## 

## Projet tutoré

## Évaluation semi-automatique des projets

Étude préalable

## 

Enseignant responsable : Philippe Dosch

## 

## 

## Table des matières

[Introduction](#h.4d34og8)

[2. Liste des fonctionnalités](#h.ameytbflk4lq)

[3. Cas d’utilisations](#h.spofiyidtcqb)

[Les acteurs](#h.2vrxj7loyye7)

[Vision du produit](#h.p13wg15w45lk)

[Risques liés à notre projet](#h.8jbfg0efkwpi)

## Introduction

À travers l’étude préalable de notre projet, nous avons dégagé diverses fonctionnalités. Néanmoins, nous étions loin d’être parvenus à une liste exhaustive, condition nécessaire afin de mener à bien ce projet. Par ailleurs, les cas d’utilisations, ainsi que certains autres types de diagrammes seront eux aussi une fondation importante pour assurer la bonne structuration de nos développements futurs. Le but de ce document est donc de fournir la liste, qui se veut la plus exhaustive possible, des fonctionnalités, cas d’utilisations, et de certains diagrammes de séquences inhérents à notre projet.

## 

## 

## Liste des fonctionnalités

Notre projet a pour but de corriger de façon semi automatique des projets d'élèves de classe informatiques. Les grandes fonctionnalités de notre application sont donc un outils de décompression automatique , d’analyse plus ou moins poussé selon les critères demandés et une génération visuelle des résultats obtenus.

Après mûres réflexions, nous sommes parvenus à dresser une liste de fonctionnalités exhaustive, que nous allons vous présenter de façon à respecter la chronologie des événements.

**Récupération et décompression**

* **Fournir une hiérarchie de fichiers décompressés ordonnée**

Cette fonctionnalité consiste à fournir une archive comportant l’ensemble des projets étudiants.

* **Générer un fichier avec le nom et prénom des étudiants n’ayant pas rendu le projet**

Ici, nous chercherons à identifier les étudiants qui n’ont pas rendu le projet, ainsi que ceux qui l’ont rendu en retard (si cela est possible).

**Analyse**

On peut regrouper les fonctionnalités de cette partie de façon chronologique

1. **Fonctionnalités applicables sans pré-requis (phase 1) :**

* **Nombre de lignes total**

Nous afficherons ici le nombre total de lignes (vides ou non) du programme.

* **Nombre de lignes de code par fichier**

Ici, nous comptons juste le nombres de ligne de code, c’est-à-dire les lignes strictement nécessaires à la compilation et à l’exécution du programme

* **Nombre de lignes de documentation**

Contrairement à la fonctionnalité précédente, ici nous compterons toutes les lignes nécessaires à la compréhension du programme : commentaires, voire documentation spécifique selon le langage (Javadoc par exemple).

* **\* Nombre de classes / Interfaces**

Nous compterons ici le nombre de classes et d’interfaces du programme fourni par l’étudiant.

* **\* Nombre de méthodes par objet**

Nous compterons ici le nombre de méthode appartenant à un objet.

* **\* Nombre total de méthodes**

Contrairement à la fonctionnalité ci-dessus, nous compterons ici l’ensemble des méthodes du programme de l’étudiant

* **Nombre de lignes dépassant un certain nombre de caractères (« ligne à rallonge »)**

Permets de compter le nombre de ligne trop longue, influent ainsi sur la propreté et la netteté du code source de l’étudiant.

* **Moyenne et écart-type de tous les critères précédents.**

Tous les résultats précédents seraient donc utilisés comme ressource statistique afin de fournir des données claires et intelligibles sur les différences entre les projets des étudiants.

\* Dans le cas de langages où ces concepts existent.

1. **Fonctionnalités exploitables après construction de la ressource principale visée par le projet étudiant (phase 2) :**

* **Informations sur la compilation, l’exécution ou l’interprétation**

Encore une fois tout dépend du langage, mais le correcteur visualisera toutes les sorties, erreurs ou avertissements générés par ces étapes.

* **Liste claire des éventuels avertissements**

Selon le langage, un regroupement sous forme textuelle des

avertissements générés par le compilateur, le programme lui-même ou l’interpréteur.

* **\* Nombre de fichiers générés**

Le nombre de fichiers générés par l’étape de compilation.

\* Pour les langages compilés

1. **Fonctionnalités réalisables après exécution des 2 phases précédentes (phase finale):**

* **Tests unitaires**

Les tests unitaires fournis serviront à l’enseignant en lui indiquant clairement les résultats, au travers des traces d'exécution.

* **Utilisation de la mémoire**

Cette fonctionnalité permettrait de connaître la quantité de mémoire utilisée lors de l’exécution du programme de l’étudiant, nous donnant ainsi des informations relatives sur son degré d’optimisation. Un indicateur pourra aussi être mis en place pour indiquer si oui ou non le programme consomme anormalement de la mémoire.

* **Code coverage**

Ce système permet de connaître le pourcentage de code qui a été couvert par les tests unitaires, information très utile pour le correcteur.

* **Timeout, temps d’exécution**

La mise en place d’un timeout permettrait de donner une indication à l’enseignant. En effet, au-delà d’un certain temps d’exécution, si le programme n’est toujours pas lancé, l’enseignant peut estimer que le programme n’est pas conforme à ce qu’il attendait. Un indicateur pourras alors être défini par l’enseignant. Par exemple : “ X secondes ont eté nécessaire pour le fonctionnement de l’application le programme va indiquer sur la feuille de résultat , “trop long” etc ...

**Génération de la feuille de calcul**

* **Sortir des résultats d’analyse**

Cette fonctionnalité permet de générer une feuille de calculs avec tous les résultats obtenus grâce aux diverses fonctionnalités de l’analyse.(par rapport au indicateurs )

* **Structurer la feuille de calcul**

La feuille de calcul doit présenter clairement les résultats par étudiant ou par projet (suivant la demande de l’enseignant). L’enseignant doit avoir la possibilité de rajouter ses critères manuellement dans la feuille de calcul.

## 3. Cas d’utilisations

### Les acteurs

Définir les acteurs de notre projet va permettre de connaître l’origine des différentes actions et interactions au cours de l’utilisation de l'environnement de correction.

Dans un premier temps, **l'environnement de correction** (le programme) représentera l’un des acteurs centraux. Il sera le coeur des interactions utilisateur.

Le projet à développer s’adressant en premier temps à **l’enseignant correcteur**, il apparaît clairement que ce dernier est l’acteur essentiel du projet.

L’enseignant correcteur devra fournir les données nécessaires à l’analyse, avant de la lancer. Son interaction avec l’environnement de correction ne se limite donc pas à celle d’utilisateur : le système ne peut fonctionner sans données relatives au projet qui lui est soumis. Une telle conception offrira au correcteur une grande liberté pour adapter cette librairie à ses besoins.

Les données a fournir sont :

* Un fichier compressé contenant les projets des éleves.
* Un fichier YML correspondant au projet demandé avec comme indication
  + le langage utilisé
  + la commande de compilation du programme type
  + la commande d'exécution du programme type
  + le chemin de destination des différents projets

Le dernier acteur dont il faudra tenir compte est **le projet informatique.** Une partie des résultats de l’analyse sera la conséquence des actions entre l’environnement de correction et un projet informatique.

**Cas d’utilisation :** Lancement de la correction semi-automatique

Ce cas d’utilisation représente le cas d’utilisation principal, ou l’on va retrouver les actions globales. Dans ce cas d’utilisation, on voit que l’analyse peut seulement être réalisée si l’enseignant a récupéré les projets et s’il a fourni les données nécessaires à l’analyse (langage, tests unitaires …).

pré-conditions :

* Pouvoir exécuter les programmes dans le langage du projet informatique donné (Java, Ruby, C, etc.)

post-conditions :

* Tous les projets informatiques ont été semi-évalués par l’environnement de correction

variantes :

* L’enseignant peut ajouter ses propres critères et à la possibilité de les noter en visualisant les projets.

**Cas d’utilisation :** Fournir les données nécessaires à l’analyse

Ce cas d’utilisation montre avec plus de détails l’étape préparant les données afin de lancer l’analyse.

**Cas d’utilisation :** Analyse du projet

Dans ce cas d’utilisation, l’acteur déclenchant les actions est l’environnement de correction.  
Le cas d’utilisation montre les actions durant la phase d’analyse d’un projet informatique. Dans un premier temps, l’environnement analyse le projet informatique sans aucune interaction ou information de pré-requise. Ensuite, l’environnement construit le programme (compilation , interprétation) afin de l'exécuter. À partir de cette étape, il est possible de compléter l’analyse avec des données extérieures, fournis par l’enseignant correcteur mais aussi en analysant les résultats de la construction du programme (nombre de fichiers, nombre de fichier construits avec réussite …)

pré-conditions :

* Avoir décompressé les projets informatiques
* Pouvoir exécuter les programmes dans le langage du projet informatique donné (Java, Ruby, C, etc.)

post-conditions :

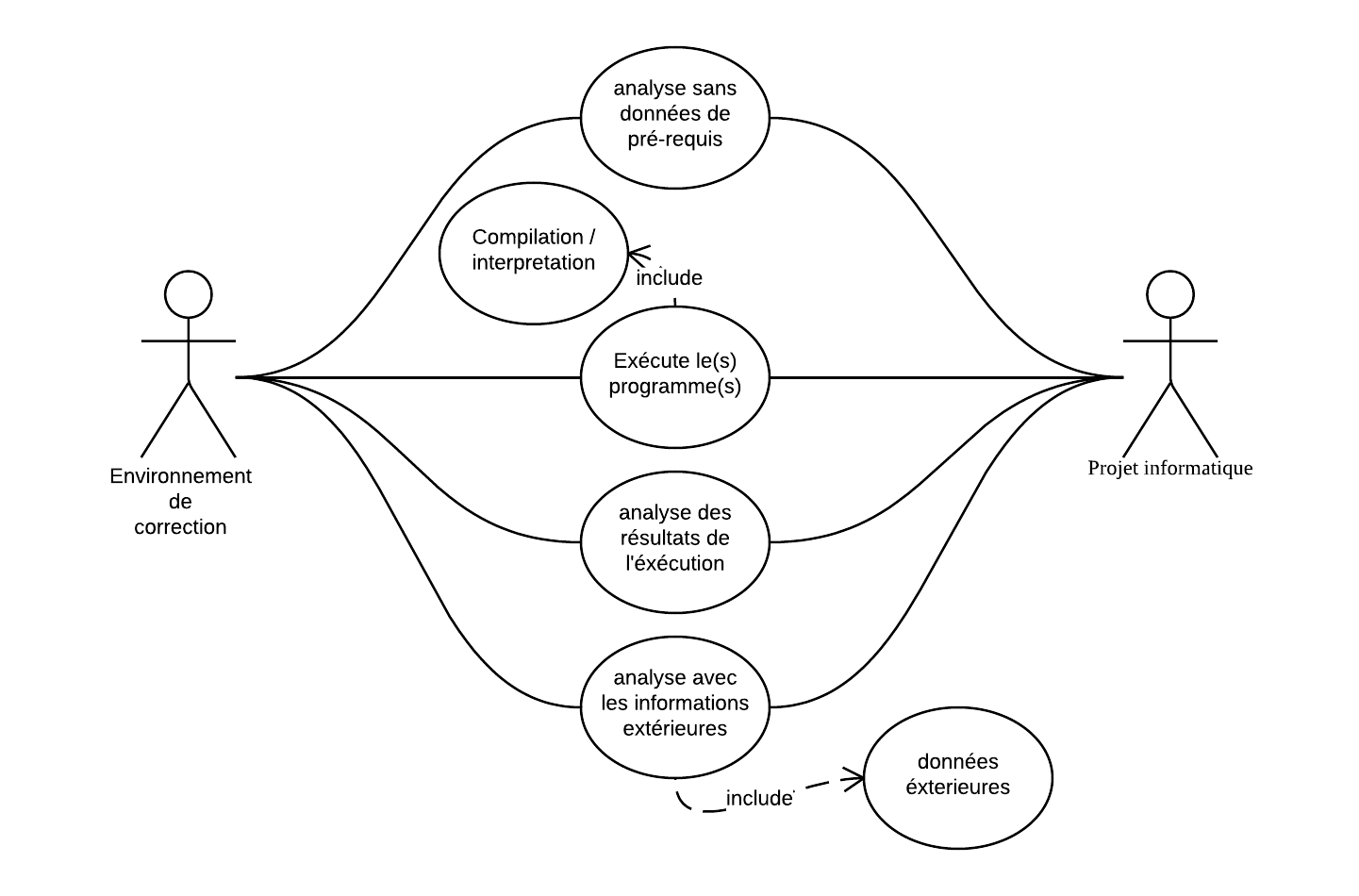
* Les résultats de l’analyse seront disponibles dans la feuille de calcul.

variantes :

* Certains fichiers peuvent ne pas être exécutés (problème compilation/interprétation), le programme doit analyser les autres si possible.

contraintes :

* Il est conseillé que l’enseignant correcteur apporte des données supplémentaires afin d’affiner l’analyse (tests unitaires, timeout, commande pour analyser l’utilisation de la mémoire, etc.)



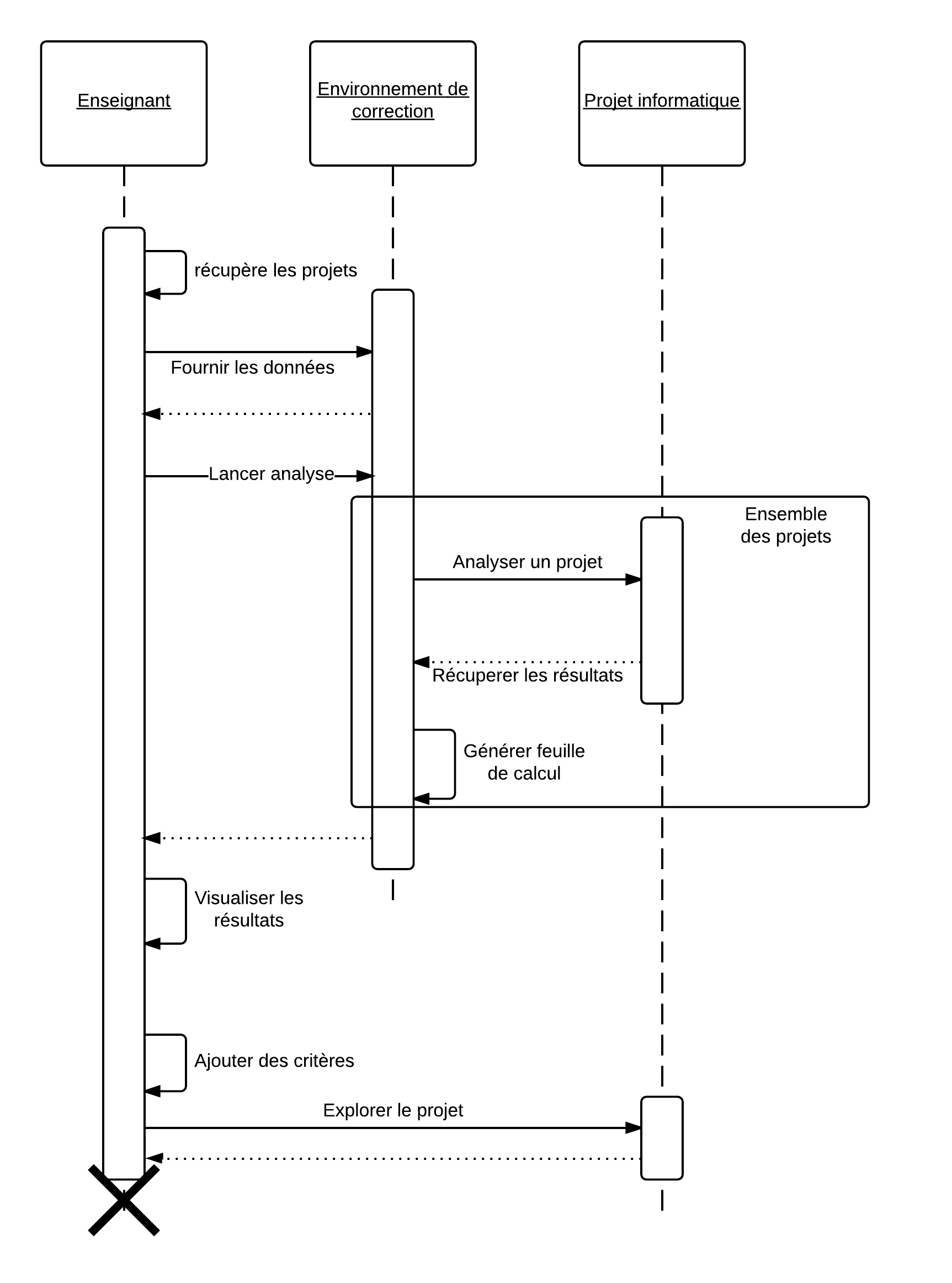
## Diagramme de séquence

Nous avons alors décidé d’effectuer un diagramme de séquence car ce dernier met l’accent sur l’organisation temporelle. De nombreuses notations annexes permettent de préciser la nature des messages (message répétitif, conditionnel, réflexif, récursif, etc.), ainsi que les données véhiculées.

Le diagramme de séquence ci-dessous représente chronologiquement les “grandes” actions entre les différents acteurs de ce projet.

On peut distinguer trois phases en visualisant le diagramme de séquence, la première phase correspond à la phase de préparation, principalement réalisée par l’enseignant.   
Durant cette phase, l’enseignant doit fournir les projets informatiques ainsi que les données nécessaires à l’analyse.  
  
S’en suit la phase d’analyse. Lorsque l’enseignant lance le programme. Ce dernier va analyser un par un les projets informatiques des étudiants. C’est pourquoi, cette partie est une boucle. A noter que dans cette boucle, on y retrouve l’étape de génération de la feuille de calcul, présentant les résultats. Si l’enseignant décide d’interrompre la phase d’analyse, il ne sera pas obligé de recommencer l’analyse pour tous les projets.

La dernière phase du diagramme concerne la visualisation des résultats de l’analyse. L’enseignant à la possibilité de rajouter des critères de notation dans la feuille de calcul ainsi que d’explorer les projets informatiques afin de répondre à ces critères.



## Recensement et évaluation des risques

### Vision du produit

La vision du produit peut être résumée par les réponses à ces 5 questions :

**Qui va acheter le produit ? Qui est la cible ?**

Le projet concerne essentiellement les enseignants de matières informatiques, les professeurs des universités où l'informatique est enseignée.

**A quels besoins le produit va-t-il répondre ?**

Le projet permettra principalement de faciliter la correction des projets informatiques, ainsi que de gagner du temps sur des opérations n'entrant pas en compte dans la notation de ceux-ci. Il permettra à l’utilisateur de limiter au plus les tâches répétitives et de d’avoir une vision aussi bien globale d'individuelle des projets rendus par les étudiants.

**Quelles sont les fonctionnalités critiques pour répondre aux besoins de façon à avoir un produit réussi ?**

Nous avons dégagé deux fonctionnalités critiques :

* La mise en place de la partie analytique du projet.
* La mise en place d’une API permettant à l’utilisateur d’ajouter de nouvelles fonctionnalités selon ses besoins.

**Comment le produit se situe-t-il par rapport aux produits existants sur le marché (parts de marché, points de ventes, etc.) ?**

On retrouve certaines similitudes avec des environnements de correction de C2I, mais aussi avec des logiciels professionnels comme les “ Automated Judge System ”, et les logiciels d’intégration continue. Mis à part ces deux cas, notre produit ne se retrouve pas dans une catégorie de produit existante. Sa spécificité et son adaptabilité sont les principales causes du manque de références sur le marché.

**Quel est le délai et quel est le budget dédié pour le développement et la livraison du produit ?**

Le coût du projet se limite aux salaires mensuels de quatre programmeurs, soit environ 12000 €.

En ce qui concerne le délai dédié au développement de notre projet, nous l’estimons d’environ 4 à 5 mois pour obtenir un programme utilisable.

### Risques liés à notre projet

Les risques liés à notre projet sont principalement liés aux données nominatives fournies au programme. De plus, la sécurité du système est primordiale, car le comportement des programmes peut être inattendu.

Afin de pallier ces risques, il ne faudra pas que le programme conserve les données qui lui sont fournies, et qu’il vérifie le comportement du programme à l’aide d’une sorte de sandbox.

La grandes diversités des projets fait en cours informatique et aussi un risque, pouvant rendre la semi-correction plus ou moins signification selon le langage, projet émis au programme etc..

1. **Première itération**

Ce document montre clairement que l’analyse est la partie centrale de notre projet. Cependant durant la première itération nous avons pensé plus judicieux d'établir un programme triviale mais complet.

En effet notre première itération contiendra les 3 parties du projet final , c’est à dire décompression , analyse , génération de résultats.

Pour ne pas être retrouvé coincé par les délais nous avons définis toutes les fonctionnalités devant être fini pour la fin du cycle. C’est dire :

* Décompression de 3 types de fichier : Zip , tar.gz , rar
* Analyse du nombres de lignes de codes , de commentaire par projets.
* génération d’un feuille de résultats de type CSV.

**6. Deuxième itération**

Notre deuxième itération a consisté à intégrer de nouvelles fonctionnalités à notre programme. En effet la structure étant déjà défini par la première itération l’objectif a été d'intégrer de la consistance à la partie analyse principalement mais aussi les deux autres parties.

En ce qui concerne la partie décompression nous avons intégré :

* décompression de type rar
* décompression de type 7zip

Pour la partie analyse nous avons ajouté les fonctionnalités suivantes :

* compilation automatique des projets et récupération du résultat de compilation
* exécution automatique des projets et récupération des résultats de l'exécution

Pour la partie feuille de résultat nous avons ajouté les fonctionnalités suivantes :

* Intégration du format html et xls en sortie.

De plus nous avons rajouté d’autres fonctionnalités externe aux parties principales (décompression, analyse, génération feuille de résultats).

Nous avons intégré la possibilité de lancer notre programme en ligne de commande, lançable donc n’ importe où sur l’ordinateur.

Il est désormais possible de plus de rendre les dossiers décompressés des différents projets compatible avec git (l[ogiciel de gestion de versions](http://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_versions) [décentralisé](http://fr.wikipedia.org/wiki/Gestion_de_versions#Gestion_de_versions_d.C3.A9centralis.C3.A9e)). Cette fonctionnalité a de nombreux avantages pour faciliter la correction des projets pour les professeurs ( possibilité de modifier les projets tout en revenant à la version non modifié etc.. ) nous l’avons donc jugé essentielle à cette deuxième itération.

**7. Troisième itération**