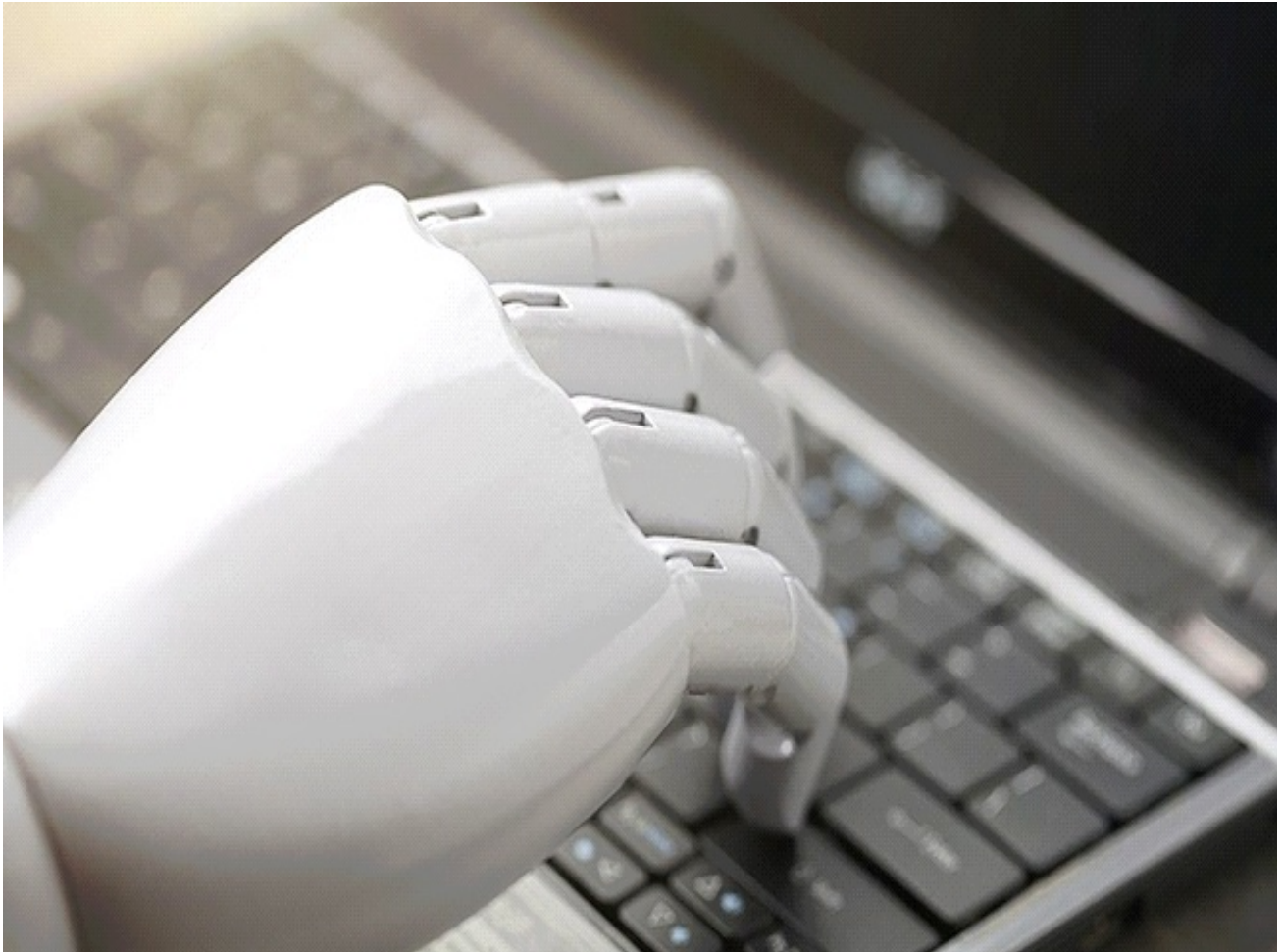


RAPPORT D'ALTERNANCE 2018-2019

Automatisation des lignes de produits logiciels en entreprise



10 mai 2019
THOMAS PETIT

thomaspetit23131@gmail.com

Table des matières

INTRODUCTION.....	8
L'informatique.....	8
Les entreprises face aux défis de la réutilisation.....	12
Les lignes de produits logiciels.....	13
L'alternance.....	17
Le plan du mémoire d'alternance.....	19
LE BUT.....	20
La réduction des coûts de développement	21
Réduction du délai de mise sur le marché des logiciels.....	21
Réduction des efforts de maintenabilité des logiciels.....	21
Amélioration de la qualité des logiciels.....	22
Etat de l'art et prérequis	23
La ligne de produit logiciel	23
Ingénierie du domaine et ingénierie d'application.....	25
Analyse du domaine.....	26
Conception du domaine.....	26
Réalisation du domaine.....	26
Ingénierie d'application.....	26
Configuration des produits.....	26
.....	27
Génération des produits	27
Les approches de conceptions traditionnelles	28
Le développement from scratch	28
Le clone and own.....	28
Le répertoire commun.....	28
Les approches fondamentales pour les LPL.....	29
L'approche Proactive	29
L'approche Extractive.....	33
L'approche Réactive.....	34
Quelle approche adaptée aux PME ?.....	37
L'analyse Formelle	39
La variabilité d'une Ligne de produit logiciel	43
La gestion de la variabilité	43
Caractéristique logicielle.....	43
Modèle des caractéristiques.....	44
L'implémentation LPL par annotation et par composition.....	45
Annotation	45
L'implémentation en compositionnel.....	47
Le degré de granularité	49
Localisation des caractéristiques.....	49
Résumé LPL.....	52
SPL generator.....	53

Les objectifs de l'approche.....	54
Mon alternance.....	55
Les étapes de L'intégration.....	56
Identification des assets et l'entrée des produits.....	56
Problématiques lors de mon alternance	57
L'entrée des produits	57
L'entrée des caractéristiques du produit.....	58
L'identification des assets du code source du produit.....	58
L'unicité des Identifiants	59
Acquisition des artefacts.....	60
Les modèles de variabilité et leur extraction.....	66
Qu'est-ce qu'un modèle de variabilité ?.....	66
Comment peut t on extraire automatiquement la variabilité SPL et ses contraintes ?.....	69
FCA pour la variabilité.....	69
Le calcul de la variabilité dans notre Outils SLP generator.....	71
Création du contexte formel (PCM) à partir des assets.....	72
Extraction de la variabilité à partir du Treillis.....	73
La localisation des caractéristiques.....	75
L'analyse de concepts formels (ACF) pour réaliser la localisation.	75
Conclusion et résultats préliminaires.....	77

Pensées et remerciements envers mes parents pour m'avoir soutenu tout au long de mes études.
Merci à Laurent pour ses conseils avisés tout au long de l'année 2018-2019.

Mon alternance avait pour but dans un premier temps de participer à la maintenance d'une Ligne de Produits Logiciels (LPL) en tant qu'ingénieur informaticien, puis dans un second temps, d'automatiser les processus de création de LPL afin d'en faciliter l'adoption pour des entreprises clientes du groupe Accenture.

Les Lignes de Produits Logiciels ont pour but d'industrialiser la fabrication de logiciels partageant des caractéristiques communes afin d'en diminuer le temps et les coûts liés au développement et à leur maintenance des produits logiciels. Les LPL permettent une réutilisation de caractéristiques et une personnalisation des produits logiciels. Malheureusement, l'adoption des LPL a un coût d'investissement conséquent pour les entreprises, et cela a fait souvent défaut au déploiement des LPL dans les entreprises de petite taille.

INTRODUCTION

L'informatique

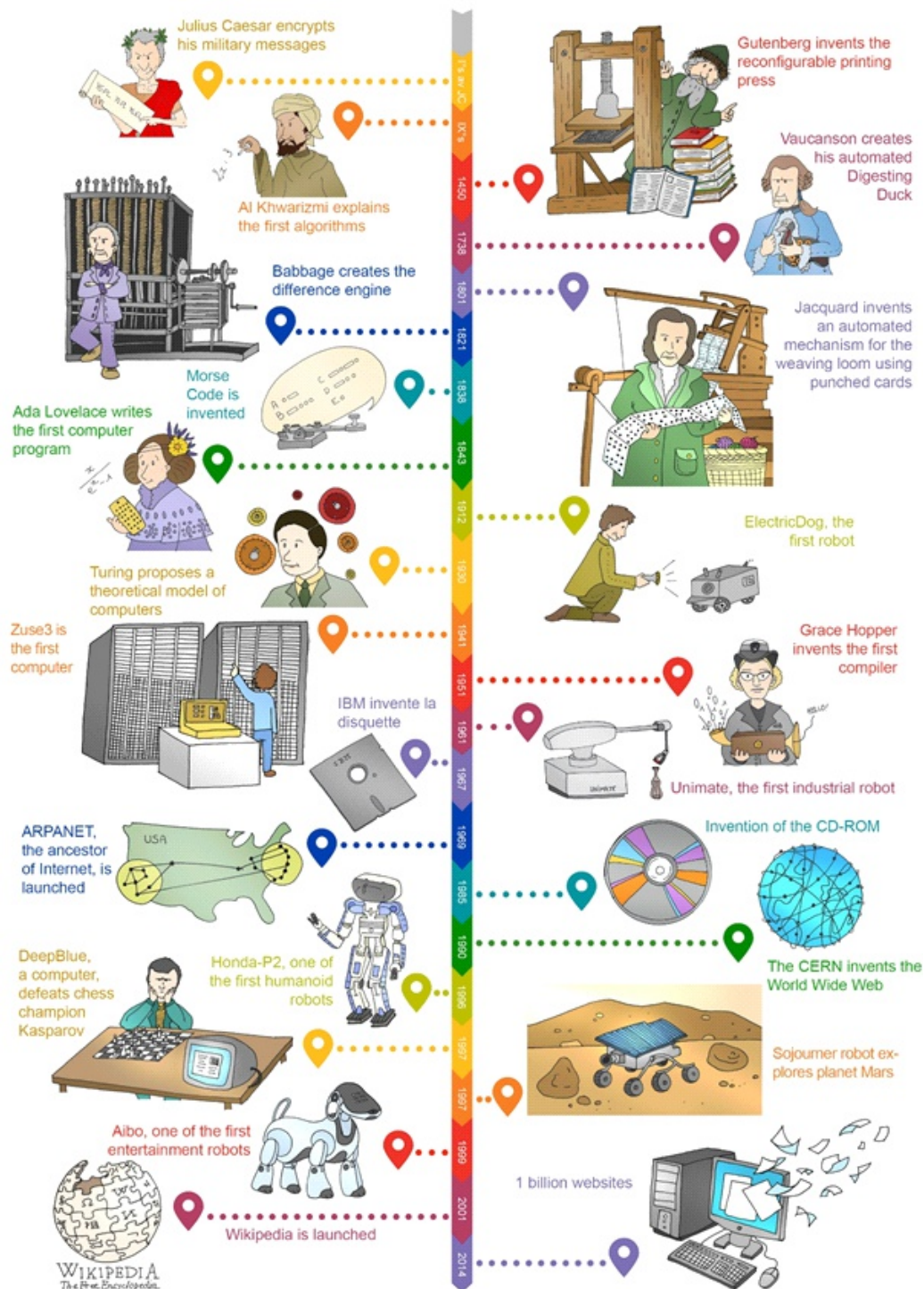
L'essor de l'informatique au début du XXème siècle, puis d'internet à la fin du XXème siècle ont révolutionné notre quotidien et notre façon de vivre. En France, La transformation numérique a fait basculer les métiers de l'industrie vers le tertiaire et les services faisant accroître de manière exponentielle le développement d'outils informatique. L'informatique, puis internet ont largement contribué à cette révolution. Cette digitalisation a permis entre autres d'automatiser des tâches par des algorithmes en passant par des applications et logiciels.

Aujourd'hui, l'informatique occupe une place immense dans notre quotidien. Les nouvelles technologies évoluent constamment chaque jour, cherchant toujours à automatiser ces processus de la vie courante. La demande croissante de nouveaux logiciels oblige les fabricants à accélérer le processus développement ainsi que des logiciels à plus grande échelle.

Les programmes sont constitués de code et de données. Le code est la série d'instructions qui indiquent à l'ordinateur la tâche à accomplir ensuite et la manière dont il doit hiérarchiser les tâches. Les programmeurs utilisent ces instructions pour ordonner à l'ordinateur d'exécuter des tâches spécifiques, comme l'enregistrement de fichiers ou l'organisation d'images sur votre bureau ; ce sont les programmes, qui doivent être écrits dans un ou plusieurs langages de programmation. Ces logiciels ont pour but d'automatiser des tâches souvent avec une interface graphique permettant à l'utilisateur d'interagir avec le logiciel.

Un logiciel, parfois appelé programme, est un ensemble d'instructions qui indiquent à l'ordinateur ce qu'il doit faire. Ce type de programme peut être écrit dans un ou plusieurs langages.

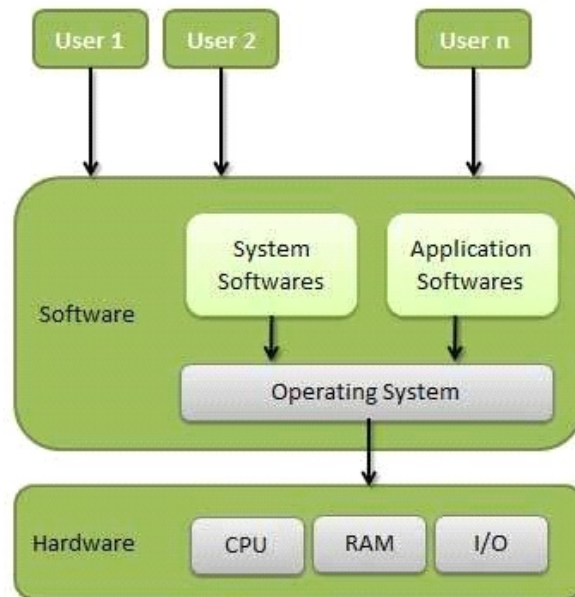




Dans le monde de l'entreprise la production de ces logiciels n'est pas adaptée pour les productions de masse, en effet chaque entreprise développe son propre logiciel sans possibilité de réutiliser du code déjà existant.

Afin de maintenir un équilibre entre vitesse de production et qualité les entreprises ont dû s'adapter et évoluer. L'émergence du génie logiciel est due à la nécessité de trouver des méthodes de

conception, des outils et des paradigmes qui répondent aux besoins des fabricants, ainsi qu'aux besoins des informaticiens généralistes dans le but de fournir une aide aux développements des logiciels.



Les entreprises face aux défis de la réutilisation

De plus les entreprises se confrontent à des problèmes de maintenances et d'évolutions des logiciels qui entraînent des pertes de temps et donc d'argent. En effet la maintenance et le support des applications, sont très chronophages pour les développeurs, qui sont parfois contraints de se relayer pour pouvoir assurer ce service en continue.

Les développeurs dans les entreprises ne maîtrisent pas forcément l'historique de l'ensemble des modifications et des améliorations acquises à travers le temps. Ils doivent parfois ainsi se plonger dans le logiciel parfois mal implémenté avant de pouvoir y apporter une modification. Ceci entraîne une perte de temps considérable.

La construction des logiciels est relativement statique au départ, une fois celui-ci développé et livré au client, toute maintenance requise et envisagée pour ces systèmes nécessite des modifications potentiellement importantes du code source existant. Ceci n'est plus acceptable pour les systèmes logiciels contemporains, car la complexité de la production et de la maintenance génère souvent des coûts qu'il faut prendre en compte.

La réutilisation du code source est donc un point important qui peut aider les entreprises à accélérer le processus de développement maximiser la réutilisation dans un ensemble de produits logiciels créera un ensemble de Le logiciel est conçu autour de fonctionnalités communes. Pour les fabricants, cela offre des possibilités de conception Ce n'est plus un produit séparé, mais une série de produits. Ce changement L'échelle (passage d'un produit unique à une série de produits) leur permettra de produire plus L'important est que la quantité de logiciels augmente, mais cela améliore également la qualité et réduit les coûts. Les lignes de produits logiciels sont un outil permettant de répondre à ces problématiques.

Les lignes de produits logiciels

La « recherche » sur l'ingénierie des lignes de produits logiciels s'est avérée utile pour l'industrie notamment les petites et moyennes entreprises (PME). Mais elle reste largement peu utilisée dans les entreprises. L'objectif envisagé des LPL n'est plus d'obtenir un système logiciel unique, mais de partir de bases et caractéristiques communes pour développer et affiner les fonctionnalités du logiciel. En somme faire série de produits logiciels avec des caractéristiques communes. Mais garde cet ensemble. Les logiciels peuvent développer de manière laborieuse, et peuvent également engendrer des coûts considérables pour les entreprises.

L'ingénierie des lignes de produits logiciels a rencontré de nombreux succès ces dernières années. Cependant, il est rapide de constater qu'encore peu d'entreprises du logiciel informatique (tous secteurs confondus) ont adopté le développement de LPL.

Ce phénomène peut être expliqué par au moins plusieurs facteurs : la difficulté d'appliquer les trois approches d'adoption existantes au contexte des PME couplée à une réticence à l'adoption de pratiques à risques dans l'entreprise.

La réutilisation logiciel peut jouer un rôle crucial dans cette diminution du temps de développement. Intuitivement, nous observons chez les développeurs une tendance à la réutilisation, qui prend la forme de copié collé successif, depuis des logiciels déjà conçus vers ceux en cours de conception. À terme, cette pratique nommée Clone&Own en anglais, fait émerger des familles de logiciels, ce sont des logiciels qui partagent des caractéristiques communes. Cette pratique est toutefois chaotique, et peut parfois être plus coûteuse que de commencer le projet de zéro. De plus, elle fait émerger des familles de logiciels dont la maintenance doit être assurée de façon individuelle. C'est pourquoi il est préférable pour les entreprises de considérer l'utilisation de L'ingénierie des Lignes de Produits Logiciels (LPL) comme paradigme de développement pour leurs logiciels. L'objectif est de rendre la réutilisation systémique : les logiciels sont conçus par la réutilisation et pour la réutilisation.

La conception d'une famille de logiciels partageant des caractéristiques communes. Pour cela, les lignes de produits logiciels reposent sur la réutilisation et la configuration en masse. De façon générale, l'adoption des LPL dans le processus de développement permet : une diminution des temps de développement ; une augmentation de la qualité des logiciels ; et une amélioration de la maintenabilité des logiciels.

Concevoir une série de produits à partir de zéro peut être compliquée, il y a de nombreux paramètres à prendre en compte comme le domaine d'activité, le domaine d'expertise du développeur, la technologie utilisée.

La demande de nouveaux logiciels a suivi l'augmentation de la popularité des appareils informatiques tels que les ordinateurs personnels, les smartphones, les voitures, appareils. Et avec cette augmentation de la demande, les logiciels sont devenus plus complexes à concevoir, nécessitant de plus en plus de temps et d'efforts pour les développeurs de plus en plus de temps et d'efforts de la part des développeurs.

Le domaine du génie logiciel s'intéresse à la recherche de méthodologies et des techniques pour faciliter le développement d'un système logiciel. Par exemple, ces méthodes concernent l'amélioration de la qualité du logiciel, son accessibilité à la maintenance et à l'évolution dans le temps, ainsi qu'à la description de nouvelles approches pour faciliter son développement. Parmi ces

méthodes, l'ingénierie des lignes de produits logiciels se concentre sur la conception de familles de logiciels de manière systématique. Une famille de logiciels est un ensemble de logiciels qui partagent plusieurs fonctionnalités ou aspects logiciels en commun. Le développement d'un logiciel standard permet de proposer une même solution globale pour l'ensemble des clients. Dans l'industrie, chaque utilisateur client accède à un logiciel contenant l'ensemble des fonctionnalités logicielles développées. Pour concevoir ce logiciel, l'entreprise s'adonne d'abord à une étude de marché pour déterminer les fonctionnalités logicielles les plus demandées. Puis, l'entreprise conçoit un logiciel qui implémente toutes ces fonctionnalités, par exemple la réservation, la création de comptes utilisateur ou encore un système de notation. Elle livre ensuite un seul et même logiciel à chacun des clients contenant l'ensemble des fonctionnalités. Le logiciel standard permet de n'avoir qu'un seul logiciel à maintenir par les développeurs. Avec le développement de logiciels individuels, une version du logiciel est conçue spécifiquement autour des besoins de chaque utilisateur.

Dans l'entreprise cliente, celle-ci développe trois versions de son logiciel pour convenir aux besoins spécifiques de ses clients. Cette stratégie de développement permet de réduire les coûts des logiciels à proposer aux utilisateurs, et de faire un logiciel qui correspond mieux à leurs besoins. On obtient alors une famille de produits logiciels au sens défini par David Parnas (1976) : « Une famille de produits logiciels est un ensemble de logiciels partageant une certaine proximité dans leurs objectifs et qui disposent de suffisamment de similarité pour justifier d'une mutualisation de leurs concepts et leur maintenance. »

Le problème est que ces logiciels sont à maintenir individuellement pour chaque utilisateur, ce qui augmente les coûts de maintenance.

Prenons l'exemple d'une automobile. Une famille de logiciels peut être construite à partir des logiciels qui tournent sur l'ordinateur principal d'un constructeur automobile, puisque ces voitures sont toutes équipées d'un système d'exploitation. Mais elles peuvent se différencier par des caractéristiques spéciales (par exemple, climatisation intelligente, écran multimédia). La conception de cette famille peut être considérée grâce à l'ingénierie des lignes de produits logiciels. Les produits logiciels sont les logiciels qui font partie des familles, tandis que la conception de ligne la conception de la plateforme pour construire ces produits' idée principale des LPL est de construire des logiciels en réutilisant la plupart de leur mise en œuvre (c'est-à-dire leur code source, la structure de données, les données, code source, la structure des données, la base de données et les ressources) pour concevoir les prochains logiciels nécessaires. La réutilisation est facilitée par le fait que la LPL encourage les développeurs à concevoir leurs logiciels autour d'éléments réutilisables. Par conséquent, la motivation de LPL est la suivante développement de logiciels pour la réutilisation et par la réutilisation.

Ces éléments seront réutilisés à travers les divers logiciels, de façon à minimiser le temps et les coûts de développement de chacun des logiciels.

Nous désignons ces éléments hétérogènes par le terme plus abstrait d'asset (ou artefacts).

Des études ont estimé que les entreprises peuvent bénéficier d'un retour sur investissement positif avec les LPL dès le développement du troisième produit (par rapport au développement d'un seul logiciel). Et pour le travail des développeurs, il y a une amélioration de la qualité du logiciel ; une réduction du délai de mise sur le marché du produit ; gain de temps et d'efforts pour tout nouveau logiciel en réutilisant les anciennes parties au lieu de redévelopper. L'idée étant de réutiliser les anciennes parties au lieu de les redévelopper.

Cependant, l'adoption des LPL requiert des compétences spéciales pour maîtriser l'ingénierie d'un SPL, et de nombreux développeurs ne sont toujours pas formés pour celles-ci. Parmi elles, il y a

l'ingénierie du domaine, l'identification des fonctionnalités et la gestion de la variabilité. Par conséquent, l'adoption des LPL comporte ses propres obstacles et présente des risques qui peuvent être trop élevés pour une entreprise.

En somme, L'ingénierie des lignes de produits logiciels promeut une réutilisation logicielle à grande échelle lors du développement de nouveaux produits. Au cours du développement, les caractéristiques de la ligne sont réutilisées pour la conception de nouveaux logiciels, ce qui réduit considérablement ses coûts de conception. Le gain est également visible sur le temps de développement qui tend à diminuer grâce à la réutilisation., cette réduction des coûts est seulement visible sur le temps long Cet investissement permet la conception des artefacts réutilisables en amont du développement des produits.

Lors de mon alternance, nous nous sommes concentrés sur la conception d'une nouvelle approche pour supprimer la barrière d'adoption à laquelle sont confrontés les développeurs. Notre fils rouge dans ce projet d'alternance était « Comment construire une approche automatisée d'adoption de Ligne de produit logiciel » Nous avons établi une liste d'objectifs intermédiaires à atteindre afin de définir notre approche.

Tout d'abord, nous décrivons notre approche automatisée et établissons les exigences pour ce que les développeurs devront avoir, afin d'utiliser la LPL et d'adopter l'approche. Pour être, nous notons que les développeurs auront besoin que la LPL soit compatible avec leur environnement de travail puisque le changer représente le principal risque pour l'entreprise.

Deuxièmement nous cherchons une approche d'automatisation qui construira la même LPL que ce que les développeurs auront fait manuellement. Enfin, nous voulons que la LPL résultante soit compréhensible pour les développeurs, afin que la maintenabilité et l'évolution soient possibles.

Lors de mon travail tout au long de l'alternance, je me suis concentré sur la conception d'une structure de données et d'un algorithme adéquat qui nous permettront d'intégrer itérativement les données. Cela nous permettra d'intégrer itérativement de nouveaux produits à l'intérieur d'une SPL, sans avoir besoin de l'intervention des développeurs. Troisièmement, nous concevons et mettons en œuvre une technique spécifique pour localiser la fonctionnalité à l'intérieur de la SPL.

Et enfin, nous développons des techniques pour simplifier la compréhension de nos sorties SPL pour les développeurs. En suivant notre objectif, nous avons développé une nouvelle approche appelée « SPL Generator », et nous l'évaluons en utilisant différents cas préexistants. Nous l'évaluons en utilisant différentes études de cas préexistantes pour simuler la construction d'une SPL.

L'alternance

Lors de cette année 2018/2019, j'ai eu l'honneur de faire mon alternance dans une petite entreprise cliente du groupe Accenture. Cette société s'occupe de la création et la maintenance de multiples logiciels pour des PME et autres sociétés. Ces logiciels peuvent être variés, l'entreprise possédant de nombreux clients très divers, d'où le possible nécessité d'avoir recours à des LPL. Les domaines d'activité sont surtout autour du numérique mais aussi de l'aviation ainsi que l'automobile.

Dès mon intégration en Septembre 2018, j'ai eu l'occasion de travailler sur les lignes de produits logiciel au sein de l'entreprise et de participer à la maintenance des multiples outils pour le développement des différentes gammes de logiciel à la demande des clients.

Outre la maintenance et le support quotidien des logiciels en production, j'ai pu participer à l'élaboration d'un outil d'automatisation de la génération des lignes de produits logiciels.

Pour cela il a fallu effectuer une étude de l'art des différentes recherche industrielles sur lignes de produits pour pouvoir par la suite proposer une nouvelle approche innovante avec les équipes R&D et IT.

La ligne directrice de mon alternance était, dans un premier temps, d'améliorer l'implémentation la ligne de produit et dans un second, de trouver un moyen d'automatiser l'intégration de nouvelles fonctionnalités dans cette ligne de produit logiciel. J'ai pu lors de mon alternance travailler avec différentes équipes :

L'équipe IT, qui s'occupe du support des logiciels et du bon fonctionnement de la ligne de produit se compose d'une dizaine de personne, elle s'occupe de la maintenance et des évolutions des logiciels avec l'équipe commerciale et les maitrises d'ouvrages.

L'équipe commerciale et les Maitres d'Ouvrages MOA : ces équipes sont au plus proches de l'environnement métier et dictent les projets ainsi que les nouvelles améliorations à intégrer dans les logiciels dans le but de satisfaire au mieux les attentes des clients. Ils prennent en compte les modification et changements du marché et ont une très bonne connaissance dans le domaine.

L'équipe Recherche et Développement, principalement composés d'ingénieurs et de chercheurs en ingénierie, cette équipe effectue une veille des dernières études de recherche dans différent domaine (génie logiciel, IA...) Leur but est d'adapter les travaux de recherche existant à notre entreprise, mais aussi de faire de la recherche en trouvant de nouvelles solutions innovantes. Le but de mon alternance s'inscrivait dans l'optique de la création de l'outil « SPL Generator », un générateur automatique de ligne de produit logiciel qui prend en compte une automatisation complète de la réingénierie de la ligne de produit.

Lors de mon alternance, j'ai donc pu être en lien permanent tout au long de l'année entre ces différentes équipes, cela m'a permis d'avoir une vision complète du fonctionnement des équipes présentes au sein d'une PME.

Ce projet, considéré comme de la recherche industrielle, s'inscrit dans une approche innovante d'automatisation du processus de réingénierie des lignes de produit logiciel liant plusieurs domaine autour du génie logiciel.

L'outil développé par mes soins « SPL_Generator » est le travail de 10 mois de recherche et

d'implémentation,(4 mois étude de l'art et découverte des LPL, 2 mois pour la proposition de la solution,3 mois pour la conception 1 mois de test). Ce fut la tâche majeure de mon alternance en plus du support quotidien des différents logiciel, et l'élaboration d'autres outils pour les équipes IT et R&D.

Les autres outils développés en marge de mon alternance sont des applications qui permettent de déplacer des dossiers ou des fichiers a des heures précises et de faire tourner des processus permettant le bon fonctionnement des autres applications en production dans l'entreprise.

« L'orchestre »

Le plan du mémoire d'alternance

Le déroulement de ce mémoire d'alternance s'effectuera en plusieurs temps, en première partie nous expliquerons le fonctionnement des lignes de produits logiciel, leur adoption en entreprise, ainsi qu'une étude de l'art des lignes de produits logiciel, leur conception, les différentes approches, ainsi que les méthodes d'ingénierie existantes qui seront utiles à notre approche innovante de réingénierie SPL_Generator.

La deuxième partie sera consacrée à l'outil SPL Generator, les diverses étapes de la conception, les méthodes utilisées pour l'automatisation du processus de réingénierie, l'intégration des produits, l'identification des assets, la gestion de la variabilité de la LPL et la localisation des caractéristiques.

Enfin nous dresserons une conclusion avec les premiers résultats de notre application sur des cas concrets ainsi que sur les gammes de logiciels de l'entreprise.

LE BUT

Comme entrevu précédemment, l'émergence de nouveaux besoins numériques, comme ceux qui accompagnent l'usage des téléphones mobiles ou des ordinateurs personnels, nos sociétés produisent davantage de logiciels et à grande échelle.

La recherche menée au sein du génie logiciel définit des méthodes et des techniques destinées à accompagner les ingénieurs dans la conception de ces logiciels. En particulier, le génie logiciel s'intéresse à l'amélioration des pratiques de conception, de maintenance, et d'évolution des logiciels. Parmi les nombreux défis du génie logiciel se pose la question de la diminution des coûts de production et du temps de conception des logiciels.

En guise de solution, le génie logiciel recherche de nouvelles méthodes de conception basées sur la réutilisation des logiciels.

La réutilisation logicielle systématique vise à concevoir des éléments logiciels réutilisables et à développer de nouveaux logiciels grâce à ces mêmes éléments.

On parle généralement de développement pour la réutilisation et de développement par la réutilisation.

Grâce à la réutilisation, les ingénieurs peuvent réduire le temps de développement en réutilisant des artefacts déjà conçus.

Ils améliorent également la qualité des logiciels puisqu'ils réemploient des artefacts déjà éprouvés.

En somme, la réutilisation permet de diminuer les coûts de production en diminuant le temps de développement des logiciels.

L'objectif est de Définir une nouvelle approche réactive adaptée au contexte de développement de logiciels des PME. En étudiant les approches d'adoption existante, il n'apparaît clairement qu'aucune n'est vraiment adaptée au contexte particulier des PME. Cependant, l'approche réactive semble être la plus proche de leur contexte, mais la difficulté de sa mise en œuvre reste un obstacle majeur

Nous verrons dans la prochaine partie en quoi L'ingénierie des lignes de produits logiciels est un paradigme de développement de familles de logiciels similaires basé sur le principe de la réutilisation systématique des artefacts. La ligne de produits logiciels permet de capturer et de factoriser dans une plateforme configurable l'ensemble des artefacts communs à une même famille de logiciels. Pour le reste des artefacts, spécifiques aux différents logiciels, la LPL capture dans sa plateforme la variabilité sous la forme d'un ensemble d'artefacts optionnels. La production de nouveaux logiciels se matérialise par une sélection méthodique des artefacts à réutiliser, où sont sélectionnés d'office les artefacts communs ; tandis que les artefacts optionnels sont sélectionnés de façon à personnaliser le nouveau produit selon les besoins exprimés par ses concepteurs. Cette réutilisation systématique, qui est organisée au travers d'une sélection d'artefacts préexistants, permet de réduire les coûts de production des logiciels, comparativement à un développement individuel.

C'est pour cela que nous proposons l'application de processus automatique pour réaliser l'ingénierie d'application de la Ligne de produit logiciel (LPL) afin d'en réduire ses coûts. Pour cela, notre approche réalisera automatiquement la construction de la plateforme variable, de

modèles de variabilités, et des processus de gestion et d'évolution de la variabilité. De plus, notre outil devra effectuer automatiquement la génération des produits depuis la plateforme, dès lors que les développeurs fournissent une configuration hypothèse est que retirer l'effort lié à cette ingénierie permettra une meilleure adoption des LPL. Dans le but d'inciter peut-être plus d'entreprises à construire une LPL, puisque les risques liés aux coûts d'ingénierie se retrouvent réduits par une automatisation des tâches.

La réduction des coûts de développement

Les avantages et bénéfices de l'adoption de l'ingénierie des LPL se mesurent par comparaison avec un développement "traditionnel". L'ingénierie des lignes de produits logiciels promeut une réutilisation logicielle à grande échelle lors du développement de nouveaux produits.

Au cours du développement, les artefacts de la ligne sont réutilisés pour la conception de nouveaux logiciels, ce qui réduit ses coûts de conception. Le gain est également visible sur le temps de développement qui tend à diminuer grâce à la réutilisation (c'est d'ailleurs la principale cause de réduction des coûts).

Cependant, cette réduction des coûts est seulement visible sur le temps long : des coûts survient à partir du troisième produit développé grâce à la ligne de produits comparativement aux coûts de développement produit par produit. De plus, la réduction des coûts nécessite un investissement initial conséquent. Cet investissement permet la conception des artefacts réutilisables en amont du développement des produits.

Réduction du délai de mise sur le marché des logiciels

Comme nous l'avons mentionné pour la réduction des coûts, l'application de cette ingénierie permet de réduire les temps de développement de nouveaux produits. Cette réduction de temps améliore les délais de mise sur le marché ou de livraison des produits logiciels. Il en découle des retours sur investissement plus rapides pour les entreprises, à condition là encore de concevoir les artefacts en amont.

Réduction des efforts de maintenabilité des logiciels.

À travers la plateforme configurable d'une LPL, la maintenance d'un artefact se propage virtuellement à tous les produits où cet artefact est implémenté. La plateforme supprime la nécessité de maintenir individuellement chaque produit et permet de factoriser les opérations de maintenance. À terme, cela permet de réduire l'effort de maintenance. De plus, la maintenance peut se faire sans connaître et maîtriser l'intégralité des implémentations des produits, ce qui réduit l'effort de compréhension.

Amélioration de la qualité des logiciels

La réutilisation logicielle impacte également la qualité des produits développés. En effet, les artefacts d'une LPL sont réutilisés et testés à travers les différents produits générés. À force de réutilisation, ces artefacts sont maintenus par les développeurs et ont donc plus de chances d'être corrigés ou améliorés que le code non réutilisé. Cette réutilisation permet ainsi de mieux détecter les erreurs de conception et de les corriger, ce qui augmente la qualité des produits d'une LPL

L'emploi de la plateforme configurable permet également d'améliorer la qualité des logiciels de la ligne, car elle regroupe les opérations de maintenance.

Etat de l'art et prérequis

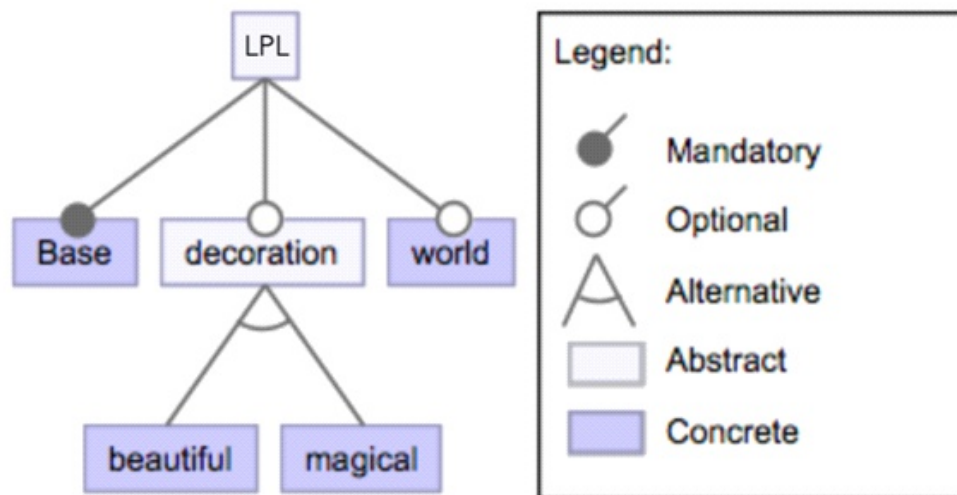
La ligne de produit logiciel

L'ingénierie des lignes de produits logiciels permet la conception d'un ensemble de produits logiciels partageant des caractéristiques communes, elles diffèrent cependant par l'implémentation de caractéristiques spécifiques. Prenons l'exemple d'une gamme d'imprimantes d'un même constructeur : Les logiciels embarqués dans ces imprimantes vont partager des fonctionnalités, comme celle de l'impression en couleur ou en noir et blanc. Mais certaines vont proposer des fonctionnalités supplémentaires et spécifiques, comme l'impression recto-verso, sur papier photo, ou encore un mode scanners de documents.

L'ingénierie des Lignes de Produits Logiciels (LPL) vise donc à regrouper des méthodes pour concevoir une famille de produits logiciels comme une ligne de produits logiciels. Ses objectifs se focalisent sur la réduction des coûts de développement, le raccourcissement des temps de livraison, l'amélioration de la qualité des logiciels et la réduction des efforts de maintenance.

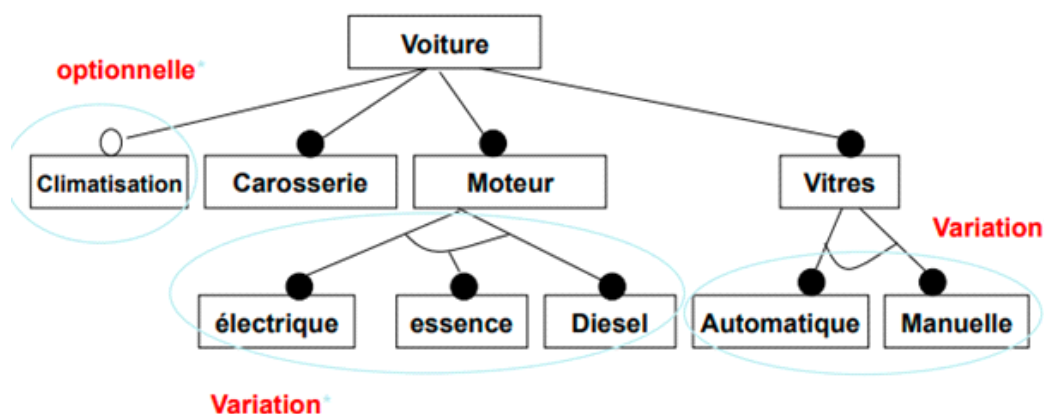
Dans cette section, nous présentons les différentes notions et concepts qui forment l'ingénierie des LPL.

Les caractéristiques logicielles peuvent être définies de plusieurs façon. Souvent, nous les désignons comme des fonctionnalités logicielles de haut niveau, perçu par l'utilisateur. Il s'agit de caractéristique utilisation. Cependant, certaines caractéristiques peuvent être invisible pour l'utilisateur. Nous parlons dans ce cas de caractéristiques métier comme par exemple l'usage d'un protocole de communication, UDP ou TCP entre l'imprimante et l'ordinateur client. Enfin, une caractéristique dépend principalement du domaine d'activité pour lequel le logiciel se destine. C'est



pourquoi il est possible de décrire un logiciel de la ligne par les caractéristiques qu'il implémente. Cela nous permet de définir pour chaque logiciel une configuration, un ensemble caractéristique que présente dans un produit logiciel particulier. Puisque les logiciels sont tous différents, et qu'ils sont construit depuis un même ensemble caractéristique, nous retrouvons dans une LPL des caractéristiques communes à tous les logiciels.

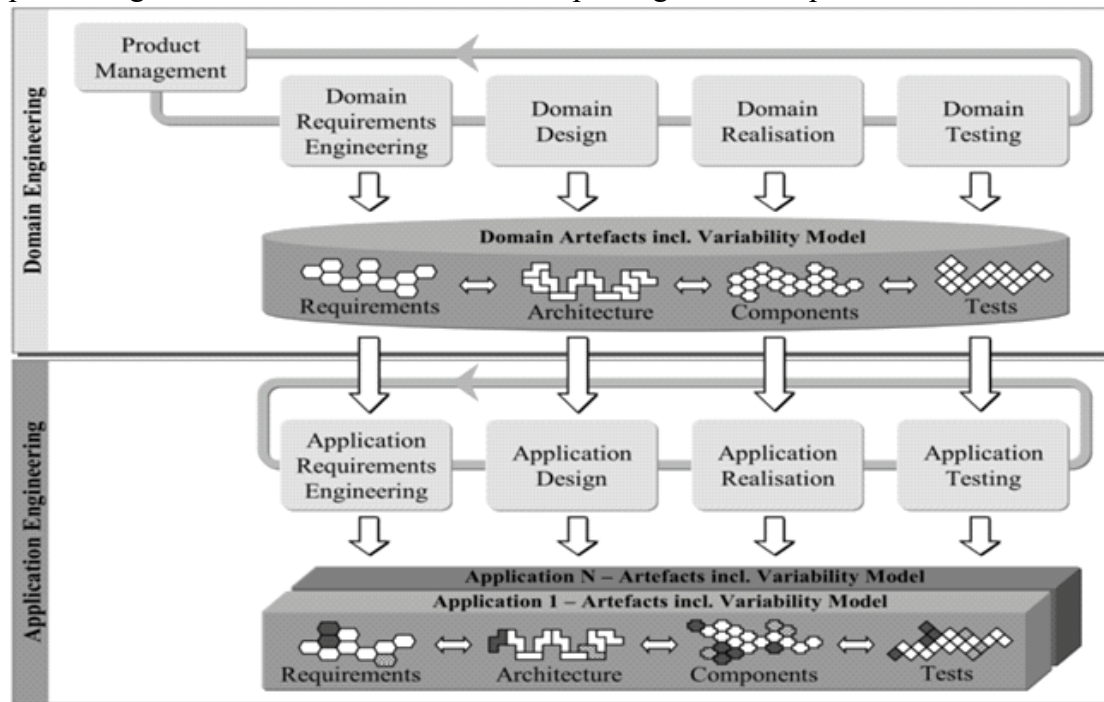
L'implémentation des différentes caractéristiques est réalisée par un ensemble de divers éléments logiciels. Il s'agit essentiellement de code source, mais également de ressources comme du texte,



des images, des scripts, des modules. Comme les caractéristiques qu'ils implémentent, un artefact peut être commun ou spécifique. C'est donc par l'intermédiaire des artefacts que la variabilité va être exprimé dans l'implémentation d'une LPL.

Plutôt que de concevoir séparément des logiciels différents, **les LPLs proposent de les concevoir**

par une réutilisation systémique, en préservant une forte personnalisation des différents logiciels. L'objectif de l'ingénierie d'une LPL est de concevoir une implémentation logicielle partagée par l'ensemble des logiciels de la ligne. Cette implémentation de la LPL regroupe les points communs entre tous les logiciels, ce qui représente un ensemble d'artefacts invariables. Elle comprend également les parties spécifiques aux produits, représentant l'ensemble des artefacts variables. Cette implémentation est en capacité de varier, ce qui permet la génération des différents produits. Pour cela, on y ajoute des mécanismes de variabilité, comme par exemple des annotations de pré-processing; ou des mécanismes basés sur le paradigme de composition de caractéristiques.



L'ingénierie des LPL intègre la réutilisation de logiciel en deux temps. Le développement pour la réutilisation des artefacts se réalise au travers d'un processus dédié appelé l'ingénierie du domaine

Dans ce processus, les artefacts relatifs à un domaine métier choisi sont identifiés, spécifiés et organisés pour former les parties communes et la variabilité des différents produits de la LPL.

Quant au développement avec réutilisation, un second processus appelé l'ingénierie d'application est dédié à la production des produits. L'ingénierie des LPL encourage l'application d'opérations de dérivation depuis la plateforme, pour sélectionner et intégrer les artefacts du domaine dans l'ingénierie du produit.

Ingénierie du domaine et ingénierie d'application

L'ingénierie du domaine est un processus d'analyse et de conception situé en amont du développement des produits d'une LPL. En fonction d'un domaine métier particulier (réservation, santé, automobile, aviation...), ce processus a pour objectif d'identifier tous les artefacts réutilisables et partagés par les produits de la future LPL. On parle généralement de l'ingénierie du domaine comme celle de l'ingénierie pour la réutilisation.

Analyse du domaine

L'ingénierie du domaine doit avant tout définir et focaliser le champ couvert par la LPL, en le réduisant à un domaine métier particulier. Ce choix est fait pour des raisons économiques et permet de restreindre le développement à des artefacts dont les développeurs pourront maximiser la réutilisation dans les produits ou à ceux qui répondent le mieux aux besoins identifiés par les clients. En d'autres termes, cela permet d'éviter le développement d'artefacts inutiles aux produits de la ligne (gain de temps et d'effort d'ingénierie). L'objectif de l'analyse du domaine est donc de définir et documenter les fonctionnalités et les comportements logiciels communs ou spécifiques aux produits de la LPL. Cette étape permet notamment de concevoir les modèles de variabilité de la LPL, souvent basée sur les fonctionnalités communes et spécifiques, identifiées pour les produits.

Conception du domaine.

Cette seconde étape s'appuie sur les connaissances acquises lors de l'analyse du domaine pour concevoir l'architecture générale de la plateforme configurable. À partir des modèles de variabilité, les concepteurs décident des points de variation dans l'architecture de la plateforme. Les modèles de variabilité peuvent être corrigés ou raffinés pour mieux correspondre à l'architecture générale de la plateforme et donc de la LPL.

Réalisation du domaine

Enfin, les concepteurs concrétisent l'architecture définie précédemment dans une plateforme logicielle, composée de l'ensemble des artefacts réutilisables du domaine.

La plateforme doit structurer les artefacts et les associer aux points de variation, afin de permettre la personnalisation de masse des produits. Cette personnalisation est faite à travers une sélection des fonctionnalités ou des comportements spécifiques que l'on souhaite avoir dans un produit ; Quant aux modèles de variabilité, ils permettent de rendre compte des combinaisons d'artefacts envisagés. Ces combinaisons définiront les différents produits d'une ligne.

Ingénierie d'application

L'ingénierie d'application vise à construire les produits de la ligne depuis le résultat de l'ingénierie du domaine. Ce second processus est donc un développement par la réutilisation. L'ingénierie d'application présente souvent deux phases distinctes : la configuration des produits et la génération des produits parfois appelée dérivation de produits dans la littérature.

Configuration des produits.

Les configurations des produits doivent respecter les contraintes métiers entre les artefacts, Par exemple. si deux fonctionnalités ont été définies comme mutuellement exclusives, alors elles ne

peuvent pas apparaître dans le même produit.

Des allers-retours sont parfois nécessaires entre l'ingénierie du domaine et celle d'application, afin de compléter l'analyse du domaine, car l'expertise des spécifications des produits peut révéler de nouveaux besoins.

Génération des produits

Dans un second temps, la génération du code des produits est réalisée à partir de la configuration du produit et de la plateforme.

Les artefacts sélectionnés dans la configuration sont assemblés suivant une structure donnée par la plateforme, afin de former correctement le code source du produit. La génération du produit est souvent une étape automatique. Le produit généré est ensuite testé, ce qui permet là encore des allers-retours avec l'ingénierie du domaine (correction, modification, ajout de fonctionnalité, etc.). Les produits peuvent ensuite être livrés aux clients.



Les approches de conceptions traditionnelles

Le développement from scratch

L'approche from scratch signifie simplement commencer à coder un projet à partir de zéro. Quel que soit le domaine métier du projet, le développeur va se lancer dans le développement sans l'utilisation de framework, librairie ou template. Ce développement radical est toutefois de moins en moins utilisé puisque de nombreuses librairies ou code source sont aujourd'hui déjà disponibles en ligne. Mais on peut aussi parler de développement from scratch de façon plus générale lorsqu'on débute avec peu d'assistance d'outils déjà existants.

Le clone and own

L'approche Clone and Own contraste particulièrement avec l'approche from scratch puisqu'elle consiste à partir du code d'un projet existant, et de le cloner pour coder le projet suivant. Cette pratique est très souvent utilisée dans l'industrie et chez les développeurs en général. Elle apparaît surtout lorsque le nouveau projet partage un grand nombre de similarités avec un projet déjà existant. Cela nécessite toutefois une grande connaissance du code existant pour être le plus efficace possible. Beaucoup de temps est souvent consacré à l'appropriation du code ou au nettoyage des parties inutiles du code. Mais cette technique peut être très complexe à mettre en place, car elle dépend trop de la qualité du code existant.

Le répertoire commun

Le répertoire commun est une approche qu'on trouve principalement à l'échelle d'une équipe de développement. Ce répertoire spécialement créé pour les projets de l'entreprise va contenir les codes sources, schémas, documents et devis client, ressources graphiques, relatives aux différents projets. Ce dossier est souvent volumineux et sa hiérarchie le rend peu pratique lorsque l'on veut y rechercher un élément pour le réutiliser. Et une fois un élément trouvé, il faudra encore l'adapter pour le faire convenir aux besoins du nouveau projet. Cette pratique peut être vue comme une façon ad hoc de réutiliser les éléments du projet.

Nous venons de le voir, les approches classiques souffrent de beaucoup de défauts, mais elles démontrent aussi une volonté de vouloir réutiliser un maximum le code ou les artefacts déjà créés. C'est ce constat qui poussa à la création d'un outil généralisant la réutilisation intelligente et pertinente du code et des autres artefacts des projets. Les lignes de produits logiciels vont offrir un environnement qui encourage le développement pour la réutilisation. Plutôt que de coder pour un seul produit, il est désormais possible de coder pour un ensemble de produits qui partageront des caractéristiques communes, mais qui auront surtout des caractéristiques variables.

Les approches fondamentales pour les LPL

Indéniablement, les lignes de produits logiciels ont montré leur potentiel industriel. Toutefois, nous pouvons encore déplorer le manque d'adoption à grande échelle de cette ingénierie, car rares sont les industriels à s'être emparés de cette ingénierie dans leur processus de développement quotidien. Pour comprendre ce constat, nous commençons par introduire les différentes stratégies d'adoption de l'ingénierie des lignes de produits.

Pour la conception d'une LPL il existe 3 approches fondamentales :

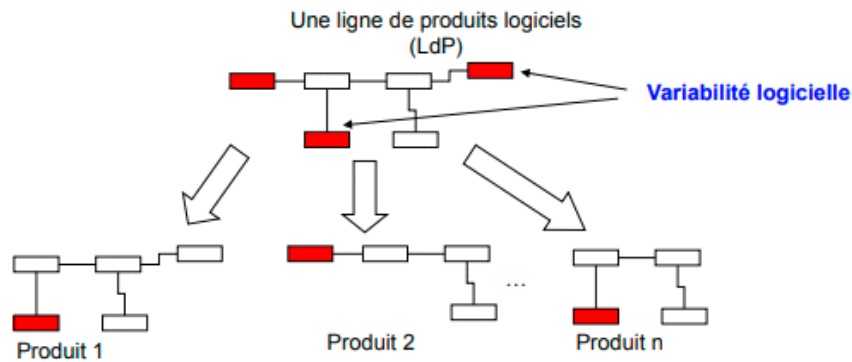
L'approche Proactive

La stratégie proactive est la stratégie conventionnelle où sont réalisées dans l'ordre l'ingénierie du domaine puis l'ingénierie d'application. En réalisant l'ingénierie du domaine en premier, les développeurs s'assurent d'avoir fait une analyse approfondie des artefacts réutilisables du domaine. L'investissement initial de cette phase est coûteux et représente donc un risque. Toutefois, il s'avère rentable une fois l'ingénierie d'application amorcée et le retour sur investissement apparaît dès le troisième produit.

Cette stratégie se destine donc davantage à des entreprises qui disposent d'une expertise dans un domaine précis, ou qui disposent des ressources nécessaires pour financer l'investissement initial. **L'approche proactive** est un processus Top-Down basé sur une analyse approfondie du domaine d'activité de la ligne de produit pour identifier et construire tous ses éléments. L'approche Proactive est souvent décrite en deux phases principales : l'ingénierie du domaine, et l'ingénierie de l'application. Comme vu précédemment dans la partie précédente, **L'ingénierie du domaine** est une phase dont l'objectif est de s'approprier en profondeur un domaine métier particulier (e.g. le e-commerce, la réservation hôtelière, l'impression, etc.), pour en faire ressortir les cas d'utilisations ou besoins métiers du domaine. Ces cas et besoins vont être décrits sous la forme d'exigences logicielles, et vont donner naissance aux différentes caractéristiques de la LPL. Les développeurs d'une LPL tirent de cette analyse l'ensemble de caractéristiques logicielles distinctes et réutilisables. L'enjeu est ici de découvrir un maximum des caractéristiques, et d'évaluer l'intérêt de chacune d'entre elles à être implémentées dans la LPL finale, afin de rester dans des coûts de productions rentable pour l'entreprise. L'ingénierie du domaine permet donc aux développeurs de déterminer la variabilité du domaine. Cette variabilité s'applique aux différentes caractéristiques découvertes et les développeurs ont pour objectifs d'obtenir les relations qui existent entre ces caractéristiques.

Par exemple, ils doivent s'interroger des conditions d'utilisation d'une caractéristique particulière dans un produit (e.g. est-il possible d'imprimer en couleur sans pouvoir imprimer en noir et blanc ?). Cette conception de la variabilité s'accompagne de la réalisation de modèles de variabilité, comme des modèles de caractéristiques.

Pour finir, l'ingénierie du domaine permet aux développeurs de définir l'architecture logicielle de leur LPL, en tenant compte des caractéristiques et de la variabilité précédemment obtenus.



L'ingénierie d'application est la seconde phase, celle-ci se consacre au développement de la LPL. Son but est de réaliser l'implémentation de la LPL et ses produits à partir de l'architecture et des caractéristiques définies lors de l'ingénierie du domaine. La structure de la LPL regroupe toutes les implémentations des différentes caractéristiques, c'est à dire leurs implémentations par un ensemble d'artefacts.

L'implémentation SPL contient également les mécanismes de la variabilité de la SPL, qui lui permettent de varier pour former les différents produits. C'est donc depuis cette implémentation que les développeurs vont faire émerger les différents produits de la LPL. Pour cela, les développeurs définissent des configurations de produits, lesquelles décrivent les caractéristiques présentes dans le produit. Pour obtenir un produit, un moteur de génération vient, le plus souvent automatiquement, générer le produit à partir de l'implémentation de la LPL en y appliquant la configuration du produit. L'ingénierie d'application se termine par le test des différents produits ou des différentes caractéristiques. En fonction des tests, il est possible de faire des allers-retours avec la phase d'ingénierie du domaine, afin de corriger d'éventuels problèmes de conception

Avantages et Inconvénients

L'approche Proactive comporte plusieurs avantages.

D'une part, son ingénierie (domaine et application) permet d'obtenir une couverture maximale du domaine d'activité de la LPL. Ainsi, le risque d'absence d'implémentation de caractéristiques essentiels à un domaine est minimisé.

D'autre part, la réalisation de l'ensemble des constituants de la LPL par les développeurs, -comme les caractéristiques, l'implémentations de la LPL, ainsi que les modèles de variabilités et produits-, leur offre une maîtrise de la maintenabilité et de l'évolution de leur ligne de produits. Les développeurs savent donc comment sont implémentées les caractéristiques et connaissent les mécanismes de variabilités mis en œuvre pour construire les produits.

Cependant, en pratique cette approche est difficile à mettre en place dans les entreprises. Tout d'abord, s'il existe des IDE pour assister les développeurs, le travail d'ingénierie de la LPL reste essentiellement manuel. Cette ingénierie étant différente de celle d'une conception traditionnelle de logiciels, la première difficulté est de trouver des développeurs compétents pour mener à bien cette approche, et notamment l'ingénierie du domaine. Cette phase est la plus risquée des deux, car elle nécessite des compétences d'analyse du domaine, de gestion de la variabilité et de construction d'architecture. Il est donc important d'anticiper et de couvrir le plus de caractéristiques possibles, afin d'éviter les allers-retours avec l'ingénierie du domaine et d'application.

Ainsi l'ingénierie du domaine est inadaptée pour les entreprises qui sont dans l'incapacité

d'anticiper les besoins de leurs produits.

L'approche Extractive

La stratégie extractive est une adoption par rétro-ingénierie, où une famille de produits préexistante est transformée en une ligne de produits logiciels. Cette stratégie permet de tirer profit du développement déjà éprouvé des produits. La LPL est construite par extraction, c.-à-d. qu'on vient identifier les artefacts communs et variables des produits pour concevoir la LPL. L'avantage de cette stratégie est son potentiel d'automatisation, et des travaux se sont déjà employés à proposer des mécanismes d'extraction outillés. Si elle est outillée, le coût d'investissement destiné à la transformation peut être largement réduit, en fonction cependant du degré d'automatisation proposé par l'outil. La stratégie extractive se destine donc à des entreprises qui disposent de famille de produits gérés individuellement et qui souhaitent bénéficier des LPL pour réduire leurs efforts de maintenance, d'évolution et de production de nouveaux produits.

L'approche extractive est un processus Bottom-Up d'extraction des constituants d'une LPL à partir d'un ensemble déjà préexistant produits. Il s'agit donc d'une réingénierie d'une famille de produits vers ligne de produit. Dans le papier « *Empirical Software Engineering* » (2017) cette réingénierie doit s'articuler selon trois phases génériques : **la détection, l'analyse et la transformation**.

La phase de détection débute le processus d'extraction. Elle se consacre à la récupération de la variabilité des différents produits, en l'étudiant au travers des artefacts des produits. L'objectif est donc étant de différencier les artefacts communs à l'ensemble des produits, des artefacts spécifiques à seulement quelques-uns d'entre eux. La détection s'accompagne souvent d'une localisation de l'implémentation des caractéristiques (ou feature location en anglais), qui permet de rétroactivement faire l'ingénierie du domaine de la ligne de produits.

La phase d'analyse s'appuie sur la variabilité acquise lors de la détection pour construire les modèles de variabilités de la LPL. Ces modèles prennent le plus souvent la forme de modèle de caractéristiques (ou feature-model en anglais).

Enfin, la **phase Transformation** construit une implémentation pour la LPL qui respecte la variabilité détectée et qui structure l'ensemble des artefacts.

Avantages et Inconvénients :

Les avantages de l'extraction pour la construction d'une LPL sont multiples. Premièrement, les constituants de la SPL peuvent être récupérés automatiquement (par exemple : les artefacts, les fonctionnalités, le Feature-Model) en analysant les codes sources et autres artefacts associés à ces produits (par exemple les exigences logicielles, le manuel utilisateur, les documents de conception...). Par conséquent, ce processus est généralement automatique, ce qui rend le coût de réingénierie des produits vers la LPL relativement faible. L'automatisation rend cette approche rapide à déployer, et sans risques particuliers pour les entreprises. Cependant, il est à noter que la réalisation et le fonctionnement de ce type d'approche repose sur plusieurs hypothèses dans le code informatique utilisé par les développeurs. Les produits préexistants ont été construits de façon ad hoc, c'est-à-dire que les développeurs ont réutilisé des portions des produits les plus anciens pour

créer les plus récents. On retrouve ainsi des portions de codes communes, et le plus souvent une architecture commune à tous ces produits. Le source code des produits est disponibles et est utilisé pour l'analyse la réingénierie.

Ainsi, le principal obstacle à l'adoption de cette approche réside dans la disponibilité d'un large ensemble de variants de produits préexistant, développés de manière ad hoc, et suffisant pour couvrir le plus de caractéristiques d'un même domaine d'activité. Si l'ensemble des produits est insuffisant, la réingénierie automatique devra logiquement être complétée manuellement : ce qui augmentera les coûts, et diminuera l'intérêt de l'approche. Il est évident que de nombreuses entreprises ne remplissent pas ces conditions comme les TPE PME.

En plus de cette limitation, les travaux existants appliquant une approche extractive se limite à proposer une implémentation fermée de la LPL, très difficile à y analyser (en boîte noire). Ils destinent souvent la LPL extraite à être utilisée par des gestionnaires de produits s'en servent comme d'un inventaire de leurs produits, plutôt que par des développeurs. Ces derniers devant maintenir et faire évoluer une LPL, si nécessaire. Selon la première loi de l'évolution des logiciels de Lehmann - qui stipule que tout logiciel qui n'évolue pas meurt -, l'implémentation fermée de la LPL, a donc un impact négatif sur sa longévité. Cela rend également impossible le fait de compléter manuellement la LPL une fois extraite. Par conséquent, l'approche extractive se révèle intéressante pour une utilisation par un gestionnaires de produits, et pour des entreprises type éditeurs de logiciels, qui souhaitent migrer un ensemble de logiciels gérés individuellement, vers une ligne de produits logiciels.

L'approche Réactive

La stratégie réactive propose une construction de la LPL par itération, en fonction des besoins. Contrairement à la stratégie proactive, les ingénieries du domaine et d'application sont réalisées de façon partielle à chaque itération, de sorte à ne couvrir que les artefacts nécessaires à la production d'un nouveau produit. Cette stratégie permet de répartir l'investissement initial sur plusieurs itérations. La prise de risque est donc réduite, car les produits conçus à chaque itération peuvent être livrés sans délai, ce qui accélère la rentabilité de chaque investissement. Cet avantage est contrebalancé par les difficultés liées à la réingénierie itérative de la LPL. Contrairement à la proactive, chaque itération demande de reprendre l'ingénierie du domaine et d'application, afin d'intégrer à la LPL de nouvelles exigences. Cette stratégie s'adapte au contexte de développement des entreprises qui ne sont pas en mesure de prévoir les exigences de leur LPL. Elle s'adapte aussi aux sociétés ayant à effectuer des cycles de développements occasionnels, ou ne disposant pas des ressources pour adopter une stratégie proactive. L'approche réactive ressemble à l'approche proactive mais elle fonctionne de manière itérative, l'approche se fait de manière plus locale par des petite analyse afin de détecter le minimum d'un nouveau produit

L'approche réactive peut être décrite comme un processus itératif afin de construire graduellement une LPL. Plutôt que de concevoir une LPL entière en une seule itération, l'approche réactive répartit l'investissement dans la LPL sur le long terme. La construction de la ligne est motivée par l'apparition de nouvelles exigences pour de nouveaux produits, et donc de nouvelles caractéristiques à inclure dans la LPL. L'arrivée occasionnelle de nouvelles exigences est donc un moteur opportuniste de construction pour la ligne de produit. Cette approche est comparable à un processus de développement logiciel dit traditionnel

Dans les deux cas, les développeurs doivent concevoir leurs nouveaux produits à partir des exigences (c'est-à-dire la description d'un nouveau produit formulé par un client).

L'approche réactive remplace le fastidieux clone&own que l'on a présenté précédemment. Comme cette approche se concentre principalement sur la livraison de nouveaux produits, elle profite aux entreprises travaillant sur des logiciels sur mesure, qui recherchent un retour sur investissement rapide de leur SPL, ou aux entreprises dont les contours sont encore incertains

Avantages et Inconvénients :

L'approche réactive apparaît comme une approche ayant un potentiel industriel important, du fait sa répartition des coûts d'investissement et son aspect itératif (qui permet d'interrompre la construction à tout moment). Cependant, d'autres études rapportent que l'approche réactive souffre encore de difficultés majeures liées à la réingénierie itérative de la LPL. En effet, le problème est que l'intégration de nouvelles exigences est difficile et nécessite une expertise liée au domaine métier et au domaine technique. Il s'agit d'identifier toutes les fonctionnalités déjà implémentées dans la LPL, les nouvelles fonctionnalités destinées à un nouveau produit, et la relation de variabilité entre elles. Pour l'expertise technique, qui est le rôle des développeurs, il s'agit de modifier l'implémentation existante de la LPL afin d'intégrer les artefacts des nouvelles fonctionnalités requises sans mettre en péril la capacité de la LPL à régénérer ses anciens produits. Comme son ensemble de produits augmente avec le temps, une LPL devient plus complexe à réingénier avec plus de fonctionnalités, plus d'interactions entre les fonctionnalités et une variabilité plus riche. De plus, les développeurs doivent mesurer soigneusement les impacts de leurs choix d'implémentation, car ils peuvent avoir un effet imprévisible sur les itérations suivantes. Dans certains cas, l'effort de réingénierie, et donc son coût, peut être si élevé qu'il devient un risque pour l'entreprise. Par conséquent, cette difficulté à construire et à faire évoluer manuellement un SPL est considérée comme le principal obstacle à l'adoption généralisée de l'approche réactive

En résumé :

L'adoption de l'ingénierie de lignes de produits propose trois stratégies : proactive, extractive et réactive.

La stratégie proactive conçoit l'ingénierie de haut en bas, de l'ingénierie du domaine à l'ingénierie d'application. Cette stratégie est coûteuse et risquée, mais qui permet de construire la LPL en une fois. Elle se destine aux entreprises capables d'en assurer les coûts ou aux éditeurs de logiciels capables de maîtriser les spécificités d'un domaine d'activité particulier.

La stratégie extractive est une ingénierie rétroactive, de bas en haut. Elle propose de transformer une famille de produits préexistante en une ligne de produits logiciels. Par conséquent, son adoption est restreinte aux entreprises disposant d'une famille de produits préexistante. Cette stratégie dispose d'un fort potentiel d'automatisation, comme l'ont montré les travaux relatifs à son automatisation.

Enfin l'approche réactive étant une version itérative de l'approche proactive, qui dilue l'investissement initial et l'ingénierie de la LPL au fur et à mesure de l'apparition de besoins pour de nouveaux produits. Si l'ingénierie par itération est sa principale qualité, elle oblige cependant à faire la réingénierie manuelle de la LPL à chaque itération. Cette réingénierie est complexe et peut engendrer plus de coûts qu'une approche proactive.

En conclusion, nous constatons que ces stratégies souffrent toutes de limitations, et ne peuvent être adoptées par tous types d'entreprises. Pour les stratégies proactive et réactive, les problèmes sont liés à une ingénierie manuelle complexe ; et les tentatives d'automatisation notamment lié à l'extractif se montrent finalement incomplètes.

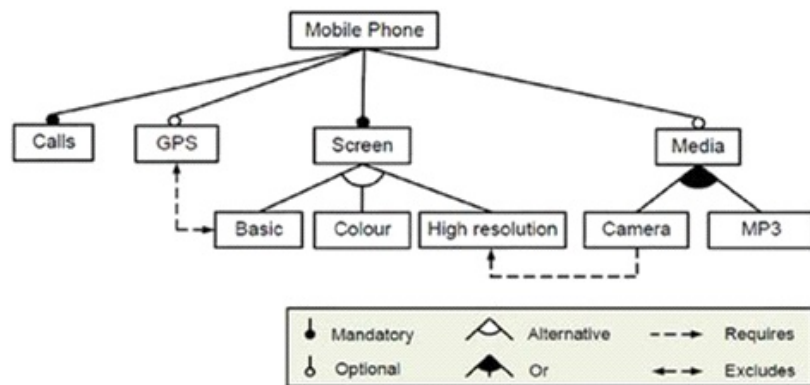


Figure 1: Voici un exemple

Quelle approche adaptée aux PME ?

Les exigences d'une PME vis-à-vis de l'automatisation des LPL s'articule autour de 3 points fondamentaux : La facilité de manipulation par des développeurs, l'automatisation par l'intégration de nouveaux produits et enfin l'évolution dans le temps de la LPL.

Nous avons vu qu'il existait trois approches pour l'adoption des lignes de produits logiciels. Parmi elles, les approches proactives et extractives s'adaptent difficilement aux PME : la première demande un coût d'investissement initiale trop important et son cycle de développement est inadéquat avec celui des PME (développement à la demande) ; La seconde repose sur l'utilisation de produits préexistant pour construire la ligne, ce qui est rarement le cas des PME. Mais surtout, l'approche extractive n'envisage pas l'évolution de la LPL une fois l'extraction réalisée. La dernière approche d'adoption, dite réactive, s'adapte mieux au rythme de développement des PME, car elle propose un développement opportuniste de la LPL, en fonction du besoin de créations de nouveaux logiciels. Cependant, cette approche demande des compétences en ingénierie des LPL, souvent absente chez les développeurs (analyse du domaine, application du domaine, compréhension et gestion de la variabilité). Et malgré cela, sans doute la plus difficile des trois à maîtriser, du fait de la réingénierie de la LPL qu'elle impose à chaque création de nouveaux produits.

En plus des problèmes propres aux approches d'adoption, la mise en œuvre des LPL est freinée par une réticence aux changements des pratiques de développement, établis dans les entreprises. En effet, imposer de nouvelles pratiques de développements est complexe, quelques soient l'entreprise. Cela implique souvent des restructurations d'équipes de développeurs, l'acquisition de nouvelles compétences (par la formation ou le recrutement de développeurs), et autres changements dans le processus de production des logiciels. Mais cela l'est sans doute d'avantage chez les PME, car elles ont construit des pratiques de développements efficaces, pour répondre aux contraintes fortes de temps auxquelles elles font face.

Par ailleurs, l'adoption et la mise en œuvre des LPL comprend deux grands défis : d'une part elle oblige souvent les entreprises à repenser et transformer leurs pratiques de développement afin d'être correctement adoptée. D'autre part, les bénéfices des LPL sont visibles sur le long terme. Leur retour sur investissement arrive souvent après le développement du troisième produit issue de la LPL. Face à aux défis que représente l'adoption, les entreprises perçoivent souvent l'adoption des LPL comme un risque, plutôt que comme un avantage stratégique. Dans ce contexte particulier des PME, l'adoption des LPL est complexe, compte tenu des par les problèmes rencontrés dans les approches d'adoption existantes, et du fait de leurs réticences l'adoption de nouvelles pratiquement de développement.

L'approche réactive apparaît donc comme la plus adaptée au contexte des petites et moyenne entreprises

Cependant cette approche possède des mauvais côtés que nous allons étudier en profondeur pour pouvoir, peut-être, résoudre par une automatisation de certains processus afin de la rendre acceptable dans le cadre des PME

Dans l'approche réactive, l'ajout de nouvelles caractéristiques (ou leurs évolutions pour répondre aux spécifications d'un nouveau produits), peut avoir des répercussions sur le fonctionnement de produits préexistants dans la LPL. Ainsi, la première difficulté pour les développeurs est de s'assurer que leur réingénierie ne compromette pas l'intégrité des caractéristiques, et donc des produits, préexistants.

Deuxièmement, il ressort d'études sur la mise en pratique de l'approche réactive, que cette phase peut être simplifiée ou complexifiée par les choix pris par les développeurs lors des réingénieries précédente. Plus l'implémentation est facile à maintenir, plus il sera facile d'y intégrer le nouveau produit. Pour faciliter cette maintenabilité, il est nécessaire que les développeurs face preuve d'une capacité à anticiper de potentiels ajouts dans leur implémentation. Or, cette anticipation vient se confronter à l'aspect réactif de l'approche, où il est difficile de prédire les besoins des futurs produits. De plus, chaque modification représente un coût, qui ne peut être rentabilisé que si l'anticipation s'avère finalement utile. Cette accumulation de coûts provoque un risque pour ces petites et moyennes entreprises, et donc une probable diminution de son adoption des LPL.

Les entreprises nouvelles ou petites n'arrivent peut-être pas à envisager les futurs produits qu'elles vont devoir créer et donc les transformations qu'elles vont devoir apporter à la LPL. Ces entreprises n'ont pas le recul nécessaire pour anticiper ces contraintes en raison de leur taille. Et la création de nouveau produit pourrait compromettre les différents produits déjà préexistants.

Comme nous l'avons vu, la stratégie réactive propose une ingénierie basée sur un cycle de développement opportuniste de la LPL. Si l'on prend le cas des PME, ces entreprises fonctionnent de façon opportuniste : ceux sont les demandes occasionnelles de clients qui motivent la production de nouveaux produits. Ainsi il apparaît nécessaire que le développement de la LPL soit compatible avec un cycle de production opportuniste et occasionnel : les développeurs doivent pouvoir construire et mettre en pause l'ingénierie de la LPL à tout moment.

Dans la deuxième grande partie de mon mémoire d'alternance, nous verrons aussi que notre outil SPL Generator peut également s'appliquer dans une stratégie extractive. Il suffit pour cela de disposer d'une famille de produits en amont de l'utilisation. Cette famille de produits est ensuite intégrée en une seule fois, c'est-à-dire que les produits sont passés un à un et sans délai pour leurs intégrations, dans le but d'obtenir une LPL en sortie.

L'analyse Formelle

L'analyse formelle des concepts (Formal Contexts Analysis en anglais, FCA) est une méthode générique de classification non supervisée et de regroupement dans sa version primitive. Elle permet, à partir d'une description de données appelée un contexte formel, de former des concepts et une hiérarchie entre ces concepts appelée le réseau de concepts. Un Contexte Formel (CF) est une relation binaire représentant la propriété entre objets et attributs. Représenté sous la forme de tableau, ils matérialisent deux contextes formels. L'un représente les produits logiciels associés aux fonctionnalités qu'ils fournissent et un second, lui, associe les produits logiciels aux artefacts qui les mettent en œuvres.

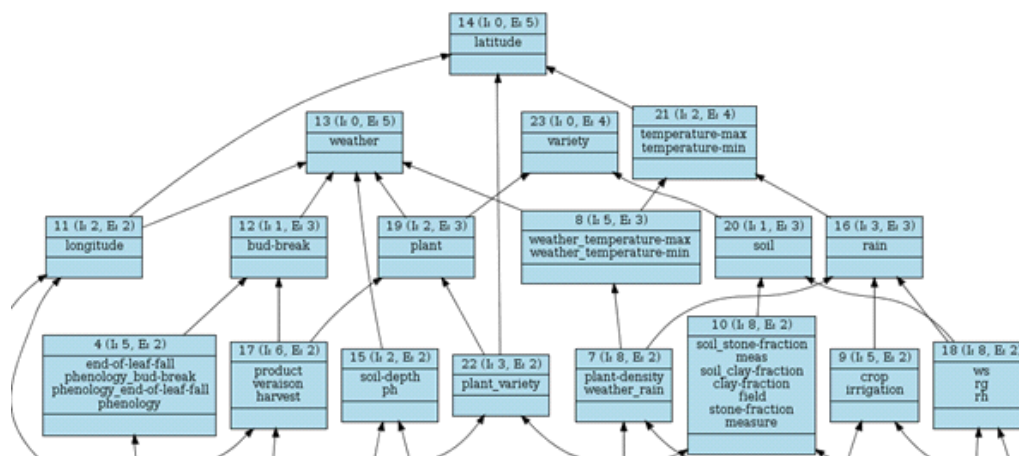
Un concept représente l'association d'un groupe maximal d'objets avec le groupe maximal d'attributs qu'ils partagent. Chaque concept est divisé en deux parties : une intention et une extension. L'intention représente l'ensemble des attributs. L'extension, quant à elle, représente l'ensemble des objets.

Une hiérarchie entre concepts est formée à partir d'un ordre partiel basé sur l'inclusion de l'intention et de l'extension. Cet ordre partiel permet donc une forme d'héritage entre les concepts. Alors un diagramme représentant une hiérarchie de concepts ne montre, pour un concept, que son réduit intention, privé de ses attributs hérités ainsi que son réduit extension, privé de ses objets hérités.

Plusieurs hiérarchies peuvent être construites avec FCA, le treillis de concepts étant la hiérarchie contenant tous les concepts. Ce travail utilisera la hiérarchie restreinte aux concepts qui introduisent au moins un objet ou un attribut.

L'extraction de relations permet de récupérer les liens entre les attributs des concepts, par exemple des relations de co-occurrence, d'implication, et d'exclusion mutuelle.

En particulier, les relations implication, co-occurrence, et exclusion mutuelle. Les implications sont déduites des liens d'héritage entre les concepts introducteurs : les attributs introduits par un concept parent sont impliqués par les attributs introduits par ses enfants. D'autre part, l'exclusion mutuelle est présente entre les attributs de deux concepts dont les extents ne se croisent pas. Une exclusion mutuelle signifie que les attributs de deux concepts ne peuvent jamais être trouvés ensemble. Des techniques sont disponibles dans la littérature pour extraire ces relations



L'Analyse de Concepts Relationnels (RCA en anglais) est une extension de FCA qui crée des hiérarchies de concepts connectés multiples pour apprendre et regrouper à partir de relations dirigées entre des contextes formels multiples. En entrée, RCA prend une Famille de Contextes Relationnels. Une FCR contient deux ensembles, un ensemble de contextes formels et un ensemble de contextes relationnels. Un Contexte Relationnel représente une relation unidirectionnelle entre deux types d'objets. Plus précisément, il s'agit d'une relation binaire, orientée des objets d'un contexte formel source vers les objets d'un contexte formel cible. Pour calculer les concepts, RCA crée des attributs relationnels quantifiés, qui représentent des quantificateurs (par exemple, existentiel $\exists$ ou universel $\forall$), r est une RC et C est un concept du treillis du Contexte Formel cible de r . Par conséquent, un attribut relationnel décrit les objets d'un contexte formel par leurs liens avec les concepts d'un autre contexte formel. Dans ce travail, nous utilisons le quantificateur $\exists$, qui établit un lien entre un produit et un concept d'artefact ou un concept de caractéristique lorsque le produit possède au moins un élément de l'extension du concept.

En sortie, RCA produit de multiples hiérarchies de concepts, par exemple de multiples AOC-Posets, un pour chaque contexte formel. Notez que l'AOC-Poset d'un CF qui est la source d'un ou plusieurs RC, contient deux types d'attributs dans ses concepts intents : attributs natifs ; et attributs relationnels ce sont les attributs pointant à travers une relation vers les concepts du contexte formel cible.

Dans les diagrammes, lorsque les concepts cibles sont des introducteurs, ils sont remplacés par leurs propres attributs natifs pour simplifier la lecture et être utilisés dans notre approche.

	Base	F1	F2	F3	F4
P1	X	X	X	X	
P2	X				
P3	X	X	X		X
P4	X	X		X	
P5	X		X		X
P6	X			X	
P7	X		X	X	
P8	X	X		X	X

Un Contexte formel (CF) est donc l'ensemble des relations entre des objets et des attributs. Il peut se décrire dans une matrice comme illustré dans la Table ci-dessus

Un concept représente l'association d'un groupe maximal d'objets avec le groupe maximal d'attributs qu'ils partagent. Chaque concept est divisé en trois parties (de haut en bas) : un nom, une intension et une extension. L'intension représente l'ensemble des attributs partagés par l'ensemble des objets du concept.

L'extension, quant à elle, représente l'ensemble des objets regroupés dans ce concept parce qu'ils partagent les attributs de l'intension. Les concepts peuvent être hiérarchisés pour former un treillis de concepts. Une hiérarchie est formée à partir d'un ordre partiel basé sur l'inclusion des intensions et extensions des concepts. Cet ordre partiel permet une forme d'héritage entre les concepts. Les concepts sont hiérarchisés pour former un treillis de concepts.

Dans le treillis de concepts, il est souvent choisi de ne représenter que l'intension réduite et l'extension réduite des concepts pour simplifier la représentation. L'intension réduite d'un concept est privée des attributs mentionnés dans les intensions des super-concepts. De même, l'extension réduite est utilisée pour ne faire apparaître les objets dans un concept seulement si l'intension complète du concept contient l'ensemble des attributs des objets.

L'avantage d'utiliser les intensions et extensions réduites est qu'elles permettent de faire apparaître explicitement les concepts introducteurs d'attributs ou d'objets. Un concept est introducteur d'attributs (respectivement d'objets), si son intension (resp. extension) réduite contient au moins un attribut (resp. objet). Dans notre projet, nous utiliserons les concepts réduits pour la représentation des AOC-Posets.

Plusieurs hiérarchies peuvent être construites avec l'ACF, le treillis de concepts étant la hiérarchie contenant tous les concepts. Dans notre projet, nous utiliserons la hiérarchie des AOC-Posets. Cette hiérarchie est restreinte aux concepts introducteurs (c.-à-d. aux concepts qui introduisent au moins un objet ou un attribut).

La variabilité d'une Ligne de produit logiciel

La gestion de la variabilité

Lors de l'ingénierie d'une ligne de produits, les concepteurs sont amenés à définir et à exprimer la variabilité des artefacts. Pour concevoir la ligne, ils doivent spécifier les artefacts communs et optionnels, en plus des contraintes de variabilité entre ces artefacts qu'il faut respecter pour construire les produits. Le paradigme de développement multiproduits des LPL exige donc une modélisation de la variabilité d'une LPL. Cette modélisation accompagne les concepteurs tout au long du cycle de vie de la LPL. Les modèles de variabilité facilitent la compréhension, la maintenance et l'évolution de la LPL. En agissant sur autant d'étapes du cycle de vie d'une LPL, il apparaît essentiel que cette modélisation soit la plus compréhensible et accessible possible pour les différents acteurs participant à l'ingénierie (p. ex. développeurs, responsables produits, clients, commerciaux, etc.).

Car lors de mon alternance, il est apparu évident de clarifier au maximum les différentes parties des fonctionnalités de la LPL et de son implémentation pour que l'intégralité des équipes IT puissent facilement s'y retrouver.

Parmi les techniques de modélisation de la variabilité existante, la modélisation de la variabilité par caractéristiques est la plus populaire comme l'approche de Czarnecki (2012)

Caractéristique logicielle

La configuration de la plateforme permet de générer l'implémentation du produit souhaité par une sélection des artefacts spécifiés par les développeurs. Cependant, il devient rapidement fastidieux de sélectionner ces artefacts un à un pour générer un produit. C'est pourquoi l'ingénierie des LPL préconise de décrire les produits en termes de caractéristiques logicielles qui les composent. Ce paradigme de description conçoit le logiciel en termes de caractéristiques. La caractéristique logicielle est une fonctionnalité logicielle ou un comportement particulier dans un logiciel, souvent perçu par l'utilisateur du logiciel. Plusieurs dizaines de définitions apparaissent dans la littérature pour tenter de définir formellement ce qu'est une caractéristique.

C'est comportement d'un système logiciel visible par un utilisateur. Les caractéristiques sont utilisées dans l'ingénierie des LPL pour spécifier et communiquer les points communs et les différences des produits aux différents intervenants et pour guider la structuration, la réutilisation et les variations à travers chaque étape de vie d'un système logiciel. Les caractéristiques permettent de décrire les spécificités implémentées par chaque produit, et de décrire un logiciel comme la somme de ses caractéristiques. Dans le contexte d'une même ligne de produits logiciels, deux logiciels partageant les mêmes caractéristiques sont considérés comme égaux. L'ensemble des caractéristiques d'un produit forme la configuration du produit.

Lors de l'ingénierie d'application, l'étape de configuration des produits consiste donc à décrire les produits d'après l'ensemble des caractéristiques qui les composent. La variabilité de la LPL est donc le plus souvent exprimée en utilisant les caractéristiques.

Les caractéristiques sont directement liées aux différents artefacts réutilisables de la LPL. Au travers de la sélection de caractéristiques, les concepteurs de la LPL peuvent donc sélectionner des artefacts à implémenter dans le produit. Les artefacts sont perçus comme l'implémentation concrète des caractéristiques dans la plateforme configurable de la LPL. La configuration de la plateforme, qui permet d'obtenir un produit, se fait en appliquant à la plateforme une configuration du produit désiré, c.-à-d. en sélectionnant les artefacts des caractéristiques mentionnées dans la configuration du produit.

Modèle des caractéristiques

L'emploi des caractéristiques pour décrire les produits peut s'appliquer pour la description de la variabilité de la plateforme configurable, et donc de la LPL. La plus souvent, la variabilité est décrite par des modèles de variabilités, le plus commun étant donc le modèle des caractéristiques

Les modèles de variabilité, et notamment celui des caractéristiques (appelé *feature-model* en anglais), décrivent l'espace de la variabilité de la LPL en fonction des caractéristiques qui la composent. Le modèle des caractéristiques représente la variabilité comme une arborescence, où les caractéristiques sont des nœuds et où les branches sont des contraintes de variabilité. Les caractéristiques abstraites servent principalement à regrouper les caractéristiques concrètes (qui en sont des sous-caractéristiques) et donc à structurer le *feature-model*.

Ce type de modèle permet de décrire l'ensemble des configurations possibles que peut accepter la plateforme configurable, et donc l'ensemble des produits de la LPL. Les éléments graphiques qui composent le *feature-model* sont les suivants :

Dans l'élaboration de notre outil, nous considérons que les interactions de caractéristiques sont donc des comportements logiciels particuliers, implémentés dans la LPL et ses produits, qui interviennent pour assurer le bon fonctionnement d'un produit lorsque plusieurs caractéristiques peuvent s'influencer entre elles.

De plus, nous considérons que chaque interaction peut avoir une implémentation concrète et spécifique dans la plateforme de la LPL.

Cette implémentation prend la forme d'un ensemble d'artefacts dans la plateforme. Enfin, nous considérons qu'une interaction n'est présente dans un produit que si les caractéristiques en interaction le sont également.

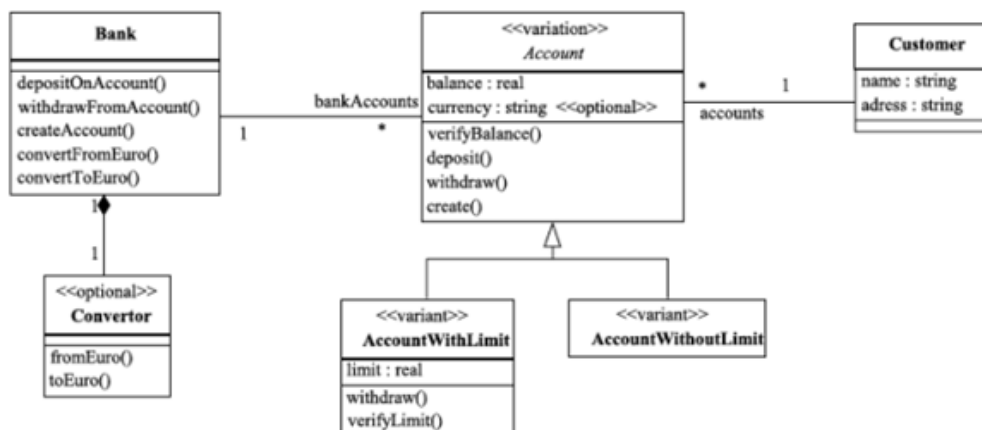
L'implémentation LPL par annotation et par composition

On distingue pour cela deux types d'implémentation possible.

Annotation

L'approche par annotation (ou approche annotative) hérite de son fonctionnement des systèmes de Macro de pré compilation, notamment populaire dans les langages C et C++

À l'image du code avec Macro, le code source d'une plateforme configurable contient des instructions spécifiques à un langage de précompilation, qui permet d'activer ou désactiver certaines portions du code source, afin d'inclure ou non le code spécifique d'une caractéristique. Une annotation consiste à ajouter un méta donné pour référencer une ou plusieurs parties de code. Chaque annotation doit être définie d'une manière à se déclencher pour une configuration donnée, après avoir exécuté une partie de code statique et commun. Les annotations peuvent être — implicites — explicites : Faite manuellement par un utilisateur. Le cas commun d'annotation explicite dans les langages de programmation c'est bien l'utilisation des blocs si et sinon (if else) à l'intérieur d'un programme, comme le montre la figure suivante.



Les annotations délimitent virtuellement le code de la plateforme pour lier les artefacts aux caractéristiques. L'annotation peut se comprendre comme une condition logique à satisfaire pour inclure ou non la portion de code annoté dans un produit. Chaque annotation déclare une formule logique où les noms des caractéristiques sont des propositions. En fonction de la configuration choisie pour un produit, un processus de précompilation parcourt le code de la plateforme à la recherche des annotations. Pour chaque annotation visitée, la formule de l'annotation est interprétée en considérant à "Vrai" les propositions correspondantes aux caractéristiques de la configuration. Si la formule renvoie "Vrai", le code annoté est conservé dans le produit, sinon il est retiré ou rendu

inactif par des commentaires

L'approche par annotation explicite les mécanismes de variabilité d'une plateforme configurable. Les concepteurs perçoivent en une seule vue l'implémentation de tous les artefacts variables, et donc de toutes leurs caractéristiques. L'implémentation des caractéristiques est dite virtuellement séparée, car elles coexistent toutes dans une même vue, mais elles sont séparées par des annotations. Avec l'approche par annotation, le code d'un produit généré conserve une grande proximité avec le code annoté dans la plateforme. Pour les concepteurs de la plateforme, cette génération permet de conserver le même code dans les produits que celui observé dans la plateforme. Cela facilite la maintenabilité de la plateforme et des produits. Le système d'annotations permet également de facilement implémenter les interactions de caractéristiques. Les interactions de caractéristiques correspondent à l'implémentation d'un comportement spécifique lorsque plusieurs caractéristiques sont présentes ensemble dans un même produit.

Pour finir, l'approche par annotations est indépendante du langage de programmation choisi et de la nature de l'élément à annoter. Elle peut d'ailleurs être appliquée à des fichiers de textes, des fichiers XML ou CSV, etc.

Limitations des annotations

La séparation virtuelle des implémentations peut engendrer des problèmes de compréhension lié à un manque de lisibilité du code de la plateforme.

Ces problèmes, dits de l'enfer des if/else, résultent d'une difficulté des concepteurs à interpréter la multiplicité des embranchements du code liée aux conditions des annotations

Cette difficulté s'accroît avec la taille des formules, c.-à-d. nombre de propositions dans la formule d'une annotation, et le nombre d'annotations au sein d'un même fichier. Il faut ajouter à ces difficultés deux autres particularités des annotations : elles peuvent être imbriquées pour implémenter la variabilité

Elles peuvent également être déclarées n'importe où dans le code, c.-à-d. à l'intérieur d'une méthode, autour du contenu d'un fichier, dans l'entête de la déclaration d'une classe, dans la signature d'une méthode, etc. Ce second point présente cependant un avantage, puisqu'il permet d'annoter finement les artefacts de la plateforme et donc d'annoter précisément l'implémentation d'une caractéristique. Enfin, le dernier point important est que la maintenabilité d'une plateforme annotée est différente de celle d'un code traditionnel. Les outils de vérification statique du code basé sur le parseur ou l'arbre de syntaxe abstrait (AST) sont désactivés ou restreints. Ces outils sont inadaptés, car ils ne sont pas conçus pour tenir compte de la variabilité de la plateforme. Par exemple, deux mêmes attributs déclarés plusieurs fois dans la même classe peuvent appartenir à deux annotations différentes (donc deux caractéristiques différentes).

En résumé, l'approche par annotation présente un système simple et intuitif pour implémenter les mécanismes de variabilité d'une plateforme. Toutefois, cette simplicité peut engendrer un manque de clarté du code source, et des difficultés de compréhension plus le nombre d'annotations augmentent (enfer des if/else). L'approche par annotation nécessite donc une certaine rigueur dans son application pour limiter des efforts de compréhension du code.

L'implémentation en compositionnel

L'approche par composition est souvent présentée comme une alternative aux limitations des approches par annotations. Son objectif principal est de résoudre la séparation virtuelle des implémentations en proposant une séparation physique. Ainsi dans l'approche par composition, les caractéristiques sont chacune physiquement isolées dans des dossiers qui sont propres à leur implémentation. Comme son nom l'indique, la composition propose de composer les différentes caractéristiques pour construire l'implémentation du produit lors de la génération. Pour cela, la composition s'appuie sur la structure du code ciblé.

Cela consiste finalement à composer une ou plusieurs parties de code. Le choix des parties de code, se fait avant l'exécution du code, en choisissant les parties à exécuter. Pour cela, on s'intéresse à la notion de point de composition, qui est une étape où on peut ajouter à une partie de code, une autre partie pour pouvoir exécuter les deux.

Avantages et inconvénients :

Il est notamment possible de comparer ces deux approches en termes de granularité, sûreté, traçabilité des produits pour choisir celui qui répond à notre besoin. Toutefois ces deux approches peuvent être combinées pour tirer le meilleur profit. Cependant, la figure suivante permet de distinguer la différence entre l'implémentation des deux approches.

L'approche par composition offre une séparation des implémentations pour chaque caractéristique, ce qui facilite la localisation de leurs implémentations pour les développeurs. Cependant, l'approche par composition est complexe et peu intuitive pour les développeurs. En premier lieu, elle introduit un nouveau paradigme de conception pour la plateforme, où chaque implémentation de caractéristiques doit être isolée. L'approche par composition demande donc plus de temps et d'efforts pour concevoir la plateforme que l'approche par annotation.

Deuxièmement, le moteur de composition des produits construit des produits dont l'implémentation est différente de celle de la plateforme. En effet, dans le cas d'une granularité Instruction, le code des méthodes peut être modifié pour réaliser une composition. Ces méthodes d'emballage modifient la structure du code source sans en changer la sémantique, mais cela peut contribuer à perturber la compréhension du code généré pour les concepteurs de la plateforme. Contrairement à l'approche par annotation pour un produit équivalent, le code généré s'éloigne de celui de la plateforme.

Troisièmement, bien que les compositeurs soient indépendants d'un langage de programmation, leur fonctionnement nécessite de disposer d'une représentation en arborescence du code source pour pouvoir être composés. De plus, ils doivent être adaptés pour chaque langage de programmation, afin de construire les règles de composition propres aux spécificités de chaque langage. Pour finir sur les limitations, l'approche par composition représente souvent difficilement une granularité fine. Ainsi à la granularité Instruction, dans la plupart des approches par composition, il est rarement possible d'introduire des instructions au milieu d'une méthode existante afin d'étendre une séquence. La manipulation d'instructions au milieu d'une méthode, de paramètres ou d'expressions est impossible dans aucune approche par composition, à cause des mécanismes de composition basés sur les emballages de méthodes.

Synthèse

Nous retiendrons de l'étude que l'approche par annotation est plus avantageuse que l'approche par composition, notamment concernant son intuitivité, la granularité permise, et son déploiement quasi

universel.

Les annotations (notamment dans le contexte du C/C++) ont une réputation négative et sont souvent critiquées pour rendre le code difficile à comprendre et à maintenir. Cependant, certaines expérimentations indiquent que ces critiques ne semblent pas avoir de réels fondements en pratique, puisque l'impact des annotations sur la maintenabilité semble minime comparé au gain précédemment

Le degré de granularité

La granularité est une notion récurrente dans l'ingénierie d'une LPL. De façon générale, la granularité définit la taille du plus petit élément qui compose un système. Elle désigne donc la limite du niveau d'analyse des éléments, niveau à partir duquel un élément est considéré comme insécable. Dans l'ingénierie des LPL, la granularité peut s'employer à différents niveaux, pour définir la finesse (ou la précision) avec laquelle doivent être déclarés les artefacts et donc la variabilité d'une LPL

Ainsi, la granularité de la variabilité donne le niveau de détails avec lequel la variabilité et les artefacts sont définis dans la LPL. Concernant l'implémentation de la variabilité et les artefacts de code source, différents niveaux de granularité existent. Nous proposons ici notre synthèse de ces différents niveaux :

Classe : définit une granularité qui considère l'intégralité d'un fichier de code ou d'une classe (en programmation orientée objet OO) comme le plus petit élément à considérer. Par exemple, la variabilité à cette granularité se fait en ajoutant ou retirant des fichiers dans le code source des produits.

Méthodes : définit une granularité qui considère les méthodes ou les fonctions qui composent respectivement une classe OO ou un fichier de code. Par extension, cette granularité considère également les déclarations d'attributs ou de variable relative à un fichier. Par exemple, la variabilité se fait en ajoutant ou en retirant une méthode dans une classe.

Instruction : définit une granularité qui considère les instructions du code source comme le niveau le plus fin. Cette granularité permet de considérer chaque instruction du corps d'une méthode ou d'une fonction comme un artefact individuel.

Expression : définit une granularité qui considère les opérande et opérateur d'une expression (plus fin qu'Instruction).

Signature : définit une granularité qui considère les paramètres d'une signature de méthode ou d'une fonction ou d'une classe. Par exemple en ajoutant ou en retirant l'expression qui déclare l'extension du type dans une classe. Plus la granularité est fine, plus elle précise l'implémentation des différentes caractéristiques dans la plateforme. Cette finesse permet également de mieux factoriser le code partagé par plusieurs produits et donc d'améliorer la maintenabilité de ces derniers. La même étude montre qu'il est donc plus pertinent d'adopter une granularité Instruction que des granularités Fichier-Classe et même Méthode-Fonction. Ainsi, plus la granularité est fine, plus le code de la plateforme configurable peut être factorisé, maximisant ainsi le code commun entre les produits de la LPL.

Localisation des caractéristiques

Dans le cadre des lignes de produits logiciels, la localisation de caractéristiques (ou Feature

Location en anglais) consiste à retrouver l'implémentation d'une caractéristique connue, c'est-à-dire à lui associer des artefacts parmi ceux de la LPL. La localisation est à distinguer de l'identification des caractéristiques feature identification, qui consiste à déterminer la caractéristique à associer à des artefacts. Les techniques de localisation s'appliquent généralement dans deux cas : sur un seul produit, ou sur une famille de produits.

Dans les deux cas, associer des caractéristiques à des artefacts consiste à lier l'ensemble des artefacts à l'ensemble des caractéristiques. Plusieurs hypothèses sont possibles concernant ces associations. Par exemple, un artefact peut être associé seulement à une caractéristique ; ou il peut être associé à plusieurs caractéristiques. Il faut également considérer la présence d'interactions de caractéristiques dans l'implémentation des produits. Dans nos travaux au sein de l'entreprise, nous nous intéressons principalement à la notion de localisation des caractéristiques dans le contexte des lignes de produits logiciels. Dans ce contexte, nous rencontrons le plus souvent la localisation dans les stratégies extractives. (Cette étape n'est pas encore terminée dans notre implémentation mais elle fera l'objet de modification en fin septembre 2019 par mes soins)

Les techniques de localisation existantes pour des artefacts de code source sont classées en trois catégories : analyse dynamique, analyse statique et analyse textuelle. Certaines techniques proposent parfois de combiner ces types d'analyse pour améliorer leur performance.

Analyse Dynamique

L'analyse dynamique consiste à localiser l'implémentation des caractéristiques lors de l'exécution des logiciels. Cette analyse s'appuie sur une instrumentation du code source des produits pour produire des traces d'exécution. Les scénarios d'exécution du programme sont nécessaires et construits en amont de l'exécution, en accord avec l'instrumentation du code source.

Il existe plusieurs techniques de localisation basées sur l'analyse dynamique (voir inventaire complet dans

Parmi les premières proposées figurent celles de Wilde (1995). Leur technique propose d'exécuter deux types de scénarios à la suite : l'un activant une caractéristique, et l'autre où la caractéristique n'est pas activée. Les traces d'exécution sont comparées pour déterminer les artefacts qui sont les candidats les plus probables pour être l'implémentation d'une caractéristique spécifique. La localisation basée sur l'analyse dynamique des traces est limitée par la construction des scénarios choisis. En effet, rien ne permet d'affirmer que l'exécution des scénarios suffise à couvrir l'ensemble des artefacts associés à une caractéristique.

Analyse statique

L'analyse statique se focalise sur une exploration du code source et des ressources ou documents associés à un logiciel pour déterminer l'implémentation des caractéristiques parmi les artefacts d'un logiciel. Nous pouvons distinguer deux types d'analyse statique : sur un seul produit et sur un ensemble de produits.

L'analyse statique sur un seul produit consiste essentiellement à regrouper les artefacts du code source sur l'hypothèse qu'il existe une cohésion parmi eux. Cette cohésion peut être structurelle ou lexicale. Dans le cas d'une cohésion structurelle, l'analyse statique considère les dépendances structurelles (p. ex. méthode définie dans une classe ; fichiers définis dans un paquet) ou de

références (p. ex. la méthode fait référence à un attribut ; la classe contient un attribut dont le type est défini dans une autre classe).

Pour faciliter le regroupement des artefacts, la localisation est parfois semi-automatique, et il est attendu d'un expert qu'il indique une graine ; dans le code à partir de laquelle la localisation peut commencer. Cette graine peut être une méthode, une instruction, une classe, ou tout autre élément du code.

Parmi ces approches de localisation sur un seul produit, Chen (2000) a proposé une approche semi-automatique. Leur approche construit un graphe de dépendances à partir des éléments du code source dont les liens sont des invocations de méthodes, références d'attributs, relation d'héritage. À partir du graff, un expert sélectionne un nœud en tant que graine (étiquetée avec le nom de la caractéristique à localiser) à partir de laquelle la localisation va se propager pour atteindre les différents nœuds et obtenir une implémentation pour une caractéristique donnée.

L'analyse statique sur une famille de produits est moins étudiée que celle sur un seul produit. De façon générale, les approches proposées se basent sur la variabilité des produits pour isoler l'implémentation des caractéristiques (car on connaît les caractéristiques associées à chaque produit. Pour cela, les approches font l'hypothèse de la cooccurrence entre les caractéristiques décrites dans les produits et les artefacts qui composent ces produits. Ainsi, elles associent artefacts et caractéristiques sur le postulat que si un groupe d'artefacts est toujours présent dans un produit lorsqu'une caractéristique l'est, alors ces artefacts doivent correspondre à l'implémentation de la caractéristique.

Les approches de localisation par analyse statique sont nombreuses et présentent toutes leurs limites et leurs forces. Nous pouvons généraliser en disant que les techniques appliquées à un seul produit sont souvent semi-automatiques et utilisent des graines pour propager la localisation. Quant aux techniques appliquées à une famille de produits, elles sont souvent automatiques, mais basées principalement sur l'étude de la variabilité. De plus, ces approches ne prennent souvent pas en compte les interactions de caractéristiques

Résumé LPL

L'ingénierie des lignes de produits logiciels est donc un ensemble de méthodes et de techniques qui tente de systématiser et d'encadrer la conception d'une famille de logiciels. Ses deux principes fondamentaux sont la promotion des principes de personnalisation de masse des logiciels et de réutilisation systématique.

Pour réaliser ces deux principes, l'ingénierie des LPL propose la conception d'une plateforme logicielle configurable, capable de varier en fonction de caractéristiques logicielles que les utilisateurs sélectionnent pour aboutir au produit désiré (une personnalisation de masse). Les produits (logiciels) d'une LPL sont donc tous construits depuis la même plateforme, ils partagent ainsi une partie de leur implémentation.

La notion de variabilité est intrinsèque aux LPL. Leur variabilité est généralement modélisée par un modèle de caractéristique (feature-model en anglais). Ce modèle met en relation les différentes caractéristiques d'une LPL par rapport des contraintes de variabilité (par exemple, implications, exclusion, co-occurrences, ou autres conditions).

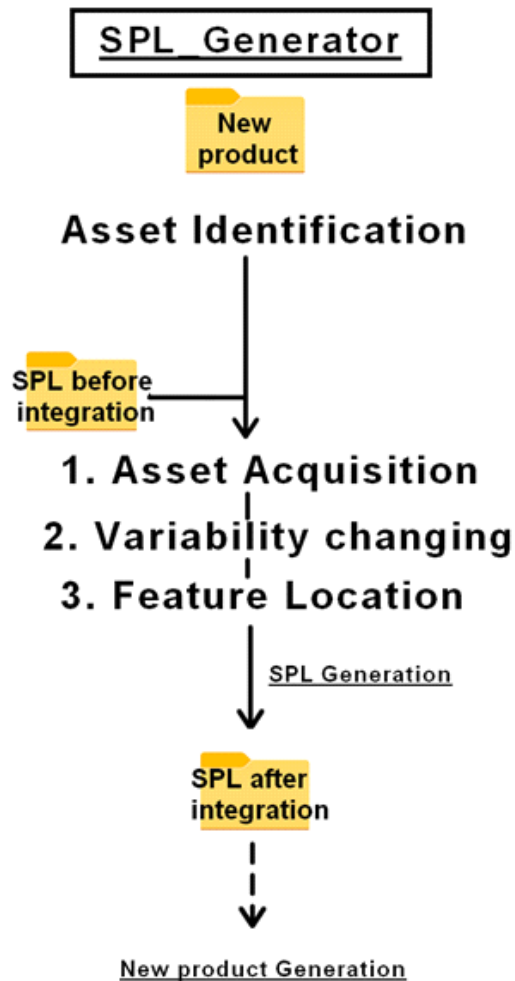
Chaque caractéristique dispose d'une implémentation spécifique dans la plateforme d'une LPL, sous la forme d'un ensemble d'artefacts. Les artefacts sont des éléments de nature hétérogène (éléments de code, fichier texte, base de données, ressource sonore, etc.) et sont souvent décrits avec une granularité définie par les concepteurs de la LPL (Fichier-Classe, Méthode-Fonction, Instruction, etc.).

La plateforme configurable regroupe et structure l'ensemble des artefacts de sorte qu'une sélection des caractéristiques puisse permettre de générer le produit correspondant. Cette génération par sélection est rendue possible par l'implémentation de mécanismes de variabilité dans la LPL. Il existe deux types d'approches pour implémenter la variabilité de la plateforme, l'approche par annotation et celle par composition. Si ces deux approches possèdent leurs forces et leurs limitations, il faut noter que l'approche par annotation reste plus accessible et intuitive pour les concepteurs que l'approche par composition ; elle permet aussi une implémentation avec une granularité plus fine.

Enfin, l'ingénierie des LPL dispose de nombreux avantages face à une ingénierie monoproduits (c.-à-d. Logiciels individuels). Grâce à la réutilisation systématique, les LPL permettent une réduction des coûts de développement, une réduction du temps de mise sur le marché des produits, une diminution de l'effort de maintenabilité et une amélioration de la qualité des logiciels. Cependant, ces bénéfices n'apparaissent qu'à partir du troisième produit mis sur le marché, du fait d'un investissement initial important à fournir pour les LPL.

Nous allons désormais vous présenter notre outil SPL Generator : ce dernier est dans l'étape finale de conception. Débuté en Novembre 2018, SPL Generator a pour but de faciliter l'adoption des Lignes de produits logicielles en entreprise en offrant aux développeurs une facilité d'exécution.

SPL generator



THOMAS PETIT

Les objectifs de l'approche

Notre l'approche SPL Generator, est une approche réactive automatisé. Elle est schématisée dans la figure ci-dessous. C'est donc une approche construite sur un cycle en trois phases : L'intégration, la génération et la maintenance de la LPL.

Nous proposons l'application de processus automatique pour réaliser l'ingénierie d'une LPL afin de réduire ses coûts. Notre hypothèse est que retirer l'effort lié à l'ingénierie manuelle permettra une meilleure adoption des LPL par les entreprises. Notre outil peut faciliter plus la construction d'une LPL puisque les risques liés aux coûts d'ingénierie sont réduits par l'automatisation des tâches. Cette réduction des coûts doit être accompagnée d'un effort d'adoption faible de l'approche, notamment pour limiter la résistance aux changements des développeur.

Dans le chapitre précédant, nous avons exposé des différentes approches d'adoptions des Lignes de Produits Logiciels. Nous avons discuté des avantages et inconvénients de chacune d'entre elle, pour finalement retenir l'approche réactive comme étant la plus adapter aux métiers des développeurs et à celui des entreprises. Toutefois, nous avons mis en avant l'importance de proposer une automatisation de l'approche réactive, afin d'en réduire ses coûts. Ces coûts sont directement à la complexité de la réingénierie d'une Lignes de Produits. Mais également, nous avons fait le constat que les développeurs d'aujourd'hui préfèrent travailler sur une conception traditionnelle de logiciels, plutôt que sur une conception en vue d'incrémenter une Ligne de produits. Dans cette partie nous présentons notre outil : « SPL Generator », d'une approche réactive automatisé pour la création d'une Ligne de Produits Logiciels. Nous évoquerons les différentes étapes notre approche, de l'intégration à la génération de produits. Nous discuterons également des objectifs et des difficultés à surmonter pour réaliser cette approche. Cette partie répond à la question suivante : Comment construire une approche réactive automatique adapter aux développeurs ? Ces trois phases coïncident trois grands scénarios d'utilisation :

Notre premier objectif est de s'appuyer sur les compétences existantes des développeurs de l'équipe IT, à savoir la conception efficace de logiciel en un minimum de temps et d'énergie.

Cette compétence est maîtrisée et recherchée par les entreprises, qui favorise un développement rapide. Notre objectif vise donc à utiliser en entrée de la réingénierie les produits conçus par les développeurs : il s'agit du code source du logiciel. Cela confère à notre approche une meilleure conservation et une intégration aux processus de développement déjà en place dans les entreprises. Mais également, cela permettra de ne pas nécessiter l'acquisition de nouvelles compétences pour les développeurs et donc formations pour les entreprises. Le second objectif est la transparence de l'approche : son utilisation doit se faire en marge du développement des produits, afin de ne pas le retarder. Le principal défaut de l'adoption de l'ingénierie des LPL rapporté par l'industrie est l'investissement de départ pour la conception d'une ligne. Cet investissement perturbe les délais de mise sur le marché des entreprises. En souhaitant faciliter l'adoption de l'approche réactive, nous devons accepter ce constat et proposer une approche qui peut être déployée sans perturber les délais de conception déjà en place dans les entreprises. En claire, notre objectif est d'avoir un impact positif sur le temps de mise sur le marché, et non négatif, et cela dès les premiers produits.

Il faut que cette génération automatique de la plateforme produise un résultat qui soit compréhensible par le plus grand nombre de développeurs, afin de renforcer le potentiel de

maintenabilité de la LPL. C'est pourquoi dans notre approche, nous choisissons de construire une plateforme avec une représentation Annotative, car elle est réputée comme intuitive pour les développeurs.

Comme annoncée précédemment, la construction de la LPL se fait au travers des produits. Dans notre approche, la LPL est utilisée pour faciliter le développement de nouveaux produits : au début d'un nouveau projet, on consulte la LPL pour générer un produit partiel, le plus proche possible des besoins exprimés. Une fois complété, ce produit sera intégré automatiquement dans la LPL et les nouveautés ou modification qu'il introduit seront ajoutés de façon disciplinée dans la LPL. Ainsi, le développement de nouveaux produits ne dépend plus d'une réingénierie manuelle et obligatoire de la LPL. Les développeurs sont libres de concevoir leurs produits comme ils ont l'habitude de le faire, i.e. sans se préoccuper de la gestion de la variabilité, de l'implémentation des caractéristiques, de l'impact des nouvelles caractéristiques sur les anciennes ou éventuellement de leurs interactions négatives. Cela a pour autre avantage de faire participer les développeurs non familiers avec les LPL aux développements de nouveaux produits. Nous faisons ici l'hypothèse que cela permettra aux entreprises de réduire l'effort de formation de leurs développeurs, voir même de le supprimer. Dernièrement, cette pratique permet s'adapter au processus de développement déjà en place dans l'entreprise. On permet aux équipes de développement de conserver une grande partie de leur façon de travailler (comme le choix de l'IDE par exemple). Ce qui permet de construire la LPL sans impacter la productivité des développeurs.

Malheureusement, peu d'entreprises ont les moyens de construire l'ensemble de leur LPL en une seule traite. Il est de plus improbable que de nouveaux produits ne viennent pas s'ajouter à la LPL. Si l'on prend le cas des PME, ces entreprises fonctionnent de manière simple : elles ont un cycle de production réduit et occasionnel en fonction des mises à jours des logiciels nécessaires transmises par les maîtrises d'ouvrages.

Mon alternance

Lors de mon alternance, les équipes métiers transmettaient un cahier des charges sur les nouvelles spécificités à intégrer dans la gamme de logiciel. Les équipes IT devaient donc comprendre les mises à jour à implémenter et ensuite appliquer ces modifications dans le logiciel. Ainsi il apparaît nécessaire que le développement de la LPL soit compatible avec un cycle de production simple et occasionnel : on doit pouvoir construire et mettre en pause la l'ingénierie de la LPL à tout moment. Notre outil peut également être appliqué dans un contexte de réingénierie d'une famille de produits vers une LPL (extractif). Cela peut se faire facilement si on considère que ce sont les produits qui motivent la construction de LPL : disposer en amont de tous les produits ou les créer un à un avec LPL, ne doit pas changer le résultat de notre approche. Enfin, nous pouvons aussi faire l'hypothèse que notre approche est adaptée à une production proactive de la LPL. Cette hypothèse suppose des prérequis. Les développeurs doivent réaliser en amont l'ingénierie du domaine pour acquérir des connaissances sur l'ensemble des caractéristiques et la variabilité de la LPL. Ils doivent également réaliser l'ingénierie d'application afin de connaître l'ensemble des configurations de produits à réaliser. Dans ce cas-là, l'outil doit être applicable pour discipliner et accompagner l'ingénierie d'un LPL, en proposant de créer un à un les produits. La compatibilité de « SPL generator » avec les trois stratégies d'adoption (proactive, réactive, extractive) permet d'adapter notre approche à différents

profils d'entreprises. Ce qui accentue le potentiel d'adoption de notre approche

Voyons désormais l'étape d'intégration des produits, étape importante de la réingénierie

Les étapes de L'intégration

Nous définissons quatre étapes internes aux processus de réingénierie par Intégration de la LPL. Ces étapes sont invisibles pour les développeurs, et donc entièrement automatique. En premier lieu, la réingénierie commence par une étape d'identification des artefacts. Cette identification a pour but de déterminer quels sont les assets (ou artefacts) du produit. Rappelons que les assets (ou artefacts) peuvent être l'ensemble des éléments se trouvant dans du code source ; des images, du contenu audio, web, des fichiers de configurations. L'identification fait donc l'inventaire de tous les assets d'un produit. Ensuite, la seconde étape est l'acquisition des artefacts au sein de la LPL existante. Chaque asset issu du produit est comparé avec les assets déjà présents dans la LPL, afin de déterminer s'il s'agit d'un nouvel élément. Cela permet d'incrémenter au fur et à mesure l'ensemble des artefacts de la LPL. L'acquisition de ces assets veille également à préserver le comportement des produits. Vient ensuite la troisième étape, celle du calcul de la variabilité. Dans cette étape, la variabilité introduites par les artefacts et caractéristiques du produit vient modifier la variabilité de la LPL précédente. Cela permet de mettre à jour les modèles de variabilités qui seront présentés aux développeurs. Enfin la dernière est la localisation des caractéristiques parmi les artefacts de la LPL. Cette étape permet de tracer les liens entre les caractéristiques renseignés sur le nouveau produit, et les groupes d'artefacts qui seront désignés comme étant leurs implémentations, ou alors l'implémentation d'une interaction entre plusieurs caractéristiques.

Identification des assets et l'entrée des produits

La première partie du projet est consacrée à l'identification des artefacts du produit à intégrer. Nous y présentons notre solution concernant l'analyse à granularité fine d'un code source. Afin de répondre au problème de construction de l'ensemble des artefacts, nous proposons une identification automatique des artefacts depuis une analyse statique du code source des produits.

Cette identification à plusieurs objectifs :

Premièrement, le processus d'identification doit parcourir le code source d'un produit pour y détecter l'ensemble de ses artefacts. C'est-à-dire qu'elle identifie toutes les classes, interfaces, méthodes ou instructions du code source. Deuxièmement, le processus d'identification doit conserver la structure initiale du produit en structurant ses artefacts de façon à maintenir leurs ordres d'apparition des éléments ordonnés du code. Nous parlons notamment de conserver l'ordre des artefacts déclarés dans des séquences, comme p. ex. les instructions. Nous devons préserver ces séquences si l'on veut éviter de modifier le comportement du produit une fois intégré.

Troisièmement, l'identification doit permettre de décrire un produit sous la forme d'un ensemble d'artefacts. Cet ensemble d'artefacts est le socle sur lequel repose la construction des configurations d'artefacts d'un produit, et donc la création du contexte formel des artefacts

Nous avons dû faire face à de multiples enjeux et problématiques décrits ci-dessous

Problématiques lors de mon alternance

L'identification des artefacts est nécessaire pour construire la plateforme, et étudier la variabilité de la LPL. Son objectif est double : construire un ensemble d'artefacts de sorte à distinguer tous les artefacts dans un produit, et utiliser cet ensemble pour comparer les artefacts entre plusieurs produits. L'identification des artefacts est relativement simple si on considère une granularité Fichier-Classe ou Méthode-Fonction, mais l'identification se complexifie au niveau Instruction. Pour le comprendre, commençons par le cas d'une granularité Fichier-Classe. A cette granularité, pour identifier les artefacts dans un produit, il suffit de considérer l'ensemble de ses fichiers. Pour les fichiers, l'unicité des éléments s'obtient du fait que les systèmes d'exploitation rendent impossible la déclaration de deux fichiers au même nom déclaré dans un même dossier

Il n'existe donc pas deux fichiers qui partagent à la fois le même nom et le même chemin. On distingue s'ils ont des fichiers homonymes, car ils ont des chemins différents. Le même principe s'applique globalement aux classes. On s'intéresse ici à la signature d'une classe dans son namespace lors de sa déclaration pour identifier l'artefact. L'identification est souvent triviale, car les compilateurs refusent l'ambiguïté d'avoir plusieurs classes homonymes dans un même fichier. Le cas des sous-classes homonymes est différent, car elles sont deux classes déclarées dans des champs différents. L'unicité des artefacts au niveau Fichier-Classe est donc garantie par les règles du système d'exploitation et/ou du compilateur.

Au niveau Méthode-Fonction, on utilise la signature de l'élément déclaré (méthode ou fonction) pour identifier l'artefact. Globalement, les compilateurs empêchent la déclaration de deux méthodes (ou fonctions) partageant la même signature dans un même champ de déclaration. Dans le cas de deux méthodes partageant la même signature (par exemple le polymorphisme), dans deux classes différentes, on utilise leurs chemins dans l'AST pour les distinguer

L'entrée des produits

L'intégration est donc le point d'entrée de notre approche. Son principe est de prendre en entrée un produit, accompagné d'une liste des caractéristiques implémentées dans celui-ci. La phase d'intégration a pour objectif de décomposer un produit en artefacts pour les ajouter à la LPL. L'intégration repose sur deux étapes : l'identification des artefacts et leurs acquisitions au sein de la plateforme. Ces deux étapes sont effectuées automatiquement à partir d'une analyse statique du code source du produit. Nous faisons l'hypothèse simplificatrice que le code source contient à lui seul l'implémentation complète des caractéristiques d'un produit. Cette hypothèse facilite notre conception pour SPL Genarator car elle restreint l'analyse de la variabilité aux artefacts de codes sources. Cependant, l'hypothèse déforme la réalité car l'implémentation d'une caractéristique peut dépendre d'éléments extérieurs au code (des images, du texte, des bases de données).

La phase de réingénierie prend comme première entrée un produit (le code source d'un logiciel conçu par les développeurs). Dans la pratique, cela revient à passer à notre outil SPL Generator, le projet du produit : son code source, mais aussi ses ressources. En somme, l'ensemble des éléments logiciels constitutifs du produit. Son principe est de prendre en entrée un produit, accompagné d'une liste des caractéristiques implémenté par celui. La phase d'intégration a pour objectif d'intégrer ce produit et ses caractéristiques dans la LPL. En d'autres termes, l'intégration automatiquement réalise le processus de réingénierie

L'entrée des caractéristiques du produit

D'autre part, une seconde entrée correspond au nom des caractéristiques implémentées dans le produit. Ces dernières prennent la forme d'une liste de noms, et sont renseignés par les développeurs. En renseignant manuellement la liste des caractéristiques implémentés dans un produit, les développeurs réalisent une analyse partielle du domaine dans la LPL. Cette analyse peut se réaliser par l'étude du cahier des charges associés au produit. Toutefois, cette tâche est loin d'être triviale et dépend principalement de la capacité des développeurs à faire émerger du cahier des charges un ensemble de caractéristiques. Or, nous pouvons noter un manque de standard dans l'industrie et donc de formalisme sur la tenue de ces cahiers des charges. Ainsi certaines pratiques spécifiques aux entreprises, viendront faciliter ou non la réalisation de l'analyse du domaine par les développeurs de l'équipe IT après mon départ de l'entreprise.

L'identification des assets du code source du produit.

Les nouveaux produits qui sont soumis à la LPL prennent la forme de d'un arbre syntaxique. Abstrait (Abstract Syntax Tree en anglais). Ces arbres sont composés des artefact ou asset du nouveau produit à intégrer. la première étape consiste à identifier les artefacts de ces produits à partir de leur AST. L'identification des assets (ou artefacts) vise à trouver tous les assets du code source d'un produit logiciel qui peuvent être intégrés dans une LPL. Cette technique peut être appliquée à n'importe quel langage possédant un arbre syntaxique

L'étape d'identification permet de distinguer l'ensemble des artefacts qui compose un produit.

Un asset (ou artefact) a une valeur, qui représente le code source de l'élément logiciel qu'il représente. L'identification crée un artefact pour chacun des nœuds principaux de l'AST. Les artefacts identifiés sont stockés dans une variable regroupant l'ensemble des Assets formant une structure d'asset (SA) sous forme d'arborescence.

A la manière d'un Arbre Syntaxique, cette SA stocke les assets en suivant une hiérarchie de parent-enfants. Ainsi, une SA ressemble à l'arbre syntaxique d'un langage ciblé.

Pour identifier ces artefacts depuis le code source, nous passons donc par sa représentation sous forme d'un arbre de syntaxe abstraite. Cela évite de devoir dépendre de la mise en forme du code et l'arborescence est simple à naviguer à l'aide d'un patron de conception. Un parcours en profondeur permet de construire une structure de donnée formant un arbre à partir des caractéristiques des assets. En fonction du nœud visité dans l'Arbre syntaxique, un artefact est créé et ajouté à la SA.

L'artefact considéré ici est un élément de l'AST. Il peut être la déclaration d'une classe, d'une méthode ou encore d'une instruction.

Chaque artefact est défini dans un modèle des artefacts, modèle qui est entendu depuis un modèle des artefacts générique. Nous détaillerons ces modèles dans la section. Les assets ont tous un identifiant différent à l'intérieur d'une SA. Ces identifiants sont essentiels pour éviter l'acquisition d'artefacts déjà intégrés dans la LPL, et ainsi éviter la redondance, lié à la duplication de l'implémentation de fonctionnalités.

Lorsque nous intégrons des artefacts à une implémentation SPL, nous devons nous assurer que nous n'ajoutons pas d'artefacts déjà intégrés et ainsi éviter d'ajouter l'implémentation de la même fonctionnalité. Ceci afin d'éviter l'effet indésirable de devoir maintenir deux versions de la même fonctionnalité dans une implémentation SPL. Par conséquent, il faut que chaque artefact soit unique dans une implémentation SPL. Cette unicité doit être garantie lors de l'acquisition de nouveaux

artefacts, en comparant les artefacts d'un nouveau produit avec ceux qui se trouvent déjà dans la SPL. Par exemple, les artefacts de classes, de méthodes et de champs sont facilement distinguables car ils ne sont déclarés qu'une seule fois par fichier. Cependant, avec notre objectif de granularité fine, l'unicité des artefacts doit prendre en compte le cas particulier des déclarations, qui peuvent apparaître plusieurs fois dans un produit. Ainsi, le premier problème est de conserver l'unicité de tous les artefacts pendant la réingénierie de la SPL, c'est-à-dire la phase d'intégration. La première question est donc la suivante : **comment garantir cette unicité des identifiants dans la SA (Structure d'Asset) ?**

L'unicité des Identifiants

Ces identifiants vont nous permettre de comparer les artefacts présents dans deux LPL. C'est à partir de cette comparaison que de nouveaux artefacts seront acquis : en comparant les deux structures d'asset de la LPL avec ceux d'un produit. Par exemple, nous pouvons comparer les deux ensembles d'artefacts de la SA et celui du nouveau produit pour trouver quels sont les nouveaux artefacts du produit. Par conséquent, nous savons qu'il existe une seule instance de chaque artefact, c'est-à-dire qu'il n'y a pas de duplication, si aucun artefact ne partage le même ID dans l'implémentation finale de la LPL. L'ID d'un artefact est calculé comme la clef de hachage de la valeur de l'artefact, ajouté à l'ID de son parent. Cette construction récursive des ID, qui démarre de la racine de la SA, garantit que chaque artefact a un ID unique dans une Structure d'Asset. De plus, nous pouvons étendre cette propriété d'unicité des ID à l'ensemble du produit. En effet, l'artefact de l'unité de compilation reçoit une valeur qui correspond au chemin relatif du fichier réel, à l'intérieur du produit. Comme deux fichiers différents auront nécessairement un chemin relatif différent, deux racines de la SA ne peuvent pas partager le même ID. Avec cette construction des ID, seuls les artefacts qui partagent la même hiérarchie parentale et qui ont la même valeur, partageront le même ID. Par exemple, cela peut se produire si deux déclarations de la même méthode partagent le même code. Si un produit possède deux artefacts identiques avec le même ID. Pour les différencier, nous ajoutons spécifiquement pour les artefacts de déclaration un autre paramètre pour construire leurs ID. Ce paramètre est défini comme étant le nombre de jumeaux qui précède l'artefact, plus un. Dans notre exemple, cela est symbolisé par l'indice donné à l'artefact.

Ainsi l'étape d'acquisition des artefacts se présentent comme une fusion entre les SA d'un produit, et ceux d'une LPL. Cette étape a pour objectif d'ajouter à la LPL tous les nouveaux artefacts introduit par le produit. En d'autres termes, l'acquisition doit ajouter au bon endroits les artefacts du produit dans la Structure d'Asset d'une LPL et éviter d'ajouter une seconde fois un artefact déjà présent dans la LPL.

Cela est essentiel pour construire correctement la plateforme, et éviter la génération de produit erroné.



Acquisition des artefacts

Une fois les artefacts identifiés sur les nouveaux produits, il faut les intégrer dans la LPL et fusionner ces nouvelles fonctionnalités c'est tout l'objet de cette seconde partie nécessaire à la construction de notre outil SPL Genrator

L'acquisition est donc l'étape d'ajout des artefacts identifiés dans la ligne de produits.

L'objectif de l'acquisition est ainsi d'intégrer à la LPL l'ensemble des artefacts spécifiques aux produits, tout en respectant les critères suivants : l'absence de redondance ; la préservation des produits déjà intégrés ; et la préservation du produit que à intégrer. Pour réaliser cela, l'acquisition s'effectue en considérant les deux arbres d'artefacts : celui identifié depuis le produit, et celui de la plateforme d'une LPL.

L'intégration se charge également d'actualiser la représentation interne d'une LPL. Chaque élément d'un produit fait l'objet d'un artefact. Nous adoptons pour l'ensemble de ces artefacts une structure de donnée sous la forme d'une arborescence. Comme vu précédemment tous les assets peuvent être stocker dans la Structure d'Asset (SA) L'essentiel à retenir ici est qu'une SA peut-être est augmentée de nouveaux assets à chaque intégration, si le produit en contient de nouveaux.

Pour acquérir les nouveaux artefacts d'un produit et les intégrer, nous devons fusionner deux ensembles : d'une part l'ensemble des Sas du produit ; et d'autre part l'ensemble des SAs déjà présents dans la SPL. Pour fusionner ces deux ensembles, nous formons des couples de Structure d'Asset : une SA provenant du produit et un de la LPL, s'ils correspondent au même fichier. Pour tout nouveau fichier introduit dans le produit, les SAs correspondantes ne correspondront pas dans la LPL, ils sont donc ajoutés sans fusion.

Pour cela il existe des algorithmes permettant de fusionner des Arborescences.

Algorithm 1: Algorithme de fusion de deux Structures d'Asset pour l'acquisition des artefacts

Il est nécessaire de préserver la séquence des assets à l'intérieur de la SA. Puisque deux SAs peuvent contenir deux séquences d'instructions différentes, leur fusion doit assurer que la SA de sortie préserve ces séquences, sans doublons. Nous proposons de construire dans la SA fusionnée une super-séquence qui couvre les deux séquences de la Structure d'Asset produit et de la SA LPL. Notre solution repose sur l'algorithme Longest Common Subsequence (LCS) , mais une présentation complète de LCS sort du cadre de cette thèse. LCS calcule la plus longue séquence d'éléments qui sont communs aux deux séquences. Nous présentons notre algorithme de superséquence basé sur LCS. Tous les artefacts nouvellement ajoutés sont mis en évidence en gris. Cependant, dans certains cas, il est obligatoire de dupliquer les artefacts pour construire.

Algorithm 2: l'algorithme Longest Common Subsequence (LCS)

```

Analysis_Procedure( G1, G2 )

    Length  $\leftarrow$  G1.length + G2.length
    Similarity  $\leftarrow$  LCS( G1.right, G2.right )  $\times$  2 / Length

    LCS( Attr1, Attr2 )
        If ( Attr1 == null or Attr2 == null )
            Return 0
        If ( Attr1.value == Attr2.value )
            Return LCS( Attr1.Right, Attr2.Right ) + 1
        If ( Attr1.value != Attr2.value )
            Return Max( LCS( Attr1.Right, Attr2 ),
                        LCS( Attr1, Attr2.Right ) )

    Level  $\leftarrow$  checkLevel( Similarity )    // Check the level of groups similarity

```

Une sous-séquence est la sous-séquence maximale d'une séquence donnée, où une sous-séquence est une liste consécutive et ordonnée d'éléments de l'ensemble initial.

Qu'est-ce que le LCS ?

Le problème de la plus longue sous-séquence commune consiste à trouver toutes les sous-séquences possibles d'une chaîne donnée ou, si elles sont finies, à calculer la plus longue. En général, de nombreux algorithmes fonctionnent bien pour trouver de longues séquences communes dans des chaînes de caractères dont les types de caractères sont aléatoires, mais dès que ces types de caractères deviennent suffisamment longs (100-150), il peut être très difficile de calculer des solutions optimales avec des algorithmes aléatoires qui ne se terminent pas après un certain nombre d'étapes. Parmi les autres problèmes informatiques qui peuvent avoir une complexité récursive similaire, citons la correspondance de chaînes de caractères et la recherche de similitudes DNF.

Bien que le problème n'ait pas de solution pour toutes les chaînes de caractères, il existe des algorithmes qui peuvent le résoudre efficacement pour de grandes classes de chaînes de caractères. Le plus efficace d'entre eux est l'algorithme de Hirschberg, mais il existe diverses améliorations et simplifications de celui-ci (par exemple, l'algorithme de Rabin-Karp).

Le problème de la plus longue sous-séquence commune est un cas particulier de la correspondance optimale des sous-séquences et de la construction d'arbres de suffixes. Si nous connaissons toutes les sous-séquences de deux séquences données, nous pouvons utiliser la programmation dynamique pour trouver la plus longue de ces sous-séquences. Puisque ces séquences ne sont pas connues dans leur intégralité, nous pouvons construire un arbre de suffixes avec la plus longue sous-séquence commune comme feuille, et itérer sur le sous-arbre.

Le problème de la plus longue sous-séquence commune est NP-dur pour toutes les chaînes de

caractères.

L'algorithme le plus efficace qui résout le problème de la plus longue sous-séquence commune pour les chaînes de longueur "n" avec une probabilité élevée (voir Algorithme) est appelé algorithme de Hirschberg (ou algorithme de Hirschberg), bien qu'il existe plusieurs améliorations basées sur la géométrie computationnelle qui rendent cette solution plus efficace. Il utilise à la fois l'algorithme glouton pour trouver les sous-chaînes et les arbres de suffixes pour trouver les sous-chaînes maximales. Lorsque les longueurs des deux chaînes sont éloignées l'une de l'autre, il peut être nécessaire de diviser la chaîne à un certain point et d'itérer en trouvant une sous-chaîne dans une chaîne qui contient un préfixe ou un suffixe de la partie. Cela peut être fait en un temps raisonnable dans la pratique, même lorsque la recherche d'une sous-séquence commune maximale n'est pas réalisable. Lorsqu'on travaille avec des graphes géométriques où les nœuds représentent des chaînes au lieu de sommets, on parle de problème de recherche d'arbre de suffixes de Hirschberg lorsqu'on travaille avec des graphes au lieu de chaînes. Dans ce cas, ce problème devient apparenté aux problèmes NP-complets en raison de sa similarité avec ces problèmes.

Pour en revenir à nos Structures d'Asset : Ces deux structures du même type sont appairées et fusionnées au sein d'une Structure de donnée mère, contenant un exemplaire de chaque artefact en provenance des deux SAs.

L'ordre des artefacts ordonnés est conservé au sein d'une super-séquence, qui est construite à partir d'un algorithme basé sur la plus longue sous-séquence commune. Cet algorithme de super-séquence nous assure de pouvoir retrouver les séquences issues des deux structures d'asset.

Notre fusion évite donc bien la redondance et la construction de super-séquences valides qui préserve le comportement de tous les produits. Pour finir, l'acquisition et les super-séquences peuvent mettre en lumière l'existence des artefacts dupliqués. Tout comme les artefacts jumeaux, les artefacts duplicata sont différents d'une redondance au sens où ils sont nécessaires pour la création d'une super-séquence valide, et donc d'une plateforme valide. Ces artefacts duplicata disposent de leur propre indice, l'indice duplicata, ce qui permet de les distinguer et de maintenir la distinction des artefacts.

Une fois l'acquisition réalisée, les structures d'Asset remplacent les anciennes de la ligne de produits. La LPL intègre désormais les nouveaux artefacts du produit, ce qui termine la première phase d'intégration de la réingénierie.

Les nouvelles structures d'Asset serviront à la génération du code de la plateforme configurable et à la génération de nouveaux produits pour la LPL.

Avec l'identification et l'acquisition, nous proposons une méthodologie pour identifier les artefacts d'un code source et les acquérir au sein de la LPL, afin d'en construire la plateforme. Comme nous l'avons décrit, ces étapes sont réalisées de sorte à minimiser la redondance du code intégré dans la plateforme.

Les modèles de variabilité et leur extraction

Dans les parties précédentes nous avons présenté l'identification des artefacts permet d'obtenir un ensemble cohérent contenant les artefacts d'un produit et comment l'acquisition des artefacts permet d'ajouter ces derniers aux arbres des assets (SA) qui composent notre représentation de la LPL.

À la suite de l'intégration se déroule l'analyse de la variabilité et la localisation des caractéristiques de la ligne de produits. À travers ces deux étapes, notre objectif est de construire des modèles décrivant la variabilité et les contraintes des caractéristiques (et artefacts). Ainsi que d'extraire des liens de traçabilité entre les caractéristiques et les artefacts qui forment leurs implémentations. Pour réaliser ces deux étapes, nous nous appuyons sur l'analyse de concepts formels (ACF), et son extension l'analyse de concepts relationnels (ACR)

Nous utiliserons également cette méthode pour la localisation de caractéristique(en plus d'autres méthodes tirée de papiers de recherches)

Nous définirons rapidement en quoi consiste un modèle de variabilité avec des exemples simples puis dans un second temps l'extraction des modèles de variabilité.

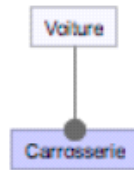
L'ACF est appliquée séparément, d'une part sur l'ensemble des artefacts, et d'autre part sur celui des caractéristiques, afin d'analyser la variabilité de ces deux ensembles et d'extraire leurs contraintes de variabilité

Nous présentons enfin succinctement l'ACR qui permet de réunir l'ensemble des artefacts et celui des caractéristiques afin de localiser l'implémentation des caractéristiques et de leurs interactions parmi les artefacts qui composent la LPL

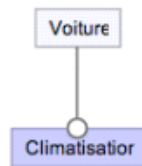
Qu'est-ce qu'un modèle de variabilité ?

La variabilité d'une LPL se visualise grâce à un modèle de variabilité. Il existe différent modèle. Le modèle le plus répandu est le modèle de caractéristiques (ou feature-model en anglais). Le Feature-Model représente une arborescence où chaque caractéristique occupe la place d'un nœud. Les branches du feature-model mettent en relations les caractéristiques, comme l'implication, mais aussi des exclusions, ou des alternatives.

Exemple : prenons l'exemple d'une voiture. Une voiture possède des éléments communs obligatoires. La carrosserie est donc une caractéristique obligatoire. Voici la notation adaptée pour le FM :

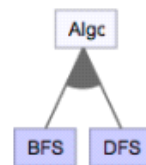


En revanche, une climatisation est optionnelle dans une voiture, la notation pour un feature Optionnelle se note

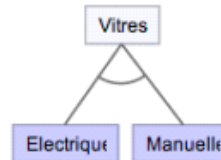


Nous pouvons avoir également une composition de feature, un choix, ou un choix exclusif

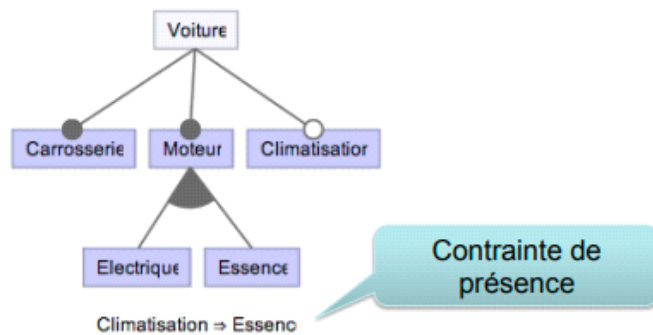
OR. Un choix



XOR. Un choix exclusive



Enfin nous avons également des contraintes de cohérence qui sont matérialisés par des dépendances de présence entre les features



Cette variabilité s'applique aux différentes caractéristiques découvertes et les développeurs ont pour objectifs d'obtenir les relations qui existent entre ces caractéristiques. Par exemple, ils doivent s'interroger les conditions d'utilisation d'une caractéristique particulier dans un produit. Cette conception de la variabilité s'accompagne de la réalisation de modèles de variabilité, comme des modèles de caractéristiques

Le processus d'extraction. Elle se consacre à la récupération de la variabilité des différents produits, en l'étudiant au travers des artefacts des produits. L'objectif est donc étant de différencier les artefacts communs à l'ensemble des produits, des artefacts spécifiques à seulement quelques-uns d'entre eux. La détection s'accompagne souvent d'une localisation de l'implémentation des caractéristiques (ou feature location en anglais), qui permet de rétroactivement faire l'ingénierie du domaine de la ligne de produit

L'étape d'analyse de la variabilité nous permet de découvrir la variabilité des artefacts et des caractéristiques, ainsi que les contraintes de variabilités (implications, cooccurrence et exclusion mutuelle).

Ces contraintes sont utilisées pour construire deux modèles de variabilités, celui des artefacts et celui des caractéristiques.

Nous avons également expliqué comment ces deux modèles pouvaient être employés par les développeurs lors de la maintenance de la LPL ou lors de la génération de nouveaux produits, c'est à dire pour la sélection des caractéristiques d'une nouvelle configuration.

Le processus décrit pour l'analyse de la variabilité adresse le défi présenté à la section

Comment peut t on extraire automatiquement la variabilité SPL et ses contraintes ?

Pour déterminer la variabilité des assets et des fonctionnalités, ainsi que leurs contraintes, nous appliquons l'analyse formelle des concepts (FCA)

FCA pour la variabilité

FCA travaille sur un contexte formel, qui est un ensemble d'associations entre objets et attributs. Chaque objet est décrit par un ensemble d'attributs qui le caractérisent. Par exemple, les produits

logiciels peuvent être l'objet, chacun d'entre eux ayant un ensemble d'artefacts comme attributs. La FCA construit un ensemble de concepts à partir d'un contexte formel. Un concept est un regroupement maximal d'objets qui partagent un ensemble minimal d'attributs. Un concept est constitué d'une intention et d'une extension. L'intention du concept donne l'ensemble de ses attributs, et son extension donne l'ensemble des objets qui partagent ces attributs particuliers. De plus, FCA établit un ordre partiel sur ces concepts, où un concept A est supérieur à un concept B si les attributs des objets dans B sont dans A, ou vice versa si les objets avec les attributs dans A sont dans B. L'ordre partiel forme une hiérarchie sur ces concepts, ce qui permet de construire un treillis de conception donne l'ensemble des objets qui partagent ces attributs particuliers.

	Base	F1	F2	F3	F4
P1	X	X	X	X	
P2	X				
P3	X	X	X		X
P4	X	X		X	
P5	X		X		X
P6	X			X	
P7	X		X	X	
P8	X	X		X	X

En sortie d'intégration, la modification de la variabilité

Après l'acquisition des artefacts, nous expliquons maintenant comment la variabilité est gérée :

Chaque SPL est caractérisée par sa capacité à varier afin de générer des produits différents. Il est donc nécessaire pour le fonctionnement d'une SPL de distinguer dans son implémentation les parties communes, de ses parties variables. Nous désignons ces parties variables par le terme points de variation. La connaissance des points de variation permet aux développeurs de comprendre la variabilité dans l'implémentation de leur SPL. De plus, cela leur permet de concevoir leurs produits, en sélectionnant les artefacts appropriés d'un point de variation à inclure dans un produit. Il est donc essentiel pour les développeurs de savoir où se trouvent ces points de variation et quels assets variables peuvent y être sélectionnés. Cependant, une vision binaire de la variabilité, c'est-à-dire variable, est insuffisante. Les développeurs doivent également connaître les contraintes entre la fonctionnalité (et donc entre les artefacts). Cela nécessite d'extraire les contraintes de variabilité nécessaires entre les fonctionnalités et entre les assets.

Le calcul de la variabilité dans notre Outils SLP generator

Pour déterminer la variabilité des artefacts et des fonctionnalités, ainsi que leurs contraintes, l'idée est d'appliquer l'analyse formelle des concepts (FCA)

Notre objectif est de construire des modèles décrivant la variabilité et les contraintes des assets.

Ainsi que d'extraire des liens de traçabilité entre les caractéristiques et les artefacts qui forment leurs implémentations. Pour réaliser ces deux étapes, nous nous appuyons sur l'analyse de concepts formels (ACF), et son extension l'analyse de concepts relationnels.

Nous effectuons l'analyse de la variabilité de la LPL. Cette étape a pour objectif de déterminer quels sont les assets communs et les assets optionnels de la LPL, et il en va de même pour les caractéristiques. Pour effectuer cette analyse, notre outil s'appuie sur l'Analyse de Context Formel. Cette méthode a été appliquée dans ce précédent travaux pour découvrir la variabilité et les contraintes entre les caractéristiques d'une famille de produits. L'analyse permet d'obtenir l'ensemble de savoir quel est l'ensemble des artefacts communs, et celui des assets optionnels (ou spécifique à quelques produits, mais pas tous). On peut faire de même pour les caractéristiques (partie suivante !)

Cette d'analyse automatique de la variabilité s'intègre dans notre outil afin de déterminer la variabilité des assets de la LPL puis de construire des modèles de variabilité à destination des développeurs, afin de représenter la variabilité.

L'objectif de cette section est donc de présenter notre processus d'analyse de la variabilité tel que nous le concevons dans pour l'automatisation.

Nous savons d'après les prérequis et l'études de l'art que la FCA permet d'extraire au moins trois types de contraintes : l'implication, l'exclusion mutuelle et la cooccurrence. Pour expliciter ces trois contraintes, prenons deux groupes d'éléments

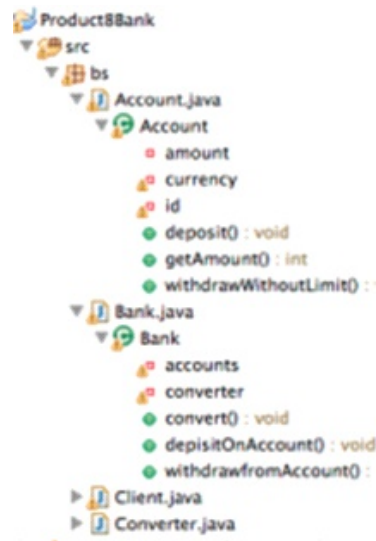
A et B L'implication indique quels sont les éléments A qui sont toujours présents dans un produit quand les éléments B sont présents. L'exclusion mutuelle indique que les éléments de A ne sont jamais présents en même temps que les éléments de B. Et enfin la cooccurrence indique que les éléments A sont toujours présent avec B, et réciproquement, pour n'importe quel produit où apparaissent A ou B.

L'analyse de la variabilité nous sert de deux manières lors de l'ingénierie automatique de la LPL : d'une part, elle permet de distinguer dans la plateforme quels sont les artefacts communs des artefacts optionnels. La même chose peut être effectuée pour les caractéristiques, ce qui permet de savoir lesquelles sont communes et lesquelles sont optionnels. D'autre part, l'analyse nous permettrait de construire deux modèles de variabilités : celui des artefacts et celui des caractéristiques. Ces deux modèles sont construits sur la base des trois contraintes extraites grâce à l'application de FCA. Pour les développeurs, l'intérêt vient surtout de l'utilisation du modèle de variabilité des caractéristiques. Il permet d'une part la sélection des caractéristiques lors de la génération de nouveaux produits. Par ailleurs, il offre une vue globale des caractéristiques de la LPL et de leur la variabilité, ce qui est utile pour la compréhension et la maintenabilité de la LPL

Les modèles de variabilités sont des diagrammes générés par notre outil. Ils permettent de rendre compte de la variabilité qui a lieu dans une LPL à une itération donnée. Puisque la LPL évolue à chaque itération, la réingénierie étant induite par l'aspect réactif de l'approche. Ces modèles évoluent également. La variabilité est automatiquement analysée depuis l'ensemble des produits passés en entrée de SPL Generator. Il existe dans notre outil deux types de modèles : le modèle de variabilité des artefacts, et le modèle de variabilités des features. Nous présentons ici les informations présentes dans le modèle de variabilité des artefacts, et comment elles peuvent être utiles aux développeurs :

1. la variabilité binaire des artefacts, qui désigne si un artefact est commun à l'ensemble des produits, ou s'il est spécifique à certains produits (donc optionnel du point de vue de la LPL).
2. L'implication entre les artefacts. L'implication est une contrainte qui désigne qu'entre deux groupes d'artefacts A et B, si A implique B alors les artefacts B doivent forcément être sélectionnés dans un produit si A est sélectionné. En d'autres termes, cela signifie qu'il existe une dépendance entre A et B, et qu'il est probable que modifier les artefacts de A est un impact sur le comportement des artefacts de B. Cette information est donc utilisable par les développeurs lors de la maintenance de leur LPL.
3. L'exclusion mutuelle entre les artefacts. L'exclusion est la seconde contrainte qui désigne que A ne peut pas être sélectionné si B l'est également, et inversement. En d'autres termes, il n'existe pas de dépendance entre ces deux groupes. Les développeurs peuvent donc modifier A sans craindre d'impacter le comportement de B, et inversement.
4. La cooccurrence des artefacts. La cooccurrence est une contrainte qui s'applique sur un groupe d'artefacts : Pour deux artefacts x et y, s'ils appartiennent aux mêmes groupes A, alors x implique y et inversement. Ils doivent toujours apparaître ensemble dans un produit. Ainsi en modifiant un artefact d'un groupe cooccurent, les développeurs doivent être vigilants et veiller à l'impact que peuvent avoir ces modifications sur l'ensemble du groupe. Le modèle des caractéristiques contient ces quatre mêmes informations, mais les présentes au niveau des caractéristiques de la LPL. Puisqu'il existe une traçabilité entre les assets et les features des deux modèles, les développeurs peuvent naviguer entre les informations de variabilité, représentée sur deux niveaux : celui des features et celui des artefacts. Cela facilite la compréhension logicielle en permettant de savoir quels sont les caractéristiques de la SPL, leurs contraintes, et quels sont les artefacts qui les implémentent.

Création du contexte formel (PCM) à partir des assets



L'application de l'ACF sur le contexte formel des artefacts permet d'obtenir un ensemble de concepts. Chaque concept regroupe des produits partageant un ensemble maximum d'artefacts. Les concepts des artefacts sont ensuite hiérarchisés pour construire l'AOC-Poset; et respectivement les concepts des caractéristiques sont construits et hiérarchisés, afin d'obtenir un AOC-Poset des caractéristiques.

Nous commençons par la variabilité des assets d'une SPL. Grâce aux ID uniques des artefacts, nous pouvons créer un contexte formel où nous garantissons que chaque ID représente un artefact spécifique dans les produits. Pour créer ce contexte formel, nous assignons à chaque produit une liste de l'ID de l'asset, qui correspond à l'asset identifié dans ce produit.

FCA construit ensuite **un treillis** à partir du contexte formel.

Extraction de la variabilité à partir du Treillis

Le but de cette partie est d'extraire les implications ainsi que les relations des contraintes à partir du treillis obtenu dans la partie précédente. Nous mettons en place la méthodologie décrite dans le papier de Al-MsieDeen (2014). L'extraction de la variabilité s'effectue donc depuis les AOC-Posets obtenues des artefacts.

Nous allons donc ici décrire l'extraction sur l'AOC-Poset des. Le treillis nous permet d'abord d'extraire la variabilité binaire (c'est-à-dire commune ou variable) : les artefacts communs.

Nous savons que l'AOC-Poset peut être exploité pour extraire la variabilité et ses contraintes. Dans un premier temps, la variabilité binaire des attributs peut s'extraire d'un AOC-Poset en considérant le concept de plus haut niveau dans la hiérarchie. Dans un AOC-Poset, ce concept sommet regroupe l'ensemble des attributs partagés par l'ensemble des objets. Puisque ces attributs sont partagés par l'ensemble des objets, alors ils désignent l'ensemble des attributs communs à tous les produits jusqu'ici intégrés dans la ligne de produit.

Le concept de sommet de l'AOC-Poset des artefacts contient dans ces attributs l'ensemble des artefacts communs.

Une fois l'ensemble des artefacts communs obtenu, on peut en déduire l'ensemble des artefacts optionnels, puisqu'il s'agit de tous les autres artefacts qui n'apparaissent pas dans le concept sommet. Ainsi, l'AOC-Poset nous permet donc de récupérer la variabilité binaire des éléments de la LPL.

Dans un deuxième temps, nous utilisons chacun des AOC-Posets pour extraire les contraintes entre leurs attributs. Nous savons que trois contraintes portant sur les attributs d'un contexte formel peuvent être extraites depuis l'AOC-Poset : l'implication, la cooccurrence et l'exclusion mutuelle.

Des algorithmes sont proposés pour récupérer ces trois contraintes de variabilité dans le papier de **Al-MsieDeen (2014)**.

L'étape d'analyse de la variabilité nous a permis de découvrir la variabilité des artefacts, ainsi que les contraintes de variabilités (implications, cooccurrence et exclusion mutuelle). Nous avons également expliqué comment ces deux modèles pouvaient être employés par les développeurs lors de la maintenance de la LPL ou lors de la génération de nouveaux produits, c'est-à-dire, pour la sélection des caractéristiques d'une nouvelle configuration.

Une fois cette étape d'extraction de la variabilité terminée il reste à localiser les caractéristiques qui viendront implémenter les assets

La localisation des caractéristiques.

L'étape de localisation des caractéristiques nous permet de tracer les caractéristiques aux artefacts qui ont été identifiés et intégrés à la LPL lors de l'intégration.

Comme vu dans l'état de l'art, cette localisation des caractéristiques donc consiste à associer chaque artefact à au moins une caractéristique et vice versa. En général, les techniques de Feature Location supposent que les caractéristiques de chaque produit sont connues.

Dans la première partie du mémoire d'alternance, nous avons discuté dans l'état de l'art des approches existantes appliquant une localisation de caractéristiques. Or beaucoup de techniques utilisées par ces approches étaient limitées par une granularité trop grosse (Méthode-Fonction) et par l'absence d'une localisation des interactions de caractéristiques. Ce faisant, l'application des techniques proposée par ces travaux est limitée, car les traces extraites ne pouvaient pas être employées pour reproduire les implémentations des produits initiaux.

Nous avons utilisé plusieurs méthodes issues de différents papiers, notamment par l'analyse textuelle préalablement définie dans la section de l'état de l'art. Nous recherchons une approche hybride pour notre projet mais cette étape n'est pas encore terminée à ce jour, car les résultats ne sont pas satisfaisants et par manque de temps. Nous planifions de terminer cette étape courant Septembre 2019.

Nos analyses basées sur la recherche textuelle pour la localisation des caractéristiques n'ont pas été probantes, donc nous allons donc chercher à mettre en pratique l'ACF, jadis utilisé pour les assets, mais cette fois dans l'optique d'une localisation de caractéristique. Comme dans le papier de **Al-MsieDeen**: *feature-location-in-a-collection-of-software-product-variants-using-formal-concept-analysis* (2014)

L'implémentation de la localisation des caractéristiques est toujours en cours à ce jour et nous ne pouvons pas donner encore de résultats précis quant aux approches que nous allons décrire par la suite :

L'analyse de concepts formels (ACF) pour réaliser la localisation.

L'intérêt d'ACF est que son application est indépendante de la nature des artefacts pourvu qu'un contexte formel puisse être construit. Ainsi, dans cette section, notre objectif est de formuler une technique de localisation basée sur l'analyse de concepts formels. Notre outil proposera une localisation à granularité fine pour les caractéristiques et de leurs interactions. De plus, en appliquant l'analyse de concepts formels à la localisation, nous réutiliserons les contextes formels, créés à l'occasion de l'analyse de la variabilité, ce qui donnera une certaine cohérence aux méthodes déployées dans l'automatisation.

L'étape de localisation des caractéristiques nous permet de tracer les caractéristiques aux artefacts qui ont été identifiés et intégrés à la LPL lors de l'intégration

Les contextes formels des artefacts et des caractéristiques, utilisés à l'étape d'analyse de la variabilité, sont utilisés pour construire un AOC-Poset relationnel. Les concepts de l'AOC-Poset relationnel introduisent à la fois des artefacts et des caractéristiques, et nous y appliquons nos algorithmes pour extraire des traces corps et des traces d'interactions. Ces traces seront ensuite transformées en formule logique puis réduite afin de simplifier leurs interprétations et

compréhension par les développeurs.

L'étape de localisation de caractéristiques extrait donc les traces des caractéristiques avec les artefacts de la plateforme configurable. L'implémentation d'une caractéristique est donc définie comme un ensemble distinct d'artefacts. La technique de localisation mise en place considère aussi bien la récupération des implémentations de chaque caractéristique, que celles des interactions entre caractéristiques. Ainsi grâce à la localisation, nous saurons quels sont les artefacts codant pour une caractéristique, ou pour une interaction spécifique. Le travail de maintenabilité des développeurs s'en voit facilité car les traces de la localisation permettent d'indiquer précisément où sont implémentés les caractéristiques parmi les artefacts de la plateforme configurable.

Conclusion et résultats préliminaires

Tout au long de mon alternance j'ai pu me familiariser avec les différentes approches des lignes de produits logiciel, la participation à la maintenance de ces outils m'ont permis de me familiariser avec les LPLs tout en ayant une vue plus complète des problématiques autour de l'automatisation de ces dernières. Ayant été dans la peau d'un informaticien en ligne de produit logiciel, j'ai pu ainsi mettre en pratique mes connaissances acquises à ce poste pour planifier cette automatisation. Notre approche a donc été guidée par la volonté de faciliter le travail des développeurs. Le développement et le suivi de la LPL était donc une de nos priorités dans la conception de SPL Generator.

Pourquoi l'outil est-il intéressant pour les développeurs ?

Il se positionne comme un processus qui s'insère dans un cycle de développement logiciel déjà établi. Il s'adapte selon une stratégie hydrique, mêlant l'itératif de la stratégie réactive et l'automatisation souvent rencontrée dans la stratégie extractive. SPL Generator permet de débiter la construction d'une nouvelle LPL selon une ingénierie dirigée par la conception de nouveaux produits, ou de transformer une famille de produits existant en une LPL. Le cycle de conception d'une LPL est itératif : lorsque le besoin d'un nouveau produit se présente, les développeurs peuvent réutiliser les artefacts d'une LPL en générant un produit partiel, à travers la sélection des caractéristiques qu'ils souhaitent inclure dans ce produit partiel.

Et enfin il permet ainsi aux développeurs d'appliquer une réutilisation systématique lors de la conception de nouveaux produits, tout en construisant la LPL de façon automatique.

L'automatisation réduit l'effort d'adoption puisqu'elle libère les développeurs des tâches de l'ingénierie du domaine et d'application, et leur permet de se concentrer sur le développement de leurs produits (tâche qu'ils maîtrisent déjà).

Le développement de l'outil SPL Generator a débuté en Novembre 2018, et il est aujourd'hui au stade de la finalisation. De nouvelles méthodes de features location devraient être mises en place et sont sur le point d'être terminées et testées sur notre outil.

Nous avons testé notre outil sur une version très simplifiée de nos logiciels afin de voir si l'intégration des produits ainsi que toute l'ingénierie fonctionnaient. Nous avons des taux proches de 100% pour l'intégration des produits pour nos premiers tests.

Comme vu dans les parties précédentes, les techniques de features location peuvent être améliorées cela sera l'objet de nos prochains travaux et améliorations de notre outils qui se déroulera durant les prochains mois.

Finalement notre outil SPL Generator s'appuie sur des techniques innovantes d'ingénierie LPL avec des contributions inédites dans notre façon méthodologie.

Une étude quantitative sur la réduction des coûts au sein de l'entreprise sera effectuée pour essayer de mesurer le gain de notre approche SPL Generator.

L'ingénierie manuelle des LPL -à travers des stratégies d'adoption que sont la proactive, la réactive et l'extractive- nécessite un coût d'investissement élevé, et l'emploi de développeurs experts pour concevoir la LPL. L'ingénierie manuelle est un investissement sur le long terme qui se trouve être

risquée pour les entreprises. Deuxièmement, les travaux existants qui tentent de réduire les coûts de cette ingénierie par l'automatisation souffrent de limitations qui font obstacle à leurs déploiements dans un contexte industriel. Leurs limitations concernent et impactent principalement la sortie du processus, à savoir la LPL obtenue. Notre outil SPL Generator permet un processus automatisé pour l'ingénierie des lignes de produits logiciels.

Durant cette alternance, j'ai pu construire un outil d'aide à la création de lignes de produits logiciel. J'ai pu détailler le fonctionnement général de mon outil SPL Generator et son insertion dans le contexte industriel des développeurs au sein de l'entreprise.

Notre processus s'applique aussi bien pour construire une nouvelle LPL selon une ingénierie dirigée par les besoins de nouveaux produits, ou bien pour migrer une famille de produits préexistante vers une LPL. De plus, après migration, les développeurs peuvent continuer d'appliquer notre outil pour ajouter de nouveaux produits. Le cycle de conception d'une LPL est itératif et se divise en trois phases : l'intégration, l'analyse et la génération. Nous avons présenté ces phases en détail dans les chapitres précédents.

J'ai pu tout au long de mon alternance, mettre en pratique des connaissances acquises au sein de mes études, tel que la programmation objet, et le management de projets.

J'ai acquis de multiples compétences dans le domaine de l'informatique, et plus particulièrement l'univers peu connu des Lignes de produits logiciels. Piloter ce projet m'a permis d'en apprendre plus sur le monde de l'entreprise mais aussi également sur moi-même et mon rapport aux autres.

Dès Septembre 2019, nous allons mettre en production les nouvelles méthodes pour la localisation de caractéristique et allons tester l'outil dans une plus grande ampleur avec à la clef de nouveaux résultats.

J'ai également eu le plaisir de signer un CDD de 1 an afin de continuer cette aventure pour améliorer encore cette automatisation.

J'aimerais encore remercier Laurent sans qui rien de tout cela n'aurait été possible.

Voici quelques ouvrages qui ont servis à préparer ce mémoire et à construire notre outil SPL GENERATOR :

M. M. Lehman. Laws of software evolution revisited.(1996)

ANTON KHRITANKOV : « INDUSTRIALIZING SPL DEVELOPMENT FOR SMALL COMPANIES » (2016)

Al-MsieDeen: feature-location-in-a-collection-of-software-product-variants-using-formal-concept-analysis (2014)

Ra’Fat Ahmad Al-Msie’Deen : Reverse Engineering Feature Models From Software Variants to Build Software Product Lines

AOC-posets: a scalable alternative to Concept Lattices for Relational Concept Analysis(Xavier Dolques, Florence Le Ber, and Marianne Huchard) (2013)

Yinxing Xue : Feature Location in a Collection of Product Variants (2000)

L. Bergroth :. A survey of longest common subsequence algorithms, (2000)

Magnus Eriksson. An Introduction To Software Product Line Development.