ENSICAEN
6, bd maréchal Juin
F-14050 Caen cedex 4

Pattern Trace Identification, Detection, and Enhancement in Java

Spécialité Informatique - 2e année

Rapport de stage de formation

Nettoyage, amélioration et mise en ligne de SQUANER

Samuel AUGUSTE

Suivi laboratoire : Yann-Gaël GUÉHÉNEUC
Suivi ENSICAEN : Christophe ROSENBERGER

Mai – Juillet 2011

Remerciements

Je tiens à remercier le professeur Yann-Gaël Guéhéneuc, pour m'avoir accepté au sein de son équipe, ainsi que tous les membres du laboratoire pour leur accueil chaleureux et leur gentillesse tout au long de mon séjour parmi eux.

Je remercie aussi Cécile Huet, directrice du bureau des relations internationales de l'ENSICAEN, et tous ceux qui contribuent à nous donner la chance d'effectuer un stage à l'étranger, en faisant d'importantes démarches pour nous aider à le trouver et à obtenir des aides financières.

Je remercie également toutes les personnes qui m'ont donné des conseils, émis des avis critiques, corrigé mon rapport, ou plus généralement toutes celles qui m'ont aidé à mener à bien mon stage.

Enfin, un grand merci à toutes les personnes que je n'ai pu rencontrer que trop brièvement sur Montréal, qui m'ont fait apprécier mon séjour et ne m'ont donné qu'une seule envie : celle d'y retourner !

Table des matières

Introduction

Pour compléter la deuxième année de formation d'élève ingénieur en informatique à l'ENSICAEN, il nous faut réaliser un stage à l'étranger. J'ai ainsi eu la chance d'aller travailler pendant trois mois à l'école polytechnique de Montréal, au sein de son département de génie informatique et génie logiciel.

Mon objectif était d'unifier tous les outils d'analyse qualitative de code orienté-objet développés par l'équipe des laboratoires Ptidej et Soccer, afin de créer un système d'analyse automatique de référentiels SVN, dès qu'un changement de contenu y serait détecté. Plusieurs tentatives inachevées de création d'un tel système existant sur le serveur du laboratoire, je devais dans un premier temps identifier leurs composants et supprimer ou archiver tous ceux qui me seraient inutiles.

Il me semble naturel de vous présenter mon environnement de travail en premier lieu. Dans une seconde partie, je parlerai du nettoyage du serveur, dresserai le bilan de l'état des lieux qui a été réalisé et expliquerai comment une solution a été retenue à partir de celui-ci. Ensuite, je traiterai des améliorations qui lui ont été apportées, afin de la rendre utilisable. L'intégration de la solution au serveur du laboratoire constituera la dernière partie sur ce sujet. Enfin, je détaillerai les autres tâches qui m'ont occupé au laboratoire durant cette période de stage.

1 Présentation du stage

1.1 École polytechnique de Montréal

L'Université de Montréal est une université publique francophone, fondée en 1878. Elle fait partie des principales universités canadiennes : c'est la deuxième du pays au niveau du nombre d'étudiants, du budget alloué à la recherche, et c'est la plus grande université francophone du monde [1].

L'École Polytechnique de Montréal est affiliée à l'Université de Montréal. Elle a été fondée en 1873 et compte actuellement près de 6300 étudiants. Ses missions sont la recherche en génie et l'enseignement de l'ingénierie dans toute la diversité de ses domaines d'applications [2].

1.2 Laboratoires d'accueil

Le département de génie informatique et génie logiciel se situe dans le pavillon Claudette Mackay-Lassonde, datant de 2005, qui offre un environnement de travail très agréable.

L'équipe Ptidej (*Pattern Trace Identification, Detection, and Enhancement in Java*) est dirigée par Yann-Gaël Guéhéneuc. Son but est de développer des méthodes et des outils pour évaluer et améliorer la qualité de programmes codés en langages orientés objets, en particulier grâce à l'utilisation de motifs et de patrons de conception [3]. Des travaux majeurs sont l'étude de leurs impacts sur la qualité du code et la création de la suite d'outils Ptidej [4].

J'étais également sous la direction du professeur Giuliano Antoniol [5], qui dirige l'équipe Soccer (*SOftware Cost-effective Change and Evolution Research*). En effet, ces deux professeurs ont décidé de travailler ensemble, répartissant ainsi leurs étudiants dans leurs différents locaux suivant le diplôme préparé. Pour ma part, j'ai été installé dans le laboratoire des étudiants en *Ph.D.* où la langue de travail était principalement l'anglais, aucun d'eux n'étant natif de la province de Québec et étant le seul français parmi ces étudiants.

1.3 Déroulement du stage

À mon arrivée, le sujet de stage n'était pas encore clairement défini. Plusieurs versions de programmes pour automatiser l'analyse qualitative de codes orientés objets à partir d'un répertoire SVN coexistaient sur le serveur mais semblaient inabouties ou inutilisables. De plus, cela encombrait le serveur de par la diversité d'outils présents. Aucune documentation n'ayant été réalisée, il a été décidé que mon premier travail serait de faire un état des lieux du serveur du laboratoire, afin de pouvoir le nettoyer, de décider de la solution à retenir, puis de la rendre utilisable sur le serveur (*figure 1*).

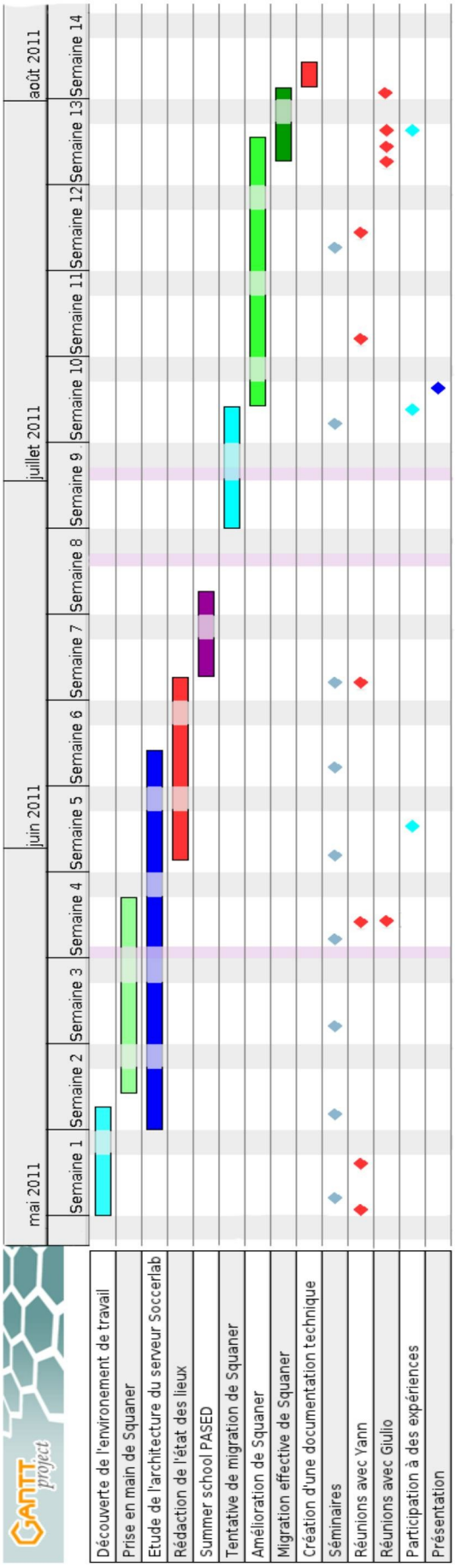


Figure 1 – Diagramme de Gantt.

2 État des lieux du serveur

2.1 Méthodologie

Le dernier système mis en place pour contrôler la qualité du code orienté objets était Squaner (*Software QUality ANalysER*), développé par un stagiaire l'année passée. Contacter celui-ci pour échanger par visioconférence grâce à Skype m'a permis d'en savoir plus sur le fonctionnement de Squaner, ses différents composants et le prédécesseur à Squaner : Gutsy.

L'autre moyen d'obtenir des informations était l'exploration du serveur du laboratoire, nommé Soccerlab. Ceci pour tenter de trouver les dossiers de serveurs Tomcat [6], de pages web, les bases de données utilisées, l'adresse des sites web, ou, plus difficile, du code source. Au final, de nombreux composants ont été mis à jour (*figure 2*), ce qui a ensuite permis de comparer les différentes solutions pour n'en retenir qu'une.

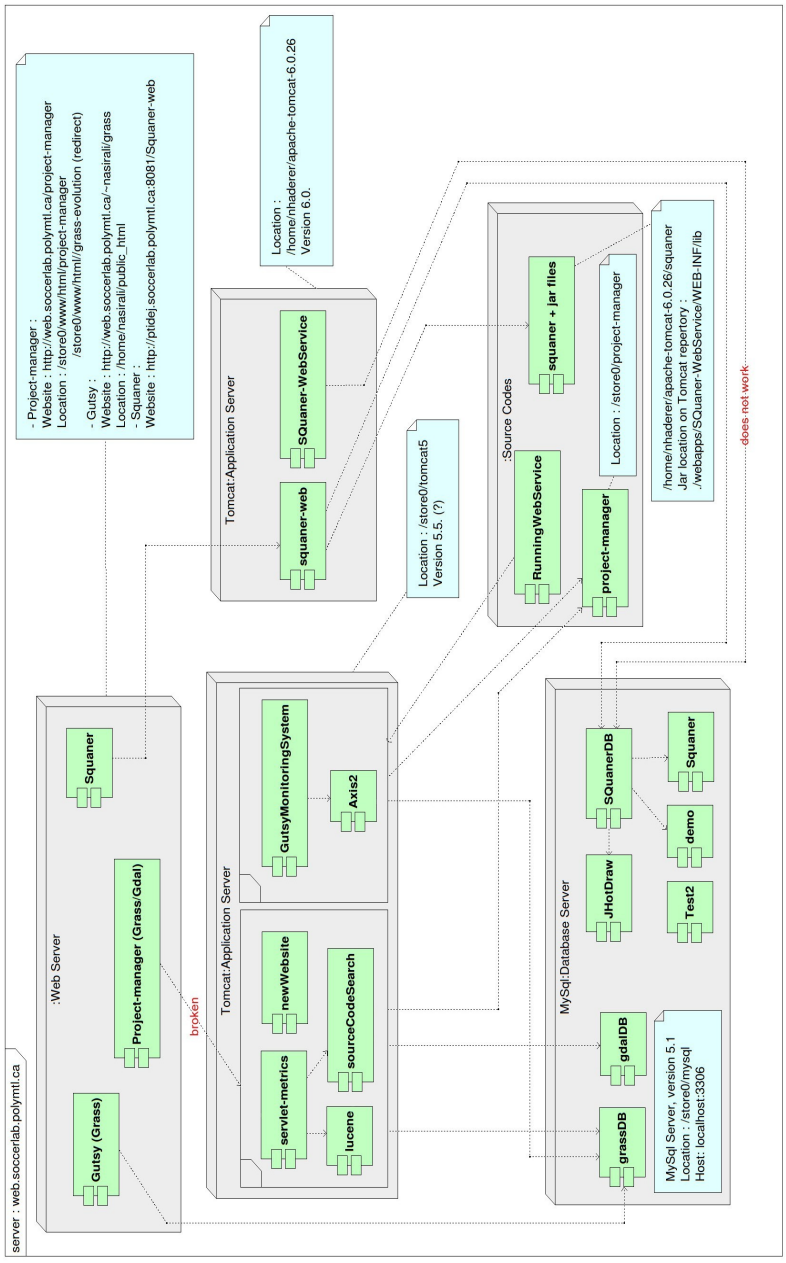


Figure 2 – Diagramme de déploiement.

Le reste de cette section correspond en grande partie à un résumé traduit de l'état des lieux rédigé en anglais dans le cadre du stage (voir *annexe 1*).

2.2 Systèmes abandonnés

Project-manager est le premier système a avoir été mis en place. Il fut créé en 2008 et 2009 par Bruno Genna, Jeremy Watso-Cournoyer, Mathieu Denis et Xavier Tremblay. Son rôle est de télécharger du code, uniquement depuis deux répertoires SVN correspondant aux projets Grass [7] et Gdal [8], de l'analyser et d'envoyer un courriel récapitulatif aux développeurs, mettant l'accent sur les principaux défauts détectés.

Son site web a un design très simple et ne marche pas (*figure 3*). Plus précisément, on ne peut pas accéder aux classes et résultats d'analyses, car l'arborescence des classes n'apparaît pas sur le coté gauche, le code contenant des liens brisés vers d'anciennes applications web du serveur Tomcat.

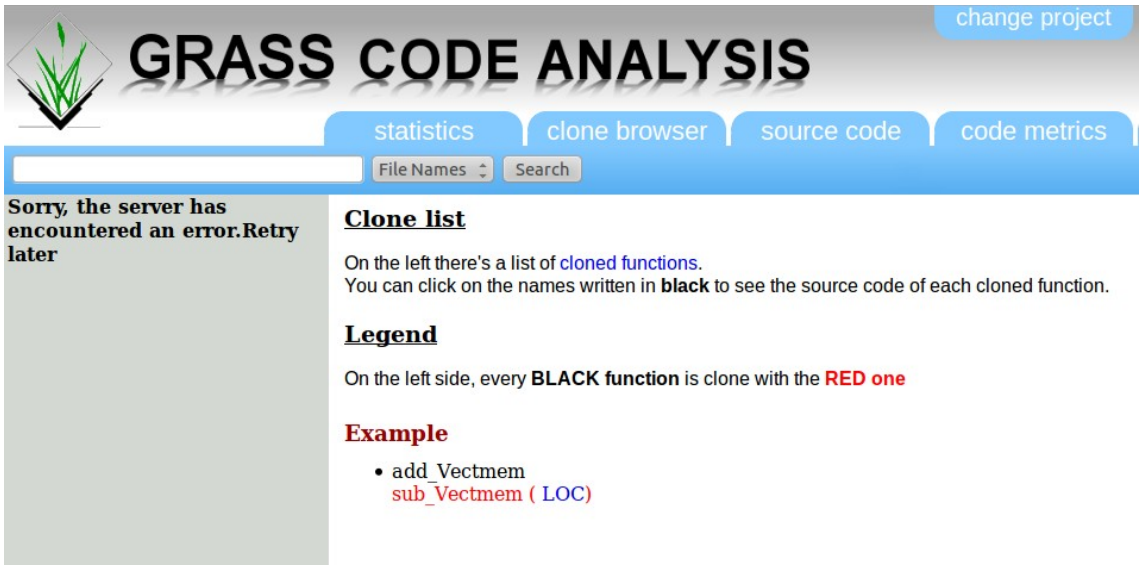


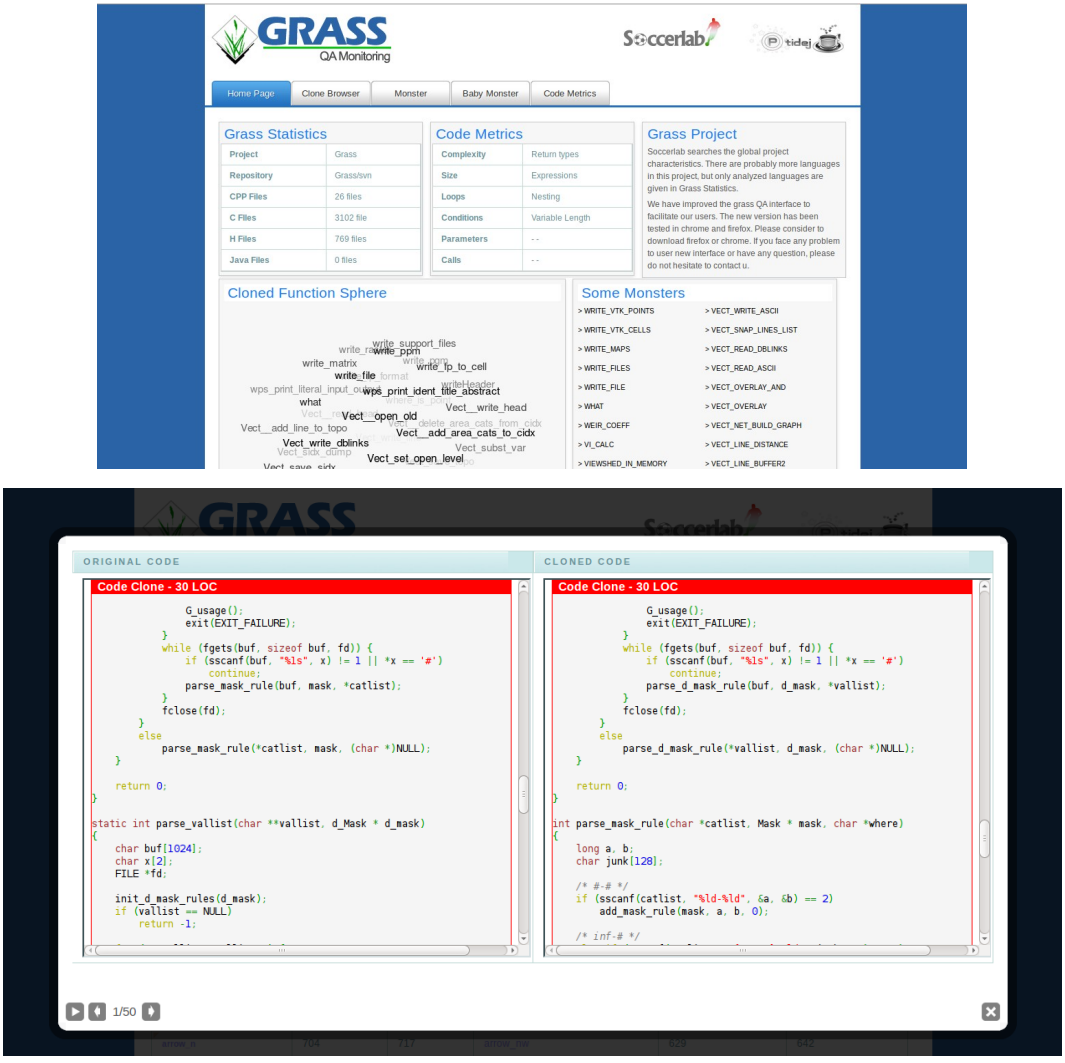
Figure 3 – Page d'accueil du site web de Project-manager.

Les fonctionnalités et métriques censées être calculées sont de bas niveaux et sont incluses dans le système suivant, Gutsy. Nous les aborderons donc plus tard (page suivante). Le problème principal est que Gutsy a été créé par dessus Project-manager, le rendant inutilisable de par la modification de certaines parties de son code et la réutilisation de l'une de ses deux bases de données.

Gutsy est basé sur Project-manager et a été développé par Nicolas Haderer en 2010. Gutsy télécharge du code C ou C++, l'analyse grâce à l'outil srcML [9] afin de créer un document XML qui résume la structure du code. Ensuite, ces informations sont stockées dans une base de données et quatre métriques sont extraites pour chaque fonction : le nombre de paramètres, le nombre d'appels, le nombre de lignes de code et la complexité cyclomatique de McCabe. Ces métriques servent à diviser les fonctions dans quatre groupes d'importance égale en les répartissant suivant leur qualité. Les métriques servent également à comparer les fonctions, afin d'identifier le code cloné.

Gutsy offre des services web par l'intermédiaire du serveur Tomcat, ce qui permet d'ajouter des projets ou de lancer l'analyse à distance.

Par la suite, son site web a été créé, par Nasir Ali. Il est totalement indépendant des services web de Gutsy et s'appuie uniquement sur les résultats d'analyse du projet Grass, stockés dans une base de données pour en afficher les résultats. L'utilisation de ce site web est intuitive et le design très moderne (*figures 4 et 5*). Il est complètement terminé et permet d'afficher tous les résultats d'analyses, y compris les métriques obtenues par le parseur du fichier XML.



Figures 4 et 5 – Site web pour l'analyse du projet Grass par Gutsy.

Bien que ce site web soit très bon en comparaison de celui de Project-manager et ait vocation à le remplacer, son rôle est très limité de par son indépendance vis à vis de Gutsy. Pour tout autre projet analysé que Grass, le site n'est pas valable.

D'autre part, on peut discuter de la pertinence des analyses faites par Gutsy. En effet, les critères utilisés et la méthode de classement des fonctions n'est pas forcément pertinente ; il suffit qu'une fonction ait beaucoup de paramètres pour qu'elle soit considérée comme de basse qualité. De plus, il peut être intéressant d'avoir ces métriques, mais l'estimation de la qualité d'une fonction devrait être fixée indépendamment de celle des autres fonctions, ce qui n'est pas le cas ici en les répartissant par quarts. On remarque aussi que les outils de la suite Ptidej ne sont jamais utilisés. Ce sont ces raisons qui ont poussé à la création de Squaner.

2.3 Squaner

Squaner est le dernier système à avoir été développé, par Nicolas Haderer en 2010. Squaner analyse la qualité de tout code orienté objets situé dans un SVN grâce aux outils de la suite Ptidej et permet également de contrôler l'évolution de la qualité du code entre deux analyses consécutives. Il calcule beaucoup plus de métriques que les précédents systèmes et fournit une interface web pour l'affichage des résultats et l'ajout de nouveaux projets pour analyses.

Un dossier nommé squaner est présent sur le serveur Soccerlab et est essentiel au bon fonctionnement du système. Celui-ci contient des fichiers de configuration, comme les identifiants à utiliser pour le serveur MySQL [11], ainsi que des dossiers contenant le code téléchargé de chaque projet ajouté à Squaner.

Sur le serveur MySQL, plusieurs bases de données ont été créées en lien avec Squaner. *SQuanerDB* est la principale, qui recense les différents projets ajoutés à Squaner. Chaque projet possède également une base de données à son nom pour y stocker entre autre la configuration et les résultats de l'analyse. De nombreuses bases de données ont d'ailleurs été créées sur le serveur pour des tests de Squaner, mais ne sont plus utilisables (*figure 1*), le code associé étant absent du dossier squaner. Quant aux tables de configuration, elles ne sont pas correctement exploitées. Elles contiennent de nombreux champs qui devraient permettre de paramétrer l'analyse, mais qui ne sont malheureusement pas modifiables au travers de l'interface web.

Le fonctionnement de Squaner est le suivant : il récupère dans la base de données l'adresse des SVN des différents projets et vérifie les numéros de révisions. En cas de changement de révision, le nouveau code est téléchargé. Dans l'état initial, Squaner est censé ne pouvoir utiliser que les fichiers C++ et les binaires Java (.class). Ceux-ci sont analysés grâce aux nombreux outils de la suite Ptidej : CPL, Ptidej, PADL, Pom, JChoco, Sad … (*voir annexe 1* pour plus de détails sur l'analyse). Ces derniers permettent ensuite de récupérer de nombreuses métriques, de détecter patrons et anti-patrons, ainsi que de calculer des caractéristiques comme le couplage, la modularité, l'efficacité, ou encore la compréhensibilité du code. Après chaque analyse, un courriel est envoyé aux développeurs avec les résultats ainsi qu'un rapport sur les classes de moins bonne qualité.

Sur le serveur Soccerlab, un serveur Tomcat est présent. Toutefois, le précédent développeur n'a pas réussi à s'en servir pour Squaner et a installé un deuxième serveur Tomcat dans son dossier personnel. Sur celui-ci, deux applications web sont présentes : *SQuaner-WebService* et *Squaner-web*. La première est une tentative de mise en place de services web pour contrôler Squaner, mais qui n'a pas été terminée, rapidement remplacée par un site web codé en JSP. De plus le serveur Tomcat ne tourne pas et, même en le démarrant, aucune analyse ne s'effectue, le programme n'étant pas opérationnel.

Le site web semble inabouti et ne marche pas correctement. Il y a des problèmes d'encodage, des liens qui n'ont pas de raisons d'être mais surtout des espaces prévus pour des fonctionnalités encore non développées (*figure 6*) et des bugs d'affichage (*figure 7*) et de conception. Par exemple, il est possible d'ajouter un projet sans nom, en laissant le champ de celui-ci vide dans le formulaire. Le projet sera alors visible sur le site web sous le nom de *null*, mais aucune base de données ne sera créée, ce qui le rend inutilisable.

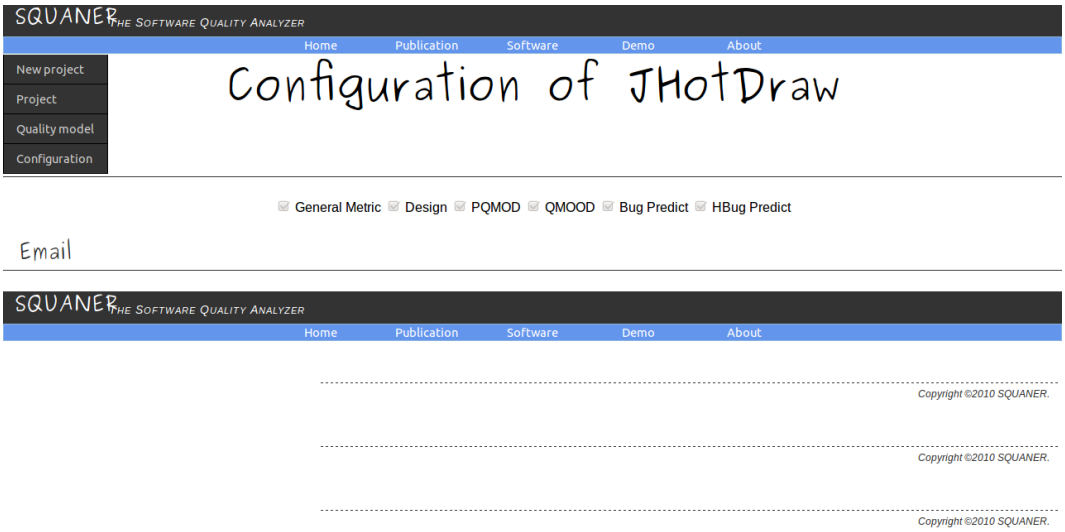


Figure 6 – Site web de Squaner, page de configuration.

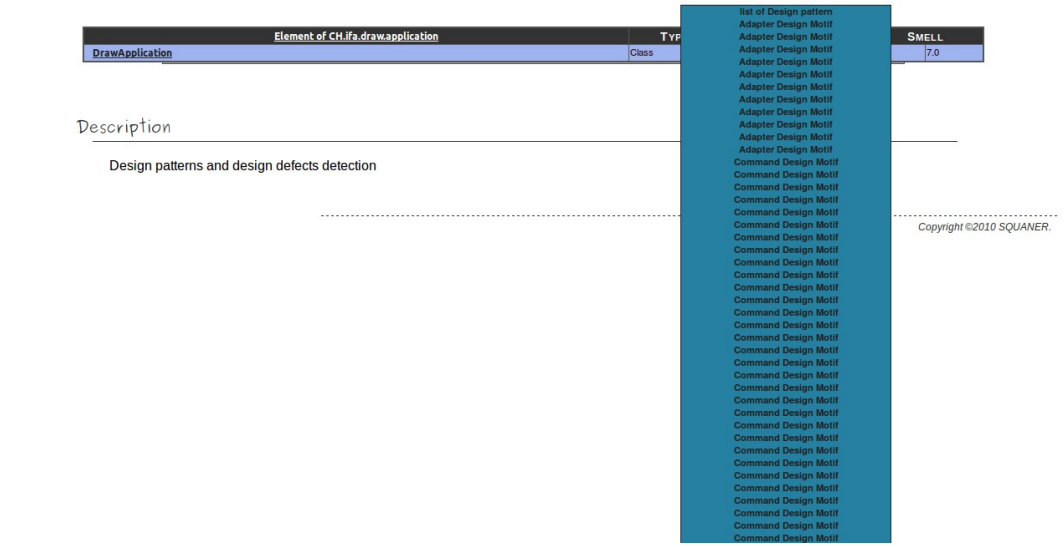


Figure 7 – Site web de Squaner, affichage des patrons de conception.

Squaner offre donc des analyses qualitatives plus complètes et précises que Gutsy et inclut le site web dans sa solution. Bien que son code soit très vaste, son architecture est plutôt bien construite et a été conçue de manière évolutive. La solution à retenir est donc Squaner mais il reste à le terminer. De nombreuses améliorations peuvent être envisagées, comme la simplification du contenu des courriels envoyés, l'optimisation du temps d'analyse, l'ajout de parseur pour le code source Java, le mise en place effective des fonctionnalités de configuration de l'analyse…

Il faut également nettoyer Squaner, de nombreux composants n'ayant pas vocation à rester, que ce soit certaines bases de données ou le serveur Tomcat faisant doublon. En conséquence, les composants des versions antérieures seront inutilisés et donc à supprimer ou à archiver. La suite logique a donc été de présenter l'état des lieux à mon encadrant, en expliquant le système retenu et les solutions envisagées, afin de définir les travaux à accomplir par la suite et leur ordre de priorité.

3 Amélioration du code de Squaner

3.1 Prise en main de Squaner

La réalisation de l'état des lieux m'a permis de comprendre les interactions entre Squaner et les différents composants dont il a besoin. Cependant, des éléments fondamentaux manquaient encore avant de pouvoir entamer les modifications, le plus important étant bien sûr le code source. En effet, seuls des fichiers compilés (.class, archives War…) étaient présents sur le serveur. Le fait d'être entré en contact avec le stagiaire de l'an passé m'a permis de récupérer son code dans mon espace de travail et d'avoir de précieuses données sur le rôle et la structure des différent projets, paquetages et classes de son espace de travail.

Dans un premier temps, une difficulté a été la complexité du système, avec de nombreux projets et outils utilisés (*figure 8*), ce qui rend le temps de familiarisation très long.

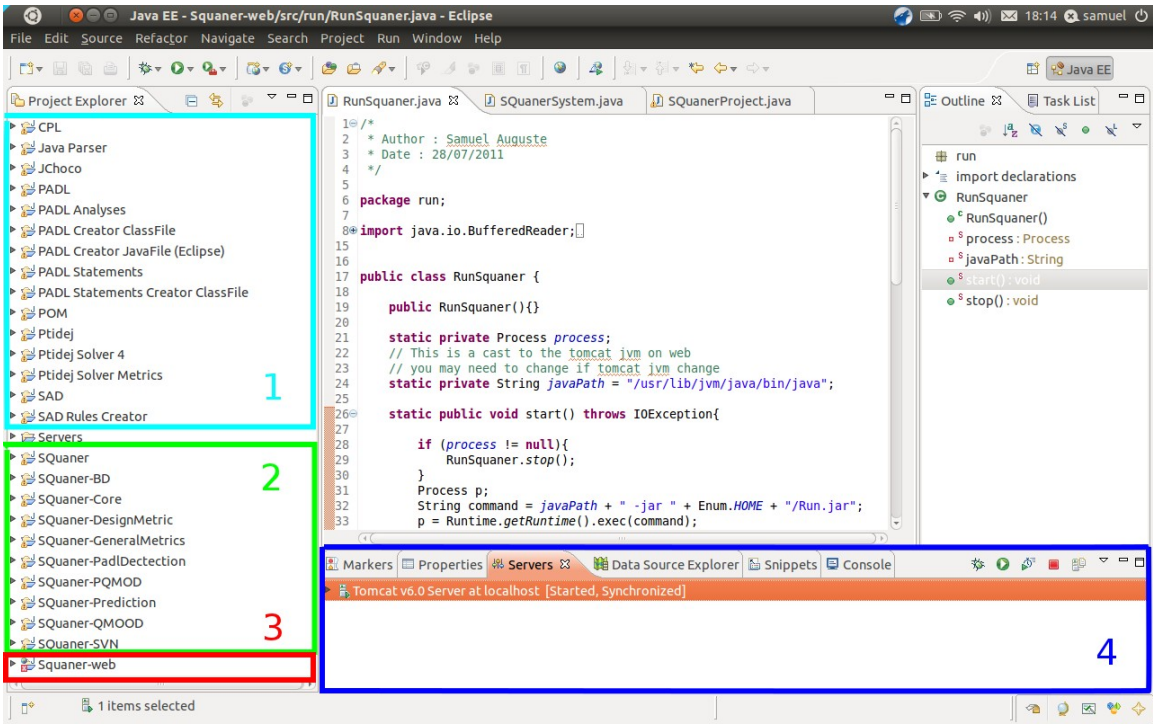


Figure 8 – Espace de travail de Squaner, ouvert sous Eclipse.

1) Ces projets correspondent à la suite d'outils Ptidej. Ils permettent donc de parcourir du code objets pour construire des modèles conformes au méta-modèle PADL [12] , puis de l'enrichir en l'annotant pour calculer des métriques, reconnaître des patrons…

2) Ce sont les projets développés pour le système Squaner. La division de son code en projets est faite suivant les différentes tâches effectuées par Squaner. La plupart appellent des fonctions de la suite d'outils Ptidej.

3) Le projet Squaner-web correspond au site web de Squaner. Codé en JSP et déployé sur un serveur Tomcat, il permet la saisie de données, la vérification des formulaires correspondants, puis l'appel des composants de Squaner interagissant avec les bases de données (pour l'ajout de nouveaux projets, configuration de l'analyse…).

4) Le serveur Tomcat est utilisé directement depuis Eclipse. Cela permet de re-déployer automatiquement le site web pour tester ses changements en temps réel.

Par la suite, il a fallu corriger de nombreux problèmes afin de pouvoir tester Squaner. La plupart concernaient une mauvaise configuration d'Eclipse avec des erreurs de *build paths*, certains projets n'acceptant que des versions spécifiques de JRE. Il y avait aussi des problèmes de compatibilités entre projets, certains étant dépendant d'autres, mais sous des versions antérieures. Enfin, le code comportait quelques chemins relatifs ne faisant pas référence à des emplacements présents sur le serveur Soccerlab.

Quelques erreurs empêchaient également Squaner de fonctionner. Par exemple, un *out.println* était présent dans le code JSP à la place d'un *out.print*, ce qui entraînait une création de la base de données incomplète lors de l'ajout d'un projet à Squaner, bloquant ainsi toute analyse. La recherche d'erreurs comme celle-ci fut une occasion d'approfondir ma maîtrise d'Eclipse, la grande taille du code nécessitant souvent d'utiliser les fonctionnalités d'Eclipse pour la recherche, l'ouverture de hiérarchie d'appels, de déclarations… Cette étape fut aussi une occasion d'utiliser les fonctions de refactoring d'Eclipse, afin d'améliorer la compréhensibilité du code. Des classes et paquetages ont été ajoutés, d'autres enlevés et des fichiers déplacés vers des emplacements plus appropriés. Certaines classes et méthodes ont aussi été renommées, parfois pour corriger des erreurs typographiques, d'autres fois pour corriger des mauvaises traductions ou des mots restés en français dans le code.

3.2 Première tentative de migration

La première étape devait être la suppression du serveur Tomcat installé par le premier développeur de Squaner et qui faisait doublon avec le serveur Tomcat principal. Pour cela, il fallait migrer l'application web Squaner-web vers le serveur Tomcat principal. La génération d'une archive War à partir du code modifié comme expliqué précédemment, puis son déploiement sur le serveur Tomcat principal ne fonctionnait pas. Cela provoquait une erreur dans le serveur Tomcat. L'interprétation de ces dernières était souvent délicate, peu d'informations sur l'origine des erreurs étant générées.

La toute première erreur était due à un problème de droits, Squaner n'ayant pas les droits nécessaires pour lire un fichier de configuration. La question des droits sur le serveur Soccerlab a été un problème courant et consommateur de temps. De nombreux allez-retours au bureau du technicien furent nécessaires pour obtenir les principaux droits souhaités. En effet, ceux-ci étaient délivrés au compte-goutte, et lors d'une demande de droits d'écriture sur un dossier, il n'y avait parfois aucun droit de délivré sur ses sous-dossiers, ou bien les droits d'écriture pouvaient être accordés mais pas ceux en lecture, ce qui empêchait toute utilisation du dossier et exigeait un nouvel aller-retour.

Après beaucoup d'essais, le projet n’avançait pas mais révélait au contraire de nouveaux problèmes. En parallèle des tests sur le serveur, j'effectuais des tests sur mon ordinateur pour m'assurer du bon fonctionnement du code et m’aperçus qu'en pratique, cette version de Squaner ne pouvait pas fonctionner sur un SVN quelconque (voir §3.5). J'ai également remarqué que les exemples présents par défaut sur la page d’accueil du serveur Tomcat principal (*JSP examples* et *servlet examples*) ne marchaient pas, et que l'installation de Tomcat sur le serveur Soccerlab n'était pas standard, ses dossiers étant parfois des raccourcis vers des dossiers installés sur des partitions différentes, pour une question de gestion de la mémoire. Au bout de plus d'une semaine de tentatives de migrations infructueuses, j'en ai donc discuté avec mon encadrant et, le problème pouvant peut-être venir du serveur Tomcat en lui-même, il a été décidé que je pouvais effectuer par la suite le développement et les tests d'intégration de Squaner à part, sur mon ordinateur personnel.

3.3 Implémentation du parseur Java

Initialement, seuls les fichiers C++ et les binaires de Java devaient pouvoir être analysés par Squaner. Cependant dans le cas d'un projet Java, lors de son ajout sur un serveur SVN, c'est le code source qui y est conservé (fichiers .java). L'équipe du laboratoire Ptidej travaillant presque exclusivement en Java, Squaner ne leur aurait donc pas permis de lancer les analyses sur leurs projets.

Le développement de la première version du parseur construisant le méta-modèle PADL à partir du code source Java ayant été terminé vers la fin de mon premier mois de stage, celui-ci pouvait être intégré au projet Squaner pour permettre l'analyse des fichiers source Java. J'ai donc ajouté au espace de travail de Squaner les projets correspondants (*Java parseur* et *PADL Creator JavaFile*) et effectué toutes les modifications nécessaires dans le code pour pouvoir appeler leurs méthodes lors de l'analyse, en fonction du type du code indiqué lors de l'ajout d'un nouveau projet.

L'analyse pouvait être lancée, mais des erreurs se produisaient durant son exécution et de nombreuses métriques n'étaient pas calculées. Cela venait du fait que le parseur n'était en réalité pas complet. Il créait le méta-modèle PADL mais ne l'annotait pas. De plus, l'intégration des projets pour le parseur Java avait demandé des versions plus récentes d'autres projets de la suite Ptidej, qui n'étaient plus compatibles avec ceux du parseur C++, celui-ci n'ayant pas été mis à jour, n'étant que très peu utilisé. Après concertation avec mon encadrant, il a été décidé de laisser uniquement l'analyse pour les projets Java et en l'état, le parseur Java pouvant être remplacé lorsque son développement aurait avancé et le parseur C++ pouvant être facilement réactivé lorsqu'il serait mis à jour.

3.4 Réécriture du site web

Le site web était l'élément le moins avancé du projet, bien que son rôle soit pourtant central puisque c'est par ce biais que l'utilisateur est censé pouvoir interagir avec Squaner, que ce soit pour ajouter des projets ou pour y lire les résultats des analyses. Un très grand nombre d'améliorations pouvaient y être apportées, mais avant de les coder, la priorité a été de corriger les nombreux bugs existants (*figures 6, 7 et 9*).

De plus, les bases de données de chaque projet possèdent une table de configuration dont les champs devaient avoir pour but initial de pouvoir configurer l'analyse (temps entre deux vérifications de la révision d'un projet, choix entre le passage à la révision suivante ou à la dernière révision …). Ces champs n'étaient malheureusement pas exploités par le site web. J'ai donc créé une page de configuration reprenant les principaux champs et permettant de les modifier (*figure 10*), bien qu'il soit encore possible d'y ajouter des fonctionnalités sur d'autres champs prévus par la table de configuration.

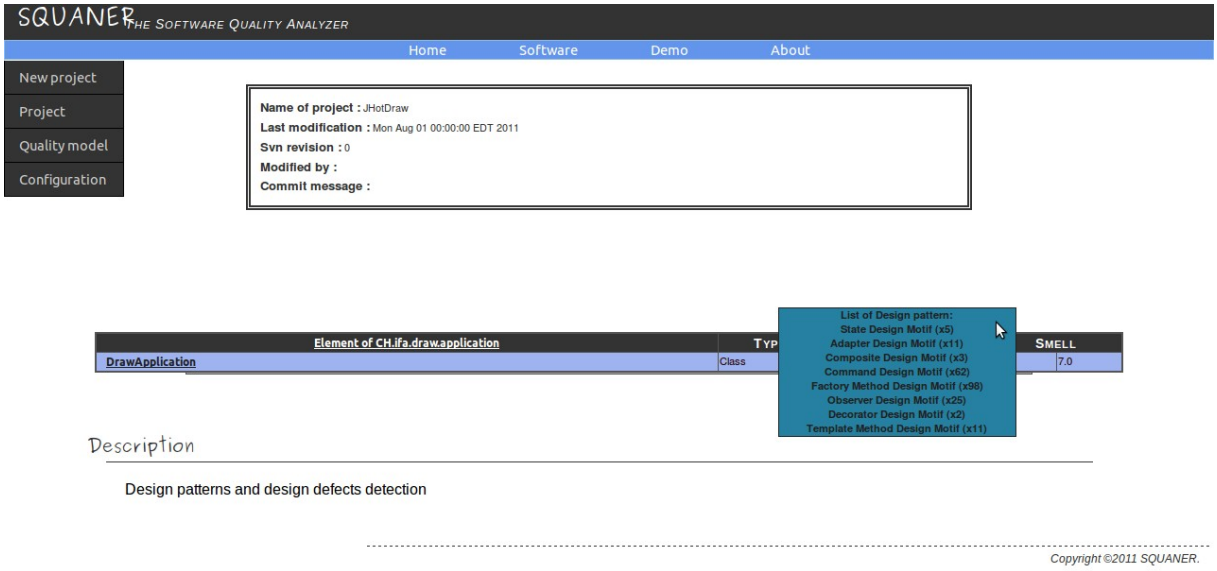


Figure 9 – Site web de Squaner, correction de l'affichage des patrons.

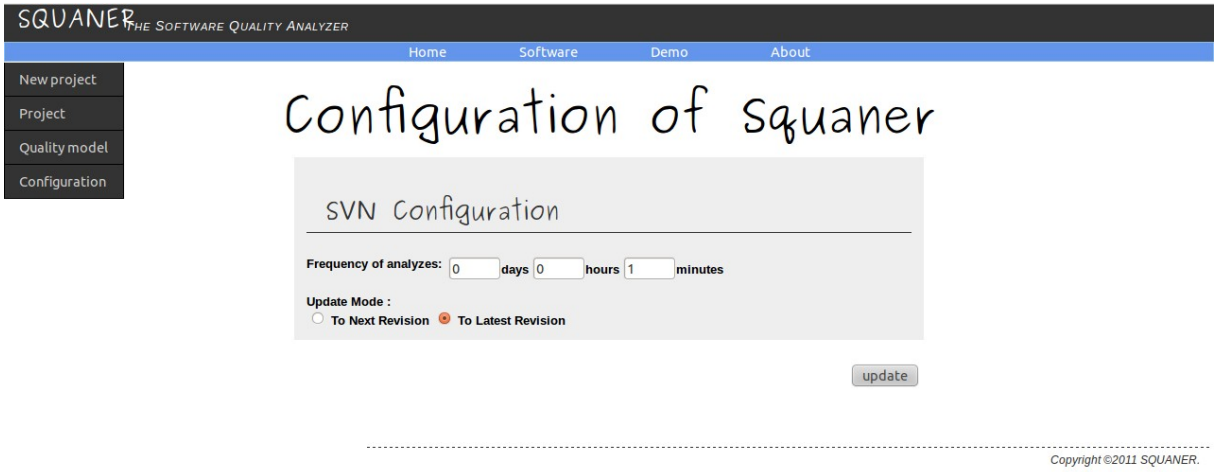


Figure 10 – Site web de Squaner, page de configuration des projets.

Une autre page a été entièrement créée, il s'agit de *run.jsp*. Cette page permet de lancer depuis Tomcat le script qui effectue automatiquement les analyses en cas de changements sur le SVN d'un des projets contrôlé pas Squaner. Nous le verrons en détails dans la section §4.2.

Enfin, beaucoup de petites retouches ont été apportées. La refonte du site web s'est faite de manière à ce que son utilisation soit plus ergonomique et cohérente. Il y a le respect d'une certaine unité et d'une charte graphique, la mise en place de nombreux messages explicatifs lorsque l'utilisateur tente de faire une action inappropriée (par exemple tenter d'aller sur la page de configuration d'un projet, sans avoir choisi de projet auparavant). Ce travail a permis de me familiariser avec l'utilisation du langage JSP et, plus particulièrement, avec l'utilisation des attributs de sessions, qui permettent d'assurer une cohérence lors du changement de pages.

3.5 Développement d'une fonction de première analyse

Le dernier gros problème de Squaner à avoir été corrigé est sa fonction *analyse()*. Cette fonction, exécutée automatiquement à intervalles réguliers, est le cœur de Squaner. En effet, c'est elle qui appelle toutes les autres fonctions, pour détecter les modifications apportées sur un SVN, puis télécharger le nouveau code, le parseur, l'analyser et, enfin, sauvegarder les résultats dans la base de données.

Initialement, cette fonction ne détectait que les nouveaux fichiers ajoutés au répertoire SVN d'un projet. Pour effectuer la première analyse, il fallait donc que le numéro de révision initiale, indiqué lors de l'ajout d'un projet, corresponde à une révision où le dossier était vide, et que le code y soit ajouté par la suite. En pratique cela est donc inutilisable, car le dossier d'un projet copié sur un SVN est rarement vide lors de son ajout et, même si cela était le cas, la plupart du temps l'utilisateur ne connaît pas le numéro de cette première révision.

Pour que le projet puisse fonctionner, une fonction *firstAnalyse()* a donc été développée et un champ *firstAnalyse* a été créé dans la base de données de Squaner pour chaque projet lui ayant été ajouté. Ainsi, le script d'analyse automatique va pouvoir savoir si une première analyse a été faite. Si cela n'est pas le cas, la fonction de première analyse est appelée. Elle permet de télécharger tous les fichiers présents sur un SVN puis de les analyser pour faire une initialisation, sans passer à la révision suivante. Ainsi, tous les répertoires SVN peuvent maintenant être analysés, quelque soit le numéro de la révision de départ, à condition bien sûr que le projet existe bien sur le SVN à la révision indiquée.

4 Intégration de Squaner

4.1 Premier déploiement

Peu avant la fin de mon stage, mon second responsable a installé un nouveau serveur Tomcat et m'a demandé d'y migrer l'application Squaner avant mon départ. J'ai donc nettoyé les bases de données et créé le dossier de configuration de Squaner sur le serveur Soccerlab, en veillant à ce que tous les utilisateurs du groupe Tomcat y aient les droits en lecture et en écriture. À partir du code source de Squaner, deux composants doivent être déployés. Le premier est une archive War, générée à partir du projet Squaner-web et qui se déploie automatiquement lorsqu'elle est placée dans le dossier des applications web du serveur Tomcat. Le second est le script permettant de lancer le processus d'analyse en cas de modification du contenu d'un SVN. Il se présente sous la forme d'un Jar exécutable nommé *Run.jar*, créé à partir de la classe possédant le *main* du processus d'analyse.

J'ai également rédigé une documentation en anglais à ce stade, à destination des personnes du laboratoire voulant utiliser Squaner. Elle recense les modifications apportées à Squaner durant mon stage, les principales évolutions qui peuvent être à envisager et des conseils pratiques pour travailler sur Squaner (voir *annexe 2*).

4.2 Problèmes dus au serveur Tomcat

Après l'ajout de ces composants qui formaient un système Squaner opérationnel sur mon ordinateur portable, des problèmes d'intégration se sont posés, les premiers étant dus à des caractéristiques du nouveau Tomcat installé sur le serveur Soccerlab.

L'exécution du script *Run.jar* fut la première difficulté. Il a fallu pour cela créer une page spécifique sur le site web (*run.jsp*), car le script doit être lancé par le groupe d'utilisateurs Tomcat pour en posséder les droits et pouvoir fonctionner. Dans un premier temps, cela ne marchait pas même avec cette page. Après des recherches effectuées avec mon responsable, il s'est avéré que plusieurs JVM étaient installées sur le serveur Soccerlab et que la version utilisée par défaut était trop ancienne pour que cela puisse permettre à Squaner d'exécuter correctement le script. J'ai donc modifié le code d'exécution du script d'analyse afin que la JVM utilisée soit la même que celle utilisée par le serveur Tomcat.

Après cette étape le script s'exécutait, mais l'analyse tournait à vide et en boucle, sans télécharger de fichiers. Toujours avec mon second encadrant de stage, nous avons donc résolu cette anomalie qui ne venait pas du code, celui-ci marchant sur mon ordinateur personnel. Après quelques vérifications, nous nous sommes aperçus que pour télécharger des fichiers depuis un SVN, Squaner utilise la librairie SVNKit [13], qui tente de créer un dossier temporaire caché intitulé *.subversion* à la racine du dossier du serveur Tomcat. Cependant, pour protéger l'état du serveur Tomcat, aucun droit d'écriture n'était accordé au groupe d'utilisateurs Tomcat sur ce dossier. Pour contourner ce problème, un dossier caché du même nom a donc été créé manuellement, et avec tous les droits nécessaires exclusivement sur celui-ci et ses sous-dossiers.

4.3 Problèmes dus au serveur MySQL

Avec l'application des changements précédents, le début de l'analyse se déroulait sans encombres, mais stoppait au bout d'un moment avec le message « *too many connections* ». La taille des projets testés sur le serveur du laboratoire et le nombre maximum de connexions autorisées par son serveur MySQL ont ainsi mis en évidence un problème de connexion aux bases de données, certaines connexions n'étant visiblement pas refermées.

La classe gérant les interactions avec la base de données s'appelle *SQuanerDB*. Elle possède un attribut de type *Statement*, une fonction de connexion et une de déconnexion. La recherche et la vérification de toutes les classes appelant ces fonctions fut longue, mais facilitée par les fonctionnalités de recherche d'Eclipse. La première correction, qui a permis d'éliminer un grand nombre d'erreurs, est la modification du code en appliquant un schéma de connexion et de déconnexion plus strict, en particulier grâce à l'utilisation systématique de la clause *finally()* pour encadrer la fonction de déconnexion. Cela n'a malheureusement pas suffit à éliminer toutes les erreurs et la nécessité de mettre en place du code pour déboguer en contrôlant chaque connexion et déconnexion s'est imposée. Les erreurs restantes venaient en fait d'une fonction qui instanciait plusieurs fois la classe *SQuanerDB* sous le même nom, avant d'utiliser la fonction de déconnexion. L'attribut de type *Statement* était alors écrasé, et ne permettait plus de retrouver les premières connexions effectuées, empêchant ainsi leur fermetures. Une fois cette erreur de conception corrigée, le projet a enfin pu fonctionner sur le serveur du laboratoire.

4.4 Bilan

Le projet Squaner a donc subit des transformations importantes et est maintenant opérationnel en interne sur le serveur Soccerlab. Il permet des analyses qualitatives précises et offre la possibilité d'effectuer des démonstrations montrant toute l'ampleur et le potentiel des outils développés par l'équipe Ptidej.

Comme indiqué dans le bilan rédigé (voir *annexe 2*), quelques modifications peuvent néanmoins encore être envisagées. Une fois les parseurs de code source Java et de C++ avancés, ceux-ci devront être mis à jour dans le système Squaner, afin d'exploiter ses capacités au maximum. De plus, la page de configuration pourrait encore être enrichie de manière intéressante, comme par exemple pour offrir à l'utilisateur le choix des métriques déterminées par Squaner, le composant Pom pouvant en calculer au total près de 60 différentes [14].

5 Tâches annexes

5.1 Summer School

Ce stage fut également pour moi l'occasion de participer à la *Canadian Summer School on Practical Analyses of Software Engineering Data* [15], qui se tenait sur presque une semaine, week-end inclus. Il m'a permis de participer à des travaux pratiques et d'assister à de nombreuses conférences de chercheurs (University of British Columbia, University of California…) et d'employés d'entreprises travaillant dans l'informatique (Benchmark consulting, SAP, Google…). J'ai trouvé très intéressant d'avoir ces témoignages, en particulier les informations concernant le rôle du département informatique dans de grandes entreprises, et les méthodes de travail des employés. De plus, cette expérience m'a permis de m'ouvrir encore plus sur de nombreux domaines du génie logiciel.

Bien que la *Summer School* ait été préparée en amont par les responsables et les étudiants du laboratoire, j'ai dans une moindre mesure pu participer à son bon déroulement avec les autres étudiants (transport de T-shirt à destination des participants, nettoyage des tables après les repas…). Cela m'a offert la possibilité de voir les étudiants dans un autre contexte, fut aussi l'occasion de beaucoup d'échanges et m'a permis d'apprendre à mieux les connaître.

5.2 Vie du laboratoire

Des réunions internes, d'une durée d'environ une heure, avaient lieu au laboratoire deux fois par semaine. La réunion du mardi après-midi était une réunion de bilan, où chaque étudiant devait présenter brièvement les évolutions de ses recherches durant la semaine écoulée. Celle du vendredi consistait en une présentation orale avec transparents d'un des étudiants à propos de ses travaux, ce qui constituait pour les autres étudiants une occasion de s'y intéresser et de poser des questions. J'ai donc participé à ces réunions et j'y ai aussi réalisé une présentation [16], en anglais.

La salle principale du laboratoire n'étant pas très grande, une bonne ambiance y régnait et l'entraide était fréquente. Des étudiants ont relu mes documentations rédigées en anglais et de mon côté j'ai aidé un étudiant à installer un double boot linux/windows sur son ordinateur portable ou encore un autre à se servir du terminal pour compiler et exécuter du code C++.

Lorsqu'un étudiant mettait en place une expérience, je me proposais rapidement comme sujet. J'ai ainsi participé à trois expériences. La première consistait à répondre à des questions sur du code comportant volontairement des anti-patrons, afin d'étudier leur impact sur la compréhension de code par les développeurs. La seconde consistait à mémoriser du code pour répondre à des questions portant sur celui-ci par la suite et utilisait un impressionnant et très intéressant dispositif d'oculométrie. Enfin, la dernière consistait à séparer les différentes unités composant des identificateurs, parfois avec l'aide de code source, pour étudier l'impact de leur syntaxe. La participation à toutes ces expériences a permis d'aider les étudiants du laboratoire en leur fournissant plus de résultats et m'a encore une fois permis de découvrir des nouveautés et de voir du code source sous un nouveau jour.

Enfin, étant passionné de photographie, je pense que mon dernier rôle notable a été la prise de photos lors des événements organisés par des membres du laboratoire, que ce soit la *Summer School* [17] ou bien tous les autres événements organisés en extérieur. Le partage des photos constituait une partie de ma contribution à la convivialité et à la bonne ambiance qui régnait au laboratoire.

Conclusion

Ce stage à l'École Polytechnique de Montréal et, plus généralement, tout mon séjour en Amérique du nord aura été très enrichissant à différents niveaux.

Avant toute chose, ce stage m'a permis de découvrir le monde de la recherche, en partageant pendant trois mois le quotidien d'étudiants en *Ph.D.* Ce fut également l'occasion de découvrir de manière plus approfondie les différentes connaissances et avancées de la recherche dans le vaste domaine du génie logiciel, ce qui me semble nécessaire pour la culture informatique de toute personne appelée à développer des logiciels.

Sur le plan professionnel, ce stage fut très instructif. D'un point de vue technique, j'ai pu approfondir mes connaissances du langage Java, de l'IDE Eclipse J2EE et me familiariser à des outils nouveaux pour moi, tels que Tomcat, MySQL Workbench, ou encore le langage JSP. Le fait d'être confronté à un milieu professionnel m'a aussi permis d'en apprendre beaucoup. Les difficultés nouvelles furent le travail au niveau de l'architecture d'un serveur, la taille conséquente du projet, ainsi que la méthode de travail utilisée : devoir mener le projet à bien même sans en connaître parfaitement la finalité, découvrir des problèmes importants en cours de développement, proposer des solutions au fur et à mesure…

Ce séjour fut également très formateur sur le plan humain. J'ai découvert le Canada et les États-Unis, ce qui était une expérience nouvelle pour moi. La ville de Montréal est très cosmopolite et animée d'un point de vue culturel. Travailler avec de nouvelles personnes est toujours bénéfique, mais l'est encore plus lorsque celles-ci viennent des quatre coins du monde (Chine, Iran, Pakistan, Cameroun…). La découverte d'autres cultures et modes de vie est très enrichissante et incite à la réflexion.

Liens et références bibliographiques

[1] http://www.umontreal.ca

[2] http://www.polymtl.ca

[3] http://www.ptidej.net/research

[4] http://www.flintbox.com/public/project/2956

[5] http://www.polymtl.ca/recherche/chercheur/gAntoniol.php

[6] http://tomcat.apache.org

[7] http://grass.fbk.eu

[8] http://www.gdal.org

[9] http://www.sdml.info/projects/srcml

[10] http://www.mysql.fr

[11] Nicolas HADERER, Foutse KHOMH, Giuliano ANTONIOL, « Squaner: A Framework for Monitoring the Quality of Software Systems », École polytechnique de Montréal, 2010. http://www-etud.iro.umontreal.ca/~foutsekh/docs/HadererSQUANER-CameraReady.pdf

[12] http://wiki.ptidej.net/doku.php?id=padl

[13] http://svnkit.com

[14] http://wiki.ptidej.net/doku.php?id=pom

[15] http://pased.soccerlab.polymtl.ca

[16] http://www.ptidej.net/teamlife/seminars/110708 - SQUANER/110708 – Presentation.pdf

[17] http://pased.soccerlab.polymtl.ca/album.php

Annexe I

État des lieux

(en anglais)

Situation Review

of Project Manager, Gutsy and Squaner

Samuel AUGUSTE

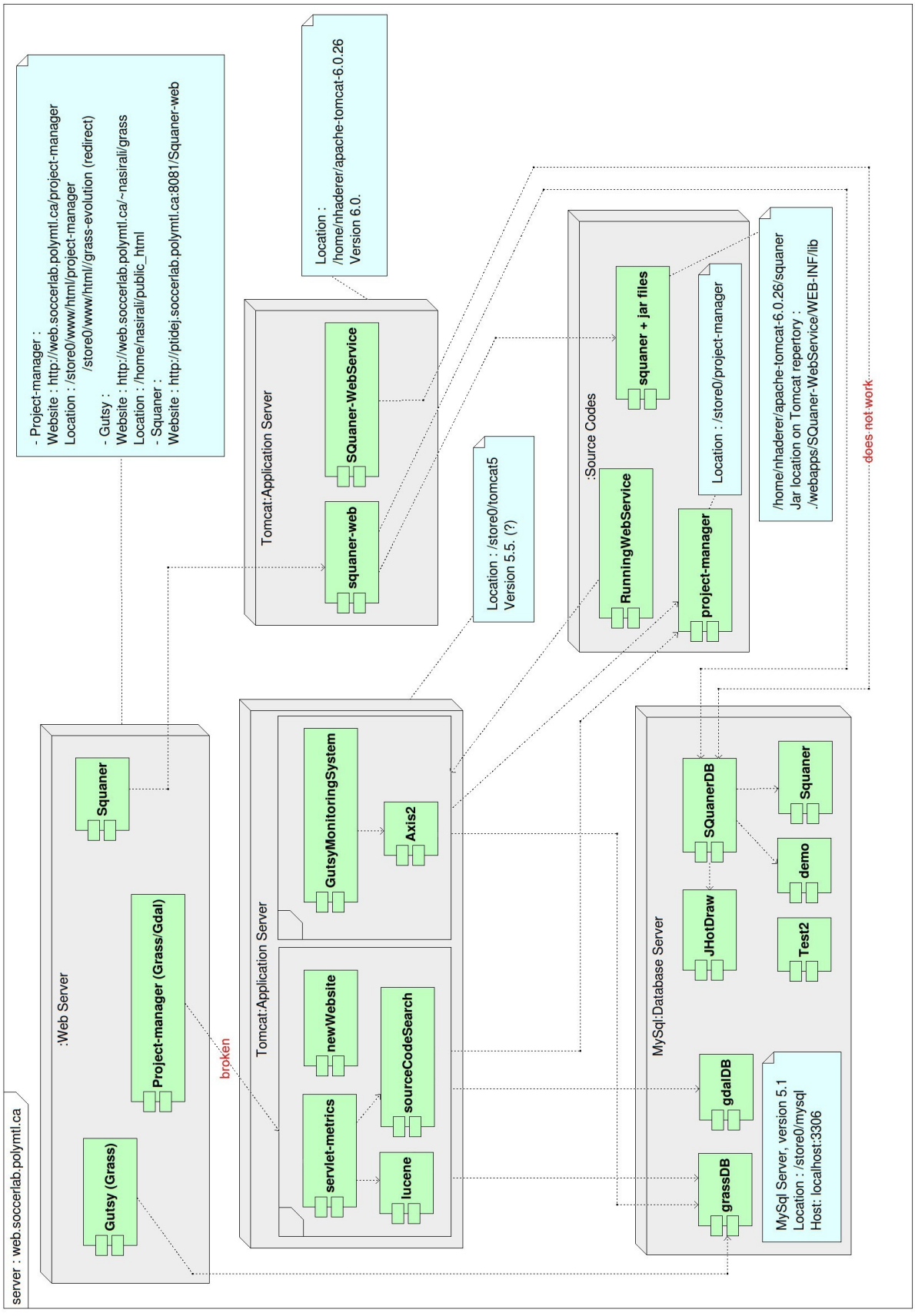
Polytechnique: Yann-Gaël GUÉHÉNEUC & Giuliano ANTONIOL
ENSICAEN: Christophe ROSENBERGER

Juin 2011

Table of contents

This document describes in details the systems for monitoring the quality of object-oriented projects available on the Soccerlab server.

1 Server Overview



As we can see, there are three systems, developed one after another, on the server: Project Manager, Gutsy, and Squaner (Software QUAlity AnalyzER).

We notice some components can be improved:
First, there is no need to have two Tomcat servers. All the web applications based on Nicolas home Tomcat should be migrated on the main Tomcat.
Second, the version of the main Tomcat server needs to be verified, because his Web site indicates a Tomcat version 6.0 whereas his folder and releases notes announce version 5.5. This maybe caused by a wrong migration. A solution could be to reinstall Tomcat server, upgrading it to the latest version.
Third, Project Manager is not working anymore. So we can also plan to remove all the components related to it.

2 Project Manager

2.1 Presentation

Project Manager uploads code from SVN repositories (grass and gdal), analyzes them, and sends an e-mail to developers, showing them the main defaults in their code. It was developed by Bruno Genna, Jeremy Watso-Cournoyer, Mathieu Denis, and Xavier Tremblay in 2008 and 2009.

Its Web site has a simple design, and is not working. On the Web site, we cannot access to the results of the analyses (see figure 2). In fact, the tree list to access all the classes/packages does not appear, because it cannot access to servlet-metrics Web application on Tomcat server.

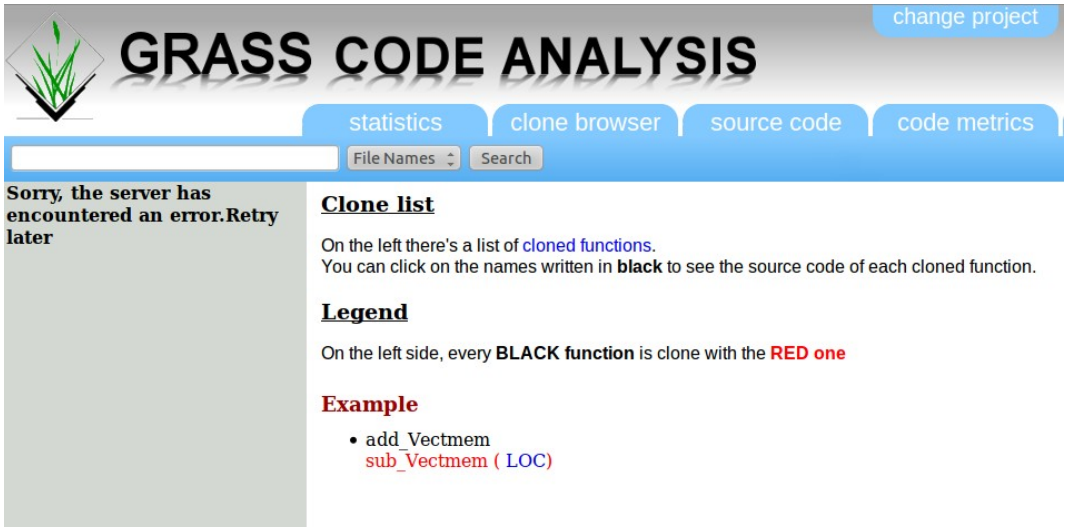


Figure 2 – Project Manager Web site.

The functions and metrics calculated by Project Manager are included in Gutsy, so we will describe them later. The main problem is that Gutsy was built on Project Manager, without deleting it. But a lot of components used by Project Manager (grassDB, source code …) have been modified by Gutsy, and Project Manager cannot use them any more.

2.2 To Do

Project Manager is not working and its functions are included in Gutsy, so Project Manager has no more reasons to exist. We suggest to delete Project Manager and the related components.

3 Gutsy

3.1 Presentation

Gutsy is based on Project Manager. It was developed by Nicolas Haderer in 2010.
Like Project Manager, Gutsy downloads a code from a SVN, analyzes it, saves the results in a database, and informs the developers. The difference is that Gutsy has smaller classes, and it corrects problems with Tomcat components.

Compared with Project Manager, Gutsy currently analyzes only Grass project, so it only uses the database grassDB. There is empty tables in, maybe remains of Project Manager, so it can surely be optimized.

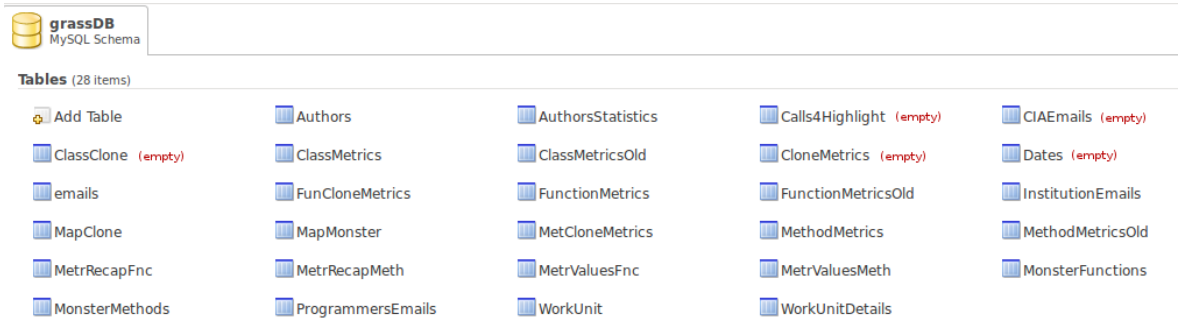


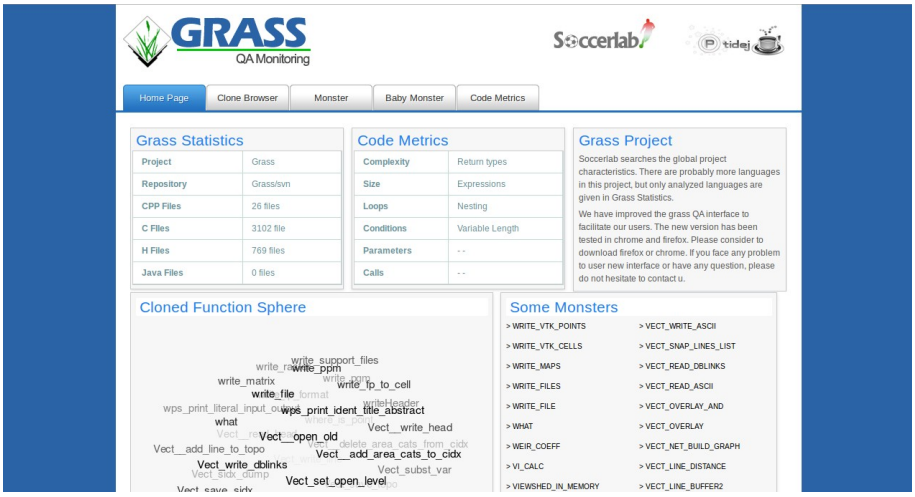
Figure 3 – grassDB tables.

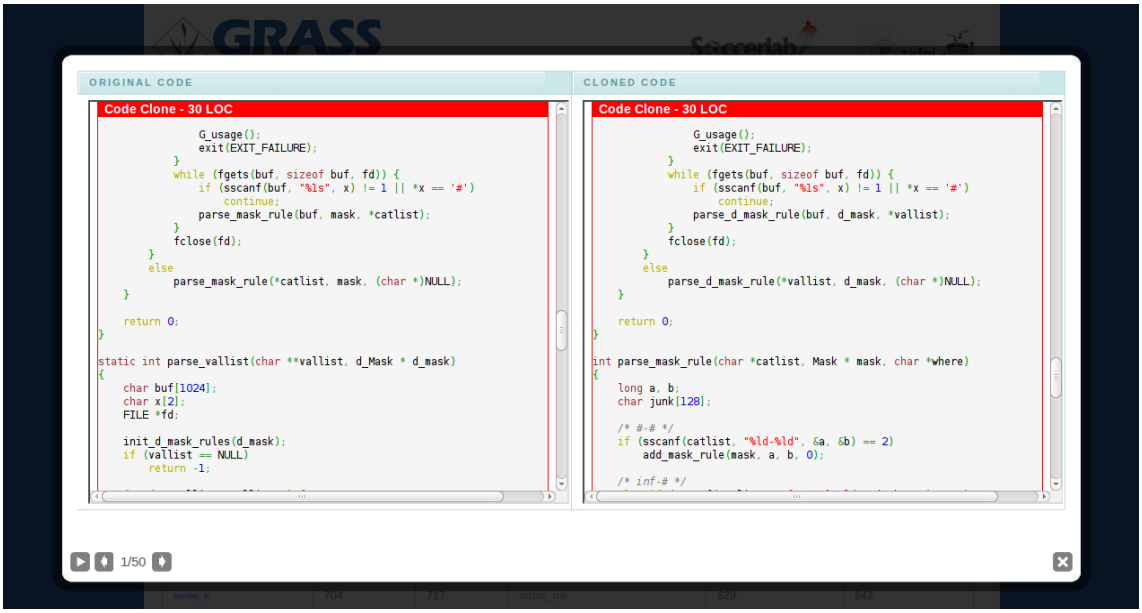
Gutsy downloads C/C++ source code from Grass SVN, and parses it with srcML to create a XML document, which contains the structure of the code. This document is stored in a database (table MethodMetric) basis elements (numbers of calls, parameters, if ...), which resume code characteristics for each functions.
Then, Gutsy calculates four code metrics for each functions: McCabe cyclomatic complexity, number of parameters, number of called functions, and number of lines of code. Those metrics are used in two analyses: Monster function analysis and Duplicate code analysis.

The Monster function analysis uses the previous code metrics to do some statistics (min, max, median … see tables MetrValues...). From this, it establishes a statistical distribution to classify all the functions into four quality levels: “Monster”, “Baby Monster”, “Warning”, and “Ok” (tables Monster...).
Duplicate code analysis uses the basis elements (number of calls, parameters, if ...) to compare functions to each others to detect similar codes (tables ...Clones).

Gusty provides Web services with Tomcat server. Users can use RunningWebService to add new projects, launch analyses … The analyses results are sent by emails to them, as Project Manager.

The Web site displaying all the results of Gutsy analyses was created after Gutsy, by Nasir Ali. It is located on Nasir homespace.
It is developed in PHP, and is independent from Gutsy services, only using grassDB. As a consequence, it cannot display the results of analyses for other projects that could be added.





Figures 4 and 5 – Gutsy Web site.

This Web site is easy to use, well-designed, and has an attractive interface (see figures 4 and 5). It is completely finished, showing all the results of the analyses, including code metrics and the basis elements obtained with the XML parse. It has few bugs, only the links in the footer being currently not working. The display of source code could be improved (code metrics do not follow the scrolling).

3.2 To Do

On the one hand, the Gutsy Web site is good for demonstrations. As long as the Squaner Web site is not finished, and after fixing its bugs, it should be used instead of the Project Manager Web site. It should also be migrated into Project Manager Web address to have a non-internal one, then replacing totally Project Manager. It needs that grassDB must also be kept.

On the other hand, we can discuss the efficiency of Gutsy on some points. First, it bases all his quality analyses only on four code metrics, sometimes quite meaningless as the number of lines of codes. Also, we can ask if the number of parameters is sufficient to define a function as a monster.
Those levels are another point of discussion. It is useful to have this information, which will show the worst functions, but it could be more relevant to measure the real quality of a function, independently to the others functions. This is one reason why Squaner was developed, so Gutsy code has less interest now.

4 Squaner

4.1 Presentation

Squaner (Software QUAlity AnalyzER) was developed by Nicolas Haderer in 2010. It analyzes the quality of a project code downloaded from SVN servers, but also controls the evolution between two SVN versions. It offers many more quality metrics than Gutsy, and also provides a Web interface. To have more information on Squaner, you can consult Nicolas Haderer's memo [1].

On the Soccerlab server, Squaner uses four databases: SQuanerDB, squaner, JHotDraw and demo. SQuanerDB is the main database, which contains the names of all the projects added for Squaner analyses. Squaner, JHotDraw and demo are the databases containing parameters and results of those analyses.
One cleaning up could be to remove Test2 database, which was probably created by Squaner but not used.

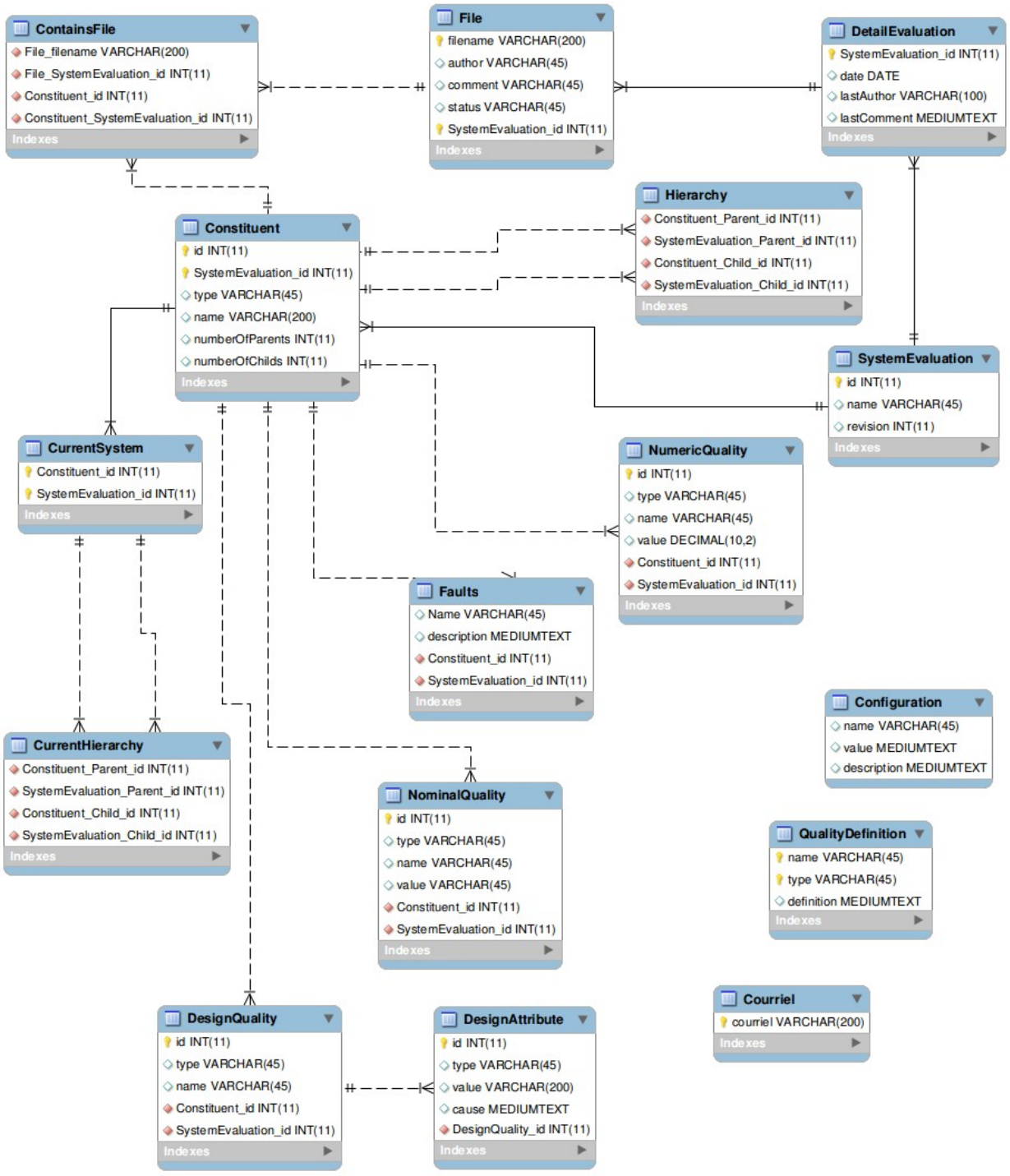


Figure 6 – A Squaner project database.

Squaner looks at the SVN repositories of all the projects referenced in SQuanerDB database and detects new files to download them. After, Squaner uses Ptidej tool suite (CPL, Ptidej, PADL, POM, JChoco, SAD...) to analyze the projects. It then creates a Squaner model:

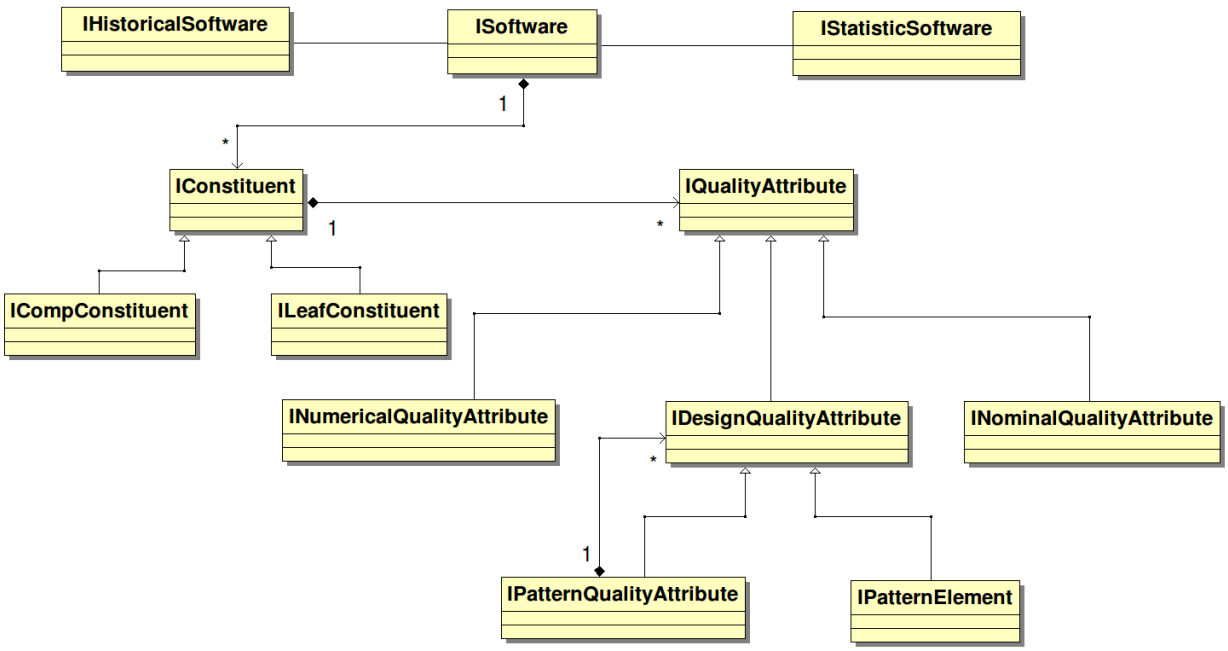


Figure 7 – Squaner model.

First, Squaner builds the PADL meta-model, using C, C++ and Java executable (.class) PADL parsers. Then, it extracts all the classes and packages to build Squaner model (IConstituent, table constituent). Next it enriches it with SAD component, creating attributes for the design motifs that it will detect (IPatternQualityAttribute, table Design Attribute), and listing all the patterns and anti-patterns (IPatternElement, table DesignQuality). It also enriches the Squaner model with seven metrics calculated by POM (CBO, LOC, LCOM5, SIX, McCabe, DIT and WMC), and introduces QMOOD model, which calculates reusability, flexibility, understandability, extendibility, functionality, and effectiveness of a class. (INumericalQualityAttribute, table NumericQuality).
After, Squaner determinates PQMOD model values, which can be bad, neutral, or good, for each following characteristics: reusability, expandability, scalability, understandability, generality, modularity at runtime, and modularity (INominalQualityAttribute, table NominalQuality). The last step is to predict if the elements have a fault in the next six months, calculating BugPredict, and HBugPredict which is supposed to be more precise being also based on the previous bugs (but we will see it actually does not work).

The values for packages are established doing the average quality values of all classes they contain. After each analysis, an e-mail in sent to the developers, and we can see all the results and the evolution compared to the previous analysis on the Web site.

On the Tomcat server, we have the SQuaner-WebService and squaner-web (see figure 1). SQuaner-WebService tried to provide a Web service to control Squaner like Gutsy, but it is not finished. Squaner-web is the interface providing control of Squaner through a Web site, developed in jsp.
Since the changes on servers, this Tomcat is not running, so we cannot access to the Web site.

The Web site is not working properly. If the number of packages is large, the access to the information will be long, a tree list should be better to access classes.
There is display bugs (see figures 8 and 9), and spaces for functionality not yet implemented (see figure 8).

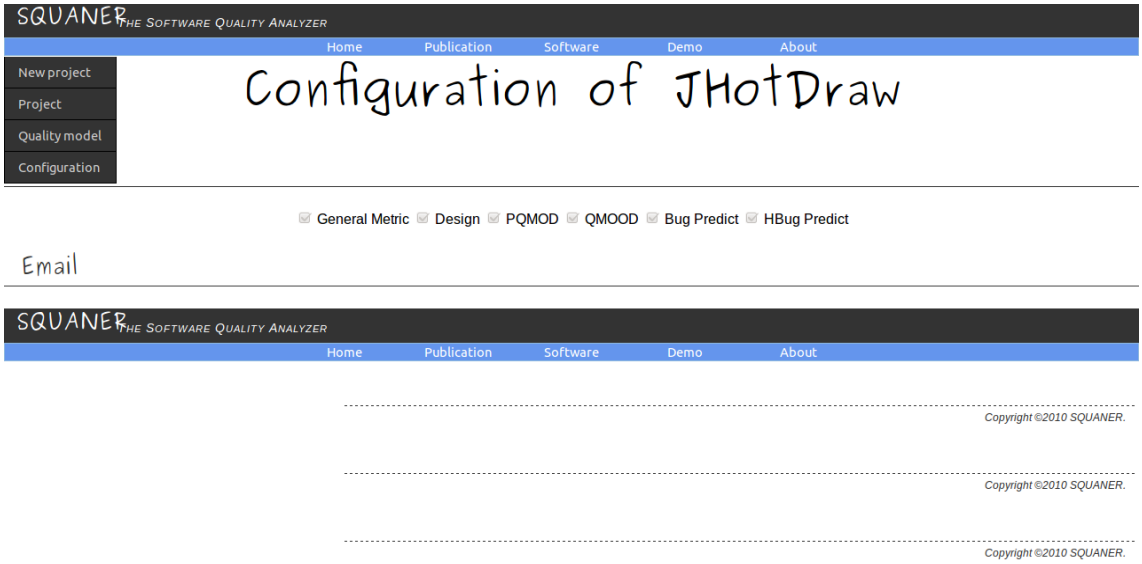


Figure 8 – Squaner Web site, configuration page.

The publication menu is a link toward the publication page of the Ptidej Web site, which has no relation with Squaner.
The page to add a project does not check all the data. We can add a project with no name, which will exist on the Web site but not in the database.

4.2 To Do

Squaner is more complete and accurate than Gutsy, including the Web site in the solution. It is a larger program, it has a clear and expandable architecture, so it is easier to add functionality on it. Squaner should replace Gutsy when it will be finished.

First, there are some problems with the analysis. Sometimes, the results are not reliable. For example, it happens that Squaner detects more patterns than number of lines of codes in some classes (see figure 9).

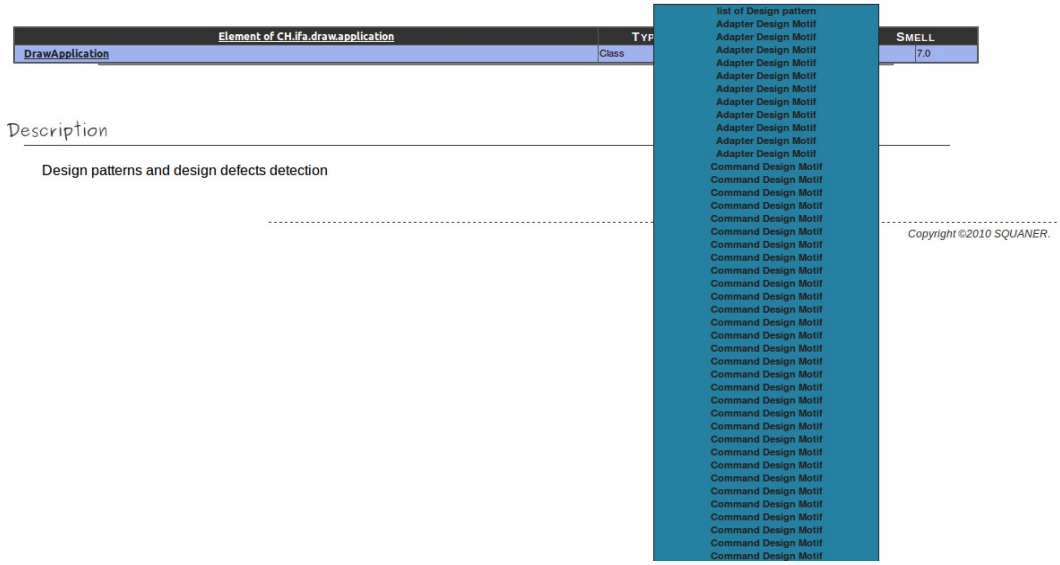


Figure 9 – Squaner Web site, display of patterns.

The results of Squaner are not always meaningful for the users. For examples, values in QMOOD have no “concrete” meaning, they only permit to monitor the evolution. QMOOD and PQMOD presents same measures, so we may merge them in one single model.

The execution time of analysis is another problem. It is generally very long, especially for the first analysis. To reduce it, we should improve Squaner. It can go with the Web site display: if we access classes through a tree list, we can stop to calculate statistics and show packages, which will save a lot of time. In return, it will be useful to have a page showing the worst classes.

The e-mail sent to the developers can also be changed. It contains the worst classes (20% of total classes) and causes for smells, but they are difficult to understand, and considerably increase the length of the mail.

When a Java project is added to a SVN, only source code is uploaded. So, to use Squaner on Java projects, we must quickly replace the .class PADL parser with the .java one.

Currently the main problem is the Web site, which does not present all functions of Squaner. This should be redone to improve the display and to make the projects configuration possible. According to the analyses and the databases (table Configuration), the user should be able to choose the analyses that he wants (more than 60 metrics with POM), the time or the number of revisions between two analyses …
The user cannot report bugs through the Web site, so HBugPredict model cannot correctly work, and Faults table stays empty in the database.
To launch an analysis from the Web site is also very helpful to use the system.

SQuaner-WebService is a not finished project, to manage Squaner services. As Squaner has a web interface doing it, we can also remove SQuaner-Webservice project.

For Squaner to work, we should also reinstall Squaner on the main Tomcat, and make sure that it has the rights to create databases. If this is not possible, the other solution might be to create a ready-for-delivery version of Squaner, accessible to users for install and use on theirs servers. An advantage is that they can use Squaner even if they do not want to share their code.

Appendix

[1] Nicolas Haderer, Mémoire de stage M2 : Software QUality ANalysER, 2010, 64 p.
https://web.soccerlab.polymtl.ca/rptidej/ptidejlab/Software/Students Workspace/SQUANER/Nicolas Haderer memo.pdf

Annexe II

Bilan

(en anglais)

ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

SOCCER LAB
SOftware Cost-effective Change and Evolution Research Lab

Pattern Trace Identification, Detection, and Enhancement in Java

Squaner changes report

Key information to use Squaner code

Samuel AUGUSTE

Polytechnique: Yann-Gaël GUÉHÉNEUC & Giuliano ANTONIOL
ENSICAEN: Christophe ROSENBERGER

Juin 2011

Table of contents

Squaner (Software QUAlity AnalyzER) is a quality code monitoring system. This software detects changes on SVN server, downloads code and analyzes its quality with PTIDEJ tools, also comparing the evolution of the quality between two SVN revisions. Results can be seen, and projects added through a web interface.

1 How to use Squaner code.

In the following document, the term Squaner SVN will refers to the location:
https://web.soccerlab.polymtl.ca/rptidej/ptidejlab/Software/Students Workspace/SQUANER

1.1 Needed folder.

To install Squaner, you need a folder which can be download on Squaner SVN as “squaner” folder. On soccerlab server, it is located on /data/squaner.

This folder contains configuration files to install new projects. In sub-folder project, a folder will be create for each project, containing its downloaded code.

1.2 Databases.

The databases are suppose to be correctly configured on soccerlab server. If you want to check or use on another machine, you must create a database called SQuanerDB with the file SQuanerDB.sql on Squaner SVN. This database references all projects added to Squaner (their names are the columns values in the Configuration table). Squaner also creates databases named after each project added to Squaner, to save the results of theirs analysis. So you should make sure that the file configuration.ini on squaner folder contains identifiers with the rights to create new databases.

1.3 Using the code.

Squaner code can be found in the folder workspace on Squaner SVN. This workspace contains a lot of projects, some are relating directly to Squaner, others to Ptidej tool suite.

To adapt the code to your working environment, you must verify that the variable HOME in squaner.configuration.Enum.java (into Squaner project) corresponds to the absolute path of your squaner folder. Also, make sure the variable javaPath in run.RunSquaner.java (into Squaner-web project) corresponds to your java version.

Of course, you can modify code. To export a version of Squaner, you must proceed in two times. First, you need to export a runnable jar from Squaner project using the class squaner.monitoring.synchronize.SQuanerSystem.java (Run.jar in squaner folder). This is the script used to analyze projects. Then, you must export a war file from your Squaner-web project. This will be used by your tomcat server.

If you want to remove a project added to Squaner, delete its folder in squaner one. Also, delete its database and in Configuration table of SQuanerDB database, remove the line where value correspond to your project.

1.4 Tomcat server.

Squaner needs a Tomcat Server to run the web interface. Actual Tomcat server on soccerlab server is located at /var/lib/tomcat6. Before replacing current war file with a new one on tomcat webapps sub-folder, it's easier to test your code using another tomcat server and Eclipse tomcat plug-in.

Once the war is deployed on your tomcat server, you can access a web page at a location like: http://<your server location>/Squaner-web. For soccerlab server, the server location is: ptidej.soccerlab.polymtl.ca:8080.

In order to add a new project for Squaner analysis, you have a dedicated page with a form. Submitting a new project with this page can take a while, due to the creation of databases, so you should be patient.

An important page is http://<your server location>/Squaner-we/run.jsp. This allows to start/stop the analysis by launching the executable jar Run.jsp. When you click on the button, there is no need to stay on this page.

2 Improvements made on Squaner.

2.1 Code refactoring.

Changes were done to improve readability of Squaner code. Some packages/classes were added, other removed, and some files moved to more appropriated locations. Some classes were also renamed, sometimes to correct typing mistakes (TableDesing instead of TableDesign for example), and sometimes to correct inappropriate English translation (courriel → email, requete → query …).
Also, few errors were corrected in the code as the use of println instead of print in jsp files, which was preventing the creation of needed fields in Squaner database.

2.2 Java parser integration.

Before, Squaner could only analyzes .class files. As the files added to an SVN repository for normal java projects are .java files, an important point was to integrate the java source code parser to Squaner. This was done by adding a few classes and projects related to the parser (Java Parser, PADL Creator JavaFile (Eclipse), ...). Squaner can now analyzes all java projects. But to add this parser, an upgrade of Ptidej projects was necessary, and newest ones are not compatible with the current version of C++ parser. So this parser is currently deactivated, but the corresponding classes have been kept, and could eventually be used with a compatible version of C++ parser.

2.3 Rewriting website code.

As shown in the situation review, Squaner web site had display bugs and dysfunctions. First, all the display bugs were corrected.

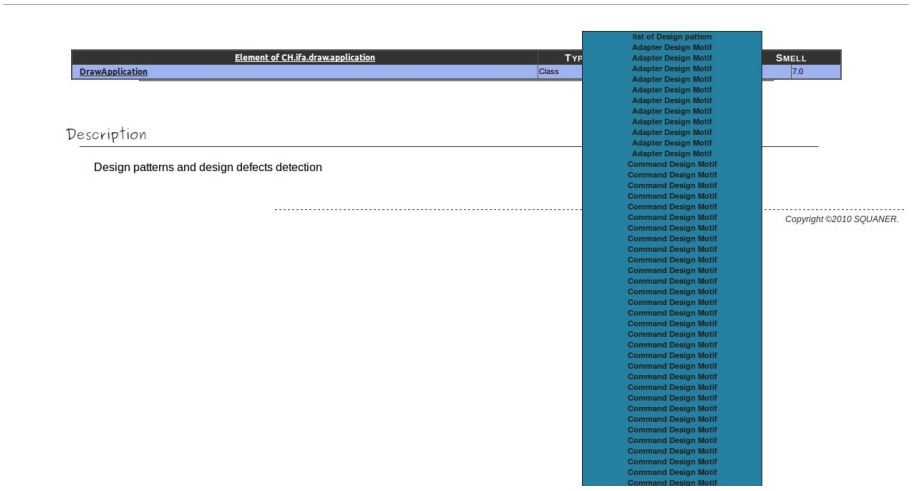


Figure 1 – Before, it can't display all the design patterns it detects.

Figure 2 – After.

Also, the interface is now more user friendly, displaying messages corresponding to user actions.

Forms were reinforced to allow a better control over the users' inputs (before, the project null was accepted).

Finally, new pages were created. The run.jsp page to launch analysis, and configuration.jsp to choose parameters to use for Squaner analysis, which are stored into each project database configuration table.

2.4 Improvement of analyze function.

At first, when you submitted a new project to the older Squaner version, you had to use an empty folder as a starting revision, and add code in future revisions. In fact, this was unusable, because when you add a folder to an SVN repository, most of the time it is not an empty folder, and even so, you can not always remember the starting revision number.

To solve this problem, a firstAnalyse function was created, and a firstAnalyze field added into configuration table of SQuanerDB database. If it is the first analysis of a project, and the revision number does not correspond to a revision with an empty folder on the SVN repository, a function for a first analysis is launch, and will analyze all the files to do an initialization.

So now, all SVN repositories can be analyze regardless of the starting revision number, thanks to this function.

3 Ideas for future changes.

3.1 Update parsers

The java source code parser used is not finished yet, and the C++ parser was deactivated. So it will be a great thing to update the corresponding projects as soon as newer versions are released.

3.2 Change databases names.

All the databases created by Squaner are named after the associated projects. So if you do not know they are related to Squaner, it can be confusing. An idea could be to include the fact they are related to Squaner in the databases names.

3.3 Clean emails sent.

The emails sent by Squaner after analysis are sometimes very long, with meaningless values for the user. Clean this content would be appreciated.

3.4 Manage users connections.

Creating a system with login and password would be welcomed. Currently all the users can see all the projects analyzed by Squaner. This is a problem if there is an important number of users adding projects.

Also, to have a system of authentication could be used to access correctly the run.jsp page trough a link, instead of be forced to write it into the navigation bar.

3.5 Improve configuration page.

The configuration page is useful to configure Squaner's analysis. All fields correspond to values in the table configuration of each project database. However, too few fields are currently used. It would be helpful if more fields could be changed from this page (like metrics calculated, …).

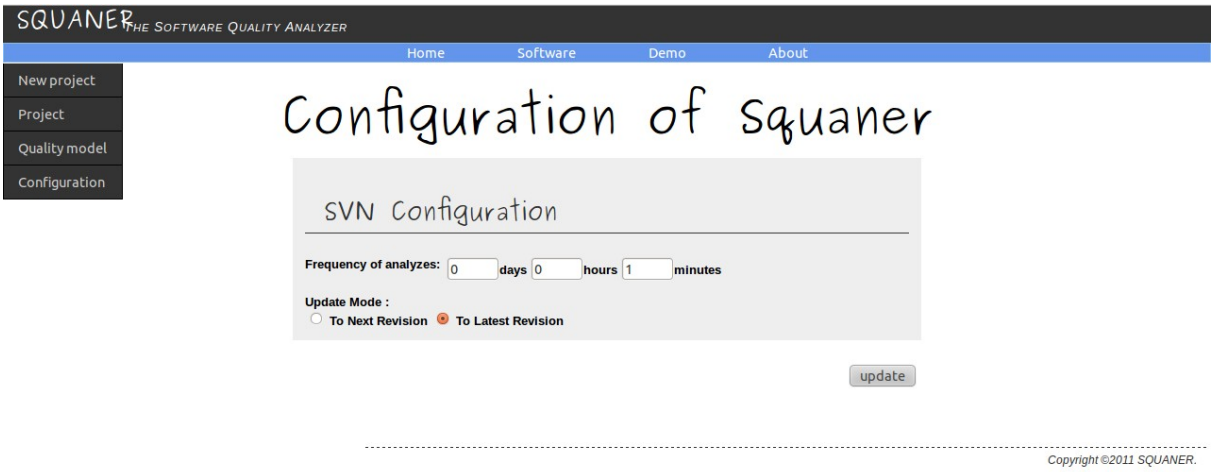


Figure 3 – Configuration page for Squaner project.

3.6 Optimize the code.

The source code of Squaner is not always clearly object-oriented, and there may remain mistakes in conception or use. It is a good idea to verify it, and try to improve analysis execution time, which is a crucial point.

Of course, this list is not exhaustive. If you plan to work on Squaner, don't hesitate to contact me at : samuel.auguste@ecole.ensicaen.fr !

Annexe III

Fiche d'appréciation

ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

Prière de retourner ce document avant le vendredi 19 août 2011 par courriel à :
cecile.huet@ensicaen.fr

Date :

FICHE D'APPRECIATION DE STAGIAIRE ELEVE INGENIEUR DE 2e ANNEE (1ère année Master)
Souhaitez-vous que cette fiche d'appréciation soit confidentielle OUI ☐ NON ■

Nom : **AUGUSTE.** Prénom **: Samuel.**

Lieu de stage : **École Polytechnique de Montréal** Dates de stage : **du 2 mai au 29 juillet.**

Nom et qualité du maître de stage : **Yann-Gaël GUÉHÉNEUC.**
Nom du tuteur école : **Christophe ROSENBERGER.**
Sujet de stage : **Nettoyage, amélioration, et remise en ligne de l'outil SQUANER.**
……..

ANALYSE DU COMPORTEMENT				
Facteurs d'appréciation	**Degrés d'appréciation**			
Présentation	**Excellente**	Bonne	Moyenne	Négligée
Conscience professionnelle	**Ne ménage pas son temps**	Participe	Se contente de l'indispensable	Absentéisme, Mauvaise volonté
Sociabilité	Animateur	**Participe**	Suit le mouvement	S'intègre difficilement
Dynamisme, ténacité au travail	**Tenace**	Actif	Moyen	Abandonne
Clarté d'expression	**Claire**	Convenable	Confuse	Incompréhensible
Attitude vis à vis du personnel	**Très appréciée**	Appréciée	Laisse indifférente	Difficilement admise
APTITUDES INTELLECTUELLES ET PROFESSIONNELLES				
Efficacité dans le travail	**Excellente**	Bonne	Moyenne	Insuffisante
Méthode - Organisation	**Très autonome Ne demande conseil qu'à bon escient**	A besoin d'un appui dans les phases clé de son travail	Sollicite un peu trop souvent de l'aide pour progresser	Trop dépendant
Connaissances techniques	**Excellentes**	Bonnes	Moyennes	Insuffisantes
Imagination pratique	**Propose des solutions**	Sort de la routine	Se contente d'appliquer ses connaissances théoriques	Aucun effort
Si vous avez eu l'occasion de tester les connaissances linguistiques en ANGLAIS				
Expression orale	Excellente	**Bonne**	Moyenne	Insuffisante
Expression écrite	**Excellente**	Bonne	Moyenne	Insuffisante
Aptitudes rédactionnelles – Rapport écrit				
Rédaction fond	**Excellente**	Bonne	Moyenne	Insuffisante
Rédaction forme	**Excellente**	Bonne	Moyenne	Insuffisante

OPINION GENERALE SUR LE STAGIAIRE :

Samuel s'est très vite intégré au laboratoire de recherche et à son nouvel environnement canadien. Il a très vite pris sont sujet de stage en main et à tout de suite proposé une méthodologie pour obtenir rapidement des résultats. Je n'hésiterais pas à la reprendre en stage (ou autre). Merci !

Cachet et Signature

Envisagez-vous de prendre un stagiaire de l'ENSICAEN en 2012 ? OUI