Formulierung und Durchführung von Modultests mit KBUnit

Administrationsdokumentation

Inhalt

1 Überblick

Der Erfolg eines Softwareprojektes ist maßgeblich abhängig von der Einbindung des Auftraggebers beziehungsweise der Auftraggeberin, im folgenden Text beide mit Auftraggeber bezeichnet, in den Entwicklungsprozess. Nur unter seiner Mithilfe können sich ändernde Anforderungen zeitnah im Entwicklungsprozess abgebildet und bisherige Teilentwicklungen überprüft werden. In der Praxis ist es bisher teilweise noch verbreitet, dass der Auftraggeber erst bei Durchführung sogenannter Akzeptanztests die Funktionsfähigkeit eines (Teil-)Programms überprüft. Wünschenswert aber ist eine möglichst frühe Einbindung des Auftraggebers in den Qualitätssicherungsprozess, beispielsweise bei der Durchführung von Modultests. Falls nicht der Auftraggeber selbst für Modultests schon zur Verfügung steht, dann sollten es ein anderer Wissensträger oder eine andere Wissensträgerin wie beispielsweise Softwaretester und Softwaretesterinnen, im folgenden Text mit Wissensträger bezeichnet, sein. Modultests ermöglichen die Identifizierung von Fehlern einzelner Programmteile (z. B. einer Klasse oder Methode) zum Zeitpunkt ihrer Entwicklung. Weit verbreitet für die Realisierung von Modultests sind Tools, mit denen Testfälle in der Programmiersprache des zu prüfenden Moduls codiert werden. Da der Wissensträger in der Regel nicht über Programmierkenntnisse verfügt, fallen Modultests bisher ausschließlich in den Aufgabenbereich der Entwicklungsabteilung. Wenn man sich aber vor Augen führt, dass der Wissensträger die Funktionsweise eines Moduls aufgrund seiner umfassenderen geschäftsspezifischen Kenntnisse besser einschätzen kann als ein Entwickler oder eine Entwicklerin, im folgenden Text beide mit Entwickler bezeichnet, verspielt man eine wesentliche Chance zur Früherkennung von Programmfehlern und den damit verbundenen erfolgreichen Projektabschluss. Um einem Wissensträger ohne Programmierkenntnisse die Möglichkeit von Modultests einzuräumen, wurde die Java-Anwendung KBUnit (*Knowledge Based Unit Testing Application*) entwickelt. Mit dieser kann auf die von einem Entwickler definierten und speziell parametrisierten JUnit Testfälle zugegriffen werden.
Eine Version der in Java geschriebenen Anwendung KBUnit wurde unter anderem auf der Software Engineering 2017, auf dem Software QS-Tag 2018 und auf dem Software QS-Tag 2022 vorgestellt: Oesing, Ursula; Georgiev, Alexander; Langenbrink, Jahn; Jonker, Stefan: *Agiles Testen: Auch Anwender können Unit Tests* in J. Jürjens, K. Schneider (Hrsg.): Software Engineering 2017, LNI – Proceedings Series oft the GI, Köllen Druck + Verlag GmbH Bonn, ISBN 978-3-88579-661-9.

Die vorliegende Version der Anwendung KBUnit unterstützt die Schritte 0 bis 6 der Abbildung 1, siehe unten. Sie wurde von Philipp Sprengholz, Alexander Georgiev, Patrick Pete, Yannis Herbig, Gokhan Arslan, Iris Grcic, Siraj Taeb, Alexander Kokoreff und Ursula Oesing entwickelt. Die vorliegende Anwendungsdokumentation wurde von Philipp Sprengholz und Ursula Oesing entwickelt und von Patrick Pete ergänzt.

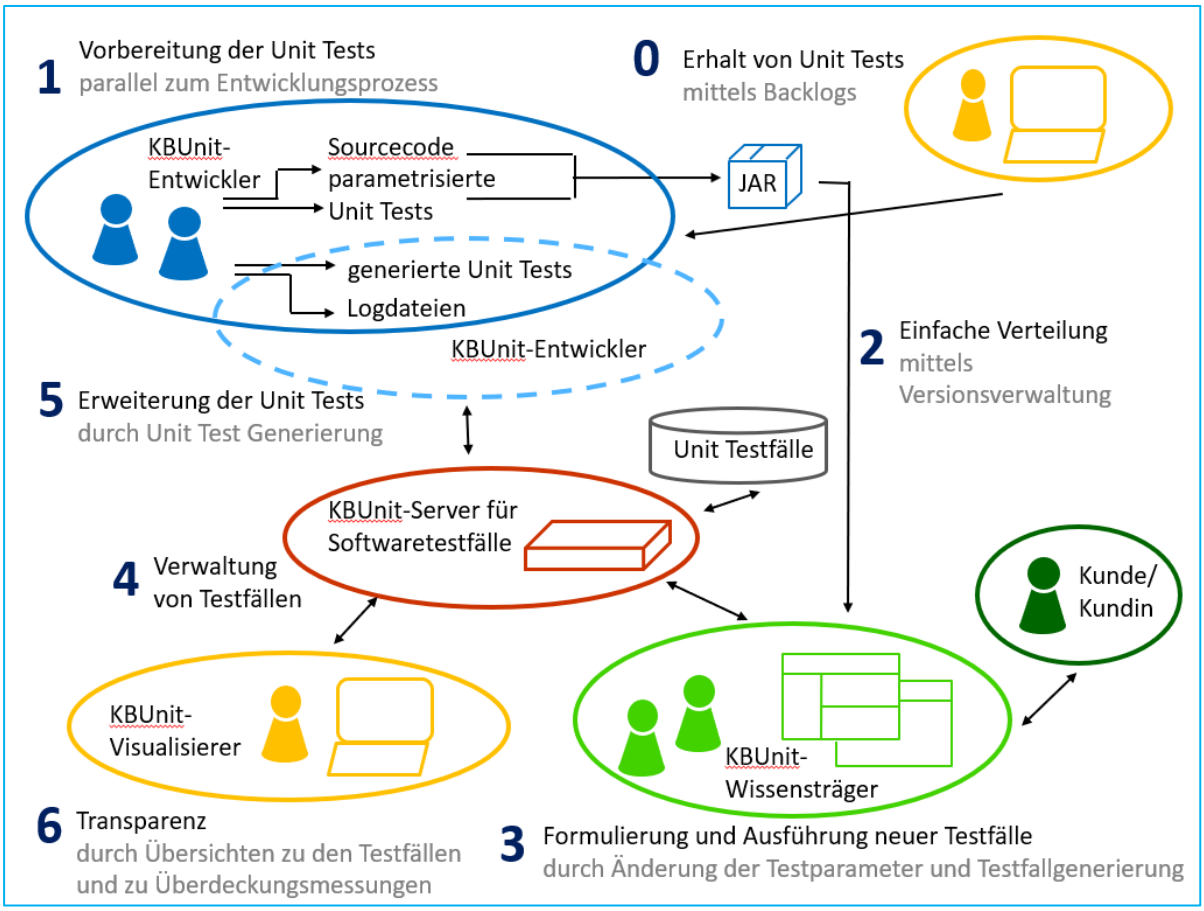

Abbildung 1: Prozess zur Durchführung von Unit Tests

Schritt 0 – Erhalt von Unit Tests:
Es besteht die Möglichkeit, sich aus den Backlogs agiler Projektmanagementtools entweder JUnit Tests oder parametrisierte JUnit Tests und jeweils zu ergänzenden Sourcecode erzeugen zu lassen.

Schritt 1 - Vorbereitung der Unit Tests:
Die Entwickler implementieren eine Funktionalität und zu dieser einen JUnit Test, welcher einen Testfall überprüft. Soll dieser von Wissensträgern um Testfälle ergänzt werden, transferieren die Entwickler den JUnit Test in einen parametrisierten JUnit Test. Er enthält diejenigen Parameter, die von Wissensträgern variiert werden sollen.

Schritt 2 - Einfache Verteilung der Unit Tests:
Der Sourcecode und die parametrisierten JUnit Tests werden den Wissensträgern zugänglich gemacht.

Schritt 3 - Formulierung und Ausführung neuer Tests:
Die Wissensträger greifen auf die von den Entwicklern entworfenen parametrisierten JUnit Tests zu und erstellen in Absprache mit dem Kunden weitere Testfälle, die aus ihrer Sicht geprüft werden müssen, indem sie die Werte der Parameter variieren und Testfälle generieren lassen. Sie können sämtliche Testfälle auch abspielen.

Schritt 4 - Verwaltung von Testfällen:
Die Wissensträger speichern die neuen Testfälle inklusive deren Testergebnis in einer Datenbank. Sie können diese auch verwalten.

Schritt 5 - Erweiterung der Unit Tests:
Die Entwickler haben die Möglichkeit, auf die neuen Testfälle zuzugreifen und diese zu verwalten. Sie können diese in ihrer Entwicklungsumgebung abspielen und entsprechende Logdateien erstellen lassen. Sie können ebenfalls sich zu den neuen Testfällen Unit Tests generieren lassen.

Schritt 6 - Transparenz:
Es besteht zusätzlich die Möglichkeit, sich den Stand zu den Softwaretests anzeigen zu lassen. Weiterhin besteht die Möglichkeit, sich den Stand der Überdeckung des Quellcodes anzeigen zu lassen, welchen man durch die vorhandenen Testfälle erhält. Ein Feedback zu Softwaretests und deren Überdeckung kann an die Entwickler gemeldet werden.

2 Vorstellung des Beispiels

Bevor Modultests in KBUnit modifiziert und ausgeführt werden können, bedarf es einiger Vorbereitungen, die anhand eines einfachen Beispiels erläutert werden. Die Ausführungen dieses Kapitels sind zum großen Teil übernommen aus: Sprengholz, Philipp: *Lean Software Development – Kundenzentrierte Softwareentwicklung durch Anwendung schlanker Prinzipien,* 1. Auflage: AVM, München 2011, ISBN 978-3-86924-052-7.

Für ein mittelständisches Unternehmen soll eine Software entwickelt werden, mit der alle Finanzierungsaktivitäten überwacht und gesteuert werden können. Die Anwendung soll sowohl Buchhaltung, Kostenrechnung als auch Controlling unterstützen und aus einer Vielzahl von Modulen bestehen. Der Wissensträger hat in Zusammenarbeit mit den Entwicklern einige User Stories erarbeitet, welche die Anforderungen an die zu entwickelnde Software beschreiben. Eine der User Stories enthält folgenden Text:

> *Als Mitarbeiter der Investitionsplanung möchte ich die Gesamtbelastung zu einem Tilgungsdarlehen berechnen, um die Konditionen verschiedener Kreditinstitute zu vergleichen.*

Es soll also ein Softwaremodul entwickelt werden, mit dem die durch Aufnahme eines Tilgungsdarlehens entstehende Gesamtschuld berechnet werden kann. An dieser Stelle sei kurz das Wesen des Tilgungsdarlehens erläutert. Es handelt sich um eine Form des Kredits, bei der die Annuitäten über die Laufzeit sinken. Während der Tilgungsbetrag in jeder Periode gleich hoch ist, nehmen die Zinsen kontinuierlich ab, da sie aus der verbleibenden Restschuld berechnet werden. Ein Beispiel zu dem Modul sei das folgende:

> *Bei einem Tilgungsdarlehen in Höhe von 100.000,00 Euro, das über zehn Perioden zu 2 % verzinst wird, ergibt sich eine Gesamtschuld von 111.000,00 Euro. Die Tilgung beträgt in jeder Periode 10.000,00 Euro, die Zinsen werden auf die jeweils verbliebene Restschuld vor Tilgung berechnet.*

Es ist ein Modul zu entwickeln, welches vom Wissensträger die Eingabe eines Darlehensbetrages in Cent, einer Laufzeit, welche der Anzahl zurückzuzahlender Raten entspricht, sowie eines Zinssatzes in Prozent erwartet. Aus diesen Informationen soll das Programm die Gesamtschuld berechnen.

3 Vorbereitung von Modultests für die Verwendung in KBUnit

Die in Kapitel 3 der Anwendungsdokumentation vorgestellten Aktivitäten entsprechen den Schritten 1 und 2 der Abbildung 1 und werden durch die Entwickler durchgeführt.

Voraussetzungen

Es wird davon ausgegangen, dass eine IDE für Java Enterprise – Projekte und ein Java JDK zur Verfügung stehen. Vorgestellt wird das vorliegende Beispiel mit der IDE Eclipse. JUnit ist bereits in Eclipse integriert. Bitte beachten Sie die Lizenzbedingungen für das verwendete JDK und für die JUnit-Bibliotheken! KBUnit unterstützt alle Versionen ab dem JDK 1.8.x. Vorgestellt und erläutert wird das vorliegende Beispiel für JUnit4.x und JUnit5.x.

Erstellen Sie in einem workspace von KBUnit – Entwickler ein Java-Projekt mit dem folgenden Aufbau.

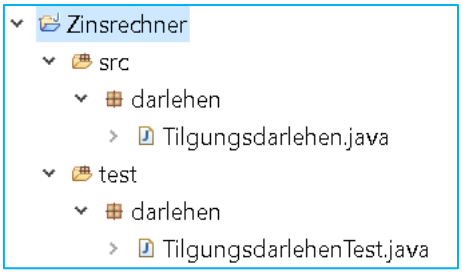

Abbildung 2: Struktur des Beispielprojekts

Im Kontextmenü des Projekts wählen Sie *Properties* aus. In dem dann folgenden Fenster können Sie unter *Java Build Path,* Tabreiter *Libraries* das verwendete jdk anpassen. Klicken Sie dazu auf *Modulepath*, *Add Library…, JRE System Library* und stellen in dem dann folgenden Fenster das jdk Ihrer Wahl ein. Unter *Java Compiler* können Sie auch das *Compiler compliance level* anpassen. Falls Sie das *Compiler compliance level* 1.8 ausgewählt haben, wird unter Java Build Path, Tabreiter *Libraries* nicht zwischen *Modulepath* und *Classpath* unterschieden.

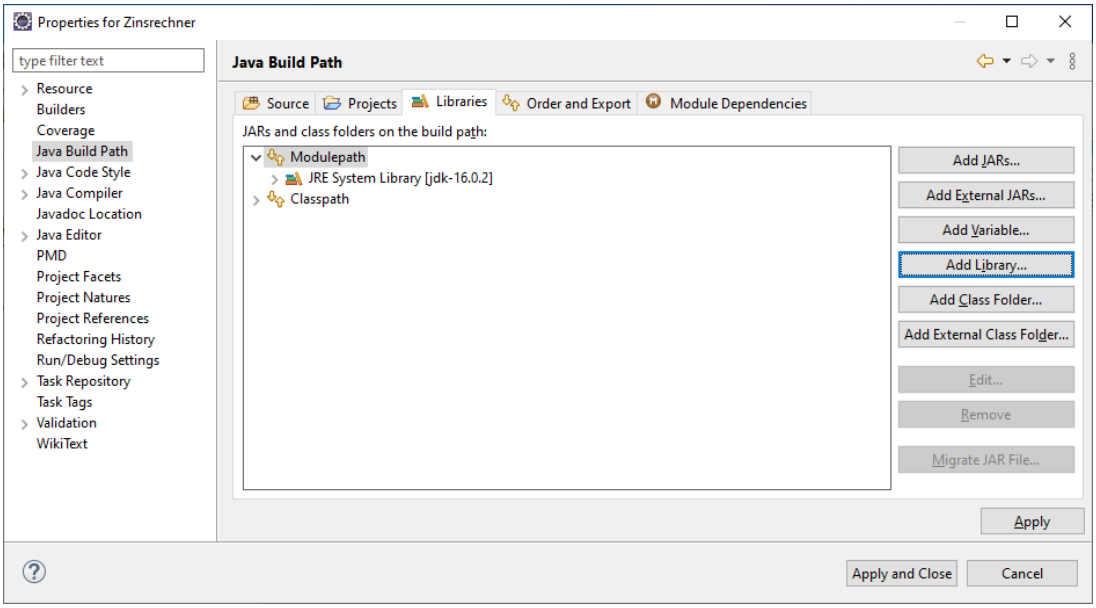

Abbildung 3: jdk festlegen

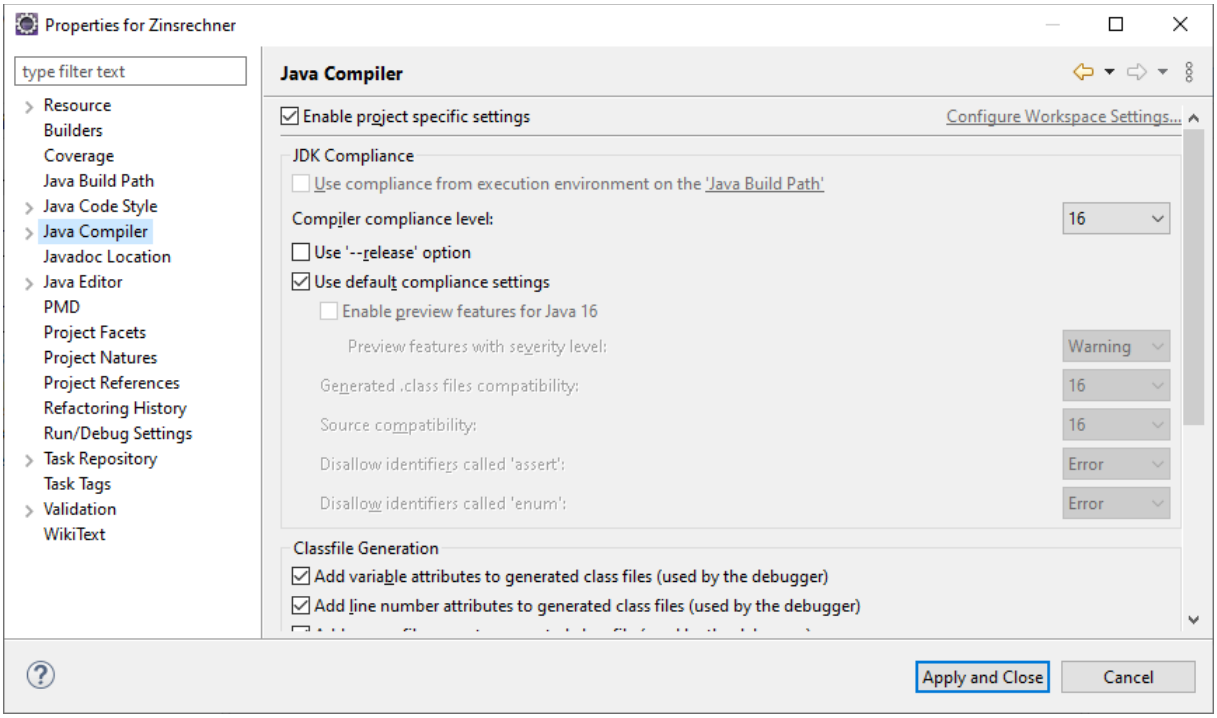

Abbildung 4: Compiler compliance level festlegen

4 Administration von KBUnit – Server

KBUnit – Wissensträger und KBUnit – Entwickler greifen auf den RESTful Webservice KBUnit - Server inklusive Datenbank zu. Der Webservice und die Datenbank müssen daher von Entwicklern oder Administratoren im Vorfeld eingerichtet werden.

4.1 RESTful Webservice

Voraussetzungen

Für den Service wird im Beispiel der Apache Tomcat Application Server verwendet.

Erstellen des RESTful Webservices

Starten Sie Eclipse und erstellen Sie ein neues Projekt mittels *File/New/Other* und dann *Web / Dynamic Web Project.* Beim Klick auf *Next >* erhalten Sie das folgende Fenster.

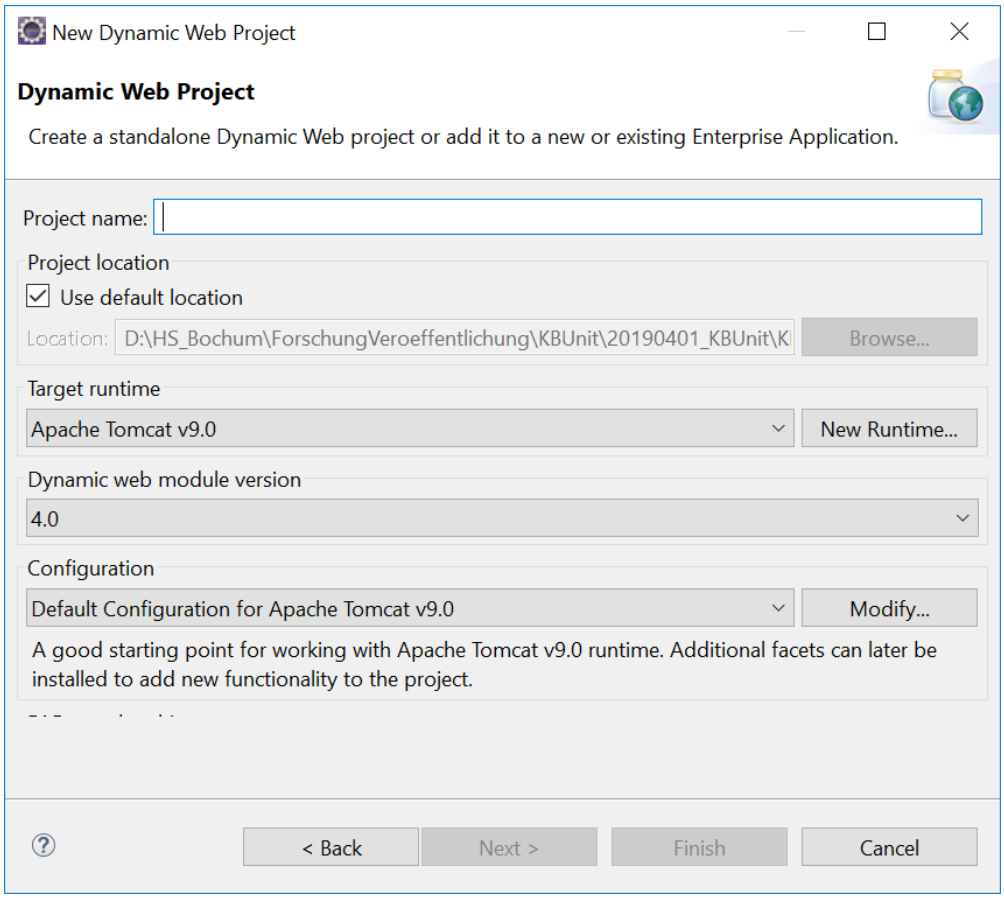

Abbildung 5: Anlage eines RESTful Webservices: 1

Geben Sie als Projektnamen *KBUnitServer* ein und das *Target Runtime* (den Web Server / Application Server). Falls das Feld noch nicht gefüllt ist, klicken Sie den Button *New Runtime* an und wählen in dem folgenden Fenster den Server aus.

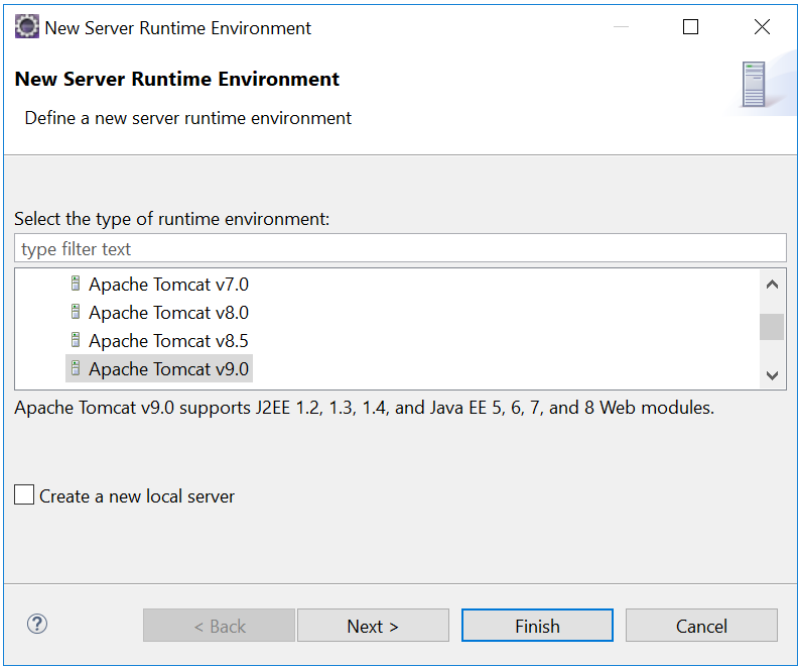

Abbildung 6: Anlage eines RESTful Webservices: 2

Bestätigen Sie Ihre Auswahl mit *Next >*. Sie erhalten das folgende Fenster.

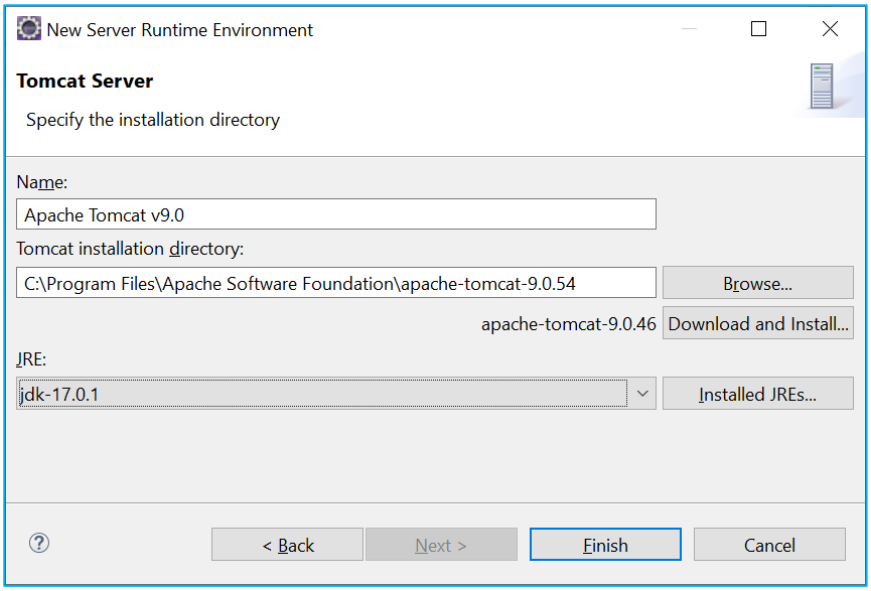

Abbildung 7: Anlage eines RESTful Webservices: 3

Hier geben Sie das Verzeichnis an, in welchem Tomcat installiert ist, und wählen als JRE ein jdk aus. (Es muss ein jdk sein, nicht ein jre.) Dann klicken Sie auf *Finish.* Sie kommen zum Ausgangsfenster, wo Sie mehrfach auf den Button *Next >* klicken. In dem folgenden Fenster wählen Sie *Generate web.xml deployment descriptor* aus.

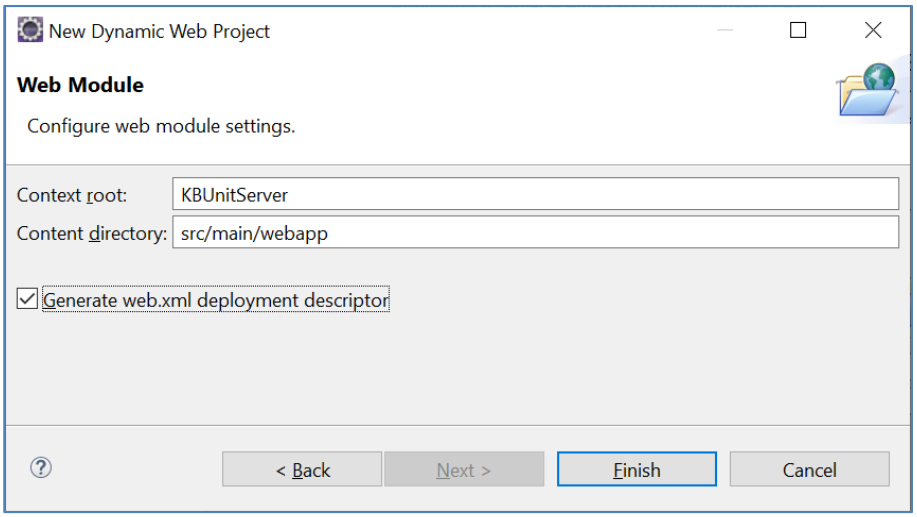

Abbildung 8: Anlage eines RESTful Webservices: 4

Das Projekt wird angelegt.

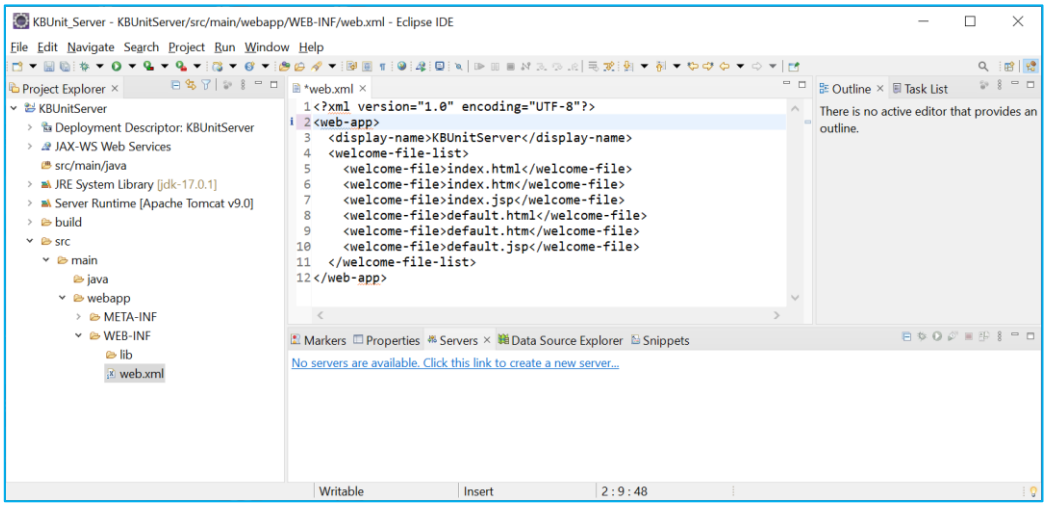

Abbildung 9: Anlage eines RESTful Webservices: 5

Es müssen aber noch weitere Einstellung erfolgen.
Sie ändern die Zeile 2 der Datei web.xml zu <*web-app*> ab. Weiterhin wählen Sie im Kontextmenü des Projekts *Properties* aus. Dann wählen Sie in dem Fenster, welches sich öffnet, *JavaCompiler* aus und passen das *Compiler compliance level* an, falls dieses nicht zum *jdk* passend ist, siehe Kapitel 3.

Um das Projekt zu einem Maven-Projekt zu konvertieren, wählen Sie im Kontextmenü des Projekts *Configure / Convert to Maven Project* aus. Sie übernehmen in dem folgenden Fenster sämtliche Voreinstellungen und bestätigen diese mit einem Klick auf *Finish.* Es öffnet sich die Datei *pom.xml.*

```
21   </build>
22   <dependencies>
23     <dependency>
24       <groupId>org.glassfish.jersey.containers</groupId>
25       <artifactId>jersey-container-servlet</artifactId>
26       <version>2.29.1</version>
27     </dependency>
28     <dependency>
29       <groupId>org.glassfish.jersey.inject</groupId>
30       <artifactId>jersey-hk2</artifactId>
31       <version>2.29.1</version>
32      </dependency>
33    </dependencies>
34 </project>
```

Abbildung 10: Ergänzen der pom.xml

Um jersey-Bibliotheken und weitere zu erhalten, ergänzen Sie diese Datei um die *dependencies,* die Sie der Abbildung 10 entnehmen können. Bei Speicherung des Projekts stehen die Bibliotheken dem Projekt zur Verfügung.

Das neue Projekt muss dem Server bekannt gemacht werden. Dazu klicken Sie in Eclipse auf den Tabreiter *Servers.* Falls der Tomcat-Server nicht automatisch gefunden wird, klicken Sie dort auf die dort angegebene Zeile. Dann erhalten Sie ein Fenster, in welchem Sie den Server aussuchen.
Im Kontextmenü des Servers wählen Sie *Add and Remove…* aus. Sie erhalten das folgende Fenster, in welchem Sie *KBUnitServer* mittels *Add >* von der Spalte *Availble* zur Spalte *Configured* befördern.

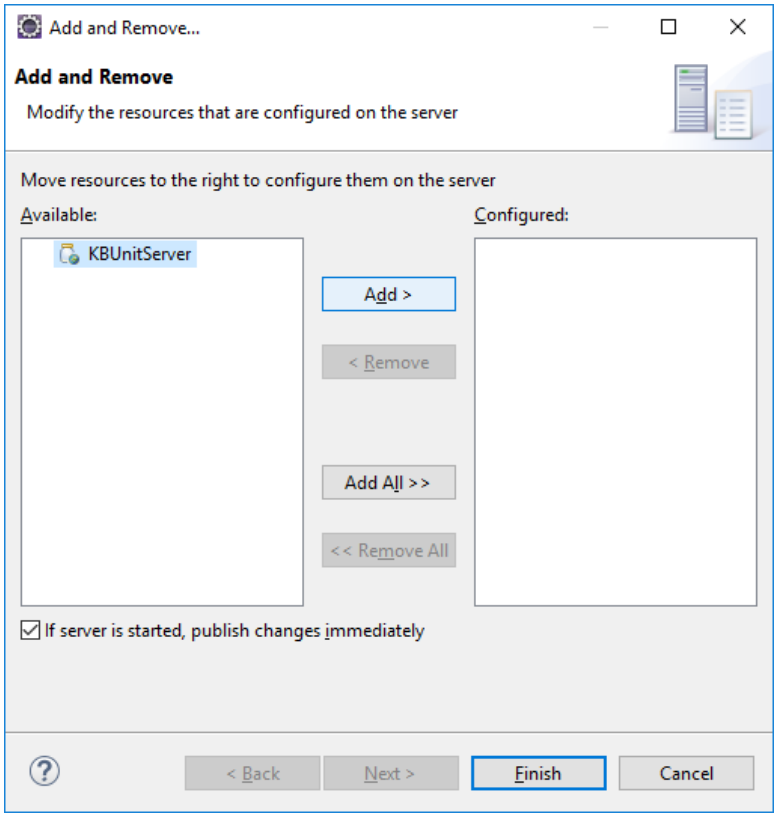

Abbildung 11: Bekanntmachen des Webservices 1

Um den Server zu starten, markieren Sie diesen und betätigen das Start-Icon.

Bisher steht in der Datei *web.xml* im Verzeichnis *WebContent/WEB-INF* nur die Startseite für den Web-Server. Jetzt müssen Sie in dieser Datei die Web-Services bekannt machen. Hier wird demnach die Weiche gestellt, ob eine HTML-Seite des Web-Servers oder ein REST-Service des Application-Servers angesteuert werden, siehe Listing 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <display-name>KBUnitServer</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>kbUnitRest</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Listing 1: Bekanntmachen des Webservices 2

Die entscheidende Zeile zum Ansteuern von Web-Services ist
<url-pattern>/rest/</url-pattern>*
Alle Anfragen der Form: *localhost:8080/Projektname/rest/ …* werden an den Application-Server weitergeleitet, alle weiteren Anfragen an den Web-Server.

Legen Sie jetzt die Java-Klassen von KBUnit - Server in das source folder *src/main/java* unter Beachtung der package-Struktur.

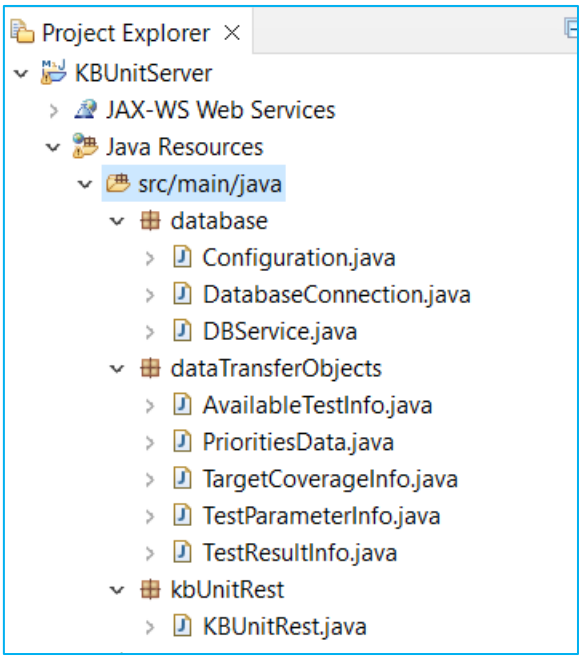

Abbildung 12: source folder src von KBUnit - Server

Erstellen Sie im Verzeichnis *src/main/webapp* die xml-Datei *KBUnitServerConfig.xml,* siehe Listing 2.

```
<?xml version = "1.0" encoding="UTF-8"?>
<root>
    <database>
        <ipAdresse>localhost</ipAdresse>
        <port>3306</port>
        <user>xxx</user>
        <password>yyy</password>
        <driver>com.mysql.cj.jdbc.Driver</driver>
    </database>
</root>
```

Listing 2: KBUnitServerConfig.xml

Hier legen Sie die Daten für den Anschluss an die Datenbank, siehe unten fest. Abschließend wählen Sie im Kontextmenü des Projekts *Properties* und dann *Server* aus und wählen Sie den Server aus, mit welchem das Projekt starten soll, siehe Abbildung 13. Bestätigen Sie Ihre Auswahl mit *Apply and Close.*

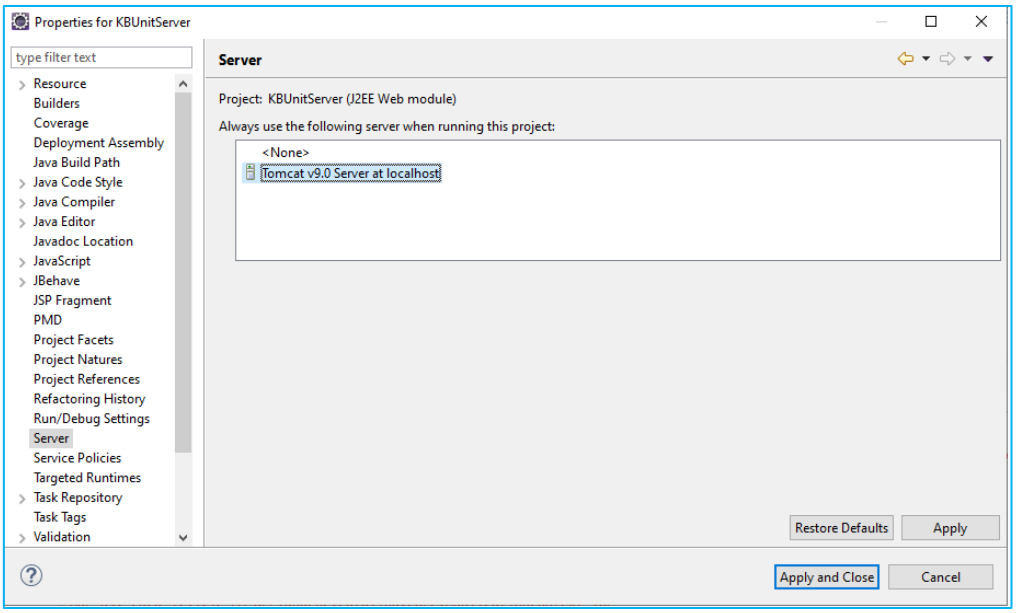

Abbildung 13: Auswahl des Servers beim Start des Projekts

Es fehlen weitere Bibliotheken. Legen Sie die jackson- und jdom-Bibliotheken der Abbildung 14 in das Verzeichnis *src/main/webapp/WEB_INF/lib*.

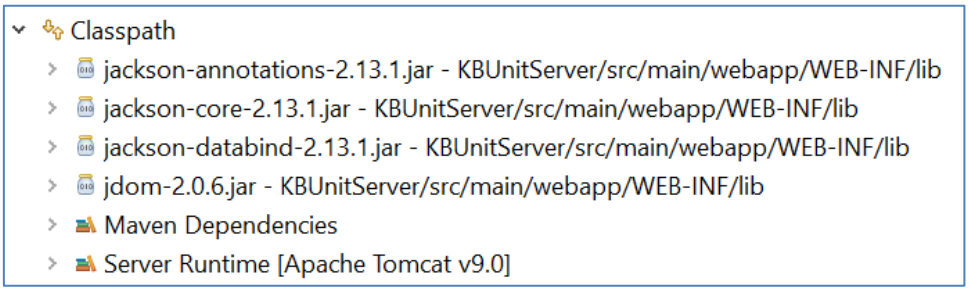

Abbildung 14: Bibliotheken von KBUnit – Server

Legen Sie weiterhin die Bibliothek *mysql-connector-java-8.0.28.jar* in dieses Verzeichnis. Binden Sie diese Bibliotheken an das Projekt an, indem Sie im Kontextmenü *Properties* aussuchen und dann *Java Build Path*. Sie markieren dann *Classpath* und wählen *Add JARS…* aus, um in den genannten lib-Ordner zu navigieren, die Bibliotheken zu markieren und Ihre Auswahl mit *OK* zu bestätigen. Im Ausgangsfenster bestätigen Sie Ihre Auswahl Sie mittels *Apply and Close.*
Wenn Sie jetzt den Server starten, steht der Service zur Verfügung. (Bitte beachten Sie die Lizenzbedingungen für das verwendete JDK und für die verwendeten Bibliotheken!).

Verteilen des RESTful Webservices

Falls Sie den Service deployen möchten, wählen Sie dazu im Kontextmenü des Projekts *Export / WAR file* aus. In dem folgenden Fenster geben Sie das Ziel der WAR-

Datei ein und bestätigen Sie mit *Finish*. Die war-Datei muss dann auf denjenigen Server gelegt werden, der den Service anbieten soll. Sie gehen dazu in das Tomcat-Verzeichnis:
x*xx:\...\apache-tomcat-y.y.y\webapps*
und kopieren das war-File dort hinein. Jetzt können Sie Eclipse schließen. Um ohne Eclipse manuell den Tomcat-Server zu starten, gehen Sie in das Tomcat-Verzeichnis, hier *D:\apache-tomcat-y.y.y\bin* und wählen dort *startup.bat* aus. Den Server können Sie mittels *shutdown.bat* stoppen.

4.2 Datenbank

An der in Listing 2 vorgegebenen Adresse muss eine MySql-Datenbank mit dem Namen *kbunittest* zur Verfügung stehen. (In diesem Beispiel wurde diese mit Hilfe des WampServers (*https://sourceforge.net/projects/wampserver*) erstellt.) Sie benötigen weiterhin die Datei *kbunittest.sql*.
Erstellen Sie demnach eine Datenbank mit dem Namen *kbunittest*. Erstellen Sie dann die Tabellen in der Datenbank, indem Sie die Datei *kbunittest.sql* importieren. Alternativ können Sie die Tabellen, wie in *kbunittest.sql* vorgegeben, auch manuell in der Datenbank erstellen.

5 Administration von KBUnit - Clients

5.1 KBUnit – Wissensträger

Die in Kapitel 4 der Anwendungsdokumentation vorgestellten Aktivitäten entsprechen den Schritten 3 und 4 der Abbildung 1 und werden von den Wissensträgern durchgeführt.

Voraussetzungen

Für KBUnit - Wissensträger benötigen Sie Sie die folgenden Bibliotheken.

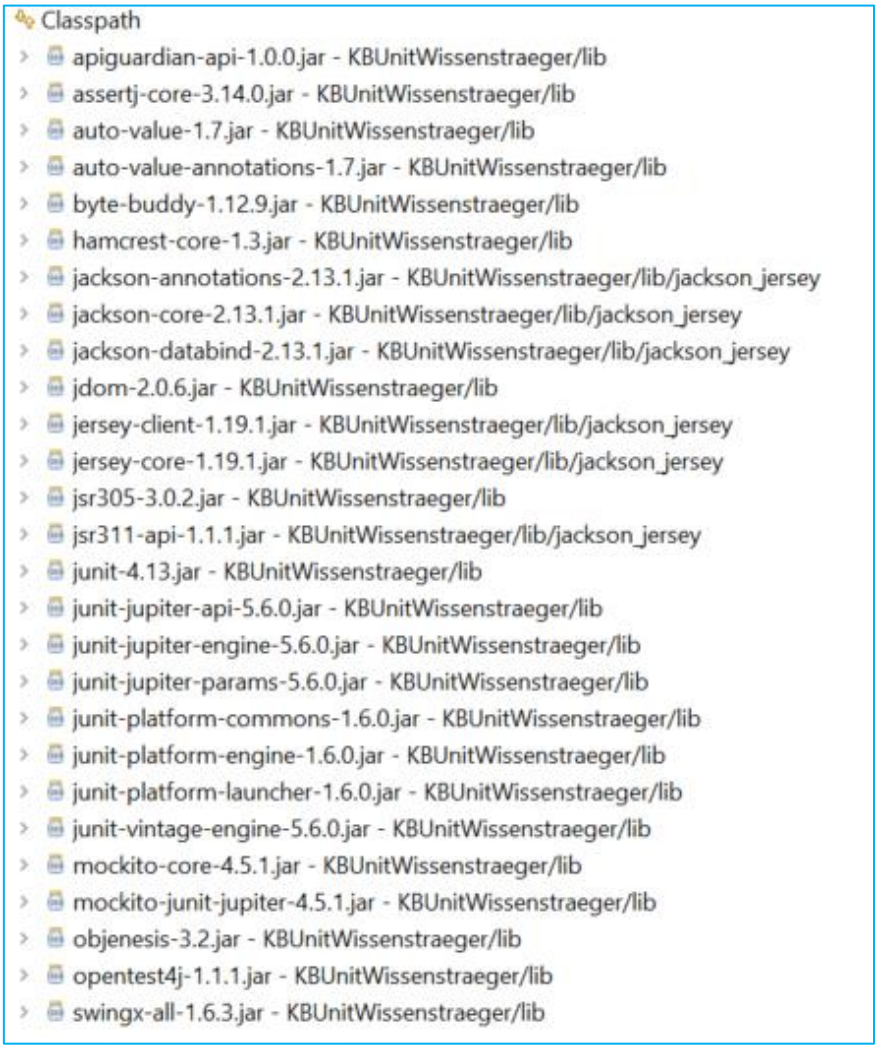

Abbildung 15: Bibliotheken von KBUnit - Wissenstraeger

Erstellen Sie ein workspace für KBUnit Wissensträger und in diesem ein Java Projekt und fügen Sie die Bibliotheken, wie im Kapitel 4 für KBUnit – Server beschrieben, dem Projekt hinzu. Der Quellcode von KBUnit – Wissensträger wird gemeinsam mit der Konfigurationsdatei *config.xml* und verwendeten Icons ausgeliefert.

Kopieren Sie die Elemente in Ihr Projekt unter Berücksichtigung der package-Struktur, siehe Abbildung 16.

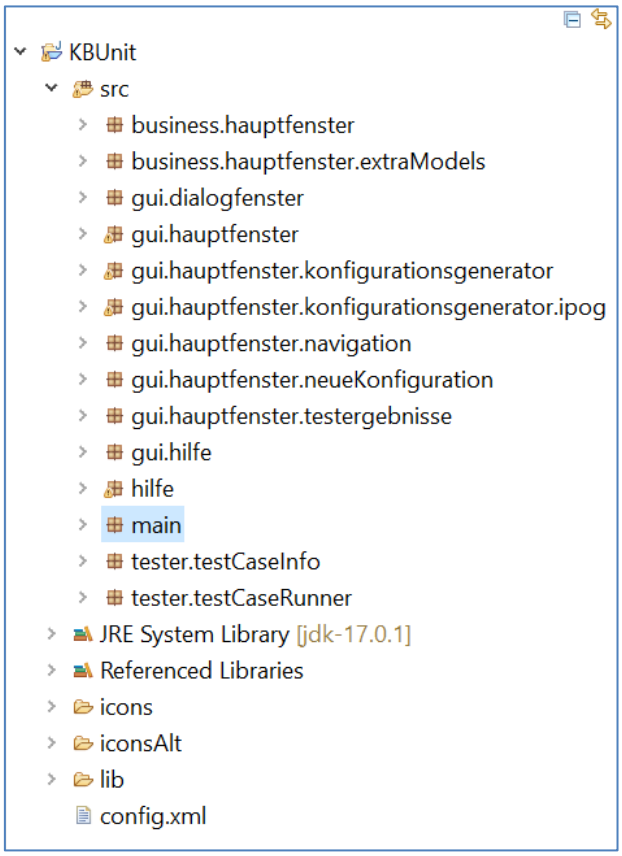

Abbildung 16: Struktur von KBUnit - Wissenstraeger

In der Abbildung 16 gibt es das Verzeichnis *iconsAlt.* Diese benötigen Sie nicht. Passen Sie die Konfigurationsparameter in der Datei *config.xml* an die vorhandene Situation an. Der Aufbau der Datei ist in Listing 3 dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <rest>
        <path>http://localhost:8080/KBUnitServer/rest/kbUnit</path>
    </rest>
    <documentation>
        <path>Anwendungsdokumentation_KBUnit.pdf</path>
    </documentation >
</root>
```

Listing 3: Inhalt der Datei config.xml

Die Datei enthält Informationen über den RESTful Webservice von KBUnit - Server, siehe Schritt 4 der Abbildung 1, und Kapitel 4 dieser Administrationsdokumentation. Der RESTful Webservice muss an dem in der Zeile 4 der Datei *config.xml* genannten

Ort zur Verfügung stehen, siehe Kapitel 4 dieser Administrationsdokumentation. Der RESTful Webservice wird benötigt, um die vom Wissensträger ausgeführten Tests zu speichern und später erneut ablaufen lassen zu können. Das Programm unterstützt standardmäßig eine MySQL-Datenbank, siehe Kapitel 4 dieser Administrations-dokumentation.

Neben der Anbindung an den Webservice wird in der Datei *config.xml* zusätzlich der Ort genannt, an dem diese Anwendungsdokumentation im pdf-Format zu finden ist. Falls die Einstellung übernommen wird, muss die Anwendungsdokumentation in demselben Verzeichnis liegen wie *config.xml*. Liegt die Anwendungsdokumentation in einem Unterverzeichnis *doc,* so muss als Pfad
doc\\Anwendungsdokumentation_KBUnit_Wissenstraeger.pdf
angegeben werden. Sie können auch den absoluten Pfad angeben.

Die icons legen Sie in das Unterverzeichnis *icons,* siehe Abbildung 16.

Wenn Sie jetzt die Anwendung starten, erhalten Sie vielleicht noch einen Fehler. (Das ist von der java-Version abhängig.) Im Fehlerfall legen Sie noch die folgende Konfiguration fest. Wählen Sie *Run / Run Configurations…* aus. In dem dann folgenden Fenster nehmen Sie den folgenden Eintrag unter *VM arguments* vor. Den Eintrag schreiben Sie im Vorfeld in eine Textdatei und arbeiten Sie dann mit der Zwischenablage.

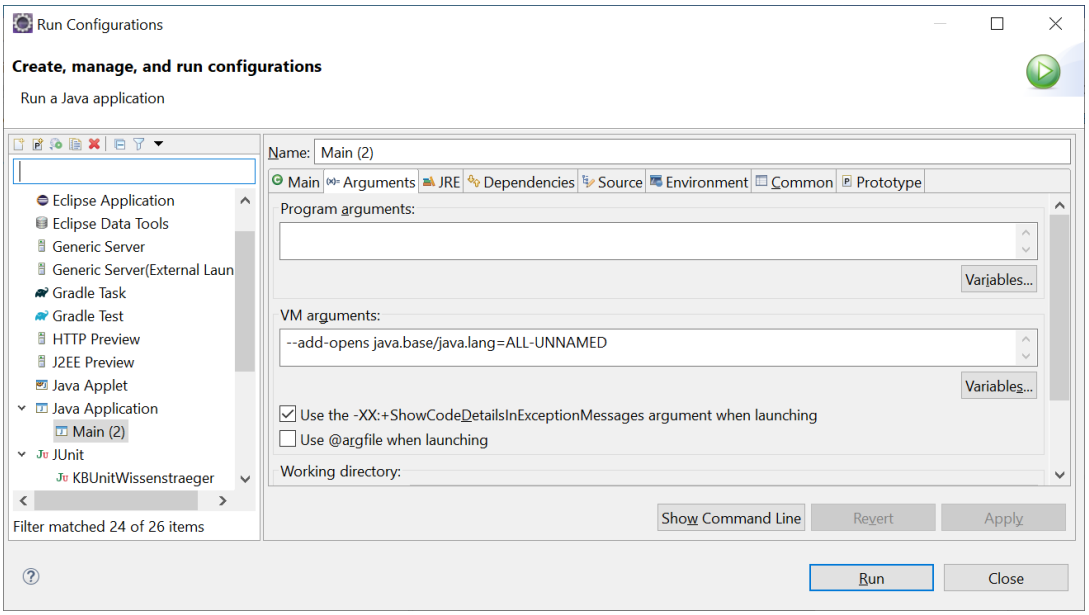

Abbildung 17: VM-Argumente von KBUnit - Wissenstraeger

Der Start von KBUnit – Wissensträger wird in der Anwendungsdokumentation in Kapitel 4 näher beschrieben.

5.2 KBUnit – Entwickler

Die in Kapitel 5 der Anwendungsdokumentation vorgestellten Aktivitäten entsprechen den Schritten 0, 4 und 5 der Abbildung 1 und werden von den Entwicklern durchgeführt.

Voraussetzungen

Für KBUnit - Entwickler benötigen Sie den Quellcode von KBUnit – Entwickler und die in Abbildung 18 aufgelisteten Bibliotheken. Legen Sie die Bibliotheken in ein Verzeichnis *lib*. Fügen Sie die Bibliotheken, wie im Kapitel 4 für KBUnit – Server beschrieben, dem bereits aus dem 3. Kapitel vorhandenen Projekt hinzu. (Bitte beachten Sie die Lizenzbedingungen für die verwendeten Bibliotheken!).

Abbildung 18: Bibliotheken von KBUnit – Entwickler

In Abhängigkeit der verwendeten Java-Version müssen JavaFX-Bibliotheken angeschlossen werden. Öffnen Sie dazu das Kontextmenü des Projekts, wählen Sie *Properties* aus, nehmen Sie die Auswahl *Java Build Path,* Tabreiter *Libraries* vor, markieren Sie *Modulpath* und wählen Sie *Add Library…* und dann *User Library, User Libraries …, New* aus. Geben Sie einen Namen, hier *JavaFX* ein und bestätigen Sie

Ihre Eingabe. Sie erhalten einen Library-Ordner, in welchem Sie mittels *Add External JARs…* die JavaFX Bibliotheken der Abbildung 4 anschließen.

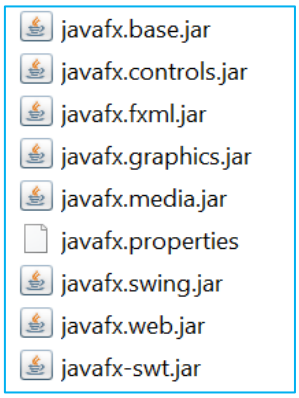

Abbildung 19: JavaFX – Bibliotheken

(Bitte beachten Sie die Lizenzbedingungen für die verwendeten Bibliotheken!).
Passen Sie dann die *Run Configurations* folgendermaßen an. Wählen Sie *Run / Run Configurations…* aus. In dem dann folgenden Fenster nehmen Sie den folgenden Eintrag unter *VM arguments* vor. Den Eintrag schreiben Sie im Vorfeld in eine Textdatei und arbeiten Sie dann mit der Zwischenablage.

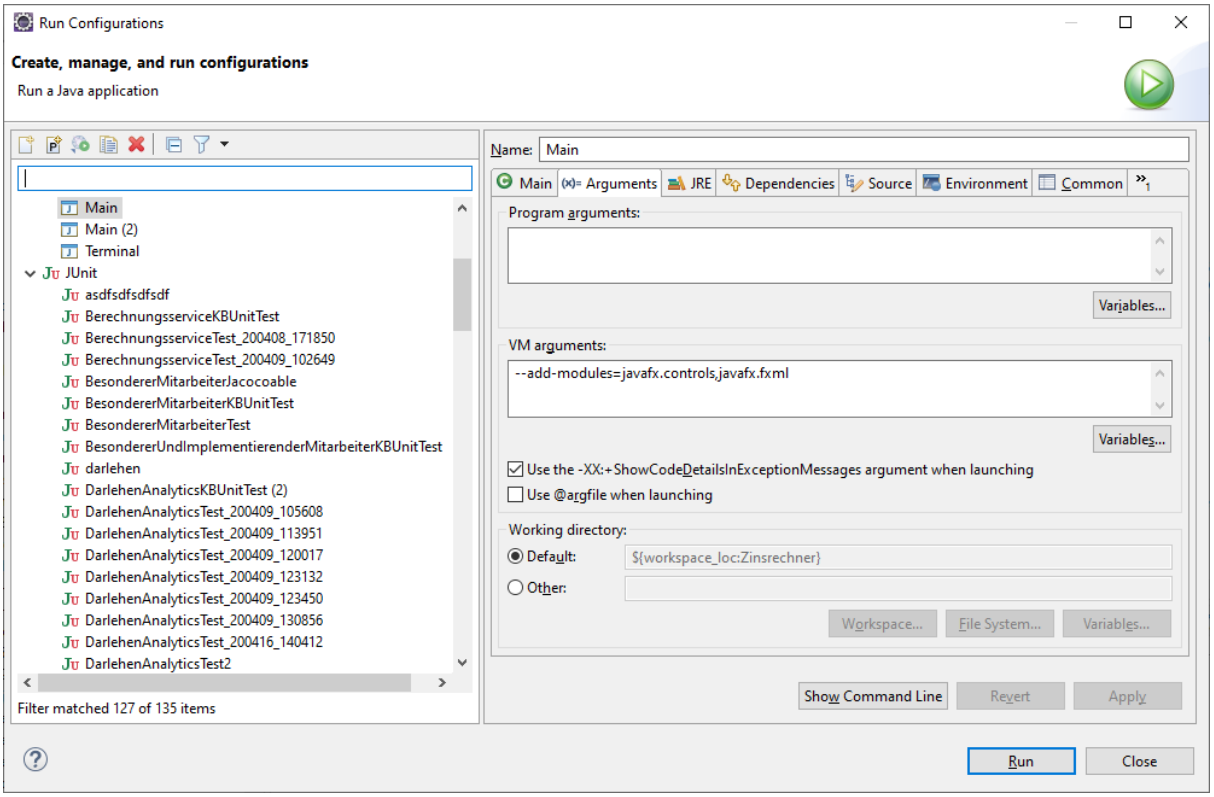

Abbildung 20: VM-Argumente von KBUnit - Entwickler

Erstellen Sie ein source folder *testForKBUnit*. Erstellen Sie weiterhin ein source folder *testKBUnit*. Kopieren Sie den Quellcode von KBUnit - Entwickler in Ihr Projekt in das neu erstellte source folder *testKBUnit*, siehe Abbildung 21.

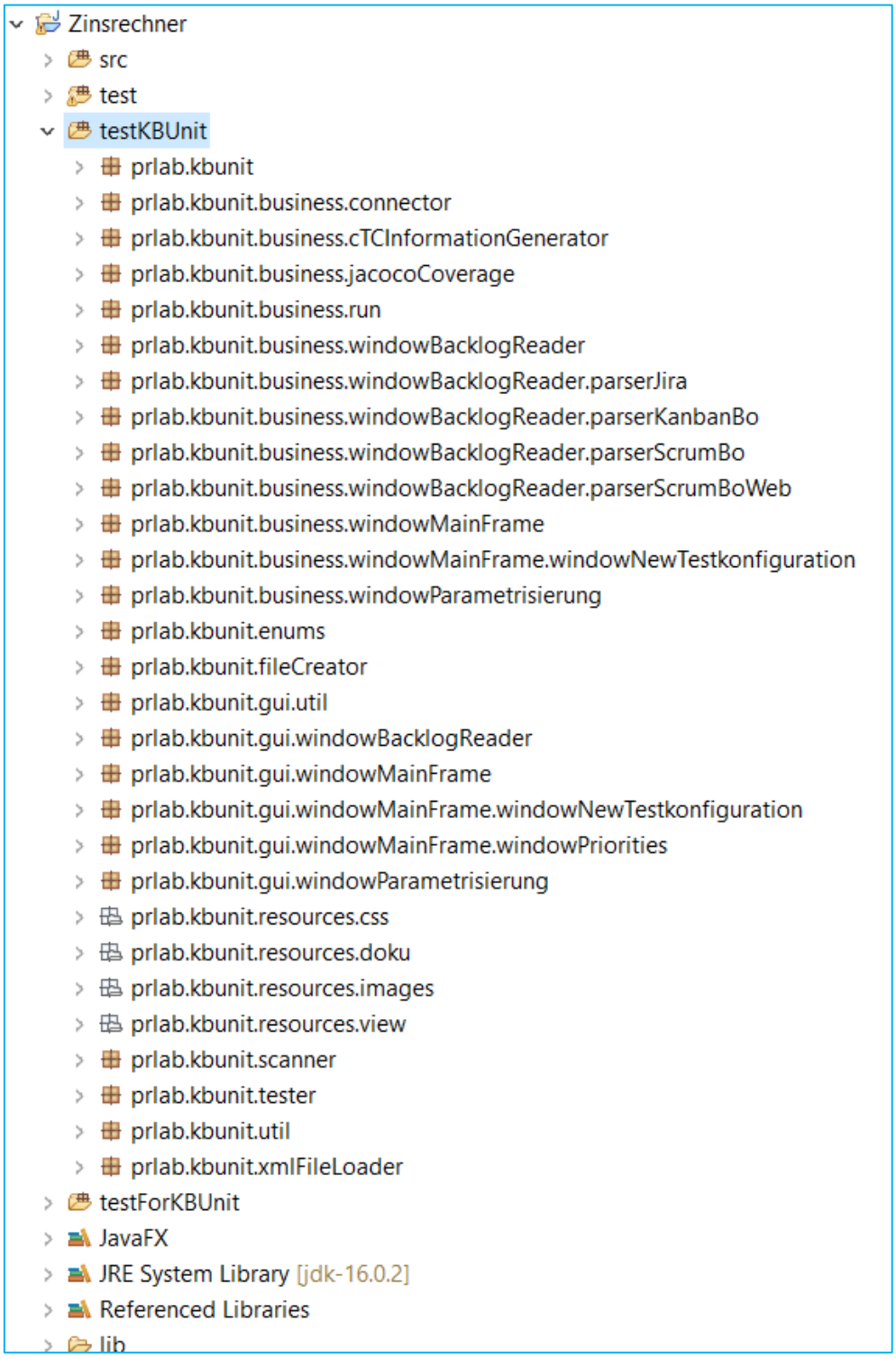

Abbildung 21: Struktur von KBUnit – Entwickler

Passen Sie die Konfigurationsparameter in der Datei *PreferencesKBUnitServer.xml* im source folder *testKBUnit*, package *prlab.kbunit* an die vorhandene Situation an. Der Aufbau der Datei ist in Listing 4 dargestellt.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <root>
03      <rest>
04          <host>http://localhost:8080/KBUnitServer/rest/kbUnit</host>
05      </rest>
12      <update>
13          <!-- Copying test cases performed after [yyyy-MM-dd  HH:mm:ss] :-->
14          <date>2022-09-24</date>
15          <time>17:03:35</time>
16      </update>
17  </root>
```

Listing 4: Inhalt der Datei PreferencesKBUnitServer.xml

Die Datei enthält Informationen über den RESTful Webservice von KBUnit - Server, siehe Schritt 4 der Abbildung 1 und Kapitel 4 dieser Administrationsdokumentation.
Die weitere Konfiguration von KBUnit – Entwickler ist eng gekoppelt an die Erstellung des zu testenden Projekts. Sie wird in der Anwendungsdokumentation in Kapitel 5 näher beschrieben.

Stellen Sie in Eclipse die Eigenschaft *Auto Refresh* ein. Gehen Sie dazu auf den Tabreiter *Window,* dann *Preferences.* Es öffnet sich ein Fenster, in welchem Sie in der Navigation links *General* und dann *Workspace* anklicken. Aktivieren Sie dann die Schaltflächen *Refresh using native hooks or polling* und *Refresh on access.* Sie bestätigen diese Auswahl mittels Klick auf *Apply and Close.*

5.3 KBUnit – Visualisierer

Die in Kapitel 6 der Anwendungsdokumentation vorgestellten Aktivitäten entsprechen dem Schritt 6 der Abbildung 1 und werden von Personen mit fachlicher Expertise und Personen mit Entscheidungsbefugnissen über die Produktivschaltung von Software durchgeführt.

Voraussetzungen

KBUnit – Visualisierer liegt als Maven-Projekt vor. Legen Sie einen workspace für KBUnit – Visualisierer an. Das vorliegende Projekt importieren Sie folgendermaßen in Ihren workspace. Sie wählen File/Import… aus. In dem dann folgenden Fenster nehmen Sie die folgende Auswahl vor, siehe Abbildung 22, und klicken auf *Next >.*

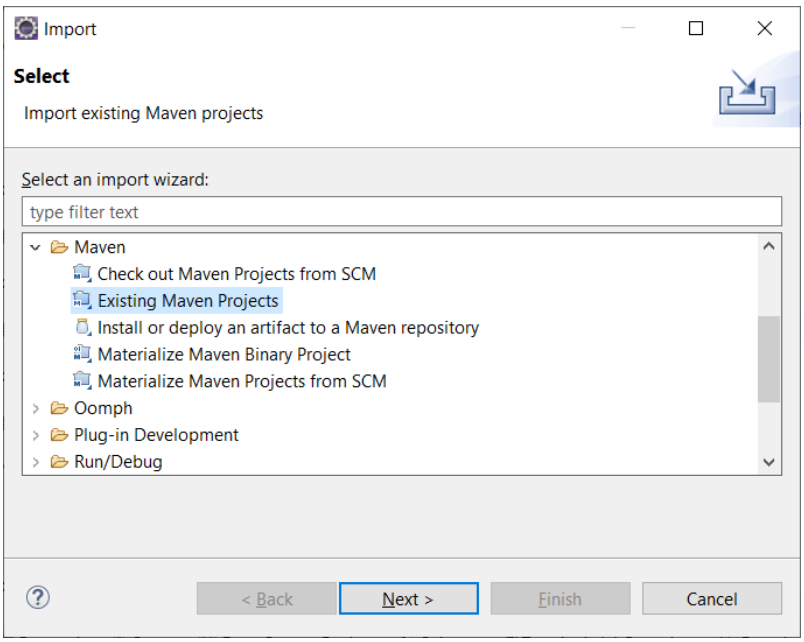

Abbildung 22: Import von KBUnit - Visualisierer

Dann wählen Sie die Root Directory des Projekts aus und bestätigen Sie die Auswahl mit *Finish*. Die erforderlichen Bibiliotheken werden mittels der pom.xml – Datei des Projekts importiert. Für die Verbindungsdaten zu KBUnit – Server gibt es die Datei *config.xml*. Diese sieht genauso aus wie die Datei *config.xml* von KBUnit – Wissensträger, siehe Listing 3, nur fehlt der Tab *documentation*.

Impressum

Prof. Dr. Ursula Oesing

ursula.oesing@hs-bochum.de

Fachbereich Elektrotechnik und Informatik
Hochschule Bochum
Am Hochschulcampus 1
44801 Bochum