Formulierung und Durchführung von Modultests mit KBUnit

Administrationsdokumentation



Inhalt

1	Überblick	3
2	Vorstellung des Beispiels	6
3	Vorbereitung von Modultests für die Verwendung in KBUnit	7
4	Administration von KBUnit – Server	9
	4.1 RESTful Webservice	9
	4.2 Datenbank	16
5	Administration von KBUnit - Clients	17
	5.1 KBUnit - Wissensträger	17
	5.2 KBUnit - Entwickler	20

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 5 der Abbildung 1, siehe unten. Sie wurde von Philipp Sprengholz, Alexander Georgiev, Patrick Pete, Yannis Herbig, Gokhan Arslan, Iris Grcic, Siraj Taeb und Ursula Oesing entwickelt. Die vorliegende Anwendungsdokumentation wurde von Philipp Sprengholz und Ursula Oesing entwickelt und von Patrick Pete ergänzt. Die Realisierung von Anwendungsteilen, welche den Schritt 6 unterstützen, sind geplant.



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.

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

Properties for Zinsrechner		— D >	<
type filter text	Java Compiler	↓ ↓ ↓	000
> Resource Builders	Enable project specific settings	Configure Workspace Settings	^
Coverage	JDK Compliance		
Java Build Path	Use compliance from execution environment on the 'Java Build Path'		
 Java Code Style Java Compiler 	Compiler compliance level:	16 ~	
Javadoc Location	Use ' <u>r</u> elease' option		
> Java Editor	Use defaul <u>t</u> compliance settings		
PMD	Enable preview features for Java 16		
Project Facets Project Natures	Preview features with severity level:	Warning \sim	
Project References	Ge <u>n</u> erated .class files compatibility:	16 🗸 🗸	
Refactoring History	Source competibility	16 🗸	
Run/Debug Settings	oource co <u>m</u> patibility.	10	
> lask Repository	Disallow identifiers called 'assert':	Error 🗸	
> Validation	Disallo <u>w</u> identifiers called 'enum':	Error \sim	
WikiText	Classfile Generation Add variable attributes to generated class files (used by the debugger) Add line number attributes to generated class files (used by the debugger) Tattitum of the second s		~
?		Apply and Close Cancel	

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, *https://tomcat.apache.org/download-70.cgi*.

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.

New Dynamic Web F	Project				×
Dynamic Web Projec	t				
Create a standalone Dy	namic Web project	or add it to a new o	r existing Enterprise A	pplication.	
Project name:					
Project location					
Use default location	1				
Location: D:\HS_Bochu	Im\ForschungVero	effentlichung\KBUnit	\20190401_KBUnit\K	Browse	<u></u>
Target runtime					
Apache Tomcat v9.0			~	New Runti	me
Dynamic web module v	version				
4.0					×.
Configuration					
Default Configuration	for Apache Tomcat	v9.0	~	Modify	·
A good starting point installed to add new fu	for working with A Inctionality to the p	pache Tomcat v9.0 ru project.	untime. Additional fac	ets can <mark>l</mark> ater	be
?	< Back	Next >	Finish	Cance	I

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.

New Server Runtime Environment –	×
New Server Runtime Environment	
Define a new server runtime environment	
Select the type of runtime environment:	
type filter text	
Apache Tomcat v7.0	^
Apache Tomcat v8.0	
🗄 Apache Tomcat v8.5	
apache Tomcat v9.0	~
Apache Tomcat v9.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, 7, and 8 Web modules.	
Create a new local server	
? < Back Next > Finish Cancel	el

Abbildung 6: Anlage eines RESTful Webservices: 2

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

New Server Runtime Environment	—	×
Tomcat Server		
Na <u>m</u> e:		
Apache Tomcat v9.0		
Tomcat installation <u>d</u> irectory:		
C:\Program Files\Apache Software Foundation\apache-tomcat-9.0.54	B <u>r</u> owse	
apache-tomcat-9.0.46	Download and	Install
JRE:		
jdk-17.0.1 ×	Installed JR	Es
? < <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cance	9

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.

New Dynamic	Web Project		×
Web Module			
Configure web m	odule settings.	 	
Context <u>r</u> oot:	KBUnitServer		
Content <u>d</u> irectory:	src/main/webapp		
Generate web.x	ml deployment descriptor		
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cance	el

Abbildung 8: Anlage eines RESTful Webservices: 4

Das Projekt wird angelegt.

KBUnit_Server - KBUnitServer/src/main/webapp	/WEB-INF/web.xml - Eclipse IDE					-		\times
<u>File Edit Navigate Search Project Run Window</u>	w <u>H</u> elp							
Image: A the second	9 😂 🛷 🔻 😼 🗉 🕇 🕹 42 🔍	× ≥ 0 = ¥ 2, ⊙ .e ∃	5 x 19 ▼ 6 ▼ ⇔ ¢ ¢	• • • 🖬			Q	1 🖪 🛛 🥵
KBUnitServer X KBUnitServer X X X X X X X	<pre>*web.xml × 1<7xml version="1.0" i 2 web-app> 3 <display-name>KBU 4 <welcome-file>i 5 <welcome-file>i 6 <welcome-file>i 7 <welcome-file>i 8 <welcome-file>i 9 <welcome-file>i 10 </welcome-file>i 11 </welcome-file>i 11 </welcome-file>i</welcome-file></welcome-file></welcome-file></display-name></pre>	encoding="UTF-8"?; nitServert> ndex.htmlndex.htmefault.htmlefault.htmlefault.jspst>	> -file> file> file> me-file> =-file> =-file>		E Outline × There is no action outline.	Task List	₽ that pro	8 □ □ vides an
✓ ≥ webapp				\sim				
> 🗁 META-INF	<			>				
✓ 🍅 WEB-INF	🗈 Markers 🗖 Properties 🤻 Ser	rvers × 🕷 Data Source Ex	plorer 🛯 Snippets			8000	5° 🔳 EP	8
ie no ⊻ web.xml	No servers are available. Click t	his link to create a new se	<u>rver</u>					
	Writable	Insert	2:9:48					1 🖓

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	
228	<dependencies></dependencies>
238	<dependency></dependency>
24	<pre><groupid>org.glassfish.jersey.containers</groupid></pre>
25	<artifactid>jersey-container-servlet</artifactid>
26	<pre><version>2.29.1</version></pre>
27	
289	<dependency></dependency>
29	<pre><groupid>org.glassfish.jersey.inject</groupid></pre>
30	<artifactid>jersey-hk2</artifactid>
31	<pre><version>2.29.1</version></pre>
32	
33	
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.

🔘 Add and Remove				×
Add and Remove				
Modify the resources that are confi	gured on the server			
Move resources to the right to confi	gure them on the ser	ver		
<u>A</u> vailable:		<u>C</u> onfigured:		
🐻 KBUnitServer				
	A <u>d</u> d >			
	< <u>R</u> emove			
	Add A <u>l</u> l >>			
	<< Re <u>m</u> ove All			
✓ If server is started, publish chang	es <u>i</u> mmediately			
? < <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel	

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.



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.



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*.

Properties for KBUnitServer		– 🗆 X	
type filter text	Server	← → ⇒ → →	,
 > Resource > Builders > Coverage > Deployment Assembly Java Code Style > Java Editor > Java Script > JBehave > JSP Fragment > Project Facets > Project References Refactoring History Run/Debug Settings > Server > Server > Server Service Policies Targeted Runtimes > Task Repository > Task Rags > Validation 	Project: KBUnitServer (J2EE Web module) Always use the following server when running this project: Some the following server at localhost Tomcat v9.0 Server at localhost	ore Defaults Apply	
?	Apply a	nd Close Cancel]

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*.

~	🍫 Classpath
	> 👼 jackson-annotations-2.13.1.jar - KBUnitServer/src/main/webapp/WEB-INF/lib
	> 👼 jackson-core-2.13.1.jar - KBUnitServer/src/main/webapp/WEB-INF/lib
	> 👼 jackson-databind-2.13.1.jar - KBUnitServer/src/main/webapp/WEB-INF/lib
	> 👼 jdom-2.0.6.jar - KBUnitServer/src/main/webapp/WEB-INF/lib
	> 🛋 Maven Dependencies
	> 🛋 Server Runtime [Apache Tomcat v9.0]

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:

xxx:\...\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 durch die Wissensträger durchgeführt.

Voraussetzungen

Für KBUnit - Wissensträger benötigen Sie Sie die folgenden Bibliotheken.

Classpath apiguardian-api-1.0.0.jar - KBUnitWissenstraeger/lib assertj-core-3.14.0.jar - KBUnitWissenstraeger/lib auto-value-1.7.jar - KBUnitWissenstraeger/lib auto-value-annotations-1.7.jar - KBUnitWissenstraeger/lib byte-buddy-1.12.9.jar - KBUnitWissenstraeger/lib hamcrest-core-1.3.jar - KBUnitWissenstraeger/lib jackson-annotations-2.13.1.jar - KBUnitWissenstraeger/lib/jackson jersey iackson-core-2.13.1.jar - KBUnitWissenstraeger/lib/jackson jersev Jackson-databind-2.13.1.jar - KBUnitWissenstraeger/lib/jackson jersey jdom-2.0.6.jar - KBUnitWissenstraeger/lib iersev-client-1.19.1.iar - KBUnitWissenstraeger/lib/jackson jersev Jersey-core-1.19.1.jar - KBUnitWissenstraeger/lib/jackson jersey jsr305-3.0.2.jar - KBUnitWissenstraeger/lib isr311-api-1.1.1.jar - KBUnitWissenstraeger/lib/jackson jersey Junit-4.13.jar - KBUnitWissenstraeger/lib iunit-jupiter-api-5.6.0.jar - KBUnitWissenstraeger/lib junit-jupiter-engine-5.6.0.jar - KBUnitWissenstraeger/lib Junit-jupiter-params-5.6.0.jar - KBUnitWissenstraeger/lib Junit-platform-commons-1.6.0.jar - KBUnitWissenstraeger/lib junit-platform-engine-1.6.0.jar - KBUnitWissenstraeger/lib Junit-platform-launcher-1.6.0.jar - KBUnitWissenstraeger/lib Junit-vintage-engine-5.6.0.jar - KBUnitWissenstraeger/lib > mockito-core-4.5.1.jar - KBUnitWissenstraeger/lib > Amockito-junit-jupiter-4.5.1.jar - KBUnitWissenstraeger/lib > objenesis-3.2.jar - KBUnitWissenstraeger/lib > opentest4i-1.1.1.jar - KBUnitWissenstraeger/lib > Swingx-all-1.6.3.jar - KBUnitWissenstraeger/lib

Abbildung 15: Bibliotheken von KBUnit - Wissenstraeger

Erstellen Sie 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 lcons ausgeliefert.

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

¥	B	KB	Unit
	۷	#	src
		>	business.hauptfenster
		>	business.hauptfenster.extraModels
		>	# gui.dialogfenster
		>	🕭 gui.hauptfenster
		>	# gui.hauptfenster.konfigurationsgenerator
		>	# gui.hauptfenster.konfigurationsgenerator.ipog
		>	gui.hauptfenster.navigation
		>	# gui.hauptfenster.neueKonfiguration
		>	gui.hauptfenster.testergebnisse
		>	# gui.hilfe
		>	J hilfe
		>	# main
		>	tester.testCaseInfo
		>	# tester.testCaseRunner
	>	=\	JRE System Library [jdk-17.0.1]
	>	=\	Referenced Libraries
	>	ß	icons
	>	B	iconsAlt
	>	B	lib
			config.xml

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.

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.

Run Configurations	-	- 🗆	×
Create, manage, and run config	urations		
Run a Java application			
	Name: Main (2)		
	O Main Main Main Arguments ▲ JRE Superiore Dependencies Source ■ Environment ■ Common Prototype O Prototype O Main Main Main Arguments ▲ JRE Main Arguments ▲ JRE Main Arguments ▲ JRE Main Main Main Arguments ▲ JRE Main Main Main Main Arguments ▲ JRE Main Main Main Main Arguments ▲ JRE Main Main Main Main Main Main Main Arguments ▲ JRE Main Main Main Main Main Main Main Main		
Eclipse Application	Program arguments:		^
Eclipse Data Tools			~
Generic Server			<u>~</u>
Generic Server(External Laun		Variables	s
Gradle lask			
Gradle lest	VM arguments:		
HIIP Preview	add-opens java.base/java.lang=ALL-UNNAMED		0
J2EE Preview		Verielder	5 1
Java Applet		variables	••••
Main (2)	Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching		
	Use @a <u>rg</u> file when launching		
Jrr KBI InitWissenstraeger	Working directory:		~
<pre>c Roometvissenstracger +</pre>			
Filter matched 24 of 26 items	Sho <u>w</u> Command Line Revert	Apply	<u>.</u>
0	Run	Close	9

Abbildung 17: VM-Argumente von KBUnit - Wissenstraeger

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

Alternativ kann KBUnit - Wissenstraeger als ausführbares Java-Archiv *KBUnit_Wissenstraeger.jar* gemeinsam mit der Konfigurationsdatei *config.xml* und dem Verzeichnis mit icons gestartet werden. Erstellen Sie aus dem vorliegenden Projekt *KBUnit_Wissenstraeger.jar*. (In Eclipse können Sie dazu im Kontextmenü des Projekts *Export.../Java/Runnable JAR file* auswählen. Klicken Sie auf *Next*> und wählen Sie in dem folgenden Fenster *Copy required libraries into a sub-folder next tot he generated JAR* aus.) Kopieren Sie alle Elemente auf Ihr System in dasselbe Verzeichnis und passen Sie wie oben beschrieben die Konfigurationsparameter in der Datei *config.xml* an die vorhandene Situation an.



Abbildung 18: Struktur von KBUnit – Wissenstraeger als jar-File

5.2 KBUnit – Entwickler

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

Voraussetzungen

Für KBUnit - Entwickler benötigen Sie den Quellcode von KBUnit – Entwickler und die in Abbildung 19 aufgelisteten Bibliotheken. Legen Sie die Bibliotheken in ein Verzeichnis *lib*. Fügen Sie die Bibliotheken, wie im Kapitel 4 für KBUnit – Server beschrieben, dem Projekt hinzu. (Bitte beachten Sie die Lizenzbedingungen für die verwendeten Bibliotheken!).

🍫 Classpath apiguardian-api-1.0.0.jar - Zinsrechner/lib asm-9.0.jar - Zinsrechner/lib asm-analysis-8.0.1.jar - Zinsrechner/lib asm-commons-8.0.1.jar - Zinsrechner/lib asm-tree-8.0.1.jar - Zinsrechner/lib assertj-core-3.14.0.jar - Zinsrechner/lib byte-buddy-1.12.9.jar - Zinsrechner/lib > a commons-csv-1.7.jar - Zinsrechner/lib i commons-lang3-3.10.jar - Zinsrechner/lib b hamcrest-core-1.3.jar - Zinsrechner/lib jackson-annotations-2.13.1.jar - Zinsrechner/lib/jackson_jersey jackson-core-2.13.1.jar - Zinsrechner/lib/jackson_jersey jackson-databind-2.13.1.jar - Zinsrechner/lib/jackson_jersey idom-2.0.6.jar - Zinsrechner/lib jersey-client-1.19.1.jar - Zinsrechner/lib/jackson_jersey jersey-core-1.19.1.jar - Zinsrechner/lib/jackson_jersey ison-20200518.jar - Zinsrechner/lib/jackson_jersey ison-simple-1.1.1.jar - Zinsrechner/lib/jackson_jersey jsr305-3.0.2.jar - Zinsrechner/lib Jisr311-api-1.1.1.jar - Zinsrechner/lib/jackson_jersey > 👼 junit-4.13.jar - Zinsrechner/lib Junit-jupiter-api-5.6.0.jar - Zinsrechner/lib junit-jupiter-engine-5.6.0.jar - Zinsrechner/lib junit-jupiter-params-5.6.0.jar - Zinsrechner/lib Junit-platform-commons-1.6.0.jar - Zinsrechner/lib junit-platform-engine-1.6.0.jar - Zinsrechner/lib junit-platform-launcher-1.6.0.jar - Zinsrechner/lib junit-platform-runner-1.6.0.jar - Zinsrechner/lib junit-platform-suite-api-1.6.0.jar - Zinsrechner/lib junit-vintage-engine-5.6.0.jar - Zinsrechner/lib > Mathematical Sciences - Scie > mockito-junit-jupiter-4.5.1.jar - Zinsrechner/lib > di objenesis-3.2.jar - Zinsrechner/lib > Bopentest4j-1.1.1.jar - Zinsrechner/lib org.jacoco.core-0.8.6.jar - Zinsrechner/lib > org.jacoco.report-0.8.6.jar - Zinsrechner/lib

> Gorg.junit.platform.runner_1.3.1.v20181005-1442.jar - Zinsrechner/lib

Abbildung 19: 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 20: 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.

Run Configurations	-	
Create, manage, and run configurations		
Run a Java application		
	Name: Main	
	🛛 🕜 Main (x)= Arguments 🛋 JRE 😽 Dependencies 🧤 Source 🖾 Environment 🔲 🖸	ommon »1
🖸 Main \land	Program arguments:	
Main (2)		~
J lerminai		
Ju asdfsdfsdf		
Ju BerechnungsserviceKBUnitTest		Variables
Ju BerechnungsserviceTest_200408_171850	VM arguments:	
Ju BerechnungsserviceTest_200409_102649	add-modules=iavafx.controls.iavafx.fxml	~
Ju Besonderer Mitarbeiter / PlusitTert		
Ju BesondererMitarbeiterTest		
Ju BesondererUndImplementierenderMitarbeiterKBUnitTest		Variable <u>s</u>
J u darlehen	Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching	
Ju DarlehenAnalyticsKBUnitTest (2)	Use @a <u>r</u> gfile when launching	
Ju DarlehenAnalyticsTest_200409_105608	Working directory	
Jt Darlehen Analytics lest_200409_113951	Default	
Ju DarlehenAnalyticsTest 200409 123132	S(Workspace_IOC:Zinsrechner)	
Ju DarlehenAnalyticsTest_200409_123450	O Ot <u>h</u> er:	
Ju DarlehenAnalyticsTest_200409_130856	Workspace <u>File System</u>	Variabl <u>e</u> s
Ju DarlehenAnalyticsTest_200416_140412		
Ju DarlehenAnalyticsTest2	·	
Filter matched 127 of 135 items	Sho <u>w</u> Command Line Revert	Apply
(?)	Run	Close
<u> </u>		

Abbildung 21: 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 22.

~	x 😫 Zinsrachpar				
) (# src				
	> /# toot				
	-	~	# odah khunit		
		ĺ,	# priab.kbunit		
		ĺ,	prior business connector		
		Ś	priab.kbunit.business.crc/mornationoeneration # priab.kbunit.business.iacocoCoverage		
		Ś	# priab.kbunit.business.gacococoverage		
		ĺ,	# priab.kbunit.business.run		
		Ś	priab.kbunit.business.windowBacklogReader parcerlina		
		Ś	# priab.kbunit.business.windowBacklogReader.parserScrumBo		
		ĺ,	priab.kbunit.business.windowbackiogneader.paiserscrumbo aprilab.kbunit.business.windowMainEramo		
		\$	priab.kbunit.business.windowMainFrame.windowNewTestkonfiguration		
		>	pride.ibunit.business.windowParametrisierung		
		>	# pridbikbunit enums		
		>	pride.countertains pride.countertains		
		>			
		>	# prlab.kbunit.gui.windowBacklogReader		
		>	# prlab.kbunit.gui.windowMainFrame		
		>	priab.kbunit.gui.windowMainFrame.windowNewTestkonfiguration		
		>	# prlab.kbunit.gui.windowParametrisierung		
		>	prlab.kbunit.resources.css		
		>	⊕ prlab.kbunit.resources.doku		
		>	prlab.kbunit.resources.images		
		>	prlab.kbunit.resources.view		
		>	# prlab.kbunit.scanner		
		>	# prlab.kbunit.tester		
		>	# prlab.kbunit.util		
		>	# prlab.kbunit.xmlFileLoader		
	>	æ	testForKBUnit		
	>	=\	JavaFX		
	>	اھ	JRE System Library [jdk-17.0.1]		
	>	14	Referenced Libraries		
	>	0	lib		

Abbildung 22: 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.*

Impressum

Prof. Dr. Ursula Oesing

ursula.oesing@hs-bochum.de

Fachbereich Elektrotechnik und Informatik Hochschule Bochum Am Hochschulcampus 1 44801 Bochum