samuti üks Apache projektidest, kuid Ant ja Maven on loodud erinevatele asjadele mõeldes. Maven on mõeldud, et teha koodi mõistmine ja arendamine lihtsamaks, kindlate struktuuride ja mallide loomise läbi. Ant on mõeldud selleks, et anda arendajale võimalikult palju vaba voli. Maveni miinuseks on limiteeritus ja Ant'i miinuseks on see, et teisel arendajal võib väga kaua aega minna, et saada aru kus miski asi asub.

Mugavuse ja kasutajasõbralikkuse suhtes on Maven palju parem valik, kui Ant. Kuna Mavenil on väga kindel struktuur, siis on Mavenist lihtsam aru saada, sest kõik on alati samal kohal ja kättesaadaval.

Kui erinevate IDE lahenduste vahel jagada projekti, siis on palju suurem tõenäosus, et projekt töötab, kui tegemist on Maveni projektiga. Ant'i projektide liigutamine erinevate IDE lahenduste ja isegi sama IDE vahel on tihti väga ajakulukas töö.

Ant'i eelistab üldiselt arendaja, kes soovib võimalikult palju ise struktuuri kontrollida. Maven annab kaasa kitsad raamid ja nendest raamidest välja saamine tülikas protsess, kuid Ant'i puhul piisab tihti lihtsalt spetsifitseerimisest ja kõik töötab.

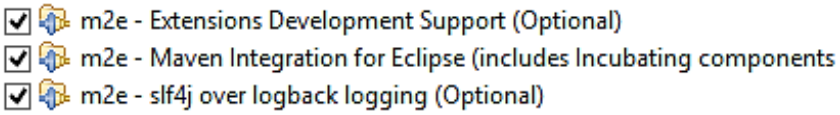
Arvestades asjaolu, et töö on suunatud pigem algajatele, siis töö autor soovitab tarkvara arendamise protsessis kasutada pigem Mavenit, kui Ant'i, sest Maveni raamide tõttu on raskem vigu teha.

2. Maveni kasutamine

Järgmisena vaatame, kuidas paigaldada Maveni lisa Eclipsele, kasutades M2Eclipse liidest. Et M2Eclipse töötaks, on vaja, et arvutis oleks installeeritud eelnevalt Eclipse ja JDK 1.5 või kõrgem. Protsessi lihtsustamiseks on dokumendis ka ekraanitõmmised.

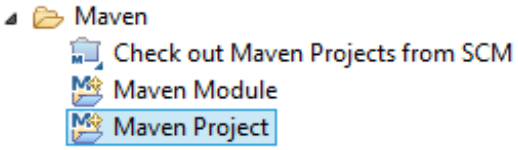
2.1. Maveni Paigaldamine

Eclipse pakub võimalust tarkvara allalaadimiseks ja installeerimiseks läbi oma integreeritud programmeerimiskeskkonna. Tarkvara installeerimise keskkonnas, *General purpose tools* all on kolm erinevat m2e liidese elementi, millest on soovitatav kõik ka installeerida.



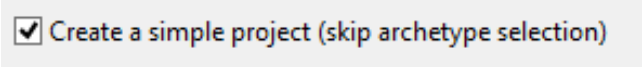
Joonis 1. m2e laiendid

Pärast installeerimise protsessi teeb Eclipse endale restardi. Pärast restarti saab alustada uue Maven projekti loomisega.



Joonis 2. Maveni projekti loomine

Järgmisel seadistuse lehel on võimalus valida töökeskkonda, lisada projekt spetsiifilisse projektide komplekti ja lisada võimaluse arhhetüüpi vahetada. Vaikimisi valikuks on *Simple project*, mis kasutab vaikevalikulist arhhetüüpi.



Joonis 3. Lihtsa projekt loomine

Artefakti seadistuse lehel küsitakse täpsemalt kirjeldamiseks vähemalt kolme elementi: *Group Id*, *Artifact Id* ja *Version*.

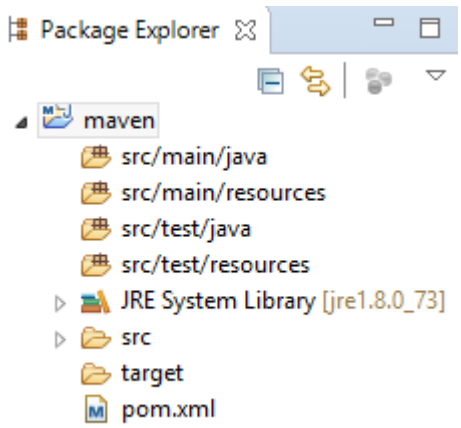
- *Group Id* on projekti unikaalne identifikaator, mille jaoks kasutatakse domeeni nime. Kui projektile mooduleid juurde lisada, siis nende indendifikaator lisandub see Gruppi identifikaatori külge.
- *Artifact Id* on versiooninumbrita JAR'i nimi.
- *Version* on versiooni number, mis on vajalik järkude vahel eristamiseks.

Samuti on võimalik lisada projektile nimi ja kirjeldus ja liigutada projekt emaprojekti alla (Apache Software Foundation, 2016).

Projekti loomise järel peab kontrollima, kas Eclipse hoiatab installeerimise vigade eest või mitte. Mõni kord peab vahetama projekti raja alt JDK või JRE versioone, et Maven sujuvalt toimiks. Kui hoiatusi ignoreerida, siis võib järgutsükkel ebaõnnestuda.

2.2. Maveni Põhikomponendid

Maveni projekti all on tuduvalt rohkem allikaskatalooge, kui tavalises Java projektis, mis tuleneb jällegi Maveni omapärast, kõik failid õigetesse kohtadesse jaotada.

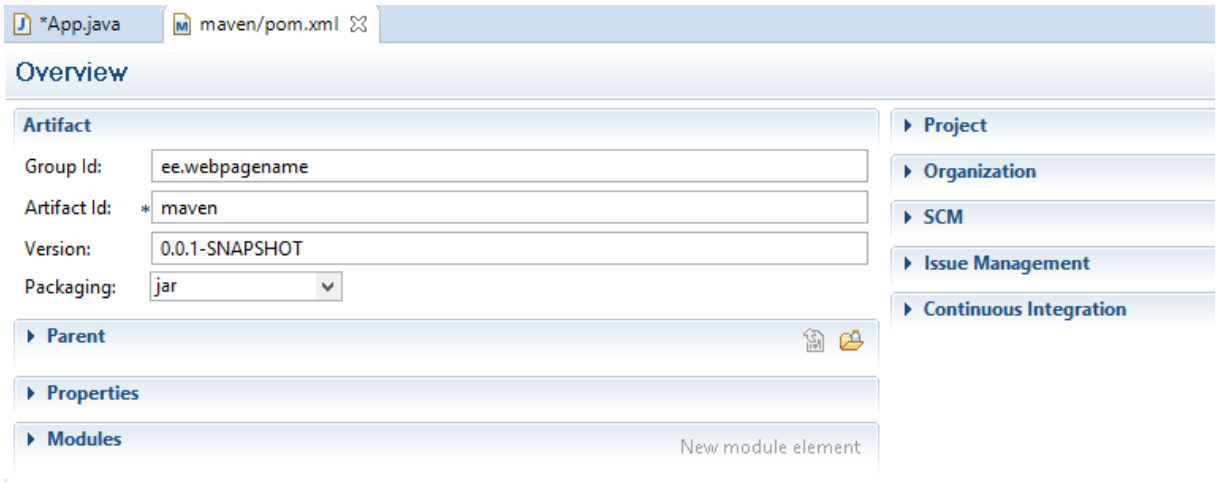


Joonis 4. Projekt kataloogid

Peale allikaskataloogide on projekti all ka sihtkataloog, mis majutab endas kõiki projekti väljudandmeid, JRE süsteemi teegide loend ja XML fail pom.xml, mis mahutab endas projekti sisemise struktuuri kirjeldust.

Faili avades avaneb ülevaade projekti infost, mida on kõike ka võimalik muuta. Saab lisada emaprojekti, avada olemasoleva emaprojekti POM faili, muuta Grupi ja Artefakti Identifikaatorit ja lisada infot projekti kohta (Tutorialspoint, kuupäev puudub).

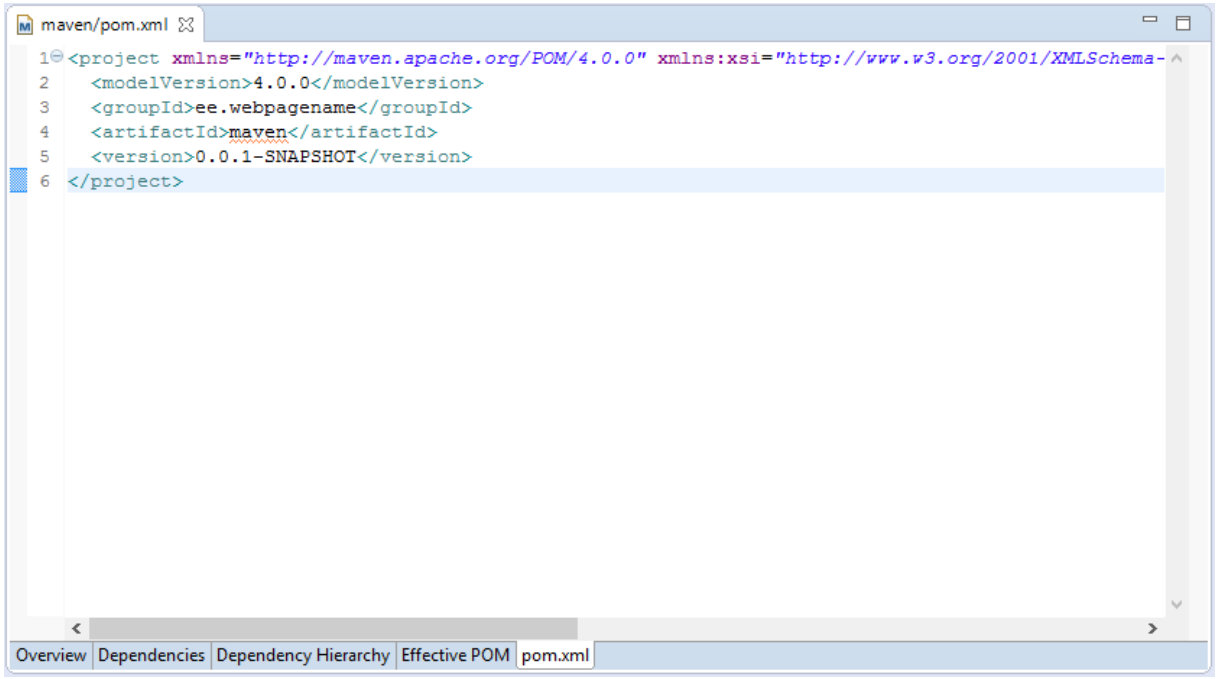
Edasi saab liikuda sõltuvuste lehtedele, kus ühel lehel saab lisada sõltuvusi ja teisel lehel redigeerida nende sõltuvuste hierarhiat.



Joonis 5. Sõltuvused

POM faili puhul on näha ja võimalik eristada kahte erinevat vaadet:

- *Pom.xml* on fail ise, mis sisaldab endas sõltuvusi ja informatsiooni, mida arendaja saab muuta ja lisada. Algselt on olemas Pom.xml failis Grupi ja Artefakti identifikaatorid, mudeli versioon ja versiooni number. Seda POM faili nimetatakse samuti *Simple POM* failiks, kuna, kui luua lihtsat projekti, siis tihti peale ei ole vaja muuta midagi rohkemat, kui seda POM faili.



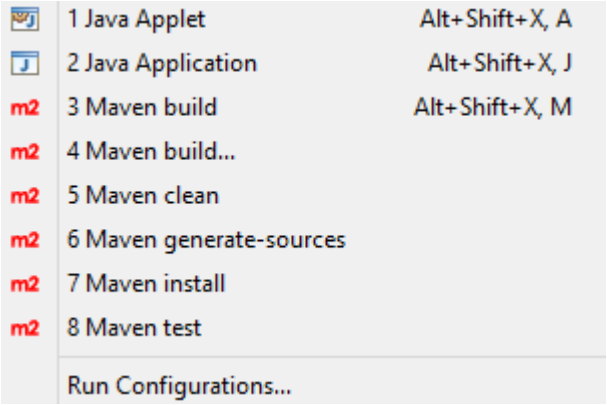
Joonis 6. pom.xml

- *Effective POM* on visuaalne kombinatsioon projekti POM failist ja *Super POM* failist ja annab võimaluse projekti põhiselt *Super POM* failis määratud näha ilma faili avamata.

Joonis 7. Effective POM

POM failide eraldamine on väga tähtis selle jaoks, et siis on näha *Super POM* faili suhtes pom.xml faili sisu asetust, kuid Effective POM faili muuta ei saa.(Apache Software Foundation, 2016).

Peale projekti loomise ja POM faili järgi haldamise, on Mavenil veel paar tavakasutajale tähtsat elementi. Üks nendest on võimalus jooksutada erinevaid Järgutsükli elemente.



Joonis 8. Maveni jooksutamise võimalused

- *Maven build* viib läbi terve Maveni järgutsükli protsessi, millega täidab kõik järgutsükli eesmärgid.
- *Maven build*...viib läbi valitud eesmärgid Maveni järgutsüklist.
- *Maven clean* puhastab projekti kaustast järgu loomise ajal kaasnenud failid.
- *Maven generate-sources* genereerib lähtekoodi.
- *Maven install* installeeri portsesseeri ja liiguta pakett kohalikku hoidlasse
- *Maven test* testi lähtekoodi sobiva raamistiku abil

Näitena kasutame failil App.java *Maven build* protsessi

```
package maven;

public class App {
	public static void main(String[] args){
		System.out.println("Tere");
	}
}
```

Kuna lähtekoodis pole vigu ja Maven on õigesti seadistatud, siis protsessi järel on tagasiside järgmine

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.488 s
[INFO] Finished at: 2016-03-28T11:39:38+03:00
[INFO] Final Memory: 13M/140M
[INFO] ------------------------------------------------------------------------
```

Joonis 9. Järk õnnestus

Logi näitab tulemust, kulunud aega, tsükli lõpetamise aega ja kulunud mälu.

Logis on ka näha tervet järgutsükli protsessi, mille järgi saab vaadata, kas mingi järk sai läbitud edukalt või mitte.

Edasi kasutame failil App.java ainult *Maven test* protsessi läbimiseks.

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.610 s
[INFO] Finished at: 2016-03-28T19:15:22+03:00
[INFO] Final Memory: 9M/114M
[INFO] ------------------------------------------------------------------------
```

Joonis 10. Test õnnestus

Võrreldes eelnevaga oli mälu ja aja kulu väiksem, kuna protsessi samme oli vähem. Väikese programmiga võib vahe olla väga väike, kuid suure projektiga on aja ja mälu kulu väga määrav ja tihti ka põhjus, miks tihti ainult teatud järgutsükli eesmärgid läbi proovitakse, kui teisi pole hetkel tarvis.

Kokkuvõte

Käesolevas seminaritöö eesmärgiks oli anda ülevaade Maveni ajaloost, põhimõtetetest, funktsionaalsustest ja kasutusvõimalustest. Töö käigus toodi välja Maveni eripärad, head küljed, kuid ka puudujäägid.

Töö käigus selgub, miks paljud arendajad on valinud Maveni, et hallata nende Java projekte ja miks Maven on hea tööriist algajatest Java arendajatele.

Töö autor leiab, et Mavenit võiks kasutada Javas programmeerimise õpetamiseks, kuna seda on lihtne kasutada ja struktuur kergesti navigeeritav.

Antud teemal annaks uurida ja võrrelda põhjalikumalt erinevaid järkude haldamiseks mõeldud tööriistu ja samuti annaks luua põhjaliku eestikeelse õppematerjali.

Kasutatud kirjandus

O'Brien T., Casey J., Fox B., Van Zyl J., Xu J., Locher T., Fabulich D., Redmond E., & Snyder B. (2008). *Maven By Example.* Sebastopol, USA: O'Relly Media, Inc.

Apache Software Foundation. (2016). *Apache Maven Project.* Loetud aadressil https://maven.apache.org/

Tutorialspoint. (kuupäev puudub). *Maven Tutorial*. Loetud 28.02.2016 adressil http://www.tutorialspoint.com/maven/

Apache Software Foundation. (2016). *Guide to naming conventions.* Loetud 28.02.2016 aadressil https://maven.apache.org/guides/mini/guide-naming-conventions.html

Apache Software Foundation. (2016). *Introduction to the Standard Directory Layout.* Loetud 28.02.2016 aadressil https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

Apache Software Foundation. (2016). *What is Maven?* Loetud 28.02.2016 aadressil https://maven.apache.org/what-is-maven.html