MASTER IN COMPUTER SCIENCE

Moldable scenario editor

Master thesis final presentation, Bern University F2020

Ivan Kravchenko

Supervised by Nitish Patkar, Andrei Chiş and Nataliia Stulova

UNIVERSITÄT BERN

Problem

How can business idea be reflected in documentation and code?

- Code is often poorly described
- *Idea to implementation* flow spans through many tools and instruments
- Ubiquitous language is hard to develop and manage for a variety of business requirements

Agile idea

Iterative business process

Small iterations

Many interested process participants

Testing helps to assert that implementation is not outdated and still conforms to the requirements.

Tests need an environment

Tests on stage/test

Production

Difficult to manage and repeat

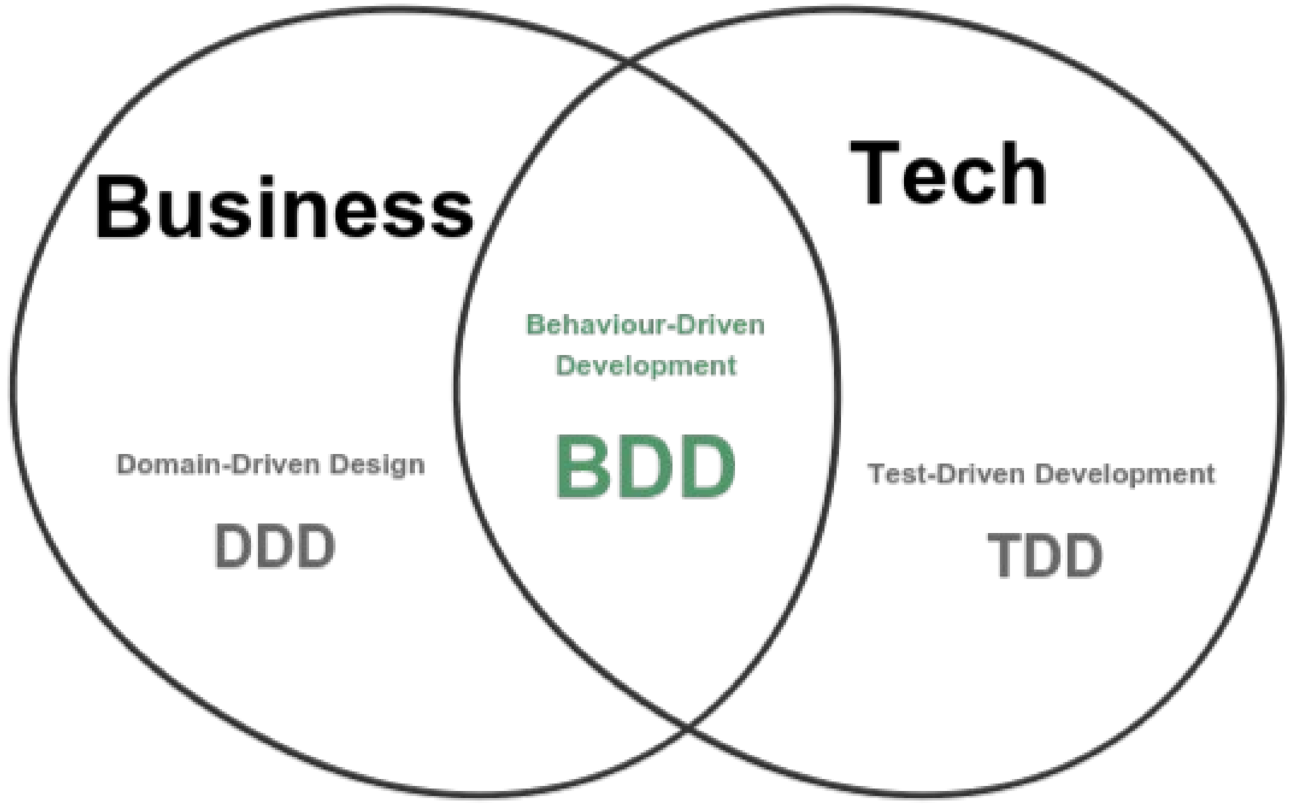
Tests as specifications

Executable specification

Code

Executable anytime, easily repeatable

BDD


Business
Tech
Behaviour-Driven Development
BDD
Domain-Driven Design
DDD
Test-Driven Development
TDD

Research question 1

What are the limitations of the
BDD tools regarding software behavior specification?

We analyzed 14 BDD tools :

Cucumber, FitNesse, Jbehave, Concordion, SpecFlow, Spock, Rspec, Mspec, LightBDD, ScalaTest, Specs2, Jgiven, phpspec, Gauge

Demo
Cucumber

Main observations

1. Input and Output are strictly defined
2. Two ways of writing specifications:
   - plain text language and 'glue code’
   - code with annotations
3. Test output is pass or fail; best case – customized report (colors, charts, coverage)
4. Providing objects as an input in a scenario is not very common
5. Code generation is not possible
6. No tools provide a graphical representation of specifications

Research question 2

How can we closely couple specification and implementation and what advantages does it give over existing tools?

Unite documentation and implementation

User-friendly scenario editing

Code generation

Demo

Results

- ✓ Connect requirements and behavior
- ✓ Allow users experiment with properties
- ✓ Generate code based on user input
- ✓ Scenario editor and creator

Future work

Reduce test development time by writing less ‘glue’ code

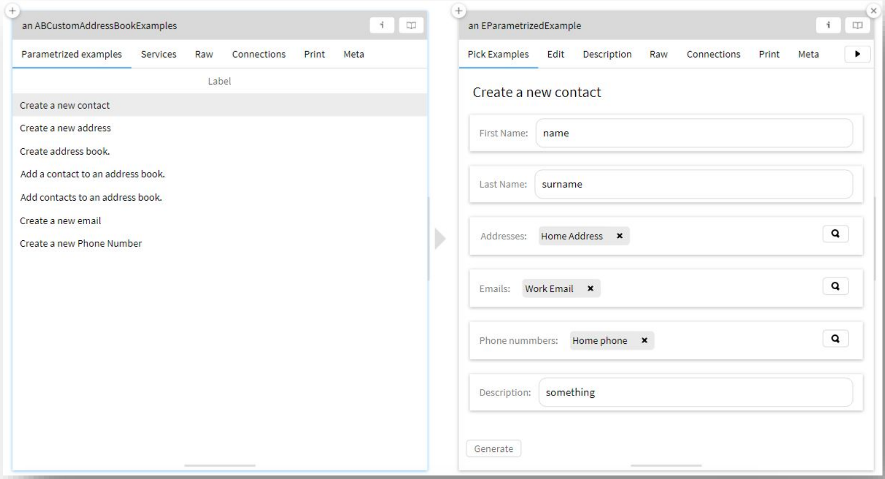
Think of unified way of defining ubiquitous language

Composable parameterized examples

Visual UI improvements

Default object views for widget elements

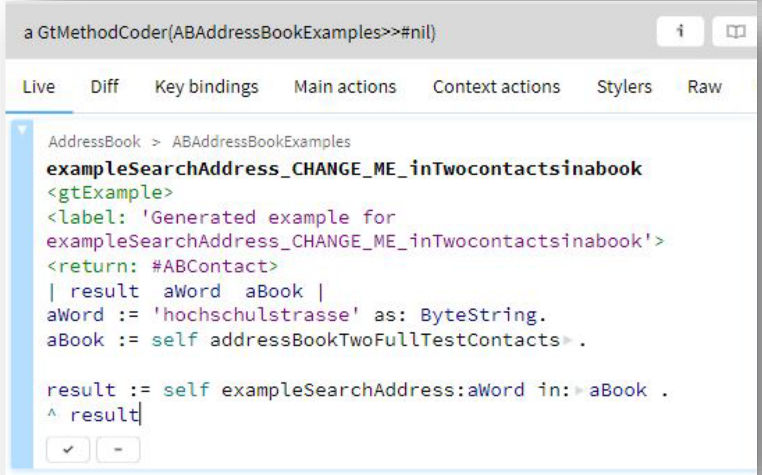
Summary



Create a new contact
Create a new address
Create address book.
Add a contact to an address book.
Add contacts to an address book.
Create a new email
Create a new Phone Number

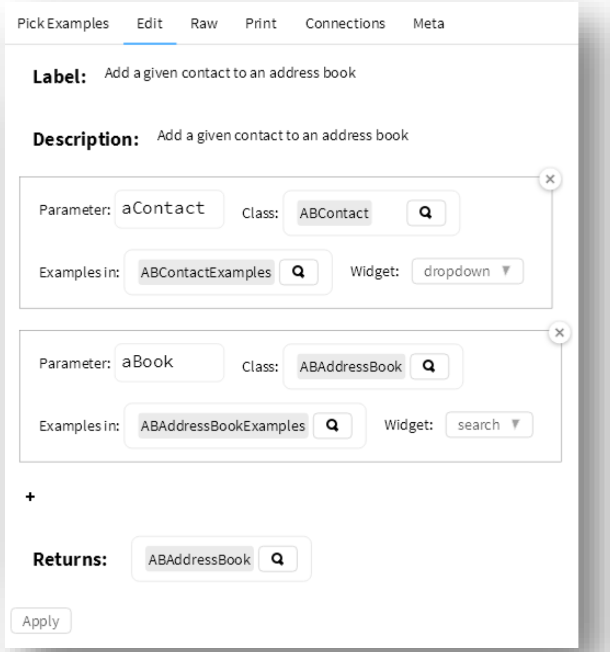
Create a new contact

First Name:	name
Last Name:	surname
Addresses:	Home Address ×
Emails:	Work Email ×
Phone nummbers:	Home phone ×
Description:	something



```
AddressBook > ABAddressBookExamples
exampleSearchAddress_CHANGE_ME_inTwocontactsinabook
<gtExample>
<label: 'Generated example for
exampleSearchAddress_CHANGE_ME_inTwocontactsinabook'>
<return: #ABContact>
| result  aWord  aBook |
aWord := 'hochschulstrasse' as: ByteString.
aBook := self addressBookTwoFullTestContacts .

result := self exampleSearchAddress:aWord in: aBook .
^ result
```



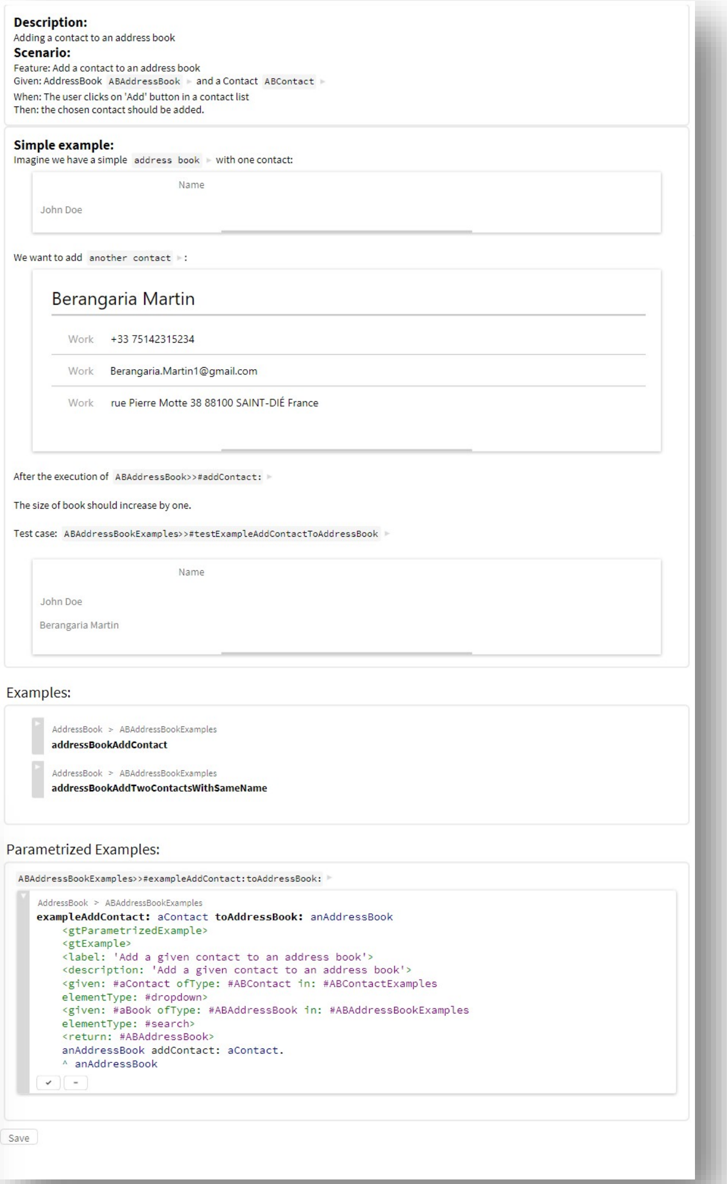
Label: Add a given contact to an address book

Description: Add a given contact to an address book

Parameter: aContact Class: ABContact
Examples in: ABContactExamples Widget: dropdown

Parameter: aBook Class: ABAddressBook
Examples in: ABAddressBookExamples Widget: search

Returns: ABAddressBook



Description:
Adding a contact to an address book
Scenario:
Feature: Add a contact to an address book
Given: AddressBook `ABAddressBook` and a Contact `ABContact`
When: The user clicks on 'Add' button in a contact list
Then: the chosen contact should be added.

Simple example:
Imagine we have a simple `address book` with one contact:

Name
John Doe

We want to add `another contact` :

Berangaria Martin

Work	+33 75142315234
Work	Berangaria.Martin1@gmail.com
Work	rue Pierre Motte 38 88100 SAINT-DIÉ France

After the execution of `ABAddressBook>>#addContact:`

The size of book should increase by one.

Test case: `ABAddressBookExamples>>#testExampleAddContactToAddressBook`

Name
John Doe
Berangaria Martin

Examples:

AddressBook > ABAddressBookExamples
addressBookAddContact

AddressBook > ABAddressBookExamples
addressBookAddTwoContactsWithSameName

Parametrized Examples:

`ABAddressBookExamples>>#exampleAddContact:toAddressBook:`

```
AddressBook > ABAddressBookExamples
exampleAddContact: aContact toAddressBook: anAddressBook
    <gtParametrizedExample>
    <gtExample>
    <label: 'Add a given contact to an address book'>
    <description: 'Add a given contact to an address book'>
    <given: #aContact ofType: #ABContact in: #ABContactExamples
    elementType: #dropdown>
    <given: #aBook ofType: #ABAddressBook in: #ABAddressBookExamples
    elementType: #search>
    <return: #ABAddressBook>
    anAddressBook addContact: aContact.
    ^ anAddressBook
```

Main observations

1. Input and Output are strictly defined
2. Two ways of writing specifications:
   - plain text language and 'glue code'
   - code with annotations
3. Test output is pass or fail; best case – customized report (colors, charts, coverage)
4. Providing objects as an input in a scenario is not very common
5. Code generation is not possible
6. No tools provide a graphical representation of specifications