

CST –Analyse multi-facettes et opérationnelle pour la transformation des systèmes d'information

HOUEKPETODJI Mahugnon Honoré

1 Description du sujet de la thèse

CIM est une SAS au capital social de 200 k détenu 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'avoir PowerBuilder version 2017 (été 2018) et est en cours de passage sur DB2 v11.

Le système de gestion est centré sur 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 [9].

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

Dans cette section, je présenterai Izy Protect et les travaux qui se penchent sur la question de la ré-ingénierie des systèmes d'information.

2.1 Présentation d'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 de la CIM. Aujourd'hui Izy Protect est maintenu par une équipe de 18 développeurs. Le code source est organisé par bibliothèques Powerbuilder. Izy Protect compte 117 bibliothèques. Les bibliothèques d'Izy Protect peuvent être regroupées en des modules métier. Par contre cette information ne peut pas être directement déduites du code. La plus large a une taille d'environ 300 KLOC. Durant toutes ces années, il y a eu beaucoup de changements 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 d'Izy Protect ont été perdues jusqu'en 2010. De plus, les développeurs risquent à tout moment de faire de régression (casser une fonctionnalité existante). Izy Protect n'est pas couvert avec des tests unitaires automatisés. Ceci augmente la crainte des développeurs pour de petites modifications. Les caractéristiques d'Izy Protect montrent qu'il est un système qui a une grande valeur pour l'entreprise. En plus il est développé avec un vieux langage de programmation : PowerBuilder. Ces caractéristiques reflètent la définition d'un système **patrimonial** selon Demeyer et al. [9]. De plus, quand un système a plusieurs décennies de vie, la rétro-ingénierie est une activité centrale pour le maintenir [9].

Dans l'entreprise, les travaux de maintenances ou de développements sur Izy Protect sont identifiés par des tickets. Les tickets sont stockés dans la base de données des tickets ou fiches navettes depuis 1998. 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(s)

Izy Protect est planifier pour les tests de régressions lorsque les modifications concernent certaines parties du système présélectionnés par l'entreprise. Il faut noter que ces parties sont inchangées. Les tests de régressions sont subdivisés en deux parties : les tests de traitements de données et les tests IHMS. Les tests de données sont purement manuels. C'est des tests *black box* : on passe des données en entrée, on suit l'exécution et on compare la sortie. Tandis que les tests IHMS sont fait avec l'outil Tosca¹. Ceci crée une latence dans la ré-exécution des tests et donc pénalise le temps de livraison du produit au client. De plus, au cas où les modifications concernent d'autres parties du système, il n'y a aucune assurance qu'il n'a pas de régressions ce qui peut parfois se faire sentir chez le client.

2.2 Les tests automatisés et le DevOps

Dudekula Mohammad Rafi et al. [13] ont étudié les avantages et les limitations de l'automatisation des tests. Ils ressortent que les tests automatisés sont répétables et réutilisables. Les tests automatisés assurent aussi une couverture du code et un gain de temps dans l'exécution des tests. Mais en contrepartie, les tests automatisés coûtent pour la mise en place (sélection d'outil, achat de l'outil de tests, conception des cas de tests), formations du personnel.

Wang et al. [30] ont présenté une étude qui examine le degré de maturité du processus d'automatisation des tests par rapport aux pratiques d'automatisation des tests adopter dans ces entreprises de logiciels. Ils remarquent que la culture du DevOps favorise un haut degré de maturité du processus d'automatisation des tests.

Diaz et al. [10] ont enquêté sur pourquoi les entreprises installent la culture du DevOps dans leurs organisations. Les résultats montrent que les entreprises avaient besoin de : être plus agile et rapide, répondre aux tendances du marché, livrer les produits/services de meilleure qualité, réduire le temps de configuration des environnements, améliorer la communication entre les équipes, introduire le processus d'automatisation et amener les commerciaux, les développeurs, et les opérateurs à avoir un objectif fonctionnel commun. L'installation du DevOps à apporter : une rapidité de livraison, une meilleure qualité des produits et services, automatisation des processus : efficacité, optimisation, productivité Améliorer la collaboration et la communication entre les équipes et une meilleure concentration sur les besoins du client.

2.3 Analyse de l'évolution de l'état d'un système logiciel patrimonial

Les systèmes patrimoniaux sont des systèmes en constant changement : production de nouvelles fonctionnalités. La deuxième loi de Lehman [19] 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. Selon Demeyer et al. [9], collecter et interpréter les données et les résumer dans une vue cohérente est une étape importante pour la compréhension initiale d'un système logiciel existant. Dans ce sens, de nombreux travaux de la littérature proposent des techniques, pour suivre l'évolution de l'état de ces systèmes.

Zhang and Kim [31] utilise les données des occurrences bugs et le temps pour modéliser l'évolution d'un système logiciel avec ccharts.

Lenarduzzi et al. [21] 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 versions. Ce qui n'est pas le cas avec Izy Protect.

Port and Taber [26] 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

1. <https://www.tricentis.com/products>

temps, le ratio entre le taux 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.

Bibi et al. [4], Kim et al. [18] proposent des modèles 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 logiciel commercial, multi-utilisateur, et donc chaque utilisateur a des fonctionnalités ou des changements qui lui sont spécifiques.

Nagappan and Ball [22] utilise l'historique le nombre de lignes de code changées pour une correction de bug 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 versions.

Raja et al. [27] a étudié différents algorithmes de modélisation des défauts d'un système à partir des données des défauts relevés sur huit projets dont les codes sources sont ouverts. Il en ressort que la moyenne glissée modélise mieux les défauts des systèmes.

2.4 Rétro-ingénierie

Dans cette section, je présenterai les travaux reliés à l'outillage pour la rétro-ingénierie que j'ai étudiée. Chikofsky and Cross II [6] définit la rétro-ingénierie comme *un processus d'analyse d'un système données 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.*

Bruneliere et al. [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 [8] ou l'interface graphique [29]. 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 - métamodèle. Mais aussi au niveau des outils eux-mêmes (par exemple en agissant sur le modèle).

C'est dans cette optique que Kienle and Müller [17] reconnaissent le besoin d'outils pour la rétro-ingénierie qui fournissent des fonctionnalités permettant d'extraire des informations de bas niveau des systèmes, d'analyser et de générer des connaissances sur les systèmes, et de visualiser ces connaissances afin que les ingénieurs puissent comprendre efficacement les aspects du système qui les intéressent. De ce point de vue ils identifient que ces outils doivent être :

(1) scalable : capacité de ces outils a fonctionné pour la rétro-ingénierie de petites aux larges logiciels ; (2) interopérable : ces outils doivent être capital de Communiquer avec des outils externes ; (3) personnalisable : les activités de rétro-ingénierie étant variables, les utilisateurs d'outils de rétro-ingénierie doivent pouvoir continuellement les adapter pour répondre aux besoins changeants ; (4) utilisable : facilité d'utilisation ; (5) adoptable : facilité d'apprentissage.

Ainsi les outils qui seront proposés pour la rétro-ingénierie d'Izy Protect doivent respecter ces règles-là.

Bellay and Gall [3] ont proposé un ensemble 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 [14] ont identifié des critères pour les outils de réingénierie. Parmi ces critères, j'ai retenu ceux qui se rapportent à la rétro-ingénierie. Ce sont les critères (1) de sélection (2) d'abstraction. Le critère de sélection exige que l'outil de rétro-ingénierie permette à l'utilisateur de sélectionner des éléments du code source d'un système qui répondent à une requête donnée. L'outil doit aussi permettre à l'utilisateur d'abstraire les caractéristiques des entités du code source à un haut niveau d'abstraction. Par exemple : déduire de la complexité cyclomatique des classes d'un package, la complexité cyclomatique du package.

Bruneliere et al. [5] a proposé et mis en œuvre le cadre MoDisco. MoDisco est divisé en 4 couches, un projet de modélisation de la plateforme Eclipse, l'infrastructure, la technologie et les cas d'utilisation. La couche infrastructure contient des outils génériques 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 remaniement de code Java. MoDisco inclut : un navigateur de modèle, un éditeur pour voir le code des entités 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

Izy Protect présente les problèmes cités dans la Section 2.1 en particulier, code non versionné, ce qui engendre les pertes et écrasement de code, puis l'absence totale de tests niveau code source automatisé etc. Mon objectif est de construire des outils visant à aider à la réingénierie de Izy Protect. Demeyer et al. [9] démontrent que les tests sont comme une assurance de vie dans le cadre de la réingénierie logicielle. Car il permet d'assurer les activités de réingénieries ne vont pas modifier la logique métier. De plus la réingénierie de Izy Protect se fera de façon incrémentale, en cas d'erreur, il est important de savoir quel changement a cassé le code. Pour ce faire, un système de contrôle de versions et les tests automatisés seront d'une grande utilité. Le fait de versionner le code non seulement permet d'éviter les problèmes de pertes de code mais aussi à long terme me permettra aussi d'avoir une base d'analyse de l'évolution du code d'Izy Protect supplémentaire et de fournir un outil de vérification des normes de développements sur les changements effectués par les développeurs.

J'ai choisi Subversion (SVN)² par le fait de sa simplicité pour la compréhension. Il faut aussi noter que le module de contrôle de versions 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 versions qu'offre Powerbuilder ne supporte pas bien les opérations avancées comme la comparaison de deux versions du code ou bien la résolution de conflit. Tout cela fait de SVN un choix simple pour commencer. En complément de Powerbuilder, j'ai amené les développeurs de la CIM à utiliser TortoiseSVN³. Il faut signaler que pour le succès de la mise en place, il a fallu une documentation et une assistance continue.

Dans le but d'avoir une base d'analyse de l'évolution du code d'Izy Protect, reconstruire l'histoire du code source d'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 sur les versions produites en 2015.

En ce qui concerne les tests automatisés, j'ai testé PUnit⁴ paraît être la librairie qui permet de tester les fonctionnalités des applications Powerbuilder au niveau du code source.

3.2 Analyse des fiches navettes

A travers l'analyse des tickets, j'espère caractériser l'état d'Izy Protect avec des données factuelles, et suivre les changements du système dans le temps. Ceci permettra d'évaluer l'impact de l'utilisation des outils d'aide à la rétro-ingénierie que je proposerai durant cette thèse. Pour cela, j'ai utilisé la base de données des fiches navettes. Après nettoyage, seul les tickets à partir de 2004 sont utilisables. Les données sont non-stationnaires. En me basant sur les résultats de [27], j'ai utilisé la moyenne glissante avant un pas de 2 mois pour modéliser l'état du système. J'ai principalement mesuré les métriques suivantes : l'évolution du temps pour fermer les tickets, l'évolution du temps nécessaire aux développeurs, l'évolution du temps des tests manuels du développeur, l'évolution de l'estimation du temps de développement par le manager. Les résultats sont présentés dans un tableau de bord qui se met à jour chaque mois. La Figure 1 montre l'aperçu du tableau de bord.

TABLE 1 – Tickets

	correction	évolution	Total
Tickets	15407	11973	27380
% temps	28%	78%	100%

Comme le montre la Table 1, 28% du temps passé sont pour des corrections tandis que 78% sont pour des évolutions. Selon Pigoski [25] la proportion de tickets de corrections doit être entre 20% et 25%. Par conséquent la proportion de tickets de corrections sur Izy Protect paraît élevée.

J'ai aussi remarqué une augmentation du temps passé par les développeurs pour traiter les évolutions ou les corrections. Ceci pourrait être une conséquence de la complexité du code d'Izy Protect, ou bien, une mauvaise compréhension des besoins du client qui fait que le développeur prend du temps à comprendre le travail à faire. De plus j'ai remarqué les courbes d'Izy Protect et de son *module Prestation* ont relativement les mêmes allures tout comme si tout le travail des développeurs ces dernières années n'est concentrée que sur ce module-là. Les données montrent aussi que les développeurs passent de moins en moins de temps à tester leur code. Est-ce une démotivation due à l'état d'Izy Protect ?

2. <https://subversion.apache.org/>

3. <https://tortoisesvn.net/>

4. <https://sourceforge.net/p/pbunit/wiki/Home/>

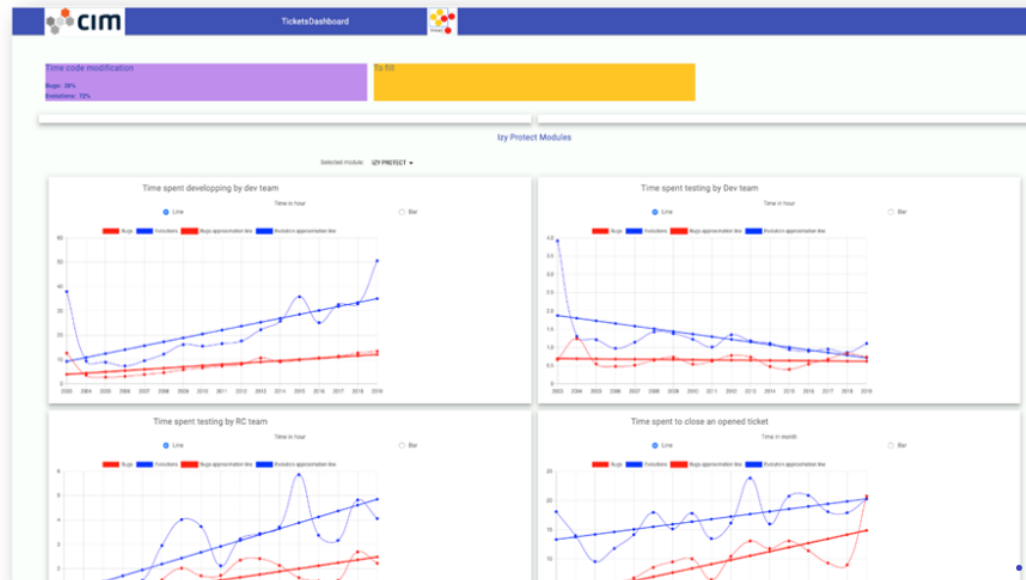


FIGURE 1 – Tableau de bord d’analyse de tickets

En résumé, les données révèlent les craintes de la société. Avec le tableau de bord de la Figure 1 je peux continuellement mesurer l’impact de l’utilisation des outils d’aide à la rétro-ingénierie que je proposerai.

3.3 Outil d’aide à la rétro-ingénierie logicielle

Pour répondre aux exigences détaillées dans la Section 2.4, j’ai développé une suite d’outils d’aide à la rétro-ingénierie. Ces outils sont développés au-dessus de la plateforme Moose [23]. En effet, la plate-forme offre un métamodèle 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 : utilisés 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

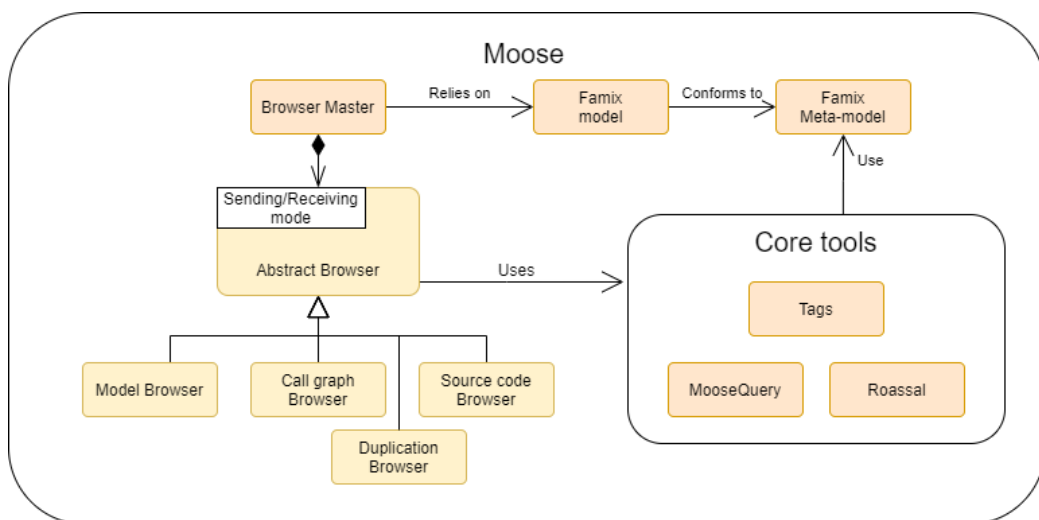


FIGURE 2 – Architecture de la suite d’outils

La Figure 2 montre l’architecture globale de la suite d’outils. L’architecture globale de la suite d’outils est principale-

ment 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 faciliter 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 ouverts. Il se charge de notifier tous les navigateurs ouverts en cas d'événement.

Chaque navigateur fonctionne sur une entité du modèle local à lui. Cette entité peut être une seule entité ou un groupe d'entité. Il peut émettre ou recevoir un événement. Tous les navigateurs présentent un bouton qui indique son mode d'émission d'événement, et trois boutons qui indiquent son mode de réception d'événement (e.g. Figure 3).

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

Le comportement d'un navigateur à 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 remplace son entité courante par l'entité reçue via l'événement. (2) *highlight* : le navigateur cherche l'entité reçue via l'événement dans son entité courante, s'il la trouve, il la colorie. (3) *ignore* : le navigateur ne fait rien à la réception de l'événement.

3.3.2 Nivigateur de modèle

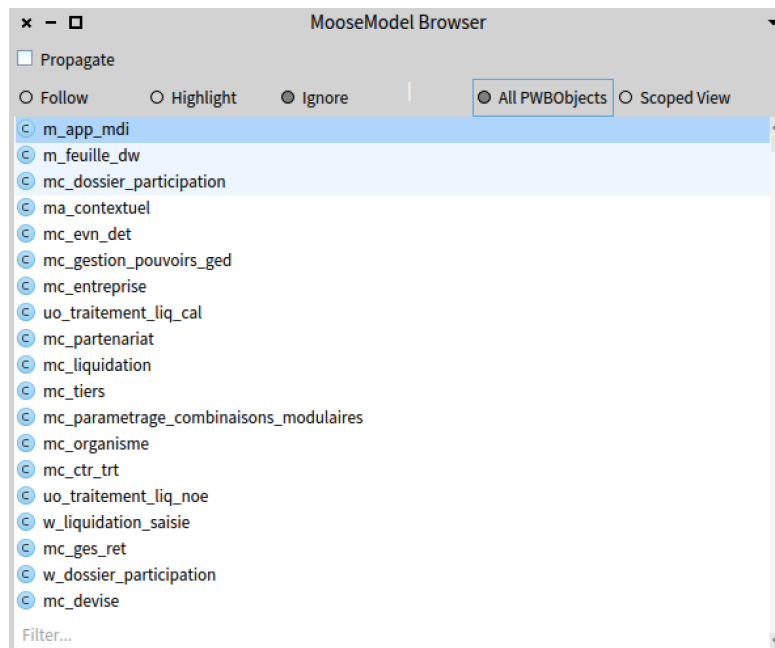


FIGURE 3 – Nivigateur de modèle

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

Le contenu navigateur présente par défaut la liste des entités que contient l'entité courante 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 structurée. Par exemple, si l'utilisateur recherche *include : FamixPWBAtribute*, le résultat sera toutes les entités contenant des FamixPWBAtribute (attributs Powerbuilder).

3.3.3 Nivigateur de graphe d'appel

La Figure 4 donne un aperçu du graphe d'appel avec les options. La fenêtre (1) de cet outil permet principalement de visualiser sous forme d'un graphe les entités qui utilisent l'entité courante du navigateur. Les nœuds représentent les entités. Les flèches représentent l'ensemble des dépendances entre deux entités. Le sens de la flèche indique le sens des dépendances. En effet pour un système comme Izy Protect par exemple, ce graphe peut rapidement devenir illisible. Pour

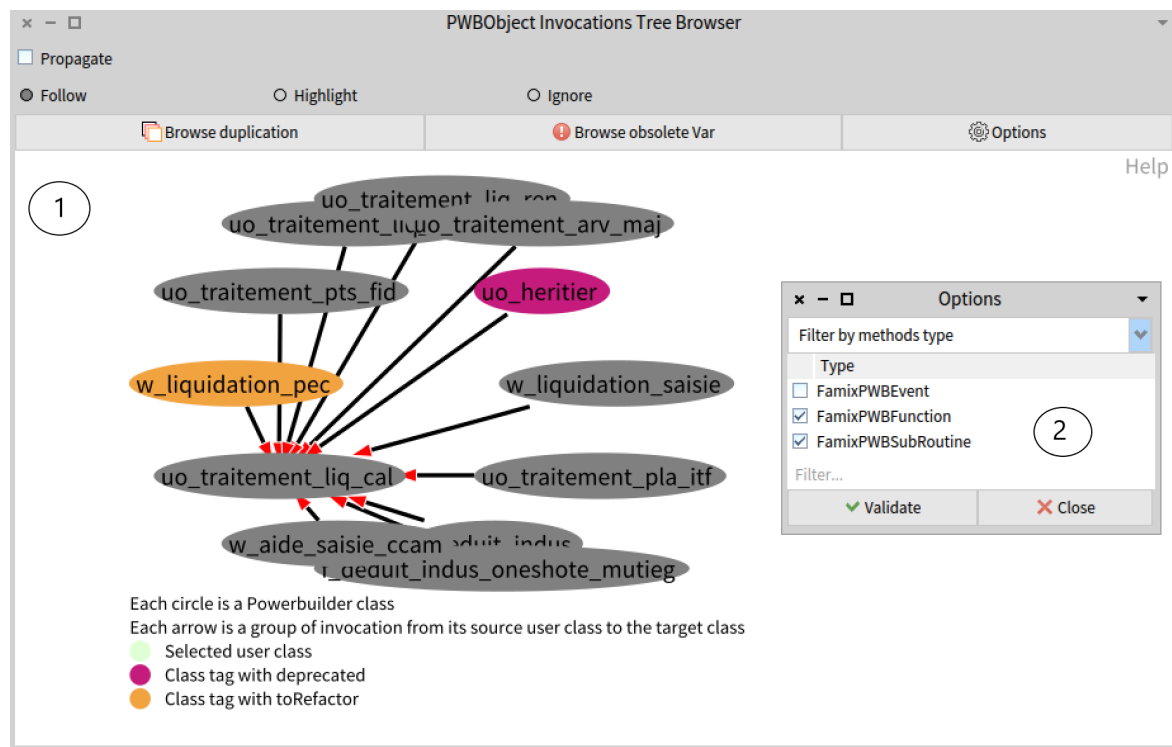


FIGURE 4 – Graphe d'appel avec l'option de reduction de complexité du graphe

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 à l'origine des appels. Afin de donner plus de contexte aux développeurs, quand il glisse la souris sur une flèche, un pop-up 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 l'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

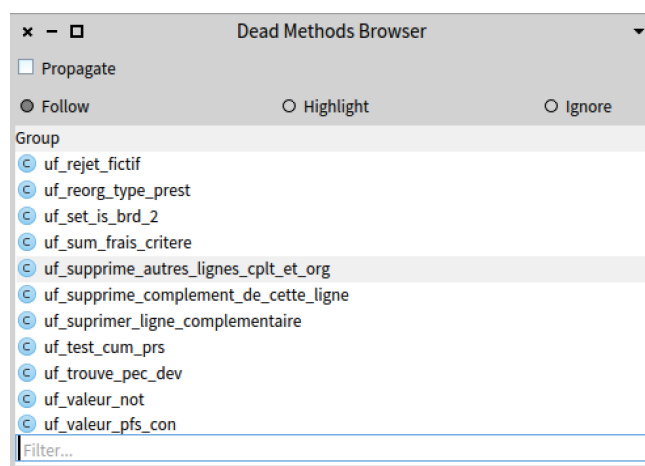


FIGURE 5 – Navigateur de code mort

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

3.3.5 Navigateur de code dupliqué

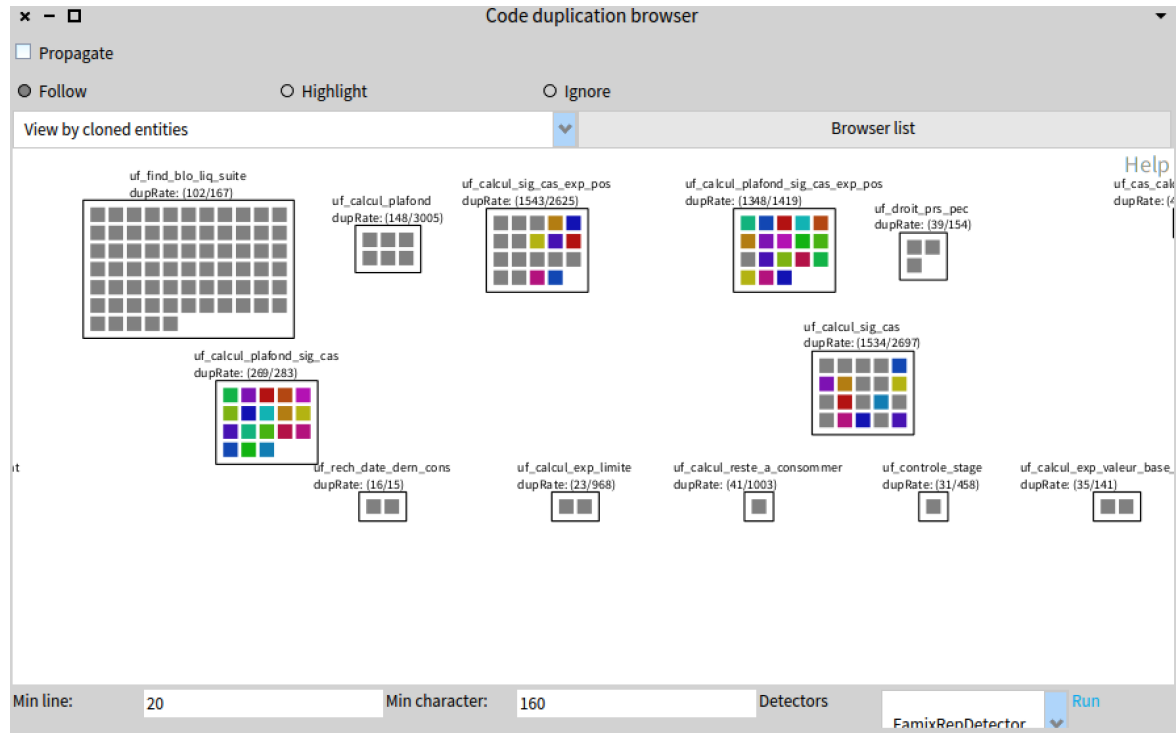


FIGURE 6 – Navigateur de code dupliqué

La Figure 6 présent le navigateur de code dupliqué. Les carrés externes représentent les entités qui présentent des clones. Dans le cas de la Figure 6, ces entités sont des méthodes. Les carrés internes représentent les clones que présente une entité. Les clones sont représentés de façon à ce que l'utilisateur puis facilement l'inspecté. Il utilise actuellement un algorithme de détection basé sur l'égalité stricte des chaînes de caractères [11]. Cet algorithme peut être remplacé par un algorithme plus sophistiqué pour détecter les doublons [28].

Sur la Figure 6 les entités *uf_calcul_sig_cas_exp_pos* et *uf_calcul_sig_cas* présentent de clones en commun. Quand l'utilisateur clique sur *uf_calcul_sig_cas_exp_pos*, les clones de *uf_calcul_sig_cas_exp_pos* prennent des couleurs différentes. Les clones que *uf_calcul_sig_cas_exp_pos* a en commun avec *uf_calcul_sig_cas*, dans *uf_calcul_sig_cas* prennent les mêmes couleurs que leurs semblables dans *uf_calcul_sig_cas_exp_pos*. Par exemple les carrés bleus claires dans *uf_calcul_sig_cas* a pris la couleur du carrés bleus claires dans *uf_calcul_sig_cas_exp_pos* parceque les deux carrés représentent le même fragment de code. Cela permet de voir plus facilement quelles entités ont des clones 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 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

3.3.7 Conformité des navigateurs par rapport aux critères des outils de rétro-ingénierie

J'ai construit les navigateurs de façon a ce qu'ils respectent les critères précités dans la Section 2.4. En effet, l'environnement Moose offre les propriétés de **scalabilité**, **d'interopérabilité** [12] dont herite les outils. A travers les différents réunions présentations des outils au développeurs, j'ai tenu compte de leurs critiques afin que les outils non seulement, répondent à leurs besoins mais aussi ,facile d'utilisation et d'apprentissage. Le *Navigateur de graphe d'appel* Section 3.3.3 répond aux critères **d'abstraction** car il permet par exemple d'abstraire l'ensemble des invocations des méthodes d'une classe au niveau de la classe .

Navigateur de modèle Section 3.3.2 se conforme au critère de sélection car il permet de selection un groupe d'entités respectants une condition bien déterminée. sélection (2) d'abstraction

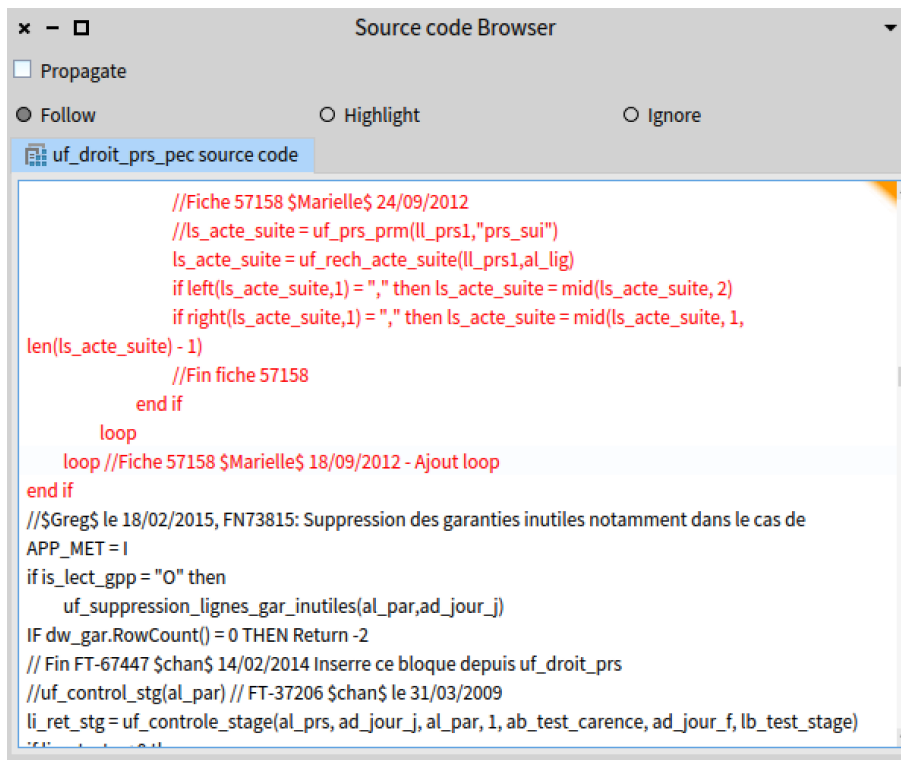


FIGURE 7 – Navigateur de code dupliqué

4 Futurs travaux

Tous les outils présents ci-dessus sont conçus dans le but de nettoyer le code de visualiser les interactions entre les différentes classes d'Izy Protect. Par contre les développeurs ne les utilisent pas encore dans leur quotidien. Pour vérifier que les modifications n'introduisent pas d'erreurs, j'envisage de mettre en place les tests automatisés avec PUnit. Ceci m'amènera à étudier les différentes approches de tests abordés dans la littérature afin d'identifier une approche qui s'adapte au contexte d'Izy Protect. Toutefois je pense que transformer les tests de données énoncés dans la Section 2.1 en tests automatisés sera un point de départ. Je projette aussi d'analyser les changements entre les versions Izy Protect pour fournir des indicateurs supplémentaires sur l'évolution du système. Sans oublier d'évaluer mes outils.

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 littérature dans ce sens en notament [7, 20, 24] qui se basent sur l'analyse statique des interactions des fonctions avec les variables pour extraire les logiques métiers d'un système patrimonial. Je souhaite combiner cette méthode avec la méthode proposée [1] pour proposer un outil d'extraction des logiques métiers dans les systèmes patrimoniaux.

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

5 Publications

Dans le cadre de cette thèse,

- l'article : *Improving practices in a medium Franch company : First step* (Honore et al. [16]) a été soumis
- l'article : *Towards a Versatile Reverse Engineering Tool Suite* (Honore et al. [15]) a été accepté en version court

6 Formations

Pour les formations doctorale, je cumule un total de 41 crédits sur les 40 crédits minimum demandés pour une thèse CIFRE. Voici la liste des formations que j'ai assistées avec leurs crédits.

- Les fondamentaux du management d'équipe Session 1 (Ecole doctorale) (7)
- Gestion de conflit (CIM)
- Formation Propriété intellectuelle au service des doctorants tronc commun (Ecole doctorale) (10)
- Intelligence économique et dynamique de l'innovation (Ecole doctorale) (10)

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 réingénierie des systèmes logiciels est un axe de recherche ou beaucoup de travaux intéressants sont menés. Néanmoins il reste à faire et je souhaite contribuer à cela. Toutefois, je ne me refuse pas l'idée de démarrer une start-up à 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] Valerio Cosentino, Jordi Cabot, Patrick Albert, Philippe Bauquel, and Jacques Perronnet. A model driven reverse engineering framework for extracting business rules out of a java application. In Antonis Bikakis and Adrian Giurca, editors, *Rules on the Web : Research and Applications*, pages 17–31, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32689-9.
- [8] Julien Delplanque, Anne Etien, Nicolas Anquetil, and Stéphane Ducasse. Recommendations for evolving relational databases. In *32nd International Conference on Advanced Information Systems Engineering*, 2020.
- [9] 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>.
- [10] Jessica Diaz, Daniel López-Fernández, Jorge Perez, and Ángel González-Prieto. Why are many business instilling a devops culture into their organization?, 2020.
- [11] 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>.
- [12] Stéphane Ducasse, Tudor Gîrba, Michele Lanza, and Serge Demeyer. Moose : a collaborative and extensible reengineering environment. In *Tools for Software Maintenance and Reengineering, RCOST / Software Technology Series*, pages 55–71. Franco Angeli, Milano, 2005. ISBN 88-464-6396-X. URL <http://rmod.inria.fr/archives/papers/Duca05aMooseBookChapter.pdf>.
- [13] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, K. Petersen, and M. V. Mäntylä. Benefits and limitations of automated software testing : Systematic literature review and practitioner survey. In *2012 7th International Workshop on Automation of Software Test (AST)*, pages 36–42, 2012.
- [14] 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>.
- [15] HOUKPEODJI Honore, Verhaeghe Benoît, Clotilde Toullec, Derras Mustapha, Fatiha Djareddir, Jérôme 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.
- [16] HOUKPEODJI Honore, Fatiha Djareddir, Jérôme 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.
- [17] 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.
- [18] 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.
- [19] Manny Lehman. Laws of software evolution revisited. In *European Workshop on Software Process Technology*, pages 108–124, Berlin, 1996. Springer.

- [20] 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.
- [21] 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.
- [22] 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.
- [23] 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.
- [24] Kestutis Normantas and Olegas Vasilecas. Extracting business rules from existing enterprise software system. In *International Conference on Information and Software Technologies*, pages 482–496. Springer, 2012.
- [25] Thomas M. Pigoski. *Practical Software Maintenance : Best Practices for Managing Your Software Investment*. Wiley Publishing, 1st edition, 1996. ISBN 0471170011, 9780471170013.
- [26] Dan Port and Bill Taber. Actionable analytics for strategic maintenance of critical software : an industry experience report. *IEEE Software*, 35(1) : 58–63, 2017.
- [27] 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>.
- [28] 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.
- [29] 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>.
- [30] Yuqing Wang, Mika V. Mäntylä, Serge Demeyer, Kristian Wiklund, Sigrid Eldh, and Tatu Kairi. Software test automation maturity – a survey of the state of the practice, 2020.
- [31] H. Zhang and S. Kim. Monitoring software quality evolution for defects. *IEEE Software*, 27(4) :58–64, July 2010. ISSN 1937-4194. doi: 10.1109/MS.2010.66.