Towards Automated Testing of Web Service Choreographies

Felipe Besson, Pedro Leal,
Fabio Kon and Alfredo Goldman
University of São Paulo

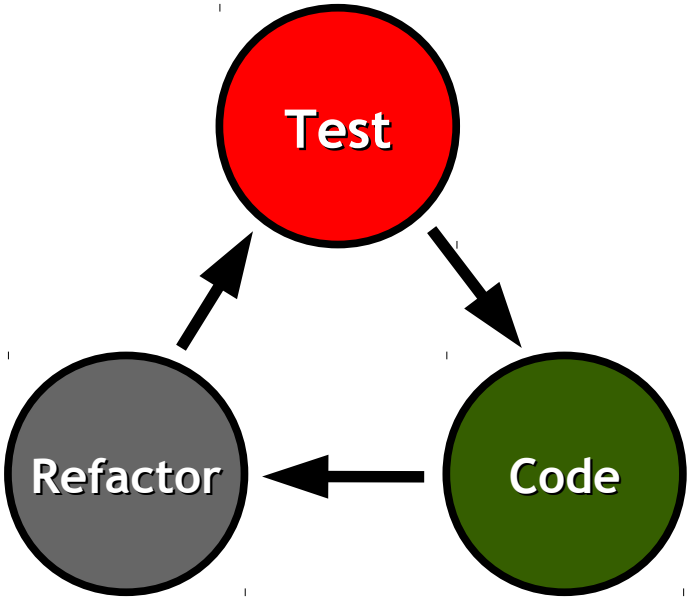
Dejan Milojicic
Hewlett Packard Laboratories

6th International
Workshop on
Automated
Software Testing

Contextualizing

Test-Driven Development (TDD)

A design technique that drives the development process through testing (Fowler, 2011; Beck, 2002):



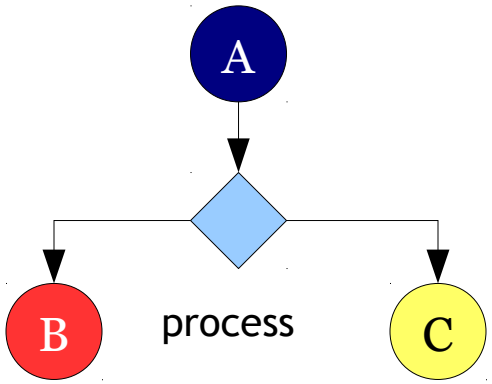
1. **Write an AUTOMATED test for next functionality you want to add;**
2. **Write the functional code until the test passes;**
3. **Refactor the new and old parts of the code.**

Contextualizing

Web Service Compositions

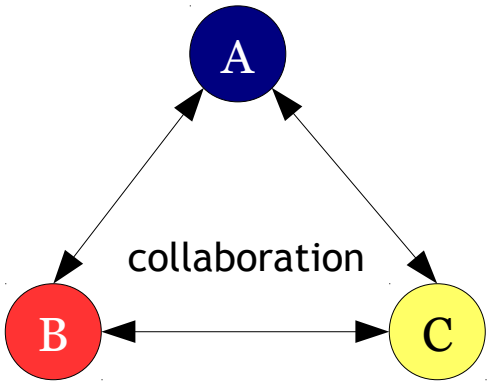
Simple internal and external (third-party) services can be composed in more complex ones:

Orchestration



- ✓ Simpler
- ✗ Not Scalable

Choreography



- ✗ Complex
- ✓ Decentralized
- ✓ Scalable

Contextualizing

Testing of Web Service Choreographies

In spite of the benefits of choreographies, the following issues:

- Decentralized flow of information
- Third-party and governance issues
- Dynamicity
- No widely-adopted standards

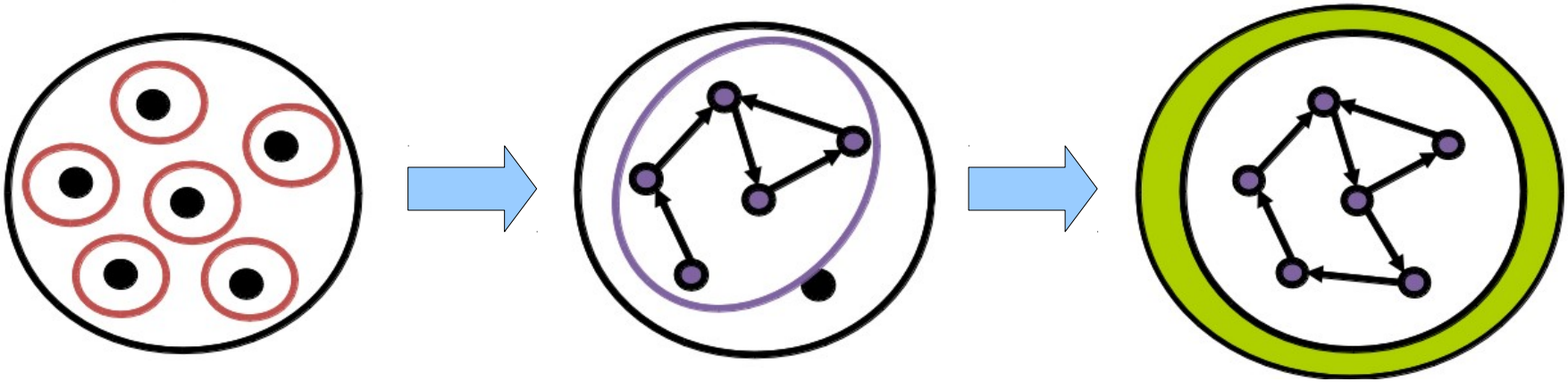
Make the testing of choreographies difficult!

Some approaches for testing choreographies (Bucchiarone, 2007; Canfora, 2009;Palacios, 2011) have been proposed

- they focus on the pre-execution of choreographies (e.g., models validation)
  - **None of them are related to the running choreographies**
    - » **Preventing TDD**

Goals

Develop a testing framework for supporting TDD of choreographies



Services isolated

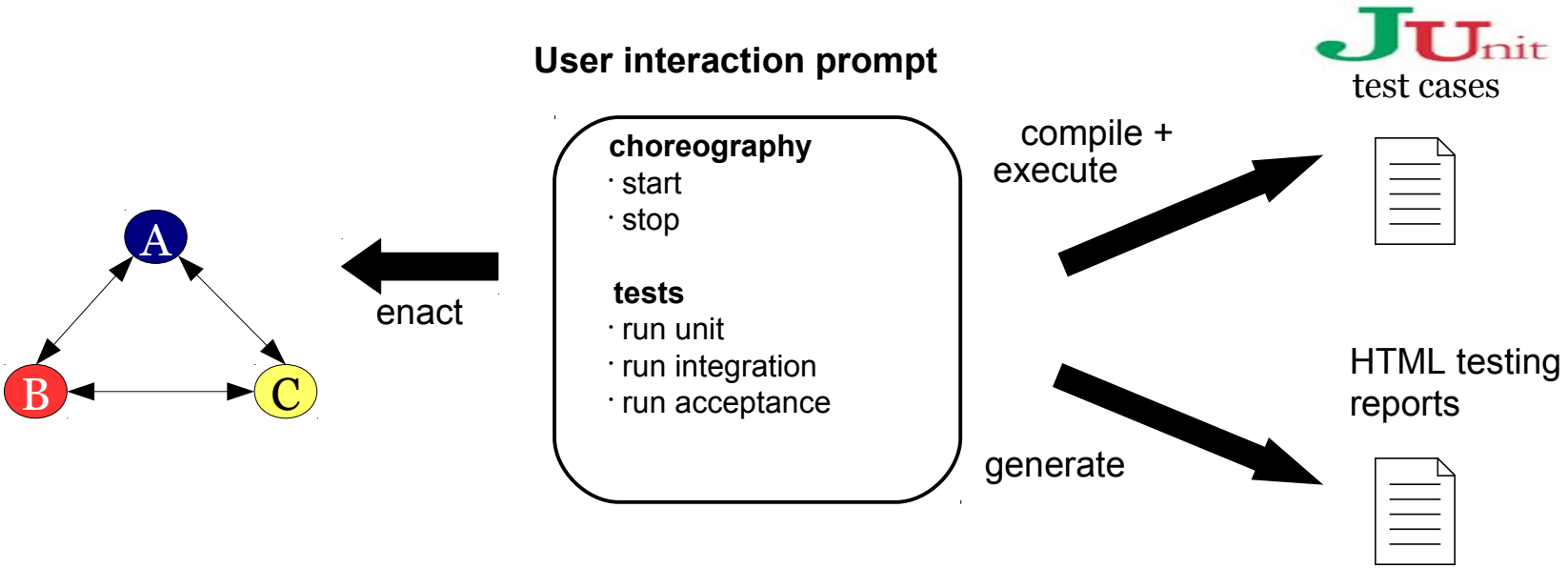
Messages exchanged by the services

The entire choreography

Into the framework

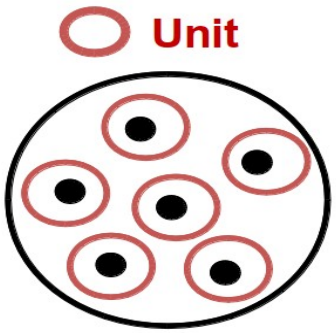
Our software prototype

- ***Ad hoc* bash scripts for a choreography enactment**
  - Book trip choreography using OpenKnowledge (OK, 2011)
- **A set of JUnit test cases for automated testing of this choreography**
  - Unit, Acceptance and Integration tests



Unit testing

Every operation of each service participating in the choreography is tested.

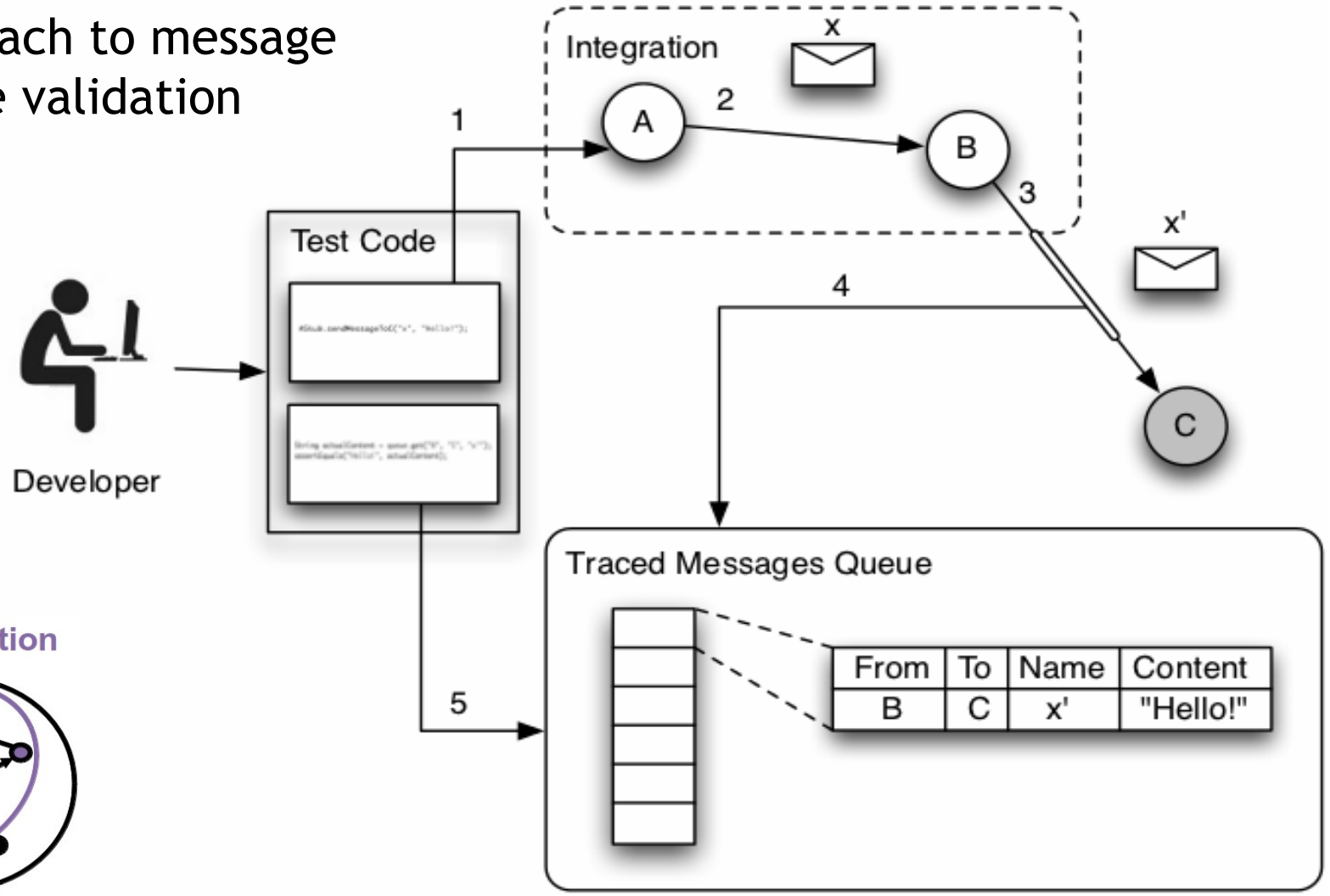


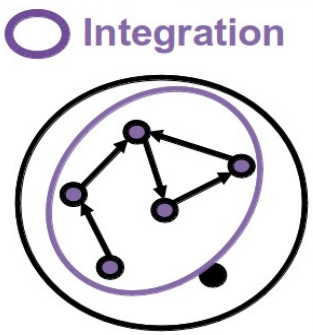
When all tests pass, the service is able to be integrated into the choreography

```
public class AirlineWSTest {

    private AirlineWSService service;
    private AirlineWS stub;

    final String TA_NAME = "Agile Travels";
    final String RESERVATION = "R3153-1|2000";
    final String USER = "John Locke";

    @BeforeClass
    public static void publishAirlineService() {
        Bash.deployService("airline");
    }
    ...

    @Test
    public void shouldFindFlight() {
        flight = stub.getFlight(destination, date);

        assertEquals("3153", flight.getId());
        assertEquals("Milan", flight.getDestination());
        assertEquals("12-21-2010", flight.getDate());
        assertEquals("09:15", flight.getTime());
    }

    @Test
    public void shouldBeAnAuthorizedTravelAgency() {
        assertTrue(stub.isTravelAgencyAuthorized(TA_NAME));
    }
```

Integration testing

An approach to message exchange validation





Acceptance testing

From the user perspective , the choreography is accessible as an atomic service.

Then, each test exercises an entire conversation as a unit.

```
@Test
public void shouldBookAndPlanTrip() {

    flight = stub.orderTrip("Paris",
                            "12-20-2010",
                            "John Locke",
                            "435067869");

    reservation = stub.reserveTicket(flight.getId());
    List<String> response = stub.book(reservation);

    statement = "Name: John Locke" + "\n" +
                "Credit card: 435067869" + "\n" +
                "Value discounted: $2100";

    eTicket = "e-ticket for flight " +
              flight.getId() + "\n" +
              "passenger: John Locke";

    assertTrue(response.contains(eTicket));
    assertTrue(response.contains(statement));

}
```

Ongoing work

We are extending our prototype by providing features for:

- **Generating web service clients dynamically**
  - From the URI, all operations can be invoked dynamically
- **Manipulating the elements of a choreography more easily**
  - Roles, services, messages are manipulated through Java objects
- **Mocking third-party services**
  - All web service operations can be mocked
- **Improving the interception of exchanged messages**
  - Providing better mechanism for intercepting, and then, collecting the name and the content of the messages exchanged among the services

Questions?

This research is funded by:

More information on: **http://ccsl.ime.usp.br/baile/VandV**

Felipe M. Besson
besson@ime.usp.br

References

Martin Fowler. Test-Driven Development. 2011. Available on: http://www.martinfowler.com/bliki/TestDrivenDevelopment.html

Kent Beck. Test Driven Development: By Example. Addison-Wesley Professional, 2002.

Gerardo Canfora and Massimiliano Di Penta. Service-oriented architectures testing: A survey. In Andrea De Lucia and Filomena Ferrucci, editors, Software Engineering, volume 5413 of Lecture Notes in Computer Science, pages 78–105. Springer Berlin / Heidelberg, 2009.

A. Bucchiarone, H. Melgratti and F. Severoni. Testing service composition. In 8th Argentine Symposium on Software Engineering (ASSE'07), Mar del Plata, Argentina, 2007.

Marcos Palacios, José García-Fanjul e Javier Tuya. Testing in service oriented architectures with dynamic binding: A mapping study. Inf. Softw. Technol., March 2011.

OpenKnowledge (OK). 2011. Available on: <http://www.openk.org>