

Eiffel Testing Framework (ETF): Acceptance Tests via Abstract User Interface



EECS3311: Software Design
Fall 2017

CHEN-WEI WANG

Bank ATM

The ATM application has a variety of *concrete* user interfaces.



Separation of Concerns

- The (Concrete) User Interface
 - The executable of your application *hides* the implementing classes and features.
 - Users typically interact with your application via some GUI. e.g., web app, mobile app, or desktop app
- The *Business Logic (Model)*
 - When you develop your application software, you implement classes and features. e.g., How the bank stores, processes, retrieves information about accounts and transactions

In practice:

- You need to test your software as if it were a real app *way before* dedicating to the design of an actual GUI.
- The model should be *independent* of the View, Input and Output.

Prototyping System with Abstract UI

- For you to quickly prototype a working system, you do not need to spend time on developing a fancy GUI.
- The *Eiffel Testing Framework (ETF)* allows you to:
 - Focus on developing the business model;
 - Test your business model as if it were a real app.
- In ETF, observable interactions with the application GUI (e.g., “button clicks”) are **abstracted** as monitored events.

Events	Features
interactions	computations
external	internal
observable	hidden
acceptance tests	unit tests
users, customers	programmers, developers

Abstract Events: Bank ATM

new name Albert Einstein

deposit Albert Einstein
\$20.25

withdraw Niels Bohr
\$10.02

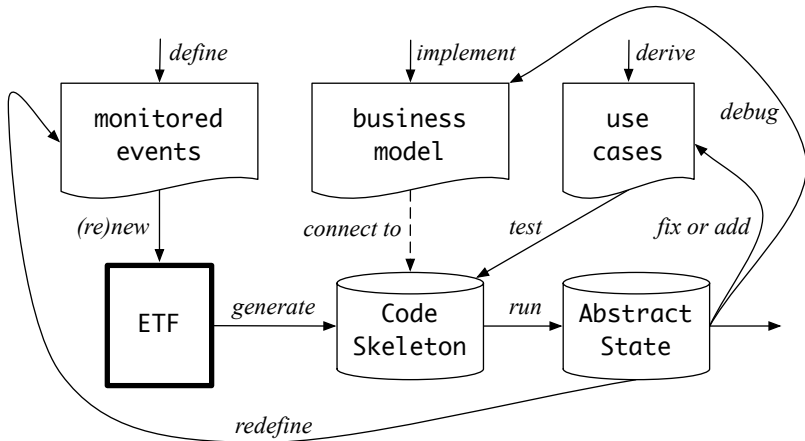
transfer Albert Einstein
\$20.25
Niels Bohr

total:
\$124.45

ETF in a Nutshell

- **Eiffel Testing Framework (ETF)** facilitates engineers to write and execute **input-output-based acceptance tests**.
 - **Inputs** are specified as traces of events (or sequences).
 - The **boundary** of the system under development (SUD) is defined by declaring the list of input events that might occur.
 - **Outputs** (from executing events in the input trace) are by default logged onto the terminal, and their formats may be customized.
- An executable ETF that is tailored for the SUD can already be generated, using these event declarations (documented in a plain text file), with a default **business model**.
- Once the **business model** is implemented, there is only a small number of steps to follow for the developers to connect it to the generated ETF.
- Once connected, developers may re-run all use cases and observe if the expected state effects take place.

Workflow: Develop-Connect-Test



ETF: Abstract User Interface

Input Grammar

```
system bank
type NAME = STRING
```

```
new(name1: NAME)
-- create a new bank account for "id"
```

```
deposit(name1: NAME; amount: VALUE)
-- deposit "amount" into the account of "id"
```

```
withdraw(name1: NAME; amount: VALUE)
-- withdraw "amount" from the account of "id"
```

```
transfer(name1: NAME; name2: NAME; amount: VALUE)
-- transfer "amount" from "id1" to "id2"
```

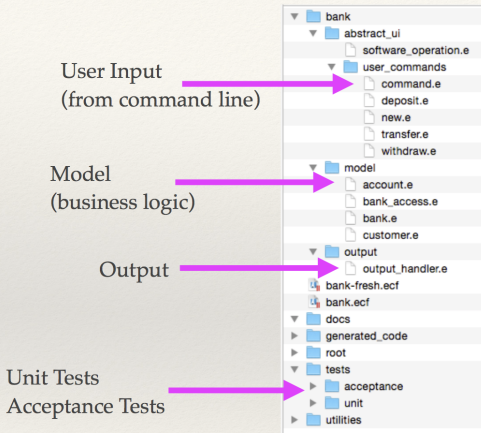


```
%bank -b at1.txt
```

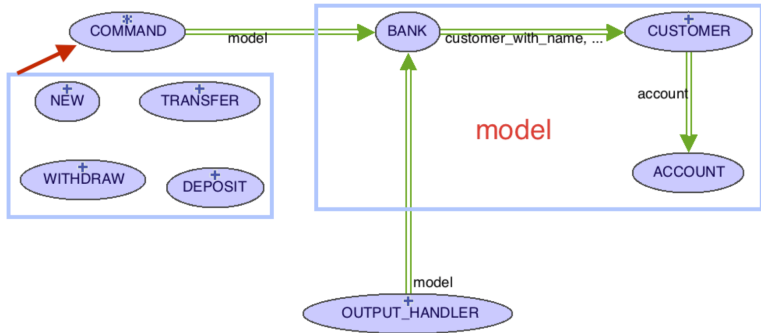
```
init
->new("Steve")
name: Steve, balance: 0.00
->new("Bill")
name: Bill, balance: 0.00
name: Steve, balance: 0.00
->deposit("Steve",520)
name: Bill, balance: 0.00
name: Steve, balance: 520.00
->new("Pam")
name: Bill, balance: 0.00
name: Pam, balance: 0.00
name: Steve, balance: 520.00
->deposit("Bill",100)
name: Bill, balance: 100.00
name: Pam, balance: 0.00
name: Steve, balance: 520.00
->withdraw("Steve",20)
name: Bill, balance: 100.00
name: Pam, balance: 0.00
name: Steve, balance: 500.00
```


ETF: Generating a New Project

```
etf -new bank.input.txt <directory>
```



ETF: Architecture



- Classes in the `model` cluster are hidden from the users.
- All commands reference to the same model (`bank`) instance.
- When a user's request is made:
 - A **command object** of the corresponding type is created, which invokes relevant feature(s) in the `model` cluster.
 - Updates to the model are published to the output handler.

ETF: Input Errors

```
class
  ETF_DEPOSIT
inherit
  ETF_DEPOSIT_INTERFACE
  redefine deposit end
create
  make
feature -- command
  deposit(id: STRING ; amount: REAL_64)
  do
    if not model.has_user (id) then
      -- Set some error message
    elseif not amount <= model.get_balance (id) then
      -- Set some other error message
    else
      -- perform some update on the model state
      model.deposit (id, amount)
    end
    -- Publish model update
    etf_cmd_container.on_change.notify ([Current])
  end
end
```

Index (1)

Bank ATM

Separation of Concerns

Prototyping System with Abstract UI

Abstract Events: Bank ATM

ETF in a Nutshell

Workflow: Develop-Connect-Test

ETF: Abstract User Interface

ETF: Generating a New Project

ETF: Architecture

ETF: Input Errors