RAPPORT DE STAGE

Revisiting the Test Runner and DrTest

11 Avril 2022 – 31 Août 2022

Entreprise : Inria de Lille (40 avenue Halley, 59650), France

Tuteur professionnel : Pablo Tesone
Tuteur universitaire : Damien Pollet

Auteur: Aboubacar DIAWARA

Formation : Licence 3 Informatique parcours Info

Résumé

A l'issue de ma 3è année de Licence Informatique, j'ai fait mon stage à **Inria Lille nord Europe** au sein de l'équipe **RMoD**. L'équipe mène des recherches dans le domaine de l'ingénierie logicielle et aide à la remodularisation d'applications orientées objets à l'aide de **Pharo**, un environnement de programmation orienté objet pur.

Durant ce stage, j'ai travaillé sur la bibliothèque de tests unitaires de Pharo (SUnit) en mettant en place une architecture minimale et extensible d'exécution de tests et de collecte des résultats. Par la même occasion, j'ai intégré dans l'outil des fonctionnalités comme l'exécution aléatoire des tests pour la détection de dépendances. Il sera par la suite intégré aux applications qui nécessiteront l'exécution des tests. Enfin pour la suite de mon stage, je dois enrichir la documentation de l'existant et ajouter certains tests unitaires.

Travailler sur ce projet m'a permis de pratiquer les connaissances acquises à l'université, de développer mon sens du travail en équipe, de communiquer de façon professionnelle.

J'ai développé mes compétences sur la pratique du TDD (Test Driven Développement), de l'utilisation de l'outil git … Grâce à Pharo, j'ai eu une compréhension davantage précise du paradigme orienté objet.

Au cours de cette période, j'ai pu gagner en maturité tant sur le plan organisationnel, technique que méthodologique. Fort de cette expérience très enrichissante, J'ai été fortifié dans mon envie de poursuivre mes études dans le génie logiciel et ce fut l'occasion parfaite d'affiner mon projet professionnel.

Internship report summary

At the end of my 3rd year of Computer Science Bachelor's degree, I did my internship at Inria Lille Nord Europe in the RMoD team. Inria is a research center in computer science and automation and RMoD conducts research in the field of software engineering and helps to remodel object-oriented applications using Pharo, a pure object-oriented programming environment.

During this internship, I worked on Pharo's unit test library (SUnit). The idea was to implement a minimal and extensible architecture for the execution of unit tests. At the same time, I integrated in the tool features such as random execution of tests for dependency detection. It will be later integrated into the applications that will require the execution of tests. Finally, I have to enrich the documentation of the existing tool and add some unit tests.

Working on this project allowed me to practice the knowledge I acquired at the university, to develop my sense of teamwork and to communicate in a professional way, to take in hand some collaborative tools at a more sustained level.

I developed my skills on the practice of TDD (Test Driven Development), the use of the git tool. Thanks to Pharo, I had a more precise understanding of the object-oriented paradigm.

During this period, I gain maturity on the organizational, technical and methodological levels. Thanks to this very enriching experience, I was strengthened in my desire to continue my studies in software engineering and it was the perfect opportunity to refine my professional project.

Remerciements

Je tiens à remercier toutes les personnes qui m'ont apporté une aide, de quelque nature qu'elle puisse être pour le bon déroulement de ce stage.

Je remercie l'ensemble des membres de l'équipe **RMoD** pour l'accueil durant cette période.

Plus particulièrement, un grand merci à **Pablo Tesone** mon tuteur de stage et **Stéphane Ducasse** le directeur de recherche de l'équipe pour l'encadrement qu'ils m'ont assuré, pour leur précieux et judicieux conseils prodigués tout au long du stage. Je les remercie également pour leurs confiances témoignées, leurs franchises sans oublier leurs qualités humaines. Qu'ils trouvent ici ma gratitude et ma profonde reconnaissance.

Un grand merci à **Damien Pollet**, mon tuteur académique, pour les précieux et judicieux conseils prodigués tout au long du stage. Je le remercie pour toute sa disponibilité et tout l'effort qu'il a fourni notamment dans la préparation du rapport de stage.

Mes vifs remerciements vont également à l'encontre de **Guillermo Polito** Ingénieur au sein de l'équipe et **Anne Etienne** professeure et membre de l'équipe pour leur disponibilité et la mise à disposition de leur expertise pour les problèmes rencontrés.

A tous mes collègues stagiaires qui m'ont rendu cette période inoubliable, je les adresse mes remerciements pour tous les conseils échangés, les aides apportées pour les difficultés rencontrées.

Je ne saurai terminer sans exprimer ma profonde gratitude à toute l'équipe pédagogique de la Licence Informatique de l'université de Lille pour l'enseignement de qualité et le cadre fournis durant ces trois années de formation.

Impact du numérique sur l'environnement

Téléphone, ordinateur portable ou bureautique, console de jeu, imprimante … j'utilise ces objets au quotidien et rarement je me questionnais sur le fait qu'ils pouvaient avoir une empreinte significative sur l'équilibre de notre planète. C'est pourquoi d'ailleurs, quand les options de l'unité d'enseignement **détermination du projet professionnel** (DPP) du Semestre 6 nous ont été proposées, j'ai été interpellé particulièrement par celle en rapport avec l'impact du numérique sur l'environnement. Je voulais comprendre concrètement l'ampleur de l'empreinte, le numérique qui me semblait un peu trop virtuel, et les conséquences à long et à court terme que cela pouvait avoir sur nos vies.

Au cours de la fresque, j'ai pu participer à une activité en groupe de 6 étudiants sous la supervision de madame **Virginie Cogez** centré sur la compréhension de l'impact et des mesures personnelles que chacun peut prendre pour le réduire au mieux.

A l'issue de cette fresque, j'ai appris à repenser l'utilisation que je fais des outils numériques au quotidien par ce que chaque action compte. Je n'hésite pas à discuter, à chaque fois que l'occasion se présente, avec des personnes autour de moi qui comme qui comme moi auparavant ne se doute pas que toute la virtualité du numérique repose sur des infrastructures physiques qui ont besoin parfois très conséquent en termes d'électricité. Sans oublier leur empreinte carbone durant leur cycle de vie (extraction de ressources premières, fabrication, acheminement, déchets produits…). Il me parait difficile d'en avoir conscience sans une réflexion sérieuse.

Les entreprises aussi bien que les individus sont tous concernés pour une planète plus habitable. C'est pourquoi je me suis intéressé à la politique environnementale du centre de recherche **Inria** qui m'a accueilli pour mon stage de fin de Licence**.**

C'est tout d'abord un travail de sensibilisation, à travers un MOOC intitulé, Inria s'engage à sensibiliser le grand publique des effets du cout des usages du numérique sur notre planète.

Quant à l'équipe de recherche **RMoD**, elle essaye au mieux de réduire les déplacements de ses chercheurs en privilégiant les réunions à distance. En plus les membres se déplacent le plus possible à l'aide de transports en communs et de bicyclettes [1].

Son objet principalement de recherche qui consiste à remodulariser les applications orientées objet est très avantageux dans le sens où on privilégie le recyclage. Cela évite d'aller réinventer de nouveaux logiciels ce qui pourrait être davantage coûteux.

Sommaire

1. Introduction

Les logiciels informatiques sont omniprésents dans notre quotidien. Des téléphones portables jusqu'aux navettes spatiales en passant par l'industrie médicale, on le retrouve partout par conséquent ses enjeux sont énormes. Conséquemment, il est indispensable de s'assurer que de tels outils répondent avec précision et sûreté au besoin.
Il devient alors évident pourquoi le logiciel passe par plusieurs phases de tests avant le déploiement.

Les développeurs utilisent le Test Driven Development pour s'assurer que leur programme répond bien aux spécifications. Cette méthode itérative leur permet de détecter tout comportement inattendu le plus rapidement possible lors de la phase de développement. En Pharo, la bibliothèque SUnit permet d'écrire des tests unitaires. Son architecture actuelle ne permet pas de modifier l'ordre d'exécution des tests. Les tests sont plus utiles s'ils sont indépendants les uns des autres. Donc on souhaite détecter si des dépendances ont été introduites entre tests parce que cela diminue leur utilité. Mon stage au sein de l'équipe a consisté essentiellement à améliorer certains aspects de la bibliothèque en fournissant un outil extensible pour l'exécution des tests, en gérant certaines responsabilités au sein des classes, en enrichissant la documentation …).

Dans un premier temps, je ferai une présentation du centre de recherche Inria ainsi que l'équipe **RMoD** qui m'accueille pour ce stage. Ensuite je présenterai les contextes général et fonctionnel du projet, les technologies utilisées ainsi que les solutions apportées. J'en profiterai pour détailler quelques activités auxquelles j'ai participé durant ce stage et qui m'ont paru très intéressante.

2. Contexte

2.1. A propos d'Inria

Inria (institut national de rechercher en informatique et automatique) est un organisme de recherche français sans but lucratif spécialisé en Mathématiques et Informatique. Il a 9 composantes, en excluant son siège situé à Rocquencourt reparties en France (Centre Inria de l'Université de Bordeaux, Centre Inria Grenoble Rhône Alpes, Centre Inria de L'université de Lille, Centre Inria de Lyon, Centre Inria Nancy-Grand Est, Centre Inria de Paris, Centre Inria Rennes-Bretagne atlantique, Centre Inria d'université Côte-d'Azur, Centre Inria de Saclay, Inria Chile).

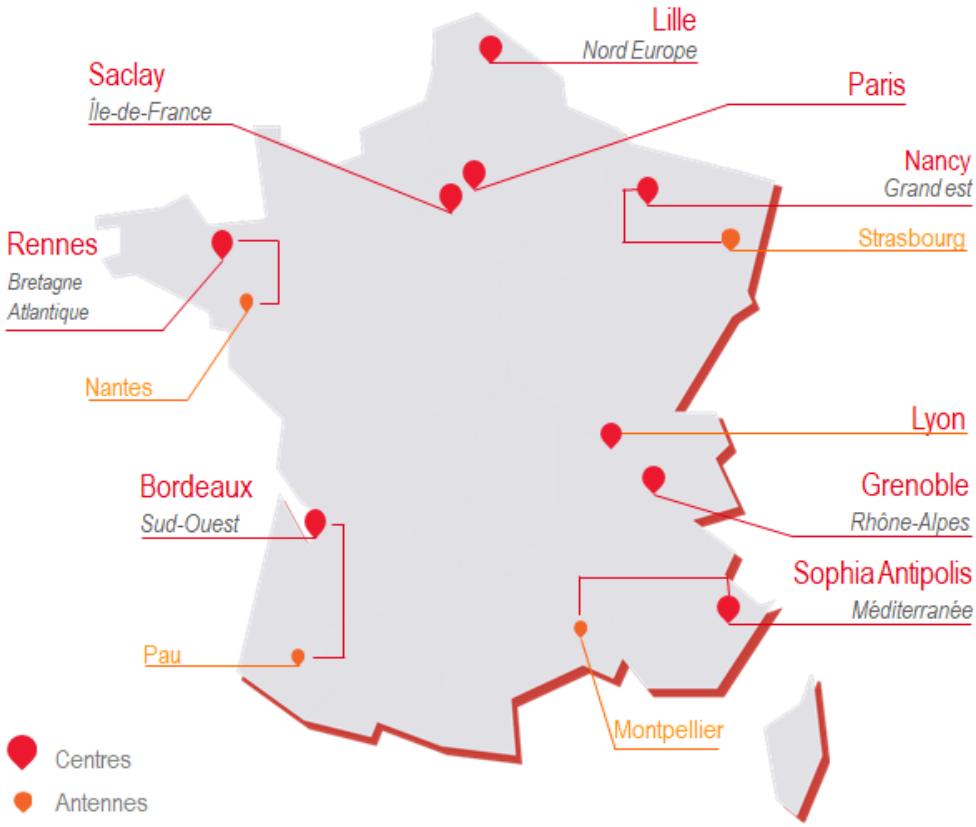


Figure 1: Composants d'Inria en France

Il compte près de 3900 scientifiques, 200 équipes de recherches. Depuis 1984 il a permis la création de près de 200 startups.

1.1. Activité d'Inria

Inria est l'institut national de la recherche en sciences technologiques et numériques en France, et l'entreprenariat font également parti de ses plans d'action. Il est également tourné

vers l'Europe et plus important encore vers l'international avec 100 équipes associées active [3].

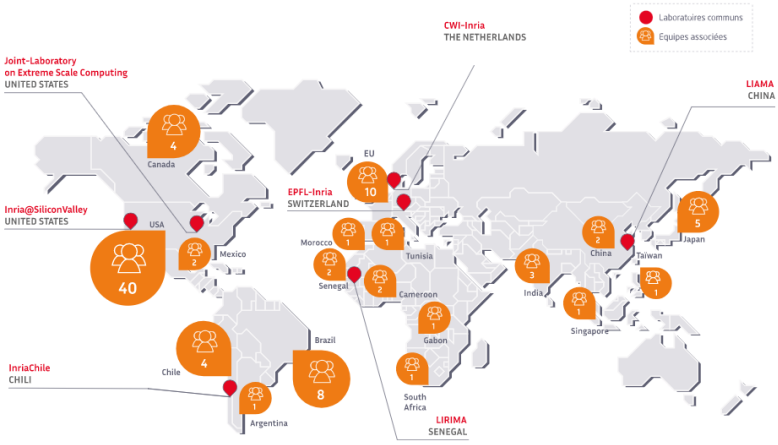


Figure 2, les implantations d'Inria dans le monde

1.2. A propos d'Inria de Lille.

Créé en 2008, Inria de Lille nord Europe est un des 9 composants d'Inria. Il est présent sur 2 sites :

- La Haute-Borne, au cœur du campus cité scientifique de l'université de Lille.
- EuraTechnologies.

Il comporte 15 équipes projets, et a participé depuis ses débuts à créer 10 startups. Les recherches menées sont principalement orientées vers la science des données, le génie logiciel, les systèmes cyber physiques.

1.3. L'équipe RMoD

L'équipe **RMoD** est l'une des 15 équipes du centre de recherche Inria de Lille Nord Europe. Ayant pour responsable **Stéphane DUCASSE**, elle travaille dans le domaine de l'ingénierie logicielle sur le site de la haute borne. Elle est actuellement Composée d'environ composée de 3 chercheurs, 2 maitres de conférences, des ingénieurs, doctorants, stagiaires. Son principal objectif est l'analyse et la construction de langage pour l'évolution d'applications orientées objet. A l'aide la technologie Pharo, un langage pur orienté objet descendant de Smalltalk, elle aide à remodulariser les logiciels orientés objet [2].

1.4. Le sujet de stage, revisiting the Test Runner and DrTest

Le stage est une partie du thème `Better Tools` qui consiste à faire une révision de certains outils de l'environnement Pharo. Le sujet du stage `Revisiting the Test Runner and DrTest` consiste à revoir de façon générale la manière dont SUnit exécute les tests.

Pour éviter toute ambiguïté dans la suite de report, chaque utilisation du terme **TestRunner** fait référence à l'outil graphique d'exécution des tests de l'environnement Pharo (voir annexe1). Quant à NewTestRunner, c'est l'outil dont que j'ai implémenté durant ce stage.

En particulier, le but est d'améliorer les tests unitaires de la bibliothèque et d'introduire une classe NewTestRunner qui permet d'exécuter des tests. Ce dernier sera utilisé comme un outil central partout où le besoin d'exécuter des tests se ressentira.

Actuellement, la bibliothèque SUnit est bien fonctionnelle. Elle est utilisée pour rédiger tous les tests unitaires dans Pharo. Etant appelée à évoluer, et ayant des outils qui se développe de plus en plus autour, il serait judicieux de bien encadré cette évolution.

1.5. Missions accomplies durant le stage

Durant ce stage, j'ai dans un premier temps eu une période de formation sur l'ensemble des outils concernant ce sujet. La compréhension de l'architecture d'exécution des tests, l'apprentissage de Spec2 qui est une bibliothèque pour la construction d'interfaces graphiques.

Aussi j'ai appris les bonnes pratique et conventions adoptées par la communauté Pharo concernant la qualité du code (règles de nommage, commentaires, opérations couteuses …) à travers le livre `Pharo with style`.

Ensuite j'ai implémenté l'architecture du NewTestRunner conformément au besoin. Pour la suite de mon stage, je vais l'intégrer dans les outils d'exécution des tests et l'étendre pour implémenter d'autres outils comme le ServerTestRunner.

Parallèlement à mon projet, j'ai eu l'occasion de participer à des activités de programmations. Toute l'équipe est impliquée et elles se déroulent en pair-programming. C'est l'occasion d'apprendre surtout à côté de nos tuteurs de leur manière de programmer. Principalement, deux activités ont eu lieu :

- Pharo Sprint :

  Pharo Sprint est un événement organisé à la fin de chaque mois, au cours duquel toute la communauté est invitée pour la correction des bugs de l'environnement. Il se déroule en pair-programming (un débutant et un développeur expérimenté). C'était l'occasion pour d'observer et d'apprendre auprès de personnes aguerries.

- Kata :

  C’est un exercice de programmation en groupe. Par équipe de deux, on continue la tâche de l’équipe précédente et ainsi de suite. Le but est de résoudre un problème qui sera posé à tous en début de séance. Les passages étant chronométré, ça développe la capacité à travailler sous pression, le sens de l’écoute, et du travail en équipe.

2. Contribution

2.1. Problématique : créer une architecture performante d'exécution de tests

Tester est un processus très important quand on développe une fonctionnalité. Pour l'équipe RMoD, c'est plus qu'une nécessité. C'est pourquoi il est important d'avoir de bons outils à disposition pour effectuer cette tâche dans les meilleures conditions possibles.

Comme chaque année, l'équipe RMoD a décidé de proposer un stage pour travailler sur certains aspects de la bibliothèque SUnit. Celle-ci permet en effet d'écrire des tests unitaires. Plusieurs outils ont déjà été construits autour comme :

- CommandLineTestRunner : Un programme qui permet depuis une ligne de commande d'exécuter des tests et d'observer leur résultat. Celui-là est particulièrement utile quand on travaille sur une machine qui n'a pas de serveur graphique. Celle-ci en conséquence ne pourra pas afficher des fenêtres.
- TestRunner : Au même niveau que `CommandLineTestRunner`, celui permet d'exécuter des tests. Il fournit une interface graphique et d'autres fonctionnalités supplémentaires comme le calcul de la couverture de tests, le profilage (analyse de la durée d'exécution)

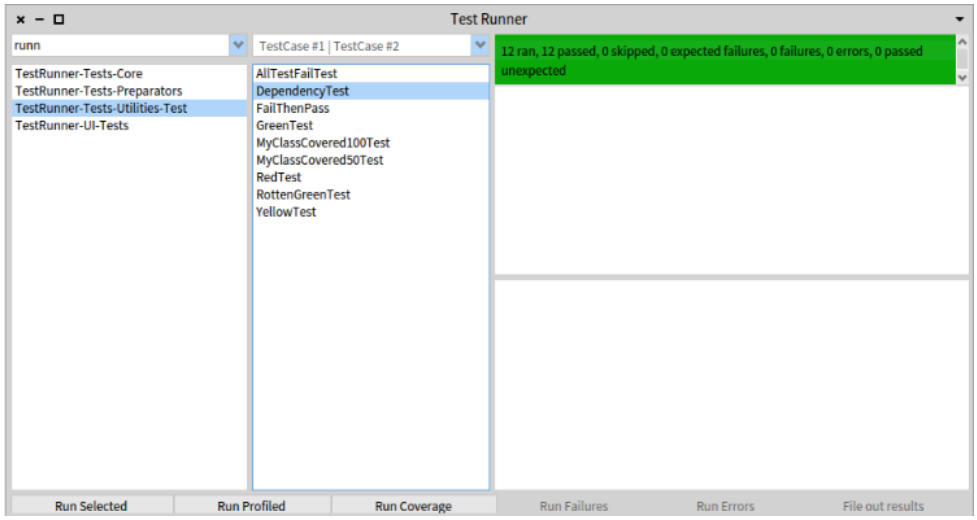


Figure 3: Test Runner

- DrTests : Ce dernier est plus personnalisable, fonctionne à l'aide de plugins.

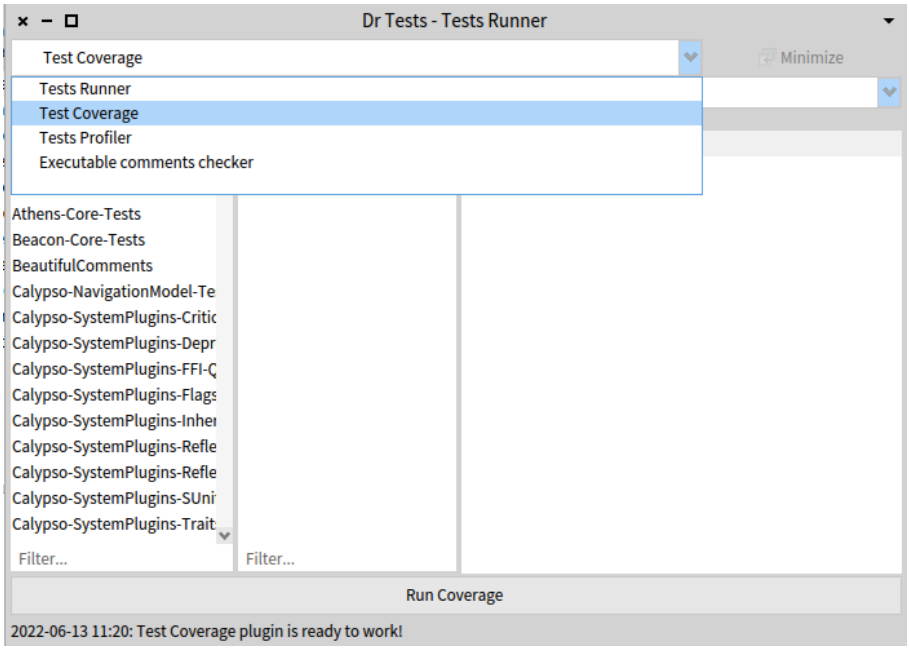


Figure 4: Dr Tests

Par contre, un problème se posait. En effet chacun de ses outils réécrivait son moteur d'exécution des tests. Le TestRunner actuel est une interface graphique, ce qui rend impossible son utilisation sans interface graphique ou encore son utilisation dans d'autres outils pour exécuter les tests en plus il a été écrit sans séparer les responsabilités (la partie graphique et la partie logique se mêlent).

Des lors, on s'est fixé comme objectif de créer un moteur d'exécution de tests extensible et pouvant être utilisé par tous ces outils. Pour mener à bien ce projet, j'ai été formé aux technologies que je serai amené à utiliser dès les premiers jours.

2.2. Technologies utilisées

Tout au long du stage, j'ai travaillé dans l'environnement Pharo et pour la collaboration et le suivi de ma progression, j'ai utilisé la plateforme Github qui héberge d'ailleurs le projet Pharo.

2.2.1. Pharo

Largement inspiré par Smalltalk, Pharo est un langage pur orienté objet sous licence MIT. Il est multiplateforme (Mac OS X, Windows, iOS, Android, Linux) car basé sur une machine qui est principalement écrite en Pharo lui-même. Sa première version a été publiée le 24 Avril 2008, par Stéphane Ducasse et Marcus Denker qui en sont les créateurs.

Figure 5 logo pharo

A ce jour, il est à sa version 11. Il possède les caractéristiques suivantes qu'il hérite de Smalltalk :

- Il est réflexif : Cette capacité d'introspection se définit par la capacité qu'à une entité d'observer et de modifier sa propre structure et son comportement en cours même de son exécution.
- Il a un typage dynamique : aucun besoin de définir le type des variables. Celles-ci peuvent en conséquence contenir n'importe quel type d'objets sans restriction.
- Une classe n'hérite exactement que d'une seule classe. L'héritage multiple n'existe pas. Par ailleurs, le concept de Trait [4] existe. Un Trait est un fragment de class, un modèle conceptuel pour structurer des programmes orientés objets.

2.2.2. SUnit

SUnit est une bibliothèque de tests unitaires de l'environnement Pharo. C'est aussi celui de Smalltalk (Pharo est un descendant de Smalltalk). Il est très simple et cela s'illustre par le fait qu'il n'a à la base que 4 classes [5] que nous découvrirons bientôt. Beaucoup de bibliothèques de tests unitaires d'autres langages de programmation s'en inspirent comme libcbdd (C), JUnit (java), CPUnit (C++) pour ne citer que celles-ci.

Durant mes premiers jours, j'ai étudié l'architecture actuelle de la bibliothèque pour comprendre comment les tests sont collectés puis exécutés.

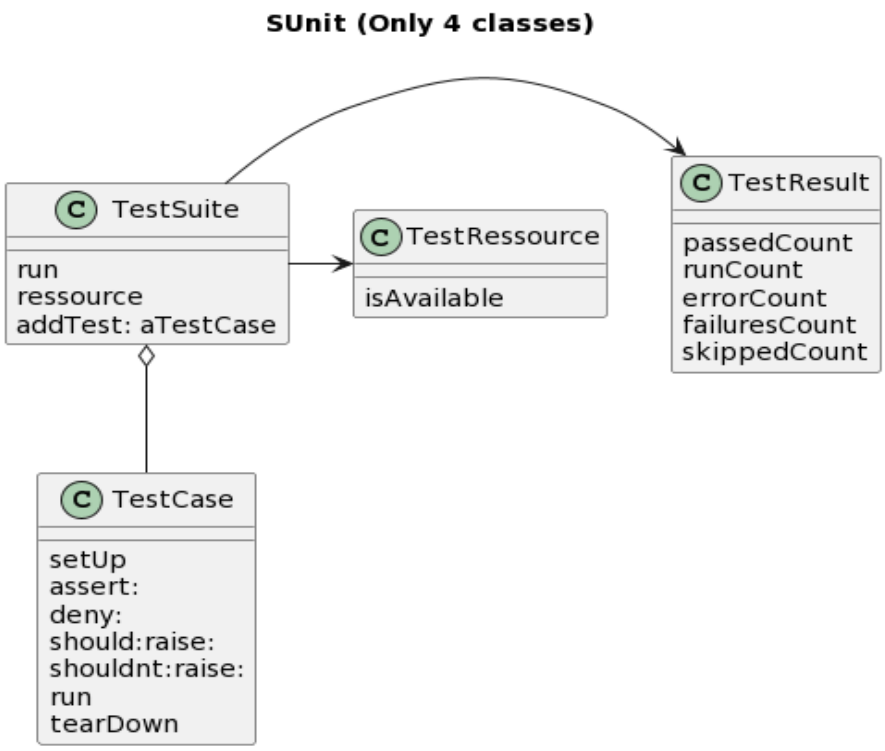


Figure 6: SUnit Framework

2.2.2.1. TestCase

Une classe de tests permet de définir des méthodes de tests. Une classe de test se distingue des autres objets par le fait qu'on retrouve la classe TestCase dans sa hiérarchie. TestCase est la classe qui définit le comportement minimal de toute classe de tests. Donc elle doit impérativement hériter directement ou indirectement de cette class. Par défaut, le nom d'une méthode de tests dans Pharo commence par 'test' (exemple : testAdditionIsAssociative). Mais ce critère de nommage peut être redéfinie par le framework, on garde tout de même à l'esprit que cela reste une convention.

2.2.2.2. TestAsserter

Cette classe définit toutes les méthodes d'assertions. C'est grâce à elles qu'on teste les comportements. Une instruction doit -elle être vraie ? doit-elle lever une exception, l'évaluation d'une expression doit-elle égale à une autre ? Toutes ces propositions peuvent être vérifiées à l'aide des méthodes de cette classe.

2.2.2.3. TestResult

Lors de l'exécution d'une méthode de tests, les situations suivantes peuvent subvenir :

- Succès : Le test passe avec succès, toutes les assertions étant vérifiées.

- Echec : Dans le cas contraire, car les conditions testées ne sont pas satisfaites.
- Erreur : Une erreur de syntaxe, l'utilisation d'un objet inexistant dans l'environnement.
- …

Quand on exécute une classe de tests, on se retrouve avec des ensembles de tests qui ont échoués (par erreur ou pour spécification), réussis, ignorés etc. Le tout est organisé dans un objet TestResult pour mieux interagir avec l'ensemble et l'exploiter.

2.2.2.4. TestRessource

Des tests peuvent partager des ressources, par exemple l'accès à une base de données.
Un objet TestRessource modélise une telle ressource et gère sa disponibilité.

2.2.2.5. TestSuite

Un objet de type TestSuite n'est rien de plus qu'un objet qui gère un ensemble de tests. L'ensemble des tests dans une classe de tests peuvent constituer un TestSuite. En plus de gérer cela, il gère les ressources partagées entre tous les tests.

2.2.3. Spec2

Spec est une bibliothèque pour le développement d'interface graphique dans Pharo. L'une de ses particularités est qu'elle est testable. La plupart des interfaces de Pharo ont été construites à partie d'elle.

2.2.4. Blueprint Project [6]

Encore en cours de développement, ce projet m'a permis d'étudier de façon aisée l'existant (la bibliothèque SUnit), d'en comprendre l'architecture grâce à sa manière de présenter pour une classe donnée, la relation entre les méthodes, les variables tout en les catégorisant (méthodes mortes, méthodes du coté class, variable de class…) et en indiquant leur fréquence d'utilisation, cela nous permet de vite cerner les méthodes/variables les plus utilisées.

2.3. Analyse du besoin

Avant tout début de développement, j'ai eu une discussion avec **Stéphane Ducasse** le directeur de recherche et **Pablo Tesone** mon tuteur de stage dans le but de bien cerner le problème en jeu. A partir de l'outil Kanban de Github, ils me laissaient une liste de ressources pour démarrer le projet dans de bonnes conditions. La figure suivante illustre un exemple d'utilisation du NewTestRunner. Il sera utilisé pour d'autres fins également comme pour la réécriture de l'actuel TestRunner pour séparer la logique de l'exécution et celle de la présentation. Par la suite, je serai amené à améliorer la documentation du cadre de travail

SUnit, en complétant les commentaires des classes et méthodes, et écrire des tests sur certaines parties qui n'ont pas une grande couverture de tests.

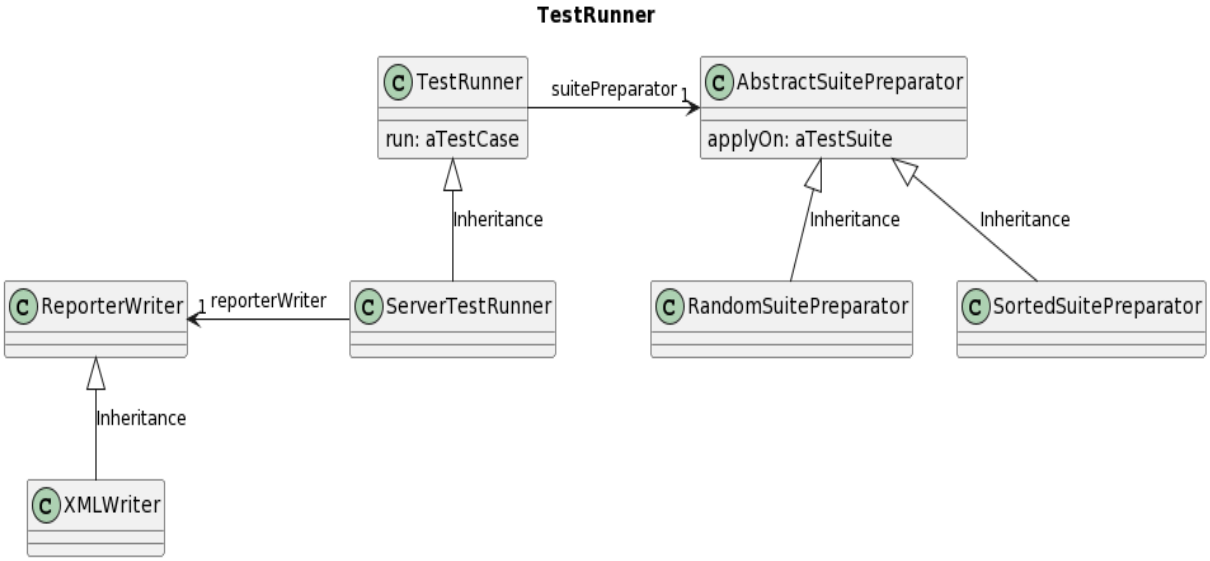


Figure 7: exemple utilisation du NewTestRunner

Pour implémenter cette solution, j'ai été sous la supervision de mon tuteur de stage **Pablo Tesone** et **Stéphane Ducasse** à qui je faisais des mises au point tout au long du stage. C'était l'occasion d'adapter les besoins et de proposer des corrections à faire.
Aussi, ils n'hésitaient pas en fonction de leur besoin à me suggérer des détails à prendre en compte.

2.4. Résultat obtenu

Après avoir passé un moment sur l'étude pour la prise en mains des outils et technologies cités précédemment, j'ai commencé la partie développement.

J'ai adopté une approche **TDD** (Tests Driven Development) lors de la programmation, Pharo étant un environnement qui supporte l'extrême TDD.

J'ai commencé à écrire des tests sur le futur NewTestRunner pour décrire son comportement. Par exemple comment se fera l'exécution d'un TestCase, d'un package de tests, d'une collection de tests ou encore quoi faire si au lieu d'une classe de tests on lui donnait une classe qui n'hérite pas de tests case ? faut-il soulever une exception ? faut-il renvoyer un résultat vide ?

A ce stade de mon stage, j'ai implémenté le NewTestRunner, et progressivement je l'enrichie de certaines fonctionnalités à chaque occasion. Il est d'ailleurs intégré dans la version 11 de Pharo.

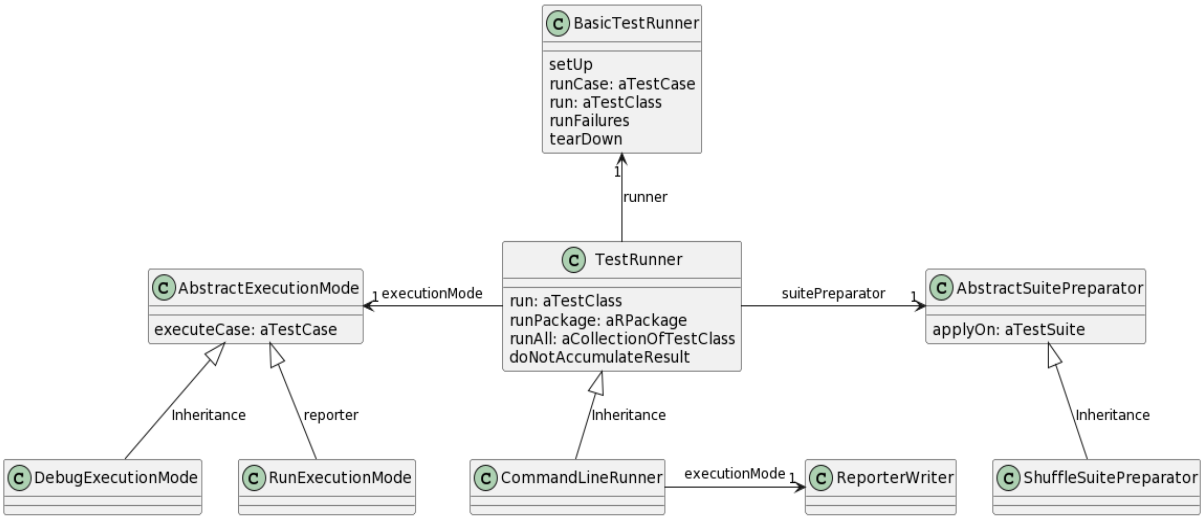


Figure 8: NewTestRunner

BasicTestRunner est construit le plus simplement possible. Il est capable d'exécuter une classe de tests, un TestCase. Le but pour cela est de préparer le terrain pour une éventuelle refactorisation sur certaines classes de la bibliothèque SUnit. Pour être plus précis, si on consulte la classe TestResult, qui est sensée contenir le résultat des tests, on voit dans son API :

```
TestResult >> runCase : aTestCase
```

J'ai donc totalement donné cette responsabilité à BasicTestRunner. D'autres responsabilités sont à voir par la suite s'il a lieu de les déplacer vers BasicTestRunner.

NewTestRunner est donc utilisé comme une sorte de Décorateur (Patron de conception) sur BasicTestRunner. Il est basé sur les fonctionnalités de cette dernière. Contrairement à BasicTestRunner, il est capable d'exécuter non seulement un seul test à la fois, mais aussi des tests à partir d'une collection, à partie d'un package, et par la suite j'envisage d'étudier la possibilité d'exécuter des tests depuis une image. Le but de ce choix est de rendre NewTestRunner constant aux changements qui seront opérés sur BasicTestRunner. Il lui prête juste son API pour exécuter de TestCase.

A s'en tenir aux tests unitaires, le NewTestRunner répond bien au comportement attendu. Il est capable d'exécuter toute une classe, un seul testCase, un testSuite, plus loin des tests dans un package et mieux encore tous les tests de l'image (annexe3). Par curiosité, j'ai créé une interface graphique (figure suivante), qu'on appellera **GuiTestRunner,** qui utilise la logique du NewTestRunner pour l'exécution et la collecte de tests. Elle est basée sur Spec2 et présente des fonctionnalités détaillées dans les lignes suivantes.

Je prévois par la suite d'enrichir le NewTestRunner, en ajoutant la possibilité d'exporter le résultat de l'exécution dans un format voulu (XML par exemple).

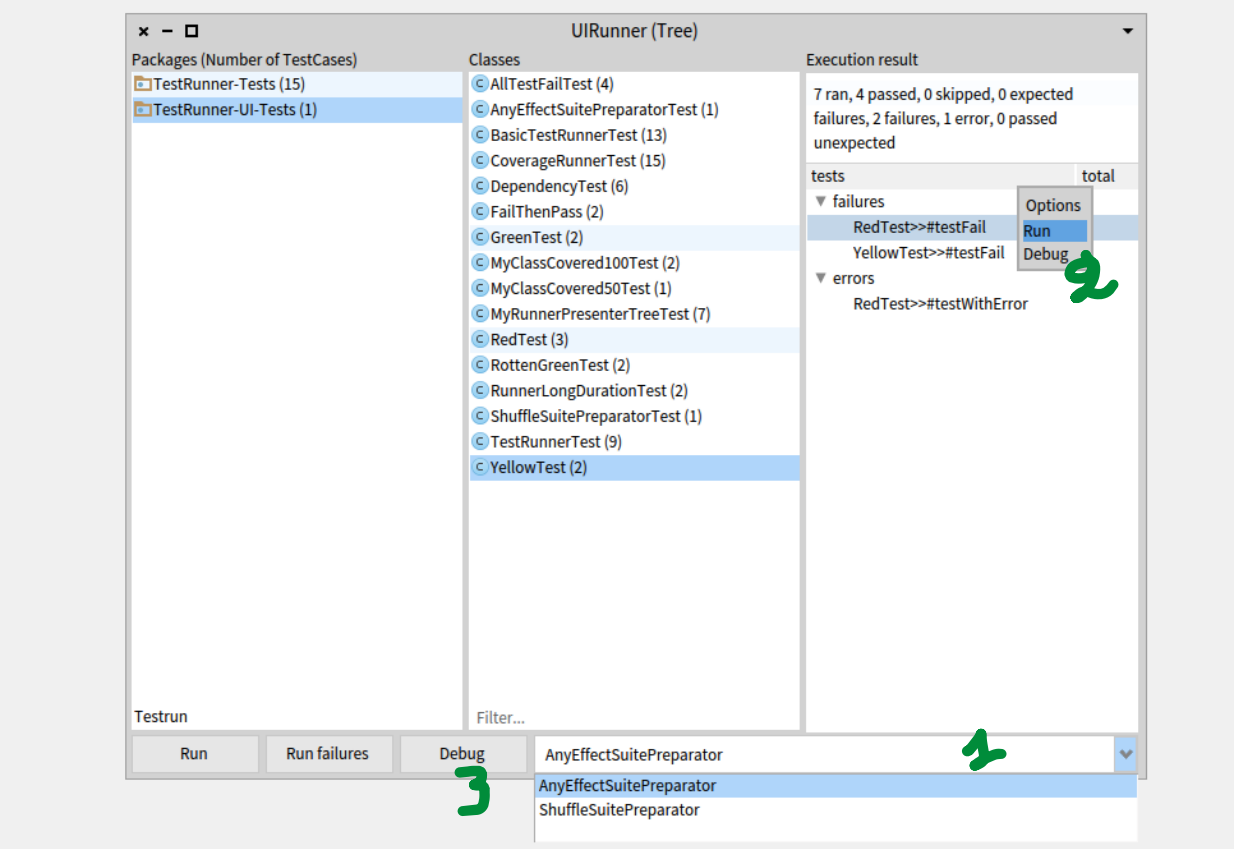


Figure 9, GuiTestRunner

Cette nouvelle interface ne couvre pas encore toutes les fonctionnalités du TestRunner actuellement en place. Il pourrait être considéré comme un complément dans le sens où :

- On peut ouvrir un menu contextuel (numéro 2) pour choisir entre débugger ou réexécuter un test parmi l'ensemble de ceux qui ont échoués.
- On peut choisir, à l'aide d'une liste déroulante (numéro 1), entre une exécution normale ou une exécution aléatoire des tests. Cela pourrait être utile pour une détection de dépendances entre tests. Avec cette approche statistique, on n'est pas certain de détecter des dépendances mais il y'a tout de même des chances qu'on tombe à un moment sur un ordre qui fait échouer les tests.
- On peut exécuter des tests de sorte :
    - Qu'on liste suivant des catégories (failures, passed, errors) les résultats de tests
    - Qu'on ait directement le debugger qui s'ouvre sur le premier échec (numéro 3). Cela peut être utile quand on des tests qui entretiennent des dépendances les uns des autres et modifie les mêmes données, on voit exactement le contexte qui a mené à l'échec alors qu'avec une réexécution on aurait pu la perdre.

Aussi, elle a juste précisé la manière de présenter les résultats (le bloc tests), sa super class étant abstraite et décrivant globalement la logique de l'exécution, l'exécution des tests échoués, le débogage. On pourrait par exemple imaginer une seconde interface qui au lieu de lister sous forme d'arborescences les liste par ligne ou par colonne.

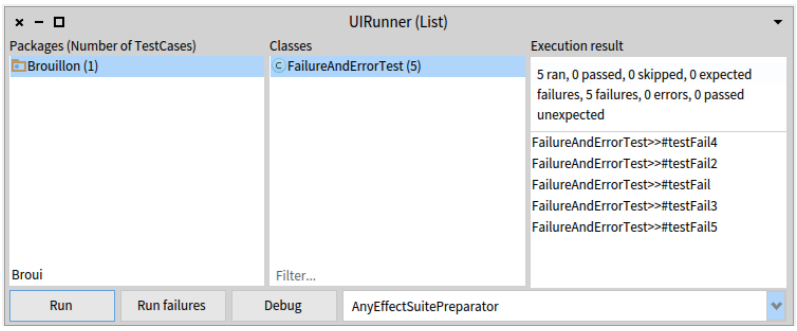


Figure 10, Résultat des tests par ligne

Précédemment, j'ai souligné que Spec2 était une bibliothèque testable. Cela m'a parmi d'écrire des tests unitaires sur le comportement de l'interface graphique. Elle est appelée à évoluer, cela me permettra de remarquer toute régression le plus vite possible.

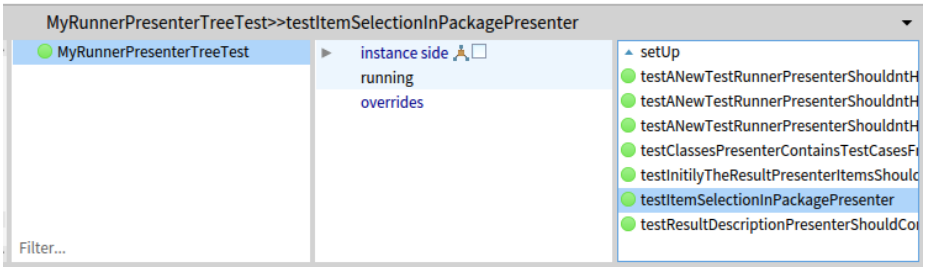


Figure 11, tests unitaires pour UI

Pour comparer illustrer l'utilité du NewTestRunner, j'ai créé une classe dont les méthodes de tests entretiennent une dépendance.

Pour cela, j'ai créé une variable de classe qui est initialisé avec la valeur 10.

Le 1er test vérifie que la variable est égale à 10 puis l'incrémente (donc elle devient 11) ;

Le 2e tests vérifie que la valeur de la variable est 11 puis l'incrémente. Ainsi de suite.

Voyons une comparaison de l'exécution de ce test par les deux outils à savoir GuiTestRunner et TestRunner. Gardons à l'esprit que GuiTestRunner est basé sur NewTestRunner. Cela lui donne la possibilité de choisir les opérations à appliquer sur le testSuite)

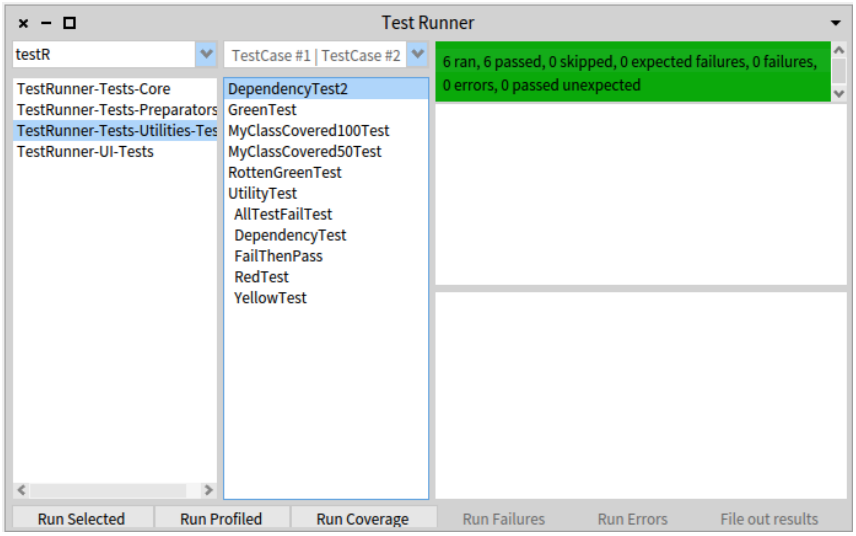


Figure 12 (TestRunner) Tous les tests passent peu importe le nombre d'exécution

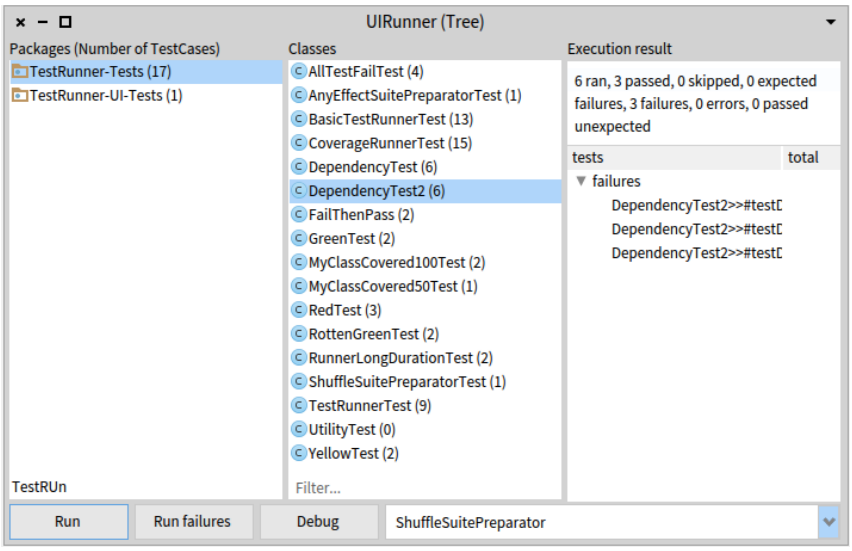


Figure 13, la randomisation fait qu'on obtient parfois un ordre qui fait échouer le test

2.5. Difficultés rencontrées

De façon générale, le stage se déroule sans difficultés majeures. Les premiers jours j'ai eu l'impression d'avoir un manque d'efficacités. Cela résultait du fait qu'en plus de mal découpé le travail, j'avais de grosses difficultés à fixer des délais. Pire encore, sur chaque j'avais envie d'améliorer continuellement encore et encore. Aussi J'ai voulu apprendre tellement vite pour m'attaquer au projet que ça m'a ralenti par la suite. S'il y'a une chose que j'ai retenu c'est surtout l'utilité de prendre du recul, faire preuve de patience et plus important encore, comprendre le besoin donner des limites à la solution et se fixer des délais.

3. Conclusion

Pour conclure, je réitère mes remerciements à toute l'équipe **RMoD** qui m'a fourni un cadre idéal pour effectuer mon stage. Ce stage a été pour moi une opportunité de pratiquer mes connaissances, quelque fois de remise en cause, d'apprentissage de nouvelles choses et surtout ça a m'a permis d'assimiler les enjeux d'un bon sens de l'organisation et de le développer, cerner l'importance de l'écoute et de la compréhension des attentes.

Mon implémentation du NewTestRunner pourra désormais être utilisé pour réécrire les outils comme (Le CommandLineRunner, le TestRunner...) ou encore pour implémenter un CoverageRunner ou un ServerRunner, cela fera d'ailleurs l'objet de la suite de mon stage.

Il répond bien à la problématique de l'exécution aléatoire et possède la responsabilité qu'avait le TestResult pour l'exécution des tests. Aussi il est suffisamment pour permettre la construction d'un CoverageRunner et d'un CommandLineRunner.

Grâce à ce projet, j'ai beaucoup appris sur les tests unitaires et la bibliothèque SUnit. Cela me semble important d'autant que la plupart des bibliothèques actuelles s'en inspirent.

Réaliser ce stage, a été une belle aventure pour moi. Fort de cette expérience, cela m'a fortifié dans ma position de poursuivre mes études de Master vers l'ingénierie logicielle.

4. Bilan

Ce stage a été un réel tremplin pour mon projet d'étude dans le sens où j'ai compris que j'avais une appétence pour le développement informatique. Ça m'a donné envie de poursuivre et d'acquérir des connaissances poussées dans les techniques de conception logiciel Informatique. Personnellement j'ai eu un aperçu et pris gout au travail en équipe, pris conscience de l'importance d'une adaptation rapide dans le processus d'apprentissage.

J'ai découvert ce qu'est le métier de chercheur dans le génie logiciel en particulier,

Par ailleurs, ayant envisagé depuis un moment de poursuivre jusqu'en doctorat, j'ai eu l'occasion de rencontrer et discuter avec des doctorants ce qui m'a permis d'affiner davantage mon projet et de répondre à certains questionnements que j'avais. J'ai notamment apprécié travailler sur un projet open source autour duquel il y'a une grande communauté.

La langue principale au sein de l'équipe étant l'anglais, j'ai eu l'occasion de m'exercer au quotidien et sur le plan écrit et sur le plan oral.

5. Bibliographie

[1] **Marcus Denker, Nicolas Anquetil, Vincent Aranega, Steven Costiou, Stéphane Ducasse, Anne Etien.** ProjectTeam RMOD 2021 Activity Report. [Research Report] INRIA Lille - Nord Europe. 2022. ffhal03629450f. *hal.* [En ligne] https://hal.inria.fr/hal-03629450/document.

[2] **Inria.** inria-partenariats-internationaux-numerique. *https://www.inria.fr/.* [En ligne] 13 06 2022. https://www.inria.fr/fr/inria-partenariats-internationaux-numerique.

[3] **pharo, files.** media/logo/logo.png. *https://files.pharo.org/.* [En ligne] 10 06 2022. https://files.pharo.org/media/logo/logo.png.

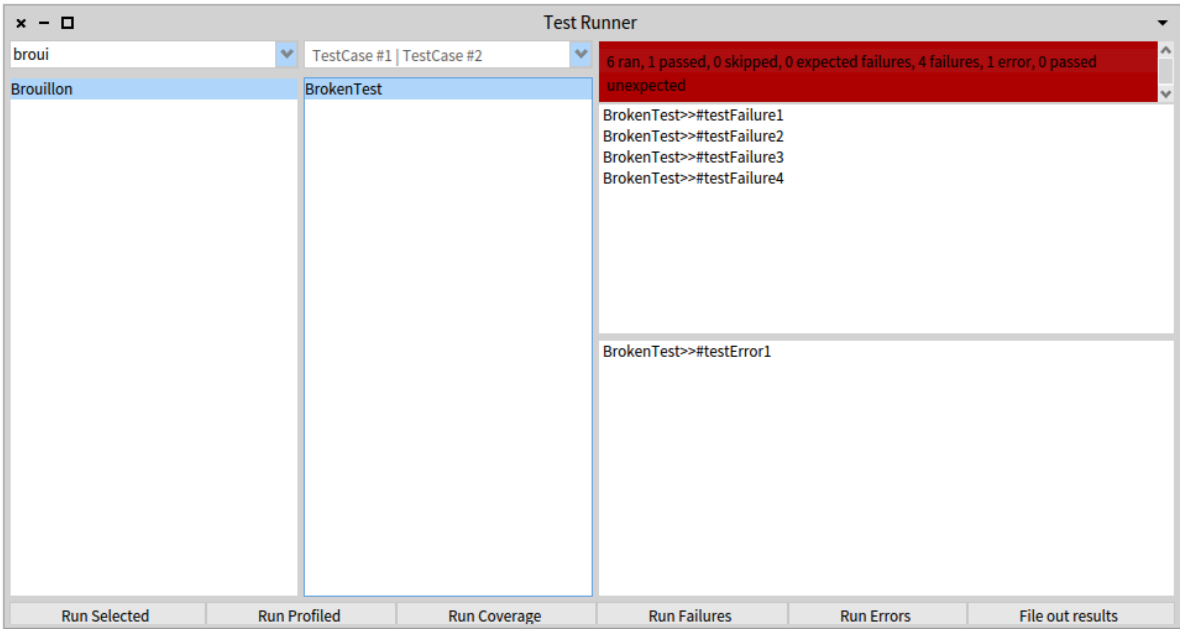
[4] **wikipedia.** wikipedia. [En ligne] 13 06 2022. https://fr.wikipedia.org/wiki/Trait_(programmation)#:~:text=Un%20trait%20est%20une%20sous,classe%20pour%20%C3%A9tendre%20ses%20fonctionnalit%C3%A9s..

[5] **Pharo, Mooc.** [En ligne] 13 06 2022. http://rmod-pharo-mooc.lille.inria.fr/MOOC/PharoMOOC/Week5/C019-W5S06-SUnit.pdf.

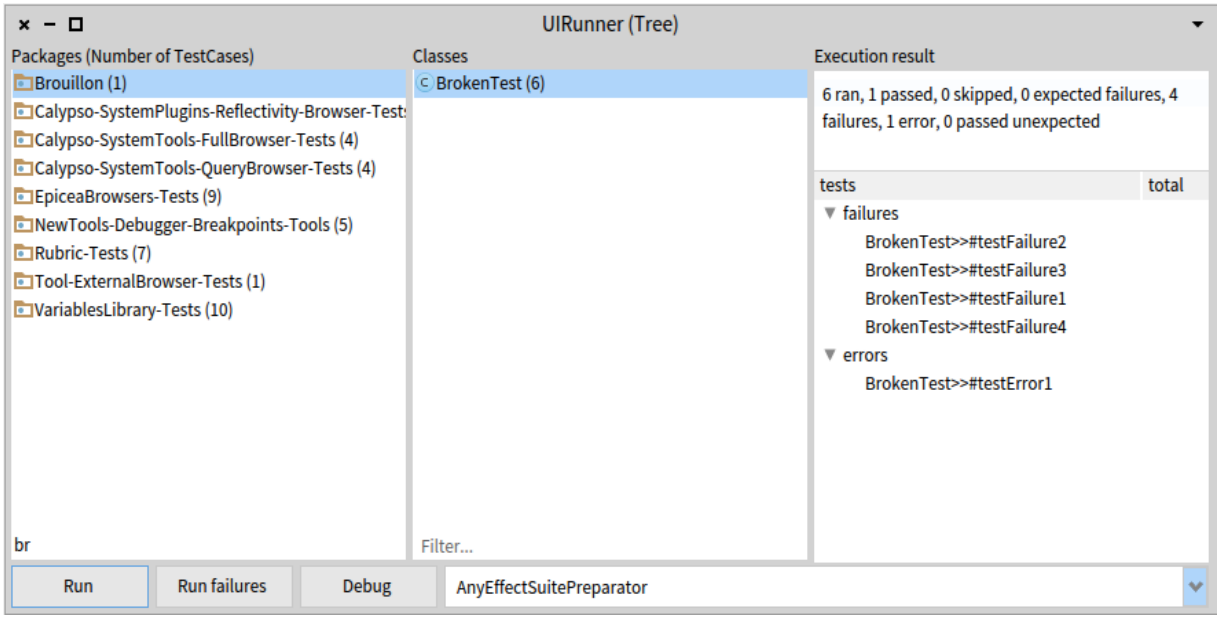
[6] **Blueprint.** [En ligne] https://github.com/NourDjihan/ClassBlueprint.

6. Annexes

- Annexe1 : Exemple d'exécution de tests avec TestRunner



- Annexe2 : Exemple d'exécution de tests avec GuiTestRunner



- Annexe3 : Exemple d'exécution de tests avec NewTestRunner

```
TestRunner2 run: GreenTest "2 ran, 2 passed, 0 skipped, 0 expected failures, 0
failures, 0 errors, 0 passed unexpected".

TestRunner2 runSuite: GreenTest suite "2 ran, 2 passed, 0 skipped, 0 expected
failures, 0 failures, 0 errors, 0 passed unexpected".

TestRunner2 runCase: (GreenTest suite tests at: 1) "1 ran, 1 passed, 0 skipped, 0
expected failures, 0 failures, 0 errors, 0 passed unexpected".

TestRunner2 runPackage: #'SUnit-Tests' asPackage "185 ran, 181 passed, 3 skipped, 0
expected failures, 2 failures, 2 errors, 0 passed unexpected".

"/!\ Too long, run all the tests in the image"
TestRunner2 runAllPackages.
```