


Unit and Integration Tests in Spring Boot

by Marco Giglione  · Jul. 18, 18 · Performance Zone · Tutorial

Sensu is an open source monitoring event pipeline. Try it today.

1. Overview

In this post, we'll have a look at how to write tests unit and integration in a Spring Boot environment. You can find tons of tutorials online on this topic but it is very difficult to find all the information that you need in just one page. I often noticed that junior developers are confusing between unit and integration test especially when speaking about spring ecosystem and I'll try to clarify the usage of different annotations used in different contexts.

2. Unit vs. Integration tests

Wikipedia says about unit testing: *"In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use."*

and about Integration testing: *"Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group."*

In simple words, when we do unit test, we test just a single unit of code, one method at a time, excluding all other components that interact with our under testing one.

In integration tests on the other side, we test the integration between components. **Thanks to unit testing, we know that components behave as required individually, but we don't know how they'll work altogether.** This is the responsibility of integration tests.

3. Java Test Unit

All Java developers know about **JUnit** as the main framework to perform test unit. It offers a lot of annotations to make assertions on expectations.

Hamcrest is an additional framework for software tests. Hamcrest allows checking for conditions in your code using existing matchers classes and it also allows you to define your custom matcher implementations. To use Hamcrest matchers in JUnit you have to use the *assertThat* statement followed by one or several matchers.

Here you can see simple tests using both frameworks:

```
1  import static org.hamcrest.CoreMatchers.allOf;
2  import static org.hamcrest.CoreMatchers.anyOf;
3  import static org.hamcrest.CoreMatchers.both;
4  import static org.hamcrest.CoreMatchers.containsString;
5  import static org.hamcrest.CoreMatchers.equalTo;
6  import static org.hamcrest.CoreMatchers.everyItem;
7  import static org.hamcrest.CoreMatchers.hasItems;
8  import static org.hamcrest.CoreMatchers.not;
9  import static org.hamcrest.CoreMatchers.sameInstance;
10 import static org.hamcrest.CoreMatchers.startsWith;
11 import static org.junit.Assert.assertArrayEquals;
12 import static org.junit.Assert.assertEquals;
13 import static org.junit.Assert.assertFalse;
14 import static org.junit.Assert.assertNotNull;
15 import static org.junit.Assert.assertNotSame;
16 import static org.junit.Assert.assertNull;
17 import static org.junit.Assert.assertSame;
18 import static org.junit.Assert.assertThat;
19 import static org.junit.Assert.assertTrue;
20
21 import java.util.Arrays;
22
23 import org.hamcrest.core.CombinableMatcher;
24 import org.junit.Test;
25
26 public class AssertTests {
27     @Test
28     public void testAssertArrayEquals() {
29         byte[] expected = "trial".getBytes();
30         byte[] actual = "trial".getBytes();
31         assertArrayEquals("failure - byte arrays not same", expected, actual);
32     }
33
34     @Test
35     public void testAssertEquals() {
36         assertEquals("failure - strings are not equal", "text", "text");
37     }
38
39     @Test
40     public void testAssertFalse() {
41         assertFalse("failure - should be false", false);
42     }
43 }
```

```

43
44     @Test
45     public void testAssertNotNull() {
46         assertNotNull("should not be null", new Object());
47     }
48
49     @Test
50     public void testAssertNotSame() {
51         assertNotSame("should not be same Object", new Object(), new Object());
52     }
53
54     @Test
55     public void testAssertNull() {
56         assertNull("should be null", null);
57     }
58
59     @Test
60     public void testAssertSame() {
61         Integer aNumber = Integer.valueOf(768);
62         assertSame("should be same", aNumber, aNumber);
63     }
64
65     // JUnit Matchers assertThat
66     @Test
67     public void testAssertThatBothContainsString() {
68         assertThat("albumen", both(containsString("a")).and(containsString("b")));
69     }
70
71     @Test
72     public void testAssertThatHasItems() {
73         assertThat(Arrays.asList("one", "two", "three"), hasItems("one", "three"));
74     }
75
76     @Test
77     public void testAssertThatEveryItemContainsString() {
78         assertThat(Arrays.asList(new String[] { "fun", "ban", "net" }), everyItem(containsString("a")));
79     }
80
81     // Core Hamcrest Matchers with assertThat
82     @Test
83     public void testAssertThatHamcrestCoreMatchers() {
84         assertThat("good", allOf(equalTo("good"), startsWith("good")));
85         assertThat("good", not(allOf(equalTo("bad"), equalTo("good"))));
86         assertThat("good", anyOf(equalTo("bad"), equalTo("good")));

```

```

87     assertThat(7, not(CombinableMatcher.&lt;Integer&gt; either(equalTo(3)).or
88     assertThat(new Object(), not(sameInstance(new Object()))));
89 }
90
91 @Test
92 public void testAssertTrue() {
93     assertTrue("failure - should be true", true);
94 }
95 }

```

4. Introducing Our Example

Let's write our simple application. The idea is to provide a basic search engine for manga.



4.1. Maven Dependencies

First of all, we need to add some dependency to our project

```

1  <dependency>
2      <groupId>org.springframework.boot</groupId>
3      <artifactId>spring-boot-starter-test</artifactId>
4      <scope>test</scope>
5  </dependency>
6  <dependency>
7      <groupId>org.springframework.boot</groupId>
8      <artifactId>spring-boot-starter-web</artifactId>
9  </dependency>
10 <dependency>
11     <groupId>org.projectlombok</groupId>
12     <artifactId>lombok</artifactId>
13     <version>1.16.20</version>
14     <scope>provided</scope>
15 </dependency>

```

4.2. Define the Model

Our model is really simple; it is made up of only two classes: Manga and MangaResult.

4.2.1. Manga Class

Manga class represents an instance of manga as retrieved by the system. I used Lombok to reduce boilerplate code.

```
1 package com.mgiglione.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Builder;
5 import lombok.Getter;
6 import lombok.NoArgsConstructor;
7 import lombok.Setter;
8
9 @Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
10 public class Manga {
11     private String title;
12     private String description;
13     private Integer volumes;
14     private Double score;
15 }
```

4.2.2. MangaResult

MangaResult is a wrapper class that contains a list of mangas.

```
1 package com.mgiglione.model;
2
3 import java.util.List;
4
5 import lombok.Getter;
6 import lombok.NoArgsConstructor;
7 import lombok.Setter;
8
9 @Getter @Setter @NoArgsConstructor
10 public class MangaResult {
11     private List<Manga> result;
12 }
```

4.3. Implementing the Service

For implementing the service, we will use API freely exposed by Jikan Moe.

RestTemplate is the Spring class that I use to make REST calls to the API.

```
1  package com.mgiglione.service;
2
3  import java.util.List;
4
5  import org.slf4j.Logger;
6  import org.slf4j.LoggerFactory;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.stereotype.Service;
9  import org.springframework.web.client.RestTemplate;
10
11 import com.mgiglione.model.Manga;
12 import com.mgiglione.model.MangaResult;
13
14 @Service
15 public class MangaService {
16
17     Logger logger = LoggerFactory.getLogger(MangaService.class);
18     private static final String MANGA_SEARCH_URL="http://api.jikan.moe/search/manga/"
19
20     @Autowired
21     RestTemplate restTemplate;
22
23     public List<Manga> getMangasByTitle(String title) {
24         return restTemplate.getForEntity(MANGA_SEARCH_URL+title, MangaResult.class).g
25     }
26
27 }
```

4.4. Implementing the Controller

The next step on the list is to write down the REST controller that exposes two endpoints, one synchronous and one asynchronous, just for testing purposes. This controller makes use of the Service defined above.

```
1  package com.mgiglione.controller;
2
3  import java.util.List;
4  import java.util.concurrent.CompletableFuture;
5
6  import org.slf4j.Logger;
7  import org.slf4j.LoggerFactory;
8  import org.springframework.beans.factory.annotation.Autowired;
```

```

9  import org.springframework.scheduling.annotation.Async;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RequestMethod;
13 import org.springframework.web.bind.annotation.ResponseBody;
14 import org.springframework.web.bind.annotation.RestController;
15
16 import com.mgiglione.model.Manga;
17 import com.mgiglione.service.MangaService;
18
19 @RestController
20 @RequestMapping(value = "/manga")
21 public class MangaController {
22
23     Logger logger = LoggerFactory.getLogger(MangaController.class);
24
25     @Autowired
26     private MangaService mangaService;
27
28     @RequestMapping(value = "/async/{title}", method = RequestMethod.GET)
29     @Async
30     public CompletableFuture<List<Manga>> searchASync(@PathVariable(name = "title") S
31         return CompletableFuture.completedFuture(mangaService.getMangasByTitle(title)
32     }
33
34     @RequestMapping(value = "/sync/{title}", method = RequestMethod.GET)
35     public @ResponseBody <List<Manga>> searchSync(@PathVariable(name = "title") Strin
36         return mangaService.getMangasByTitle(title);
37     }
38
39 }

```

4.5. Launching and Testing the System

```
mvn spring-boot:run
```

Then let's try it:

```
curl http://localhost:8080/manga/async/ken
```

```
curl http://localhost:8080/manga/sync/ken
```

Example of output:

```

1  {
2      "title": "Rurouni Kenshin: Meiji Kenkaku Romantan",
3      "description": "Ten years have passed since the end of Bakumatsu, an era of war tha
4      "volumes": 28,
5      "score": 8.69
6  },
7  {
8      "title": "Sun-Ken Rock",
9      "description": "The story revolves around Ken, a man from an upper-class family tha
10     "volumes": 25,
11     "score": 8.12
12 },
13 {
14     "title": "Yumekui Kenbun",
15     "description": "For those who suffer nightmares, help awaits at the Ginseikan Tea H
16     "volumes": 9,
17     "score": 7.97
18 }

```

5. Unit Testing the Spring Boot Application

Spring boot offers a great class to make testing easier: `@SpringBootTest` annotation

This annotation can be specified on a test class that runs Spring Boot based tests.

Provides the following features over and above the regular Spring TestContext Framework:

- Uses `SpringBootTestContextLoader` as the default `ContextLoader` when no specific `@ContextConfiguration` (loader=...) is defined.
- Automatically searches for a `@SpringBootTestConfiguration` when nested `@Configuration` is not used, and no explicit classes are specified.
- Allows custom Environment properties to be defined using the properties attribute.
- Provides support for different web environment modes, including the ability to start a fully running web server listening on a defined or random port.
- Registers a `TestRestTemplate` and/or `WebTestClient` bean for use in web tests that are using a fully running web server.

We basically have two components to test here: `MangaService` and `MangaController`

5.1. Unit Testing MangaService

To test `MangaService`, we need to isolate it from external components. In our case, we only have one external component required: *RestTemplate*, which we use to call a remote API.

What we need to do is to mock the `RestTemplate` bean and let it always respond with a fixed given response. Spring

What we need to do is to mock the restTemplate bean and let it always respond with a fixed given response. Spring Test incorporates and extends the Mockito library to configure mocked beans through the @MockBean annotation.

```
1 package com.mgiglione.service.test.unit;
2
3 import static org.mockito.ArgumentMatchers.any;
4 import static org.mockito.Mockito.when;
5
6 import java.io.IOException;
7 import java.util.List;
8
9 import org.junit.Test;
10 import org.junit.runner.RunWith;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.boot.test.context.SpringBootTest;
13 import org.springframework.boot.test.mock.mockito.MockBean;
14 import org.springframework.http.HttpStatus;
15 import org.springframework.http.ResponseEntity;
16 import org.springframework.test.context.junit4.SpringRunner;
17 import org.springframework.web.client.RestTemplate;
18 import static org.assertj.core.api.Assertions.assertThat;
19
20 import com.mgiglione.model.Manga;
21 import com.mgiglione.model.MangaResult;
22 import com.mgiglione.service.MangaService;
23 import com.mgiglione.utils.JsonUtils;
24
25 @RunWith(SpringRunner.class)
26 @SpringBootTest
27 public class MangaServiceUnitTest {
28
29     @Autowired
30     private MangaService mangaService;
31
32     // MockBean is the annotation provided by Spring that wraps mockito one
33     // Annotation that can be used to add mocks to a Spring ApplicationContext.
34     // If any existing single bean of the same type defined in the context will be re
35     @MockBean
36     private RestTemplate template;
37
38     @Test
39     public void testGetMangasByTitle() throws IOException {
40         // Parsing mock file
41         MangaResult mRs = JsonUtils.jsonFile2Object("ken.json", MangaResult.class);
42         // Mocking remote service
```

```

42      // Mocking Remote Service
43      when(template.getForEntity(any(String.class), any(Class.class))).thenReturn(n
44      // I search for goku but system will use mocked response containing only ken,
45      List<Manga> mangasByTitle = mangaService.getMangasByTitle("goku");
46      assertThat(mangasByTitle).isNotNull()
47          .isEmpty()
48          .allMatch(p -> p.getTitle()
49              .toLowerCase()
50              .contains("ken"));
51
52    }
53
54 }

```

5.2. Unit Testing MangaController

As done in the unit testing of the service, we need to isolate components. In this case, we need to mock the *MangaService* bean.

Then, we have a little further problem... Controller part is the part of the system that manages *HttpRequest*, so we need a system to simulate this behavior without starting a full HTTP server.

MockMvc is the Spring class that does that. It can be set up in different ways:

1. Using Standalone Context
2. Using WebApplication Context
3. Let Spring autoconfigure it by loading all context by using these annotations on test class `@SpringBootTest`
`@AutoConfigureMockMvc`
4. Let Spring autoconfigure it by loading just the web layer context by using these annotations on the test class
`@WebMvcTest`

```

1  package com.mgiglione.service.test.unit;
2
3  import static org.hamcrest.Matchers.is;
4  import static org.mockito.ArgumentMatchers.any;
5  import static org.mockito.Mockito.when;
6  import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.asy
7  import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get
8  import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print
9  import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonP
10 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.reque
11 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.statu
12 import static org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSe
13

```

```

14 import java.util.ArrayList;
15 import java.util.List;
16
17 import org.junit.Before;
18 import org.junit.Test;
19 import org.junit.runner.RunWith;
20 import org.springframework.beans.factory.annotation.Autowired;
21 import org.springframework.boot.test.context.SpringBootTest;
22 import org.springframework.boot.test.mock.mockito.MockBean;
23 import org.springframework.http.MediaType;
24 import org.springframework.test.context.junit4.SpringRunner;
25 import org.springframework.test.web.servlet.MockMvc;
26 import org.springframework.test.web.servlet.MvcResult;
27 import org.springframework.web.context.WebApplicationContext;
28
29 import com.mgiglione.controller.MangaController;
30 import com.mgiglione.model.Manga;
31 import com.mgiglione.service.MangaService;
32
33 @SpringBootTest
34 @RunWith(SpringRunner.class)
35 public class MangaControllerUnitTest {
36
37     MockMvc mockMvc;
38
39     @Autowired
40     protected WebApplicationContext wac;
41
42     @Autowired
43     MangaController mangaController;
44
45     @MockBean
46     MangaService mangaService;
47
48     /**
49      * List of samples mangas
50      */
51     private List<Manga> mangas;
52
53     @Before
54     public void setup() throws Exception {
55         this.mockMvc = standaloneSetup(this.mangaController).build();// Standalone co
56         // mockMvc = MockMvcBuilders.webAppContextSetup(wac)
57         // build().

```

```
// .build(),
Manga manga1 = Manga.builder()
    .title("Hokuto no ken")
    .description("The year is 199X. The Earth has been devastated by nuclear
        .build();
Manga manga2 = Manga.builder()
    .title("Yumekui Kenbun")
    .description("For those who suffer nightmares, help awaits at the Ginseik
        .build();

mangas = new ArrayList<>();
mangas.add(manga1);
mangas.add(manga2);
}

@Test
public void testSearchSync() throws Exception {

    // Mocking service
    when(mangaService.getMangasByTitle(any(String.class))).thenReturn(mangas);

    mockMvc.perform(get("/manga/sync/ken").contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.title", is("Hokuto no ken")))
        .andExpect(jsonPath("$.description", is("The year is 199X. The Earth has been devastated by nuclear"))));
}

@Test
public void testSearchASync() throws Exception {

    // Mocking service
    when(mangaService.getMangasByTitle(any(String.class))).thenReturn(mangas);

    MvcResult result = mockMvc.perform(get("/manga/async/ken").contentType(MediaType.APPLICATION_JSON)
        .andDo(print()))
        .andExpect(request().asyncStarted())
        .andDo(print())
        // .andExpect(status().is2xxSuccessful()).andReturn();
        .andReturn();

    // result.getRequest().getAsyncContext().setTimeout(10000);

    mockMvc.perform(asyncDispatch(result))
```

```

0         mockMvc.perform(asyncDispatch(result))
10
1         .andDo(print())
10
2         .andExpect(status().isOk())
10
3         .andExpect(jsonPath("$.title", is("Hokuto no ken")));
10
4
10
5     }
10
6 }

```

As you can see from the code, I chose the first solution because it is the lightest one, and we have the best governance on what we load in the Spring context.

In the async test, I had to simulate the asynchronous behavior by first calling the service and then starting the *asyncDispatch* method.

6. Integration Testing the Spring Boot Application

For the integration tests, we want to check our main components with downstream communication.

6.1. Integration Testing of MangaService

This test is really simple. We don't need to mock anything because we want to call the remote mangas API.

```

1  package com.mgiglione.service.test.integration;
2
3  import static org.assertj.core.api.Assertions.assertThat;
4
5  import java.util.List;
6
7  import org.junit.Test;
8  import org.junit.runner.RunWith;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.boot.test.context.SpringBootTest;
11 import org.springframework.test.context.junit4.SpringRunner;
12
13 import com.mgiglione.model.Manga;
14 import com.mgiglione.service.MangaService;
15
16 @RunWith(SpringRunner.class)
17 @SpringBootTest
18 public class MangaServiceIntegrationTest {
19

```

```

20     @Autowired
21     private MangaService mangaService;
22
23     @Test
24     public void testGetMangasByTitle() {
25         List<Manga> mangasByTitle = mangaService.getMangasByTitle("ken");
26         assertThat(mangasByTitle).isNotNull().isNotEmpty();
27     }
28
29 }

```

6.2. Integration Testing of MangaController

This test is pretty similar to the unit test, but in this case, we haven't a mocked service.

```

1  package com.mgiglione.service.test.integration;
2
3  import static org.hamcrest.Matchers.hasItem;
4  import static org.hamcrest.Matchers.is;
5  import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.async
6  import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get
7  import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print
8  import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonP
9  import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.reque
10 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.statu
11 import static org.springframework.test.web.servlet.setup.MockMvcBuilders.standaloneSe
12
13 import org.junit.Before;
14 import org.junit.Test;
15 import org.junit.runner.RunWith;
16 import org.springframework.beans.factory.annotation.Autowired;
17 import org.springframework.boot.test.context.SpringBootTest;
18 import org.springframework.http.MediaType;
19 import org.springframework.test.context.junit4.SpringRunner;
20 import org.springframework.test.web.servlet.MockMvc;
21 import org.springframework.test.web.servlet.MvcResult;
22 import org.springframework.web.context.WebApplicationContext;
23
24 import com.mgiglione.controller.MangaController;
25
26 @SpringBootTest
27 @RunWith(SpringRunner.class)
28 public class MangaControllerIntegrationTest {

```

```
29
30 // @Autowired
31 MockMvc mockMvc;
32
33
34 @Autowired
35 protected WebApplicationContext wac;
36
37 @Autowired
38 MangaController mangaController;
39
40 @Before
41 public void setup() throws Exception {
42     this.mockMvc = standaloneSetup(this.mangaController).build();// Standalone co
43     // mockMvc = MockMvcBuilders.webAppContextSetup(wac)
44     // .build();
45 }
46
47 @Test
48 public void testSearchSync() throws Exception {
49     mockMvc.perform(get("/manga/sync/ken").contentType(MediaType.APPLICATION_JSON
50         .andExpect(status().isOk())
51         .andExpect(jsonPath("$.*.title", hasItem(is("Hokuto no Ken")))));
52 }
53
54 @Test
55 public void testSearchASync() throws Exception {
56     MvcResult result = mockMvc.perform(get("/manga/async/ken").contentType(MediaType
57         .andDo(print())
58         .andExpect(request().asyncStarted())
59         .andDo(print())
60         .andReturn());
61
62     mockMvc.perform(asyncDispatch(result))
63         .andDo(print())
64         .andExpect(status().isOk())
65         .andExpect(jsonPath("$.*.title", hasItem(is("Hokuto no Ken"))));
66
67 }
68
69
70
71 }
```

7. Conclusions

We have seen the main differences between unit and integration tests in a Spring Boot environment, taking a look at frameworks like Hamcrest that simplify test writing, as well. Of course, you can find everything in my GitHub repository.

Sensu: workflow automation for monitoring. Learn more—download the whitepaper.

Like This Article? Read More From DZone



Unit and Integration Tests in Spring Boot



Mockito Mock vs. Spy in Spring Boot Tests



Start to Love Spring Testing With the Unit Test Assistant for Java



Free DZone Refcard Introduction to Docker Monitoring

Topics: SPRING BOOT , UNIT TESTING , PERFORMANCE , INTEGRATION TESTING , TUTORIAL , HAMCREST

Opinions expressed by DZone contributors are their own.

Performance Partner Resources

Open source resources for monitoring Chef deployments

Sensu

|

[Whitepaper] Workflow automation for monitoring

Sensu

|

How to] Run Nagios as time-series data with Sensu, InfluxDB and Grafana

Sensu

|

The Power of Automated Testing and Test Management with SmartBear and Zephyr

by Tom Alexander • Feb 11, 19 • Performance Zone • Tutorial

Dr. Milan Verma, Zephyr's Lead Client Services Engineer, and Greg Hanson, Senior Director of Global Sales

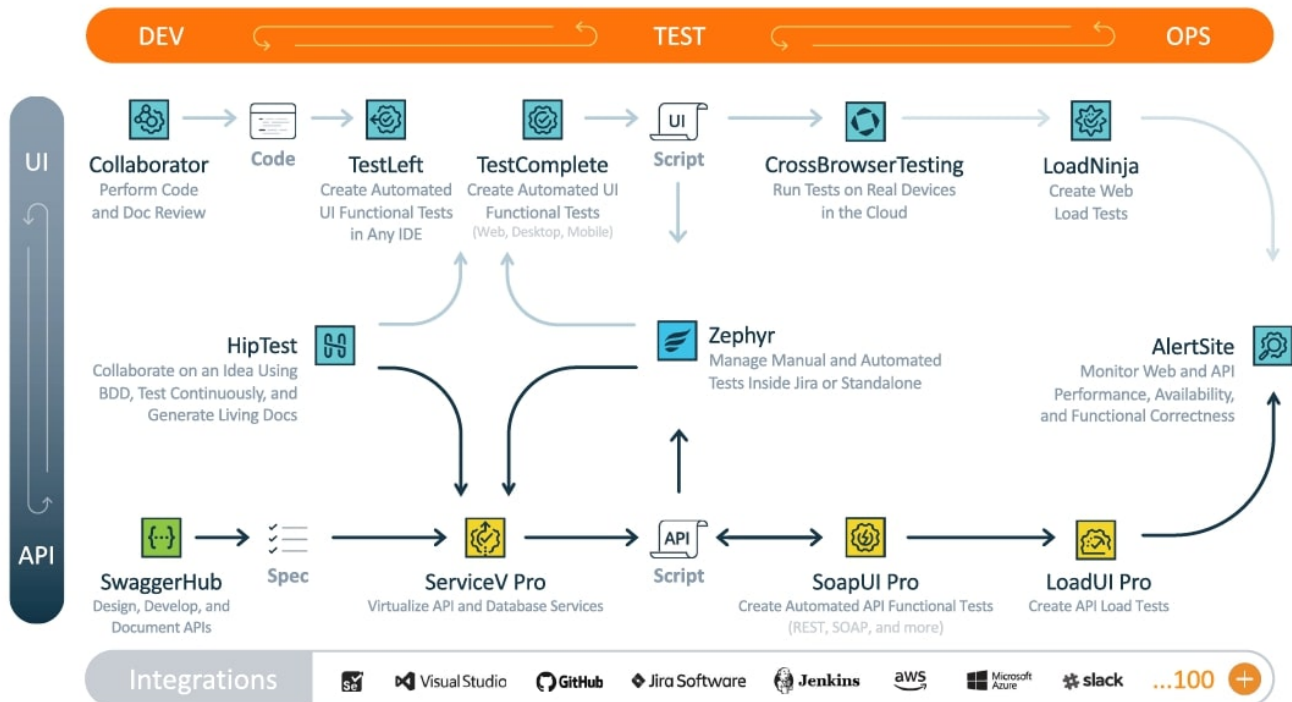
Engineering for SmartBear, recently gave a presentation and answered some essential questions on automated testing and test management with SmartBear and Zephyr. "The focus of our talk today is test automation and the power it brings to your testing journey," Greg said, by way of introduction.

"Once we help you figure test automation, we're going to talk about how to combine it with your test management platform to ensure that you get the highest level of insight into your testing practices. Then, I'll pass the mic to Milan and he will give you an in-tool demonstration of Zephyr and TestComplete, combining those two worlds of test management and test automation."

SmartBear Overview

Greg noted that SmartBear was founded in 2009 and currently supports more than six and a half million software professionals and over 22,000 companies in 194 countries, on almost every continent. "We do have some deals with penguins in Antarctica that we're working on, which may take a little more time," he joked.

High-Level Overview of SmartBear Product Offerings



Beginning with a high-level overview of SmartBear product offerings, Greg explained that SmartBear has products that stretch the entire software development life cycle, from development through testing and operations — for both front-end and back-end applications.

"From the development side, we have collaboration tools for traditional and API development to allow developers, testers and product teams to really communicate as the code is being developed to help find those defects earlier in the process."

"Moving into the testing phase, we have test automation tools for developers, for people brand new to testing, and for sophisticated automation engineers--again for both the front-end and APIs in the back end."

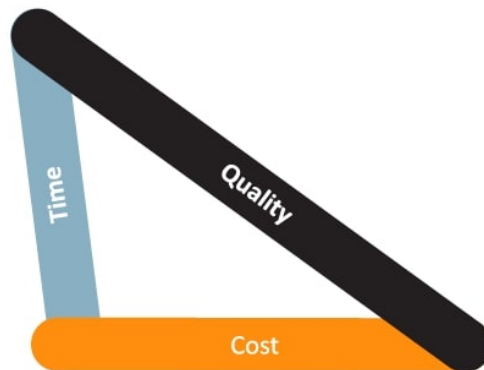
"All the way through into Ops and monitoring where we have our brand new LoadNinja tool for SaaS and web

application load testing, as well as AlertSite for monitoring all of your applications and web assets after they've been deployed — because nothing is quite as satisfying as finding a defect before your customers do and fixing it before anyone knows that there's an issue."

He called attention to two tools in the middle of the flowchart, Zephyr and Hiptest, which are both recent SmartBear acquisitions. "These tools are vital for test management, which you need to have full insight into your testing practices and how well your manual testing efforts are integrated with any or all of your test automation tools."

Test Automation

Iron Triangle Tradeoffs



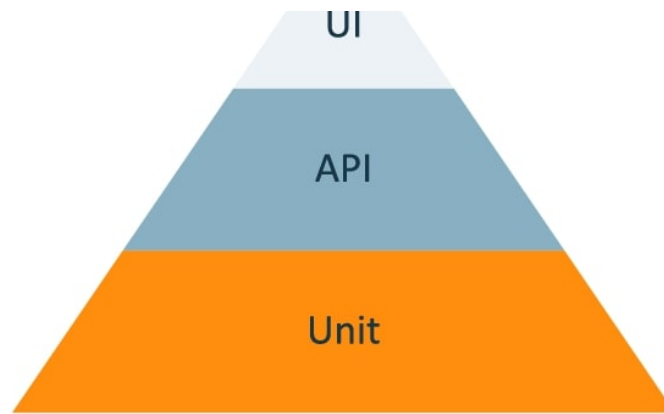
Time, cost, and quality are three widely-recognized constraints of software project management, Greg said. "There are always going to be tradeoffs. You're either going to spend more money to increase quality and save time, or you're going to increase quality and cost yourself more time." Test automation helps conquer the constraints of time, cost, and quality in this iron triangle of tradeoffs. It is a cost-effective solution to accelerate software development while ensuring quality through earlier bug identification and quick fixes.

"We don't want to go all the way to the point of automating 100 percent of the tests as they did with Windows Vista because I think we all know how that turned out, but we want to make sure that we are automating the right things in an intelligent way," Greg said.

After a real-time poll found that the majority of audience members were just getting started with automated testing, Greg emphasized the need to concentrate test automation efforts at the unit testing level. "This is really important because that's the level where you can actually test the true functionality, whether it works, yes or no. If programmers are doing this testing as they're doing development, they're going to be able to find the defects and fix them much more rapidly."

The Automated Testing Pyramid

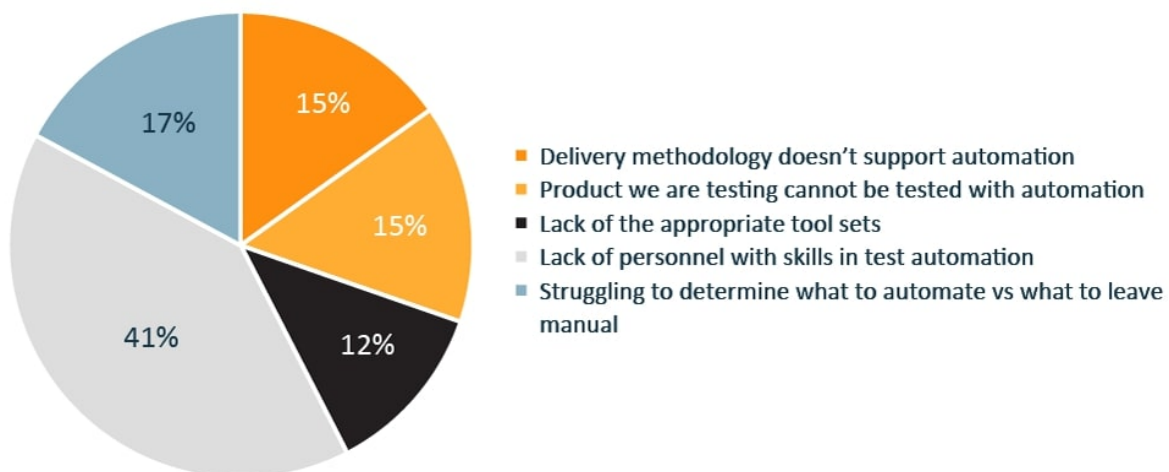




The Automated Testing Pyramid, Greg said, is a popular strategy guide that agile teams often use when in planning their test automation strategy. As shown in the illustration above, the base or largest section of the pyramid is made up of Unit Tests, which will be the case if developers in your organization are integrating code into a shared repository several times a day. This involves running unit tests, and a variety of API and User Interface (UI) tests on every check-in. "After unit tests, you move up to API assets because more and more functionality is now powered by APIs. You can really make drastic strides in full test coverage by creating healthy and robust API tests that will allow you to say, 'Yep, our application is, for the most part, going to work.'"

"The UI testing at the top is the icing on the cake that uncovers user experience hiccups, design layout flaws, and workflow issues. This robust framework, where you have pillars of unit testing underlying API and UI tests, supports your manual testing efforts. It's all designed to give the human tester as much flexibility and time as possible to do what we do best, which is the intelligent part of testing. "The Automated Testing Pyramid is just one of many things that should be part of your organization's test automation strategy. He cautioned that there are numerous challenges in getting test automation right, citing results from a recent Zephyr How the World Tests, 2018 survey.

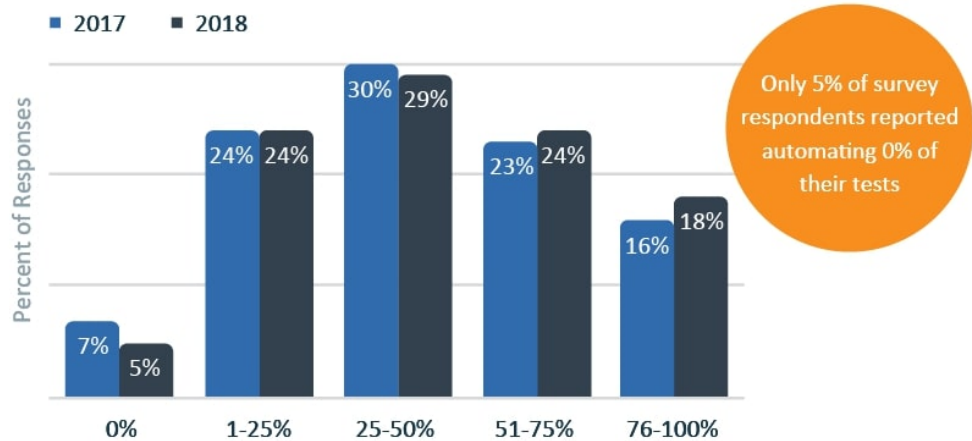
Top Challenges with Test Automation



"There are several different ideas that we can take from this, but what it boils down to is people don't have the knowledge necessary to really implement automation... Everyone wants to automate, everyone wants to save time, everyone wants to release on a much more rapid development cycle, but they just don't know how to tackle that," he said.

In spite of these problems, test automation continues to grow year over year, Greg said, citing the results of SmartBear's State of Software Testing, 2018 Industry Report.

What Percentage of Your Tests are Automated?

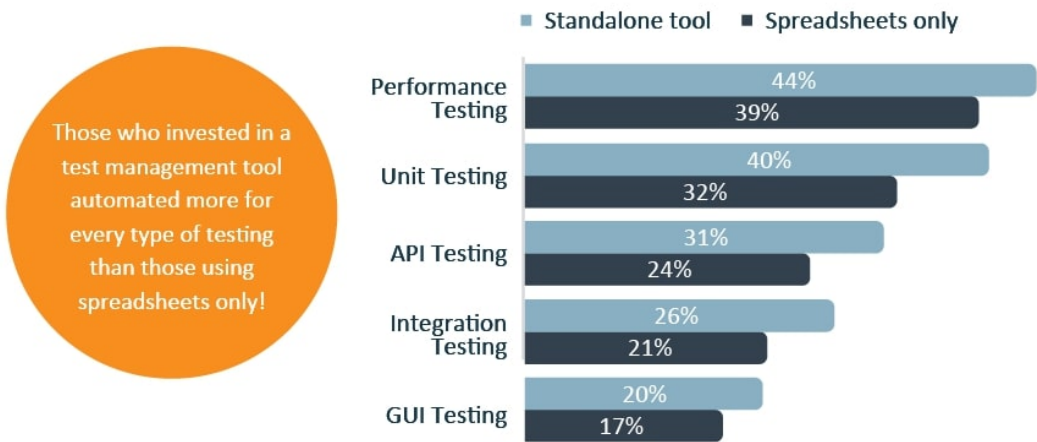


Benefits of Test Automation and Test Management

Greg stated that one notable fact from this year's survey is that a third of the people are still using spreadsheets and Word documents for test automation instead of a proprietary tool or integration with JIRA.

Perhaps the most surprising result of the survey is that respondents using a standalone test management tool are more likely to automate their testing across a majority of testing types, including integration testing, unit testing, UI testing, API testing, and performance testing, as compared to those who use spreadsheets and Word documents. Greg added emphasis to this point by using a comic book voice: "the kind I use when I'm reading to my kids" to read the following: "Those who invested in a test management tool automated more for every type of testing than those using spreadsheets only! "

Test automation and test management tool use



Zephyr Overview

Milan began the next part of the presentation with an overview of the two products that make up the Zephyr test management suite: Zephyr for Jira and Zephyr Enterprise. Both products provide scalable test environments, he said, via the data center inside Zephyr for Jira and also via Zephyr Enterprise's scalable SOA- and microservices-based architecture. Zephyr has an architecture that is continuously "bashed" or tested, using APIs, virtual users, physical users, and various configurations to ensure that the application holds up from a load testing, security, and safety perspective and can perform at scale.

He said there are quite a few approaches to reporting in Zephyr, including widgets within the Zephyr dashboard as well as live charts that are clickable to allow you to drill down to view data points in more detail. In addition to a range of traceability and custom reports, he stated that it's also possible to link up automation frameworks within Zephyr Enterprise so that you can have a consolidated view of semi-automated, fully automated, and manual testing in one place.

Since most of the webinar's audience considered themselves heavy users of JIRA, Milan said Zephyr offers multi-JIRA, multi-project support: meaning you can simultaneously connect multiple instances of JIRA and multiple projects, which can be set up within both Zephyr Enterprise and Zephyr for JIRA. It's also possible to report against all of these instances and projects by leveraging widgets available in the dashboard.

Using Zephyr and TestComplete to Do Test Automation

Milan said Zephyr offers a couple of different ways to do test automation. Zephyr Enterprise is a live HTML5 environment that connects with JIRA, and it has a whole host of applications inside it. For example, there is an artificial-intelligence (AI) engine add-on that looks at all of your requirements and testing data and helps figure out whether you're going to finish your sprint on time and what steps you can take if you're running short on time like adding additional users. It also uses a probabilistic model to ascertain whether a test case is likely to fail and flags it as a predicted fail. There's also a DevOps add-on and an add-on that allows you to work with automation frameworks using a full set of REST-based APIs to design your desired automation scripts. "Since we use these APIs to develop the UI for Zephyr in the first place, you can look at network traffic and see which APIs are being used," Milan said. You can also use the Vortex application to trigger automation jobs directly from Zephyr Enterprise.

TestComplete + Zephyr

Using Vortex, it's relatively straightforward to configure one-click automation in TestComplete and Zephyr. Vortex features include control of test executions, auto-creation of test cases, the ability to track and manage automation runs, centralized location of results for easy reporting, and analytics on testing activities. "You can trigger TestComplete suites directly from Zephyr," Milan said. "You can also automatically create test cases that have been executed using a build automation solution, a CI engine, or a CI/CD framework like Jenkins, Bamboo or TeamCity. The scripts will be auto-created inside of Zephyr Enterprise for you. For instance, if you have a job or a Java class that's being executed that runs a Selenium script, Vortex will auto-create a test case and place that into the test repository. It also places the executed version of that test case into the test case execution folder."

Milan explained that Vortex is an automation platform that uses automation agents called Zephyr Bots or ZBots, which can be installed on laptops, machines, servers, or any consolidated results location on either Linux- or

Windows-based machines. Once you've installed an agent on a given machine, you have the option of setting up a job that triggers different automation frameworks directly from the Zephyr server or setting up a folder watcher that listens in on a given results location.

"In this case," he said, "the ZBot is looking at a particular location where TestComplete results are pumped in from a Jenkins build. As the Jenkins build happens, it pushes the results into the folder and the ZBot checks every 60 seconds to see whether there are new results in this folder. As soon as it finds new results, it then populates the Zephyr test planning application with new details."

"If we look at the execution results updated by Vortex, we see all the passes and indeed a failure (red circle above) as well. The results of any automation run can be viewed in live reports (blue circle), but also in a dashboards area (green circle). You can also have a sprint-oriented view, which shows all of your manual, semi-automated and fully automated test results in one single place."

There are a number of dashboard widgets for showing automation results including one that shows the automated status of a particular release and another one that displays what percent of tests are automated with respect to every phase or by tags in a release. This information can be published by the Daily Pulse widget that offers a running view on testing activity and tracks the progress of tests created, tests executed, and defects found.

"When you observe a test case execution failure," Milan said, "you can immediately click on a button and file a defect against any failure you've observed. These defects will go straight into JIRA for you. And then you can come back into Zephyr and build a comprehensive traceability matrix for the defect, as well."

Questions and Answers

In response to a question about Vortex in the Q&A section of the presentation, Milan clarified that Vortex is an application within Zephyr Enterprise. "It's there on the bottom left-hand side (orange circle, above) within Zephyr Enterprise. Vortex doesn't work directly with Zephyr for JIRA. In order to bring TestComplete results over into Zephyr for JIRA, you need to leverage ZAPI (or Zephyr API)." (ZAPI is an add-on to Zephyr for JIRA that allows access to its testing data including the ability to view and upload information programmatically.) Zephyr for JIRA users can integrate with test automation tools using ZAPI capabilities, he explained. "You can also leverage the Zephyr for Jira, Jenkins or Bamboo add-ons. In this case, the Zephyr for Jira Jenkins add-on is the better one to use since it's open source, so you'll be able to configure it and tweak the data sent over."

Greg answered in the affirmative when asked whether the Zephyr/TestComplete integration would work "headless" on a device without a graphical user interface. He referred the questioner to SmartBear's TestExecute product, which runs TestComplete tests on computers where TestComplete is not installed. TestExecute is a lightweight version of TestComplete that supports all the testing functionality provided by the Desktop, Web, and Mobile modules, but uses fewer resources than TestComplete. "This is the most common use case in a CI/CD pipeline where people will have TestExecute set up on all of their test runner machines and Jenkins on their other CI/CD platforms, which will call out to TestExecute to run the tests," he said. "All of that backend information can then be pushed into JIRA (where you can create JIRA issues for failed tests.)"

Asked whether Vortex works with Selenium, Milan said Vortex has automation triggering capabilities that let you can run Selenium scripts directly from Vortex within Zephyr Enterprise. "You can also have Jenkins trigger your Selenium scripts and put the output in a folder you've set up to listen for Selenium results."

In response to another question about Zephyr and TestComplete, Greg said that on the SmartBear product roadmap going into 2019, the company was committed to integrating Zephyr with SmartBear's TestComplete and SoapUI automation tools. "We're really taking a very thoughtful approach to how we want to integrate all of our tools into Zephyr to make sure that it's the best for all of our users," he said.

Asked about TestComplete's ease-of-use, Greg recommended that the questioner download a full feature trial of TestComplete at SmartBear.com or TestComplete.com and take it for a spin. "It's designed to be easy to use: you just have to click, record and navigate through your application. Once you're done with that, you click play and it will run the script back so you'll know how well your app runs on different browsers and different machines." SmartBear also has a plethora of video tutorials, trainings, and documentation available freely online without any login necessary.

Milan responded positively to a question about using Vortex to run Selenium scripts against code developed with AngularJS, a JavaScript-based open-source front-end web application framework. "Absolutely. It doesn't matter which application or type of application you're testing using Selenium. You can use it to test mobile platforms, AngularJS Javascript applications, or HTML5 applications. It really doesn't matter."

He concluded: "If your question is whether X, Y, or Z tool can be integrated with Vortex and Zephyr, the answer is generally 'Yes.'"

Like This Article? Read More From DZone



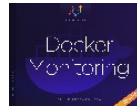
What's the Difference Between Automated Testing and Manual Testing?



HPE Software Testing Tools Changed Hands: Why It Doesn't Matter



The Power of Automation: Is 85% Test Coverage Really Possible?



**Free DZone Refcard
Introduction to Docker Monitoring**

Topics: PERFORMANCE, TESTING, ZEPHYR, TESTCOMPLETE, SMART BEAR, TEST AUTOMATION

Published at DZone with permission of Tom Alexander . [See the original article here.](#) 
Opinions expressed by DZone contributors are their own.