Migrating to Service-oriented Systems
(Why and How to avoid developing customized software applications from scratch)

Harry M. Sneed &
Chris Verhoef

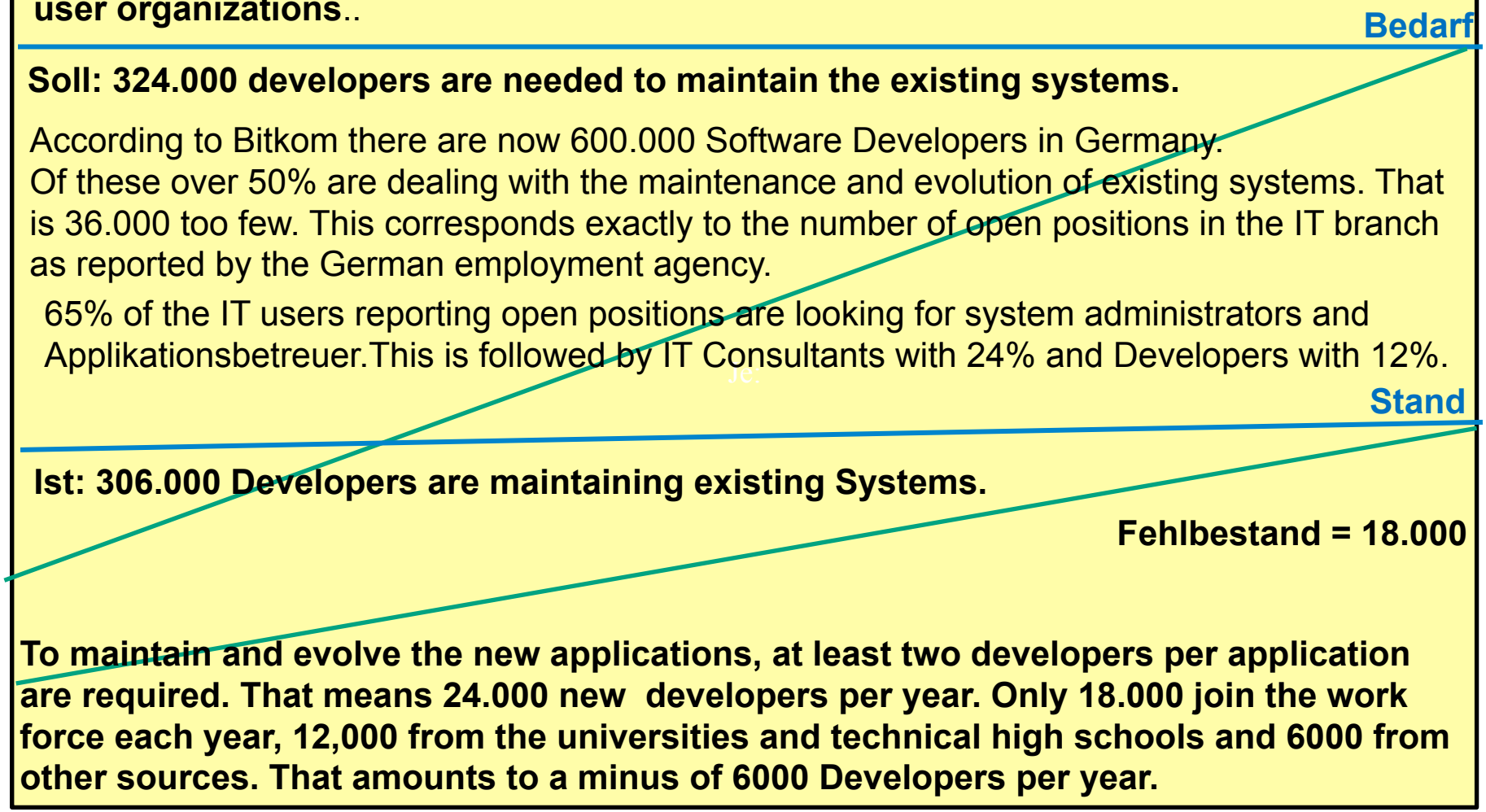
for

WSE-2013

Eindhoven

The Consequences of unconstrained individual Software Development

Every year more than 12,000 new applications are released by German user organizations..



Soll: 324.000 developers are needed to maintain the existing systems.

According to Bitkom there are now 600.000 Software Developers in Germany.
Of these over 50% are dealing with the maintenance and evolution of existing systems. That is 36.000 too few. This corresponds exactly to the number of open positions in the IT branch as reported by the German employment agency.

65% of the IT users reporting open positions are looking for system administrators and Applikationsbetreuer.This is followed by IT Consultants with 24% and Developers with 12%.

Ist: 306.000 Developers are maintaining existing Systems.

Fehlbestand = 18.000

To maintain and evolve the new applications, at least two developers per application are required. That means 24.000 new developers per year. Only 18.000 join the work force each year, 12,000 from the universities and technical high schools and 6000 from other sources. That amounts to a minus of 6000 Developers per year.

The Numbers from a typical Java Application

A Java Application in Numbers:

- 1.977 Source Members
- 270.729 genuine Code Lines without comments
- 183.474 Statements, including
  - 18.299 if statements
  - 152 switch statements
  - 1.591 loop statements
  - 7.071 exception handling statements
- 31 Components (Packages)
- 1.654 Classes
- 15.384 Methods
- 788 Interfaces
- 47 Database Tables
- 155 Database Accesses
- 129 Input Operations
- 420 Output Operations
- 6.866 Function-Points = 27 Statements per Function-Point

If this system changes at a rate of only 10% per year, then some 18.347 statements, or 686 Function-Points are impacted. With a productivity of 800 Statements, or 28 Function-Points, per Person month 23-24 Person months are needed per year to maintain and evolve the system. That means that two Java experts are bound by the preservation of this system.

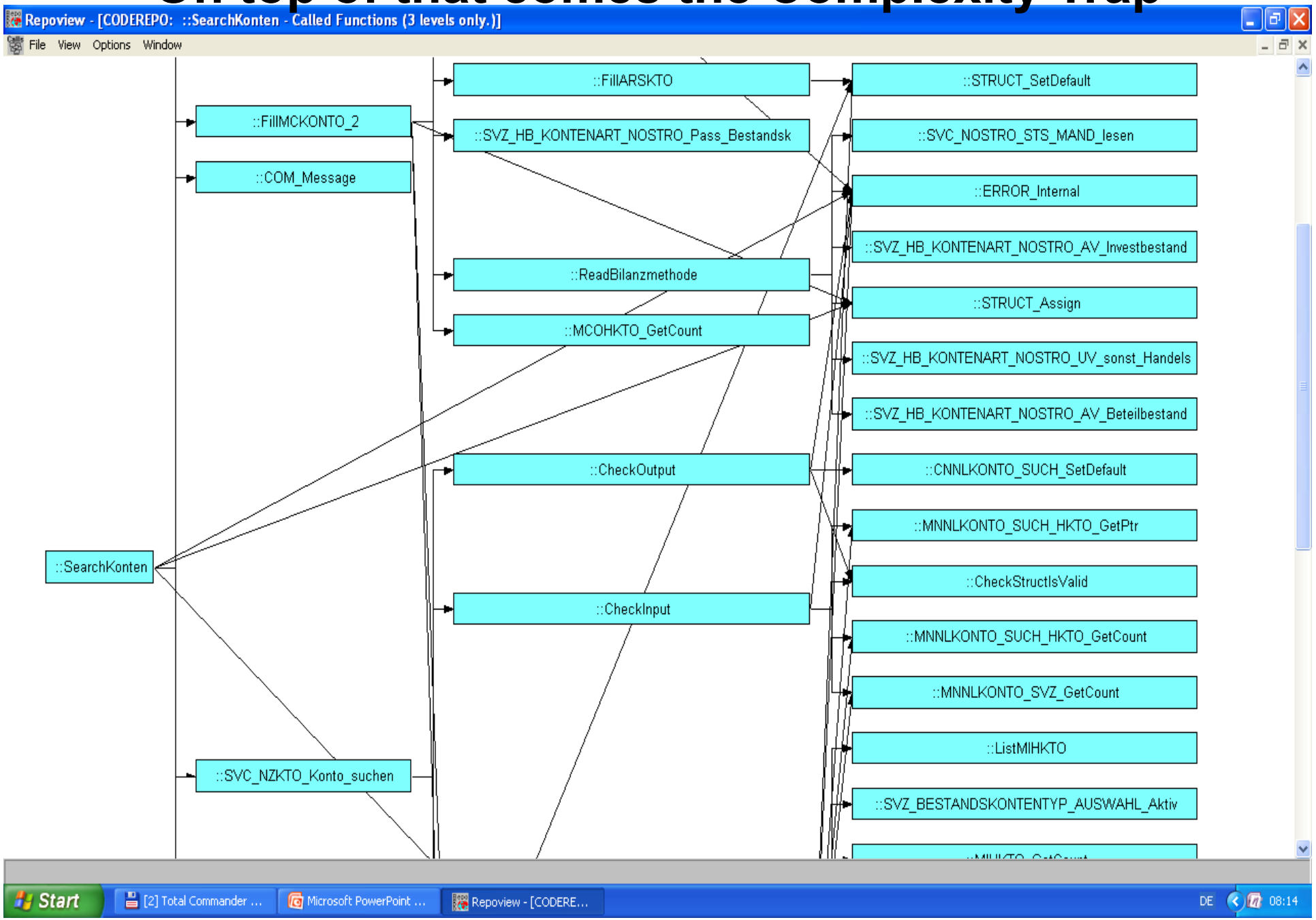
The Numbers from a typical DotNet Application

A DotNet Application in Numbers:

- 1.804 Source Members
- 388.576 genuine Code Lines, without comments
- 309.823 Statements, including:
  - 18.608 if statements
  - 474 switch statements
  - 3.119 loop statements
  - 5.434 exception handling statements
- 36 Components
- 2.606 Classes
- 16.832 Methods
- 492 Interfaces
- 204 Database Tables
- 716 Database Accesses
- 286 Input Operations
- 530 Output Operations
- 7.540 Function-Points = 41 Statements per Function-Point.

If this system changes at a rate of 10% per year, 30.982 statements, or 754 Function-Points, are impacted. With a productivity of 800 statements per Person month some 40 Person months per year are needed to keep the system going. That means that four DotNet experts are bound by the preservation of this system.

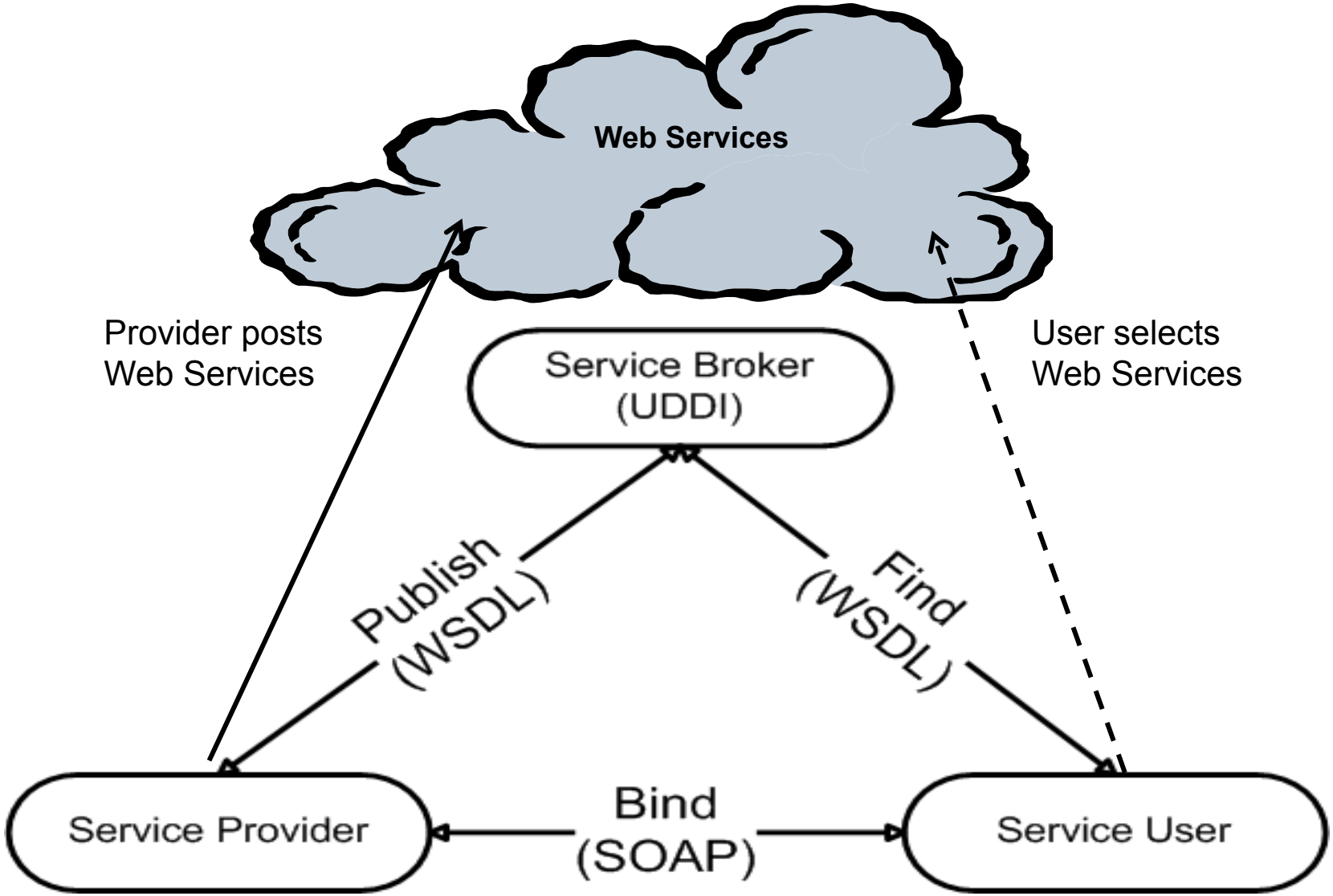
On top of that comes the Complexity Trap



And technical Debt

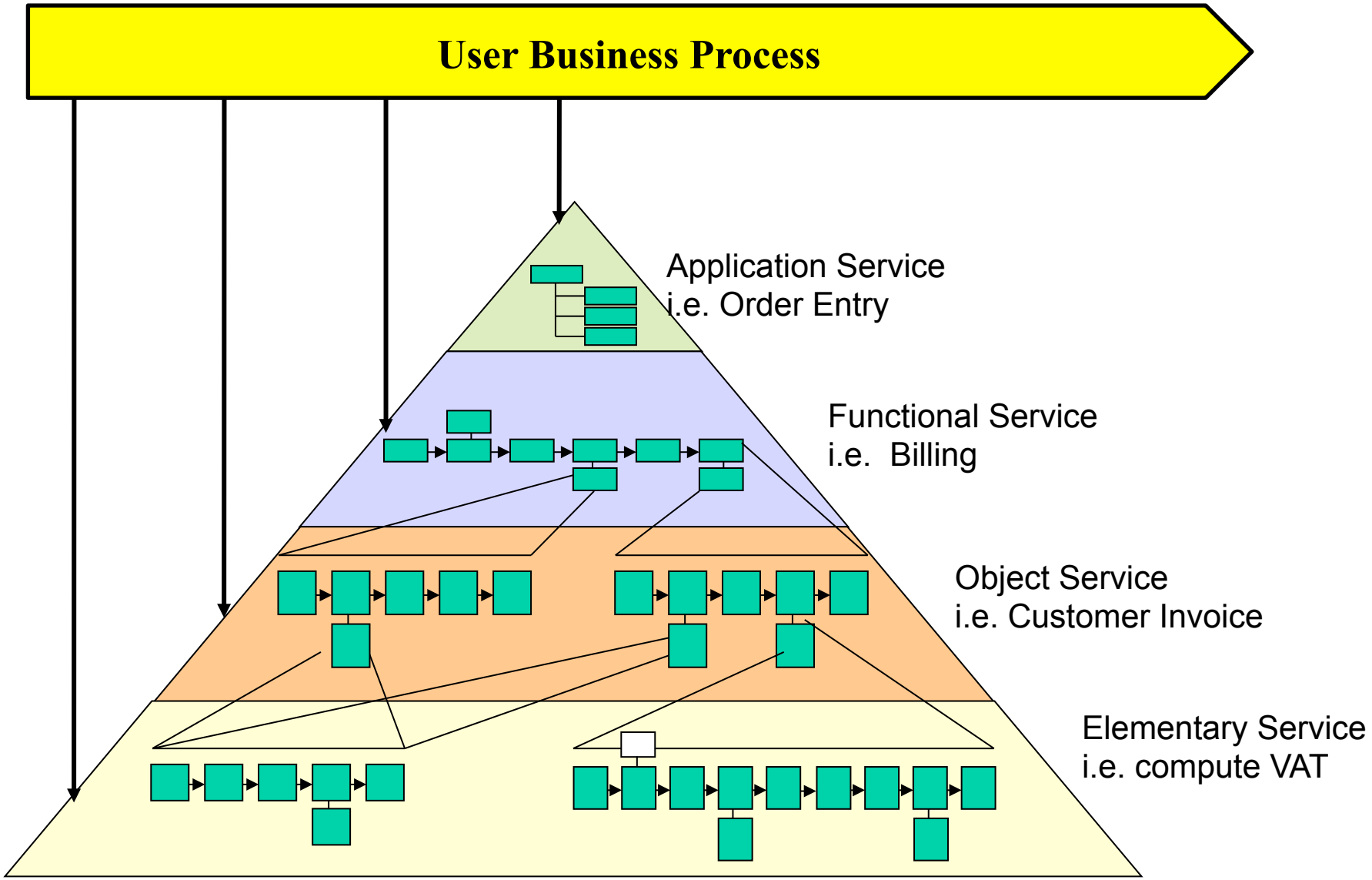
Major deficiences sorted by number of occurences	**Renovation Effort in Person Hours**	
(10) Return Value is not controlled after Method Call 2435	x 0,5	= 1217
(22) Public Variables should be avoided in Classes 2280	x 1	= 2280
(01) IO-Operations are not in a try block 913	x 1,5	= 1370
(25) Class is not derived from a Superordinate Class 219		
(14) Control logic exceeds maximum allowed nesting level 962	x 2	= 1924
(15) Missing Final clause in method declaration 856	x 0,5	= 428
(04) Data Casting should be avoided 692	x 2	= 1384
(18) Check on incoming public request is missing 535	x 4	= 2140
(26) Nested Classes are not allowed 376	x 4	= 1504
(27) Returning a Function my cause an endless loop 239	x 6	= 1434
(11) Conditions should not contain an Assignment 150	x 1	= 150
(17) Try and Catch clauses do not match 112	x 1	= 112
(13) Default is missing in Switch Statement 85	x 1	= 85
(12) Case block should contain a Break statement 77	x 2	= 154
(08) Method Invocation with Array is not in a try Block 31	x 3	= 93
(07) External Variables are not allowed 29	x 2	= 58
(06) Standard IO Functions are prohibited 21	x 4	= 84
(09) There should be no global Data Definitions in C# 5	x 2	= 10
(02) Two Dimensional Arrays violate 1. Normal Form 2	x 8	= 16
Total Renovation Effort required		= 1**4.443**

Standard, ready-made Services can be the Answer

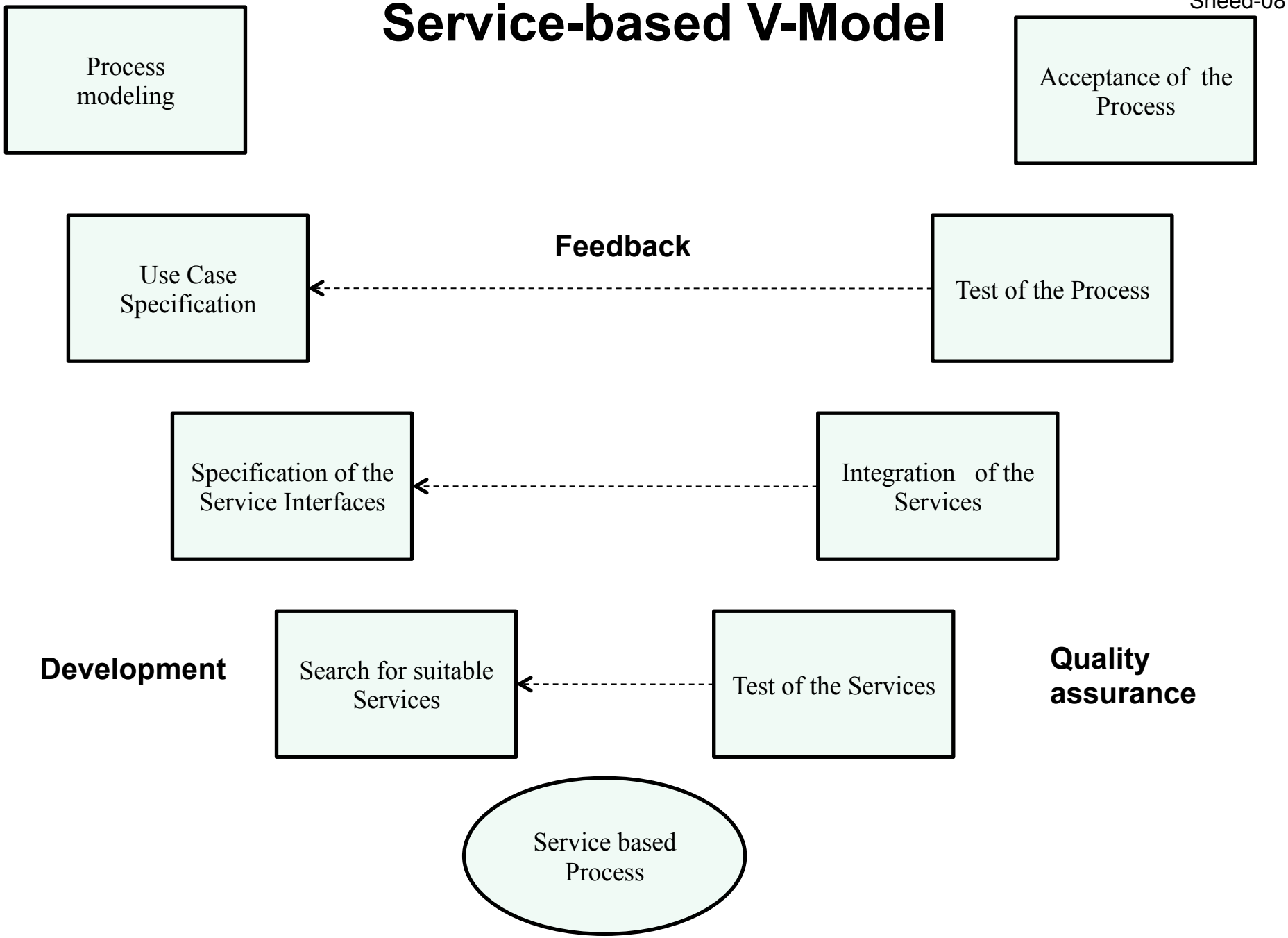


Service-oriented Architecture

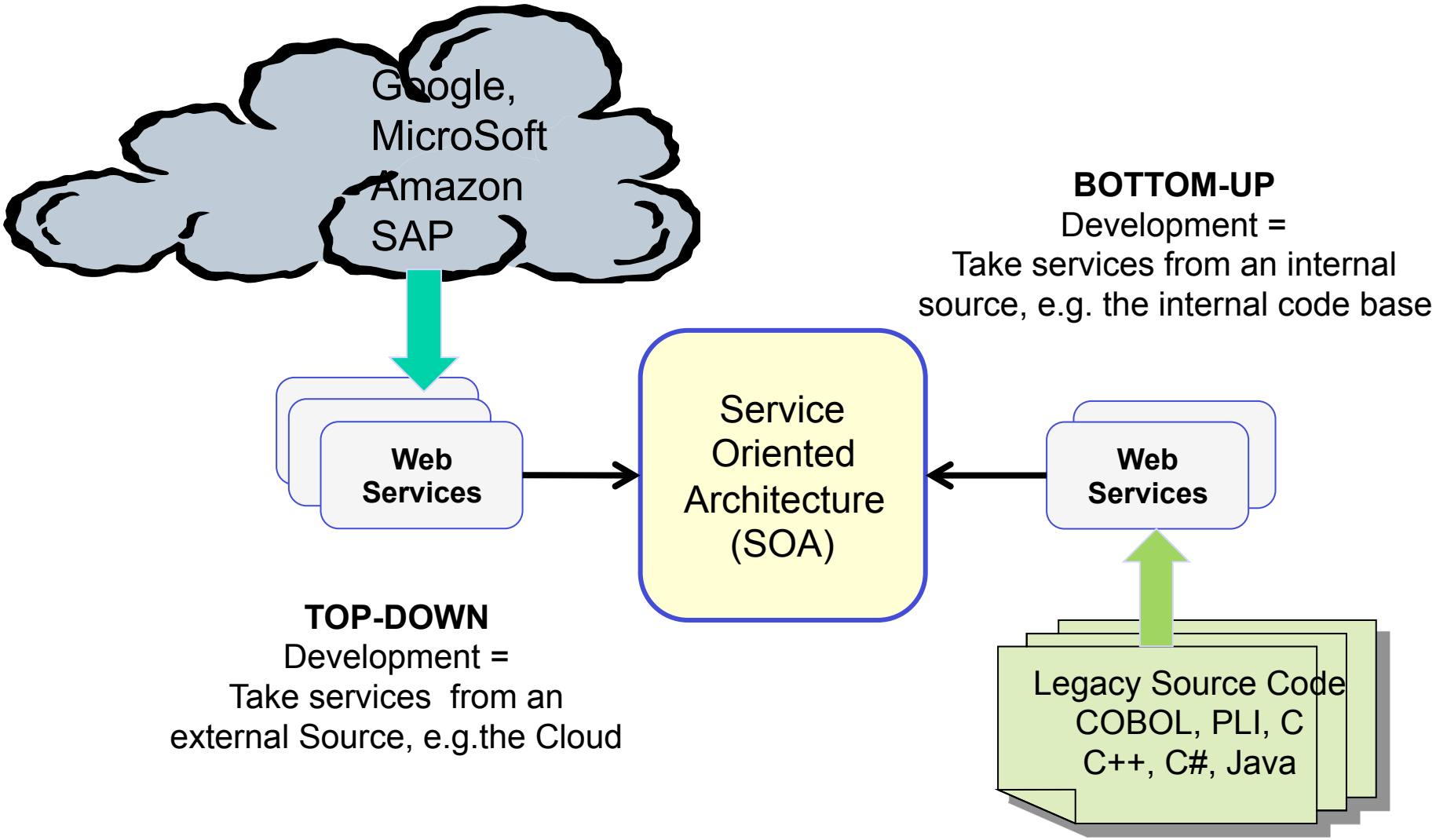
Services are obtained from a service-oriented Architecture (SOA)



Service-based V-Model



Two Paths to a SOA



BOTTOM-UP
Development =
Take services from an internal source, e.g. the internal code base

TOP-DOWN
Development =
Take services from an external Source, e.g.the Cloud

The Service-Oriented Approach

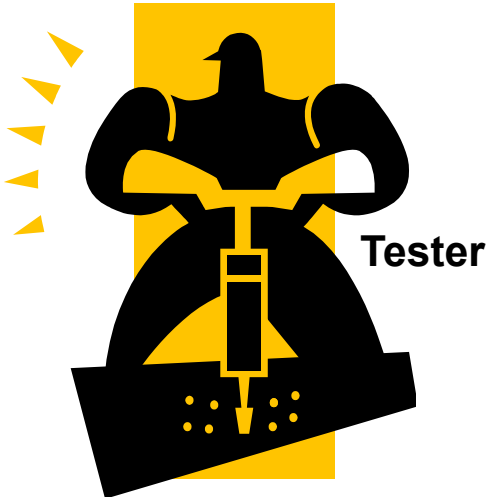
- Requirement Document is no longer the basis of a new development but a basis of comparison with potential services.
- Business Analyst does not ask User what he would like to have, but presents him instead with a list of potential services.
- The User selects those services which appear to fulfill his requirements.
- The Tester then tests those services to determine if they really satisfy the requirements.
- Those services which come closest to fulfilling the functional and non-functional requirements are finally selected.
- It may be necessary to wrap those services behind a functional wrapper which adapts their inputs and outputs.

Actors in a Service-Oriented Project

User

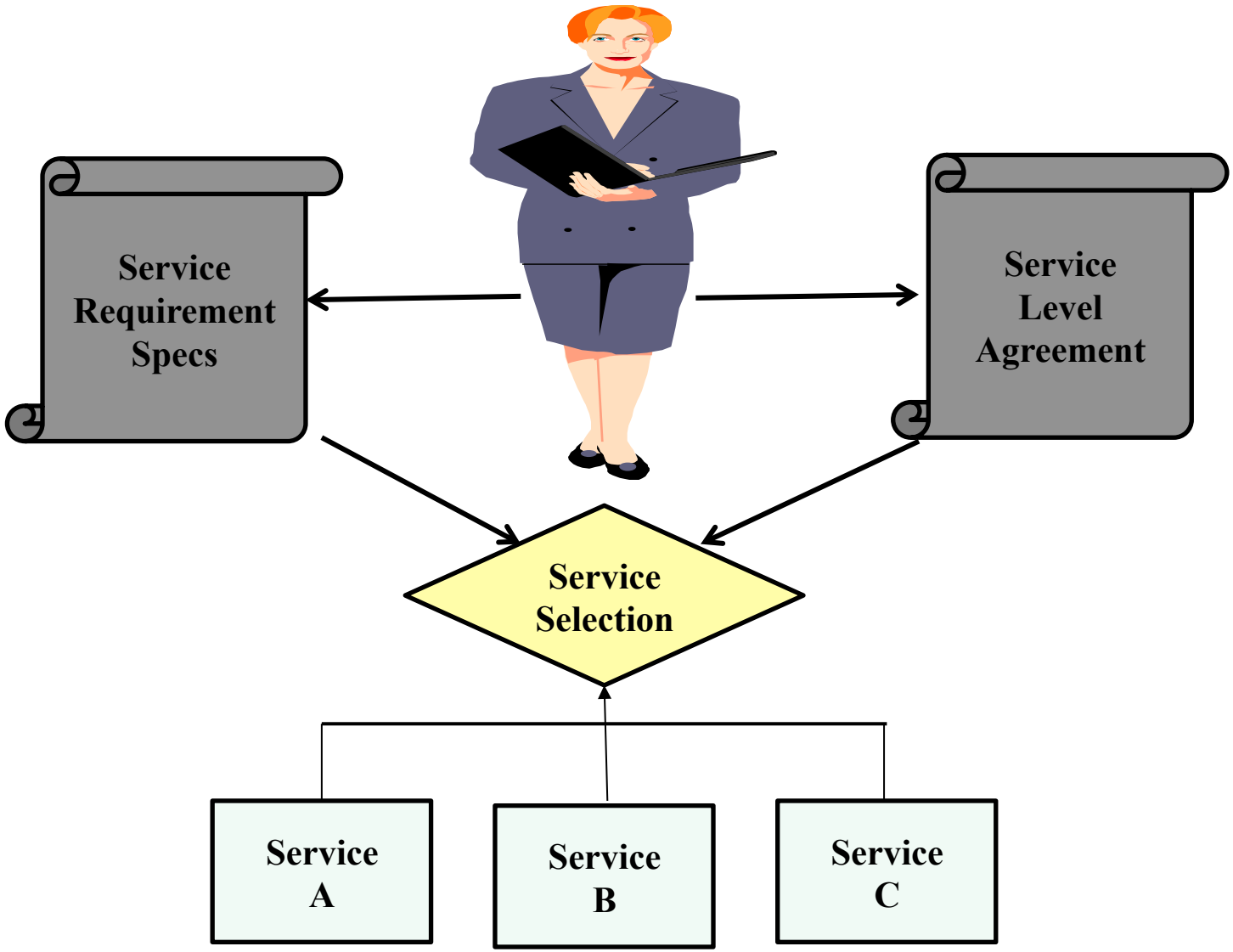
Analyst

Developer



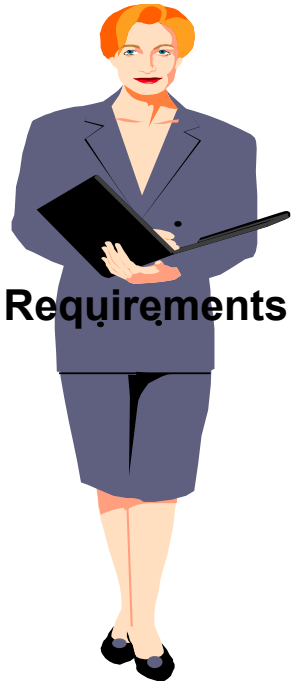
Tester

The Role of the User

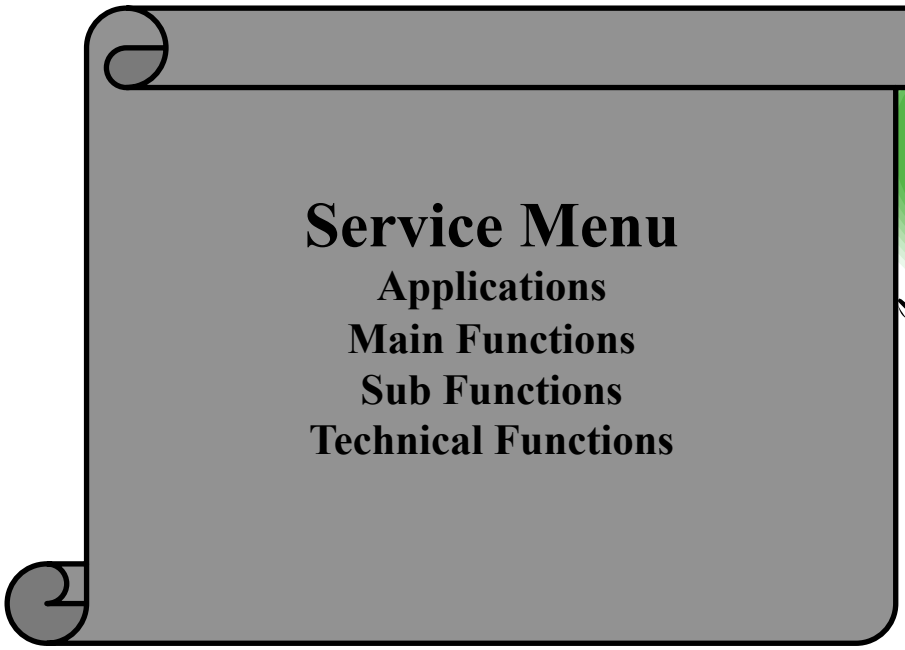


The Role of the Analyst

Gives the User a Choice of Services



Requirements

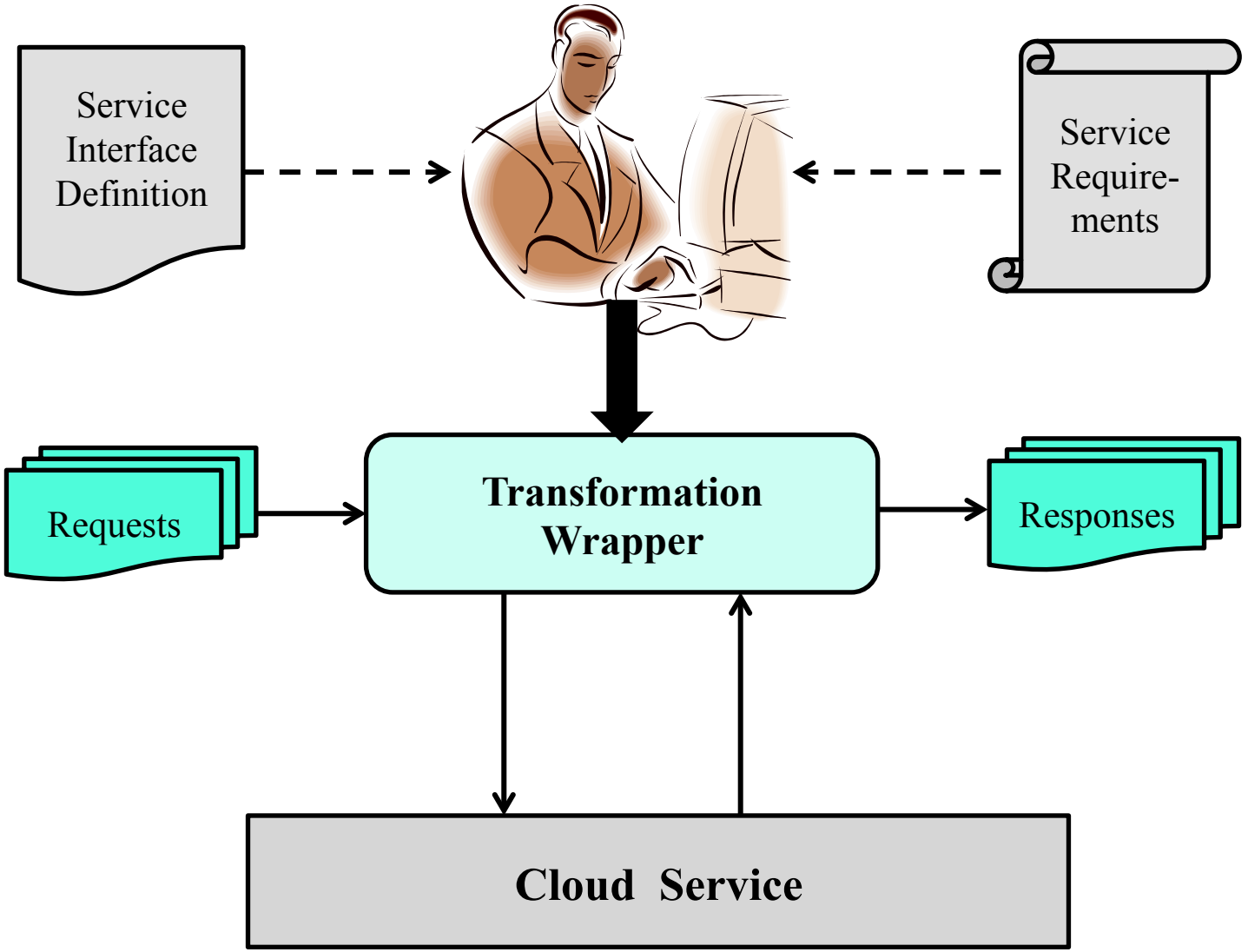


User

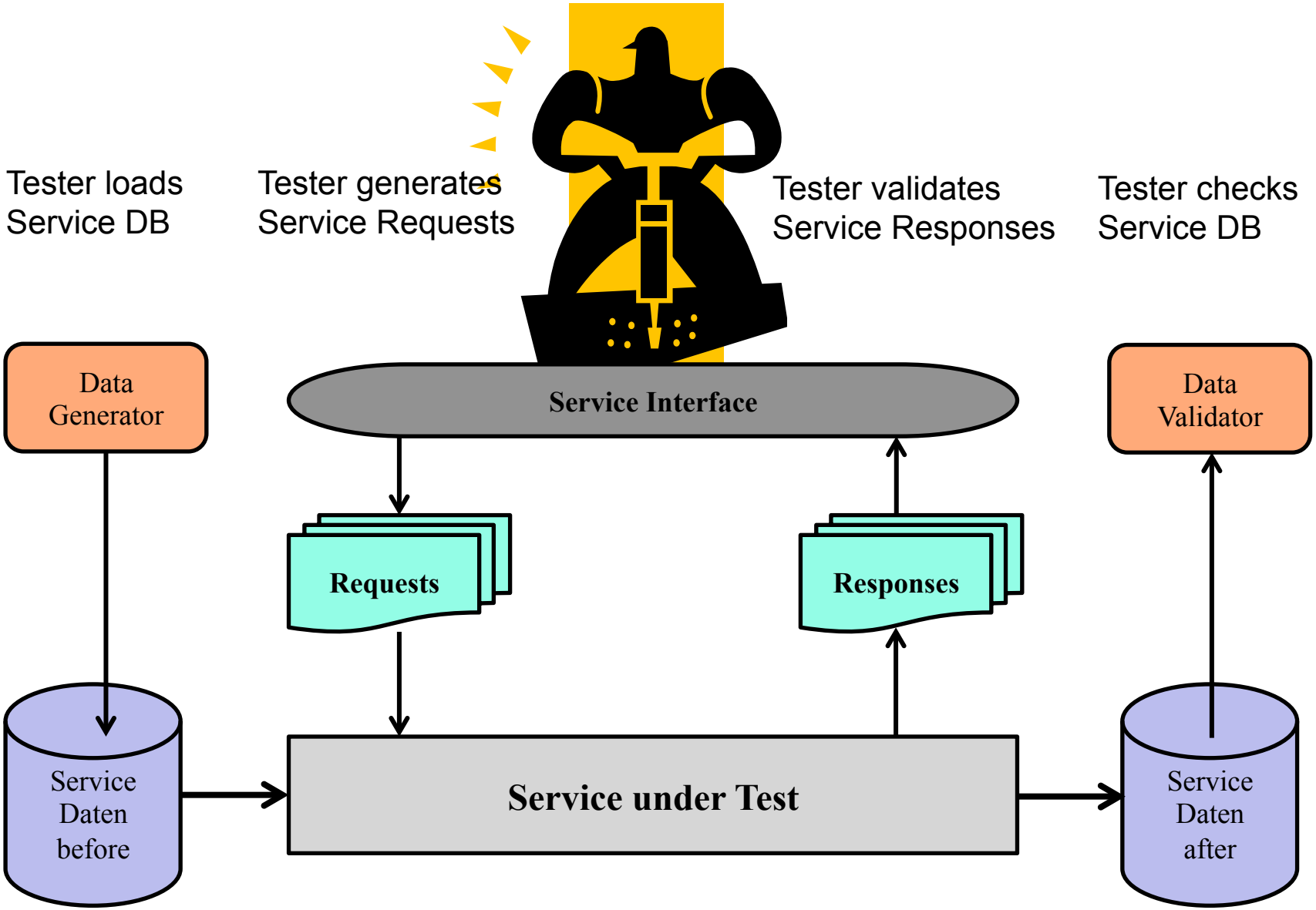
Business Analyst

User can choose from a wide range of Services at different Levels of Granularity

The Role of the Developer



The Role of the Tester



Paradigm Change must come

- Individual, customized software development has reached its limit. It cannot go on this way!
- Object-oriented applications continue to grow and become increasingly complex as the number of system elements increases and the number of dependencies between those elements increases even more.
- The number of applications is increasing at a rising rate of 16% per year.
- The number of qualified software developers is increasing at less than 10% annually.
- The it world has reached the limits of its capacity to produce new software while still preserving the existing software.
- A paradigm change is inevitable.