



PLAN PROJET INTERPRÉTEUR DU LANGAGE LIR

PROJET PROPOSÉ PAR FRÉDÉRIQUE BARRIOS

Nicolas CAMINADE, Sylvan COURTIOL,
Pierre DEBAS, Heïa DEXTER,
Lucàs VABRE

Sommaire

I Plan projet	4
Introduction	5
1 Présentation du projet	6
1.1 Définition générale du besoin : l'Interpréteur LIR	6
1.2 Cahier des charges	6
1.3 Définitions et acronymes	6
1.4 Charte de projet	7
1.4.1 Objectifs du projet	7
1.4.2 Périmètre du projet	7
1.4.3 Demandes hors périmètre	7
1.4.4 Principaux livrables identifiés	8
1.4.5 Les acteurs du projet	8
1.4.6 Autres moyens et ressources	8
1.4.7 Conditions d'acceptation	8
1.4.8 Principaux risques identifiés et politique de gestion des risques	8
1.5 Étude générale du besoin	9
1.5.1 Les acteurs	9
1.5.2 Résumés de cas d'utilisation	9
1.5.3 Récits d'utilisation (user stories)	11
2 Organisation du projet	12
2.1 Présentation du cycle de vie itératif	12
2.2 Répartition des rôles	12
2.3 Plan communication	13
2.3.1 Localisation géographique des intervenants	13
2.3.2 Moyens de communication utilisés	13
2.3.3 Réunions projets MOE	13
2.3.4 Comités de Pilotage	13
2.4 Assurance qualité	14
2.4.1 Normes et standards de travail à observer (forma- lisme de modélisation, méthodes de contrôle, mé- thodes de développement, cycle de vie, conven- tions de code...)	14
2.5 Ressources matérielles et logicielles	14

3	Pilotage du projet	15
3.1	Cycle de vie itératif	15
3.2	Estimation initiale	15
3.3	Planification prévisionnelle initiale	16
3.4	Durée et ordonnancement des principales tâches et itérations	16
3.5	Identification des premiers jalons	17
3.6	Calendrier prévisionnel	17
3.7	Organisation des réunions projets et comités de pilotage .	18
3.8	Suivi du projet pour la première itération	18
3.8.1	Planification et ordonnancement des tâches	18
3.8.2	Suivi d'avancement et mesure des écarts par rapport au prévisionnel	20
3.8.3	Identification des principaux écarts et problèmes constatés, solutions possibles	20
3.8.4	Propositions de modification de la planification prévisionnelle pour tenir compte des corrections à apporter	21
3.8.5	Comptes-rendus des réunions projets de la période .	21
3.8.6	Compte-rendu du comité de pilotage de la période	22
3.8.7	Planification prévisionnelle révisée pour les périodes suivantes (en fonction des décisions prises)	22
3.9	Suivi du projet pour la seconde itération	22
3.9.1	Suivi d'avancement et mesure des écarts par rapport au prévisionnel revu lors de la période précédente	22
3.9.2	Identification des principaux écarts et problèmes constatés, solutions possibles	23
3.9.3	Propositions de modification de la planification prévisionnelle pour tenir compte des corrections à apporter	23
3.9.4	Comptes-rendus des réunions projets de la période .	23
3.9.5	Compte-rendu du comité de pilotage de la période	23
3.9.6	Planification prévisionnelle révisée pour les périodes suivantes (en fonction des décisions prises)	24
3.10	Suivi du projet pour la troisième itération	25
3.10.1	Suivi d'avancement et mesure des écarts par rapport au prévisionnel revu lors de la période précédente	25
3.10.2	Identification des principaux écarts et problèmes constatés, solutions possibles	26
3.10.3	Compte-rendu du comité de pilotage de la période	26
3.10.4	Idées d'améliorations	26
4	Bilan	28
II	Annexes	29
A	Sujet Interpréteur LIR	30
B	Gestion de la configuration de l'Interpréteur LIR	39

Première partie

Plan projet

Introduction

Dans le cadre des projets tuteuré du semestre 2 de première année de DUT informatique de l'année 2020-2021, le sujet de l'Interpréteur LIR a été proposé par F. Barrios, un des enseignants de l'IUT de Rodez.

Ce document a pour but de rassembler les informations fondamentales relatives à la gestion du projet. Ce plan projet est un document de référence du projet qui sera complété tout au long de son avancement.

Chapitre 1

Présentation du projet

1.1 Définition générale du besoin : l'Interpréteur LIR

L'Interpréteur LIR est un interpréteur d'un langage de programmation simple, il sera nommé LIR pour Langage IUT de Rodez. Un interpréteur est un automate enchaînant les tâches suivantes : analyse lexico-syntaxique d'une ligne de commande puis interprétation.

Une ligne entrée par un utilisateur sera donc : soit une commande à exécuter immédiatement, soit une ligne de programme à mémoriser pour une exécution ultérieure. Une ligne de programme se distinguera d'une ligne de commande par le fait qu'elle sera toujours précédée d'un "numéro d'ordre" appelé aussi "étiquette".

1.2 Cahier des charges

Le document en annexe fourni par la maîtrise d'ouvrage (MOA) définit l'interpréteur attendu avec les éléments du Langage IUT de Rodez, la syntaxe des instructions de programmation et des commandes générales attendues dans le logiciel final. Le document précise également le comportement attendu de l'interpréteur lors de son utilisation suivi d'un exemple d'une session sous cet interpréteur LIR.

1.3 Définitions et acronymes

Analyse syntaxique : La vérification de la conformité aux contraintes syntaxiques définies par une grammaire.

Analyse lexicale : L'identification des éléments du vocabulaire d'un langage dans une description textuelle (scanning) et la recherche des unités lexicales (lexèmes).

Grammaire : Contraintes syntaxiques définissant les constructions correctes (autorisées) d'un langage.

Interpréteur : Programme capable d'analyser les instructions d'un langage (évolué) et de les exécuter directement.

Langage : Outil de description et d'expression.

Langage IUT de Rodez (LIR)

Sémantique : Étude du sens des unités linguistiques et de leurs combinaisons.

Aspect de la logique qui traite de l'interprétation et de la signification des systèmes formels, par opposition à la syntaxe, entendue comme l'étude des relations formelles entre formules de tels systèmes (d'après le dictionnaire Larousse).

Syntaxe : Partie de la grammaire qui décrit les règles par lesquelles les unités linguistiques se combinent en phrases. En logique, étude des relations formelles entre expressions d'un langage (d'après le dictionnaire Larousse).

Aussi, la syntaxe est spécifiée par des grammaires et des notations formelles.

Vocabulaire : Symboles de base utilisés dans un langage.

1.4 Charte de projet

1.4.1 Objectifs du projet

Réaliser un interpréteur capable d'exécuter un script ou une série d'instructions dans le langage LIR avec les outils et connaissances et mis à disposition par l'IUT de Rodez.

1.4.2 Périmètre du projet

Ce projet doit être mené jusqu'à obtention d'un interpréteur capable d'exécuter toutes les commandes précisées dans le cahier des charges fourni.

1.4.3 Demandes hors périmètre

Il n'y a pas de demandes hors périmètre.

1.4.4 Principaux livrables identifiés

Livrables : plan projet, dossier de projet, CD (de préférence un dossier compressé plutôt qu'un CD) contenant les codes exécutables les fichiers de données, les codes sources et la version numérique du dossier et le manuel utilisateur.

Définition du cadre

Coût : À définir par le chef de projet (P. Debas).

Délais : Deux dates butoirs identifiées.

— Remise du projet le vendredi 28 mai 2021.

— Soutenance du projet la semaine du 7 juin 2021.

Qualité : Projet codé en Java dans les respects des conventions et bonnes pratiques.

1.4.5 Les acteurs du projet

L'équipe MOE : N. CAMINADE, S. COURTIOL,
P. DEBAS, H. DEXTER,
L. VABRE

La MOA : F. Barrios

Le contrôle qualité : F. Barrios et J. Accot

1.4.6 Autres moyens et ressources

Pas de moyens ou ressources supplémentaires.

1.4.7 Conditions d'acceptation

Pas d'exigence ou de contraintes supplémentaires.

1.4.8 Principaux risques identifiés et politique de gestion des risques

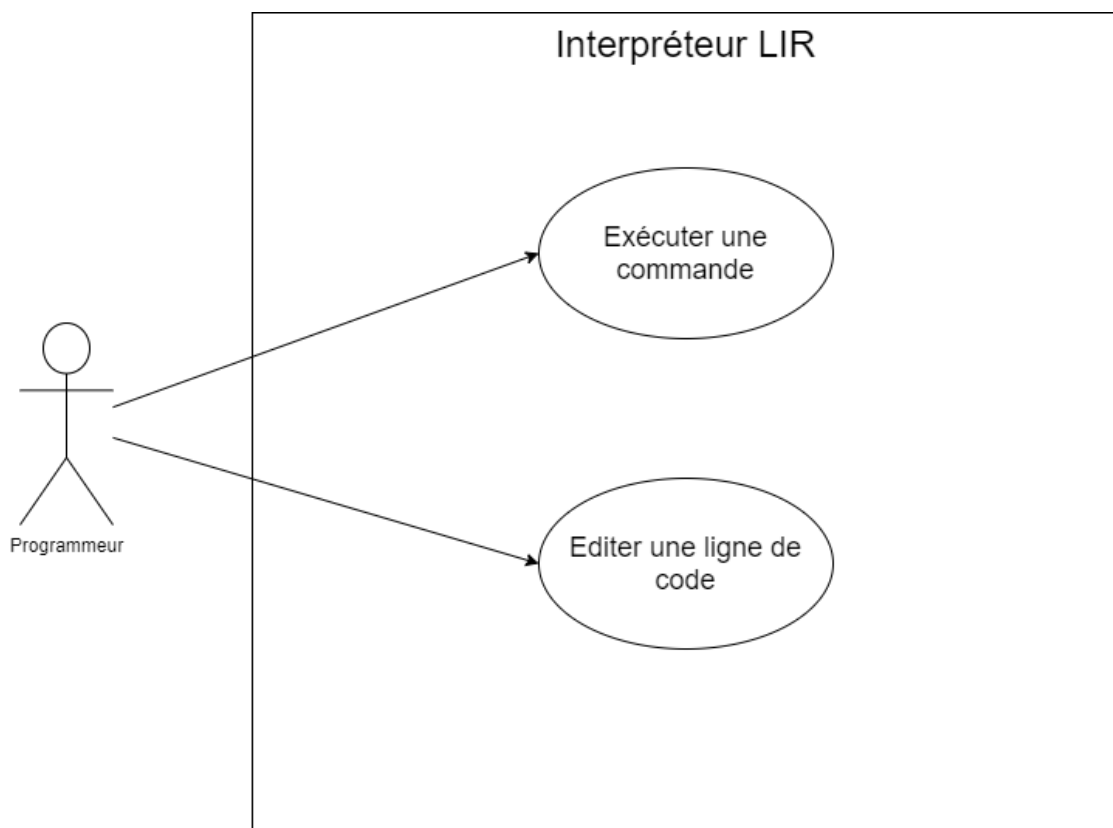
Si possible tous les membres du groupe auront les mêmes droits sur les fichiers communs. En conséquence chaque membre du groupe ne doit pas donner des droits sur ces fichiers à une personne extérieure au projet (autre que MOA). Cf. Gestion de la configuration (produit par S. Courtiol).

Des sauvegardes du dépôt GitHub (contenant toutes les données du

projets) seront effectuées régulièrement (fréquence à définir) par le gestionnaire de configuration. Toutes données qui ne sont pas dans le dépôt sont à la responsabilité de chacun. Cf. Gestion de la configuration (produit par S. Courtiol).

1.5 Étude générale du besoin

Diagramme de cas d'utilisation général de l'Interpréteur LIR comprenant un acteur (le programmeur) et deux cas d'utilisation identifiés comme suit :



1.5.1 Les acteurs

Programmeur : Personne utilisant l'interpréteur.

1.5.2 Résumés de cas d'utilisation

— Exécuter une commande

Acteurs Programmeur : il entre une commande à faire exécuter immédiatement par l'interpréteur.

Objectifs Exécuter la commande entrée dans l'interpréteur.

Pré-Conditions L'interpréteur LIR est lancé et le curseur est derrière l'invite.

Post-Conditions La commande est exécutée et un résultat ou un feedback est affiché.

Scénario nominal (grandes étapes)

1. Le programmeur écrit derrière l'invite une ligne de commande.
2. Le programmeur valide cette commande.
3. L'interpréteur effectue une analyse lexico-syntaxique.
4. L'interpréteur interprète la ligne de commande.

Scénarios d'échec

Point 3 du scénario nominal : la syntaxe de la ligne écrite est incorrecte.

- Un message d'erreur explicite informe le programmeur.
- Retour au point 4 du scénario nominal.

Point 4 du scénario nominal : la commande conduit à une erreur d'exécution.

- Un message d'erreur explicite informe le programmeur.
- Retour au point 4 du scénario nominal.

— Éditer une ligne de code

Acteurs Programmeur : Il écrit ou modifie une ligne de code dans un programme à faire exécuter par l'interpréteur.

Objectifs Écrire une ligne de code dans nouveau programme ou un existant afin d'exécuter ou de sauvegarder ce programme.

Pré-conditions Le curseur est derrière l'invite suivi d'une étiquette correspondant au numéro de la ligne de code à éditer.

Post-conditions Le code source édité est prêt à être exécuté, abandonné ou sauvegardé, selon l'intention du programmeur.

Scénario nominal (grandes étapes)

1. Le programmeur écrit une instruction ou commande par ligne de code, en la faisant précéder de son étiquette.
2. Le programmeur consulte le code déjà écrit à tout moment avec la commande `liste`. Selon la syntaxe choisie, l'interpréteur affiche la plage demandée ou la totalité des lignes de code du programme dans l'ordre croissant des étiquettes.
3. Le programmeur consulte la liste des identificateurs déclarés et leurs valeurs en entrant la commande `defs`.
4. Au besoin, le programmeur efface une ou plusieurs lignes avec la commande `efface`.
5. Au besoin, le programmeur efface les lignes de code et identificateurs mémorisés et commence un nouveau code avec la commande `debut`.

Scénarios d'échec

Point 2 du scénario nominal : Aucune ligne de code n'est écrite ou la plage de code à afficher n'est pas correcte.

- L'interpréteur en avise le programmeur au moyen d'un message d'erreur.
- Retour au point 1.

Point 3 du scénario nominal : Aucun identificateur n'a encore été déclaré.

- L'interpréteur affiche un message informant le programmeur.
- Retour au point 1.

Point 4 du scénario nominal : La plage de ligne à effacer est incorrecte.

- Un message d'erreur en informe le programmeur
- Retour au point 1.

1.5.3 Récits d'utilisation (user stories)

Les récits d'utilisation

Des récits d'utilisation ont été rédigés pour chaque commande et instruction.

Chapitre 2

Organisation du projet

2.1 Présentation du cycle de vie itératif

Pour développer l'Interpréteur LIR, le modèle de cycle de vie itératif a été choisi. Ce modèle de développement de logiciel consiste en une succession de cycles de spécification, de conception, de réalisation et de tests, le but est d'enrichir et de « remodeler » des prototypes du logiciel successifs. Par conséquent, une version du logiciel sera un « dernier prototype ».

La gestion du risque va entraîner la mise en place d'un noyau architectural avec des fonctions indispensables du logiciel dès les deux premières itérations. Les itérations suivantes apporteront des corrections et de nouvelles fonctions au logiciel.

Les versions successives des prototypes permettent de matérialiser l'avancement et d'éviter « l'effet tunnel » sur le projet. Ces prototypes (versions 0.x) entretiennent la motivation des différents acteurs du projet : l'équipe MOE, la MOA.

Le principe fondamental à chaque début d'itération est de ne spécifier en détail que les fonctionnalités nécessaires pour cette itération. Ainsi la prise en compte d'évolutions du besoin reste possible jusqu'à la dernière itération. De même le « refactoring » de la conception (largement facilité par les outils) a lieu à chaque étape pour intégrer des évolutions et des ajouts. Le but étant bien sûr de fabriquer le logiciel adapté au besoin en laissant la possibilité de « mûrir » au cours du temps.

Ce type de cycle implique une taille homogène de l'équipe et une polyvalence des équipiers.

2.2 Répartition des rôles

Rôles des membres de l'équipe impliqués dans le projet jusqu'au mois de mai 2021 :

Chef de projet MOE	Pierre Debas
Secrétaire de projet	Heïa Dexter
Gestionnaire de configuration	Sylvan Courtiol
Développeur	Nicolas Caminade
Développeur	Lucàs Vabre

2.3 Plan communication

2.3.1 Localisation géographique des intervenants

L'équipe MOE, la MOA et les contrôleurs qualités sont basés sur Rodez (12).

La MOA, les contrôleurs qualités, H. Dexter sont basés sur Rodez (12), S. Courtiol sur Luc-La-Primaube à côté de Rodez (12), P. Debas est basé à la fois sur Rodez et à Albi (81), L. Vabre sur Gages (12) et N. Caminade sur Rodez et Moncaut (47).

2.3.2 Moyens de communication utilisés

Les communications formelles sont effectuées via les mails de l'IUT (généralement par le chef de projet) avec les autres membres du projet en CC.

Serveur Discord spécifique au projet pour communication écrite ou vocale de la MOE.

Cf. le document Configuration interpréteur du langage LIR produit par le gestionnaire de configuration (S. Courtiol).

2.3.3 Réunions projets MOE

Les réunions projet MOE seront hebdomadaires voire bi-hebdomadaires et dans le contexte de la crise sanitaire elles se dérouleront en distanciel via Discord (vocal, visio-conférence). Seront prévue des réunions courtes de 20 minutes et des réunions longues de 1h30.

Ces réunions auront pour objectif de faire le point sur l'avancement du projet, le respect des objectifs fixé sur la période et de fixer les prochains objectifs à remplir d'ici la prochaine réunions. Aussi ces réunions seront l'occasion de faire part de difficultés éventuelles rencontrées par les membres de l'équipe au cours de la semaine et de communiquer les informations sur les prochaines rencontres avec la MOA.

Les comptes-rendus seront rédigés par la secrétaire de projet (H. Dexter) et diffusés sur le serveur Discord de l'équipe sous format texte.

2.3.4 Comités de Pilotage

Les comités de pilotage rassembleront la MOA et toute l'équipe de MOE. Les COPIL seront dirigé par le chef de projet éventuellement assisté

par le secrétaire.

La fréquence des COPIL est au mieux hebdomadaire et d'une durée d'une demi-heure à trois quarts d'heure selon l'avancement du projet. Les comptes-rendus des COPIL seront rédigés par l'actuelle secrétaire de projet (H. Dexter) et diffusés le lendemain à la MOE du projet.

2.4 Assurance qualité

2.4.1 Normes et standards de travail à observer (formalisme de modélisation, méthodes de contrôle, méthodes de développement, cycle de vie, conventions de code...)

Pour mener à bien ce projet l'équipe MOE travaille en utilisant le langage UML comme formalisme de modélisation, la méthode de développement dirigé par les tests i.e. la méthode TDD (Test Driven Development) en respectant les Java Code Convention pour un modèle de cycle de vie itératif.

2.5 Ressources matérielles et logicielles

La partie qui suit est un résumé du document de gestion de la configuration joint au dossier technique. Merci de vous y référer pour plus de détails.

La conception en langage UML sera effectuée sous Modelio. La rédaction des différents documents du dossier sera faite en utilisant \LaTeX . Nous utiliserons Eclipse configuré avec un *workspace* similaire à celui utilisé lors de nos cours de programmation. Les dépôts en ligne et le contrôle de l'historique des versions seront assurées par Git, plus précisément via GitHub. L'avancement sera contrôlé sur un tableau Trello. Enfin, la communication sera assurée via un serveur Discord dédié ou Google Meet pour les visio-conférences.

Chapitre 3

Pilotage du projet

3.1 Cycle de vie itératif

Pour développer l'Interpréteur LIR, le modèle de cycle de vie itératif a été choisi. Ce modèle de développement de logiciel, rappelons-le, consiste en une succession de cycles de spécification, de conception, de réalisation et de tests, le but est d'enrichir et de « remodeler » des prototypes du logiciel successifs. Par conséquent, une version du logiciel sera un « dernier prototype ».

Si le choix de modèle de cycle de vie s'est porté sur le modèle itératif, c'est parce qu'il s'agit d'un modèle "réaliste" et possible à mettre en place dans le cadre des projets tuteurés :

- Une limitation de "l'effet tunnel" pour une meilleure dynamique et motivation des équipes (MOA et MOE).
- Une meilleure acceptation des changements grâce aux prototypes.
- Une meilleure gestion des risques.
- Est adapté pour une équipe de cinq personnes polyvalentes.
- Le principe d'itérations où seules les fonctionnalités nécessaires sont spécifiées en détail en début d'itération ce qui permet une évolution du besoin.

3.2 Estimation initiale

En se focalisant sur un développement de type Organic d'une taille attendue de 2000 lignes de code (2KLOC), le projet nécessite de base un effort de 4,97 mois.homme répartis sur une durée de 4.60 mois. Cette estimation se base sur une équipe d'un seul développeur.

Étant donné que notre équipe se constitue de 5 personnes, il nous suffit d'adapter cette estimation en divisant l'effort par le nombre de membres. Nous obtenons ainsi un effort de 0,99 mois de 20 jours. Notre contexte de travail étant toutefois différent de celui d'une entreprise (pas d'horaires fixes, travail les jours fériés,...), il nous a paru plus judicieux de convertir notre estimation vers un format plus réaliste.

En considérant une durée de deux heures par journée de travail par personne, nous arrivons à un total de 10 heures quotidiennes, soit un total de 200 heures hebdomadaires. Nous conservons la durée de 20 jours par mois pour pallier les indisponibilités de chacun. La durée totale du projet est donc estimée à 0,99 mois ou 200 heures de travail total, soit 40 heures.homme.

3.3 Planification prévisionnelle initiale

Le développement de l'interpréteur LIR suivant un cycle de vie itératif, nous agréé avec la MOA de trois itérations, avec à l'issue de chacune la livraison d'un prototype fonctionnel ou, dans le cas de la dernière itération, du programme complet.

Pour la première itération, nous avons convenu de livrer un prototype incluant les mécanismes de base du fonctionnement des commandes et instructions. Cette première mouture de l'interpréteur Doit être en mesure d'analyser son entrée standard, d'affecter des variables de type chaîne à son contexte d'exécution et de reconnaître les expressions sur les chaînes de caractères. Le contexte pourra être réinitialisé. Enfin, ce prototype devra afficher les variables déclarées, afficher le résultat d'une expression de type chaîne, et être en mesure mettre fin à la session.

Lors de la seconde itération, notre prototype devra, en plus des fonctionnalités sus-mentionnées, incorporer l'affectation de variables de type entier et l'arithmétique entière sur les expressions. L'interpréteur, à ce stade du développement, devra être en mesure de reconnaître des étiquettes de ligne de code et garder en mémoire centrale un programme pour pouvoir ensuite l'afficher ou l'exécuter. Il doit également être en mesure de supprimer des lignes de code dans le programme global. Il sera également possible à l'utilisateur de saisir puis d'affecter la valeur d'une variable via l'entrée standard ou d'effectuer des sauts non-conditionnels dans un programme.

Enfin, le programme final de la troisième itération pourra lire et écrire des programmes vers et depuis des fichiers. Il sera possible au programmeur d'effectuer des sauts conditionnels. Seront livrés avec ce prototype final ce plan de projet complété, ainsi que le manuel d'utilisateur de l'interpréteur LIR. Le prototype final devra pouvoir être lancé à partir d'un fichier exécutable.

3.4 Durée et ordonnancement des principales tâches et itérations

Afin d'avoir un interpréteur LIR fonctionnel, nous avons identifié un jeu de tâches critiques devant être remplies à chaque itération. Ainsi, pour la présentation du premier prototype, devront être implémentés et testés

les littéraux, les identificateurs, les variables, le contexte d'exécution, les commandes et instructions, et l'analyseur lexicale.

Lors de la deuxième itération, devront être traités en priorité la gestion des étiquettes et l'implémentation de programmes à stocker en mémoire. Enfin, l'analyseur devra prendre en compte cette nouvelle fonctionnalité et référencer le programme global de la session.

Pour finir, la troisième itération consistera à rajouter la possibilité au programmeur d'effectuer des sauts conditionnels. Pour ce faire, les conditions devront être traitées sous la forme d'une expression booléenne simple. Cela implique donc la création d'un littéral de type booléen, non accessible directement au programmeur dans cette version de l'interpréteur (le programmeur ne pourra donc pas créer de variable de type Booléen). La troisième itération se terminera par la revue du code et des jeux de tests, l'édition du manuel utilisateur, ainsi que la complétion du dossier technique.

3.5 Identification des premiers jalons

Chacun des prototypes de l'interpréteur LIR devant être livré à l'issue de chaque itération, nous pouvons donc en déduire les premiers jalons ci-dessous. La date de soutenance du projet, quand à elle, nous sera précisée à une date ultérieure à la rédaction de ce plan. Nous ne pouvons donc en donner qu'une estimation.

- Premier entretien MOA / MOE et lancement du projet : jeudi 8 avril
- Livraison du premier prototype : lundi 10 mai
- Livraison du deuxième prototype : mercredi 19 mai
- Livraison du prototype final : Vendredi 28 mai
- Soutenance du projet : entre le 9 et le 11 juin

3.6 Calendrier prévisionnel

À partir des dates citées plus haut, nous pouvons donc en déduire le calendrier prévisionnel suivant. Les comités de pilotage ont été convenus avec la maîtrise d'ouvrage afin de conserver un suivi le plus régulier possible sur l'avancement du développement.

- Première itération : du 3 au 10 mai
- Deuxième itération : du 11 au 19 mai
- Comité de pilotage : mardi 18 mai
- Troisième itération : du 20 au 28 mai
- Comité de pilotage : mercredi 26 mai
- Remise du dossier technique : mardi 8 juin

3.7 Organisation des réunions projets et comités de pilotage

Comme mentionné, les comités de pilotage se tiennent entre deux livraisons afin de faire le point sur l'avancement de l'itération. Compte tenu des restrictions sanitaires imposées au cours de la crise due à la Covid-19, nous devons adapter ces réunions aux modalités de présence imposées par l'IUT de Rodez. Selon la semaine, certains membres de l'équipe devront y assister en visio-conférence.

Les réunions de projet, quand à elles, se tiennent à chaque fois qu'un ensemble de tâches est complété pour mettre en commun le travail effectué et faire le point sur les difficultés techniques rencontrées et la répartition des ressources sur les tâches restantes (nous privilégions en effet le travail en binôme et faisons en sorte que chaque membre de l'équipe ait l'occasion de travailler au moins une fois avec tous les autres).

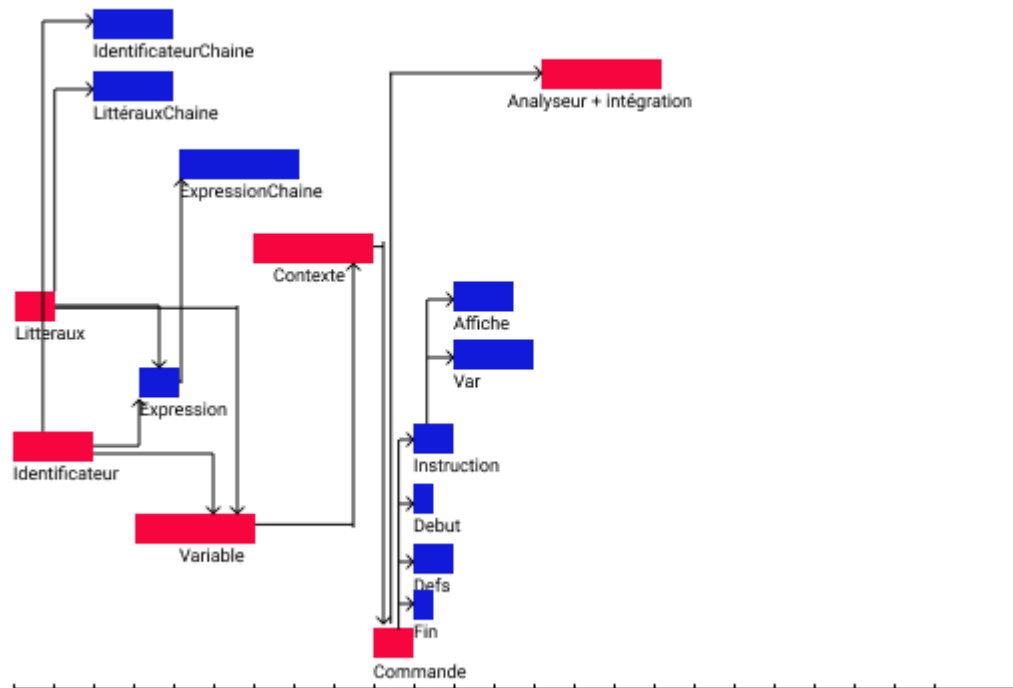
3.8 Suivi du projet pour la première itération

3.8.1 Planification et ordonnancement des tâches

Au tâches déjà spécifiées dans la section nous souhaiterions ajouter quelques fonctionnalités de base de l'interpréteur LIR. La planification de cette itération comprend donc l'implémentation et le test des commandes `defs`, `debut` et `fin`, ainsi que des instructions `affiche` et `var`.

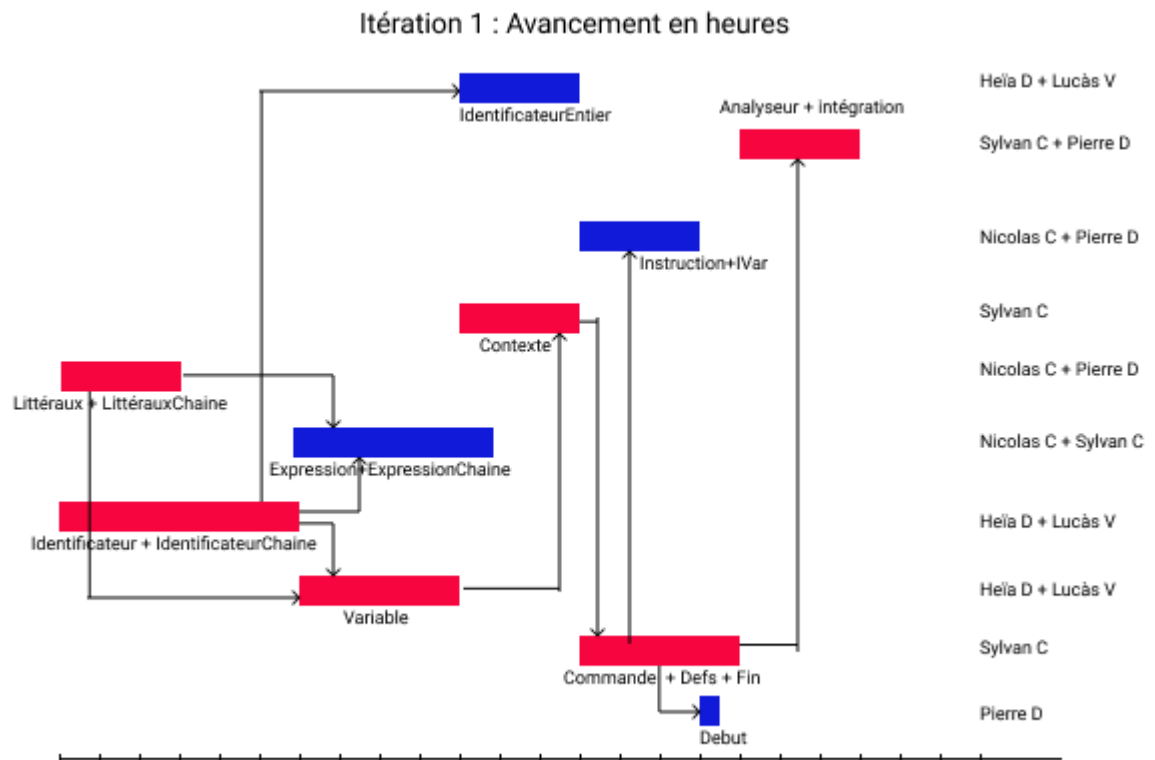
Ce premier prototype fonctionnera uniquement avec des données de type chaîne de caractère. Cela implique donc l'ajout d'identificateurs, de constantes littérales et d'expressions correspondants.

Itération 1 : Planification en heures



Le diagramme de planification de l'itération 1 suggère un total de 46 heures de travail, soit l'équivalent de 23 jours.homme. En raison du nombre limité de ressources de travail, toutes les tâches, notamment les instructions et commandes, ne pourront être réalisées en concomitance. Certaines devront donc se voir repousser le temps qu'un binôme se libère.

3.8.2 Suivi d'avancement et mesure des écarts par rapport au prévisionnel



À l'issue de cette première itération, nous constatons un volume de travail total de 61,5 heures, soit 15,5 heures ou 7,75 jours.homme de plus que ce qui était estimé. Cet écart s'explique d'une part dans une estimation trop optimiste de la durée des tests unitaires d'une part et un manque d'habitude à travailler en binôme d'autre part. Cette itération portant sur des aspect structurels importants de l'interpréteur, il nous sera plus aisé à l'avenir de rajouter les commandes et instructions.

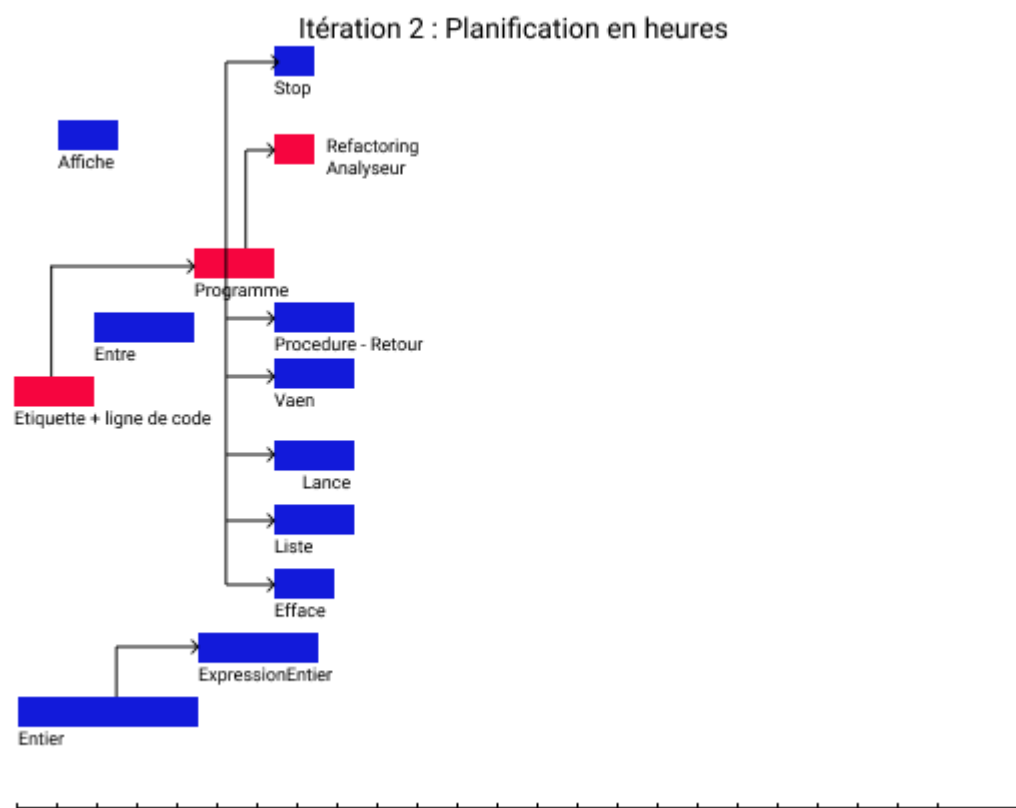
3.8.3 Identification des principaux écarts et problèmes constatés, solutions possibles

À l'issue de cette première période, nous avons pu livrer un prototype fonctionnel et ce malgré des difficultés liées à la conception. En revanche, l'instruction `affiche`, initialement prévue pour cette itération, n'a pas été implémentée, faute de temps et de ressources disponibles, et a donc été reportée à l'itération suivante. Cette instruction n'étant pas critique pour le fonctionnement de l'application, nous pouvons donc considérer la gravité de ce retard comme minime.

3.8.4 Propositions de modification de la planification prévisionnelle pour tenir compte des corrections à apporter

Afin de tenir compte de ce léger retard, nous rajouterons l'implémentation de l'instruction `affiche` aux tâches à réaliser pour la prochaine itération. En plus des programmes, et l'adaptation de l'analyseur à interagir avec, nous en profiterons pour implémenter un maximum de fonctionnalités, ce qui nous permettra de rattraper le léger retard pris sur la planification originelle.

Nous laisserons toutefois l'instruction `si . . vaen` et les expressions conditionnelles qu'elle utilise de côté. En effet, ces dernières nécessiteront probablement un *refactoring* de la classe `Expression` et de ses dérivées.



En suivant la planification ci-dessus, la deuxième itération devrait donc occuper un total de 42 heures de travail, soit une durée similaire à la précédente. L'objectif du prochain prototype sera de rajouter la majorité des fonctionnalités de l'interpréteur LIR, dont notamment l'arithmétique entière et toute la partie d'édition et d'exécution de programmes.

3.8.5 Comptes-rendus des réunions projets de la période

Voir

- Compte rendu de la réunion MOE du 15 avril
- Compte rendu de la réunion MOE du 19 avril
- Compte rendu de la réunion MOE du 22 avril

— Compte rendu de la réunion MOE du 26 avril

3.8.6 Compte-rendu du comité de pilotage de la période

Voir

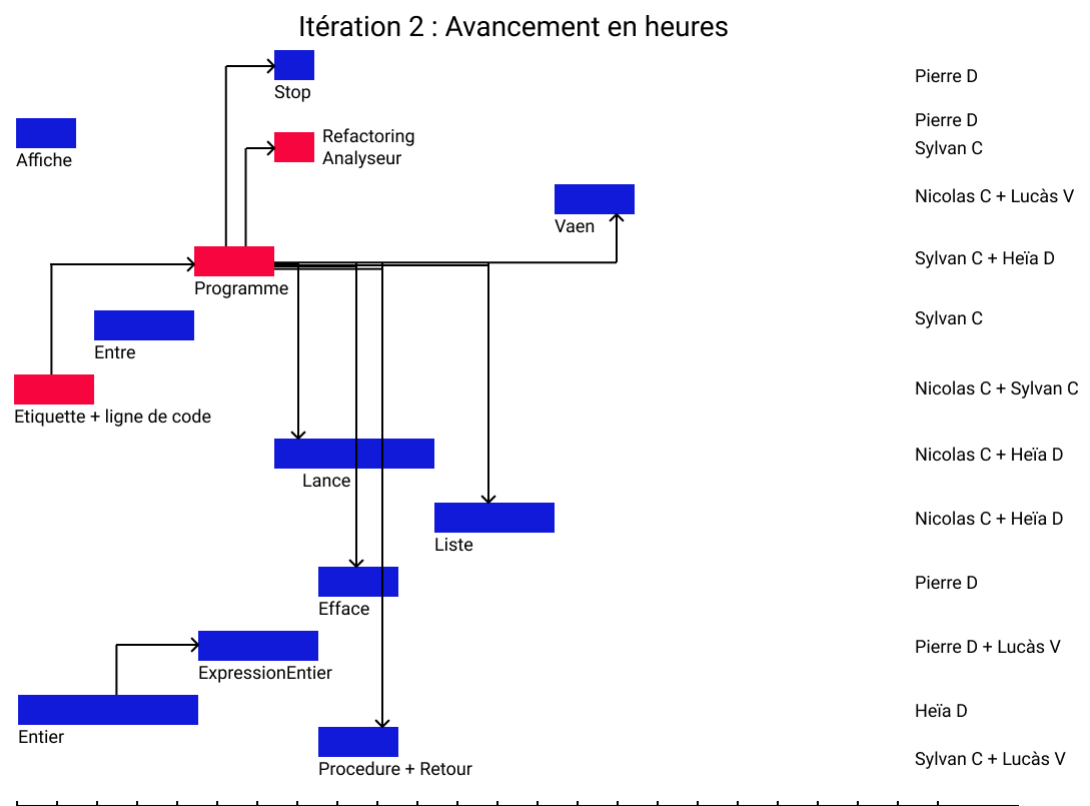
- Compte rendu de la réunion MOA du 4 mai
- Compte rendu de la réunion MOA du 10 mai

3.8.7 Planification prévisionnelle révisée pour les périodes suivantes (en fonction des décisions prises)

Exception faite de l'ajout de *affiche*, la planification de la deuxième itération ne dévie pas de l'ordonnancement initial. Elle a donc été entérinée à l'unanimité.

3.9 Suivi du projet pour la seconde itération

3.9.1 Suivi d'avancement et mesure des écarts par rapport au prévisionnel revu lors de la période précédente



Au total, la seconde itération aura englobé un temps de travail total de 50,5 heures, soit 8,5 heures ou 4,25 jours.homme de retard par rapport à la planification initiale. Ce retard s'explique dans des difficultés à

gérer les dépendances avec la classe `Programme` et à écrire des tests concluants. Les instructions `lance` et `liste` sont celles nous ayant posé le plus de problèmes.

Nous pouvons également remarquer, à travers ce graphe, l'étalement dans le temps des tâches comparé à la planification précédente. Cela est dû à un inconvénient du travail en binôme. En effet, le nombre de membres de notre équipe étant fini, il nous a fallu par moments attendre que certains se libèrent d'une tâche pour entamer la suivante.

En entreprise, cet inconvénient est en général mitigé par la quotité horaire fixe et convenue dans le contrat de travail. Dans le cadre d'un travail étudiant, nous avons aussi dû jouer avec les disponibilités de chacun, en plus des contraintes imposées par le travail à deux. En dépit de ces facteurs, nous n'avons aucun écart majeur à déplorer dans notre ordonnancement.

3.9.2 Identification des principaux écarts et problèmes constatés, solutions possibles

À ce stade du projet, aucun gros écart n'est constaté. Nous avons cependant été confrontés à des difficultés à mener nos tests correctement, ce qui a fait augmenter la durée de certaines tâches (non critiques).

3.9.3 Propositions de modification de la planification prévisionnelle pour tenir compte des corrections à apporter

La seconde itération n'ayant pas pris de retard, aucune modification ne sera apportée à la planification de la troisième. Le dernier prototype inclura donc toutes les fonctionnalités manquantes à ce stade, à savoir la lecture et l'écriture de fichiers textes pour la sauvegarde de programmes, ainsi que les expressions et sauts conditionnels. Nous profiterons du temps restant pour effectuer une revue générale du code et des jeux de tests, rédiger le manuel d'utilisation et achever ce plan projet.

3.9.4 Comptes-rendus des réunions projets de la période

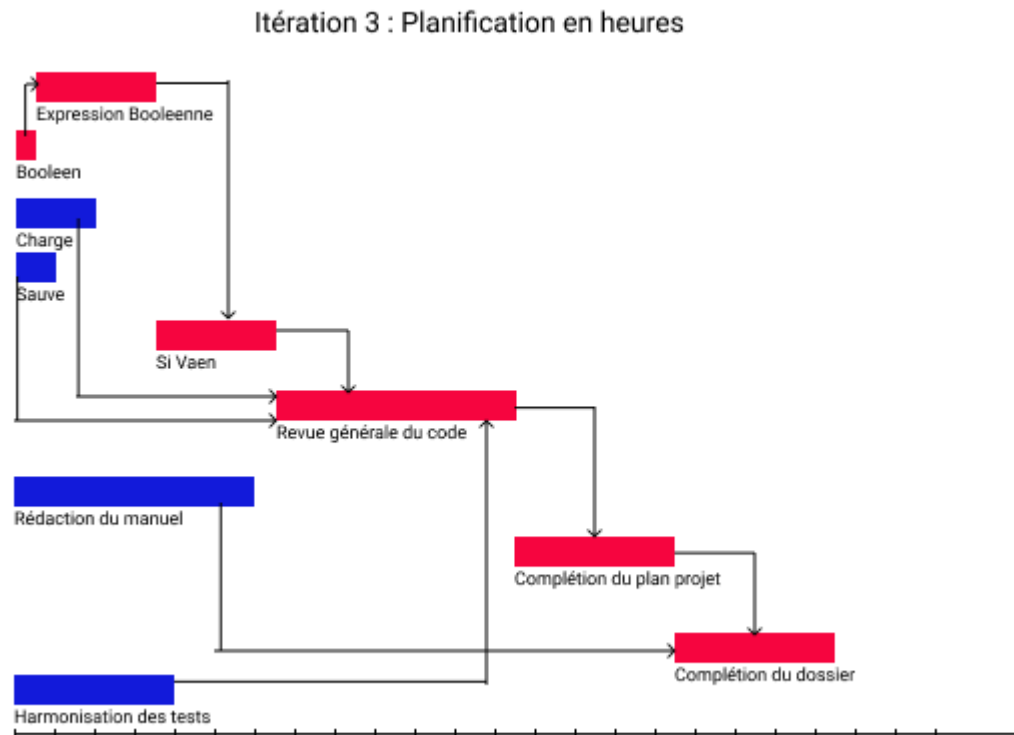
Voir Compte rendu de la réunion MOE du 11 mai

3.9.5 Compte-rendu du comité de pilotage de la période

Voir

- Compte rendu de la réunion MOA du 10 mai
- Compte rendu de la réunion MOA du 18 mai
- Compte rendu de la réunion MOA du 19 mai

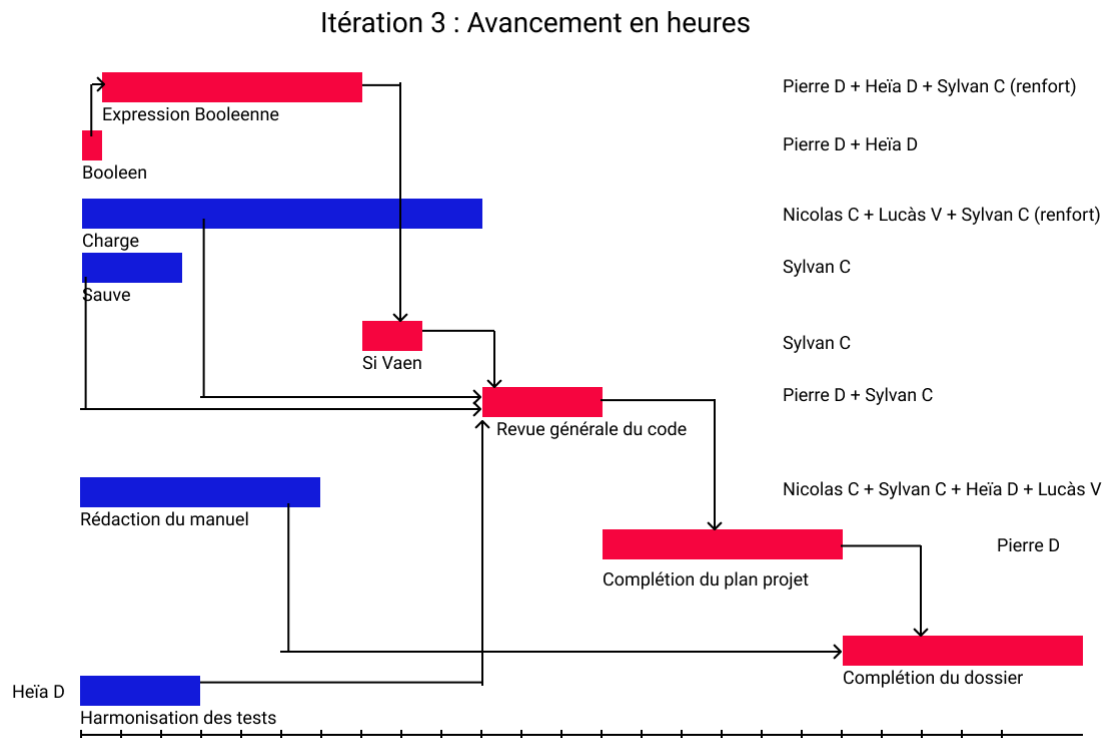
3.9.6 Planification prévisionnelle révisée pour les périodes suivantes (en fonction des décisions prises)



Au vu des estimations effectuées, la troisième itération devrait couvrir un temps de travail de 79 heures, soit 39,5 jours.homme. Ce volume important est dû aux nombreuses tâches à effectuer avec l'équipe complète. Cette itération comportant de nombreuses tâches critiques (le 28 mai marquant le dernier jalon de la phase de développement et la livraison du prototype final), nous avons délibérément pris des estimations potentiellement larges afin de nous assurer suffisamment de temps pour mener ces tâches à bien.

3.10 Suivi du projet pour la troisième itération

3.10.1 Suivi d'avancement et mesure des écarts par rapport au prévisionnel revu lors de la période précédente



Cette troisième itération offre un total d'heures de travail porté à 107 contre les 79 prévues. Cela équivaut donc à un retard de 28 heures, soit l'équivalent de 14 jours.homme. Ce retard s'explique par deux problèmes d'envergure auxquels nous avons été confrontés. Ces deux problèmes ont eu un impact direct sur le cheminement critique de l'ordonnancement et a par conséquent entraîné un retard qu'il a fallu compenser dans la revue de code. Nous reviendrons plus en détail sur ces problèmes un peu plus bas.

Voici donc un bilan des totaux des heures de travail estimées, confrontées aux heures de travail réelles sur l'ensemble des trois itérations du projet.

Itération	Estimation	Réel	Ecart
n°1	46	61,5	15,5
n°2	42	50,5	8,5
n°3	79	107	28
total	167	219	52

Au total, nous constatons donc un écart de 52 heures de travail par rapport à nos estimations. Cela correspondrait à un écart de 7,43 jours.homme.

3.10.2 Identification des principaux écarts et problèmes constatés, solutions possibles

Le premier des deux problèmes mentionnés plus haut concerne l'écriture des algorithmes de reconnaissance des expressions booléennes. Ce problème aurait pu être évité via l'usage d'expressions rationnelles (*regex*). Malheureusement, cette notion est arrivée tard dans notre apprentissage de la théorie des langages et nous n'avons pas eu l'opportunité de pouvoir l'implémenter à temps.

Le deuxième problème, plus grave, a concerné le fonctionnement de la commande charge. Il nous est apparu en effet que la plupart du code de cette commande réutilisait celui de la classe Analyseur. Il nous est alors apparu une faiblesse dans notre conception, étant donné que si ce code est réutilisé à cet endroit, cela signifie qu'une factorisation a été mal faite. Une solution serait de repenser l'analyseur comme un objet chargé d'analyser une seule ligne de code et de le dissocier de la `mainLoop()` de l'interpréteur.

Malheureusement, cette conclusion ne nous est arrivée que tardivement et il a fallu que nous prissions une décision. Nous avons donc opté pour la solution la moins coûteuse en durée afin de ne pas accumuler de retard et de pouvoir terminer le projet dans les temps. Dans une hypothétique phase de maintenance du logiciel, nous pensons qu'implémenter la solution évoquée ci-dessus serait une des premières tâches à accomplir dans le but de proposer un logiciel fonctionnel et plus facilement maintenable.

3.10.3 Compte-rendu du comité de pilotage de la période

Voir

- Compte rendu de la réunion MOA du 18 mai
- Compte rendu de la réunion MOA du 19 mai

3.10.4 Idées d'améliorations

L'interpréteur LIR que nous livrons à l'issue de ce projet respecte les spécifications du cahier des charges. Au cours de sa conception et de son implémentation, nous nous sommes efforcés, avec plus ou moins de succès, à conserver un code le plus simple et réutilisable possible, en accord avec l'état de nos connaissances et de notre expérience du développement logiciel à cette période de notre formation.

Afin de rapprocher davantage notre travail de ce que l'on pourrait attendre d'une équipe professionnelle, nous avons réfléchi à un bagage de connaissances et de savoirs faire dont nous aurions souhaité disposer dès les premiers instants de la conception. Ce dernier nous permettra à l'avenir de surmonter plus facilement les difficultés auxquelles nous avons été confrontés.

D'abord, l'utilisation du *pattern matching* et des expressions rationnelles citées plus haut nous aurait grandement simplifié la conception et surtout le codage de notre analyseur. Nous ne nous priverons d'ailleurs pas de nous en servir lors de projets ultérieurs.

De plus, l'étude des *design patterns*, des structures de données et de leur usage nous simplifiera grandement la tâche de la conception à l'avenir. Avec le recul que nous a procuré cette expérience, nous constatons que nous aurions pu être bien plus efficaces lors des phases de conception avec ces outils.

Ensuite, les tests que nous avons réalisés l'ont été avec l'exécuteur universel développé au cours des TDs de programmation du deuxième semestre. Si cet outil ne nous a à aucun moment fait défaut, nous lui aurions certainement préféré JUnit3. Sachant que les tests occupent au moins la moitié de la phase de développement d'un logiciel, nous aurions certainement gagné en efficacité avec cet outil.

Pour finir et pour aller plus loin, à présent que l'interpréteur fonctionne, il pourrait être intéressant, comme projet d'été par exemple, de développer un IDE dédié à notre interpréteur LIR. Cette application proposerait une interface fenêtrée, l'autocomplétion, l'utilisation des commandes à travers des menus déroulants ou encore l'affichage du contexte en temps réel.

Chapitre 4

Bilan

Outre le développement en équipe d'une solution logicielle en programmation orientée objet, ce projet tutoré de fin d'année nous a donné un aperçu de ce que pouvait être un travail en équipe sous la supervision d'une maîtrise d'ouvrage.

Nous avons ainsi pu nous familiariser avec les postes de chef de projet, de gestionnaire de configuration et de secrétaire de projet. Au cours des échanges avec notre enseignant tuteur, nous avons été amenés à comprendre les enjeux de la planification et de l'ordonnancement des tâches, mais aussi l'importance de la communication écrite de d'assurer la traçabilité de l'information. Enfin, l'utilisation d'un outil commun nous a appris l'importance de conserver un paramétrage et une configuration équivalente d'un poste à l'autre et de conserver un historique des versions de notre code.

Dans le monde de l'entreprise, pouvoir assurer un tel niveau de rigueur et de transparence vis-à-vis de ses client est primordial dans l'élaboration d'une relation de confiance. Cela permet de plus d'attester de sa bonne foi et de prouver que rien n'a été laissé au hasard dans l'éventualité où le projet n'aboutirait pas selon les critères dictés par la charte de projet.

Deuxième partie

Annexes

Annexe A

Sujet Interpréteur LIR

Interpréteur de langage LIR

Projet proposé par Frédéric Barrios pour des groupes de 4 personnes

Ce sujet vous propose d'écrire un interpréteur d'un langage de programmation simple.
Cet interpréteur sera nommé **LIR** (pour Langage IUT de Rodez) dans la suite du sujet.

Un interpréteur est un automate enchaînant les tâches suivantes : analyse lexico-syntaxique d'une ligne de commande puis interprétation.

Une ligne entrée par un utilisateur sera donc : soit une commande à exécuter immédiatement, soit une ligne de programme à mémoriser pour une exécution ultérieure. Une ligne de programme se distinguera d'une ligne de commande par le fait qu'elle sera toujours précédée d'un "numéro d'ordre" appelé aussi « étiquette ».

Syntaxe d'une ligne de programme :

numéro **instruction** argument(s)

Exemple : 10 **var** a=110

Syntaxe d'une commande :

commande argument(s)

Exemple : **sauve** nomfichier

Les instructions et les commandes sont des mots clés du langage.

1 LES ELEMENTS DU LANGAGE

1.1 Mots clés

Dans la suite les mots clés du langage LIR, c'est-à-dire les instructions et les commandes, seront présentés en **gras**.

Leur liste est la suivante par ordre lexicographique :

affiche charge debut defs efface entre fin lance liste procedure retour sauve si stop vaen var

Ils ne peuvent pas être redéfinis par le programmeur et ne peuvent donc pas servir en tant qu'identificateurs.

1.2 *Etiquettes*

Les étiquettes sont des numéros servant à repérer les lignes d'un programme et à les mettre en ordre. Ces numéros sont des entiers positifs entre 1 et 99999.

L'usage est en général de numéroté initialement les lignes de 5 en 5, ou bien de 10 en 10, afin de laisser de la place pour une insertion éventuelle.

Dans la suite les étiquettes seront désignées par <etiquette>

1.3 *Constantes littérales*

Elles seront de 2 types : entier signé ou chaînes de caractères.

Les valeurs entières sont exprimées en base 10 et donc formées des caractères +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Leur valeur sera comprise dans l'intervalle $[-2^{31} = -2147483648, 2^{31} - 1 = 2147483647]$

Exemples : 12 -243 +15

Les chaînes seront entourées de guillemets et limitées à 70 caractères.

Exemple : "Ceci est une chaîne"

1.4 *Identificateurs*

Les noms identifiant des entiers commenceront obligatoirement par une lettre qui sera suivie d'au plus 24 lettres ou chiffres.

Exemples : `alpha` `indice1` sont des identificateurs corrects pour des variables entières

Les noms identifiant des chaînes de caractères commenceront par le symbole \$, suivi d'une lettre, qui sera suivie d'au plus 24 lettres ou chiffres.

Exemples : `$prenom` `$ligne0` sont des identificateurs de chaîne corrects

Les identificateurs n'ont pas besoin d'être déclarés : leur première utilisation suffit à les créer.

Si cette première utilisation n'est pas une affectation ou une entrée leur valeur sera choisie arbitrairement et cela conduira à un comportement aléatoire...

=> il incombe donc au programmeur de bien maîtriser les initialisations.

Toutes les données utilisées seront variables. Leurs valeurs sont conservées et modifiables durant la session de l'interpréteur. Elles ne sont détruites que par la commande **debut**.

La portée des variables est donc globale (aucun principe de localité).

Leur type est choisi lors de leur première utilisation et il ne peut changer (pas de transtypage)

=> si une instruction suivante dénote une incompatibilité de type alors une erreur intervient (pas de conversion implicite)

Dans la suite un identificateur sera repéré par <identificateur>

1.5 Expressions arithmétiques

Syntaxe d'une expression : opérande1 opérateur opérande2

Les expressions concernent donc toujours deux opérandes séparés par un opérateur (notation infixe). La présence d'espace (séparateur neutre) n'est pas obligatoire (mais conseillée)

Un opérande est soit une constante littérale, soit un identificateur.

L'opérateur est un caractère symbolisant les opérations arithmétiques courantes : + - * / %

+ = addition entière

- = soustraction entière

* = multiplication entière

/ = quotient de la division entière

% = reste de la division entière

Exemples : n + p a / 100 a * a

1.6 Expressions sur chaînes

La seule opération utilisée sur les chaînes est la concaténation.

Syntaxe : chaîne1+chaîne2

où chaîne1 et chaîne2 sont des opérandes chaînes c'est-à-dire constantes chaînes ou identificateurs commençant par \$.

1.7 Expressions logiques

Les expressions logiques ne seront utilisées qu'avec l'instruction **si**.

Syntaxe d'une expression logique : opérande1 oprel opérande2

Les expressions logiques concernent donc toujours deux opérandes séparés par un opérateur relationnel (notation infixe). Un opérande est soit une constante, soit un identificateur.

L'opérateur relationnel oprel est un symbole parmi : = <> < <= > >=

= représente le test d'égalité

<> représente l'inégalité

< infériorité stricte

> supériorité stricte

<= inférieur ou égal

>= supérieur ou égal

Ces opérateurs ont le sens habituel pour les entiers.

Pour les chaînes, c'est l'ordre lexicographique habituel qui sera utilisé.

Exemples : "TATA" < "TU" (car A < U dans l'alphabet)

"LONG" < "LONGUEUR" (la chaîne la plus longue est supérieure)

2 LES INSTRUCTIONS DE PROGRAMMATION

Les instructions de programmation permettent de faire effectuer des actions. Elles peuvent être utilisées comme commande immédiate ou comme ligne d'un programme (lorsque précédée par une <étiquette>).

1. Affectation : **var**

Syntaxe: **var** <identificateur>=<expression>

Sémantique : affecte la valeur de l'expression à la variable nommée par l'identificateur.

2. Entrée d'une valeur depuis le clavier: **entre**

Syntaxe: **entre** <identificateur>

Sémantique : attend que l'utilisateur entre une valeur sur l'entrée texte courante, l'évalue, puis l'affecte à la variable <identificateur>.

3. Sortie d'une valeur à l'écran : **affiche**

Syntaxe1 : **affiche** <expression>

Sémantique : évalue la valeur de l'expression et l'affiche sur la sortie texte courante.

Syntaxe2 : **affiche**

Sémantique : provoque un saut de ligne sur la sortie texte courante.

4. Branchement impératif : **vaen**

Syntaxe: **vaen** <étiquette>

Sémantique : continue l'exécution à partir du numéro spécifié par étiquette.

5. Branchement conditionnel : **si ... vaen**

Syntaxe: **si** <expression_logique> **vaen** <étiquette>

Sémantique : si la condition est vraie alors l'exécution continuera à partir du numéro de ligne spécifié par l'étiquette, sinon l'exécution continuera en séquence.

6. Appel d'une procédure : **procedure**

Syntaxe: **procedure** <étiquette>

Sémantique: L'exécution est transférée au numéro d'étiquette spécifié et reprendra en séquence lorsque la procédure sera terminée. Il n'y a pas de paramétrage de l'interface

=> la communication repose sur des « effets de bord » concernant les identificateurs.

=> les identificateurs déclarés sont donc tous à portée globale même s'ils ne semblent utilisés que dans une seule procédure (pas de principe de localité).

7. Fin d'une procédure : **retour**

Syntaxe: **retour**

Sémantique : cette instruction, rencontrée après un appel de procédure, provoque un retour à l'instruction qui suit son appel.

=> attention aux appels en cascade...

Remarque : une instruction **retour** ne peut être rencontrée que suite à une instruction **procedure** exécutée précédemment ; dans le cas contraire une erreur sera signalée.

8. Arrêt du programme : **stop**

Syntaxe: **stop**

Sémantique : Cette instruction arrête l'exécution du programme. Elle en marque donc la fin.

3 COMMANDES GENERALES

De façon à pouvoir manipuler plus facilement ses programmes, l'utilisateur disposera de commandes. Ces commandes ne sont pas des instructions et ne peuvent pas faire partie d'un programme.

3.1 *La commande de nettoyage de l'environnement: debut*

Syntaxe: **debut**

Sémantique : efface toutes les lignes de programme mémorisées ainsi que tous les identificateurs mémorisés.

3.2 *La commande d'effacement de ligne(s) : efface*

Syntaxe: **efface** <etiquette_debut>:<etiquette_fin>

Sémantique : efface toutes les lignes de programme dont le numéro d'étiquette est dans la plage comprise entre <etiquette_debut> et <etiquette_fin>.

3.3 *La commande d'affichage des lignes d'un programme : liste*

Syntaxe 1: **liste**

Sémantique : affiche toutes les lignes de programme mémorisées dans l'ordre croissant des numéros de ligne.

Syntaxe 2: **liste** <etiquette_debut>:<etiquette_fin>

Sémantique : affiche les lignes mémorisées dont les numéros d'étiquettes sont compris entre <etiquette_debut> et <etiquette_fin> (dans l'ordre croissant des numéros de ligne).

3.4 *La commande de liste des identificateurs définis : defs*

Syntaxe: **defs**

Sémantique : affiche la liste des identificateurs définis durant la session avec leur valeur (le programmeur voit donc le contexte du programme)

3.5 *La commande de lancement de l'exécution : lance*

Syntaxe 1: **lance**

Sémantique : démarre l'exécution d'un programme à partir de son plus petit numéro d'étiquette.

Syntaxe 2: **lance** <etiquette>

Sémantique : démarre l'exécution d'un programme à partir de l'étiquette spécifiée.

Remarque : une instruction **vaen** utilisée comme commande aura un effet similaire.

3.6 *La commande de sauvegarde dans un fichier : sauve*

Syntaxe: **sauve** cheminFichier

Sémantique : sauvegarde les lignes de programme dans le fichier texte indiqué en argument

3.7 *La commande de chargement depuis un fichier : charge*

Syntaxe: **charge** cheminFichier

Sémantique : charge dans le contexte les lignes de programme sauvegardées dans le fichier texte indiqué en argument

3.8 *La commande de fin de session : fin*

Syntaxe: **fin**

Sémantique : quitte l'interpréteur...

4 TRAVAIL A EFFECTUER

Réaliser un logiciel en langage java permettant à un programmeur d'utiliser un interpréteur de Langage LIR pour écrire et exécuter des programmes.

L'interpréteur affiche un caractère d'invite au programmeur afin qu'il puisse entrer une instruction ou une commande sur une ligne de commande.

Suivant la ligne de commande entrée, l'interprète :

- affiche un message d'erreur si la syntaxe de la commande ou de l'instruction ne permet pas de l'interpréter
- affiche un message d'erreur si la commande ou l'instruction conduit à une erreur d'exécution
- exécute la commande ou l'instruction puis affiche le résultat de la commande ou de l'instruction si elle produit un résultat à afficher
- exécute la commande ou l'instruction puis affiche « ok » en cas de commande ou d'instruction correcte ne générant pas de sortie particulière
- affiche « ok » en cas de mémorisation d'une ligne de programme correcte commençant par un numéro de séquence correct, indiquant ainsi que la ligne de programme a correctement été « insérée » dans la séquence.

Ainsi l'interpréteur donne toujours un « feedback » au programmeur lui permettant de savoir si la ligne de commande a été correctement interprétée ou s'il y a un problème. En cas d'erreur, un message expliquant le problème le plus clairement possible doit être affiché.

5 EXEMPLE D'UNE SESSION SOUS L'INTERPRETEUR LIR

(Lancement de LIR)

Interpréteur Langage IUT de Rodez, bienvenue !

Entrez vos commandes et instructions après l'invite ?

? charge bonjour.lir

ok

? liste

10 affiche "Entre ton nom : "

20 entre \$nom

30 affiche "Bienvenue "+\$nom

40 stop

? 40 var an=2021

ok

? 50 affiche "Quelle est ton année de naissance : "

nok : instruction inconnue : affiche

? 50 affiche "Quelle est ton année de naissance ? "

ok

? 60 entre

nok : paramètre obligatoire pour l'instruction entre

? 60 entre naissance

ok

? 200 stop

ok

? 70 affiche "Tu as autour de "

ok

? 65 si naissance > an vaen 50

ok

? 80 affiche an-naissance

ok

? 90 affiche "ans "

ok

? 100 affiche

ok

? 35 affiche

ok

```

? liste
10 affiche "Entre ton nom : "
20 entre $nom
30 affiche "Bienvenue "+$nom
35 affiche
40 var an=2021
50 affiche "Quelle est ton année de naissance ? "
60 entre naissance
65 si naissance > an vaen 50
70 affiche "Tu as autour de "
80 affiche an-naissance
90 affiche "ans "
100 affiche
200 stop
? lance
Entre ton nom : marc
Bienvenue marc
Quelle est ton année de naissance ? 2001
Tu as autour de 20 ans
? defs
an = 2021
naissance = 2001
$nom = "marc"
? sauve age.lir
Le programme age.lir a été sauvegardé.
? debut
ok
? liste
ok
? fin
Au revoir, à bientôt !

```

Annexe B

Gestion de la configuration de l'Interpréteur LIR



GESTION DE LA CONFIGURATION INTERPRÉTEUR DU LANGAGE LIR

PROJET PROPOSÉ PAR FRÉDÉRIQUE BARRIOS

version : 26 mai 2021

Nicolas CAMINADE, Sylvan COURTIOL,
Pierre DEBAS, Heïa DEXTER,
Lucàs VABRE

Chapitre 1

Gestion de la configuration de l'Interpréteur LIR

Introduction

Ce document a pour but de confirmer par écrit la configuration logicielle choisie pour le projet.

Le contenu de ce document n'est pas fixé et des changements peuvent être apportés. Cependant ce document doit être connu et suivi par les membres du groupe. En cas de modifications, une annonce sur discord sera faite.

Pour toute question ou suggestion se référer au gestionnaire de configuration (présentement Sylvan COURTIOL).

1 Logiciels de développement

1.1 Environnement de Développement Intégré

Eclipse JEE (version 2020-12)
JDK 15

1.2 Contrôle des versions du code

Git avec dépôt sur GitHub. (Un apprentissage est nécessaire donc pour commencer certaines libertés sont possibles).

1.3 Organisation

Via le site Trello (non utilisé pour le moment).

1.4 Modélisation

La modélisation UML sera effectuée sur Modelio Open Source (version 4.1).

2 Logiciels généraux

2.1 Communication

Les communications formelles sont effectuées via les mails de l'IUT (généralement par le chef de projet) avec les autres membres du projet en CC.

Serveur discord spécifique au projet pour la communication écrite ou vocale de la MOE.

Google Meet pour les réunions avec les personnes autres que MOE. Adaptable à ce qui convient le mieux à cette personne.

2.2 Éditeur de texte

Le traitement de texte sera fait sous LaTeX notamment avec la distribution MiKTeX et l'IDE TexStudio. Les documents texte sont partagés en PDF ou version papier à la MOA/MOE et en format modifiable .tex seulement à la MOE via la solution de partage distant des fichiers (voir sous-section suivante).

2.3 Partage distant des fichiers

Les partages de tous les fichiers généraux et codes sources se feront sur GitHub via le site, le logiciel GitHub desktop ou git. Il y aura également une intégration Discord informant des commits.

3 Sécurité

Si possible tous les membres du groupe auront les mêmes droits sur les fichiers communs. En conséquence aucun membre du groupe ne doit donner des droits sur des fichiers à une personne extérieure au projet (autre que MOA).

Les sauvegardes du dépôt GitHub (contenant toutes les données du projets) seront effectuées régulièrement (tous les 1 ou 2 jours) par le gestionnaire de configuration. Toutes données qui ne sont pas dans le dépôt sont à la responsabilité de chacun. Les sauvegardes sont enregistrées en local par le gestionnaire de configuration ainsi que sur le Google drive partagé du projet.