ASQT 2020 Virtual Conference

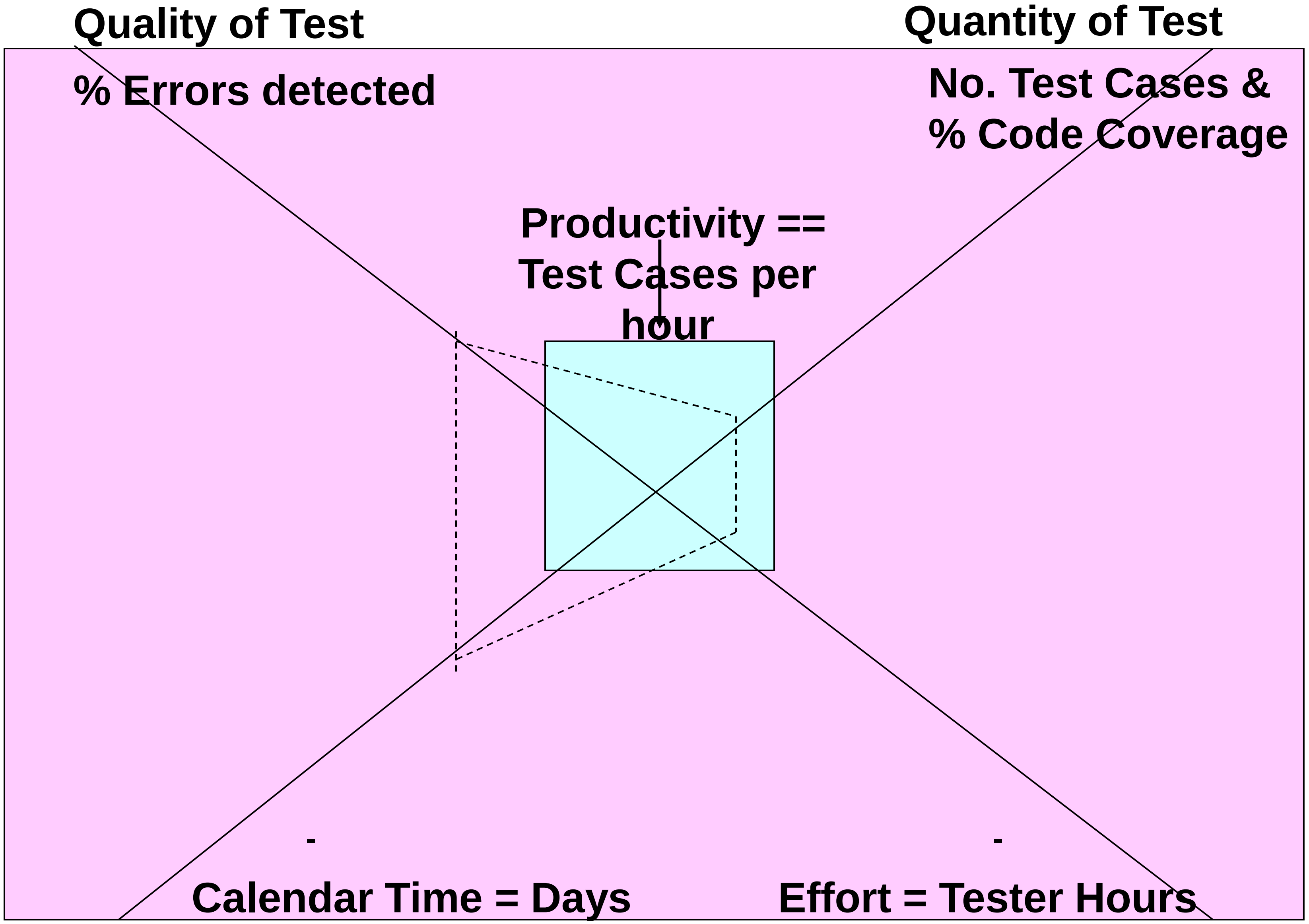
Estimating the Costs of Testing Microservices in an Agile Project

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

November, 2020

The Devil's Quadrant

(Sneed in SW-Mgmnt 1988)



Explanation of the Devil‘s Quadrant

1 The Productivity of a project group is given for a particular period of time. It may grow from year to year but hardly ever more than 15% per annum.

2 The Devil‘s Quadrant is based on a quadratic equation with four variables and one constant. The constant is the Productivity. The four variables are:

- ➔ **Quality,**
- ➔ **Quantity,**
- ➔ **Time,**
- ➔ **Effort.**

3 Two of the four variables can be determined, the other two are dependent. Quality is a multiplication factor ranging from a low bound of 0.2 to an upper bound of 1.8 for the highest attainable quality.

4 If Time and Effort are given, then Quantity and Quality are derived from the relation of Time and Effort to Productivity. It only remains to determine which functionality should be provided with what quality. This is referred to as backwards planning.

5 If Quantity and Quality are given, then Time and Effort are derived from the relation of Quantity and Quality to Productivity. This is the method used in algorithmic estimations. It is forward planning.

$$\text{Effort} = \frac{\text{Quantity x Quality}}{\text{Productivity}} \qquad \text{Time} = \frac{\text{Effort}}{\text{Persons ** 0.7}}$$

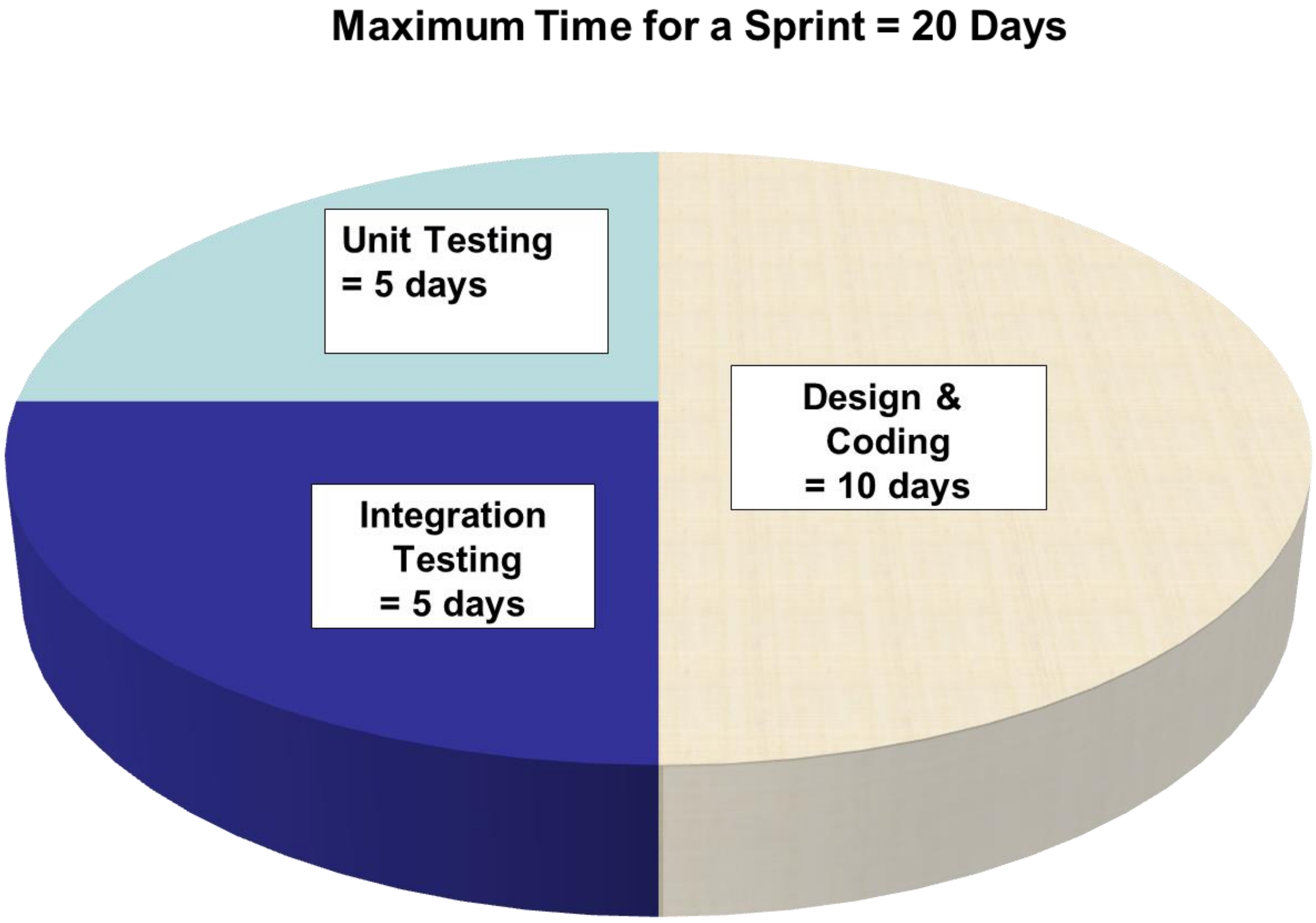
$$\text{Quality} = \frac{\text{Effort}}{\text{Quantity/Productivity}} \qquad \text{Quantity} = \text{Effort x Productivity x (2 – Qual)}$$

Planning the Service Test

- Agile Service Testing is a backward planning process
- The extent of the test is determined by the time available
- Planners must first define the time and effort allowed for a single sprint or new release
- Agile Scrum projects are governed by a release cycle of maximum 4 weeks [Beck01].
- From this the testers can deduce the amount of code they can test in this time and to that cost.
- Every sprint should deliver a new release, i.e., a new microservice with a test coverage of at least 90% branch = C1 coverage.

- 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.
- Of the two weeks for testing one week will be for unit testing and one week for integration testing [Crispin&Gregory09]
- The challenge is to fit the amount of test cases to the time and cost allowed

Test Effort in an agile Projekt



Adapting Service Size and Complexity to the Time and Cost Limits

- Services delivered must fulfill the test goals within the time & cost limits.
- The maximum size and complexity of a 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 [Boehm99]:

$$\text{Effort} = (\text{Size} * \text{Complexity}) / \text{Productivity}$$

- If the service is being developed new, it should be designed to remain within these limits from the beginning.

- If the service is being cut out of existing code it should be cut so that no single Service exceeds the size limit.

Test Cases for Calendar Service

```
service: DayofWeek
   if (testcase = „DayofWeek_TC01"); // German
       if ( operation =   "GetWeekDay");
           if ( request  =   "GetWeekDayInput");
               assert in.P1-DATE = „12101977";
               assert in.P2-LANGUAGE = “1“;
               assert in.P3-ALIGNMENT = “L“;
           endRequest ;
           if ( response  = "GetWeekDayOutput");
               assert out.P4-DAYNAME = „Mittwoch";
           endResponse ;
       endOperation;
   endCase;

 if (testcase = „DayofWeek_TC02");
       if ( operation =   "GetWeekDay"); //French
           if ( request  =   "GetWeekDayInput");
               assert in.P1-DATE = „12101977";
               assert in.P2-LANGUAGE = “2“;
               assert in.P3-ALIGNMENT = “L“;
           endRequest ;
           if ( response  =  "GetWeekdayOutput");
               assert out.P4- DAYNAME = “Mercredi";
           endResponse ;
       endOperation;
   endCase;
```

```
if (testcase = „DayofWeek_TC03"); // Italian
       if ( operation =   "GetWeekDay");
           if ( request  =   "GetWeekDayInput");
               assert in.P1-DATE = „12101977";
               assert in.P2-LANGUAGE = “3“;
               assert in.P3-ALIGNMENT = “L“;
           endRequest ;
           if ( response  = "GetWeekdayOutput");
               assert out.P4- DAYNAME = “Mercoldi";
           endResponse ;
       endOperation;
   endCase;

 if (testcase = „DayofWeek_TC04"); //English
       if ( operation =   "GetWeekDay");
           if ( request  =   "GetWeekDayInput");
               assert in.P1-DATE = „11312000";
               assert in.P2-LANGUAGE = “4“;
               assert in.P3-ALIGNMENT = “R“;
           endRequest ;
           if ( response  =  "GetWeekdayOutput");
               assert out.P4- DAYNAME = “Unknown";
           endResponse ;
       endOperation;
   endCase;
end; // service DayofWeek
```

Detecting Errors in Service Testing

- It is not enough just to test random test cases with the hope of taking a new path thru the service each time.
- The test must be measured as to what branches and paths are actually traversed.
- A minimum test coverage must be defined for every service tested.

For this experiment 7 Java microservices with varying sizes were tested with 4 different coverage measurements:

- Statement coverage
- Branch coverage
- Path coverage
- Parameter coverage [Crispin&Gregory09]

Errors are detected primarily by comparing the results returned with the expected results. If an actual result differs from an expected result, the tester is expected to take a closer look at the test case. If it turns out to be a real error the tester records and weighs it.

Reengineered Java Services

The Java services used for the benchmark test were:

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 calendar service had been converted from an earlier Assembler date routine.

The order entry service was cut out of a former COBOL application converted to Java.

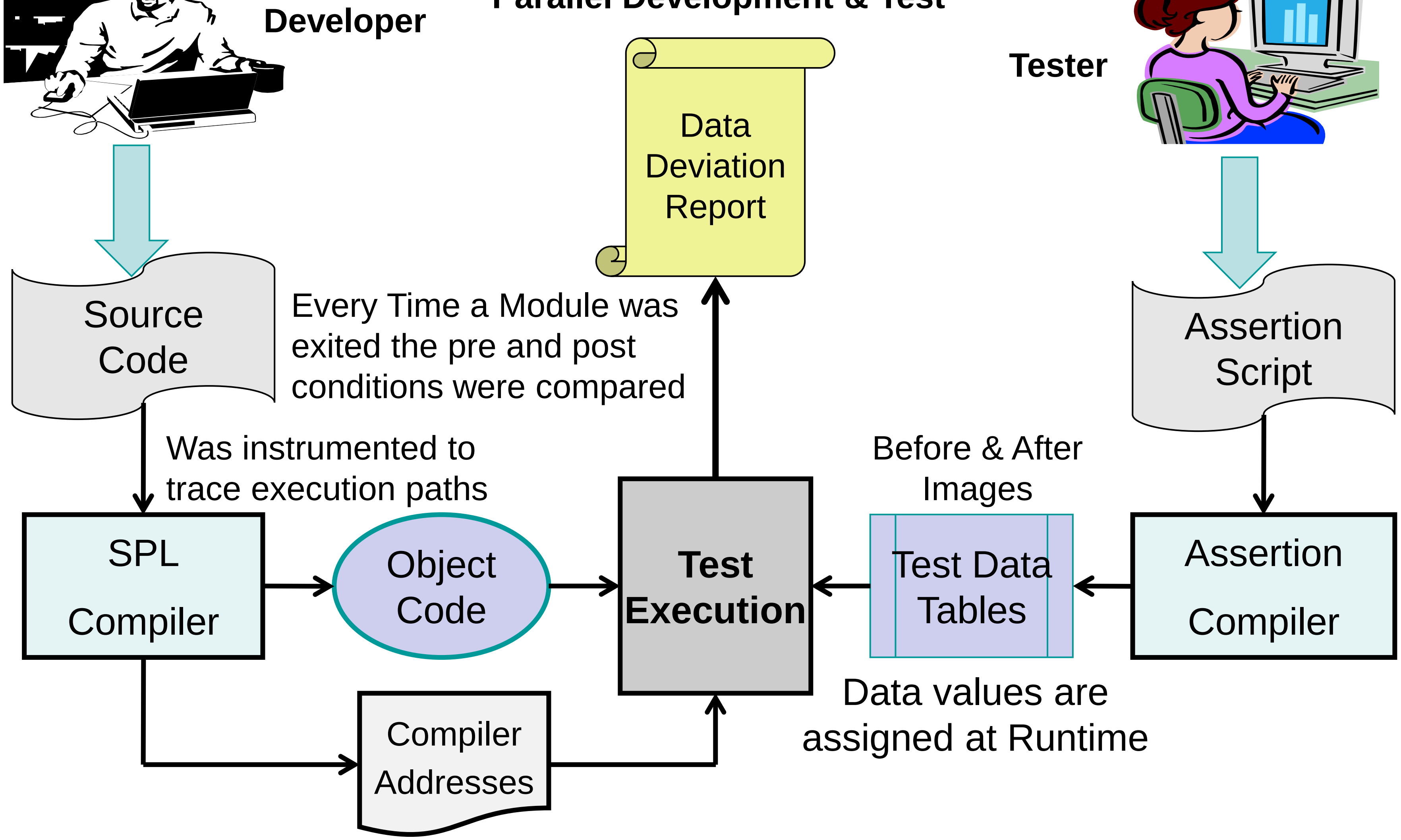
The geometric service was converted from C++.

The other services were original Java services, reengineered for wrapping.

Services were tested separately by mocking their environment in a service testbed.

Testing SPL Modules in 1978

Prüfstand compared the View of the Tester with the View of the Developer


Developer
Parallel Development & Test
Tester
Data Deviation Report
Source Code
Every Time a Module was exited the pre and post conditions were compared
Assertion Script
Was instrumented to trace execution paths
Before & After Images
SPL Compiler
Object Code
Test Execution
Test Data Tables
Assertion Compiler
Data values are assigned at Runtime
Compiler Addresses

Charging Test Services

The big Question was how should the testing service be charged. Siemens was not about to pay by time. The German managers did not trust the Hungarians.

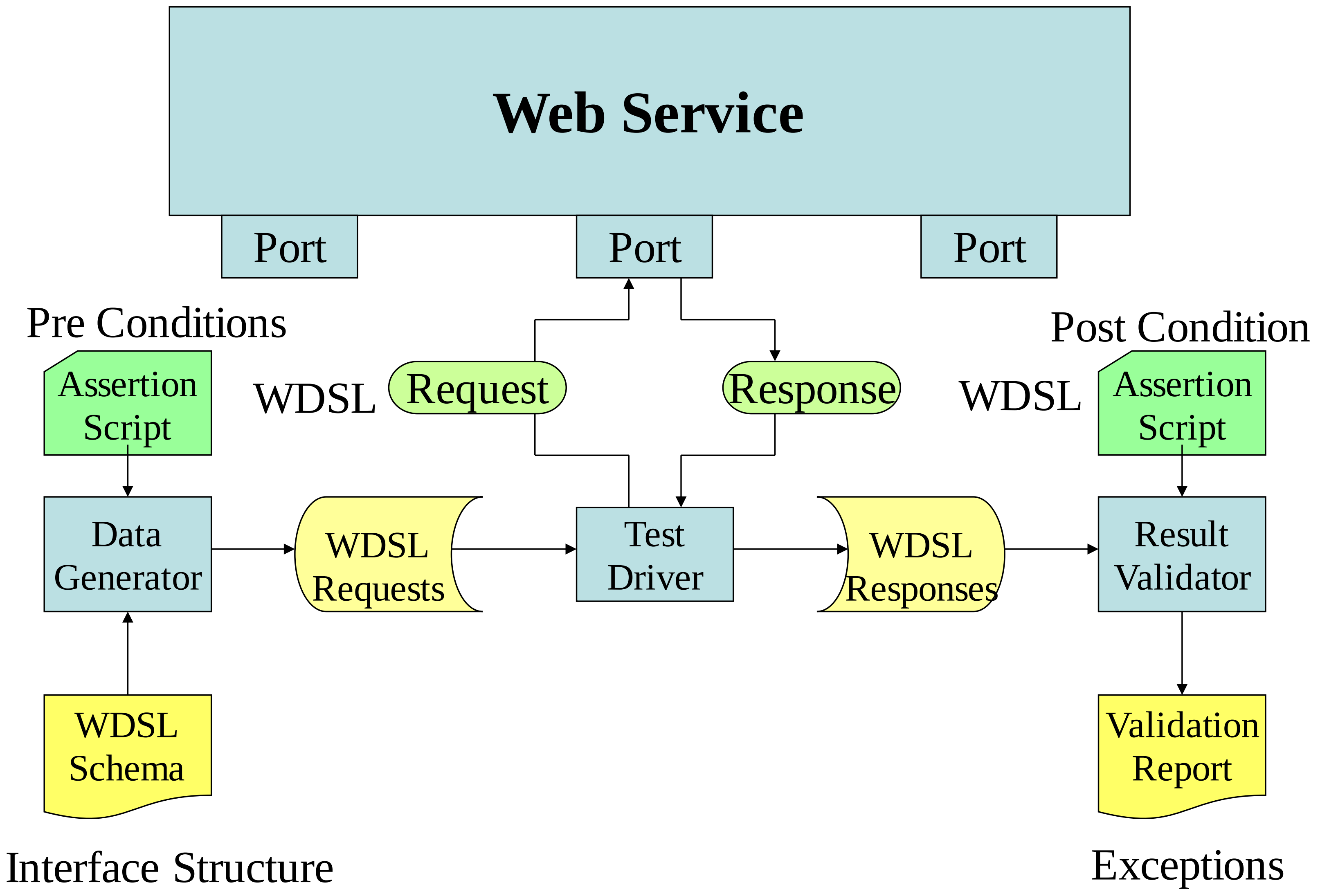
- Sneed suggested that they pay for finding errors and increasing their trust in the software.
- Errors reported is measurable, it can be expressed in absolute numbers. The errors can also be weighted by severity.
- Trust is not so easy to measure. Ed Miller proposed using test coverage in connection with the number of test cases tested and the number of errors found.
- To this end the test laboratory charged Euro 150,- for each proven error reported and Euro 40,- for each test case tested. Each test case had to be a unique path through the code.
- The paths executed, the code coverage and the errors found were documented and reported to the customer every month.

Metrics of the SPL Module Test in 1978

Comp	Modul es	Stmts	Test cases	Bran ches	Cover age	Defis	Errors	Test Days	Test Hours
A	6	4029	196	183	89%	138	5	130	780
B	37	7588	232	603	91%	130	22	162	972
K	71	40735	1064	2843	87.5%	868	143	380	2280
N	6	2847	101	140	94%	110	14	34	204
S	8	5682	147	792	95%	150	6	54	324
Total	**128**	**60881**	**1544**	**4378**	**91.9%**	**1396**	**192**	**760**	**4560**

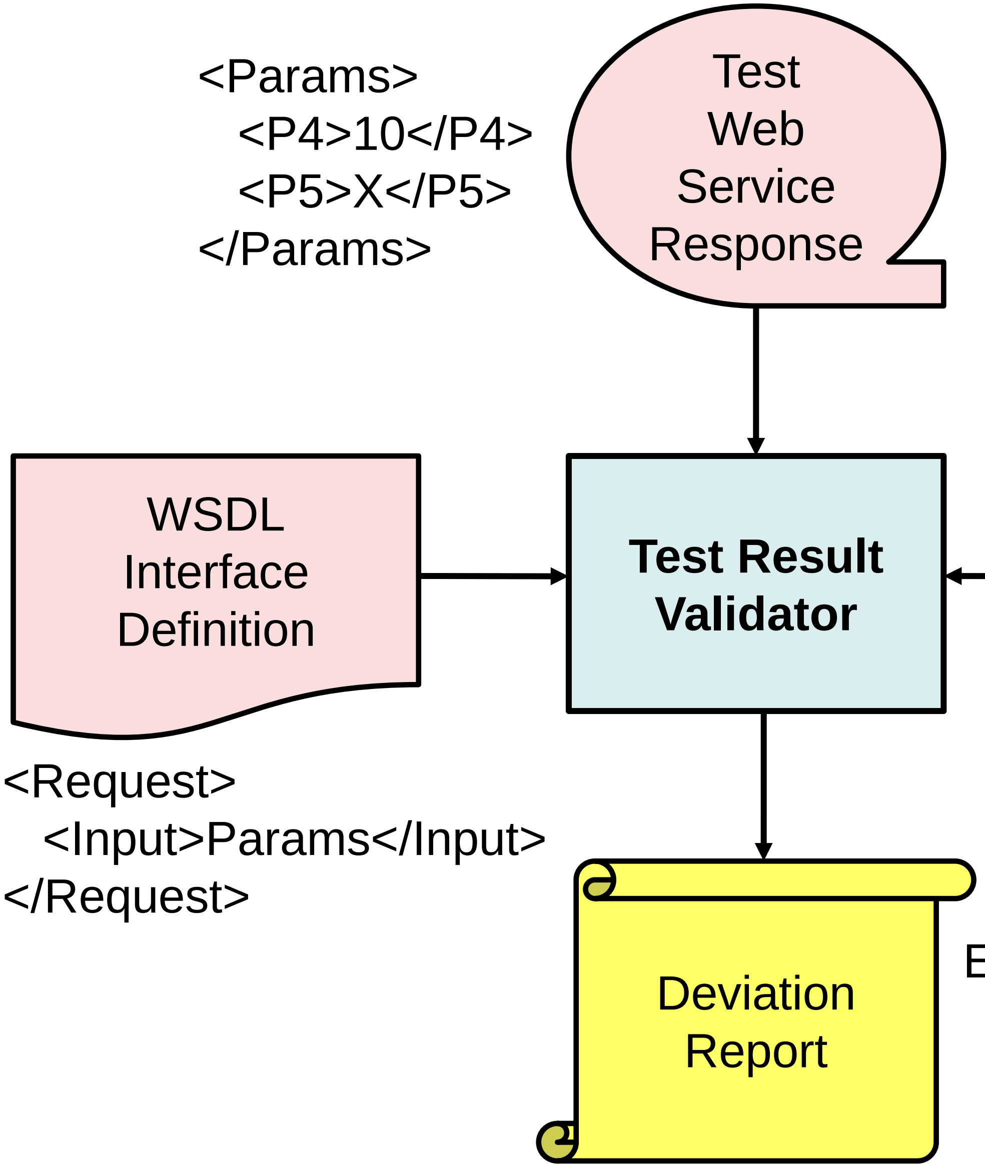
Test Productivity = 0.33 Test Cases per Hour or 3 Hours per Test Case
Test Efficiency = 0.0002 Errors per Hour or 24 Hours per Error

Microservice Test Execution



Microservice Response Validation

```
<Params>
   <P4>10</P4>
   <P5>X</P5>
</Params>
```



```
Assert out P4 = Range (10:20);
Assert out P5 = Set (“X“,  “Y“,  “Z“)
Assert out P6 = P1 + 7;
```

```
<Request>
   <Input>Params</Input>
</Request>
```

```
Exceptions
   P4 Target = Range (10:20)
   P4 Actual = 21
```

Metrics of the Java Service Test in 2020

Service	Opers	Stmts	Logic Branchs	Params	FuncPt	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

Test Productivity = 0.41 Test Cases per Hour or 2.4 Hours per Test Case

Experience with the SPL Test Labor

- In the first half year 128 modules with 60881 SPL statements passed through the test labor.
- 4378 code branches were tested with a coverage of 89.7% by 1544 test cases.
- The average path had **39** SPL statements.
- 192 program errors were discovered via dynamic analysis.
- 1396 design and code deficiencies were discovered via static analysis.
- The costs of the quality assurance remained below 2 Euro per statement. This was less than 10% of the total development costs.

Experience with the Java Service Test

- The most important conclusion is that a microservice should not have more than **48** procedural test cases if it is to be tested within one week’s time.
- The average path included **10.2** statements i.e. per test case.
- To be able to fit into the schedule of an agile test the service to be tested can not exceed the size of **490** statements.
- Of the 7 Java microservices 2 had to be refactored to be testable within a week – *Authorize* and *MailService.*
- Java microservices should be designed or reengineered to meet these criteria.

Interesting Observations

- The length of an average Java path is only ¼ of that of an average SPL path.
- The effort to test a test case in SPL is almost the same as that for a test case in Java.
- The key to calculating test effort is the number of paths and parameters to test.
- The key coverage metric is branch, i.e. decision coverage. Parameter coverage might be more appropriate for testing services. This has to be investigated.

References

- **[Beck01]** Beck, K. et al.: “Manifest for Agile Software Development”, agilemanifesto.org/ iso.de, 2001
- **[Boehm03]** Boehm, B. / Turner, R.: „Balancing Agility and Discipline – A Guide for the Perplexed“, Addison-Wesley, Reading, Ma., 2003
- **[Boehm99]** Boehm, B. et al.: „Software Cost Estimation with COCOMO-II“, Prentice-Hall, Upper Saddle River, New Jersey, 1999
- **[Crispin&Gregory09]** Crispin, L. / Gregory, J.: „Agile Testing – A practical Guide for Testers and agile Teams“, Addison-Wesley-Longman, Amsterdam, 2009.