

CST –Analyse multi-facettes et opérationnelle pour la transformation des systemes d’information

HOUEKPETODJI Mahugnon Honoré

1 Description du sujet de la thèse

CIM est une SAS au capital social de 200k détenue à 100 par DL Software. CIM est éditeur, intégrateur, hébergeur et infogéreur de solutions pour l'assurance de personnes en santé, prévoyance. Elle offre une expertise Santé et Prévoyance acquise après plus de 30 ans auprès de ses clients. CIM est hébergeur de ses solutions pour 90 de ses clients et plus de 1000 utilisateurs. Toutes les thématiques d'infrastructure et de surveillance des flux sont intégrées à cette offre. CIM est propriétaire de ses infrastructures serveurs, tous les éléments actifs des systèmes et tous les éléments de stockage sont achetés par CIM, gérés et supervisés par les équipes de CIM. Aucun sous-traitant n'intervient dans les opérations quotidiennes d'hébergement, d'exploitation des solutions et des données hébergées.

CIM est certifiée Microsoft GOLD Partner. Elle est l'éditeur des progiciels de la gamme Izy Links et assure l'intégration de l'ensemble des briques de cette gamme ainsi que des briques partenaires nécessaires à la bonne réussite du projet. Cette solution est développée en PowerBuilder sur base de données DB2. L'équipe de développement vient d'upgrader en PowerBuilder 2017 (été 2018) et est en cours de passage sur DB2 v11 (avec l'aide d'un DBA IBM – prestation 2018).

Le système de gestion est centré sur le back office Izy Protect, autour duquel gravite l'ensemble des briques complémentaires répondant à l'ensemble des besoins, et pouvant être activées ou non. La société CIM a effectué une analyse de risque pour son évolution et croissance en 2017 d'où il ressort que Izy Protect souffre des problèmes (1) vieux langage, (2) logiciel vieillissant, (3) perte savoir, (4) changements à haut risque. Ces problèmes sont récurrents chez les organismes gérant des systèmes d'information [8].

Ce travail de doctorat consiste de proposer des modèles et des mécanismes permettant d'assurer une ré-ingénierie des systèmes d'information. Les expériences et validation des prototypes se feront dans le contexte de l'application du système d'information écrit en PowerBuilder de la société CIM

2 Etat de l'art

Cette section présente Izy Protect, ainsi que les mécanismes de ré-ingénierie des systèmes d'information patrimoniaux.

2.1 Présentation de Izy Protect

Izy Protect est un système de plus de 3 MLOC écrit en Powerbuilder et maintenu depuis plus de 20 ans par les développeurs la CIM. Le code source est organisé par bibliothèques Powerbuilder. Izy Protect compte 117 bibliothèques. La plus large à une taille supérieure à 300 KLOC. Durant toutes ces années, il y a eu beaucoup de changement dans l'équipe de développeurs. Vu la complexité actuelle du système, les développeurs ont de plus en plus du mal à le maintenir. Les anciennes versions du système sont stockées sur un disque dur. Pour des raisons internes à la CIM, les versions de Izy protect ont été perdues jusqu'en 2010. De plus les développeurs risquent à tout moment d'écraser leurs travaux. Les développeurs modifient Izy protect en réponse à ce qui décrit le travail à faire. De plus, il n'y a pas de tests unitaires automatisés. Ce qui augmente la crainte des développeurs au petit changement. Quand un système à plusieurs décennies de vie, la rétro-ingénierie est une activité centrale pour le maintenir [8].

Dans l'entreprise, les tickets sont stockés dans la base de données des tickets ou fiches navettes depuis 1998. Un ticket représente un travail unitaire. La base de données des tickets pilote l'ensemble du processus d'évolution du logiciel : attribution du travail aux développeurs, gestion du flux de travail pour répondre à une demande du client, informations de facturation sur chaque tâche. Il existe des tickets pour la correction de défauts, la rédaction de documentation, l'ajout de nouvelles fonctionnalités, etc.

Un ticket comporte entre autres les caractéristiques suivantes :

- la date de création
- la date de clôture
- l'estimation du temps nécessaire au développeur pour travailler sur le ticket
- temps passé par un développeur
 - le temps d'analyser
 - le temps de mettre en œuvre une solution
 - le temps de test
- le(s) bibliothèque(s) impactées

2.2 Analyse de l'évolution de l'état d'un logiciel patrimonial

Les systèmes patrimoniaux sont des systèmes en constante changements : production de nouvelles fonctionnalités. La deuxième loi de Lehman [13] stipule qu'à mesure que les logiciels évoluent, la complexité croissante et l'augmentation des défauts entraîneront une baisse de la satisfaction des parties prenantes, à moins que les équipes de projet n'entreprennent le travail nécessaire pour maintenir la qualité. Dans ce sens nombreuses travaux de la littérature propose des techniques, pour suivre l'évolution de l'état de ces systèmes.

[?] utilise les données des occurrences bugs et le temps pour modéliser l'évolution d'un système logiciel avec ccharts.

[15] propose un système de recommandation d'action au développeur pour un nouveau bug. Le system se base sur l'historique des bugs, le code source ainsi que qu'un algorithme de prédiction. Pour que ça marche, le code source doit être gérer dans un système de contrôle de version. Ce qui n'est pas le cas avec Izy Protect.

[18] utilise les modèles et l'analyse de l'historique des défauts pour évaluer la qualité d'un système et prédire l'effort nécessaire pour améliorer le système. Les métriques mesurés sont : le taux de bugs sur une période de temps, le ratio entre le taux de d'augmentation de la taille du système et le taux de bugs, le temps moyen entre la découverte des bugs, l'effort pour résoudre les bugs, estimation des risque de future bugs, etc. Certain de ces métriques comme : le taux de bugs par période de temps, l'effort pour résoudre les bugs sont intéressantes dans le cadre de Izy Protect. Les autres ne se conforme pas au contexte car les tickets de Izy protect ne sont directement liés au code de façon standard. Chaque développeur note le numéro de tickets a sa façon dans le code.

[4, 12] propose un modèle de prédiction des défauts avec des algorithmes d'apprentissage en utilisant l'historique des bugs de système. Les algorithmes de prédiction de bugs ne sont pas toujours consistant [2]. De plus, ils ne tiennent pas compte des changements qui peuvent être imprévisible dans le code. De plus Izy Protect est système commercial multi-utilisateurs, et donc chaque utilisateur a des fonctionnalités ou des changements qui lui est spécifique.

[16] Utilise l'historique le nombre de lignes de code changer pour un bug fix ou une nouvelle fonctionnalité pour prédire la densité de bug le code. Pour que ceci soit réalisable, il faut que le code soit préalablement versionné dans un system de contrôle de version.

[19] à étudier différents algorithme de modélisation des défauts d'un système a partir des donnée des défauts relevé sur huit projet open source. Il en ressort que la moyenne glissée modélise mieux les défauts des systèmes.

2.3 Rétro-ingénierie

Dans cette section, je presenterais les traveaux reliés à outillage pour la rétro-ingénierie que j'ai étudié. [6] définit la rétro-ingénierie comme *"un proccessus d'analyse d'un système donnée pour identifier les composants du système et leurs relations, afin de créer des représentations du système sous une autre forme ou à un niveau d'abstraction plus élevé."*

[5] affirme que la rétro-ingénierie n'est pas limitée à certains langages courants mais est universelle. Par exemple, elle peux concerner la base de données [7] ou l'interface graphique [21]. Il est donc nécessaire de disposer d'une suite d'outils polyvalents et extensibles, indépendants du langage : cette extension peut se faire à plusieurs niveaux - méta-modèle. Mais aussi au niveau des outils eux-mêmes (par exemple en agissant sur le modèle).

Kienle et Müller [11] ont explorer la question de la construction d'outils de rétro-ingénierie. Ils proposent trois axes : (1) les critères, (2) la construction et (3) l'évaluation.

Bellay et Gall [3] ont proposé une large liste de critères pour comparer les outils de rétro-ingénierie. Par contre ils ne couvrent pas le métamodèle, l'extension du métamodèle et des outils de manière exhaustive.

Govin et al [10] ont identifié des critères pour les outils de réingénierie. Parmi ces critères j'ai retenu ceux qui rapportent à la rétro-ingénierie. Ce sont les critères (1) de selection , (2) d'abstraction.

Bruneliere et al. [5] a proposé et mis en oeuvre le cadre MoDisco. MoDisco est divisé en 4 couches, le projet de modélisation des éclipses, l'infrastructure, la technologie et les cas d'utilisation. La couche infrastructure contient des outils génériques tels que celui permettant de naviguer et d'interroger les modèles. La couche technologie contient le métamodèle spécifique, (Java, XML, JSP, ...). Et la couche des cas d'utilisation contient l'action spécifique pour un modèle, par exemple

le refactoring de code Java. MoDisco inclut , navigateur de model, un editeur pour voir le code des entité d'un model, un support graphique pour faire des requetes sur les elements du model. Par contre Bruneliere et al. [5], ne propose que l'UML pour visualiser les elements d'un model. Alors que pour un système large, l'UML devient vite une toile d'araigné et fraîne une compréhension rapide du système.

3 Avancées actuelles

3.1 Route vers le DevOps

Les problèmes préalablement cités sur Izy Protect à savoir : code non versionné, ce qui engendre les pertes et écrasement de code, puis l'absence total de test unitaire automatisé, m'ont amenés à :

- Mettre en place un serveur subversion (SVN) pour versionné les code. Le choix de SVN est guidé par le fait de sa simplicité pour la compréhension. De plus le module de contrôle de version que offre Powerbuilder n'est pas trop stable. Par exemple parfois le module considère un code déjà versionné comme un code non versionné. De plus le module de contrôle de version que offre Powerbuilder ne support pas bien les opérations avancés comme la comparaison de deux versions du code ou bien la résolution de conflit. En complément de Powerbuilder, j'ai amener les développeurs de la CIM à utiliser TortoiseSVN.
- Reconstruire l'histoire du code source de Izy Protect depuis 2012 afin de procéder à des analyse de l'évolution du système a partir des changements entre les version. Les versions de Izy Protect sont nombreuses. J'en suis actuellement a fin des versions produites en 2014.
- Tester des outils de tests unitaire sur Powerbuilder. PUnit parait être la librairy qui permet de tester les fonctionnalités des applications Powerbuilder au niveau du code source. Je suis entrain de le mettre en place.

3.2 Analyse des fiches navettes

Afin d'évaluer l'état de Izy Protect, et contrôler l'effet de la ré-ingénierie sur Izy Protect, j'ai utilisé la base de donnée des fiche navettes. Après nettoyage, seul les tickets à partir de 2004 sont utilisable. Les données sont non-stationnaires. En me basant sur les résultats de [19], j'ai utilisé la moyenne glissant avant un pas de 2 mois pour modéliser l'état du système. J'ai principalement mesuré les métriques suivantes : l'evolution du temps pour ferme les tickets, l'evolution du temps nécessaire au developpeur, l'evolution du temps des tests manuel du developpeur, l'évolution de l'estimation du temps de developpement par le manager. Les resultats sont présenté dans un dashboard qui se met a jours chaque mois. La Figure 1 montre l'aperçu du dashboard. Les developpeurs ont confirmés que les resultats de l'analyse reflètent bien leurs ressenti de l'état de Izy Protect.

3.3 Outil d'aide a la rétro-ingénierie logiciel

Pour répondre aux exigences détaillées dans la section Section 2.3, j'ai développé une suite d'outils d'aide à la retro-ingénierie. Ces outils sont developpeur au dessus de la platform Moose [17]. En effet, la plate-forme offre un méta-modèle générique et quatre outils principaux pour l'analyse des systèmes logiciels. Il s'agit de (1) Famix : un meta-model qui permet au developpeur de représenter un programme, (2) Moose Query : un API pour naviguer dans un model Famix, (3) Les tags : utilisées pour enrichir le code source avec des informations qui ne peuvent pas être directement déduites du code source et (4) Roassal : Un framework de visualisation intégré dans Moose. Dans la suite, je presenterai d'abord l'achitecture mise en place pour les outils puis chaque outils.

3.3.1 Architecture des outils

La Figure 2 montre l'architecture globale de la suite d'outils. L'architecture global de la suite d'outils est principalement composé d'un *BrowserMaster* et des navigateurs. Le tout s'exécute sur une instance d'un model Famix. Il a été pensé dans le but de faciliter l'experience du developpeur dans les différentes activités de la retro-ingenierie logiciel. Le *BrowserMaster* est responsable de l'échange d'information entre les navigateurs. Il est au courant de tous les navigateurs ouvert. Il se charge de notifier tous les navigateurs ouvert en cas d'évènement.

Chaque navigateur fonction sur une entité du model Famix local a lui. Cet entité peut être un seul entité ou un group d'entité. Il navigateur est peut émettre ou recevoir un evenement.

émission d'évènement :

L'émission d'un évènement par un navigateur est gouverner par son mode de propagation. En mode de propagation actifs, le navigateur emit un evenement pour publier son entité courant a chaque fois que l'entité change. Dans le contraire,

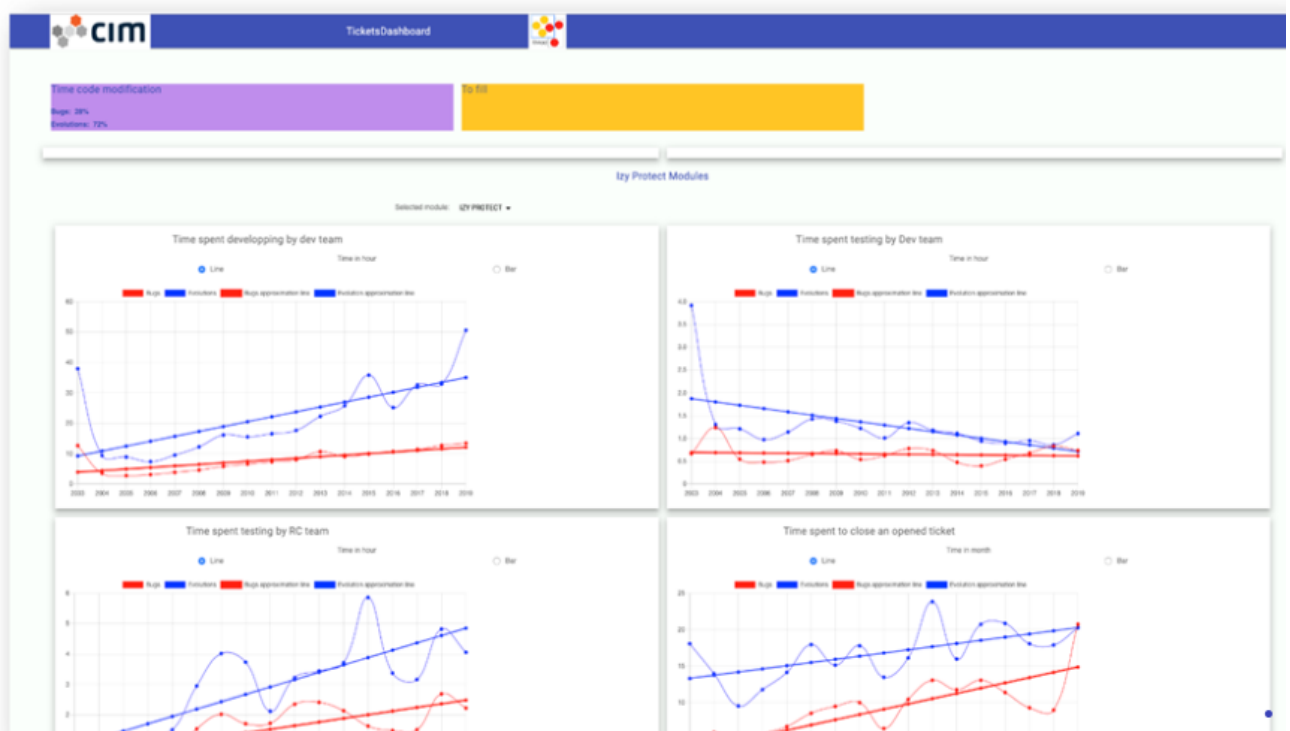


FIGURE 1 – Dashboard d’analyse de tickets

l’entité courant est gardé localement.

reception d’évènement :

Le comportement d’un navigateur a la reception d’un évènement dépend du mode de réception de ce dernier. Ainsi chaque navigateur possède trois mode de réception d’évènement. Il s’agit des modes (1) *follow* : le navigateur change remplace son entité courant par l’entité reçu via l’évènement, (2) *highlight* : le navigateur cherche l’entité reçu via l’évènement dans son entité courant, s’il le trouve, il le colorie, (3) *ignore* : le navigateur ne fait rien a la reception de l’évènement.

Tous les navigateurs presentent un bouton qui indique sont mode d’émission d’évènement, et trois boutons qui indiquent sont mode de reception d’évènement

3.3.2 Nivagateur de model

Le navigateur de model comme le montre la Figure 3 outre la partie commune a tous les navigateurs, comporte deux boutons (*All PWBObjects*, *Scoped View*), un contenu et un champ de recherche.

Le contenu navigateur presente par défaut la liste des entités que contient l’entité courant du navigateur. L’utilisateurs peut décidé de se concentrer sur un certain nombres d’entité. Dans ce cas il les selectionne et il active le bouton *Scoped View*. Pour le moment le navigateur de model est juste une liste. Mais Il est prévu de l’étendre.

Le champ de recherche du navigateur permet à l’utilisateur d’écrire une requête de recherche sur son entité Famix courante. Cette entité est pour la plus part du temps un groupe d’entité Famix. Cette requête peut être lexicale, sous forme de simple chaîne pour une recherche lexicale, ou structurée. Par exemple, si l’utilisateur recherche *include : FamixPWBAtribute*, le résultat sera toutes les entités contenant des FamixPWBAtribute (attributs Powerbuilder).

3.3.3 navigateur de graph d’appel

La Figure 4 donne un aperçu du graphe d’appel avec les options. Ce outil permet principalement de visualiser sous forme d’un graph les entités qui utilisent l’entité courant du navigateur. Les noeuds representent les entités. Les fleches representent l’ensemble des utilisations entre deux entités. Le sens de la flèche indique le sens des utilisations. En effet pour un système large comme Izy Protect par exemple, ce graph peut rapidement devenir illisible. Pour palier a ce problème, l’outil integre un panel d’option qui permet de filtrer le graph par type d’entité a l’origine des appels. Afin de donner plus

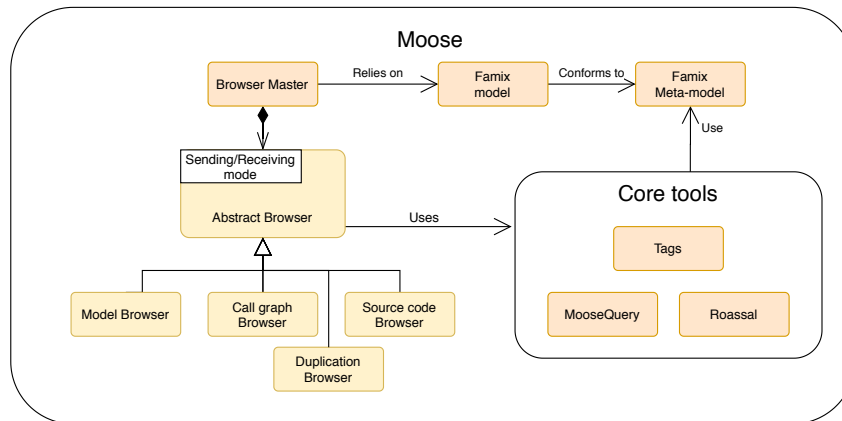


FIGURE 2 – Architecture de la suite d'outils

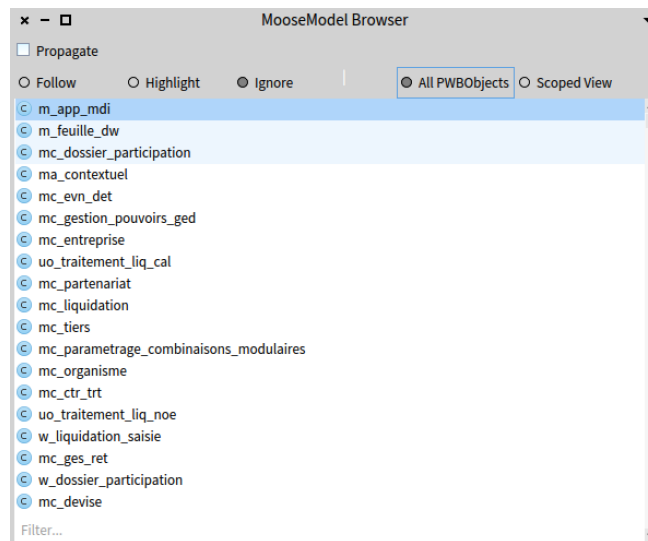


FIGURE 3 – Navigateur de model

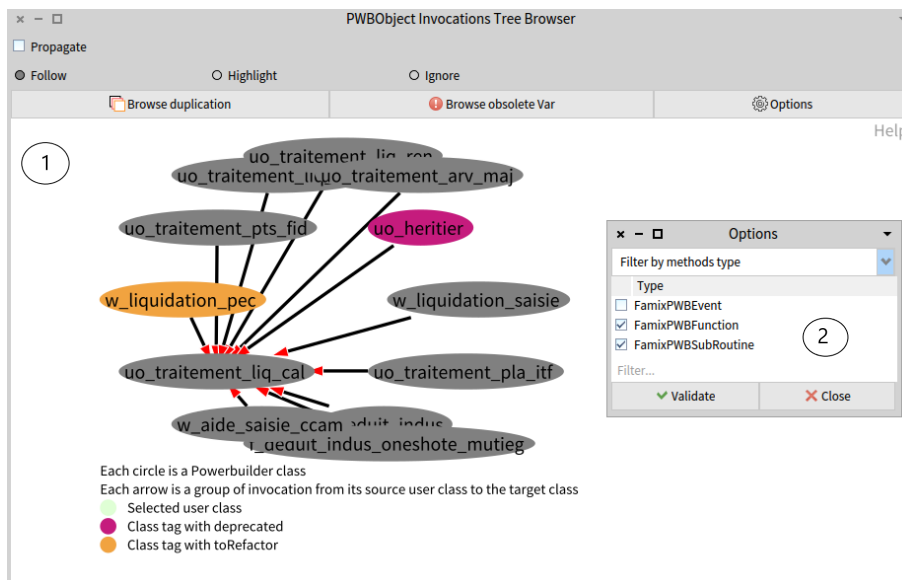


FIGURE 4 – Graphe d'appel avec l'option de reduction de complexité du graphe

de contexte au developpeur, quand il glisse la souris sur une flèche, un popup lui montre toutes les utilisations avec leurs code source.

Le navigateur de graph d'appel permet aussi au developpeur de marquer les entités afin de lui ajouter une information qu'on ne peut pas extraire directement du code source. Une connaissance qui ressort de l'expérience du developpeur sur le système.

3.3.4 navigateur de Code mort

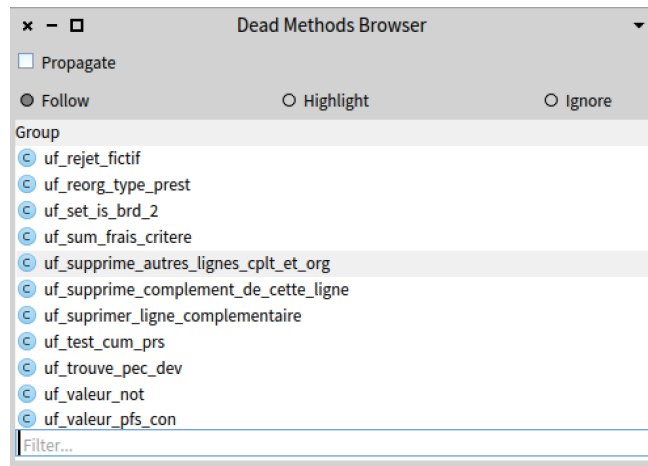


FIGURE 5 – navigateur de code mort

La Figure 5 montre le navigateur de code mort. Ce navigateur présente pour le moment les méthodes de l'entité courant du navigateur qui ne sont jamais appelées dans le système et les que ces méthodes appellent.

3.3.5 navigateur de code dupliqué

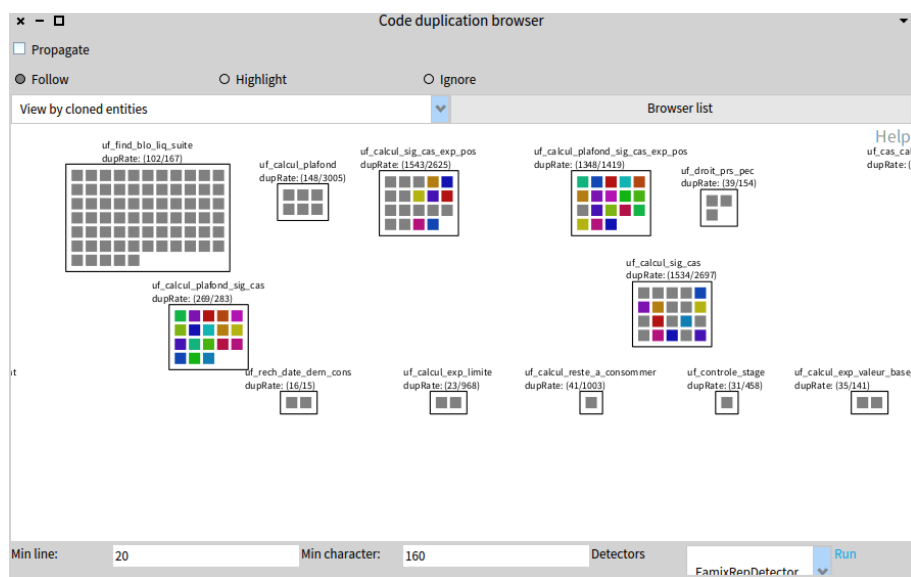


FIGURE 6 – navigateur de code dupliqué

La Figure 6 présente le navigateur de code dupliqué. Les carrés externes représentent les entités qui présentent de clone. Dans le cas de la Figure 6 c'est des méthodes. Les carrés internes représentent les clones qui présentent une entité. Les codes dupliqués sont représentés de façon de ce que l'utilisateur puisse facilement l'inspecter. Il utilise actuellement un algorithme de détection basé sur l'égalité stricte des chaînes de caractères [9]. Cet algorithme peut être remplacé par un algorithme plus sophistiqué pour détecter les doublons [20].

Supposons deux entités $e1$ et $e2$ qui présentent des clones en commun. Quand l'utilisateur clique sur $e1$, les clones de $e1$ prennent des couleurs différentes. Les clones que $e1$ a en commun avec $e2$, dans $e2$ prennent les mêmes couleurs que leurs semblables dans $e1$. Cela permet de voir plus facilement quel entités on a de code en commun et de comparer les codes sources dans le navigateur de code que je présenterai dans la suite.

3.3.6 Navigateur de code source

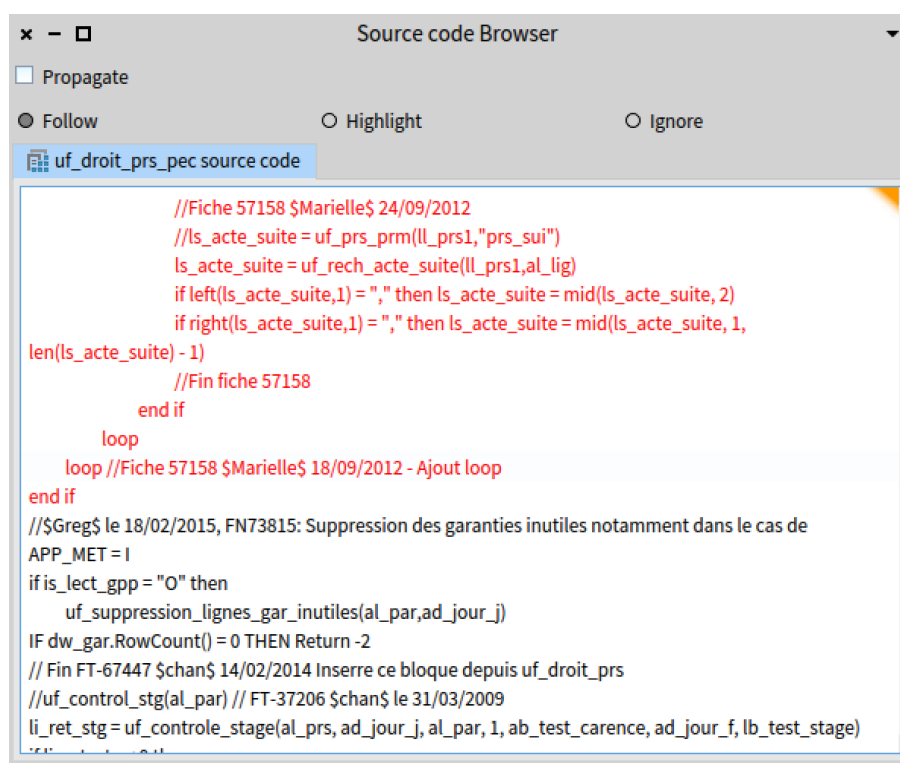


FIGURE 7 – navigateur de code dupliqué

La Figure 7 présente le navigateur de code source. Ce navigateur est une fenêtre qui affiche le code source de l'entité courante. En particulier dans le cadre d'un fragment dupliqué, le code source de l'entité qui contient ce fragment est affiché normalement sauf que la partie du code qui représente le fragment dupliqué est en rouge comme sur la Figure 7

4 Future travaux

Toutes les outils présentés ci-dessus sont conçus dans le but de nettoyer le code de visualiser l'interaction entre les différentes classe de Izy Protect. Par contre les développeurs ne l'ont pas encore utilisés dans leurs quotidien. Dans ce sens, j'identifie la validation de ces outils dans la suite mes travaux.

D'un autre côté, l'entreprise prévoit de migrer une partie du système dans une architecture orienté service. Les développeurs ont donc exprimé le besoin d'extraire des règles métier du code. Plusieurs travaux sont présentés dans la littérature dans ce sens en particulier [14] qui se base sur l'analyse statique des interactions des fonctions avec les variables pour extraire les règles métier. Je souhaite combiner cette méthode avec la méthode proposée [1] pour proposer un outil d'extraction de règles métier dans les systèmes patrimoniaux.

Je mettrai en place et j'étudierai l'impact du DevOps à la CIM.

5 Publications

Cette section présente la liste des soumissions liées à cette thèse.

1. Journée de travail du groupe Rimel du GR/GPL : HOUKPE TODJI Honoré, Nicolas Anquetil *Improving practices in a medium French company : First step*
2. 13th International Conference on the Quality of Information and Communications Technology 2020 : HOUKPE TODJI Honoré,

6 Formation doctorale

7 Projet professionnel

Références

- [1] Nicolas Anquetil, Anne Etien, Gaelle Andreo, and Stéphane Ducasse. Decomposing God Classes at Siemens. In *International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, United States, October 2019. URL <https://hal.inria.fr/hal-02395836>.
- [2] Abdul Ali Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. On the time-based conclusion stability of software defect prediction models. *arXiv preprint arXiv :1911.06348*, 2019.
- [3] Berndt Bellay and Harald Gall. An evaluation of reverse engineering tools. *Journal of Software Maintenance : Research and Practice*, 1998.
- [4] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahvas. Software defect prediction using regression via classification. In *IEEE International Conference on Computer Systems and Applications*, 2006., pages 330–336, 2006.
- [5] Hugo Bruneliere, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. Modisco : A model driven reverse engineering framework. *Information and Software Technology*, 56(8) :1012–1032, 2014.
- [6] Elliot Chikofsky and James Cross II. Reverse engineering and design recovery : A taxonomy. *IEEE Software*, 7(1) :13–17, January 1990. doi: 10.1109/52.43044. URL <http://dx.doi.org/10.1109/52.43044>.
- [7] Julien Delplanque, Anne Etien, Nicolas Anquetil, and Stéphane Ducasse. Recommendations for evolving relational databases. In *32nd International Conference on Advanced Information Systems Engineering*, 2020.
- [8] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2002. ISBN 1-55860-639-4. URL <http://rmod.inria.fr/archives/books/OORP.pdf>.
- [9] Stéphane Ducasse, Matthias Rieger, and Serge Demeyer. A language independent approach for detecting duplicated code. In Hongji Yang and Lee White, editors, *Proceedings of 15th IEEE International Conference on Software Maintenance (ICSM'99)*, pages 109–118. IEEE Computer Society, September 1999. doi: 10.1109/ICSM.1999.792593. URL <http://rmod.inria.fr/archives/papers/Duca99bCodeDuplication.pdf>.
- [10] Brice Govin. *Support à la rénovation d'une architecture logicielle patrimoniale : Un cas réel chez Thales Air Systems*. PhD thesis, Université de Lille, June 2018. URL <http://rmod.inria.fr/archives/phd/PhD-20180-GovinBrice.pdf>.
- [11] Holger M. Kienle and Hausi A. Müller. The tools perspective on software reverse engineering : Requirements, construction, and evaluation. In *Advanced in Computers*, volume 79, pages 189–290. Elsevier, 2010.
- [12] Sunghun Kim, Thomas Zimmermann, E James Whitehead Jr, and Andreas Zeller. Predicting faults from cached history. In *29th International Conference on Software Engineering (ICSE'07)*, pages 489–498. IEEE, 2007.
- [13] Manny Lehman. Laws of software evolution revisited. In *European Workshop on Software Process Technology*, pages 108–124, Berlin, 1996. Springer.
- [14] Lei Wu, H. Sahraoui, and P. Valtchev. Coping with legacy system migration complexity. In *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, pages 600–609, 2005.
- [15] Valentina Lenarduzzi, Alexandru Cristian Stan, Davide Taibi, Davide Tosi, and Gustavs Venters. A dynamical quality model to continuously monitor software maintenance. In *The European Conference on Information Systems Management*, pages 168–178. Academic Conferences International Limited, 2017.
- [16] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering*, pages 284–292, 2005.
- [17] Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gîrba. The story of Moose : an agile reengineering environment. In Michel Wermelinger and Harald Gall, editors, *Proceedings of the European Software Engineering Conference, ESEC/FSE'05*, pages 1–10, New York NY, 2005. ACM Press. ISBN 1-59593-014-0. doi: 10.1145/1095430.1081707. URL <http://rmod.inria.fr/archives/papers/Nier05cStoryOfMoose.pdf>. Invited paper.
- [18] Dan Port and Bill Taber. Actionable analytics for strategic maintenance of critical software : an industry experience report. *IEEE Software*, 35(1) : 58–63, 2017.
- [19] Uzma Raja, David P. Hale, and Joanne E. Hale. Modeling software evolution defects : a time series approach. *Journal of Software Maintenance and Evolution : Research and Practice*, 21(1) :49–71, 2009. doi: 10.1002/smr.398. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.398>.
- [20] Chanchal Kumar Roy and James R Cordy. A survey on software clone detection research. *Queen's School of Computing TR*, 541(115) :64–68, 2007.
- [21] Benoît Verhaeghe, Anne Etien, Nicolas Anquetil, Abderrahmane Seriai, Laurent Deruelle, Stéphane Ducasse, and Mustapha Derras. Gui migration using mde from gwt to angular 6 : An industrial case. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Hangzhou, China, 2019. URL <https://hal.inria.fr/hal-02019015>.