Projet Génie-Logiciel
Bilan

BECK BALIHAUT-UTARRE Loula GILLET Maxime
GILLIOT Mathis KRZISCH Pierre-Emmanuel TROMPETTE Thomas
Équipe 27

Janvier 2017

1 Description critique de l'organisation adoptée dans l'équipe

Dès le début du projet, il a été fait comme choix de donner à chaque membre du groupe une responsabilité organisationnelle pour des raisons de cohésion et d'implication dans le projet. Chaque responsable doit être en mesure de répondre aux questions des autres membres de l'équipe pour pouvoir gagner du temps. La distribution s'est faite en fonction des fiches d'auto-évaluation de chacun. La répartition des rôles dans l'équipe est la suivante :

- Chef de projet : pour distribuer les rôles efficacement, optimiser cette distribution en fonction des connaissances de chacun et rôle de médiateur en cas de problème :

  Mathis GILLIOT

- Responsable planning : estime le temps nécessaire pour chaque tâche après discussion avec les personnes concernées, mise à jour régulière du planning et informe le responsable organisation si du retard est :

  Pierre-Emmanuel KRZISCH

- Responsable organisation : veille au respect des deadlines imposées. Doit savoir quelles sont les tâches à prioriser (méthode agile) :

  Maxime GILLET

– Responsable tests : automatise les tests et vérifie la couverture de test des programmes effectués :

  Thomas TROMPETTE

– Responsable communication : crée des supports de communication pour les suivis et présentations :

  Loula BECK

Pour le code en lui même, la distribution des rôles de chacun a évolué au fil du temps et des objectifs à atteindre.

Début du projet

Objectif Prendre connaissance des spécifications et des sources fournies.

Organisation Le groupe a été scindé en deux équipes : une équipe de deux sur le lexer et une équipe de trois sur le parser.

Regard critique Cette organisation a été efficace pour le début du projet. Elle a permis à tout le groupe de s'immerger dans le projet et d'avoir un regard global sur ce dernier.

Avant le premier suivi

Objectif "Hello, world"

Organisation L'équipe a été divisée pour avoir un rendu correct au premier suivi. Deux personnes ont travaillé sur l'étape B, une personne sur la charte, une personne sur le planning et une personne sur l'étape C.

Regard critique L'organisation a été efficace puisque l'objectif a été quasiment atteint ("Hello, world" avec des guillemets) et a été rempli très peu de temps après le suivi.

Langage Deca sans objet

Objectifs Rendre le compilateur `decac` pour le langage Deca sans objet ; Avoir commencé l'extension

Organisation Le groupe a été découpé en trois équipes. Une personne a travaillé sur l'étape C, une sur l'extension et les trois autres sur le parser.

Problèmes rencontrés La distribution des rôles n'était pas optimale en terme de priorité pour l'implémentation du compilateur. Nous avons donc recentré nos forces vers les tâches les plus prioritaires (en particulier l'étape C).

Nouvelle organisation Deux personnes se sont concentrées sur l'étape B pendant que deux autres ont avancé sur l'étape C. La cinquième personne a continué son travail de recherche sur l'extension avec l'intention de démarrer son implémentation.

Regard critique Avant le suivi, la partie A n'était pas complètement implémentée nous avons donc mis pratiquement tous nos efforts sur celle-là, les ressources étaient cependant mal distribuées car nous n'avions pas réalisé la complexité de la partie B et C. Le suivi nous a donc permis d'en prendre conscience et de réorienter nos efforts sur ces dernières. La nouvelle organisation était beaucoup plus efficace et nous a permis d'avoir un compilateur correct au rendu intermédiaire.

Au niveau des différences entre les plannings prévisionnel et effectif, la répartition des équipes sur les parties B et C est restée la même. En revanche, nous avons plus finement réparti les membres du groupe sur les points de l'étape C. Nous nous sommes tenus au nombre de jours prévus pour l'aboutissement de la partie sans objet.

Nous avons pris plus de temps que prévu pour nous documenter sur l'extension. Son implémentation n'a pas été commencé pour le rendu intermédiaire.

Bilan sur le rendu intermédiaire La partie A a été implémentée et la base de tests était correcte. Quelques problèmes soulevé dans le compte-rendu ont été rapidement corrigés.

La partie B a été consciencieusement réalisée et testée.

Sur la partie C, des problèmes importants mis en évidence. En particulier, une spécification et la sémantique des options ont été mal implémentées. Ce problème a été réglé immédiatement le compte-rendu. Cependant un problème plus contraignant a été soulevé dans la classe `AbstractBinaryExpr` et a limité les tests. Il a été corrigé avant l'entame de la partie avec objets mais a nécessité un temps non-négligeable.

Des scripts ont été réalisés pour automatiser les tests et comprendre l'utilisation et l'intérêt de Cobertura.

Langage Deca avec objets

Objectif Rendre le compilateur `decac` pour le langage Deca avec objets.

Organisation L'organisation a été la même que celle à la fin de la partie sans objet.

Regard critique L'implémentation de l'étape B a été efficace car l'équipe était déjà familière avec la grammaire et la réalisation des tests pour cette partie. Pour l'étape C, l'implémentation de la génération de code pour l'objet n'était pas forcément analogue au code de la partie sans objet (accès aux champs par exemple).

Après la partie sans objet, nous avons revu la répartition des membres du groupe sur le planning prévisionnel. Elle a été conservée sur le planning effectif. C'est en effet de cette façon que nous avons travaillé le plus efficacement. Par ailleurs, nous avons découpé la partie C en plusieurs étapes. Afin de nous tenir au nombre de jours prévus pour l'aboutissement de la partie avec objet, nous avons dû empiéter sur nos heures de sommeil et nos heures de repos le week-end. Cet écart a peut-être nuit à notre concentration et donc à notre efficacité. Parmis les leçons que nous avons tirées de ce projet, nous savons désormais qu'il est judicieux de se fixer des deadlines à des dates antérieures aux dates de rendus.

Pour l'extension, nous pensions, dans le planning prévisionnel, que nous disposerions d'au moins 4 à 5 jours. Cependant, du fait des informations non fiables ou contradictoires de certains sites, de l'impossibilité de se servir des PC de l'Ensimag pour travailler sur l'extension et du manque de documentation suffisamment claire pour des novices sur le sujet, nous avons maîtrisé fonctionnement de la compilation ainsi que de l'exécution seulement la veille du rendu. Cela explique que nous ne disposions au final plus que de deux jours pour l'implémentation.

Extension Nous avons choisi cette extension pour avoir une application concrète de notre compilateur. Le choix d'organisation pour l'implémentation de cette extension était initialement de ne spécialiser qu'un seul membre.

Dans la pratique, les recherches sur cette extension ont été réalisées à tour de rôle par chacun des membres du groupe. En effet, celles-ci n'étant pas concluantes, nous avons attribué ce rôle à un autre membre pour rechercher avec un nouvel oeil. Ainsi, énormément de temps a été perdu en terme de code pour le compilateur, ce qui a impacté la qualité du rendu de celui-ci.

Face à l'absence d'informations, nous avons contacté M. Viardot pour lui demander un vecteur valide. Cette décision nous a permis, mais tardivement, d'avoir un environnement de travail pour l'ARM. À ce stade, nous avons dû mettre la priorité sur le rendu du compilateur. Ainsi, malgré l'implémentation de méthodes pour générer le code en ARM, le compilateur ARM ne fonctionnait pas le jour du rendu final.

Le choix a donc été fait de mettre trois personnes au travail sur l'extension deux personnes sur les présentations et la documentation durant les trois jours séparant le rendu de la soutenance.

2 Présentation de l'historique du projet : ordre choisi pour la conception et le développement des étapes B et C, temps passé sur les différentes activités

2.1 Étape B

L'étape B consiste à vérifier la syntaxe contextuelle de Deca, ainsi que d'enrichir l'arbre abstrait et d'y ajouter des informations contextuelles. Durant le projet, deux d'entre-nous se sont spécialisés sur l'étape B et ont effectué l'ensemble des tâches nécessaires à son implémentation.

La première étape a été d'implémenter la classe `EnvironmentExp`. Il a donc été nécessaire de comprendre la définition d'un identificateur et puis celle de l'environnement qui associe à un identificateur sa définition. Cela nous a permis de comprendre le rôle de la classe `EnvironmentExp`. Nous avons donc décidé de représenter un environnement par une table de hachage pour assurer l'unicité de la définition associée à chaque identificateur déclaré. Pour cela, nous avons ajouté un attribut de type `HashMap` dans la classe `EnvironmentExp` et nous avons implémenté une fonction pour ajouter une définition dans un environnement. Cette étape nous a pris quelques heures et a été testée par la suite grâce à `LOG.debug` en affichant des environnements où des variables ont été ajoutées.

Ensuite, nous nous sommes penchés sur l'implémentation des fonctions `verifyXyz` présentes dans chaque classe `AbstractXyz` qui concernent le langage sans objet. Pour cela, il a d'abord fallu comprendre les notations utilisées pour introduire les règles de syntaxe contextuelle, puis comprendre comment vérifier si ces règles sont vérifiées ou non. Cette analyse nous a pris un temps considérable mais nous a permis d'être plus méthodique par la suite. Nous avons donc redéfini les méthodes `verifyXyz` dans chaque sous-classe en levant une erreur contextuelle lorsqu'une règle de grammaire n'est pas respectée et en appelant les autres méthodes `verifyXyz` sur les différents attributs qui nécessitent une vérification contextuelle. L'analyse du problème nous a pris un temps considérable mais nous a permis de comprendre les enjeux de l'étape de vérification contextuelle et ainsi d'être méthodique et d'optimiser notre efficacité dans l'implémentation des fonctions `verifyXyz`. Nous avons d'abord testé sur des programmes simples les premières fonctions que nous avons complétées. Nous nous sommes répartis le travail pour la suite entre la déclaration des variables et la liste des instructions. Après avoir de nouveau testé notre code sur des programmes plus ou moins complexes, nous nous sommes intéressés à la décoration de l'arbre. L'un d'entre-nous s'est occupé de l'enrichissement avec le noeud `ConvFloat` pendant que l'autre s'est occupé de l'enrichissement avec les types des expressions et les définitions des identificateurs. En parallèle, une troisième personne a pris en charge l'implémentation des fonctions de décom-

pilation. Nous avons ensuite construit une base solide de tests en créant un programme valide et un programme invalide pour chaque règle de la syntaxe contextuelle à vérifier, en affichant l'arbre enrichi ou le message d'erreur généré à l'aide du script `test_context.sh` et en vérifiant que le résultat était conforme. La construction de cette base de tests nous a pris deux jours. Ces tests nous ont permis de faire apparaître un nombre limité de bogues grâce notre implémentation méthodique des fonctions de vérifications. Enfin, nous avons listé les erreurs de l'étape B pour le langage sans-objet. Ainsi, nous avons été en mesure de livrer une partie B robuste pour le langage sans objet.

Pour le langage avec objets, nous avons dû effectuer trois passes de l'arbre abstrait. Nous avons donc créé une fonction de vérification pour chaque passe. Le principe de codage a été le même que pour le langage sans objet. L'un d'entre-nous s'est donc occupé des deux premières passes pendant que l'autre s'est occupé de la troisième. Cependant, la complexité de la traduction des règles de grammaire nous a incité à beaucoup plus travailler ensemble particulièrement pour déboguer. Le codage nous a pris trois jours. Comme pour le langage sans objet, nous avons construit une base de tests conséquente pendant les deux jours et demi restants.

2.2 Étape C

La stratégie adoptée a été de traîter les tâches les plus urgentes en priorité.

Langage Deca sans objet

1. Implémentation des opérations arithmétiques `ADD`, `SUB`, `MUL`, `DIV` et `MOD`

   D'abord en nous concentrant sur l'instruction `ADD`, nous avons ensuite propagé et factorisé le code commun aux opérations. Dans le même temps, nous avons créé une classe `Registres` qui permet la gestion des registres (occupation et libération). On utilise cette classe pour les évaluations d'expressions. Initialement statique, on a finalement choisi d'en déclarer une instance dans `DecacCompiler` afin de répondre à la spécification de parallélisable (option -P). On a ensuite propagé en paramètre l'instance de `Registres`. Il aurait peut-être été plus adapté de créer un getter sur l'attribut `regs` dans `DecacCompiler`.

   Des tests ont été conduits en parallèle par une tierce personne.

2. Dès la décoration de l'arbre effectuée, on a géré la déclaration des variables

   Essentielle pour effectuer des tests plus poussés que de simples opérations entre littéraux, la déclaration des variables a été implémentée rapidement. En parallèle, nous

avons créé une classe `Pile` contenant les méthodes et attributs nécessaires pour gérer la pile. Même remarque que pour `Registres` concernant la pile.

3. Ensuite, nous avons implémenté l'initialisation et l'affectation de ces variables.

   Cette implémentation n'a pas posé de problème pour la partie sans objet.

4. Nous avons réalisé le codage des opérateurs booléens `true`, `false`, `Not`, `AND`, `OR`

   Cette partie a posé un problème majeur. L'évaluation de la condition ayant pour but un branchement. Il n'est pas toujours nécessaire de stocker le résultat dans un registre mais seulement pour une affectation ou une initialisation.

5. Enfin, nous avons implémenté les structures de contrôles

   Il a été difficile de créer des étiquettes uniques pour chaque `IfThenElse` ou `While`. Nous avons donc créé une classe dédiée à ce comptage : `CompteurEtiquette`.

Langage Deca avac objets

1. Construction de la table des méthodes
2. Codage des champs
3. Code d'une sélection de champ
4. Initialisation des champs
5. Codage des méthodes
6. Codage des appels de méthodes
7. Codage de new

Finalement, pour cette étape, l'analyse a nécessité une journée de travail, la conception deux et le codage et la validation cinq.

3 Résumé des enseignements que vous avez pu tirer de ce projet

Le projet Génie Logiciel est le projet le plus important que nous avons effectué depuis le début de nos études à l'Ensimag. Il nous a permis de nous mettre dans des conditions similaires à celles dans lesquelles peut travailler un ingénieur Ensimag. En effet, nous avons été amené, par exemple, à livrer un produit à un client, à respecter des deadlines et à se conformer à un cahier des charges. Il nous a aussi permis de découvrir les méthodes agiles et leurs avantages en terme d'organisation du travail. Durant ce projet, nous avons dû nous organiser pour gérer au mieux notre temps et nos ressources humaines. Au vu de cette expérience nous avons pu tirer de nombreux enseignements, en particulier sur l'importance d'une vision extérieure et sur les méthodes de travail.

Durant ce projet, nous avons remarqué que le plus important est d'être bien organisé, que ce soit au niveau de la gestion du temps que de la distribution des ressources humaines. Ainsi, nous nous sommes rendu compte que la vision d'un œil extérieur au code du projet permet de prendre conscience de la priorité des différentes tâches à accomplir et ainsi de redistribuer plus efficacement les ressources disponibles. Cela a mis en évidence la nécessité de « sortir » du code par moment et l'importance des suivis qui apportent un œil neuf à la fois sur les méthodes de travail et sur les directions de la conception.

De plus, nous nous sommes aperçus que travailler à plusieurs sur la même tâche pouvait se révéler beaucoup plus efficace que d'avancer chacun sur une partie. En effet, le codage et le débogage à plusieurs (la plupart du temps à deux) sur les parties complexes n'a pas été une perte de temps mais au contraire s'est avéré précieux. En outre, cela facilite la prise en compte de l'avis des autres membres du goupe même lorsqu'ils ne sont pas spécialisés sur la partie en question et évite qu'un membre reste bloqué seul sur un problème.

L'autre principal enseignement de ce projet concerne les méthodes de travail. En effet, nous avons par exemple pu nous rendre compte que les engagements pris au début du projet, comme le respect d'un planning ou la construction de tests avant l'implémentation, peuvent être difficiles à respecter, particulièrement avec la pression de la date du rendu. Cependant, lorsque celles-ci sont respectées, elles peuvent se révéler très efficaces notamment dans l'organisation du travail. C'est le cas de la prise de responsabilités par chaque membre de notre équipe qui nous a permis de nous impliquer totalement dans ce projet en ayant un regard plus attentif sur un aspect particulier.

Au vu du temps investi et de la quantité de travail dans ce projet, nous avons aussi appris que vouloir en faire toujours plus n'était pas toujours judicieux. En effet, cela peut entraîner

un manque d'efficacité et augmenter le risque que l'un des membres du groupe s'enferme dans un problème. Enfin, nous avons dû gérer la pression de la date du rendu au niveau de la qualité du code et de l'organisation. Cette gestion est grandement facilitée par un rituel fixé à l'avance sur les différentes actions à réaliser avant le rendu. Cependant, avec le stress, il est très difficile de s'y conformer totalement.

Le projet Génie Logiciel nous a permis de nous immerger dans de réelles conditions de travail. Dans l'ensemble nous avons pu tirer des enseignments positifs de ce projet, en particulier sur les méthodes de travail. Cependant, nous avons aussi été confronté à des difficultés comme la gestion de la pression de la date du rendu et le temps pris par la formation des ressources (notamment concernant l'extension).