LIVRE BLANC | SEPTEMBRE 2015

Critique de la phase de test

ca technologies

Table des matières

Des pommes et des poires 3
L'impératif d'objectivité 3
Opportunité 3
Objectif 3

Principes de la phase de test 4
Transformation des informations 4
Mesurabilité et incertitude relatives aux informations 4

Méta-mathématique et méta-développement 6

Critique des méthodes de conception des plans de test 7

Défauts observables dans la logique 7

Méthodes combinatoires : rapides mais peu efficaces 8

Sur-spécialisation : méthodes de couverture multiplicatives 9

Introduction aux modèles formels 10

L'organigramme, ou l'outil de visualisation des exigences et des systèmes 11

Spécification de la logique pure - Graphique des causes et des effets 12

Comparaison relative 13

L'avantage de CA Technologies 14

Section 1

Des pommes et des poires

L'impératif d'objectivité

La rédaction de ce livre blanc a été motivée par la nécessité de fournir une analyse comparative des méthodologies de conception de plans de test. En compilant cette analyse, j'ai rapidement découvert qu'il n'existait aucun critère quantitatif objectif pouvant être appliqué à toutes les méthodologies de conception de plans de test utilisées. Il est vrai qu'il existe plusieurs critères qualitatifs subjectifs, récupérés par certains fournisseurs pour faire la promotion de leurs solutions. Le problème avec ces critères est qu'il est impossible d'en prouver les mérites. Tout passe par des anecdotes et des données, plutôt que de s'appuyer sur des principes de départ. Autrement dit, ces comparaisons sont faites a posteriori. En termes d'évolution des systèmes, l'analyse a posteriori a une limite indéniable : elle ne permet pas de prouver la progression a priori.

Solution

Pour obtenir des critères a priori, il a été nécessaire d'abandonner toute subjectivité et de procéder à partir d'une structure fondamentale qui permet une analyse objective. Cette étape a été capitale pour les mathématiques et la philosophie, puisque la création d'une structure basée sur des principes de départ (également appelée structure axiomatique) a libéré ces deux disciplines de la contrainte de prouver après coup le résultat à partir de données, leur permettant ainsi de réaliser un formidable bond en avant. Les testeurs sont les mieux placés pour réaliser la tâche délirante de vérification a posteriori, qui est une pratique courante dans leur domaine. Au lieu de tester des idées dès le début, il est courant d'exécuter des tests pendant et après la phase de mise en œuvre, mais cela revient à tester la résistance d'un bâtiment sans en vérifier les fondations !

Objectif

Ce document a pour but de définir une structure objective, qui sera divisée en trois concepts clés associés :

1. Utilisation des informations comme langage universel, réparties en données mesurables et non mesurables. Toutes ces étapes du cycle de développement logiciel (SDLC) sont considérées comme des informations présentées sous différentes formes.
2. L'incertitude modelée en tant qu'entropie d'informations : les défauts indécelables et logiciels impossibles à tester relèvent de cette catégorie.
3. Transformations des informations : de l'idée au concept et de la conception à l'implémentation. Le cycle de développement logiciel étant présenté sous des formats d'informations différents, les étapes intermédiaires sont modélisées en tant que transformations d'informations. Dans notre analyse, nous les appellerons des machines de Turing.

Lors de la compilation de cette analyse, il est vite apparu qu'il n'existait aucun critère quantitatif objectif pouvant être appliqué à toutes les méthodologies de conception de plans de test utilisées.

La plupart de ces notions relèvent du domaine de la mécanique quantique, qui peut être simplement définie comme la modélisation d'informations à un niveau très détaillé. Tout au long de ce travail d'analyse, le concept d'échelle d'observation sera largement employé, car, c'est un fait avéré : quelle que soit l'échelle à laquelle il est pris en compte, un système reste un système et seules les spécifications changent. Par exemple, il est courant pour un analyste métier de créer des exigences pour un système à différentes échelles : à un niveau élevé, puis à un niveau légèrement inférieur, etc., pour finir à un niveau très inférieur où sont fournis des détails d'implémentation précis. Autre exemple : il est fréquent chez les développeurs de mettre en œuvre un grand système en tant que collection de systèmes plus petits, reliés entre eux par des connecteurs pour que l'ensemble fonctionne comme un tout. Les exigences et les implémentations étant deux types d'informations pouvant être affichées dans des échelles différentes, il semble logique d'introduire la notion d'échelle dans notre façon de traiter l'information.

Section 2

Principes de la phase de test

Maintenant que nous avons établi le besoin d'objectivité, l'étape suivante consiste à définir les axiomes indispensables à notre analyse objective. Comme indiqué précédemment, les axiomes sont basés sur le concept d'information tiré de la mécanique quantique. Le concept abstrait d'information doit être envisagé comme une série d'instructions descriptives ou exploitables. Les informations descriptives doivent être simples à comprendre : l'information est le moyen par lequel nous décrivons une entité ou un objet donné. Par exemple, dans le cas d'un objet, nous utiliserions des adjectifs pour décrire ses caractéristiques esthétiques et des adverbes pour décrire la façon dont il influe sur le monde. Par ailleurs, les informations actives sont composées d'instructions qui expliquent comment construire un objet donné. Par exemple, une spécification contenue dans une instruction d'assemblage d'un meuble est considérée comme une information active.

Transformations d'informations

La différence entre les deux est assez subtile, mais capitale dans notre analyse, car nous n'étudierons que les informations actives, dans le cadre desquelles les exigences et la documentation de conception peuvent être assimilées à des instructions sur la manière d'assembler un système ou un élément logiciel. Étant donné que les systèmes et les logiciels peuvent être résumés sous forme d'instructions sur la manière de manipuler les données, nous devons les traiter, au même titre que les données, comme des informations. C'est l'un des concepts fondamentaux que nous utiliserons, c'est pourquoi il est vital d'en comprendre toutes les nuances. Cela donne lieu à notre deuxième concept fondamental, celui de la transformation des informations. En termes clairs, nous recevons les informations sous une forme et elles ressortent sous une autre forme. L'exemple le plus élémentaire est celui du développeur : un développeur transforme l'information sous forme d'exigence ou de conception d'un élément logiciel ou d'un système.

Il s'avère que l'ensemble du cycle de développement logiciel peut être comparé à une chaîne de formes d'informations différentes séparées par des transformations. Tous les morceaux du puzzle, des exigences au produit fini, peuvent être considérés comme des informations sous une forme ou une autre.

Mesurabilité et incertitude à l'égard des informations

Il n'y a pas si longtemps, nous considérions le concept de qualité, pour lequel nous devons évoquer deux autres concepts connexes : la mesurabilité des informations et l'incertitude à leur égard. En termes clairs, la mesurabilité de l'information est une mesure de la quantité d'informations pouvant être observées. Le fait que l'information existe est totalement indépendant de notre capacité à la capturer. Des informations non mesurables ne nous sont globalement d'aucune utilité ; cependant, il est important que nous ayons au moins une petite idée du volume de ce type d'informations. Nous aborderons la question des méta-informations (ou information de second ordre) dans une autre section. Pour le moment, concentrons-nous sur les informations de premier ordre. L'incertitude est un concept associé à la mesurabilité de l'information, appelé dans le cadre de notre analyse « entropie d'informations », dont la définition relève du domaine de la mécanique quantique et qui est à l'origine de formes d'informations diverses. Pour faciliter la conceptualisation de la différence entre des informations de premier et de second ordre, pensez à la différence qui existe entre des données et des métadonnées : les métadonnées servent à décrire des propriétés de données. Les méta-informations seront définies comme il se doit dans la section intitulée « Méta-mathématique et méta-développement ».

Il s'agit en fait d'une généralisation des ambiguïtés au niveau des exigences : les ambiguïtés entraînent des incertitudes au niveau de l'instruction, et différentes interprétations donnent lieu à plusieurs résultats possibles (ou implémentations de ces exigences). Si le testeur et le développeur optent pour deux interprétations différentes de la même exigence, il est très probable que les plans de test ne reflèteront pas l'implémentation.

Il reste une dernière série de termes à définir avant d'énoncer les grands principes ; nous devons distinguer deux types d'entropie :

- Épistémique : qualifie des informations cachées, au sens d'oubliées ; degrés de liberté perdue en raison du fait que l'information n'est pas présente. Dans notre univers, cela désigne des informations à « inventer » au cours du processus de transformation. C'est le cas, par exemple, lorsqu'un développeur doit combler les blancs lorsque l'exigence est trop floue ou incomplète. Il s'agit, en principe, d'une quantité mesurable. En pratique, cela nécessite une quantité de travail considérable : même lorsque les développeurs font des suppositions, cela transparaît toujours dans le code final. Voilà un exemple de ce que signifie le terme « réversible » dans le cas d'une transformation.
- Systémique : qualifie une entropie inhérente au système impossible à mesurer. Une entropie systémique est comparable au principe d'incertitude d'Heisenberg en mécanique quantique. Dans notre univers, l'entropie systémique est liée au fait que les instructions/systèmes d'axiomes ne sont jamais à la fois cohérents et complets. Il s'agit par conséquent d'une quantité non mesurable, mais à l'aide de la notion de taille d'ensemble relative, il est possible de quantifier précisément la quantité d'entropies systémiques présentes, étant donné que l'ensemble d'ensembles non mesurables est, contrairement à toute attente, mesurable.

Les principes fondamentaux du testing sont au nombre de six. Il s'agit de conséquences directes de la structure théorique des informations que nous avons établie. Ces principes peuvent être résumés comme suit :

1. Il n'est pas possible de tout tester. Il existera toujours une entropie systémique au sein d'un système. C'est la conséquence directe de l'incomplétude des systèmes d'axiomes de Gödel (ou de l'indétermination de Turing).
2. Il est toujours possible de savoir ce qui peut être testé. Une entropie épistémique peut être invalidée, si une quantité d'informations suffisante est disponible, et elle peut aussi être mesurée.
3. Les défauts observables représentent l'entropie mesurable d'un plan de test. Les défauts (ou l'absence de défaut) étant des indicateurs de qualité d'un élément logiciel, il est judicieux de les considérer comme des incertitudes. En particulier, les meilleurs plans de test sont ceux qui permettent de déceler un maximum de défauts. Pour relier ces indicateurs entre eux au sein d'une mesure, il est courant d'attribuer une sévérité aux défauts, ce qui permet d'obtenir une mesure sous forme de moyenne pondérée. Il faut toutefois rester prudent car une pondération arbitraire introduit un degré de subjectivité qu'il faut à tout prix éviter.
4. La couverture est la mesure de la fidélité de la stratégie/du plan de test. En d'autres termes, si la stratégie/le plan de test reflète précisément les exigences, tous les défauts observables seront alors décelables. Une stratégie/un plan de test mal ficelé(e) aura pour conséquence de ne pas voir des défauts qui, dans d'autres circonstances, auraient été observables.
5. L'entropie ne peut jamais être perdue. Ce principe relève de la mécanique quantique. Dans notre univers, cela signifie que si une étape quelconque du cycle de développement logiciel introduit une entropie, il n'est plus possible de l'éliminer même si toutes les étapes successives sont totalement fidèles. En particulier, la qualité d'un système logiciel est directement proportionnelle à la qualité des exigences et de la stratégie de test. Cela ne signifie pas que nos logiciels ne fonctionnent jamais, mais plutôt qu'ils ne sont jamais parfaits.
6. Aucun modèle n'est capable de détecter tous les défauts. Conformément au premier principe, il n'est pas possible de tout tester, mais l'association de plusieurs modèles permet de déceler différents ensembles de défauts. Néanmoins, la mise en évidence de tous les défauts requiert, au bas mot, un nombre infini de modèles. C'est une conséquence directe des théorèmes d'incomplétude de Gödel.

Section 3

Méta-mathématique et méta-développement

Dans notre discussion précédente sur les informations, nous nous sommes limités aux informations de premier ordre, à savoir les informations pures. Toutefois, il existe également des « informations sur les informations », surnommées méta-informations ou informations de second ordre. Les statistiques d'agrégation sont l'exemple type : par exemple, dans une population donnée, il est possible de glaner de nombreuses informations à partir de la taille moyenne (et de l'écart type), même si la taille de chaque individu n'est pas connue.

Pour savoir pourquoi les méta-informations sont utiles, les informations sont réparties en catégories selon une matrice appelée Classification de Rumsfeld (du nom de l'ancien Secrétaire à la Défense américain), qui contient les éléments suivants :

- Connus connus
- Connus inconnus
- Inconnus connus
- Inconnus inconnus

Cette classification peut être représentée sous forme d'un tableau qui intègre tous les concepts étudiés jusqu'à présent :

	Connus	Inconnus
Connus	Informations	Entropie épistémique Défauts observables
Inconnus	Entropie épistémique Défauts observables	Entropie systémique

Les méta-informations sont très utiles pour au moins estimer la taille de ce que nous ignorons, même si nous ne disposons pas des informations. C'est précisément ce pour quoi la discipline de la méta-mathématique a été développée : établir, par l'analyse des structures mathématiques, ce qui peut être prouvé et ce qui ne le peut pas. Même si une grande partie des propositions mathématiques, comme l'hypothèse du continu relative à l'existence de certains cardinaux infinis, ne peuvent pas être prouvées, cela ne pose aucun problème à la communauté mathématique, car cela n'empêche pas de se concentrer sur les propositions qui peuvent l'être. Le même raisonnement doit s'appliquer au test de logiciels : l'utilisation de méthodes analogiques permet de se concentrer sur ce qui peut être testé (les connus connus) et de se mettre d'accord sur les méta-informations concernant le reste. Les inconnus inconnus (l'entropie systémique), quant à eux, posent problème, car il est impossible de les mesurer. Les deux autres sections (les inconnus connus et les connus inconnus) peuvent au minimum être mesurées, et permettent d'établir des statistiques, particulièrement utiles dans l'analyse des risques.

Cette analogie, que j'ai désignée par le terme « méta-développement », repose principalement sur l'idée que les logiciels testables (les connus connus) doivent capter toute l'attention et que tous les efforts doivent porter sur l'optimisation des informations disponibles et observables. Pour le reste, les méta-informations suffisent. Malheureusement, les méthodologies actuelles de conception de plans de test n'exploitent pas au maximum les informations disponibles, comme nous le verrons dans notre critique. Bien que la « plupart » des algorithmes, en termes de machines de Turing, ne puissent pas être testés (comme l'a démontré le problème de l'arrêt), un nombre infini de configurations peuvent bel et bien être testées, avec une couverture fonctionnelle d'au moins 100 %. Dans ce contexte, j'utilise le concept de taille relative qui existe en théorie de la mesure pour comparer des ensembles infinis : l'ensemble d'algorithmes non testables représente la majorité (un nombre infini indénombrable), tandis que l'ensemble d'algorithmes testables ne représente qu'un sous-ensemble négligeable (un nombre infini dénombrable), désigné dans la théorie de la mesure comme un « ensemble de mesure 0 » ou « ensemble nul », car infiniment petit en comparaison au reste.

Section 4

Critique des méthodes de conception des plans de test

Sur la base des principes fondamentaux susmentionnés, attachons-nous à présent à dresser une critique objective des méthodes de conception de plans de test. Conformément au premier principe, il ne sera pas possible de détecter tous les défauts. En revanche, conformément aux deuxième, troisième et quatrième principes, nous disposons de critères sur lesquels nous appuyer pour évaluer les méthodes de conception de plans de test :

1. Quelle part des informations de l'application peut être encodée dans le processus de conception de plans de test ?
2. Combien de défauts sont observables ?
3. Couverture : combien de défauts sont observables avec cette méthode par rapport au nombre maximal théorique de défauts observables ?
4. Quel est le nombre de plans de test relatifs requis pour obtenir une couverture optimale ?

Sur la base de ces quatre critères, chaque méthode de conception de plans de test reçoit une note sur 10, évaluant les éléments suivants :

- **Capacité d'encodage** : désigne la quantité d'informations quantitatives sur l'application pouvant être encodées dans la méthode. (Dérivé du point 1 ci-dessus.)
- **Facilité d'encodage** : degré de simplicité d'encodage des informations.
- **Applicabilité** : nombre de scénarios pouvant être raisonnablement encodés à l'aide de la méthode.
- **Nombre de plans de test** : nombre relatif de plans de test générés (1 : trop peu, trop ; 10 : nombre optimal). (Dérivé du point 4 ci-dessus.)
- **Défauts détectables** : nombre relatif de défauts décelables. (Dérivé du point 2 ci-dessus.)
- **Couverture** : couverture fonctionnelle relative pouvant être atteinte. (Dérivé du point 3 ci-dessus.)

Rappelons que toutes les notes doivent être comprises entre 1 et 10, 10 étant la « meilleure » note.

Section 5

Défauts observables dans la logique

Avant de passer au cœur du sujet, il est nécessaire de définir quelques propriétés des défauts observables par rapport aux arguments logiques. La plupart des modèles de test formels qui utilisent des informations sur les applications dépendent fondamentalement de l'analyse des arguments logiques. Il est donc judicieux d'entamer la recherche de défauts à ce niveau.

Examinons une phrase simple : **SI A ET B ALORS C**

Cette phrase peut être divisée entre causes (A et B) et effets (C). Concernant les défauts, les deux causes peuvent avoir trois états : correctement implémenté (OK), bloqué à 0 (0) et bloqué à 1 (1). Mais que passe-t-il, au vu de cette information, lorsque nous étudions les défauts associés à C ?

Tous les défauts possibles associés à C sont encodés dans le tableau ci-dessous (en rouge) :

A	B	C	1	1	OK	OK	1	1	0	0	A
			OK	OK	0	1	1	0	1	0	B
0	0	1	1	1	1	1	1	1	1	**0**	
0	1	1	**0**	1	1	1	1	1	1	**0**	
1	0	1	1	1	**0**	1	1	1	1	**0**	
1	1	0	0	**1**	0	**1**	**1**	**1**	**1**	**1**	

Comme vous pouvez le constater, les trois dernières variantes fonctionnelles (ensemble d'entrées vraies/fausses) suffisent pour détecter tous les défauts possibles. La première variante ne permet pas de détecter des défauts supplémentaires. Il est donc possible de prouver que tous les défauts possibles peuvent être détectés par cette méthodologie. Ce tableau indique les chiffres généraux d'un opérateur logique avec n entrées :

Nombre d'entrées	Nombre de variantes fonctionnelles requises	Nombre de combinaisons possibles	Nombre de défauts possibles
2	3	4	8
3	4	8	26
4	5	16	80
5	6	32	242
6	7	64	728
n	*n+1*	2^n	$\sum_{x=1}^{n} \binom{n}{x} 2^x = 3^n - 1$

Comme indiqué précédemment, il existe deux catégories de défauts, les défauts observables et non observables. La différence ne vient pas du caractère observable ou non des effets, mais du fait que leurs causes premières ne sont pas observables. Cela est capital si nous voulons nous assurer d'avoir la bonne réponse pour la bonne raison. L'observabilité dans ce contexte peut être définie comme la quantité d'informations minimum requises pour identifier, reproduire ou corriger le défaut. En d'autres termes, il devrait être possible, pour un défaut donné, d'indiquer précisément où le défaut s'est produit. À maints égards, une bonne méthode de conception de plans de test doit pouvoir inverser la machine et fournir le nombre de plans de test minimum pour mettre en évidence le plus de défauts observables possible. Rappelez-vous que, dans notre univers, les défauts observables représentent l'entropie épistémique et peuvent être invalidés et mesurés, alors que les défauts inobservables sont systémiques et ne peuvent être testés à l'aide du même modèle. La couverture de test est donc une mesure de la quantité d'entropies épistémiques pouvant être invalidées par le processus de test.

Section 6

Méthodes combinatoires : rapide mais peu efficaces

Inspirées d'arguments de comptage simples, les méthodes combinatoires sont la forme la plus basique de conception de plans de test. En quelques mots, toutes les combinaisons de paires, triples, etc. possibles sont créées et rapidement renseignées à partir d'une liste de colonnes et de valeurs possibles pour chacune. L'aspect le plus intéressant de ces méthodes est qu'elles ne reposent sur aucune connaissance particulière en matière d'application, mais c'est aussi leur principal point faible. C'est pourquoi l'entrée d'informations quantitatives est limitée aux données et que leurs relations sont totalement ignorées. Conformément au principe fondamental précédent, la quantité d'informations qualitatives sur le système révélées par ces tests sera également relativement faible. Étant donné qu'elles ne sont associées à aucun point fonctionnel quel qu'il soit, il est impossible de définir la quantité de logique de l'application qui sera testée. Dans la plupart des scénarios, les tests portant sur les informations endémiques sont insuffisants, tandis que certains aspects sont testés inutilement, compte tenu des conditions de logique redondantes.

Les méthodes combinatoires souffrent d'un autre point faible, qui est le corollaire du précédent : des combinaisons de données non valides sont souvent produites, entraînant des faux positifs où le test échoue à cause des données et non à cause d'un défaut réel. Le problème peut être atténué par l'introduction de contraintes : cela montre que lorsque davantage d'entrées quantitatives sont fournies, les données qualitatives sont nettement plus claires. Troisièmement, les résultats escomptés ne peuvent pas être produits automatiquement, puisqu'ils reposent sur les entrées quantitatives. D'autres méthodes de niveau supérieur permettent d'encoder ces informations, c'est pourquoi notre analyse révèle que les méthodes combinatoires sont un choix déplorable pour la conception de plans de test.

Il est donc préférable de réserver les méthodes combinatoires aux tests de configuration et non fonctionnels, autrement dit à des scénarios pour lesquels le seul résultat attendu est « ça fonctionne ».

Notes sur 10 des méthodes combinatoires :

Informations en entrée			Informations en sortie		
Fonctionnalité	Simplicité	Applicabilité	Nbre de plans de test	Défauts détectables	Couverture
1	9	5	2	1	1

Section 7

Sur-spécialisation : méthodes de couverture multiplicatives

Adaptation hautement spécialisée des méthodes combinatoires, cette méthode n'est utile que lorsque le scénario à tester dépend uniquement du nombre d'instances d'une entité de données en particulier. Par exemple, supposons que nous disposions d'une base de données de comptes client, et qu'un écran permette d'afficher tous les comptes d'un client donné. Les scénarios à tester seraient les suivants :

1. Le client n'a ouvert aucun compte.
2. Le client a ouvert un compte.
3. Le client a ouvert plusieurs comptes.

Notez que cette technique utilise de façon implicite une certaine logique fonctionnelle, c'est pourquoi, contrairement aux versions plus simplistes, cette méthode de conception de plans de test intègre des informations sur les applications. À vrai dire, j'utilise cette technique chaque fois que je traite des relations parent-enfant dans une base de données relationnelle ou dans des structures hiérarchiques (comme le XML). Plus généralement, cette technique se révèle particulièrement efficace pour tester des méthodes d'agrégation par rapport à des structures de données complexes. Cela tient au fait que certaines fonctions d'agrégation relèvent de cas spécifiques (ou dégénérescents) à traiter séparément. Par exemple :

- L'obtention de la moyenne d'un ensemble de chiffres nécessite un traitement spécial dans le cas d'un ensemble vide, faute de quoi vous obtenez un bogue divisible par zéro.
- L'obtention de l'écart-type d'un ensemble de chiffres nécessite également un traitement spécial dans le cas d'un ensemble vide et d'un ensemble comprenant une seule valeur ; dans le premier cas, vous obtenez un résultat négatif et dans le second, un résultat divisible par zéro, si ces deux cas ne sont pas traités séparément.

Le résultat qualitatif de cette méthode est une confirmation que tous les ensembles d'entrées possibles fonctionnent, sur la base de leur taille uniquement. Il s'agit d'un cas spécial de test de catégorie d'équivalence, où la plage d'entrée est regroupée en catégories disjointes afin de réduire le nombre total de scénarios, tout en maintenant l'intégrité de la couverture fonctionnelle.

Notes sur 10 des méthodes de couverture multiplicatives :

Informations en entrée			Informations en sortie		
Fonctionnalité	Simplicité	Applicabilité	Nbre de plans de test	Défauts détectables	Couverture
3	7	1	5	3	3

Section 8

Introduction aux modèles formels

Le reste de ce document sera consacré aux techniques de modélisation formelle, qui fournissent les informations qualitatives les plus précises sur le système. Comme évoqué dans le principe précédent, cela nécessite une très grande expertise, car pour tirer le maximum d'informations quantitatives, encore faut-il disposer d'un volume considérable de ces informations. Ces méthodes sont uniques par la quantité d'informations pouvant être encodées au cours du processus de conception des plans de test.

Avant de poursuivre, une brève description de la notion de modèles formels s'impose. Les modèles formels sont essentiellement des descriptions précises sur le plan mathématique d'une exigence, à partir desquelles des opérations supplémentaires peuvent être effectuées afin d'obtenir des informations qualitatives sur l'exigence. Les trois opérations majeures sont les suivantes :

1. Vérification de la cohérence : garantit que la logique de l'exigence est cohérente avec elle-même. Cette opération élimine toute ambiguïté due aux contradictions.
2. Vérification de la complétude : garantit que la logique de l'exigence est complète. Cette opération élimine toute ambiguïté due aux omissions.
3. Dérivation des résultats escomptés : une logique complète et cohérente produira les résultats attendus pour tout scénario possible, ce qui permet aux plans de test d'obtenir les résultats attendus dérivés automatiquement. De l'avis des testeurs, c'est le plus grand avantage des modèles formels.

Les formalismes s'alimentent eux-mêmes afin d'être plus « utiles » simplement en vertu des trois opérations ci-dessus : toutes les données précédentes fournissent des informations qualitatives incroyablement détaillées sur le système à tester. En fait, l'introduction d'informations qualitatives plus tôt dans le cycle de développement permet de les extraire directement des exigences, et de donner ainsi au projet une base solide dès le départ. Par ailleurs, la plupart des méthodologies de spécification des exigences sont inertes et inutiles en dehors de la phase de conception (en dehors d'une intervention manuelle), tandis que les modèles formels peuvent fournir des informations à toutes les étapes du cycle de développement. Sans entrer trop dans les détails, ces formalismes constituent le pilier sur lequel la plupart des avancées mathématiques et informatiques du siècle passé ont été réalisées. Si c'est assez bien pour eux, il va sans dire que c'est assez bien pour nous.

Il existe des modèles formels de forme et de taille différentes. Je me concentrerai donc sur les deux modèles les plus pertinents pour la phase de test, à savoir l'organigramme et le diagramme des causes et des effets. Il s'agit des formes canoniques ; chaque modèle formel s'inspire dans une certaine mesure de ces deux modèles.

Section 9

L'organigramme, ou l'outil de visualisation des exigences et des systèmes

L'organigramme est de loin la forme de spécification d'exigences la mieux comprise ; ses avantages par rapport à des approches de type « mur de texte » sont nombreux, mais aucun n'est plus pratique que la capacité à créer des plans de test de façon systématique directement à partir des exigences (ce que peuvent faire les algorithmes automatisés). L'organigramme est bien plus direct qu'il n'y paraît, mais cette idée n'est pas facile à faire admettre.

Pour créer des plans de test, il convient d'évaluer les chemins possibles de l'organigramme. Ce n'est pas plus compliqué que cela (officiellement, cela s'appelle une analyse par homotopie graphique). Cela devient intéressant lorsque les techniques d'optimisation sont utilisées pour réduire le nombre de plans de test, tout en préservant la couverture fonctionnelle en vertu de la structure logique. En substance, il suffit de prendre en compte les blocs décisionnels : si chaque bloc (ou opérateur) est entièrement testé en termes de chemins directs entrants et sortants, tout le flux est considéré comme étant testé. Cela est toutefois particulièrement difficile (et frustrant) à expliquer, à moins de s'écrier « les mathématiques fonctionnent ! »

En résumé, cela fonctionne car la présence d'un ensemble de conditions locales identique (au niveau de l'opérateur) à tester a pour effet de créer une redondance au niveau de certains chemins, ce qui se produit si nous tentons d'en factoriser un autre (global) à tester. Ce n'est qu'une question de variations fonctionnelles redondantes qui s'annulent mutuellement. C'est le genre de chose qu'il est plus facile d'expliquer en général que pour des instances particulières (comme c'est bien souvent le cas en mathématiques abstraites).

Étant donné que les organigrammes n'admettent pas la spécification d'exigences ambiguës, la quantité d'informations quantitatives pouvant être encodées est considérable (dans des proportions largement supérieures aux deux méthodes précédentes). Là encore, c'est vrai de toutes les méthodes de modélisation formelle, la seule différence réside dans le degré. Par ailleurs, au vu du concept de test au niveau de l'opérateur, la quantité de défauts pouvant être détectés est tout aussi considérable, c'est pourquoi la quantité d'informations qualitatives obtenues à partir du modèle est phénoménale.

Notes sur 10 de l'organigramme :

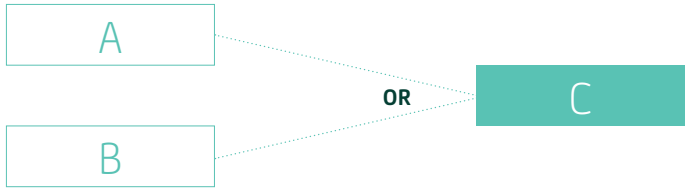
Informations en entrée			Informations en sortie		
Fonctionnalité	Simplicité	Applicabilité	Nbre de plans de test	Défauts détectables	Couverture
9	9	9	9	9	9

Section 10

Spécification de la logique pure - Graphique des causes et des effets

Il existe un autre type de modèle formel. Cette fois-ci, les arguments logiques sont modélisés en tant que causes et effets, et les relations entre les deux sont entièrement spécifiées. À l'instar de l'organigramme, ce modèle a fait l'objet d'une analyse particulière compte tenu de sa capacité à garantir une spécification non ambiguë des exigences concernant la prose.

Comme dans l'organigramme, les blocs sont reliés entre eux, mais dans ce modèle, c'est la relation causale qui est spécifiée. Les blocs sont par ailleurs reliés entre eux à l'aide d'opérateurs logiques, tels que ET, OU et NON. Par exemple, la phrase « Si A ou B alors C » peut être encodée comme suit :



Il doit être évident que l'information peut être encodée de manière très détaillée, et que cette méthode compte parmi les meilleures à cet égard. Toutefois, il apparaît également clairement que la méthode n'est pas si simple. Elle est d'ailleurs considérée comme l'une des plus difficiles.

En revanche, sa capacité à détecter des défauts est surprenante, plus qu'avec un organigramme, car les défauts peuvent être déduits de manière analytique d'une condition logique (voir la section sur les « défauts observables à partir de la logique » pour une explication complète du processus). Il est notamment possible de prouver que cette méthodologie permet de déceler tous les défauts possibles. Par conséquent, cette méthodologie est de loin la meilleure pour la détection de défauts observables, bien qu'elle soit nettement plus ardue qu'un organigramme, ce qui a malheureusement freiné son adoption. Sa complexité réside notamment dans l'encodage d'exigences dans un ordre spécifique (contrairement aux organigrammes qui sont parfaits à cet égard, mais dans lesquels il est difficile d'encoder correctement des exigences pour lesquelles l'ordre n'a aucune importance).

Note sur 10 des graphiques de causes et d'effets :

Informations en entrée			Informations en sortie		
Fonctionnalité	Simplicité	Applicabilité	Nbre de plans de test	Défauts détectables	Couverture
10	5	8	10	10	10

Section 11

Comparaison relative

Le tableau ci-dessous compare les notes générales obtenues par toutes les méthodes de conception de plans de test. Toutes les notes sont comprises dans une échelle allant de 1 à 10 et, bien qu'elles soient les plus objectives possibles, elles sont attribuées sur la base des critères suivants :

- **Capacité d'encodage** : quantité d'informations quantitatives sur l'application pouvant être encodées dans la méthode.
- **Facilité d'encodage** : degré de simplicité d'encodage des informations.
- **Applicabilité** : nombre de scénarios pouvant être raisonnablement encodés à l'aide de la méthode.
- **Nombre de plans de test** : nombre relatif de plans de test générés (1 : trop peu, trop ; 10 : nombre optimal).
- **Défauts détectables** : nombre relatif de défauts décelables.
- **Couverture** : couverture fonctionnelle relative pouvant être atteinte.

REMARQUE : Les catégories de modèles formels sont également incluses. Les notes sont incluses sous forme de fourchettes, car certains modèles sont davantage capables, moins applicables, etc. que d'autres.

Méthode	Informations en entrée			Informations en sortie		
	Fonctionnalité	Simplicité	Applicabilité	Nbre de plans de test	Défauts détectables	Couverture
Combinatoire	1	9	5	2	1	1
Multiplicative	3	7	1	5	3	3
Modèles formels (généraux)	**7-10**	**5-10**	**5-10**	**5-10**	**5-10**	**5-10**
Organigramme	9	9	9	9	9	9
Causes et effets	10	5	8	10	10	10

Section 12

Avantages de CA Technologies

CA Technologies (NASDAQ : CA) fournit des solutions de gestion des systèmes d'information qui aident ses clients à gérer et à sécuriser des environnements informatiques complexes pour supporter des services métier agiles. Les organisations s'appuient sur les logiciels et les solutions SaaS de CA Technologies pour accélérer l'innovation, transformer leur infrastructure et sécuriser les données et les identités, du cœur des data centers jusqu'au Cloud. CA Technologies s'engage à ce que ses clients atteignent les résultats souhaités et la valeur métier attendue grâce à l'utilisation de sa technologie. Pour en savoir plus sur nos programmes de succès clients, rendez-vous sur le site **ca.com/fr/customer-success.** Pour plus d'informations sur CA Technologies, rendez-vous sur le site **ca.com/fr**.

ca technologies

Restez connecté à CA Technologies sur ca.com/fr

CA Technologies (NASDAQ : CA) fournit les logiciels qui aident les entreprises à opérer leur transformation numérique. Dans tous les secteurs, les modèles économiques des entreprises sont redéfinis par les applications. Partout, une application sert d'interface entre une entreprise et un utilisateur. CA Technologies aide ces entreprises à saisir les opportunités créées par cette révolution numérique et à naviguer dans « l'Économie des applications ». Grâce à ses logiciels pour planifier, développer, gérer la performance et la sécurité des applications, CA Technologies aide ainsi ces entreprises à devenir plus productives, à offrir une meilleure qualité d'expérience à leurs utilisateurs, et leur ouvre de nouveaux relais de croissance et de compétitivité sur tous les environnements : mobile, Cloud, distribué ou mainframe. Pour en savoir plus, rendez-vous sur **ca.com/fr**.

CS200-154749