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 200 k détenu a 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'assemble 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 à 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 supérieur 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 a le maintenir. Les anciennes versions du système sont stockées sur un disque dur. Pour des raisons internes à la CIM, les versions d'Izy Protect ont été perdues jusqu'en 2010. De plus, les développeurs risquent à tout moment d'écraser leur travaille. Les développeurs modifient Izy protect en réponse a qui décrit le travaille à faire. De plus, il n'y a pas de tests unitaire automatisés. Ce qui augment la craint 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) impactée(es)

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

Les systèmes patrimoniaux sont des systèmes en constant changement : production de nouvelles fonctionnalités. La deuxième loi de Lehman [15] 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, nombreux travaux de la littérature proposent 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.

[17] propose un système de recommandation d'action au développeur pour un nouveau bug. Le système 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 géré dans un système de contrôle de version. Ce qui n'est pas le cas avec Izy Protect.

[20] 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 mesurées 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 risques de futurs bugs... Certain de ces métriques comme : le taux de bugs par période de temps, l'effort pour résoudre les bugs sont intéressants dans le cadre d'Izy Protect. Les autres ne se conforment pas au contexte, car les tickets d'Izy protect ne sont directement liés au code de façon standard. Chaque développeur note le numéro de tickets à sa façon dans le code.

[4, 14] 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 consistants [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.

[18] Utilise l'historique le nombre de lignes de code changer pour un bug fixe 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 système de contrôle de version.

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

2.3 Rétro-ingénierie

Dans cette section, je présenterais les travaux reliés à outillage pour la rétro-ingénierie que j'ai étudié. [6] définit la rétro-ingénierie comme *un processus 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 peut concerner la base de données [7] ou l'interface graphique [23]. 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 [13] on 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 méta-modèle, l'extension du méta-modèle 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 sélection (2) d'abstraction.

Bruneliere et al. [5] a proposé et mis en œuvre 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 l'infrastructure contient des outils génériques permettant de naviguer et d'interroger les modèles. La couche technologie contient le méta-modèle spécifique

(Java, XML, JSP, ...). Et la couche des cas d'utilisation contient l'action spécifique pour un modèle, par exemple le remaniement de code Java. MoDisco inclut, navigateur de modèle, un éditeur pour voir le code des entité d'un modèle, un support graphique pour faire des requêtes sur les éléments du modèle. Par contre Bruneliere et al. [5], ne propose que l'UML pour visualiser les éléments d'un modèle. Alors que pour un système large, l'UML devient vite une toile d'araignée et freine 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 le code. Le choix de SVN est guidé par le fait de sa simplicité pour la compréhension. De plus, le module de contrôle de version qu'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 qu'offre Powerbuilder ne supporte pas bien les opérations avancées comme la comparaison de deux versions du code ou bien la résolution de conflit. En complément de Powerbuilder, j'ai amené les développeurs de la CIM à utiliser TortoiseSVN.

— Reconstruire l'histoire du code source de Izy Protect depuis 2012 afin de procéder à des analyses de l'évolution du système à partir des changements entre les versions. Les versions d'Izy Protect sont nombreuses. J'en suis actuellement a fin des versions produites en 2014.

— Tester des outils de tests unitaires sur Powerbuilder. PBUnit paraît être la libraire 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 d'Izy Protect, et contrôler l'effet de la ré-ingénierie sur Izy Protect, J'ai utilisé la base de donnée des fiches navettes. Après nettoyage, seul les tickets à partir de 2004 sont utilisable. Les données sont non-stationnaires. En me basant sur les résultats de [21], 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'évolution du temps nécessaire aux développeurs, l'evolution du temps des tests manuel du developpeur, l'évolution de l'estimation du temps de développement par le manager. Les résultats sont présentés dans un dashboard qui se met à jours chaque mois. La Figure 1 montre l'aperçu du dashboard. Les développeurs ont confirmé que les résultats de l'analyse reflètent bien leur ressentit de l'état d'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 rétro-ingénierie. Ces outils sont développés au-dessus de la plateforme [19]. 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-modèle qui permet aux développeurs de représenter un programme, (2) Moose Query : un API pour naviguer dans un modèle 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 présenterai d'abord l'architecture mise en place pour les outils puis chaque outil.

3.3.1 Architecture des outils

La Figure 2 montre l'architecture globale de la suite d'outils. L'architecture globale de la suite d'outils est principalement composée d'un *BrowserMaster* et des navigateurs. Le tout s'exécute sur une instance d'un modèle Famix. Il a été pensé dans le but de facilité l'expérience du développeur dans les différentes activités de la rétro-ingénierie 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 ouverts en cas d'événement.

Chaque navigateur fonction sur une entité du modèle local a lui. Cette entité peut être une seule entité ou un groupe d'entité. Il navigateur est peut émettre ou recevoir un événement.

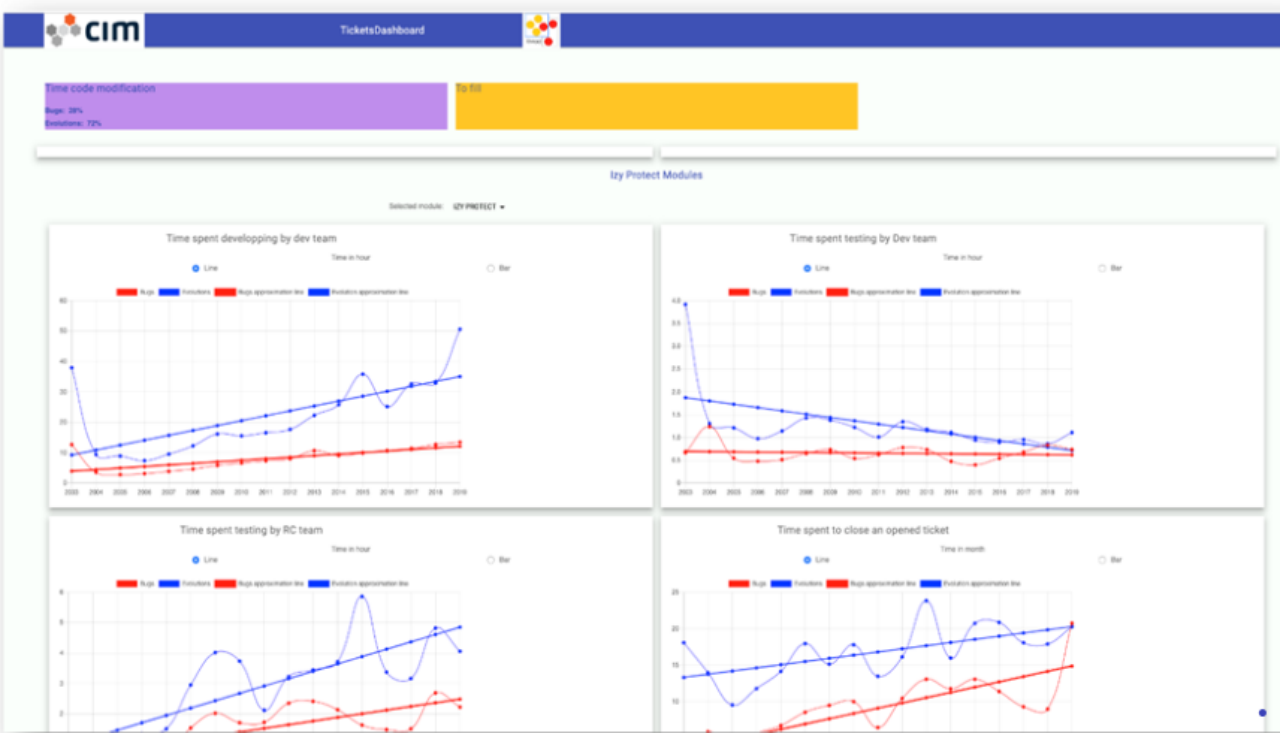

FIGURE 1 – Dashboard d'analyse de tickets

émission d'événement :

l'émission d'un événement par un navigateur est gouverner par son mode de propagation. En mode de propagation active, le navigateur émet un événement pour publier son entité courant a chaque fois que l'entité change. Dans le contraire, l'entité courant est gardée localement.

réception d'événement :

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

Tous les navigateurs présentent un bouton qui indique sont mode d'émission d'événement, et trois boutons qui indiquent sont mode de réception d'événement.

3.3.2 Nivagateur de modèle

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

Le contenu navigateur présente par défaut la liste des entités que contient l'entité courant du navigateur. L'utilisateur peut décider de se concentrer sur un certain nombre d'entités. Dans ce cas, il les sélectionne et il active le bouton *Scoped View*. Pour le moment le navigateur de modèle 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 plupart du temps un groupe d'entité Famix. Cette requête peut être lexicale, sous forme de simple chaîne pour une recherche lexicale, ou structurelle. Par exemple, si l'utilisateur recherche *include : FamixPWBAttribute*, le résultat sera toutes les entités contenant des FamixPWBAttribute (attributs Powerbuilder).

3.3.3 navigateur de graph d'appel

La Figure 4 donne un aperçu du graphe d'appel avec les options. La fenêtre (1) de ce outil permet principalement de visualiser sous forme d'un graphe les entités qui utilisent l'entité courant du navigateur. Les nœuds représentent les entités. Les flèches représentent l'ensemble des utilisations entre deux entités. Le sens de la flèche indique le sens des utilisations.

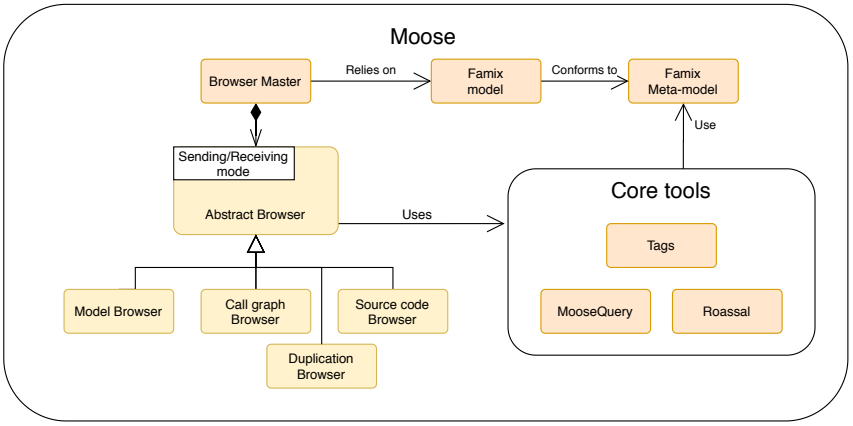

FIGURE 2 – Architecture de la suite d'outils

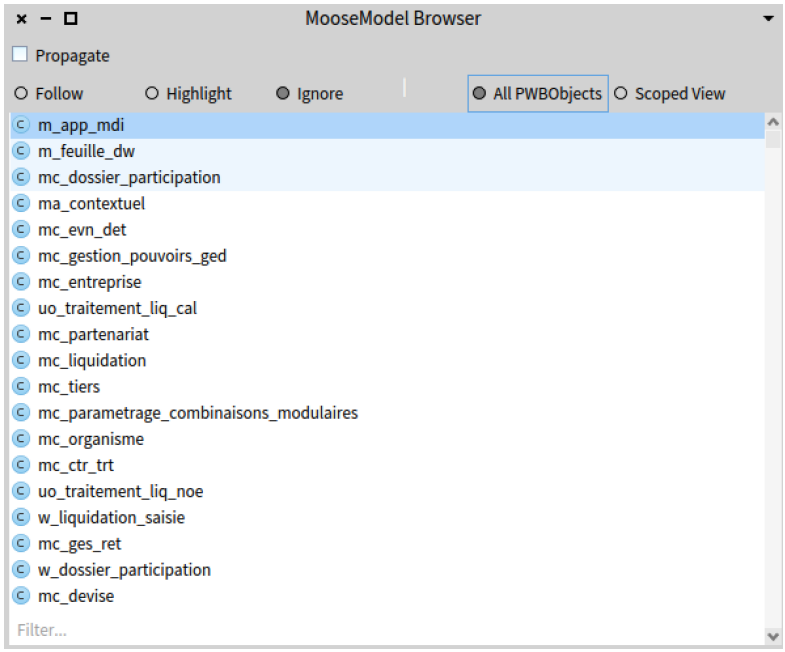

FIGURE 3 – Navigateur de modèle

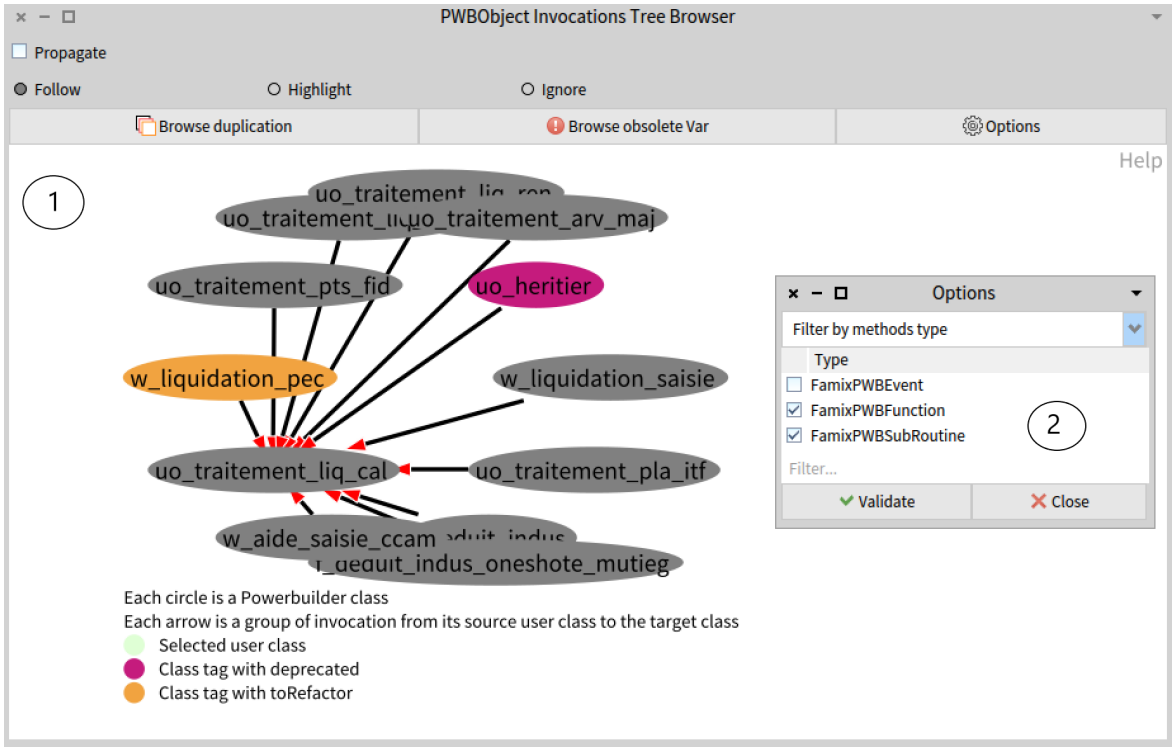

FIGURE 4 – Graphe d'appel avec l'option de reduction de complexité du graphe

En effet pour un système large comme Izy Protect par exemple, ce graphe peut rapidement devenir illisible. Pour palier à ce problème, l'outil intègre un panel d'option(la fenêtre (2) de la Figure 4) qui permet de filtrer le graphe par type d'entité a l'origine des appels. Afin de donner plus de contexte aux développeurs, quand il glisse la souris sur une flèche, un popup lui montre toutes les utilisations avec leur code source.

Le navigateur de graphe d'appel permet aussi aux développeurs 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 développeur sur le système.

3.3.4 navigateur de Code mort

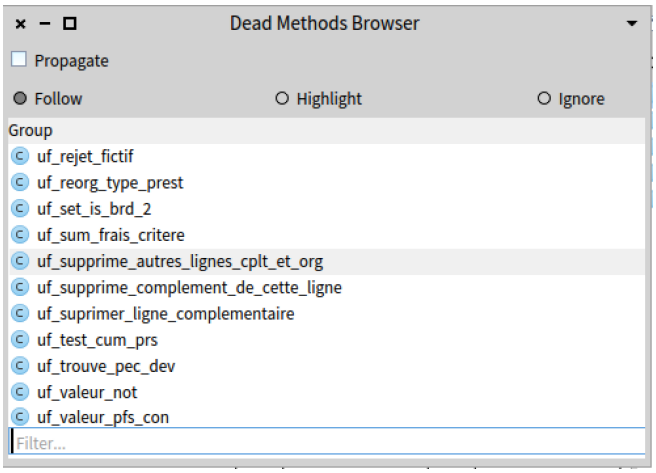

FIGURE 5 – navigateurde code mort

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

3.3.5 navigateur de code dupliqué

La Figure 6 présent 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 que présente une entité. Les clones sont représentés de façon de ce que l'utilisateur puis facilement l'inspecter. Il utilise actuellement un algorithme de détection basé sur l'égalité strict des chaînes de caractères [9]. Cet algorithme peut être remplacé par un algorithme plus sophistiqué pour détecter les doublons [22].

Supposons deux entités *e1* et *e2* qui présentent de clones en commun. Quand l'utilisateur clique sur *e1*, les clone de *e1* prennent des couleurs différentes. Les clones que *e1* à en commun avec *e2*, dans *e2* prennent les mêmes couleurs que leurs semblables dans *e1*. Cela permet de voir plus facilement quelles entités ont 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

La Figure 7 présente le navigateur de code source. Ce navigateur est une fenêtre qui affiche le code source de l'entité courant. 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 présentes ci-dessus sont conçus dans le but de nettoyer le code de visualiser l'interaction entre les différentes classe d'Izy Protect. Par contre les développeurs ne l'ont pas encore utilisé dans leur 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 logiques métier du code. Plusieurs travaux sont présentés dans la literature dans ce sens en particulier [16] qui se base sur l'analyse statique des interaction des fonctions avec les variables pour

FIGURE 6 – navigateur de code dupliqué

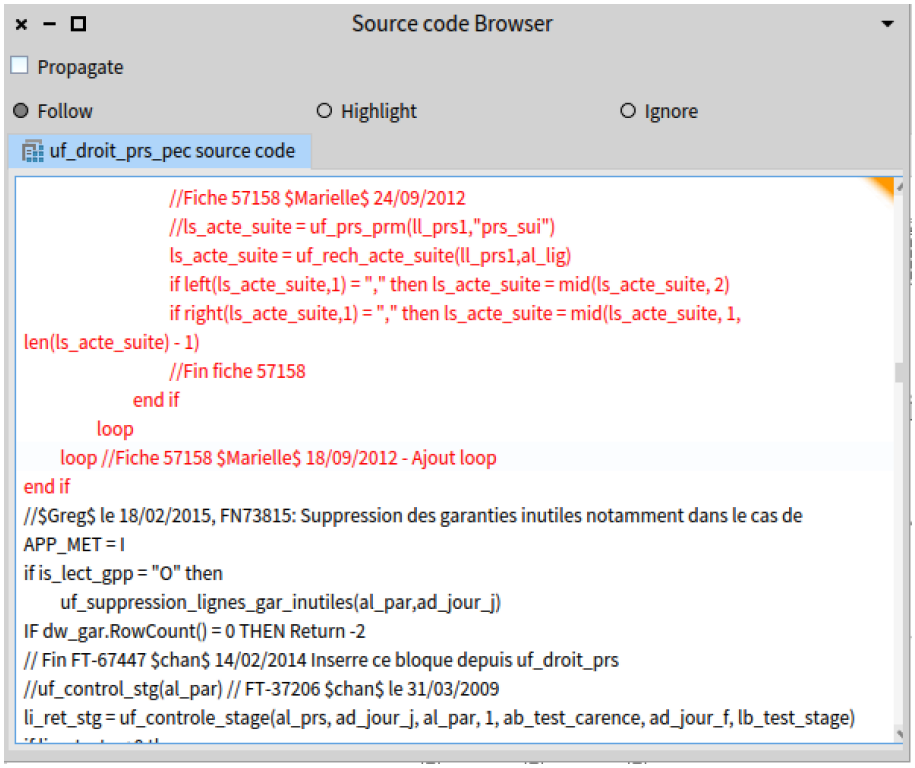

FIGURE 7 – navigateur de code dupliqué

extraire les logiques metier d'un systemes patrimonial. Je souhaite combiner cette méthode avec la méthode proposée [1] pour proposer un outil d'extraction de logique métier dans les systèmes patrimoniaux.

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

5 Publications

Les articles soumis dans le cadre de cette thèse sont :

1. Improving practices in a medium Franch company : First step (Honore et al. [12])
2. Towards a Versatile Reverse Engineering Tool Suite (Honore et al. [11])

6 Formations

Voici la liste des formations que j'ai assistées :

— Les fondamentaux du management d'équipe Session 1 (Ecole doctorale)
— Gestion de conflit (CIM)
— Formation Propriété intellectuelle au service des doctorants tronc commun (Ecole doctorale)
— Intelligence économique et dynamique de l'innovation (Ecole doctorale)
— Communiquer en Anglais - Niveau confirmé - Stage intensif (Ecole doctorale)

7 Projet professionnel

En ce qui concerne mon projet professionnel, je souhaite continuer dans l'enseignement supérieur : donner des cours et continuer dans la recherche. Je pense la reengenierie des système est un axe de recherche ou beaucoup de travaux intéressant sont mener. Néanmoins il reste à faire et je souhaite contribuer a cela. Toutefois, je ne me refuse pas l'idée de démarrer une start-up a l'issue de ma thèse ou travailler dans une entreprise.

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] HOUEKPETODJI Honore, Verhaeghe Benoît, Clotilde Toullec, Derras Mustapha, Fatiha Djareddir, Jerôme Sudich, Ducasse Stéphane, and Etien Anne. Towards a versatile reverse engineering tool suite. In *QUATIC 2020 : 13th International Conference on the Quality of Information and Communications Technology*, Faro, Portugal, 2020.

[12] HOUEKPETODJI Honore, Fatiha Djareddir, Jerôme Sudich, and Nicolas Anquetil. Improving practices in a medium french company : First step. In *RIMEL : Journée de travail Rimel / Lignes de produit / Sécurité*, Paris, France, 2020.

[13] 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.

[14] 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.

[15] Manny Lehman. Laws of software evolution revisited. In *European Workshop on Software Process Technology*, pages 108–124, Berlin, 1996. Springer.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Dan Port and Bill Taber. Actionable analytics for strategic maintenance of critical software : an industry experience report. *IEEE Software*, 35(1) : 58–63, 2017.

[21] 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.

[22] 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.

[23] 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.