Estimating the Costs of Testing Microservices in an Agile Project

Folien:
Englisch

Vortragssprache:
Englisch/Deutsch

Umfang (mit Diskussion):
30 min. Vortrag + Diskussion

Vortragender: **Harry M. Sneed, TU Dresden, ZTP-Prentner Digital, Wien**

Harry Marsh Sneed, MPA, is one of the pioneers of software testing technology and lecturer for software engineering at the Institute of Business Informatics at the Universities of Regensburg, Passau, Dresden and Budapest (H). He is a guest lecturer at the Universities of Sannio (I), Szeged (H), Koblenz, FH Technikum Wien und Hagenberg (A).

Since 1969 Harry M. Sneed has worked as a programmer, analyst, project manager, laboratory manager, managing director, author, researcher and lecturer in the United States and Europe. He is the author of more than 450 professional articles and 23 books in German and English.

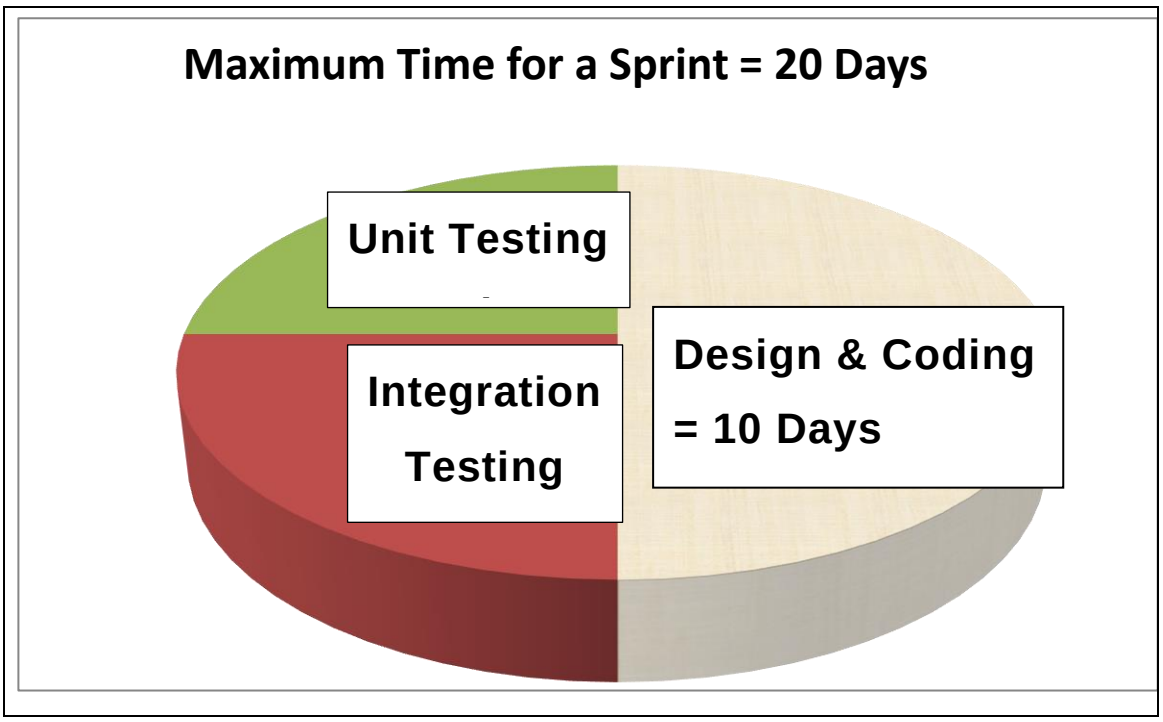
He is a multiple award-winner (ISTQB® International Software Testing Excellence Award 2013, Deutscher Preis für SW-Qualität 2011, Stevens Award 2008) and active in the IEEE, ACM and in professional societies of the GI (GI-Fellow). Harry M. Sneed is a member of the program committees for the international conferences on maintenance, metrics, reengineering, program comprehension and web evolution.

ASQT 2020 Virtual Conference
Estimating the Costs of Testing Microservices in an Agile Project

Harry M. Sneed
Technical University of Dresden
ZTP-Prentner Digital, Vienna, Austria

1. Introduction to testing Microservices

The testing approach presented here is to fit the testing effort required to test a web service with the time allowed in an agile project and still attain a minimum test coverage. Agile projects are governed by a release cycle of maximum 4 weeks [Beck01]. The project reported on here is such an agile scrum project with a tight release schedule. Every sprint should deliver a new release, i.e., a new microservice with a test coverage of at least 90% branch = C1 coverage. Past experience has shown that testing requires at least 50% of the total effort to deliver a new service [Black99]. Of that, one half of the effort is for unit testing and the other half for integration testing. Thus, if the goal is to produce a new microservice every 4 weeks, then two weeks will be for design and coding and two weeks for testing. Even if the code of a service is taken over from existing code, i.e. stripped and wrapped, it still has to be tested. That means the effort for testing remains even when the effort for designing and coding is eliminated. Of the two weeks for testing, 5 days will be for unit testing, i.e. testing the service in a simulated environment, and 5 days for integration testing, i.e., testing the service in the target environment together with the other existing services. Based on those assumptions the release delivery dates can be fixed at the start of a sprint and cannot be changed during the course of the sprint [Linz13].



This means the services to be delivered must be testable within these time limits. This is referred to as backward planning [Boeh03]. The size and complexity of the target service is determined by the effort and time required to test it. If the effort is too great and the time too long, the service size and complexity must be reduced according to the Boehm equation [Boeh99]:

Effort = (Size * Complexity) / Productivity

2. Conducting a Benchmark Test

If the services are newly developed ones, they should be designed to fulfill these criteria. If they are reengineered services, they must be refactored to fulfill the criteria. That means the designer or reengineer of the services must know how much code and how much functionality can be tested within a given time period. For this experiment 7 Java microservices with varying sizes were tested with 4 different coverage measurements:

- Statement coverage
- Branch coverage
- Path coverage
- Parameter coverage [CrGr09].

The Java services used for the benchmark test were as follows:

1) a calendar conversion service,
2) an order entry service ,

3) a savings-and-loan partner update service,
4) a beauty salon billing service,
5) a geometric form query service,
6) a user authorization service,
7) a bank mail service.

The goal of the benchmark test was to measure the average effort required to execute a unit test case in a typical Java microservice. This effort in person hours was then multiplied by the complexity-adjusted number of test cases required to attain the desired test coverage of each target service. The results were as follows:

Service	Opers	Stmts	Logic Branches	Params	FuncPts	Test Paths	Tester Hours
Calendar	3	473	31	38	12	15	8
OrderEntry	16	625	187	43	29	92	37
BauSparer	17	276	47	64	35	22	13
BeautySalon	24	429	72	54	18	33	21
Geometry	5	510	73	19	9	36	18
Authorize	27	573	265	19	22	130	65
MailService	48	3317	762	211	126	278	88
Total	**140**	**6203**	**1437**	**448**	**251**	**606**	**250**

As is shown here the total test costs of all services were 250 hours with an average of 36 hours per service.

1) Calendar Service

For the calendar it took 8 hours to test the service with 3 operations, 473 statements, 31 branches, 38 parameters, 12 function-points and 15 test-paths, meaning that this service was well within the time limitation of 40 hours. Within the same time, we could have tested 4 services of a similar size and complexity.

2) Order Entry Service

For the OrderEntry it took 37 hours to test the service with 16 operations, 625 statements, 187 branches, 43 parameters, 29 function-points and 92 test-paths, meaning that this service was just within the time limitation of 40 hours. Within the same time, we could not have tested any other services of a similar size and complexity.

3) Bausparer Service

For the BauSparer it took 13 hours to test the service with 17 operations, 276 statements, 47 branches, 64 parameters, 35 function-points and 22 test-paths, meaning that this service was well within the time limitation of 40 hours. It would have been possible to test two additional services of a similar size and complexity.

4) BeautySalon Service

For the BeautySalon it took 21hours to test the service with 24 operations, 429 statements, 72 branches, 54 parameters, 18 function-points and 33 test-paths, meaning that this service was equally well within the time limitation of 40 hours, but it would not have been possible to test an additional service of a similar size and complexity without proscribing overtime.

5) Geometry Service

For the Geometry exercise it took 18 hours to test the service with 5 operations, 510 statements, 73 branches, 19 parameters, 9 function-points and 36 test-paths, meaning that this service was also well within the time limitation of 40 hours. It would have been possible to test an additional service of a similar size and complexity.

6) Authorization Service

For the Authorization it took 65 hours to test the service with 27 operations, 573 statements, 265 branches, 19 parameters, 22 function-points and 130 test-paths, meaning that this service was beyond the time limitation of 40 hours. It would be necessary to either lower the test coverage criteria by 60% or to split it up into two services to be able to test it in a single sprint. This is where agile testing requires compromises.

7) Mailing Service

For the Mailing it took 88 hours to test the service with 48 operations, 3317 statements, 762 branches, 211 parameters,126 function-points and 278 test-paths, meaning that this service required more than double the time limitation of 40 hours. Here it was out of the question to lower the test coverage. The service has to be cut up into two. This is where the time limitations of agile testing conflict with the current functionality.

3. Conclusions of the Benchmark Test

The quintessence of the study was that most microservices, at least those contained in this experiment, can be tested within the time limitations imposed by an agile scrum project. The time limitation of 40 hours can be transposed into circa 96 test cases, i.e. test paths, at the rate of 2.4 test cases per tester hour. The developers who cut the services out of existing code must see to it that the services are self-contained and do not require more than 96 test cases to be adequately tested. According to the analysis of the Java code for billing medical costs for pensioners

in Austria there were 2.745.224 lines of code, 1.094.684 statements, 262389 branches and 124.364 possible execution paths. By dividing the number of execution paths, i.e. procedural test cases, by the average test productivity taken from the bench mark study = 2.4 the total tester hours was calculated to be 51.818 tester hours or 2.590 tester days. Of course, the test coverage criteria had to be lowered, so that in the end less than 1.940 tester days or 97 tester months were actually required. This only goes to show how costly testing can become, even in agile testing, if the test coverage goals are set too high. The service test of the health insurance services had cost 2.912 tester days before it was finally terminated with a branch coverage of 83%.

The most important conclusion is that microservices should not have more than 96 procedural test cases if they are to be tested within one week's time. That amounts to an average of 8.8 statements per test case or a maximum of 845 statements per service.

4. References

[Beck01] Beck, K. et al.: "Manifest for Agile Software Development", agilemanifesto.org/ iso.de, 2001

[Black99] Black, Rex: "Managing the Testing Process", Microsoft Press, Redmond, 1999

[Linz13] Linz, T.: "Testen in Scrum-Projekten", dpunkt.verlag, Heidelberg, 2013

[Boeh03] Boehm, B./ Turner, R.: "Balancing Agility and Discipline – A Guide for the Perplexed", Addison-Wesley, Reading, Ma., 2003

[Boeh99] Boehm, B. et al: "Software Cost Estimation with COCOMO-II", Prentice-Hall, Upper Saddle River, New Jersey, 1999

[CrGr09] Crispin, L. / Gregory, J.: "Agile Testing – A practical Guide for Testers and agile Teams", Addison-Wesley-Longman, Amsterdam, 2009