TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika õppekava

Janar Ojalaid

Projekti sõltuvuste andmebaas

Bakalaureusetöö (9 EAP)

Juhendajad: Priit Liivak
Helle Hein

Tartu 2015

Projekti sõltuvuste andmebaas

Lühikokkuvõte:

Tarkvaraettevõtetes on tihti käsil mitmeid projekte. Peaaegu iga projekt sõltub kolmandate osapoolte teekidest, mis oma olemuselt on taaskasutatavad tarkvara osad. Lihtne on leida, missugused teegid on kasutusel mingis kindlas projektis, aga hoopis raskem on teha kindlaks, missugustes projektides on mingit kindlat teeki kasutatud. Bakalaureusetöö teoreetilise osa eesmärgiks on uurida erinevaid lahendusi projekti sõltuvuste analüüsiks ning leida neist parim. Praktilise osa eesmärgiks on valmistada prototüüp kasutades kogutud informatsiooni. Selline andmebaas tagab ettevõttesisese teadmuse efektiivsema jagamise, mis omakorda tähendab seda, et projektid valmivad kiiremini ning on kvaliteetsemad.

Võtmesõnad:

Projekti sõltuvused, andmebaas, kolmandate osapoolte teegid

Project dependency database

Abstract:

Software development companies are often working on several projects. Almost every project depends on third party libraries. It is easy to find what technologies and libraries are in use in a single project but it is much more complicated to find what projects use a technology or specific library. The aim of the bachelor's thesis theoretical part is researching different dependency analysers. The aim of the practical part is making a project dependency database prototype using the gathered information. Such database ensures effective sharing of knowledge within the company, which in turn means that the projects are completed faster and have better quality.

Keywords:

Project dependencies, database, third party libraries

Sisukord

1. Sissejuhatus

Peaaegu iga projekt sõltub kolmandate osapoolte teekidest, mis oma olemuselt on taaskasutatavad tarkvara osad. Üpris lihtne on kindlaks teha, missugused teegid on ühes kindlas projektis kasutusel. Hoopis raskem on aga leida, missugustes projektides on mingit kindlat teeki kasutatud. Kui mingi kindla teegiga on probleeme, tuleks abi saamiseks leida inimesi, kes selle teegiga on kokku puutunud. Kuna ühes ettevõttes võib olla sadu erinevaid projekte, ei ole neist käsitsi informatsiooni otsimine otstarbekas. Samuti ei ole praktiline käsitsi sellist andmebaasi luua ja hallata, kuna kasutusel olevad teegid võivad tihti muutuda.

Hetkel eksisteerib mitu tasulist lahendust, nagu näiteks WhiteSource [1] ja BlackDuck [2], mis suudavad analüüsida projekti sõltuvusi, anda informatsiooni teekides olevatest turvariskidest ning nõutavatest litsentsidest. Need lahendused on aga üpris kallid, hinnad algavad 2500 dollarist aastas ühe projekti kohta, mistõttu ei ole otstarbekas kasutada neid väiksemate projektide puhul. Vaja oleks odavamat, soovitatavalt vabavaralist lahendust, mis suudaks anda informatsiooni projektis kasutusel olevatest teekidest ning informeerida, kui teegist on uuem versioon saadaval.

Käesoleva bakalaureusetöö teoreetilise osa eesmärgiks on uurida missuguseid lahendusi on projekti sõltuvuste analüüsiks ning leida neist parim. Arvesse tuleb võtta nii hinda, projektiga liidestamise keerukust, saadava informatsiooni kogust, täpsust ja formaati. Lisaks peab leitud lahendus toetama kahte tüüpi projekte – Maven [3] ja Gradle [4]. Ant ja Ivy projekte saab kerge vaevaga konvertida Gradle projektideks, mistõttu neid eraldi pole vaja uurida. Nagu näha Jooniselt 1 ja 2, kirjeldatakse sõltuvusi neis erinevalt. Kui sellist lahendust, mis rahuldavalt analüüsiks nii Maveni kui ka Gradle sõltuvusi, ei leidu, on võimalik Maveni projektide analüüsiks kasutatada üht ning Gradle projektide jaoks teist lahendust.

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>${servlet.version}</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
        <version>${spring.version}</version>
    </dependency>
</dependencies>
```

Joonis 1. Sõltuvuste kirjeldamine - Maven

```
dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.+'
    compile 'javax.servlet:javax.servlet-api:3.1.0'
    compile 'org.springframework:spring-context-support:4.1.5.RELEASE'
}
```

Joonis 2. Sõltuvuste kirjeldamine - Gradle

Bakalaureusetöö praktilise osa eesmärk on luua kogutud informatsiooni kasutades projekti sõltuvuste andmebaasi rakenduse prototüüp. Rakendus peab andma informatsiooni järgneva kohta:

1. missugustes projektides mingi otsitav teek kasutusel on;
2. kas teegist on uuemaid versioone saadaval;
3. missugused on teegis teadaolevad turvariskid;
4. mis litsentsiga teek on.

Esimene punkt on väga oluline teadmuse efektiivsemaks jagamiseks. See võimaldab lihtsalt leida inimesi, kes on teegiga tuttavad. Teine ja kolmas punkt on olulised turvalisuse seisukohalt. Neljas punkt on eriti oluline suurfirmades. Mõni arendaja võib projektis kasutusele võtta mingi sobimatu litsentsiga teegi, mis võib hiljem põhjustada kohtuvaidlusi (intellektuaalse omandi vargus). Projektis kasutusel olevate teekide kaardistamine maandab riske tunduvalt.

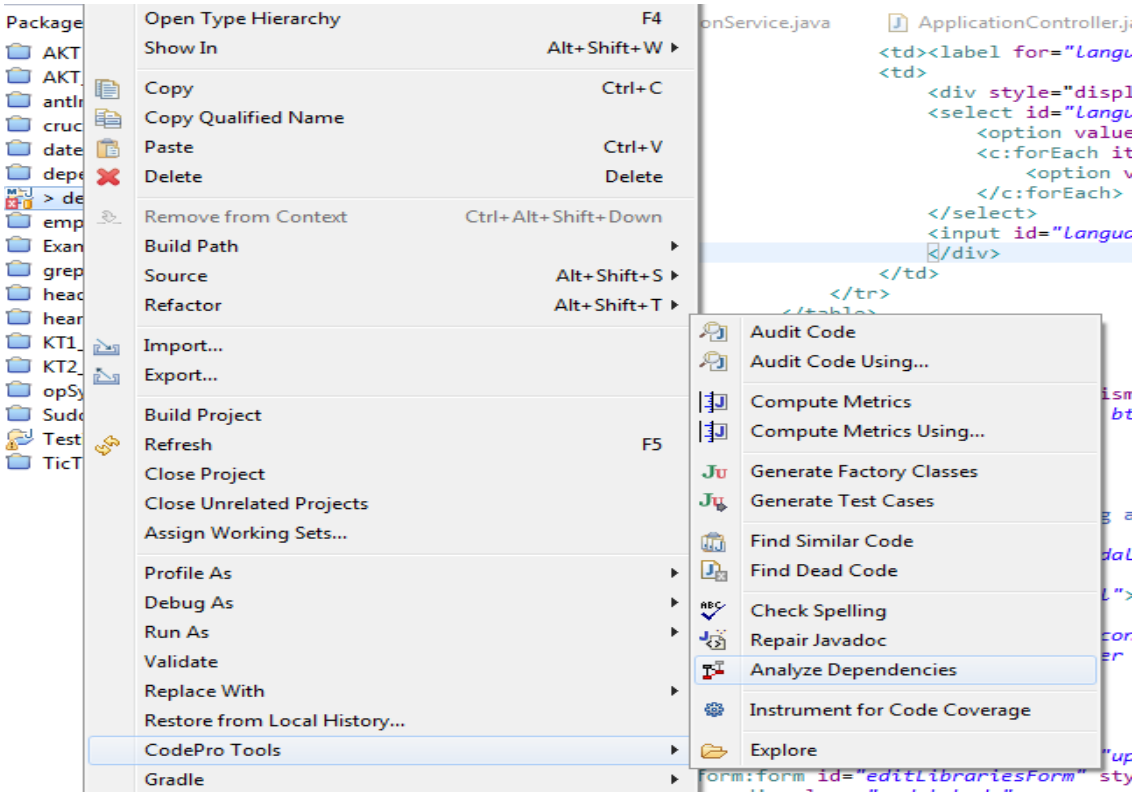
2. Projekti sõltuvuste analüüsivahendid

Tänapäeval on väga palju erinevaid vahendeid, mis võimaldavad analüüsida projektis kasutusel olevaid teeke. Töös on välja valitud 11 tööriista, mis tundusid lootustandvad, kuid kindlasti ei ole see nimekiri lõplik.

2.1 CodePro AnalytiX

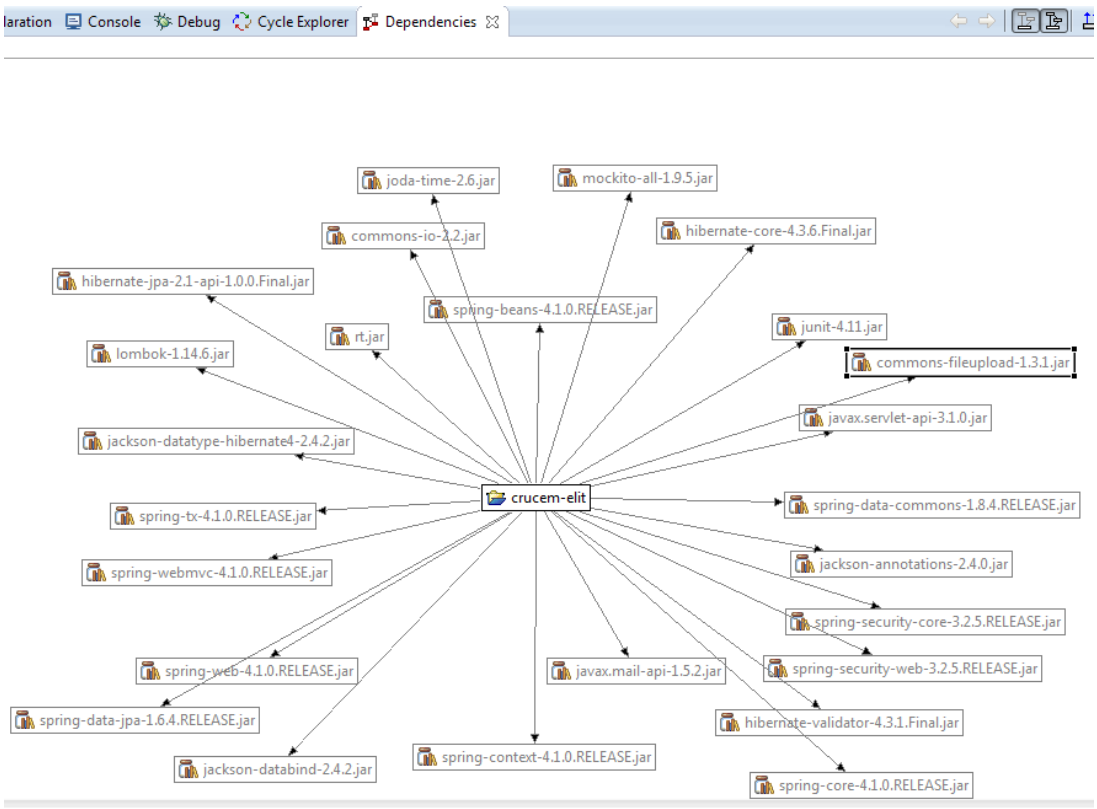
CodePro AnalytiX [5] on Google'i poolt välja antud Java tarkvara testimise tööriist Eclipse'i kasutajatele. Tööriist tehti avalikuks 2001. aastal, kuid järk-järgult on seda edasi arendatud. Viimane uuendus sellele tuli aastal 2010. Lisaks projekti sõltuvuste analüüsimisele, suudab CodePro AnalytiX analüüsida koodi, genereerida ühikteste, mõõta koodi kaetust testidega ning koostada erinevaid diagramme.

CodePro AnalytiX kasutamiseks tuleb Eclipse'i installeerida pistikprogramm. Seejärel tuleb Eclipse taaskäivitada ning pistikprogramm ongi kasutamiseks valmis. Projekti sõltuvuste analüüsiks tuleb parema hiireklõpsuga vajutada projekti peale ning rippmenüüst valida sõltuvuste analüüs nagu näha Joonisel 3.



Joonis 3. CodePro AnalytiX projekti sõltuvuste analüüsimine

Projekti sõltuvuste analüüs toimub väga kiiresti, kuna analüüsimiseks projekti kompileerima ei pea. Väljundit on võimalik saada kahes formaadis. Esimene neist on graafiline nagu näha Joonisel 4. Teine variant on genereerida graafilisest väljundist veebileht. Veebilehel on küll palju erinevaid tabeleid, kuid kogu vajalik informatsioon on ühes kohas olemas nagu näha Jooniselt 5. Seepärast oleks ilmselt mõistlikum parsida veebilehte.



Joonis 4. CodePro AnalytiX graafiline väljund

Referenced Projects

commons-fileupload-1.3.1.jar
commons-io-2.2.jar
hibernate-core-4.3.6.Final.jar
hibernate-jpa-2.1-api-1.0.0.Final.jar
hibernate-validator-4.3.1.Final.jar
jackson-annotations-2.4.0.jar
jackson-databind-2.4.2.jar
jackson-datatype-hibernate4-2.4.2.jar
javax.mail-api-1.5.2.jar
javax.servlet-api-3.1.0.jar
joda-time-2.6.jar
junit-4.11.jar
lombok-1.14.6.jar
mockito-all-1.9.5.jar
rt.jar
spring-beans-4.1.0.RELEASE.jar
spring-context-4.1.0.RELEASE.jar
spring-core-4.1.0.RELEASE.jar
spring-data-commons-1.8.4.RELEASE.jar
spring-data-jpa-1.6.4.RELEASE.jar
spring-security-core-3.2.5.RELEASE.jar
spring-security-web-3.2.5.RELEASE.jar
spring-tx-4.1.0.RELEASE.jar
spring-web-4.1.0.RELEASE.jar
spring-webmvc-4.1.0.RELEASE.jar

Joonis 5. CodePro AnalytiX veebilehel olev tabel

CodePro AnalytiX kõige positiivsemad küljed on kindlasti kasutamise lihtsus ning kiirus. Pistikprogrammi installeerimine on ülimalt kerge ning analüüsida saab nii Maveni kui ka Gradle projekte. Negatiivseks küljeks on aga informatsiooni nappus – näiteks ei saa informatsiooni uuemate versioonide, turvariskide ega litsentside kohta.

2.2 Degraph

Degraph [6] on tööriist, mis on mõeldud spetsiaalselt Java virtuaalmasina peal töötavate rakenduste sõltuvuste analüüsiks. Degraph sai alguse aastal 2011. Töö projekti kallal endiselt jätkub. Kuigi Degraphi on arendatud juba neli aastat, on väljas alles versioon 0.1.2.

Degraphi kasutamiseks tuleb alla laadida *JAR*-fail. See on kättesaadav nii Degraphi lehelt kui ka Maveni repositooriumist. Seejärel tuleb vastavalt vajadustele luua konfiguratsiooni fail. Konfiguratsiooni failis saab ära määrata:

- väljundfaili nime;
- faili, mida analüüsida;
- klassid, mis analüüsist välja jätta;
- klassid, mida analüüsida, kusjuures vaikimisi analüüsitakse kõiki klasse.

Konfiguratsiooni faili loomine on lihtne ning see on tavaliselt lühike nagu näha Jooniselt 6.

```
1  output = example2.graphml
2  classpath = ../lib/dateutils.jar
3  exclude = java*.**
4  exclude = scala.**
5  exclude = org.scalatest.**
```

Joonis 6. Degraph konfiguratsiooni fail

Degraph analüüsib baitkoodi, mistõttu tuleb projekt enne analüüsimist kokku ehitada. Samuti peab arvutis olema installeeritud vähemalt Java 7. Seejärel tuleb Degraph käsurealt käivitada, andes talle ette loodud konfiguratsiooni fail (Joonis 7). Analüüsida saab nii Gradle kui ka Maveni projekte.

```
C:\Users\Janar\Downloads\degraph\bin>degraph -f ../example/example2.config
Found 24 nodes, with 0 slice edges in violation of dependency constraints.
C:\Users\Janar\Downloads\degraph\bin>_
```

Joonis 7. Degraphi käivitamine käsurealt

Tulemuseks on *GRAPHML*-vormingus fail. Fail on üpris mahukas ning seda on võimalik vaadata graafiliselt programmiga yEd [7]. Tekkinud faili on võimalik vaadata ka tekstikujul kasutades selleks ükskõik millist tekstiredaktorit. Jooniselt 8 on näha, et tekstikujul väljund sisaldab palju ebaolulist, mistõttu seda on raske parsida. Olulist informatsiooni on väljundis väga vähe – olemas on ainult klassid, mida projektis on kasutatud. Puudub informatsioon teekide versioonidest, uuematest saadaolevatest versioonidest, turvariskidest ja litsentsidest ning kuna analüüsitakse failide baitkoodi, võivad mõned sõltuvused jääda tuvastamata. Näiteks jääksid tuvastamata klassid, mida kasutavad staatilised muutujad.

```
</edge><edge id="SimpleNode(Class,example.janar.App)::SimpleNode(Class,java.lang.StringBuffer)"
source="SimpleNode(Class,example.janar.App)" target="SimpleNode(Class,java.lang.StringBuffer)">
    <data key="d10">
        <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:BendStyle smoothed="true"/>
        </y:PolyLineEdge>
    </data>
</edge><edge id="SimpleNode(Class,example.janar.App)::SimpleNode(Class,java.util.Arrays)"
source="SimpleNode(Class,example.janar.App)" target="SimpleNode(Class,java.util.Arrays)">
    <data key="d10">
        <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:BendStyle smoothed="true"/>
        </y:PolyLineEdge>
    </data>
</edge><edge id="SimpleNode(Class,example.janar.App)::SimpleNode(Class,org.joda.time.LocalDate)"
source="SimpleNode(Class,example.janar.App)" target="SimpleNode(Class,org.joda.time.LocalDate)">
    <data key="d10">
        <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:BendStyle smoothed="true"/>
        </y:PolyLineEdge>
    </data>
```

Joonis 8. Degraphi väljund

Degraphi eelisteks on kasutamise mugavus. Kasutaja saab ise konfigureerida, mida ta täpsemalt analüüsida soovib ning jätta välja ebaolulised klassid. Teisest küljest on see veel väga algeline, olulist informatsiooni on väljundis vähe ning kõiki sõltuvusi ei tuvastata.

2.3 Pistikprogramm Gradle Project Report

Gradle Project Report pistikprogramm [8] on Gradle'i poolt väljaantud ametlik pistikprogramm. Selle kasutamiseks tuleb *build.gradle.* faili lisada '*project-report'*-nimeline pistikprogramm (vt joonis 9).

```
apply plugin: 'project-report'
```

Joonis 9. Pistikprogrammi lisamine

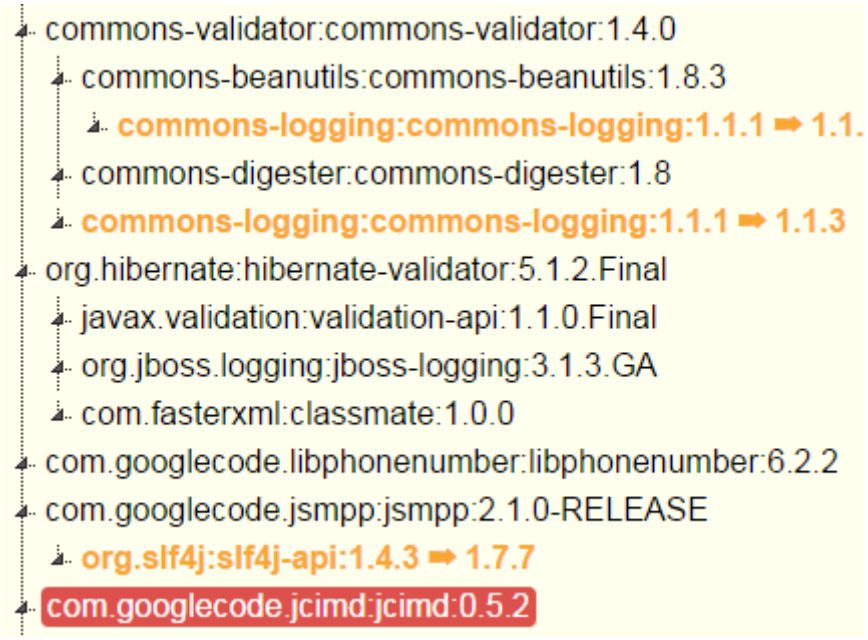
See pistikprogramm lisab projektile juurde viis tegumit. Nendeks tegumiteks on:

- *dependencyReport,*
- *htmlDependencyReport,*
- *propertyReport,*
- *taskReport,*
- *projectReport.*

Projekti sõltuvuste analüüsimiseks saab kasutada neist kahte – *dependencyReport* ning *htmlDependencyReport*. Sisemiselt teevad need sedasama, mis Gradle'is juba olemasolev *dependencies* tegum, ainus erinevus on selles, et tulemist genereeritakse kas siis tekstifail (vt Joonis 10) või veebileht (vt Joonis 11).

```
compile - Compile classpath for source set 'main'.
+--- org.hsqldb:hsqldb:2.3.2
+--- org.hibernate:hibernate-entitymanager:4.3.8.Final
|    +--- org.jboss.logging:jboss-logging:3.1.3.GA
|    +--- org.jboss.logging:jboss-logging-annotations:1.2.0.Beta1
|    +--- org.hibernate:hibernate-core:4.3.8.Final
|    |    +--- org.jboss.logging:jboss-logging:3.1.3.GA
|    |    +--- org.jboss.logging:jboss-logging-annotations:1.2.0.Beta1
|    |    +--- org.jboss.spec.javax.transaction:jboss-transaction-api_1.2_spec
|    |    +--- dom4j:dom4j:1.6.1
|    |    |    \--- xml-apis:xml-apis:1.0.b2 -> 1.3.03
|    |    +--- org.hibernate.common:hibernate-commons-annotations:4.0.5.Final
|    |    |    +--- org.jboss.logging:jboss-logging:3.1.3.GA
|    |    |    \--- org.jboss.logging:jboss-logging-annotations:1.2.0.Beta1
```

Joonis 10. Gradle Project Report poolt genereeritud tekstifail



Joonis 11. Gradle Project Report poolt genereeritud veebileht

Gradle Project Report pistikprogrammi eeliseks on kasutamise lihtsus. Väljundis on ära toodud lisaks kasutusel olevate teekide versioonidele ka teekide kõige uuemad versioonid. Negatiivseks küljeks on parsimise keerukus. Sellisel kujul olevat tekstifaili ning HTML-faili võib olla keeruline parsida ning seda saab kasutada ainult Gradle projektide puhul. Maveni projektide puhul tuleks siis kasutada mõnda teist lahendust. Samuti ei ole väljundis informatsiooni turvariskide ega litsentside kohta. Samas see võiks olla mõeldav lahendus, kui oleks vaja analüüsida ainult Gradle faile.

2.4 Pistikprogramm Maven Projects Info Reports

Maven Projects Info Reports pistikprogramm [9] tuli välja aastal 2005. Seda pistikprogrammi arendatakse edasi siiani – viimane uuendus on olnud aastal 2015. Selleks, et seda pistikprogrammi kasutada saaks, tuleb Maveni *pom.xml* faili lisada selle sõltuvus (vt Joonis 12).

```
<dependency>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-project-info-reports-plugin</artifactId>
  <version>2.8</version>
  <type>maven-plugin</type>
</dependency>
```

Joonis 12. Maven Projects Info Reports pistikprogrammi sõltuvus

Maven Projects Info Reports pistikprogramm lisab Mavenile juurde 17 tegumit. Projekti sõltuvuste analüüsiks saab neist kasutada aga ainult ühte – *'project-info-reports:dependencies'*. Seda saab käivitada käsurealt käsitsi või siis lasta teha seda automaatselt iga kord kui projekt kompileeritakse. Väljundiks genereeritakse veebileht, kus on kirjas kõik kasutatud teegid, nende versioonid ja litsentsid (vt Joonis 13).

GroupId	ArtifactId	Version	Type	License
com.fasterxml.jackson.core	jackson-databind	2.4.2	jar	The Apache Software License, Version 2.0
com.fasterxml.jackson.datatype	jackson-datatype-hibernate4	2.4.2	jar	The Apache Software License, Version 2.0
commons-dbcp	commons-dbcp	1.4	jar	The Apache Software License, Version 2.0
commons-fileupload	commons-fileupload	1.3.1	jar	The Apache Software License, Version 2.0
de.schauderhaft.degraph	degraph-check	0.1.1	jar	The Apache License, Version 2.0
de.schauderhaft.degraph	degraph-core	0.1.1	jar	The Apache License, Version 2.0
javax.mail	javax.mail-api	1.5.2	jar	CDDL/GPLv2+CE
javax.servlet	javax.servlet-api	3.1.0	jar	CDDL + GPLv2 with classpath exception
joda-time	joda-time	2.6	jar	Apache 2

Joonis 13. Maven Projects Info Reports pistikprogrammi väljund

Maven Projects Info Reports pistikprogrammi eeliseks on kasutamise lihtsus. Projektiga liidestamine on ülilihtne ning väljundi saab genereerima panna automaatselt. Lisaks on selgelt eraldatud nii teegi väljaandja, nimi, versioon ja litsents. Negatiivseks küljeks on see, et analüüsida saab ainult Maveni projekte ning puudu on ka teegis esinevad turvariskid ja uuemate versioonide olemasolust teavitamine.

2.5 Pistikprogramm Versions Maven

Versions Maven pistikprogramm [10] on loodud aastal 2008. Pistikprogrammi on viimati uuendatud aastal 2015. Selle kasutamiseks tuleb Maveni *pom.xml* faili lisada selle sõltuvus (vt Joonis 14).

```
<dependency>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>versions-maven-plugin</artifactId>
  <version>2.2</version>
  <type>maven-plugin</type>
</dependency>
```

Joonis 14. Versions Maven pistikprogrammi sõltuvus

Versions Maven pistikprogrammi projektile lisamine annab juurde 24 tegumit. Projekti sõltuvuste analüüsi koha pealt pakub huvi ainult kaks – *'versions:display-dependency-updates*' ja '*versions:dependency-updates-report*'. Esimene neist annab informatsiooni, millistest teekidest on uuemaid versioone olemas – väljundi võib suunata näiteks tekstifaili (vt Joonis 15). Teine tegum genereerib veebilehe, kus saadud informatsioon on kantud tabelisse (vt Joonis 16).

dep.txt

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Crucem Elit 0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- versions-maven-plugin:2.2:display-dependency-updates (default-cli) @ crucem-elit
[INFO] The following dependencies in Dependencies have newer versions:
[INFO]   com.fasterxml.jackson.core:jackson-databind ........... 2.4.2 -> 2.5.3
[INFO]   com.fasterxml.jackson.datatype:jackson-datatype-hibernate4 ...
[INFO]                                                          2.4.2 -> 2.5.3
[INFO]   com.newrelic.agent.java:newrelic-agent ............... 3.11.0 -> 3.15.0
[INFO]   de.schauderhaft.degraph:degraph-check ................. 0.1.1 -> 0.1.3
[INFO]   de.schauderhaft.degraph:degraph-core .................. 0.1.1 -> 0.1.3
[INFO]   javax.mail:javax.mail-api ............................. 1.5.2 -> 1.5.3
[INFO]   joda-time:joda-time ....................................... 2.6 -> 2.7
[INFO]   junit:junit ............................................ 4.11 -> 4.12
[INFO]   org.hibernate:hibernate-core ............... 4.3.6.Final -> 5.0.0.Beta2
[INFO]   org.hibernate:hibernate-entitymanager ..... 4.3.6.Final -> 5.0.0.Beta2
```

Joonis 15. Versions Maven '*display-dependency-updates'* väljund

Dependencies

Status	Group Id	Artifact Id	Current Version	Scope	Classifier	Type	Next Version	Next Incremental	Next Minor	Next Major
⚠	com.fasterxml.jackson.core	jackson-databind	2.4.2	compile		jar		**2.4.3**	**2.5.0**	
⚠	com.fasterxml.jackson.datatype	jackson-datatype-hibernate4	2.4.2	compile		jar		**2.4.3**	**2.5.0**	
⚠	com.newrelic.agent.java	newrelic-agent	3.11.0	provided		jar			**3.12.0**	
✓	commons-dbcp	commons-dbcp	1.4	compile		jar				
✓	commons-fileupload	commons-fileupload	1.3.1	compile		jar				
⚠	de.schauderhaft.degraph	degraph-check	0.1.1	compile		jar		**0.1.2**		
⚠	de.schauderhaft.degraph	degraph-core	0.1.1	compile		jar		**0.1.2**		
⚠	javax.mail	javax.mail-api	1.5.2	compile		jar		**1.5.3**		
✓	javax.servlet	javax.servlet-api	3.1.0	compile		jar				
⚠	joda-time	joda-time	2.6	compile		jar			**2.7**	
⚠	junit	junit	4.11	test		jar		**4.12-beta-1**	**4.12**	
✓	org.apache.maven.plugins	maven-dependency-plugin	2.10	compile		maven-plugin				
⚠	org.hibernate	hibernate-core	4.3.6.Final	compile		jar				**4.3.7.Final**

Joonis 16. Versions Maven '*dependency-updates-report*' väljund

Teine tegum annab informatsiooni väga selgel kujul. Eraldi lahtrites on teegi väljaandja, nimi, versioon ning uuemad saadaolevad versioonid. Lisaks ei pea seda tegumit käsitsi käivitama, vaid saab panna selle automaatselt käivituma iga kord, kui projekti kompileeritakse. Negatiivseks küljeks on see, et analüüsida saab vaid Maveni projekte ning puudu on ka teegis olevad turvariskid ning teegi litsentsid.

2.6 Pistikprogramm Gradle Versions

Gradle Versions pistikprogramm [11] on loodud aastal 2012. Viimased uuendused sellele on tehtud aastal 2015. Selle loomisel on võetud eeskujuks Versions Maven pistikprogramm. Selleks, et seda pistikprogrammi kasutada tuleb *build.gradle* faili lisada selle pistikprogrammi sõltuvus ning seda rakendada (vt Joonis 17).

```
buildscript {
  repositories {
    jcenter()
  }
  dependencies {
    classpath 'com.github.ben-manes:gradle-versions-plugin:0.9'
    classpath 'org.codehaus.groovy:groovy-backports-compat23:2.3.5'
  }
}

apply plugin: 'com.github.ben-manes.versions'
```

Joonis 17. Gradle Versions sõltuvus ning pistikprogrammi rakendamine

Sõltuvuste analüüsimiseks tuleb kutsuda välja Gradle tegum nimega '*dependencyUpdates*'. Argumendiks saab sellele anda väljundi formaadi (vt Joonis 18). Hetkel toetab pistikprogramm järgnevaid formaate:

- *json*,
- *xml*,
- *plain*.

```
C:\Users\Janar\gradlew dependencyUpdates -DoutputFormatter=json,xml,plain
```

Joonis 18. Gradle Versions pistikprogrammi käsurealt käivitamine

Toetatud formaatidest on ilmselt kõige raskem parsida tavalist (*plain*) formaati, kuna ebaolulist informatsiooni on liiga palju (vt Joonis 19).

```
------------------------------------------------------------
: Project Dependency Updates (report to plain text file)
------------------------------------------------------------

The following dependencies are using the latest milestone version:
 - org.hsqldb:hsqldb:2.3.2

The following dependencies have later milestone versions:
 - org.spockframework:spock-core [0.7-groovy-1.8 -> 1.0-groovy-2.4]
 - org.springframework.ws:spring-ws-security [2.2.0.RELEASE -> 2.2.1.RELEASE]

Failed to determine the latest version for the following dependencies (use --info for details):
 - com.googlecode.jcimd:jcimd
```

Joonis 19. Gradle Versions formaat *plain*

Üks võimalus oleks kasutada *JSON*-formaati (vt Joonis 20) ning see otse andmebaasi salvestada – selleks ilmselt oleks otstarbekas kasutada kas *NoSQL* või *MongoDB* andmebaasi.


```
{
    "current": {
        "dependencies": [
            {
                "group": "org.hsqldb",
                "version": "2.3.2",
                "name": "hsqldb"
            }
        ],
        "count": 1
    },
    "exceeded": {
        "dependencies": [
        ],
        "count": 0
    },
    "outdated": {
        "dependencies": [
            {
                "group": "org.springframework.security",
                "available": {
                    "release": null,
                    "milestone": "4.0.1.RELEASE",
                    "integration": null
                },
                "version": "3.2.5.RELEASE",
                "name": "spring-security-web"
            }
        ],
        "count": 1
    },
    "unresolved": {
        "dependencies": [
            {
                "group": "com.googlecode.jcimd",
                "version": "0.5.2",
                "reason": "Could not find any version that matches com.googlecode.jcimd:jcimd:latest.milestone.",
                "name": "jcimd"
            }
        ],
        "count": 1
    },
    "count": 3
```


Joonis 20. Gradle Versions formaat *json*

Teine võimalus on kasutada *XML*-formaati (vt joonis 21) ning seda parsida.


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
  <count>3</count>
  <current>
    <count>1</count>
    <dependencies>
      <dependency>
        <name>hsqldb</name>
        <group>org.hsqldb</group>
        <version>2.3.2</version>
      </dependency>
    </dependencies>
  </current>
  <outdated>
    <count>1</count>
    <dependencies>
      <outdatedDependency>
        <name>super-csv</name>
        <group>net.sf.supercsv</group>
        <version>2.2.0</version>
        <available>
          <milestone>2.3.1</milestone>
        </available>
      </outdatedDependency>
    </dependencies>
  </outdated>
  <exceeded>
    <count>0</count>
    <dependencies/>
  </exceeded>
  <unresolved>
    <count>1</count>
    <dependencies>
      <unresolvedDependency>
        <name>jcimd</name>
        <group>com.googlecode.jcimd</group>
        <version>0.5.2</version>
        <reason>Could not find any version that matches com.googlecode.jcimd:jcimd:latest.milestone.</reason>
      </unresolvedDependency>
    </dependencies>
  </unresolved>
</response>
```


Joonis 21. Gradle Versions formaat *xml*

Gradle Versions pistikprogrammi on mugav kasutada – väljundfaili loomist saab automatiseerida. Lisaks saab formaadi ise valida ning vajaliku informatsiooni kättesaamine peaks olema üpris lihtne. Suurimaks miinuseks on see, et analüüsida saab ainult Gradle projekte. Samuti on puudu informatsioon teekides esinevate turvariskide ning litsentside kohta. Potentsiaali sellel pistikprogrammil on ning juhul kui oleks vaja analüüsida ainult Gradle projekte, siis oleks see kindlasti hea valik.

2.7 Pistikprogramm Maven Dependency

Maven Dependency pistikprogramm [12] tuli välja aastal 2002. Viimane versioon sellest – versioon 2.10 – tuli välja aastal 2015. See pistikprogramm võimaldab lisaks projekti sõltuvuste analüüsile ka teeke kopeerida, lahti pakkida ja repositooriumist koos lähtekoodiga alla laadida. Pistikprogrammi kasutamiseks tuleb Maveni *pom.xml* faili lisada selle sõltuvus (vt Joonis 22).

```
<dependency>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.10</version>
  <type>maven-plugin</type>
</dependency>
```

Joonis 22. Maven Dependency pistikprogrammi sõltuvus

Maven Dependency pistikprogramm lisab projektile juurde 21 tegumit. Projekti sõltuvuste analüüsiks on mõistlik kasutada ühte järgnevaist:

- ’*dependency:analyze*’,
- ’*dependency:analyze-only*’,
- ’*dependency:analyze-report.*’

Esimesed kaks neist teevad sisuliselt sama asja – ainuke erinevus on selles, et esimene neist on mõeldud eraldiseisvaks kasutamiseks ning teine on mõeldud projekti elutsüklisse lisamiseks. Analüüsi tulemuse saab suunata faili (vt Joonis 23). Kolmas tegum analüüsib projekti ning seejärel loob tulemusest veebilehe nagu näha Jooniselt 24. Analüüsitakse baitkoodi, mistõttu on projekti kokkuehitamine kohustuslik (tegumid teevad seda ka ise automaatselt). Väljundis on kirjas teegi grupp, nimi ja versioon.

```
1  [INFO] --- maven-dependency-plugin:2.3:analyze (default-cli) @ crucem-elit ---
2  [WARNING] Used undeclared dependencies found:
3  [WARNING]    org.springframework.data:spring-data-commons:jar:1.8.4.RELEASE:compile
4  [WARNING]    org.springframework:spring-web:jar:4.1.0.RELEASE:compile
5  [WARNING] Unused declared dependencies found:
6  [WARNING]    org.springframework:spring-context-support:jar:4.1.0.RELEASE:compile
7  [WARNING]    org.springframework:spring-jdbc:jar:4.1.0.RELEASE:compile
8  [WARNING]    org.projectlombok:lombok:jar:1.14.6:provided
9  [WARNING]    commons-dbcp:commons-dbcp:jar:1.4:compile
10 [WARNING]    postgresql:postgresql:jar:9.0-801.jdbc4:compile
11 [WARNING]    org.springframework.hateoas:spring-hateoas:jar:0.16.0.RELEASE:compile
```

Joonis 23. Tegumi ’*dependency:analyze*’ väljund

Dependency Analysis

Used and declared dependencies

GroupId	ArtifactId	Version	Scope	Classifier	Type	Optional
junit	junit	4.11	test		jar	false
javax.servlet	javax.servlet-api	3.1.0	compile		jar	false
org.springframework	spring-tx	4.1.0.RELEASE	compile		jar	false
org.springframework	spring-webmvc	4.1.0.RELEASE	compile		jar	false
org.springframework.security	spring-security-web	3.2.5.RELEASE	compile		jar	false

Used but undeclared dependencies

GroupId	ArtifactId	Version	Scope	Classifier	Type	Optional
org.springframework.data	spring-data-commons	1.8.4.RELEASE	compile		jar	false
commons-io	commons-io	1.4	compile		jar	false
org.springframework	spring-web	4.1.0.RELEASE	compile		jar	false

Joonis 24. Tegumi '*dependency:analyze-report*' väljund

Maven Dependency pistikprogrammi eeliseks on kasutamise lihtsus. Projektiga liidestamine on lihtne ning analüüsimist saab automatiseerida. Negatiivseteks külgedeks on see, et analüüsida saab ainult Maveni projekte ning saadavat informatsiooni on vähe – puudu on info uuematest versioonidest, turvariskidest, litsentsidest. Lisaks, kuna analüüsitakse baitkoodi, võivad mõned kasutusel olevad teegid jääda leidmata.

2.8 OWASP Dependency Check

OWASP (*Open Web Application Security Project*) Dependency Check [13] on aastal 2012 välja antud utiliitprogramm, mis identifitseerib projekti sõltuvused ning kontrollib, kas neil on teadaolevaid turvariske. Turvariskide kontrolliks kasutatakse NVD (*National Vulnerability Database*) andmebaase. Viimane versiooniuuendus on sellele tulnud aastal 2015.

OWASP Dependency Check kasutamiseks on mitmeid erinevaid võimalusi. Võimalus on kasutada:

- käsurida,
- Maveni pistikprogrammi,
- Anti tegumit,
- Jenkinsi pistikprogrammi.

Sõltuvuste analüüsimiseks kasutatakse mitmeid erinevaid analüsaatoreid. Olemas on palju erinevaid analüsaatoreid, kõige tähtsamad on ilmselt *jarAnalyzer* ja *archieveAnalyzer*. Andmeid kogutakse iga projektis oleva teegi kohta ning seejärel hinnatakse kui kindlad kogutud tõendid turvariskide kohta on. Hinnatakse järgneval skaalal:

- madal (*low*),
- keskmine (*medium*),
- kõrge (*high*),
- kõige kõrgem (*highest*).

Väljundis on kirjas teegi grupp, nimi, versioon ning turvariskide arv. Väljundit saab vaadata nii *XML*-formaadis (vt Joonis 25) kui ka veebilehelt (vt Joonis 26).

```
<dependency>
    <fileName>NUL</fileName>
    <filePath>NUL</filePath>
    <md5>159f2ca0cc8c40bcaa2a48c3f1c5690f</md5>
    <sha1>159f2ca0cc8c40bcaa2a48c3f1c5690f</sha1>
    <description>372</description>
    <license>UNKNOWN</license>
    <evidenceCollected>
        <evidence>
            <source>dependency-track</source>
            <name>name</name>
            <value>hibernate-entitymanager</value>
        </evidence>
        <evidence>
            <source>dependency-track</source>
            <name>vendor</name>
            <value>org.hibernate</value>
        </evidence>
        <evidence>
            <source>file</source>
            <name>name</name>
            <value>NUL</value>
        </evidence>
    </evidenceCollected>
</dependency>
```

Joonis 25. OWASP Dependency Check *XML*-formaadis väljund

Dependency	CPE	GAV	Highest Severity	CVE Count	CPE Confidence	Evidence Count
axis-1.4.jar	cpe:/a:apache:axis:1.4	axis:axis:1.4	Medium	2	HIGHEST	10
axis2-kernel-1.4.1.jar	cpe:/a:apache:axis2:1.4.1	org.apache.axis2:axis2-kernel:1.4.1	High	6	HIGHEST	15
commons-fileupload-1.2.1.jar	cpe:/a:apache:commons_fileupload:1.2.1	commons-fileupload:commons-fileupload:1.2.1	Medium	2	HIGHEST	16
commons-httpclient-3.1.jar	cpe:/a:apache:commons-httpclient:3.1 cpe:/a:apache:httpclient:3.1	commons-httpclient:commons-httpclient:3.1	Medium	1	LOW	9
daytrader-ear-2.1.7.ear: dt-ejb.jar	cpe:/a:apache:geronimo:2.1.7	org.apache.geronimo.daytrader:daytrader-ejb:2.1.7	High	2	HIGHEST	10
daytrader-ear-2.1.7.ear: geronimo-jaxrpc_1.1_spec-2.0.0.jar	cpe:/a:apache:geronimo:2.0	org.apache.geronimo.specs:geronimo-jaxrpc_1.1_spec:2.0.0	High	4	HIGHEST	11

Joonis 26. OWASP Dependency Check väljund veebilehel

OWASP Dependecy Check eeliseks on see, et informatsiooni saab kergesti parsitaval kujul ning infot on lisaks teegi nimele ja versioonile ka turvariskide kohta. Negatiivseks küljeks on, et esineda võib nii valenegatiivseid kui ka valepositiivseid, analüüs võtab üpris kaua aega ning puudub informatsioon uuemate versioonide olemasolu kohta.

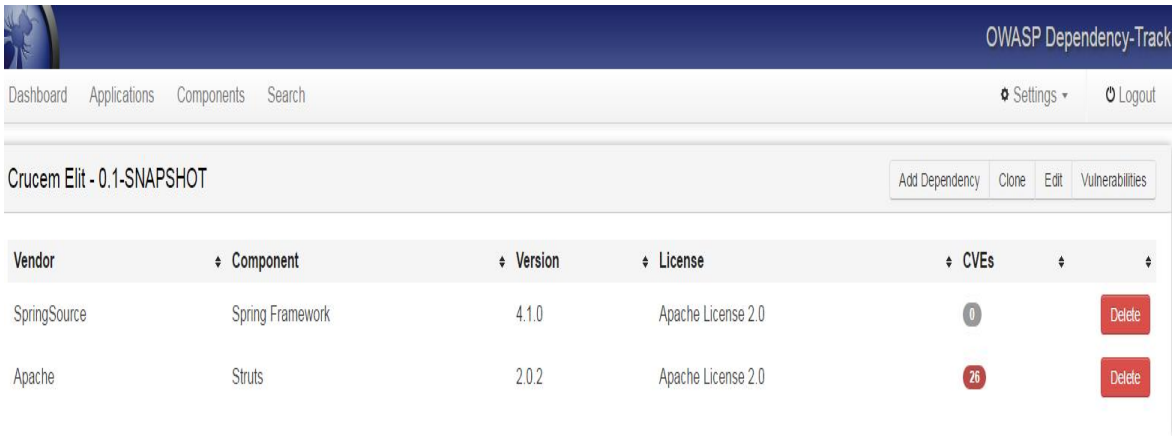
2.9 OWASP Dependency Track

OWASP Dependency Track [14] on aastal 2015 välja antud Java veebirakendus. Rakendus ise on kokkupakitud *WAR*-formaadis ning selle käivitamiseks on vaja mingit rakendusserverit nagu näiteks Tomcat (versioon 7+) või Jetty (versioon 8+). Vajalik on ka Java 7+ olemasolu. Rakendus kasutab info hoidmiseks automaatselt loodud lokaalset andmebaasi.

OWASP Dependency Track võimaldab dokumenteerida projektis kasutusel olevaid teeke. Teekide lisamisel tuleb täpsustada teegi:

- väljaandja,
- nimi,
- versioon,
- litsents,
- programmeerimiskeel.

Peale teegi lisamist toimub automaatne teegi turvariskide kontroll kasutades selleks OWASP Dependency Checki. Seejärel saab hakata teeki lisama erinevatele projektidele. Olemas on ka otsinguvõimalus, mis näitab ära, mis projektides mingi kindel teek kasutusel on. Projekti vaates on lisaks teegi informatsioonile näha ka turvariskide arv (vt Joonis 27).



Joonis 27. OWASP Dependency Track projekti vaade

OWASP Dependency Track positiivseks küljeks on see, et veebirakendus on juba valmis. Olemas on ka andmebaas ning läbi veebirakenduse on võimalik otsida, kus projektis mingi kindel teek kasutusel on. Ent siiski on sellel ka negatiivseid külgi. Kõige suurem miinus selle juures on see, et projekte ei ole võimalik automaatselt analüüsida – kogu informatsioon projektis olevate teekide kohta tuleb sisse kanda ning uuendada käsitsi. Lisaks on puudu informatsioon teekide uuemate versioonide kohta ning rakendust võib esialgu olla keeruline kasutada. Projekti autoril on tulevikus plaanis juurde lisada projektide automaatne analüüsimine, kasutajaliidese ilusamaks muutmine ja konfigureeritava teavituste süsteemi loomine. Hetkel aga ilma automaatse projekti sõltuvuste analüüsita seda kasutada ei ole mõistlik.

2.10 Google DepAn

Google DepAn [15] on projekti sõltuvuste analüüsivahend, mille on välja andnud Google. Viimane versioon sellest on välja tulnud aastal 2009. Ametlikult on olemas ainult Linuxi väljalase, kuid võimalik on ka ise Windowsi jaoks see kokku ehitada. Kasutamiseks on vajalik arvutisse alla laadida Eclipse.

Google DepAn peaks võimaldama kiiresti analüüsida väga suuri Java projekte. Väljundit peaks saama vaadata graafiliselt. Kahjuks on selle dokumentatsioon väga primitiivne ning aegunud. Tehes kõike dokumentatsiooni järgi, ei hakanud see tööle ei Linuxi ega ka Windowsi peal. Proovitud sai mitmete erinevate Eclipse ning Java versioonidega.

2.11 Programm eDepend

eDepend [16] on Soyateci poolt aastal 2006 välja antud Eclipse tööriist, mis algselt kuulus eUML2 juurde. Nüüdseks on eUML2 jagunenud neljaks väiksemaks osaks, millest üks on eDepend. Viimane uuendus sellele on olnud aastal 2013.

Tööriist ise ei ole vabavaraline – litsents maksab 690 eurot. See tasu aga on ühekordne ning seejärel saaks seda kasutada iga projekti juures. Soyatec pakub oma kodulehel prooviversiooni, kuid selle Eclipse installeerimisel ilmnes probleeme litsentsiga. Uurides nende foorumit selgus, et see probleem esineb paljudel ning üle viie aasta pole lahendust tulnud. Võimalik on kasutada ka vähesemate võimalustega tasuta versiooni, kuid selgus, et see ei suuda projekti sõltuvusi analüüsida.

2.12 Tööriistade võrdlus

Tabelis 1 on esitatud kirjeldatud tööriistade võrdlus. Siit on näha, et sellist tööriista, mis vastaks kõikidele nõuetele, ei leitud. Lisaks on näha, et seatud eesmärkide täitmiseks sobivad kõige paremini OWASP poolt välja antud tööriistad.

Tabel 1. Tööriistade võrdlustabel

	Teegi nimi ja versioon	**Uusima versiooni tuvastamine**	**Litsentsid**	**Turva-riskid**	**Maven tugi**	**Gradle tugi**
Codepro AnalytiX	✓				✓	✓
Degraph					✓	✓
Gradle Project Report	✓	✓				✓
Maven Projects Info	✓		✓		✓	
Versions Maven	✓	✓			✓	
Gradle Versions	✓	✓				✓
Maven Dependency	✓				✓	
OWASP Dependency Check ja Track	✓		✓	✓	✓	✓

3. Rakenduse kirjeldus

Rakendus peab suutma tuvastada, mis teegid projektis kasutusel on, mis litsents neil on ning kas teekidest on uuemaid versioone,. Lisaks peab rakendus andma informatsiooni teekides esinevatest turvariskidest. Eelnevale analüüsile toetudes otsustati valida aluseks OWASPi Dependency Track veebirakendus.

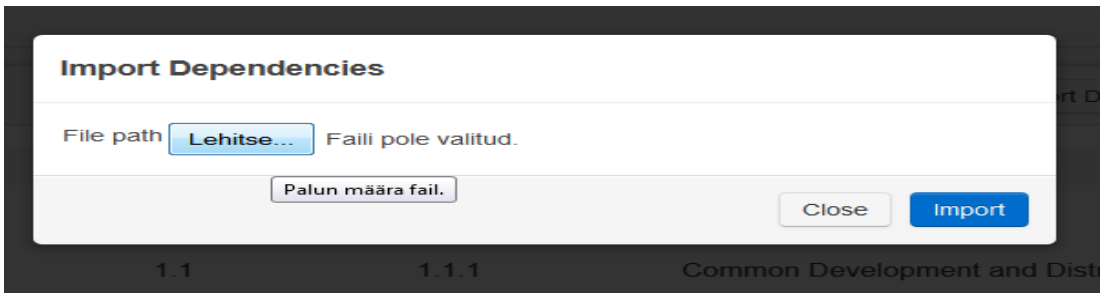
Rakenduses on juba olemas teekide lisamise, projektide haldamise ning otsingu võimalus – vaja on implementeerida projektide importimine ja andmebaasi automaatne täitmine ning teegi uusima versiooni tuvastamine. Lisaks on vaja muuta seda, kuidas litsentse tuvastatakse, sest hetkel on väljund ebaühtlasel kujul. Lisaks tuleb koodi optimeerida ning refaktoreerida. Seega tuleb muudatusi teha nii kasutajaliideses, andmebaasis kui ka serveri poolel. Alljärgnevalt kirjeldatud funktsionaalsus on kõik loodud bakalaureusetöö raames, kuna vastavad võimalused puudusid OWASP Dependency Track rakendusest.

3.1 Projektide importimine ning andmebaasi automaatne täitmine

Väga oluline on, et projekti sõltuvusi ei peaks käsitsi sisse kandma. Selle automatiseerimiseks kasutatatakse OWASPi Dependency Check utiliitprogrammi, millega analüüsitakse etteantud arhiivifaili. See suudab analüüsida nii Maveni kui ka Gradle projekte. Genereeritud raportist saadakse kätte kõik projektis kasutusel olevad teegid ning nende versioonid. Need tehakse serveri poolel objektideks ning seejärel salvestatakse andmebaas.

Kasutajaliidese poole pealt lisati „impordi“ nupp (vt Joonis 28), millele vajutades avaneb modaalvorm (vt Joonis 29), kus kasutaja saab üles laadida faili oma failisüsteemist.

Joonis 28. Projekti vaate juures olev ’*Import Dependencies*’ nupp



Joonis 29. Faili üleslaadimise modaalvorm

Serveri poolel kirjutati loogika, mis võtab üleslaetud faili vastu, teostab selle peal analüüsi, koostab vastavalt analüüsi tulemusele Java objektid ning salvestab loodud objektid andmebaasi. Andmebaasi ennast muutma ei pidanud.

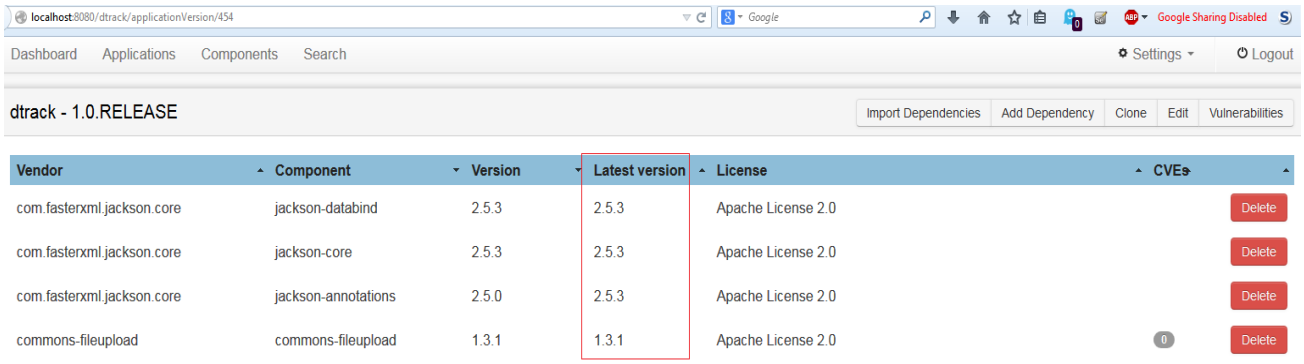
3.2 Teegi uusima versiooni tuvastamine

Teine oluline nõue oli, et rakendus annaks informatsiooni sellest, kui teegist on saadaval uuemaid versioone. Selle funktsionaalsuse implementeerimiseks kasutati Maveni REST-liidest, mille vastu on võimalik päringuid teha ning saada vastuseid *JSON*-formaadis.

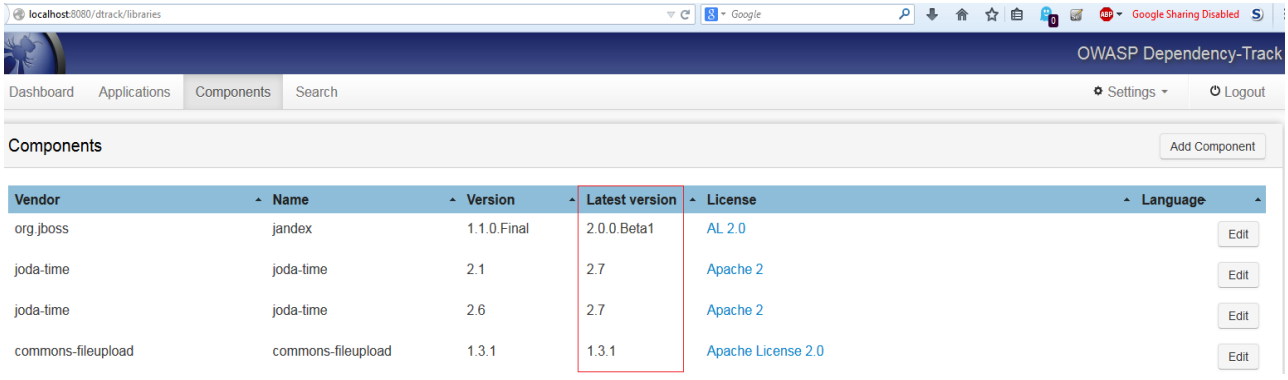
Muudatusi tuli teha kõigis kolmes kihis:

- andmebaasis,
- serveri poolel,
- kasutajaliideses.

Andmebaasi tuli lisada uus väli, kus hoitakse teegi viimase versiooni numbrit. Serveri poolel tuli luua meetod, mis iga kindla aja tagant (24h) pärib Maveni repositooriumist teegi uusima versiooni numbrit ning salvestab selle. Selleks kasutatatakse päringut[1], kus *group* tuleb asendada teegi grupiga ning *artifact* teegi nimega. Kasutajaliideses tuli tabelitesse, kus kuvatakse teeke (vt Joonised 30 ja 31), lisada loodud väli.



Joonis 30.Teegi uusim versioon konkreetse projekti vaates

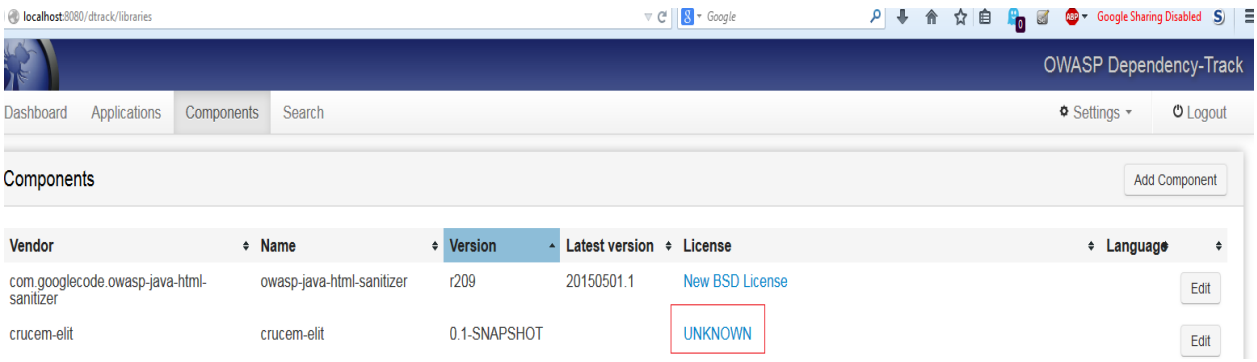


Joonis 31.Teegi uusim versioon teekide vaates

[1] https://search.maven.org/solrsearch/select?q=g:{group}+AND+a:{artifact}&core&rows=20&wt=json

3.3 Teegi litsentsi tuvastamine

OWASP Dependency Check, millega üleslaetud failide analüüs toimub, suudab ise tuvastada teekide litsentse. Need aga on tihti erinevates formaatides – mõnel juhul on antud litsentsi nimi, teisel kõigest link litsentsi asukohale ja mõnikord nii litsents kui ka viide. Selleks, et litsentsi nime saaks kätte ühtlaselt, teeme taaskord päringu vastu Maveni REST-liidest. Selleks päritakse teegi *pom.xml* faili ning regulaaravaldisega tehakse kindlaks, kas seal on olemas informatsioon teegi litsentside kohta. Üks võimalus on teha päring[1], kus *group* tuleb asendada teegi grupiga, *artifact* teegi nimega ning *version* teegi versiooniga. See variant sai ka selles rakenduses kasutusele võetud. Kui *pom.xml* failis litsentsi ei ole, siis määratakse litsentsiks *'UNKNOWN'* nagu näha jooniselt 32.



Joonis 32. Litsents, mida ei suudetud tuvastada

3.4 Optimeerimine ja refaktoreerimine

OWASP Dependency Track oli esialgu mõeldud käsitsi projektide sõltuvuste haldamiseks. Kui lisada teeke ükshaaval, ei võta teegi analüüs väga palju aega. Kui aga analüüsida tervet projekti ning automaatselt lisada kõik seal kasutusel olevad teegid, muutub rakendus väga aeglaseks. Kui lisati 50 teeki, siis programmi töö võttis aega peaaegu pool tundi. Tehti kindlaks, et peale iga teegi lisamist uuendati kohalikku andmebaasi, kus hoitakse infot turvariskide kohta. Selle peale kulub aga liiga palju aega. Probleemi lahenduseks muudeti seda, kui tihti informatsiooni uuendatakse – nüüd uuendatakse seda siis, kui rakendus käima pannakse ning seejärel kord iga 24h tagant.

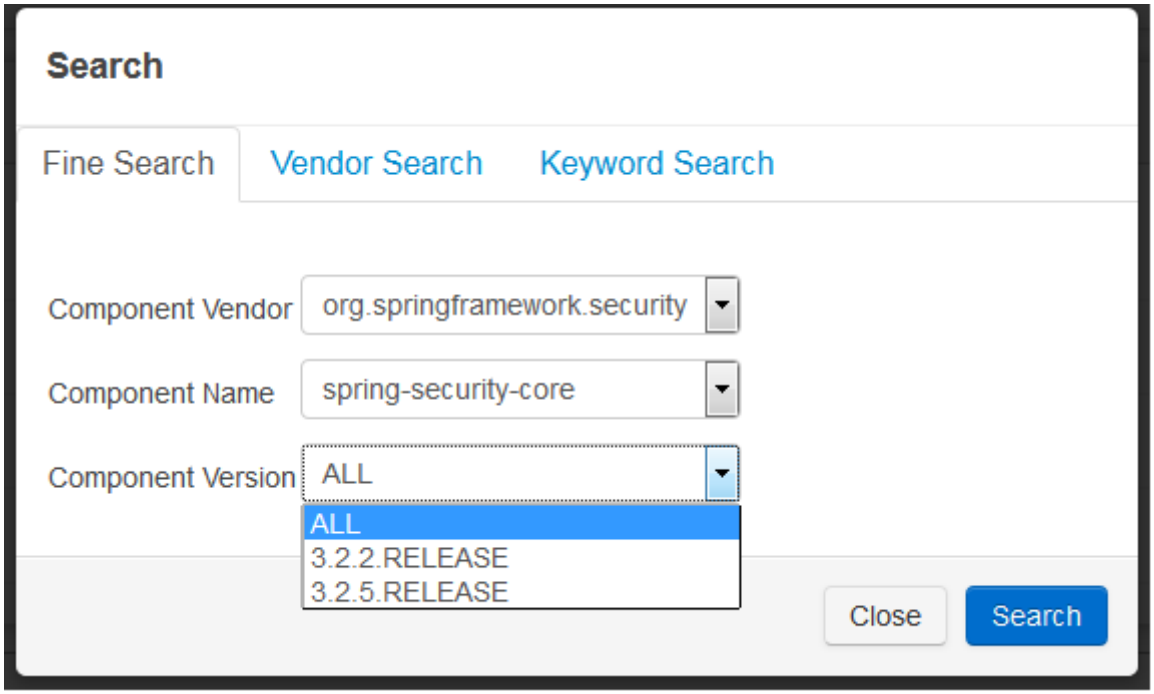
Koodi loetavamaks tegemiseks võeti kasutusele Project Lombok [17]. See võimaldas kaotada ära kõik *get- ja set*-meetodid, *toString* meetodid ning *equals* ja *hashCode* meetodid kasutades selleks *@Data* annotatsiooni. Samuti lihtsustas see vajaliku informatsiooni logimist kasutades selleks *@Slf4j* annotatsiooni.

Väiksemate uuendustena võib välja tuua paari uue teegi kasutuselevõtu, olemasolevate teekide versioonide uuendamise ning seadete muutmise.

3.5 Tulemus

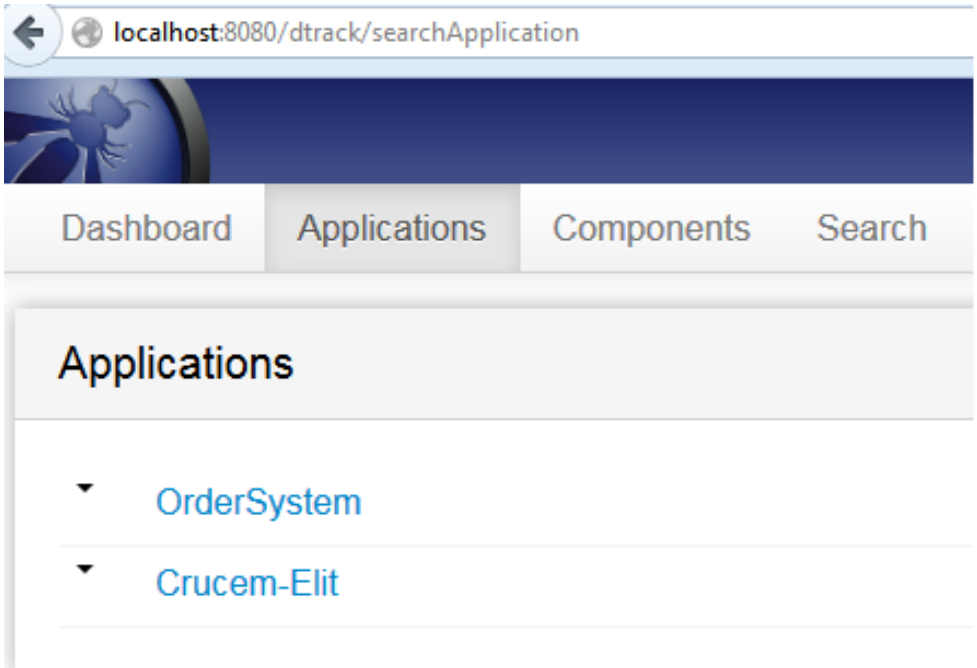
Valmis rakendus, mis vastab seatud eesmärkidele. Otsida saab nii teegi grupi, nime kui ka versiooni järgi (vt Joonis 33). Olemas on ka võtmesõna otsing, mis otsib võtmesõnade järgi.

[1] https://search.maven.org/remotecontent?filepath={group}/{artifact}/{version}/{artifact}-{version}.pom



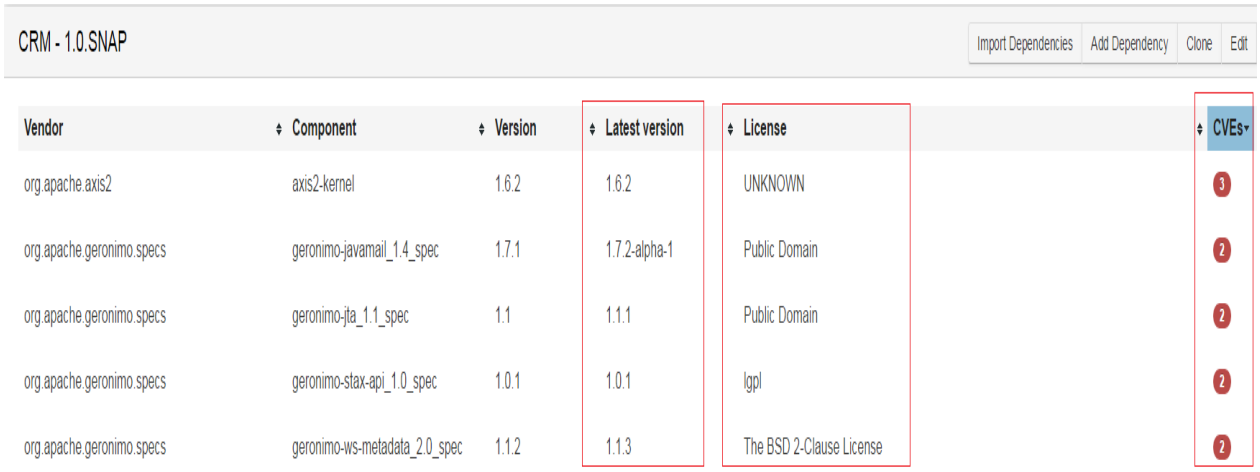
Joonis 33. Otsing

Vajutades ’*Search*’ nupule tulevad nähtavale ainult need rakendused, kus antud teek kasutusel on (vt Joonis 34).



Joonis 34. Otsingu tulemus

Lisaks on rakenduses olemas informatsioon nii teekide uusimate versioonide, litsentside kui ka turvariskide kohta nagu näha jooniselt 35.



Joonis 35. Projekti sõltuvuste vaade

Projektide importimine on testitud ja töötab nii *WAR*-, *EAR*-, *JAR*- kui ka *ZIP*-formaadis failidega.

4. Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli uurida erinevaid lahendusi projekti sõltuvuste analüüsimiseks, neist parim välja valida ning luua rakendus, mis võimaldaks teegi nime järgi otsida, missugustes projektides see kasutusel on. Lisaks pidi rakendus andma informatsiooni sellest, kas teegist on saadaval uuemaid versioone.

Bakalaureusetöö teoreetilise osa käigus uuriti 11 erinevat tööriista projekti sõltuvuste analüüsimiseks. Mitte ükski neist tööriistadest ei olnud võimeline saama kätte kogu soovitud informatsiooni. Kõige lähemal oli OWASP Dependency Check, mis suutis tuvastada kõik peale uusima versiooni numbri. Edasisel uurimisel selgus, et Maveni repositooriumi REST-liidest kasutades on see võimalik üpris lihtsalt kätte saada. Lisaks selgus, et on olemas OWASP Dependency Track projekt, mille veebirakendus kasutab analüüsimiseks utiliitprogrammi OWASP Dependency Check.

Bakalaureusetöö praktilise osa käigus täiendati OWASP Dependency Track veebirakendust. Juurde lisati projekti importimisvõimalus ning teegi uusima versiooni ja litsentsi tuvastamine. Lisaks tuli rakendust vähesel määral refaktoreerida ja optimeerida.

Bakalaureusetöö püstitatud eesmärgid said täidetud ning valminud veebirakendus võimaldab lihtsalt leida projekte, kus mingi kindel teek on kasutusel ning lisaks sellele, et antakse informatsiooni teegi uusimatest versioonidest, on olemas ka informatsioon teegi litsentside ning turvariskide kohta.

Valminud veebirakendust on kindlasti võimalik edasi arendada. Üks arendussuund oleks teavitussüsteemi loomine – näiteks võiks rakendus saata emaili, kui mõnes teegis on teatavad turvariskid. Teine suund oleks rakendusele luua oma REST-liides, mille kaudu saaks infot edastada teistele programmidele. Kolmas võimalus on oodata OWASP Dependency Track 2.0 versiooni väljatulekut ning sinna peale hakata edasiarendusi tegema.

Loodud lahendust on plaanis juurutada Nortalis kõikide projektide analüüsimisel.

5. Kasutatud kirjandus

[1] WhiteSource. [Veebis, 30.04.2015]. http://www.whitesourcesoftware.com/

[2] BlackDuck. [Veebis, 30.04.2015]. https://www.blackducksoftware.com/

[3] Maven. [Veebis, 30.04.2015]. https://maven.apache.org/

[4] Gradle. [Veebis, 30.04.2015]. https://gradle.org/

[5] CodePro AnalytiX. [Veebis, 10.05.2015]. https://developers.google.com/java-dev-tools/codepro/doc/

[6] Degraph. [Veebis, 10.05.2015]. https://schauder.github.io/degraph/

[7] yEd. [Veebis, 10.05.2015]. https://www.yworks.com/en/products/yfiles/yed/

[8] Gradle Project Report pistikprogramm. [Veebis, 10.05.2015]. http://gradle.org/docs/current/userguide/project_reports_plugin.html

[9] Maven Projects Info Reports pistikprogramm. [Veebis, 10.05.2015]. https://maven.apache.org/plugins/maven-project-info-reports-plugin/index.html

[10] Versions Maven pistikprogramm. [Veebis, 10.05.2015]. http://mojo.codehaus.org/versions-maven-plugin/index.html

[11] Gradle Versions pistikprogramm. [Veebis, 10.05.2015]. https://github.com/ben-manes/gradle-versions-plugin

[12] Maven Dependency pistikprogramm. [Veebis, 12.05.2015]. https://maven.apache.org/plugins/maven-dependency-plugin/

[13] OWASP Dependency Check. [Veebis, 12.05.2015]. https://www.owasp.org/index.php/OWASP_Dependency_Check

[14] OWASP Dependency Track. [Veebis, 12.05.2015]. https://www.owasp.org/index.php/OWASP_Dependency_Track_Project

[15] Google DepAn. [Veebis, 12.05.2015]. https://code.google.com/p/google-depan/

[16] eDepend. [Veebis, 12.05.2015]. http://www.soyatec.com/euml2/features/eDepend/

[17] Project Lombok. [Veebis, 13.05.2015]. https://projectlombok.org/

Lisad

I. Mõisted

Teek Kollektsioon komponente, mis on mõeldud korduvkasutuseks programmides.	Library Collection of resources used by computer programs.
Tegum Iseseisvana täidetav programmiosa.	Task, goal A single piece of work for a build, such as compiling classes or generating Javadoc.
Pistikprogramm Tarkvarakomponent, mis lisab tarkvarale mingi kindla lisavõimaluse.	Plugin A Software component that adds a specific feature to an existing software application.

II. Rakendus

Rakendus on vabavaralisena kättesaadav GitHubis https://github.com/janaroj/dependency-track

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Janar Ojalaid** (sünnikuupäev: 02.04.1993)
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Projekti sõltuvuste andmebaas,
(*lõputöö pealkiri*)

mille juhendajad on Priit Liivak ja Helle Hein.
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **14.05.2015**