Retour d'expérience sur une UE Projet en licence informatique

Aurélien Esnard et Nicolas Bonichon [1]

Introduction

Cet article présente un retour d'expérience sur une unité d'enseignement (UE) projet en licence informatique à l'université de Bordeaux. Dans le cadre de cette UE, nous avons mis en place plusieurs stratégies pour atteindre efficacement nos objectifs pédagogiques. Nous ne prétendons pas que notre approche est la meilleure, mais nous pensons néanmoins que ce retour d'expérience pourra nourrir la réflexion d'autres collègues.

La suite de l'article est organisée de la manière suivante. Dans un premier temps, nous rappellerons le contexte de cette UE et pourquoi nous avons évolué vers la réalisation d'un projet sur l'année. Ensuite, nous verrons comment nous avons organisé cette UE projet. Enfin, nous présenterons les outils techniques que nous avons utilisés pour mettre en place cette UE projet.

1. LaBRI, CNRS, université de Bordeaux, Bordeaux INP, France.

DOI:10.48556/SIF.1024.22.73 **1024** – Bulletin de la SIF, numéro 22, novembre 2023, p. 73–87

Contexte

Positionnement de l'UE dans le cursus

L'UE considérée ici est une UE à 6 crédits (ECTS), qui se déroule sur toute l'année, soit 40 heures de présentiel au total, avec 3 cours et 12 séances de TD (sur machine) de 1h20 chaque semestre. Au final, c'est un temps très court chaque semaine, ce qui implique beaucoup de travail des étudiants en dehors des séances encadrées. Nous avons un effectif assez important, avec environ 250 étudiants en licence informatique à gérer. Nous nous retrouvons ainsi avec 12 groupes de TD et une dizaine de chargés de TD qui se répartissent ces groupes. Dans chaque groupe de TD, les étudiants se répartissent en *équipe* de trois. Ainsi, nous avons environ 75 équipes à gérer.

Cette UE se trouve en deuxième année de licence informatique (L2). Elle a quelques prérequis en première année (L1) en algorithmique élémentaire, en programmation C, et un peu en programmation web. Notons que lors du semestre d'automne (S3), les étudiants suivent en parallèle une UE de Programmation C, qui fait suite à l'UE Initiation à la programmation C (S2). Après leur L2, nos étudiants suivent en L3 une autre UE projet de développement logiciel.

Objectifs pédagogiques

L'objectif pédagogique principal est de former nos étudiants à des méthodes et des bonnes pratiques de *développement logiciel*, et de leur apporter une certaine maîtrise des outils techniques, afin qu'ils puissent participer à des projets logiciels en entreprise. Cela se décline donc en une liste de compétences assez génériques :

- renforcer le niveau en programmation et en algorithmique ;
- apprendre à travailler en équipe ;
- savoir faire des tests unitaires ;
- savoir évaluer la couverture de ses tests ;
- savoir écrire du code propre ;
- savoir documenter son code ;
- savoir déboguer efficacement son code ;
- savoir rendre son code maintenable ;
- comprendre l'importance du découpage modulaire et de l'interopérabilité ;
- respecter un cahier des charges.

Adossés à ces premiers objectifs, on a également des objectifs plus techniques, liés à des outils :

- maîtriser un IDE (VS Code) ;
- maîtriser le processus de compilation (gcc, make) ;
- maîtriser un outil de gestion de version (Git) et les fonctionnalités de base d'une *forge* logicielle (GitLab) ;
- maîtriser un système de construction logicielle (CMake) ;

— savoir s'approprier une bibliothèque logicielle externe (SDL) ;
— maîtriser quelques technologies Web (HTML, CSS, JS, WASM) ;
— rédiger un rapport en LaTeX.

Pourquoi un projet sur toute l'année ?

Historiquement, il s'agissait d'une UE qui avait à peu près le même contenu pédagogique, mais avec des modalités un peu plus classiques, c'est-à-dire avec 1h20 de cours et 2h40 de TD chaque semaine, pendant un semestre. On présentait essentiellement la même liste de méthodes et de technologies à maîtriser. La mise en œuvre de tout cela se faisait sous la forme de TD indépendants les uns des autres et d'un mini-projet à rendre en fin du semestre (puisque cette UE se déroulait sur un seul semestre). Malheureusement, nous avons remarqué que les étudiants avaient du mal à s'approprier ces méthodes et ces technologies. Une des causes principales était, selon nous, que les étudiants ne voyaient pas l'intérêt de ces méthodes et de ces technologies lorsqu'elles étaient présentées en cours et manipulées en TD. En effet, les réticences des étudiants étaient les suivantes :

— « *Monsieur, moi quand je code ma petite bidouille, je fais du* quick and dirty *et ça marche très bien comme ça* » ;
— « *Je n'ai pas besoin d'apprendre tous vos outils. Pour compiler, je fais simplement* `gcc *.c` *et ça marche bien* » ;
— « *Pour debugger, pas besoin de* `gdb`*, je rajoute des* `printf` *de temps en temps, ça suffit !* » ;

Et pour le reste, c'est un peu la même chose. Ces remarques sont certes caricaturales, mais elles sont néanmoins tout à fait légitimes. Pour de tout petits programmes, c'est une méthode suffisamment efficace de développement. Et comme nos étudiants ont principalement développé des petits programmes, principalement seuls, leur expérience personnelle leur donnait raison. C'est pourquoi nous avons décidé de migrer vers un projet plus conséquent se déroulant sur toute l'année. L'idée était de mettre les étudiants dans une situation où il devient utile (sinon nécessaire) d'utiliser les méthodes et les outils présentés. Le fait que le projet se déroule sur toute l'année nous permet de travailler sur un temps de développement plus long, qui permet d'atteindre une taille du logiciel suffisante pour rendre nos objectifs pédagogiques pertinents.

Lorsque le projet avance, la maîtrise des outils n'est plus simplement utile, elle devient nécessaire du fait de la montée en complexité du logiciel. Ici, le *timing* est assez difficile à trouver. Car, si on avance trop vite, les étudiants n'ont pas le temps d'acquérir les compétences associées et ils s'enlisent dans le projet. Chaque année, nous ajustons ce *timing* en fonction de la difficulté intrinsèque du projet, mais aussi de l'avancement de la promotion.

Organisation pédagogique

Au fil des années, nous avons convergé vers une organisation et un *scénario* qui se veut motivant pour les étudiants et qui permet d'introduire progressivement les méthodes et les outils associés.

Sujet du projet

Le projet à réaliser est en fait un prétexte (un fil rouge) pour introduire les outils et méthodes de développement. Il s'agit d'un petit jeu de logique (type *Sudoku*), qui change chaque année. L'important est que le projet soit suffisamment simple pour que les étudiants puissent le réaliser en autonomie, mais suffisamment complexe pour qu'ils aient besoin d'utiliser les outils et méthodes de développement présentés dans cette UE.

Nous avons choisi comme thématique un petit jeu, mais cela pourrait être tout autre chose, tant qu'il est possible de faire varier le choix du projet chaque année tout en gardant la même structure. Ce dernier point est particulièrement important si l'on souhaite pouvoir réutiliser au maximum les supports pédagogiques d'une année sur l'autre. Dans notre cas, il s'agit d'un puzzle logique, qui se joue à un seul joueur sur une grille 2D. Notons par ailleurs que dans le contexte de notre UE, le sujet du projet est le même pour toutes les équipes. Ces dernières années, nous avons choisi des jeux parmi ceux présentés dans la collection de *Simon Tatham*[2]. Cette année, nous avons choisi le jeu *Takuzu* (également connu sous le nom *Unruly* dans cette collection). Dans ce jeu, le joueur part d'une grille où se trouvent déjà des cases noires et blanches fixes. Le joueur doit compléter la grille en respectant les règles suivantes : chaque ligne et chaque colonne doit contenir autant de cases noires que de cases blanches et il ne doit pas y avoir plus de deux cases blanches ou noires consécutives. La figure 1 illustre plusieurs exemples de jeux réalisés ces dernières années avec différents types d'interface utilisateur que les étudiants devront implémenter au cours de l'année : mode texte, mode graphique, Android, Web.

Le scénario pédagogique

Le scénario pédagogique que nous avons mis au point au fil des années est décrit dans ce qui suit (voir figure 2).

Étape a. Au début, nous fournissons aux étudiants le noyau fonctionnel du jeu déjà implémenté sous forme d'une bibliothèque compilée (un fichier `game.a`, sans le code source) avec une API documentée (un fichier `game.h`), ainsi que la documentation complète générée par *Doxygen*. La première étape consiste simplement à implémenter le jeu en mode texte. Cette première étape, qui ne comporte aucune difficulté algorithmique, permet aux étudiants de se familiariser avec le jeu et de se remettre dans la programmation en langage C. C'est également le prétexte pour parler de

2. `https://www.chiark.greenend.org.uk/~sgtatham/puzzles`.

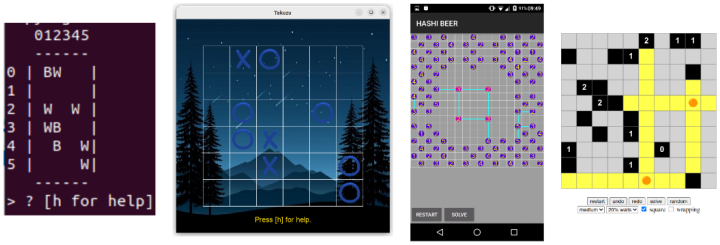

FIGURE 1. Différents exemples de jeux réalisés ces dernières années : *Takuzu, Bridges, Light Up*.

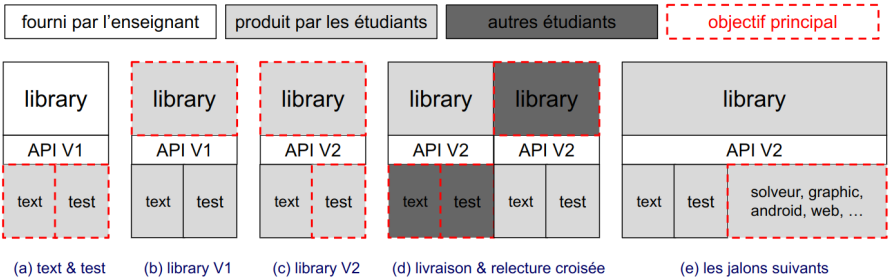

FIGURE 2. Scenario pédagogique, illustrant les principales étapes dans la réalisation du projet par les étudiants.

compilation, d'abord avec `gcc`, puis avec `make` et enfin avec `cmake`. Parallèlement à cela, nous introduisons Git, nous formons les équipes et les étudiants commencent à travailler en équipe avec Git.

Une fois cela établi, nous leur demandons d'écrire des tests unitaires de la bibliothèque fournie. C'est la première fois qu'ils ont à développer des tests et ils sont un peu « frileux » à ce sujet. Pour les motiver, nous leur donnons des versions buggées de notre bibliothèque et ils doivent trouver ces différents bugs avec leurs tests, ce qui les encourage à les améliorer. Comme nous avons un peu d'expérience, les bugs que nous introduisons sont à peu près ceux que nous retrouvons régulièrement dans les projets précédents. Donc, s'ils arrivent à trouver nos bugs, ils vont probablement trouver les bugs qu'ils vont faire par la suite.

Au sein de chaque équipe, les étudiants se répartissent les fonctions à tester. Ils commencent à se rendre compte de l'intérêt d'utiliser un outil de gestion de version. Comme ils sont encore en phase de rodage avec Git, il leur arrive également de perdre

du temps à cause de Git. C'est pourquoi la charge de travail est relativement faible à ce moment-là.

Étape b. Une fois que tout cela est mis en place, nous enlevons notre bibliothèque compilée et c'est aux équipes étudiantes d'implémenter leur propre version de la bibliothèque. L'API est fixée, mais c'est à eux de choisir leurs structures de données (type opaque) et les algorithmes de chaque fonction. Assez rapidement, ils perçoivent l'intérêt des tests unitaires développés précédemment. C'est aussi l'occasion de mettre en application les techniques de debugging présentées en parallèle.

Étape c. Lorsque cette première version (V1) est complètement implémentée, nous faisons évoluer le cahier des charges en considérant une variante un peu plus générale du jeu à implémenter. Cette évolution du cahier des charges est annoncée dès le début de l'année : on l'appelle *le caprice du client* ! L'objectif de cette évolution est de leur faire sentir *concrètement* l'importance d'avoir du code propre et maintenable.

Étape d. Une fois arrivé là, il est demandé à chaque équipe de *livrer* une version propre de leur projet. On rentre alors dans une phase de relecture de code où chaque étudiant doit récupérer le code d'une autre équipe et le lire, l'évaluer, le tester et puis vérifier l'interopérabilité entre son code et celui de l'autre équipe en faisant travailler sa bibliothèque avec l'interface texte et les tests de l'autre équipe, et inversement. C'est la première fois où ils sont confrontés au code d'autres étudiants. Ils se disent : « *J'avais fait des choix qui me paraissaient évidents, mais finalement il y a d'autres choix possibles, tout aussi pertinents* ». Ils sont également confrontés à du code soit meilleur, soit moins bon que le leur. Dans les deux cas, c'est très formateur. Cette relecture croisée est ensuite évaluée par l'équipe enseignante, qui note d'une part le code livré et d'autre part les relectures croisées. C'est un travail assez chronophage pour l'équipe enseignante, mais que l'on estime « payante » pour nos étudiants.

Étape e. Cette étape débute généralement un peu après le début du second semestre. Nos étudiants sont maintenant à l'aise avec la programmation C, et les outils de base du développement logiciel (`CMake`, `Git`, debugging...) intégrés à leur IDE. On commence alors à aborder des étapes du projet un peu plus sophistiquées. Comme certaines équipes commencent à rencontrer des difficultés avec des bugs dans leur bibliothèque, nous leur fournissons une version corrigée de la bibliothèque afin que chacun puisse repartir sur des bases saines. C'est également l'occasion pour les étudiants de relire le code des enseignants, et de comprendre ce que nous appelons un code propre. Le premier jalon de cette étape consiste à implémenter un *solveur* pour ce jeu. Nous avons à cet endroit deux niveaux d'exigences. Un niveau standard pour les étudiants moins à l'aise où il suffit d'implémenter un algorithme de type « brute force ». Puis un deuxième niveau avec un algorithme plus sophistiqué de type *branch and cut* pour aller plus loin. Pour les équipes les plus fortes, capables de produire du code optimisé, nous présentons sur une page web un classement automatique des performances des différents solveurs. Ce challenge permet de stimuler les meilleurs

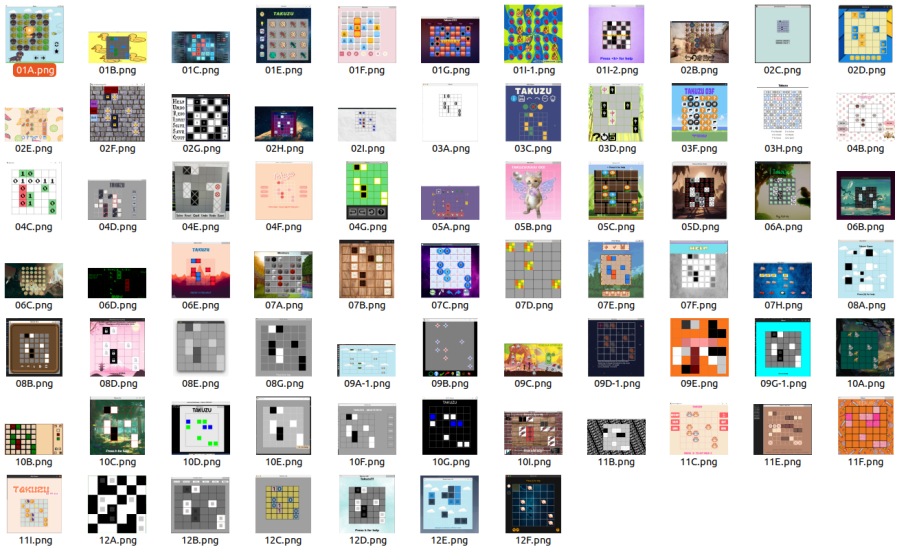

FIGURE 3. Ensemble des interfaces graphiques produites cette année par les équipes étudiantes pour le jeu Takuzu.

étudiants tout en laissant aux autres le temps de consolider leurs apprentissages plus essentiels. Il faut noter que pour debugger le solveur, les outils de debugging présentés ne sont plus simplement utiles, ils deviennent indispensables.

Jusqu'à présent, les sujets étaient très guidés, mais pour le jalon suivant, nous leur offrons plus de liberté, nous leur demandons également plus d'autonomie. En effet, dans cette partie, ils doivent implémenter une interface graphique pour leur jeu en utilisant la bibliothèque graphique SDL et pour cela, ils doivent apprendre par eux-mêmes à utiliser cette bibliothèque. En ce qui concerne l'interface graphique, ils ont toute latitude pour faire ce qu'ils veulent, du moment que le jeu soit jouable et convivial (voir figure 3).

Enfin, pour conclure, selon les années, nous ajoutons une partie de portage Android ou Web (en se basant sur WASM). C'est l'occasion de leur faire découvrir d'autres technologies plus modernes. L'autre intérêt de cette partie vient du fait qu'ils peuvent montrer à leur famille ou à leurs amis ce qu'ils ont réalisé pendant l'année. Ils sont souvent très fiers de ce qu'ils ont accompli et ils ont raison de l'être.

Déroulement sur l'année

Le scénario présenté ci-dessus s'articule autour d'une quinzaine de TD qui durent

de une à deux semaines environ. À chaque fois, des évaluations sont réalisées, mélangeant des évaluations individuelles et en équipe (voir figure 4). La plupart des évaluations individuelles concernent les exercices préliminaires des feuilles de TD, qui présentent les méthodes et outils nécessaires à la réalisation du nouveau jalon. Ce mélange permet d'éviter que certains étudiants ne se laissent porter par l'équipe et ne travaillent pas suffisamment les acquis essentiels au travail du projet en équipe.

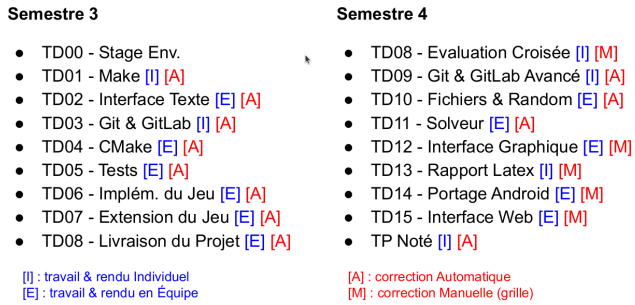

FIGURE 4. Découpage des jalons sur l'année.

Au total, cela représente plus d'une trentaine d'évaluations réparties sur l'année, et donc de notes avec des coefficients variant de 1 à 5. Cela demande un travail important et régulier de nos étudiants tout au long de l'année. Pour réussir à maintenir ce rythme de travail soutenu, nous avons mis en place de nombreuses *évaluations automatiques*, complétées par quelques évaluations manuelles (relecture de code, rapports, SDL, Web, Android).

L'année se conclut par une évaluation finale, qui prend la forme d'un TP individuel noté (coefficient 15), où les étudiants doivent réaliser en temps limité de nombreuses tâches sur machine, validant l'ensemble des acquis de l'année. Notons qu'un TP noté d'entraînement est mis à disposition pour que les étudiants s'entraînent et préparent leur évaluation finale.

En marge de ce dispositif pédagogique, nous accompagnons nos étudiants tout au long de l'année avec des séances supplémentaires de soutien, la mise à disposition d'une FAQ sur les erreurs classiques (que nous complétons chaque année), ou encore l'animation d'un forum de discussion sur `RocketChat`. Ce forum permet aux enseignants de répondre aux questions des étudiants, en précisant certains points de détail sur le sujet du projet ou encore aux étudiants de s'entraider (sans divulguer de solution évidemment).

Mise en œuvre et aspects techniques

Organisation d'un TD avec Moodle

Chaque TD s'organise de la même manière sur Moodle. Pour illustrer notre propos, nous allons examiner plus en détail l'exemple du TD 5. L'objectif pédagogique de ce TD est de savoir implémenter des tests unitaires (voir figure 5).

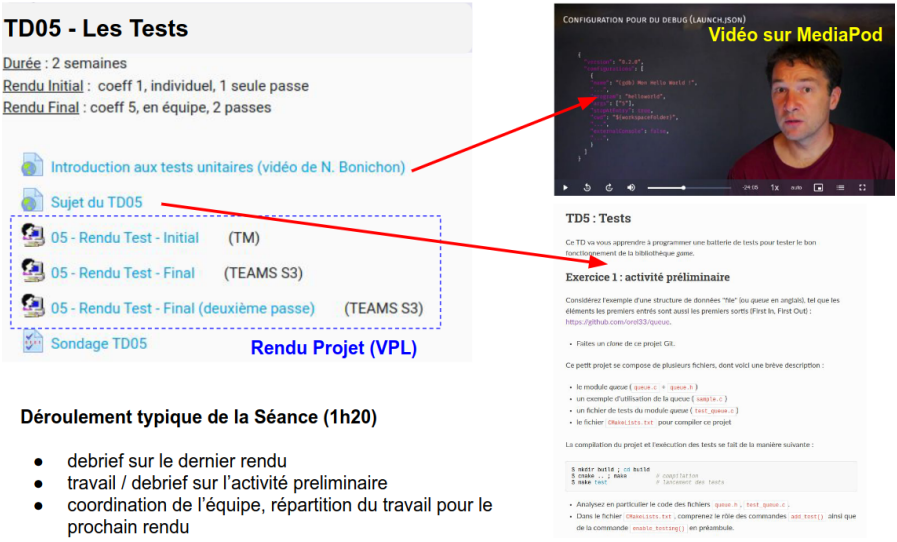

FIGURE 5. Supports pour un TD type sur Moodle.

Au début de la page, on trouve des informations de base sur la durée du TD (ici deux semaines), ainsi que des informations sur les rendus à effectuer avec leurs coefficients. Vous noterez que le premier rendu (le rendu initial) est individuel, tandis que le rendu final s'effectue en équipe. Nous reviendrons plus tard sur la notion de « passes ». On trouve ensuite un certain nombre de ressources préliminaires, telles que des vidéos ou des supports de cours (nous n'avons que 6 séances de cours sur l'année). Ces ressources doivent être consultées « à la maison » avant la séance de TD. Ensuite vient la feuille de TD à proprement parler (ou plutôt un lien vers une page web qui contient le sujet du TD). Dans cette feuille de TD, les étudiants trouvent le plus souvent une activité préliminaire présentant les méthodes et les outils à utiliser, puis une activité principale consistant à mettre en œuvre ces nouveaux acquis pour le projet. Immédiatement après, on trouve des activités Moodle pour effectuer les rendus demandés dans le sujet, qui feront l'objet d'une ou plusieurs évaluations. Enfin, nous effectuons un petit sondage pour recueillir auprès des étudiants des retours sur le TD : combien de temps cela vous a-t-il pris ?trouvez-vous que c'était

difficile ou facile ? Ces retours sont très précieux, car ils nous permettent d'améliorer nos sujets et d'adapter la charge de travail (pour les prochaines séances ou pour l'année suivante).

Pour réaliser cette feuille de TD, nous disposons de deux séances de 1h20 en présentiel (une par semaine). C'est un temps très court finalement qu'il faut optimiser. Pour l'enseignant, il s'agit essentiellement d'un travail de coordination, de synchronisation et de motivation des équipes. Notons qu'un chargé de TD gère en moyenne 5 ou 6 équipes de 3 étudiants. Il commence généralement sa séance par faire un *debriefing* sur le dernier rendu. En général, les étudiants ont des doléances à nous faire remonter, dont nous discutons. Quand il s'agit de l'activité préliminaire, nous donnons le plus souvent une correction. Puis, nous lançons l'activité suivante, typiquement celle concernant le projet. Il faut alors expliquer le sujet, coordonner les équipes et les encourager à se répartir le travail pour les semaines à venir.

Stratégie d'évaluation (automatique)

Comme expliqué auparavant, chaque TD se découpe typiquement en deux activités : une activité préliminaire avec un rendu initial (plutôt individuel) et une activité principale sur le projet avec un rendu final (en équipe). Concernant le rendu de l'activité principale, nous utilisons généralement une évaluation en *deux passes* selon les principes suivants :

— une première passe « à l'aveugle » (ou presque) avec des retours partiels à volonté avant la deadline, puis un retour complet qui sera délivré aux étudiants (avec une note) après la deadline ;

— une deuxième passe avec cette fois-ci un retour complet (à volonté) jusqu'à la seconde deadline, généralement une semaine après la première.

La plupart de ces rendus sont évalués automatiquement sur Moodle grâce au plugin VPL (cf. la section suivante), ce qui nous permet de maintenir un rythme soutenu d'évaluation tout au long de l'année. L'avantage principal de ces évaluations automatiques, c'est que cela nous permet de fournir plus de retours aux étudiants, et plus rapidement. Cela garantit également une évaluation très équitable. En revanche, ces évaluations peuvent sembler assez rigides pour nos étudiants. Cette rigidité est à la fois un avantage et un inconvénient. L'avantage, c'est que cela enseigne une certaine rigueur à nos étudiants. L'inconvénient, c'est que certains étudiants pourraient se retrouver avec un zéro parce qu'ils n'ont pas respecté les consignes, alors qu'ils ont fourni un travail qui ne mérite pas une telle note, ce qui est très frustrant. Pour remédier à cela, nous avons mis en place un système d'évaluation avec des retours à deux niveaux, plus ou moins détaillés. Le premier niveau de retour est accessible à volonté dès la première passe. À chaque fois qu'un étudiant dépose ou modifie un rendu, il reçoit un retour partiel sur son rendu, qui lui indique si ce qu'il a remis est bien ce qui était attendu. (Exemples : le fichier porte-t-il le bon nom ? est-ce qu'il compile correctement ? est-ce que cela répond aux premières questions ?). Ainsi, l'étudiant qui

se serait fourvoyé peut rectifier le tir avant la date limite de rendu. Généralement, un étudiant (ou une équipe) qui n'a aucun avertissement à ce stade a la garantie d'avoir au minimum entre 30 % et 50 % des points selon les rendus. À l'inverse, si son rendu n'est pas conforme (par exemple, s'il ne compile pas), alors il n'a pas d'excuse et nous ne négocions pas, ce qui économise pas mal d'énergie à l'équipe pédagogique ! Une fois la date limite passée, l'évaluation complète de tous les rendus est lancée par l'enseignant. À l'issue de cette première passe, les étudiants ont une note ainsi qu'un retour complet sur leur rendu. Le fait de ne pas rendre accessible à volonté les retours complets est là pour trois raisons. Il permet de limiter la triche. En effet, si les étudiants avaient accès à l'évaluation complète, ils pourraient se contenter de copier le code d'un autre étudiant et de le soumettre jusqu'à ce que le système leur donne un retour positif. Ensuite, les étudiants pourraient optimiser leur note sans comprendre le fond de l'exercice. Enfin, cela incite également les étudiants à vérifier leurs rendus avant de les soumettre.

Une fois la première passe terminée, on va débloquer une seconde passe qui va se dérouler en général à cheval sur la prochaine séance. Dans cette seconde passe, les étudiants ont accès à une évaluation complète de leur travail à chaque modification. Soient N_1 et N_2 les notes respectives de la première et de la seconde passe. La note finale (N_f) de cette activité sera calculée comme le maximum entre la première passe et la moyenne des deux passes : $N_f = max(N_1, avg(N_1, N_2))$. Avec cette formule, on privilégie quand même la première passe, mais on en encourage les étudiants à se rattraper ou à se perfectionner. En effet, la seconde passe permet aux étudiants en retard (ou en difficulté) de continuer à améliorer leurs rendus même après la date limite de la première passe. Ceci est d'autant plus important que, dans le cadre de cette UE, les compétences s'empilent. Cette stratégie d'évaluation « non punitive » est très motivante pour les étudiants qui peuvent continuer à améliorer leur code (et leur note), même après la fin de la première passe. De notre point de vue, cela les encourage à travailler encore davantage !

Mise en œuvre dans Moodle avec VPL

Rentrons un peu plus dans la partie technique. Pour implémenter cette stratégie d'évaluation, nous avons utilisé l'activité *Virtual Programming Lab*[3] (VPL) dans Moodle (voir figure 6). VPL est un plugin de Moodle qui offre un environnement pour l'enseignement et l'évaluation de la programmation, accessible depuis son navigateur Web. Il permet aux étudiants d'écrire du code dans différents langages de programmation (dans un éditeur de texte intégré à VPL), de le compiler et de l'exécuter directement dans Moodle (via le bouton *Run*). VPL facilite également l'évaluation automatisée des exercices de programmation (via le bouton *Eval*), fournissant un retour immédiat aux étudiants.

3. https://vpl.dis.ulpgc.es.

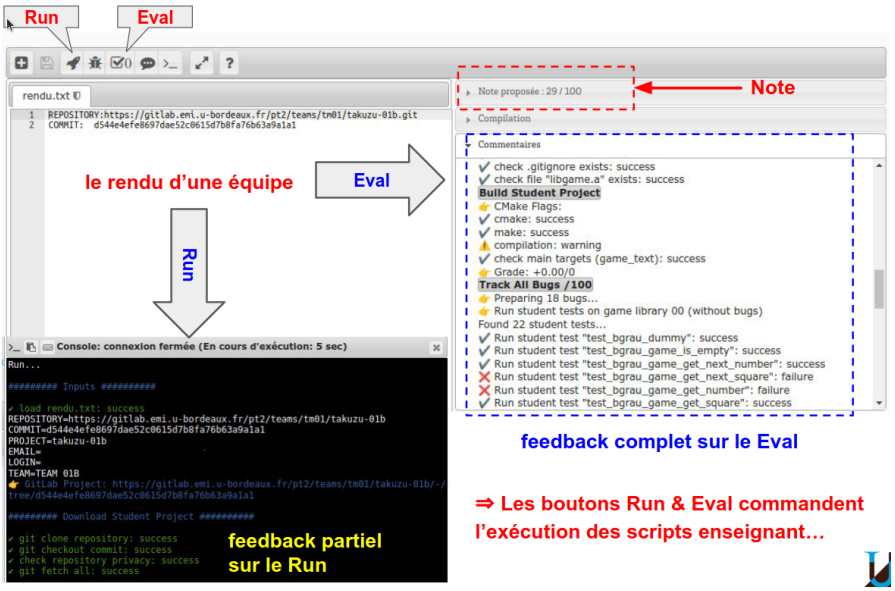

FIGURE 6. Utilisation du plugin VPL dans Moodle.

Dans le cadre de cette UE, nous avons légèrement modifié l'utilisation de VPL, car nous ne souhaitons pas que les étudiants écrivent leur code dans ce plugin. En pratique, les étudiants utilisent leur IDE (*VS Code*) et déposent leur code dans un dépôt Git. Ainsi, ils ne doivent rendre dans VPL qu'un simple fichier texte de deux lignes avec l'adresse de leur dépôt Git et le *hash* du commit qu'ils souhaitent rendre (voir figure 6, en haut à gauche). De plus, dans notre UE, la sémantique du bouton *Run* n'est pas tout à fait la même. En effet, ce bouton nous sert à lancer une *évaluation partielle* du code des étudiants, tandis que le bouton *Eval* sert à lancer une évaluation complète. Dans la figure 6, on peut voir en bas à gauche le début du retour partiel fourni aux étudiants et sur la droite, on peut voir le retour complet (accompagné de la note) fourni aux étudiants lors de l'évaluation complète.

Notons que VPL est assez flexible et permet aux enseignants de mettre en place une configuration fine du rôle des boutons *Run* et *Eval*, conforme à nos exigences pédagogiques en deux passes. En effet, si nous laissons les étudiants utiliser librement le bouton *Run* lors des deux passes, il n'en est pas de même pour le bouton *Eval*, qui est désactivé pendant la première passe, puis débloqué pour la seconde.

Infrastructure technique pour VPL

Concrètement, que se passe-t-il lorsqu'un étudiant clique sur le bouton *Eval* ? Dans notre cas, le script VPL commence par récupérer le code de l'étudiant depuis un dépôt Git extérieur (à partir du *commit* renseigné dans l'éditeur de texte VPL). Comme les dépôts sont privés, les étudiants doivent inviter un utilisateur « enseignant » qui pourra avoir accès au dépôt. Une fois le code téléchargé, le script shell de l'enseignant est exécuté par le serveur VPL dans une *jail*. Ce script teste le code de l'étudiant et produit en sortie une note (ou *grade*) et des commentaires (ou *feedback*) qui seront affichés aux étudiants dans Moodle. La figure 7 donne une vue d'ensemble du framework que nous avons mis en place. Notons que pour cette UE nous avons écrit environ 5000 lignes de script shell, ce qui est assez fastidieux, reconnaissons-le. Heureusement, la plupart de ce code est réutilisable d'une année sur l'autre quand nous changeons de jeu.

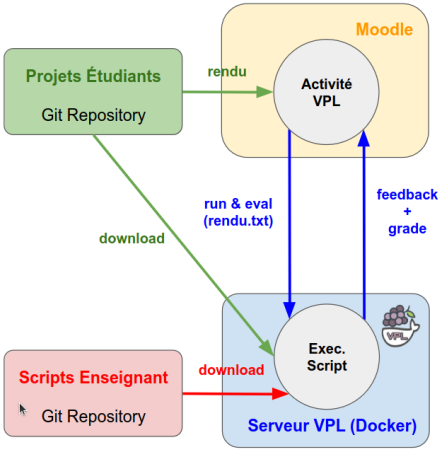

FIGURE 7. Vue d'ensemble de notre infrastructure technique pour VPL.

Par rapport à ce scénario standard, nous avons apporté quelques ajustements. Tout d'abord, nos scripts d'évaluation ne sont pas codés et hébergés dans la plateforme Moodle, mais dans un dépôt Git à part. Cela permet de les versionner et de faire un peu d'intégration continue sur nos scripts. Enfin, par rapport à l'infrastructure classique, nous avons mis en place un système de Docker [4] pour contrôler précisément l'environnement d'exécution et garantir qu'il soit vraiment similaire aux conditions de travail en TD. Tous ces ajustements, ainsi que le développement d'une boîte à outils [5], nous ont permis d'avoir assez de flexibilité pour mettre en place de nombreuses

4. `https://github.com/GuillaumeBlin/vplbdx`, développé par Guillaume Blin.
5. `https://github.com/orel33/vpltoolkit` développé par Aurélien Esnard.

évaluations en plusieurs passes, conformément à notre stratégie d'évaluation.

Écrire et publier son cours avec GitLab CI

Un dernier aspect technique que nous souhaitons aborder concerne l'écriture et la publication des cours avec GitLab. C'est un sujet qui concerne principalement les enseignants, car nous formons une équipe pédagogique composée d'environ dix membres, dont certains ne sont pas permanents. Aussi, leur degré d'implication varie en conséquence. Afin de mieux gérer les ressources pédagogiques produites par les enseignants, nous utilisons des dépôts Git. Cette approche collaborative nous permet de travailler ensemble sur l'intégralité des supports, et plus précisément sur les sujets de TD. Afin de faciliter cette démarche, nous avons décidé d'externaliser au maximum les ressources en dehors de Moodle. Nous évitons ainsi les fastidieuses manipulations telles que les glisser-coller de documents PDF vers Moodle. Pour ce faire, nous rédigeons nos supports en LaTeX ou Markdown, puis nous utilisons la chaîne d'intégration continue du framework GitLab pour compiler automatiquement ces ressources avec les bons outils. Par exemple, un sujet rédigé en Markdown sera publié automatiquement sur des *GitLab Pages* (compilé avec `mkdocs`) et la page Moodle sera mise à jour d'elle-même grâce à un simple lien vers ce site web, qui externalise et rend publique toutes les ressources du cours. Cela facilite grandement la correction d'une simple coquille ou d'une faute de frappe.

Les avantages de cette approche sont doubles. Toute l'équipe pédagogique s'approprie et contribue activement aux supports de cours. De plus, au lieu de restreindre l'accès aux cours avec des authentifications (intrinsèques à Moodle), nous pouvons rendre les ressources accessibles en dehors de Moodle, ce qui nous permet de partager notre travail avec le reste de la communauté universitaire [6].

Conclusion

Nous trouvons cette approche très motivante pour les étudiants, et nous avons de très bons retours [7]. C'est une UE qui demande beaucoup de travail et nous l'annonçons dès le début aux étudiants : « *Attention, c'est difficile, il va falloir travailler !* ». Ils le comprennent bien. C'est également une UE qui demande beaucoup de travail pour les enseignants et notamment pour le responsable qui coordonne tout cela. Mais c'est également une UE qui apporte beaucoup de satisfaction.

À la fin de l'année, les étudiants sont souvent très fiers du travail accompli, et de l'ensemble des compétences acquises. Pour l'équipe enseignante, c'est également agréable d'avoir ces retours très positifs des étudiants, et cela nous conforte dans le choix des modalités pédagogiques que nous avons présentées dans cet article. En

6. `https://pt2.gitlabpages.inria.fr/support/site`.

7. Taux de satisfaction de 78 % sur ces quatre dernières années d'après nos propres enquêtes (anonymes) auprès des étudiants.

effet, nous avons dépensé beaucoup d'énergie pour mettre en place ces systèmes d'évaluation automatique qui permettent de gérer de grosses cohortes. Un effort a également été fait sur les supports pour gérer une grande équipe pédagogique. Notons que ces efforts, étalés sur plusieurs années, ont été rentabilisés assez rapidement du fait du nombre important d'étudiants à gérer et d'enseignants à coordonner.

Malgré tout, il y a des difficultés intrinsèques. L'écriture de ces scripts et leur maintenance restent des activités fastidieuses et chronophages. Les retours des scripts ne sont pas toujours suffisamment clairs pour les étudiants. De plus, il est difficile d'être exhaustif en anticipant tous les cas possibles. Par ailleurs, il y a toujours des ambiguïtés, des flous qui restent et qui doivent être levés en TD par les enseignants. La gestion des équipes et la configuration de Moodle restent également une charge de travail importante qu'il ne faut pas sous-estimer.

La mise en place au sein de l'équipe pédagogique de toute cette chaîne d'intégration continue facilite grandement la mise à jour collaborative des supports de TD. Cette fluidité est un ingrédient important qui nous encourage à toujours améliorer nos supports.

Pour finir, il faut noter qu'il y a aussi des bénéfices cachés au système de correction automatique. D'abord, il est plus satisfaisant pour l'enseignant de programmer des scripts (même en *bash*) plutôt que de lire n fois le même code avec les mêmes erreurs. Il y a également un changement de posture intéressant de l'enseignant vis-à-vis de l'étudiant. En effet, il n'est plus celui qui pointe du doigt les erreurs. Il se trouve du côté des « gentils » qui aident les étudiants à affronter ce « méchant » script pour que les étudiants puissent finalement avoir de bonnes notes. Notons toutefois, que ces scripts aussi sophistiqués soient-ils, ne sont qu'une aide à la notation, et qu'il est toujours possible à l'enseignant « humain » d'amender manuellement une note si cela se justifie, ce que nous faisons régulièrement pour prendre en compte les situations particulières des différentes équipes.

Remerciements

Merci à tous ceux qui sont intervenus dans cette UE et qui ont participé à son amélioration au cours de ces 10 dernières années, en particulier Guillaume Blin, Pierre-André Wacrenier, Paul Dorbec, Abdou Guermouche et Vincent Penelle. Merci également à tous les enseignants qui ont pris courageusement l'aventure en chemin. Merci également à l'équipe technique du CREMI, et plus particulièrement Christophe Delmon, pour leur soutien permanent.