



Exigences & Acceptation

Sébastien Mosser
MGL7460 - Cours #4 - H20

UQÀM | Département d'informatique

Crédit Images: Pixabay & Pexels



Crédits : Conférence “Req. Engineering”

Reconciling Requirements and Continuous
Integration in an Agile Context

- S. Mosser & J.M. Bruel



1

Nouvelles du cours

2

Exigences ?

3

Récits Utilisateurs

4

Acceptation ?

5

Tests d'acceptations

6

Projet Technique

Nouvelles
du cours

#Semaine	Cours (UQAM)	Atelier (SFL)
2	Leçon introductive: Réaliser, Maintenir et Modéliser du logiciel ?	(Lab optionnel)
3	Gestion de versions, Tests	--
4	--	Git, GitLab, GitLab CI
5	Exigences, Scénarios d'acceptations	--
6	Déploiement continu, Test-driven development	--
7	--	BDD, TDD
8	Analyse de code	Sonarqube
9	Semaine de relâche	--
10	Principes de développement (CleanCode, SOLID)	--
11	Métriques Logicielles & Visualisation pour la maintenance	--
12	Présentation intermédiaire du Projet Technique	idem
13	Leprechauns du Génie Logiciel	Suivi projet technique
14	Cours invité DevOps (Pr Francis Bordeleau, ÉTS)	Suivi projet technique
15	Cours invité qualité (Pr Xavier Blanc, Univ. Bordeaux)	Suivi projet technique
16	Rencontres Projet Individuel	Suivi projet technique
17	Rencontres Projet Individuel	Suivi projet technique

Proj. Individuel

- 17 inscrits
 - **13 projets choisis**
- Peu d'interactions sur les cas
- 1h de cours = 2h travail perso
 - cf règlement UQAM
 - **18h de travail supposé effectué à date**

Ceci est un EXAMEN

Et mon intervalle de notation est [0,100], pas [70,95].

- Marc-André: Logiciel Krita
 - Dépôt de code : <https://github.com/KDE/krita>
- Nasseredine: Logiciel Spring Initializr
 - Dépôt de code : <https://github.com/spring-io/initializr>
- Gary: Logiciel Angular
 - Dépôt de code : <https://github.com/angular/angular>
- Cristina: Logiciel React
 - Dépôt de code : <https://github.com/facebook/react>
- Leonardo: Logiciel NGXS
 - Dépôt de code : <https://github.com/ngxs/store>
- Simon: Logiciel Dolphin
 - Dépôt de code : <https://github.com/dolphin-emu/dolphin>
- Gérald: Logiciel NodeRed
 - Dépôt de code : <https://github.com/node-red/node-red>
- Lilian: Logiciel Jenkins
 - Dépôt de code : <https://github.com/jenkinsci/jenkins>
- Dionisie: Logiciel JHipster Generator
 - Dépôt de code : <https://github.com/jhipster/generator-jhipster>
- Pape Tahyre: Logiciel Mockito
 - Dépôt de code : <https://github.com/mockito/mockito>
- Jean-Pierre: Logiciel Wordpress
 - Dépôt de code : <https://github.com/WordPress/WordPress>
- Majed: Logiciel JUnit5
 - Dépôt de code : <https://github.com/junit-team/junit5>
- Alexandre: Logiciel MySQL-Server
 - Dépôt de code : <https://github.com/mysql/mysql-server>

Dates de remise à venir

Date(s)	Travail à rendre	Poids
19.01.20	Choix du cas d'études individuel	0%
01.03.20	Projet Individuel - V1	20%
15.03.20	Projet Technique - MVP	20%
12.04.20	Projet Individuel - V2	40%
26.04.20	Projet Technique - Final	20%

*Les dates de remise s'entendent sur le fuseau horaire de Montréal, à 23:50 le jour de la date de remise.
Tout rendu hors délai recevra la note de zéro (0)*

Évaluations des projets

- Quel moyen de communication pour Lévis ?
 - Skype ?
- Quel lieu ?
 - En parallèle des entretiens, vous travaillez le projet ...
 - Traverser et prendre un local au PK pour les rendez-vous ?

Signalez
les
problèmes !



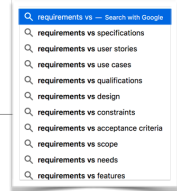
Exigences ?

2

Qualité Logicielle

"Software does the right
thing, and does them right"

Requirements



What the **system should do**

Requirements vs **Specification**

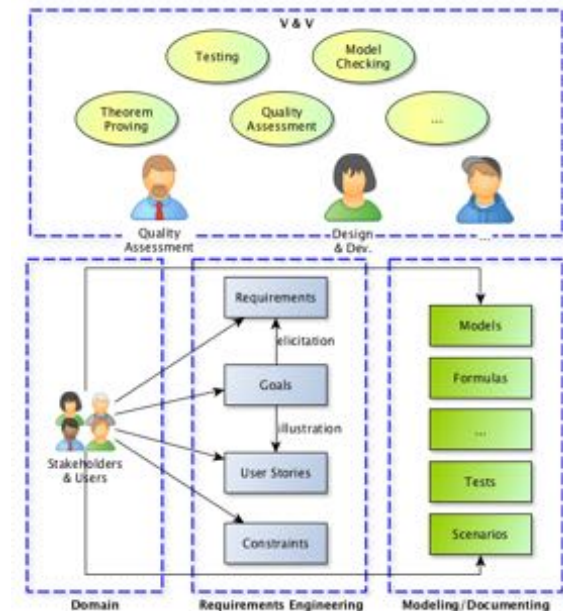
Both describe the **what** (more than **how**)

Different purpose and scope:

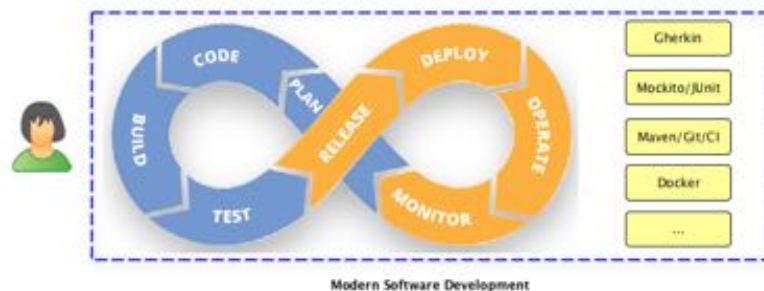
- specification => technical properties
- reqs => properties from the user or environment

13

Aperçu Général

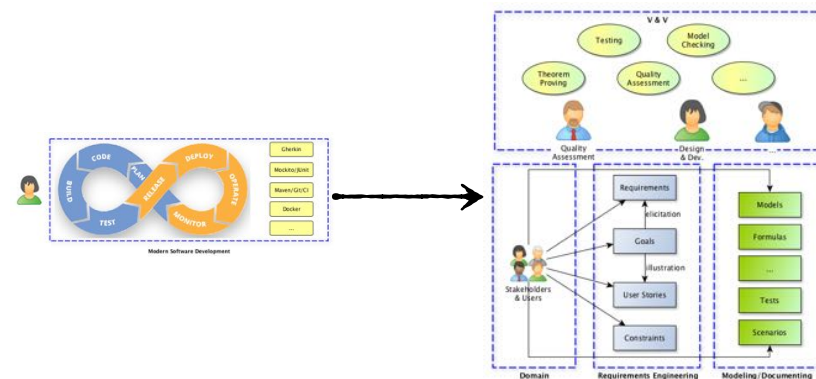


Modern software development: **DEVOPS**



15

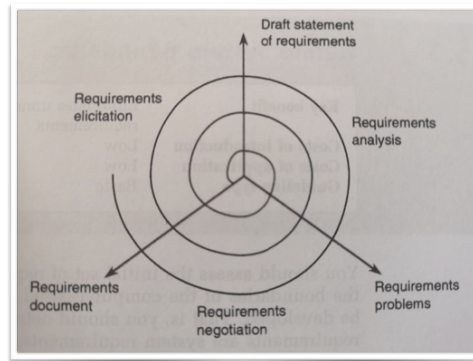
Apply good practice to RE



16

Processus d'Ingénierie des Exigences

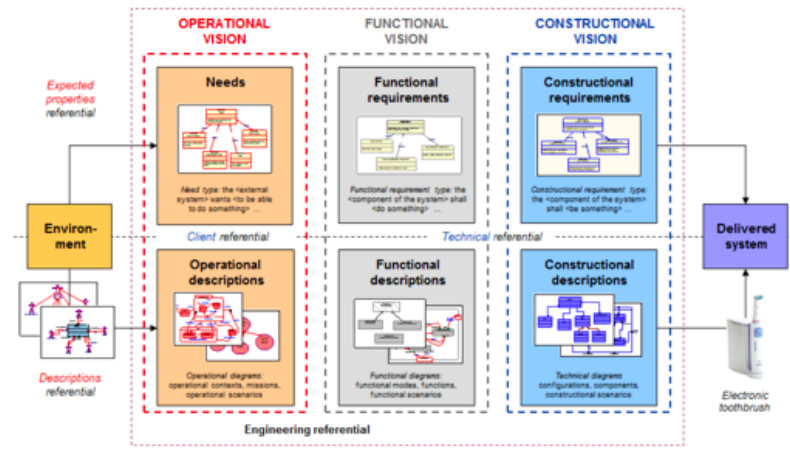
- Requirements Elicitation
- Requirements Analysis & Negotiation
- Requirements Validation
- Requirements Documentation
- Requirements Management



[Sommerville & Sawyer 1997]

17

Propriétés attendues / Décrites



<http://www.cesames.net/wp-content/uploads/2017/05/CESAM-guide.pdf>

18

Exercice !

Une main de poker comprend 5 cartes tirées d'un seul jeu de 52 cartes.

Chaque carte à une couleur, Trèfle, Carreau, Coeur, Pique (dénotée Tr, Ca, Co, Pi) et une valeur parmi 2, 3, 4, 5, 6, 7, 8, 9, 10, valet, dame, roi, as (dénotée 2, 3, 4, 5, 6, 7, 8, 9, 10, V, D, R, A).

Pour le calcul du score, toutes les couleurs ont le même niveau, par exemple l'as de carreau n'est pas battu par l'as de pique, ils sont égaux. Les valeurs sont ordonnées comme définies précédemment, le 2 étant la plus petite valeur et l'as la plus grande.

Les mains sont classées de la plus faible à la plus forte :

- **Plus haute carte** : les mains qui ne correspondent à aucune autre catégorie sont classées par la valeur de leur plus haute carte. Si les plus hautes cartes ont la même valeur, les mains sont classées par la plus haute suivante et ainsi de suite.
- **Paire** : 2 des 5 cartes de la main ont la même valeur. Deux mains qui contiennent une paire sont classées par la valeur des cartes formant la paire. Si les valeurs sont les mêmes, les mains sont classées par les cartes hors de la paire, en ordre décroissant.
- ...

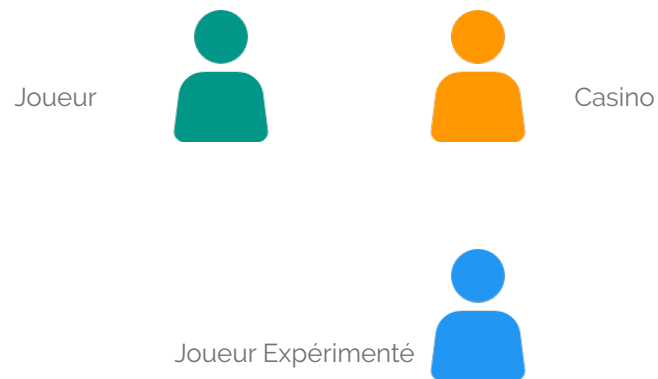
Comparer deux mains de poker saisie sur l'entrée standard, déterminer laquelle est la plus forte et afficher ce résultat.

Exigences: Analyse & Négociation




- Identifier les parties prenantes et leurs besoins
- Passer des besoins aux exigences "formelle"
- Prioriser le jeu d'exigences
 - Utiliser différents "modaux" (p.-ex., should, must, ...)
- Mesurer le risque associé à chaque exigence

20

Parties prenantes






Liste des exigences

Reqs Priority / Stakeholders	Player 	Casino 	Exp. Player 
Must			
Should			
Could			
Will			




22

Liste des exigences

Reqs Priority / Stakeholders	Player 	Casino 	Exp. Player 
Must	Comparing Poker Hands	See Rules*	Comparing Poker Hands
Should	Create hands		Provide stats
Could	Provide stats		Select card games (style,...)
Will	Have an API for game plugin		Have the menu in French

23

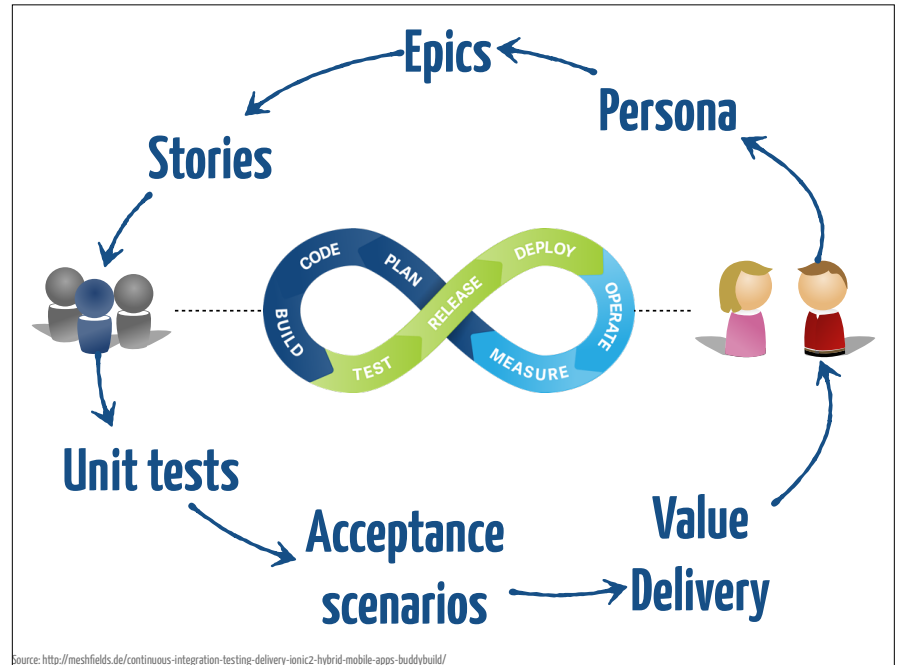
Évaluation du risque

Reqs Priority / Stakeholders	Player 	Casino 	Exp. Player 
Must	Comparing Poker Hands	See Rules*	Comparing Poker Hands
Should	Create hands		Provide stats
Could	Provide stats		Select card games (style,...)
Will	Have an API for game plugin		Have the menu in French

24

Récits
Utilisateurs

3



User Story!

User Story ?

Description d'une valeur ajoutée
pour un persona



Critères d'acceptations ⊕ Tests

Persona : Utilisateur type. Description précise



• CONTEXT •

En recherche d'une activité une dimanche après-midi à Paris, avec une amie. Pour l'aider dans sa démarche : mettre directement les informations nécessaires, Juliette n'aimant pas perdre son temps. Il faudra lui proposer de manière simple le lieu ainsi que le descriptif des activités.

• SKILLS •

- Réseaux sociaux
- Online shopping
- Internet
- Organisation

• WANTS & NEEDS •

- Être avec ses amies
- Se promener dans Paris
- Découvrir des choses

• FEARS •

- Louper une nouveauté
- Tomber sur des réseaux sociaux pas à jour
- Perdre son temps

Juliette Patel
25 ans
Community manager
Paris

“ Community manager dans une grande entreprise de ventes de chaussures. Passionnée de nouvelles technologies et de mode, elle passe énormément de temps sur le web. Célibataire, elle profite de son temps libre pour faire les magasins et voir les nouveautés de la mode avec ses amies. Elle fait partie de la classe sociale élevée et sort d'un master en communication web.

<https://www.behance.net/gallery/71853957/Etche-Persona-UX-design-persona>

Description

En tant qu'**utilisateur de Gégémail**,
je veux **me connecter à ma boîte mail**
afin de **gérer mes mails de façon sécurisée**

— PODOJO, USTA

Critères d'acceptation

- Sur la page d'accueil, je saisis mon login / mot de passe
- Si le login saisi n'existe pas, message d'erreur
- Si le mot de passe ne correspond pas au login, message d'erreur
- Si le login existe et que le mot de passe correspond au login, j'accède à ma boîte de réception

— PODOJO, USTA

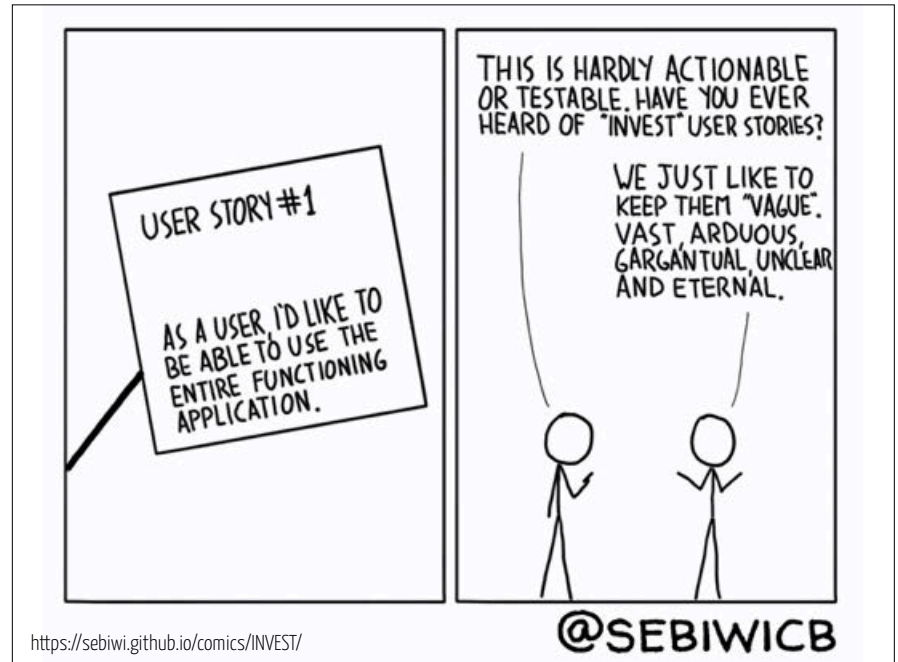
Test d'acceptation

- Initialisation : Je vais sur la page d'accueil de Gégémail
- Action : Je saisis un login qui n'existe pas et un mot de passe
- Résultat attendu : Message d'erreur

— PODOJO, USTA

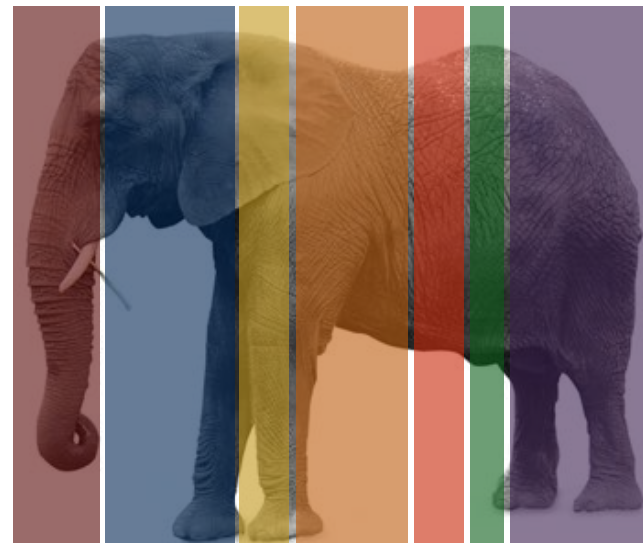
Une story est un INVESTissement

I ndépendante
N égociable
V alorisée
E stimable
S imple ~ small
T estable



En
apportant
de la
valeur,
une story
est par
nature

VERTICALE



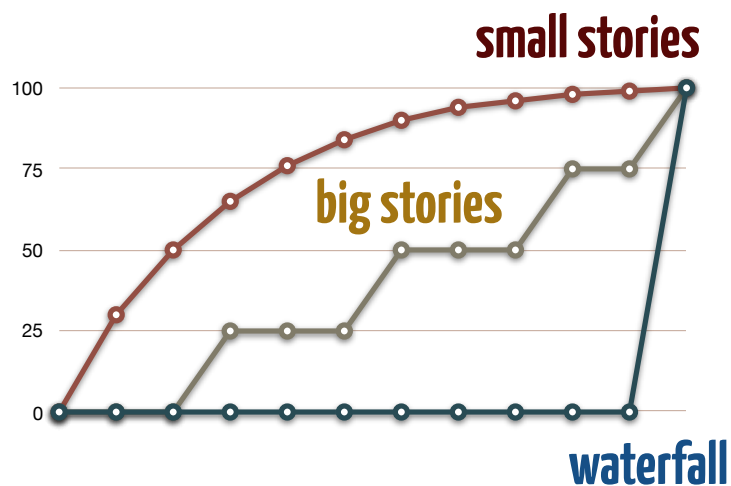
A **user story** is to a **use case**
as a **gazelle** is to a **gazebob**.

- Alistair Cockburn

A **user story** is the **title of one scenario**, where a **use case** is the **contents of multiple scenarios**

- Alistair Cockburn

Cumulative value

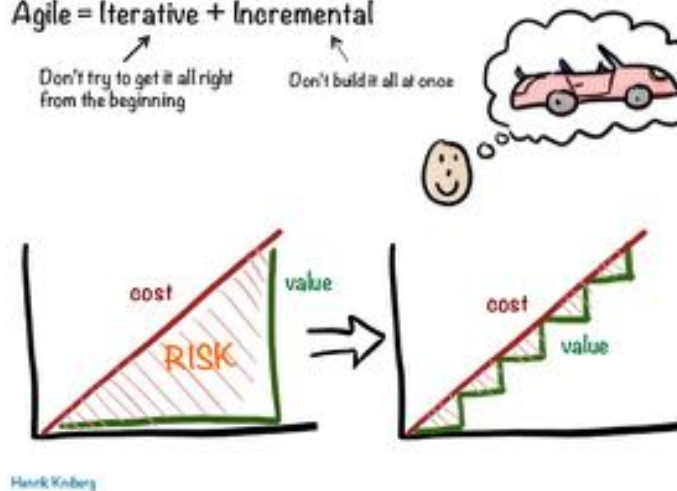


39

Agile = Iterative + Incremental

Don't try to get it all right from the beginning

Don't build it all at once



Henrik Kniberg

Estimation

Planning Poker

**Quel est le poids moyen d'un
berger des Pyrénées
adulte en bonne santé ?**

Berger ?



Berger ?

Berger des Pyrénées

Westie

Chihuahua

Absolu \neq Relatif

Estimer en Story Point(s)

pas en jours / heures

Jamais

Sérieusement. Jamais.

Estimer la **valeur**

Estimer l'**effort**

Définissez un jeu de stories pour le Poker!

Une main de poker comprend 5 cartes tirées d'un seul jeu de 52 cartes.

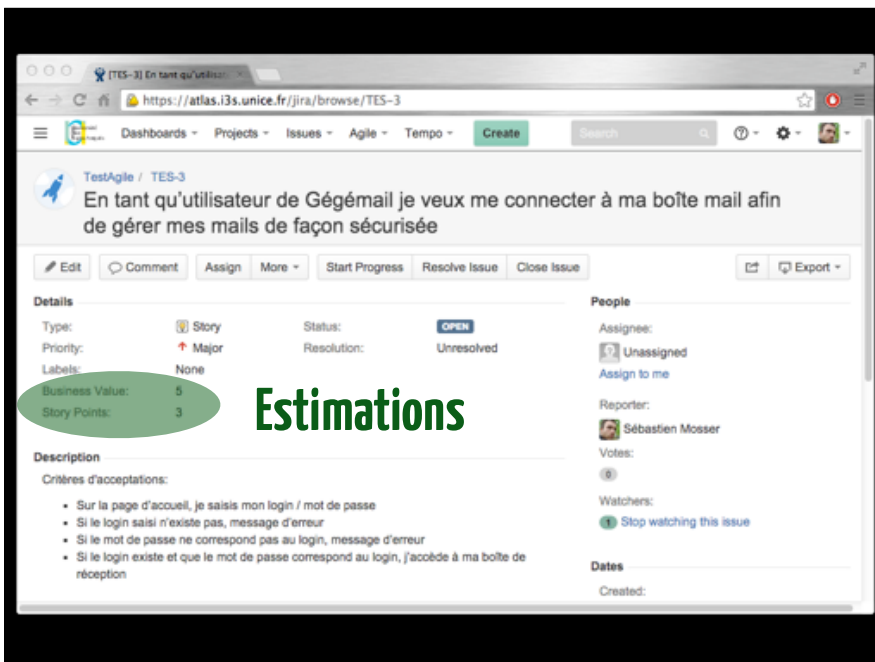
Chaque carte à une couleur, Trèfle, Carreau, Coeur, Pique (dénotée Tr, Ca, Co, Pi) et une valeur parmi 2, 3, 4, 5, 6, 7, 8, 9, 10, valet, dame, roi, as (dénotée 2, 3, 4, 5, 6, 7, 8, 9, 10, V, D, R, A).

Pour le calcul du score, toutes les couleurs ont le même niveau, par exemple l'as de carreau n'est pas battu par l'as de pique, ils sont égaux. Les valeurs sont ordonnées comme définies précédemment, le 2 étant la plus petite valeur et l'as la plus grande.

Les mains sont classées de la plus faible à la plus forte :

- **Plus haute carte** : les mains qui ne correspondent à aucune autre catégorie sont classées par la valeur de leur plus haute carte. Si les plus hautes cartes ont la même valeur, les mains sont classées par la plus haute suivante et ainsi de suite.
- **Paire** : 2 des 5 cartes de la main ont la même valeur. Deux mains qui contiennent une paire sont classées par la valeur des cartes formant la paire. Si les valeurs sont les mêmes, les mains sont classées par les cartes hors de la paire, en ordre décroissant.
- ...

Comparer deux mains de poker saisie sur l'entrée standard, déterminer laquelle est la plus forte et afficher ce résultat.



TestAgile / TES-3

En tant qu'utilisateur de Gégemail je veux me connecter à ma boîte mail afin de gérer mes mails de façon sécurisée

Details

Type: Story Status: OPEN

Priority: Major Resolution: Unresolved

Labels: None

Business Value: 5

Story Points: 3

Estimations

Description

Critères d'acceptations:

- Sur la page d'accueil, je saisis mon login / mot de passe
- Si le login saisi n'existe pas, message d'erreur
- Si le mot de passe ne correspond pas au login, message d'erreur
- Si le login existe et que le mot de passe correspond au login, j'accède à ma boîte de réception

People

Assignee: Unassigned

Reporter: Sébastien Mosser

Votes: 0

Watchers: 1 Stop watching this issue

Dates

Created:



Acceptation (definition of done)

4

Comment valider mes récits ?



As Bob, I want to enter my hand on the command line so that the game knows the contents of my hand ?

As Alice, I want to identify cheaters that trick the card deck so that I can report cheating attempts to management ?

```
azrael:agile-tutorial mosser$ java -jar target/poker-game.jar
Enter 1st player hand: QD QH KC KH 3S
1st: [KING of HEARTS, QUEEN of DIAMONDS, THREE of SPADES, QUEEN of HEARTS, KING of CLUBS]
Enter 2nd player hand: 7S 6H 4S 3D 2C
2nd: [FOUR of SPADES, SEVEN of SPADES, TWO of CLUBS, SIX of HEARTS, THREE of DIAMONDS]
Exception in thread "main" java.lang.UnsupportedOperationException: Cannot determine winner!
    at re.poker.Game.declareWinner(Game.java:23)
    at re.poker.Main.main(Main.java:22)
```

51

Avec des tests !



poker (re)	411 ms
GameTest	327 ms
rejectNonLegitHands	327 ms
declareTheWinner	0 ms
HandTest	84 ms
readAPokerHandFromAStream	74 ms
properlyBuildAnHand	9 ms
rejectAnHandWithDuplicatedCards	1 ms
rejectAnHandWithTooManyCards	0 ms
rejectAnHandWithNotEnoughCards	0 ms
CardTest	0 ms
identifyDifferentCardsBasedOnSuits	0 ms
comparisonDiffersFromEquality	0 ms
identifyEqualsCard	0 ms
orderCardsBasedOnValues	0 ms
identifyDifferentCardsBasedOnValues	0 ms
CardValueTest	0 ms
checkMappingBetweenSymbolsAndValues	0 ms
SuitTest	0 ms
checkMappingBetweenSymbolsAndSuits	0 ms

52

Tests unitaires



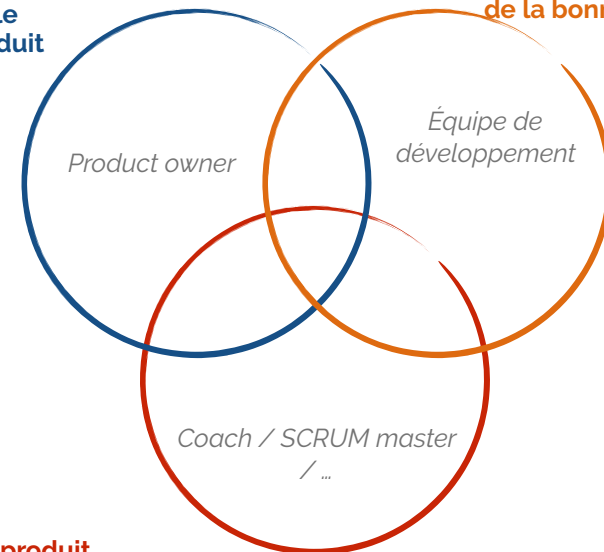
Est-on au bon niveau d'abstraction ?

```
@Test
public void thisIsATest() {
    Game theGame = new Game();
    theGame.submit("Bob", new Hand("AC KC QC JC TC"));
    CardValue v = CardValue.valueOf("ACE");
    Suit s = Suit.valueOf("CLUBS");
    Card theCard = new Card(v.getSymbol()+" "+s.getSymbol());
    Card max = Collections.max(theGame.getByPlayer("Bob").getCards());
    assertEquals(theCard, max);
}
```

53

Faire le
bon produit

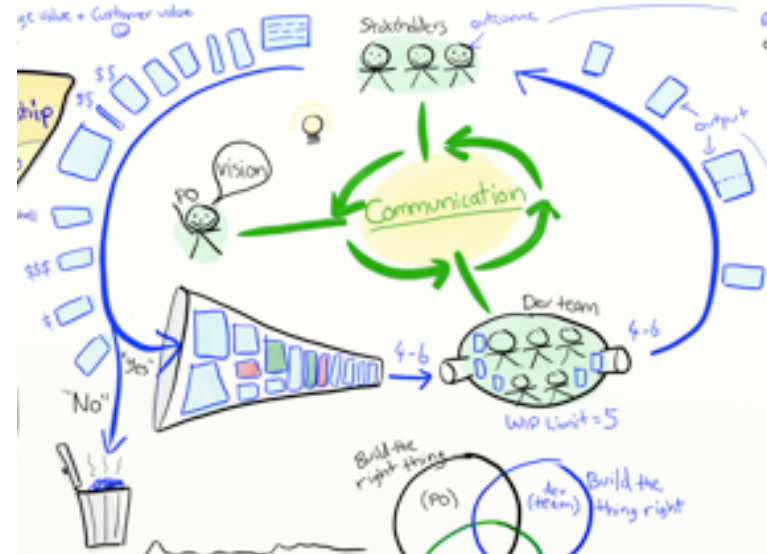
Faire le produit
de la bonne manière



Faire le produit
rapidement

L'acceptation est critique !

- Un récit utilisateur vient avec :
 - Une "definition of ready"
 - Une "definition of done"
- Sans mesure d'acceptation "formelle", on ne peut pas conclure à l'arrêt d'un développement.
- Plus l'acceptation est automatisé, plus rapide est la boucle de rétroaction



<https://www.youtube.com/watch?v=502HJHX9EE>

Que pensez vous de ce scénario ?

Scenario: Identify the highest card in an hand

Given a new game

When Bob submits the following cards: AC KC QC JC TC

Then Bob's highest card is the ACE of CLUBS

```
@Test
public void IdentifyTheHighestCardInAnHand() {
    Game theGame = new Game();
    theGame.submit("Bob", new Hand("AC KC QC JC TC"));
    CardValue v = CardValue.valueOf("ACE");
    Suit s = Suit.valueOf("CLUBS");
    Card theCard = new Card(v.getSymbol()+" "+s.getSymbol());
    Card max = Collections.max(theGame.getByPlayer("Bob").getCards());
    assertEquals(theCard, max);
}
```

57

IDE Integration (IntelliJ example)



58

Steps auto-completion

```
Feature: Reading a Poker Hand
  Scenario: Read a regular hand
    Given a new game
    When Bob submits the following cards: QD TS 2C KD 3C
    Then Bob's hand contains 5 cards
      And Bob's hand contains the QUEEN of DIAMONDS
      And Bob's hand contains the TEN of SPADES
      And Bob's hand contains the TWO of CLUBS
      And Bob's hand contains the KING of DIAMONDS
      And Bob's hand contains the THREE of CLUBS
  Scenario: Reject an hand with a duplicated card
    Given a new game
    When Bob submits the following cards: QD TS 2C KD QD
    And
    Then <string>'s hand contains <number> cards
      <string>'s hand contains the <string> of <string>
  Scenario: Give
    Given
    When <string> submits the following cards: <string>
    And
    Then a cheat attempt is detected!
  Scenario:
    Given a new game
```

59

Gherkin

```
Feature: Logout from application
Scenario:
  Given I am logged in
  When I click "log out" button
  Then I am informed about successful logout
  And I am redirected to login page
```

To create a new requirements description, we need to define the **Feature** which gives us the name of a new **Functionality**. Then, we go ahead with writing the **Scenario**.

60

Tests d'acceptation
(e.g., Cucumber)

5



Feature: Cocktail Ordering

As Romeo, I want to offer a drink to Juliet, so that we can “discuss”.

**Minimal
(~Viable)
Scenario**

?

Créer la commande vide

```
public class OrderingCocktailTest {  
    private Order order;  
  
    @Test  
    public void empty_order_by_default() {  
        order = new Order();  
        order.declareOwner("Romeo");  
        order.declareTarget("Juliette");  
        List<String> cocktails = order.getCocktails();  
        assertEquals(0, cocktails.size());  
    }  
}
```

Given

Romeo who wants to buy a drink

When

an order is declared for **Juliette**

Then

there is **0** cocktails in the order

(this.language = Gherkin)

Given **Romeo** who wants to buy a drink
When an order is declared for **Juliette**
Then there is **0** cocktails in the order

Mapping



```
@Test  
public void empty_order_by_default() {  
    order = new Order();  
    order.declareOwner("Romeo");  
    order.declareTarget("Juliette");  
    List<String> cocktails = order.getCocktails();  
    assertEquals(0, cocktails.size());  
}
```

```
public class CocktailStepDefinitions {  
    private Order order;  
  
    @Given("Romeo who wants to buy a drink")  
    public void romeo_wants_to_buy_a_drink() {  
        order = new Order();  
        order.declareOwner("Romeo");  
    }  
  
    @When("an order is declared for Juliette")  
    public void an_order_is_declared_for_juliette() {  
        order.declareTarget("Juliette");  
    }  
  
    @Then("There is 0 cocktails in the order")  
    public void there_is_no_cocktails_in_the_order() {  
        List<String> cocktails = order.getCocktails();  
        assertEquals(0, cocktails.size());  
    }  
}
```

Bonne Pratique

Scénarios d'acceptations décrits dans
les tickets associés à vos stories

Scenario: Creating an empty order
Given Romeo who wants to buy a drink
When an order is declared for Juliette
Then there is 0 cocktails in the order

Cocktail.feature

cucumber

POJO + Annotations

```
public class CocktailStepDefinitions {
```

```
    private Order order;
```

```
    @Given("Romeo who wants to buy a drink")
```

```
    public void romeo_wants_to_buy_a_drink() {
```

```
        order = new Order();
```

```
        order.declareOwner("romeo");
```

```
    }
```

```
    @When("an order is declared for Juliette")
```

```
    public void an_order_is_declared_for_Juliette() {
```

```
        order.declareTarget("juliette");
```

```
    }
```

```
    @Then("there is 0 cocktails in the order")
```

```
    public void there_is_n_cocktails_in_the_order(int n) {
```

```
        List<String> cocktails = order.getCocktails();
```

```
        assertEquals(0, cocktails.size());
```

```
    }
```

```
}
```

Test Results	433ms
Feature: Cocktail Ordering	433ms
Scenario: Creating an empty order	167ms
Given Romeo who wants to buy a drink	162ms
When an order is declared for Juliette	0ms
Then there is 0 cocktails in the order	5ms

Romeo, Juliet, Tom & Jerry?



Tom & Jerry?

Given Tom who wants to buy a drink
When an order is declared for Jerry
Then there is 0 cocktails in the order

Given Romeo who wants to buy a drink
When an order is declared for Juliette
Then there is 0 cocktails in the order

Tom & Jerry?

Given Tom who wants to buy a drink
 When an order is declared for Jerry
 Then there is 0 cocktails in the order

```
@Given("Romeo who wants to buy a drink")
public void romeo_who_wants_to_buy_a_drink() { ... }

@When("an order is declared for Juliette")
public void an_order_is_declared_for_juliette() { ... }

@Given("Tom who wants to buy a drink")
public void tom_who_wants_to_buy_a_drink() { ... }

@When("an order is declared for Jerry")
public void an_order_is_declared_for_jerry() { ... }

@Then("There is 0 cocktails in the order")
public void there_is_no_cocktails_in_the_order() { ... }
```

Duplication!

Tom & Jerry?

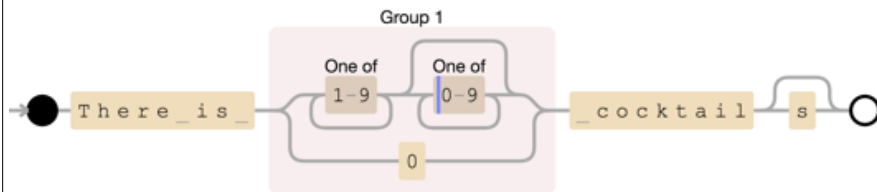
Tom Jerry

Given aString who wants to buy a drink
 When an order is declared for aString
 Then there is n cocktails in the order

Romeo Juliet

0

Expression Rationnelles



^There is ([1-9]+[0-9]*|0) cocktails?\$

Digression

Office québécois
 de la langue
 française Québec

A regular language is recognised
 by a regular expression.

Un langage rationnel est
 reconnu par une expression ...

(et tant qu'on y est une librairie c'est là où on achète des livres, alors qu'une bibliothèque ...)

```

public class CocktailStepDefinitions {
    private Order order;

    @Given("^(.*) who wants to buy a drink$")
    public void someone_wants_to_buy_a_drink(String romeo) {
        order = new Order();
        order.declareOwner(romeo);
    }

    @When("^an order is declared for (.*)$")
    public void an_order_is_declared_for_someone(String juliette) {
        order.declareTarget(juliette);
    }

    @Then("^there is (\\d+) cocktails in the order$")
    public void there_is_n_cocktails_in_the_order(int n) {
        List<String> cocktails = order.getCocktails();
        assertEquals(n, cocktails.size());
    }
}

```

File Cocktail.feature

Scenario: Creating an empty order
Given Romeo who wants to buy a drink
When an order is declared for Juliette
Then there is <number> cocktails in the order

Automated completion

Paresse
du développeur

Don't Repeat Yourself

Given Romeo who wants to buy a drink
When an order is declared for Juliet
And a message saying "Ciao!" is added
Then the ticket must say "From R to J: Ciao!"

Given Romeo who wants to buy a drink
When an order is declared for Tom
And a message saying "Hey!" is added
Then the ticket must say "From R to T: Hey!"

Background & Outline

Background:

Given **Romeo** who wants to buy a drink

Scenario Outline: Sending a message with an order

When an order is declared for **<to>**

And a message saying "**<msg>**" is added

Then the ticket must say "**<expected>**"

Examples:

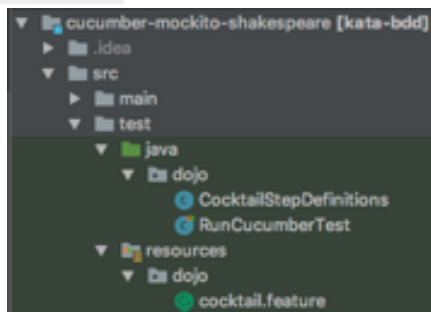
	<i>to</i>	<i>msg</i>	<i>expected</i>
	Juliette	Ciao!	From T to J: Ciao!
	Tom	Hey!	From R to T: Hey!
# ...			

(templating)

▼	✓	Scenario Outline: Sending a message with an order	1ms
▼	✓	Examples:	1ms
▼	✓	Scenario: Line: 20	0ms
	✓	Given Romeo who wants to buy a drink	0ms
	✓	When an order is declared for Juliette	0ms
	✓	And a message saying "Wanna chat?" is added	0ms
	✓	Then the ticket must say "From Romeo to Juliette	0ms
▼	✓	Scenario: Line: 21	1ms
	✓	Given Romeo who wants to buy a drink	0ms
	✓	When an order is declared for Jerry	1ms
	✓	And a message saying "Hei!" is added	0ms
	✓	Then the ticket must say "From Romeo to Jerry: Hei!"	0ms

Technical Details

```
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-java</artifactId>
<scope>test</scope>
</dependency>
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-junit</artifactId>
<scope>test</scope>
</dependency>
```



```
@RunWith(Cucumber.class)
public class RunCucumberTest { }
```

```

Feature: Cocktail Ordering

As Romeo, I want to offer a drink to Juliette so that we can discuss together (and maybe more).

Background:
  Given Romeo who wants to buy a drink
  When an order is declared for Juliette

Scenario: Creating an empty order
  Then there is 0 cocktails in the order

Scenario Outline: Sending a message with an order
  When an order is declared for <to>
  And a message saying "<message>" is added
  Then the ticket must say "<expected>"

Examples:
| to           | message           | expected                                     |
| Juliette    | Wanna chat?      | From Romeo to Juliette: Wanna chat?      |
| Jerry       | Hei!              | From Romeo to Jerry: Hei!                 |
# ...

Scenario: Offering a mojito to Juliette
  When a mocked menu is used
  And the mock binds #42 to mojito
  And a cocktail #42 is added to the order
  Then there is 1 cocktails in the order
  And the order contains a mojito

Scenario: Paying the mojito offered to Juliette
  When a mocked menu is used
  And the mock binds #42 to $10
  And a cocktail #42 is added to the order
  And Romeo pays his order
  Then the payment component must be invoked 1 time for $10

Scenario: Not paying the empty bill
  When Romeo pays his order
  Then the payment component must be invoked 0 time for $0
  
```

cocktail.feature

cocktail.feature

Run Feature: cocktail

Test Results

Feature: Cocktail Ordering

Scenario: Creating an empty order

Given Romeo who wants to buy a drink

Then there is 0 cocktails in the order

Scenario Outline: Sending a message with an order

Examples:

Scenario: Line: 20

Given Romeo who wants to buy a drink

When an order is declared for Juliette

And a message saying "Wanna chat?" is ad

Then the ticket must say "From Romeo to J

Scenario: Line: 21

Given Romeo who wants to buy a drink

When an order is declared for Jerry

And a message saying "Hell" is added

Then the ticket must say "From Romeo to J

Scenario: Offering a mojito to Juliette

Given Romeo who wants to buy a drink

When a mocked menu is used

And the mock binds #42 to mojito

And a cocktail #42 is added to the order

Then there is 1 cocktails in the order

And the order contains a mojito

Scenario: Paying the mojito offered to Juliette

Given Romeo who wants to buy a drink

When a mocked menu is used

And the mock binds #42 to \$10

And a cocktail #42 is added to the order

And Romeo pays his order

Then the payment component must be invoked 1

Scenario: Not paying the empty bill

Given Romeo who wants to buy a drink

When Romeo pays his order

Then the payment component must be invoked 0

Testing started at 21:36 ...

/Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java

objc32000): Class JavaLaunchHelper is implemented in both /Library/Java/

6 Scenarios (6 passed)

25 Steps (25 passed)

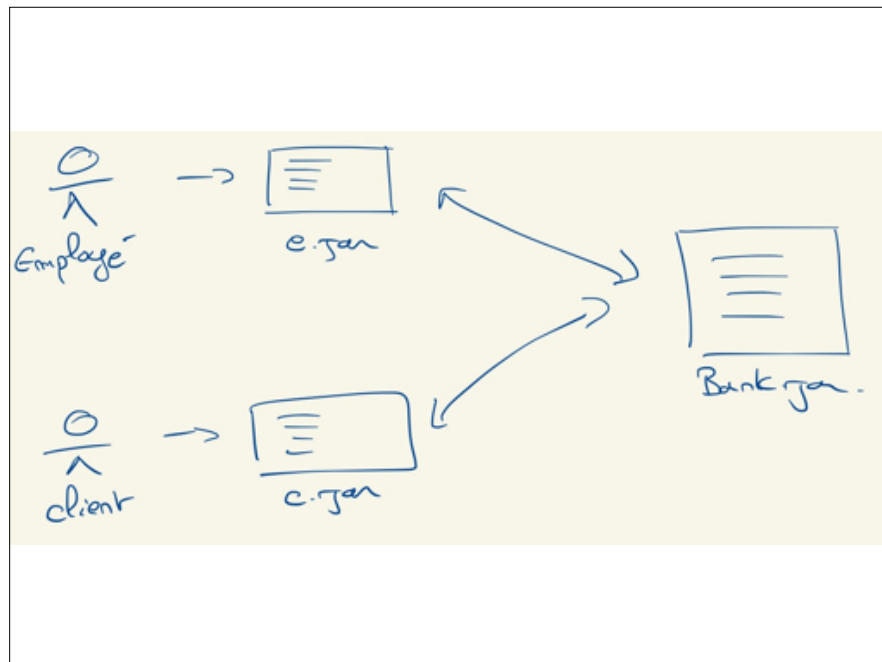
0m0.468s

Process finished with exit code 0

IDE

Integration

Projet Technique (lancement)



Travaux

- Projet Individuel
 - Mémoire de synthèse sur la maintenabilité d'un logiciel
- Projet Technique :
 - Mise en oeuvre d'un pipeline de déploiement continu

Projet Technique MGL7460-90 (Hiver 2020)

- Auteur : Sébastien Mosser (UQAM)
- Date de publication : Janvier 2020

Objectifs pédagogiques

Dans ce projet technique à réaliser par équipe de deux, vous êtes responsable de la "réalisation" d'un logiciel de petite envergure. Cette réalisation couvre toutes les phases du développement, du recueil des exigences à la mise en place d'un environnement de déploiement continu pour le projet.

Contraintes

- Le développement est effectué en Java;
- La compilation du projet d'effectue à l'aide de Maven (ou de Gradle), lancé depuis la racine de votre dépôt;
- Le pipeline d'intégration continue est automatiquement lancé au `push` sur le dépôt
- les scénarios d'acceptations sont implémentés en cucumber.

Fonctionnalités du module employé

- `./employee --add CLIENT_NAME`
 - Crée un client de nom `CLIENT_NAME`
- `./employee --list CLIENT_NAME`
 - Liste les produits attaché au compte de ce client
- `./employee --accept PRODUCT_ID --client CLIENT_NAME`
 - Accepte la mise en place du produit pour le client
- `./employee --reject PRODUCT_ID --client CLIENT_NAME`
 - Rejette le produit demandé par le client
- `./employee --tasks`
 - Liste les clients avec des produits en attente de validation
- `./employee --upgrade CLIENT_NAME`
 - augmente le statut du client, qui accède à de nouveaux produits
- `./employee --downgrade CLIENT_NAME`
 - diminue le statut du client, restreignant les produits auquel il a accès

Fonctionnalités du module client

Plutôt que renseigner à chaque invocation son numéro client, on pourrait le stocker dans une variable d'environnement `MGL_CLIENT_NAME`.

- `./client -n CLIENT_NAME --status`
 - Liste tous les produits du client
- `./client -n CLIENT_NAME --avail`
 - Liste tous les produits auquel le client a accès
- `./client -n CLIENT_NAME --subscribe PRODUCT_ID`
 - Souscrit à un produit. Si le produit est automatique, la souscription est immédiate. Sinon, elle requiert une approbation d'un employé.
- `./client -n CLIENT_NAME --unsubscribe PRODUCT_ID`
 - Quitte un produit. S'il est impossible de quitter ce produit immédiatement, cela requiert une approbation d'un employé

Travail à effectuer

1. Créer un dépôt Git pour le projet (un seul dépôt pour les trois modules)
2. Mettre en place un système de suivi d'exigences - définition de *user stories* associée au projet - Estimation des *stories* en terme de risque technique et de valeur métier - les *stories* doivent être tracées au code via les *commits*
3. Développement des 3 modules
 - mise en place d'un système de build tenant compte des dépendances
4. Tests du système - tests unitaires pour chaque modules - tests d'intégration entre les modules "interface" (employé et client) et le module de la banque - scénario d'acceptations pour valider automatiquement les *stories*
5. Pipeline de déploiement continu - au push sur le dépôt, lancement des tests, fabrication des images docker, envoi dans le registre.
6. Mise en place de scénarios de démonstration - Démontrant la valeur ajoutée du pipeline de déploiement continu pour le développement du produit.

Critères d'évaluation

Thème	Critère	Poids
Code	Qualité du code source	10%
	Qualité communication inter-module	5%
	Qualité du dépôt	10%
	Qualité des tests unitaires & intégration	10%
Exigences	Qualité des <i>stories</i>	10%
	Trace <i>stories</i> <-> code	5%
Pipeline	Qualité des scénarios d'acceptation	10%
	Qualité du pipeline	15%
	Automatisation du déploiement via Docker	5%
Description	Prise de recul sur le travail effectué	20%

Produit minimal viable attendu pour le 15 mars !

