UNIVERSITE CATHOLIQUE DE LOUVAIN

cpme Formation interdisciplinaire en création d'entreprise

TITRE : usiSketch - vers un logiciel de prototypage d'interfaces par l'esquisse

MÉMOIRE CONFIDENTIEL

Mémoire présenté en vue de l'obtention du grade de

Promoteurs

Master 120 a finalité spécialisée en

Prof. J. Vanderdonckt LSM

Ingénieur civil en informatique - INFO2M

Olivier BOURDOUX

Prof. C. de Moerloose LSM

Ingénieur Commercial - INGE

Louis VERSPREEÙWEN

Table des matières

Abréviations

GUI Interface Utilisateur Graphique : UI basée sur l'affichage graphique d'éléments. Windows, Mac Os, Linux sont basés sur des GUI. Les smartphones sont eux aussi basés sur des GUI.. 13, 14

IT IT signifie Technologie de l'Information. Cela regroupe l'ensemble des technologies liées à l'informatique en général, à la téléphonie et au multimédia. 128

MVC Le Modèle-Vue-Contrôleur (en abrégé MVC) est une architecture et une méthode de conception qui organise l'interface homme-machine (IHM) d'une application logicielle. Ce paradigme divise l'Interface Homme-Machine en un modèle (modèle de données), une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des événements, synchronisation), chacun ayant un rôle précis dans l'interface. (source : Wikipedia). 23, 40

NUI Interface Utilisateur Naturelle : UI dont l'objectif est l'interraction naturelle avec l'utilisateur, par la voix, le mouvement, le dessin... La NUI la plus connue à ce jour est le module Kinect pour XBOX 360 de Microsoft, où le corps de l'utilisateur fait office de contrôleur. 13, 14

OS Un Operating System est le logiciel responsable de la gestion globale d'un ordinateur. Les plus connus (sur ordinateur classique) sont Windows, Mac OS X, et Linux. 38, 39, 88, 151

SDLC L'acronyme SDLC signifie "Software Development Life Cycle" (en français Cycle de développement logiciel). Il s'agit d'une représentation théorique du développement d'un logiciel. En théorie, quelques modèles typiques sont définis. En pratique, ces modèles ne sont que partiellement respectés, résultant en un nombre de modèles SDLC quasiment égal au nombre de logiciels existants. 10, 12

UI Interface Utilisateur. 3, 8–10, 12–14, 16, 23, 29, 106, 123, 124

W3C Le World Wide Web Consortium (W3C) est une communauté internationale qui développe des standards pour assurer la croissance du Web sur le long terme. (source : www.w3.org). 23

WYSIWYG L'acronyme WYSIWYG signifie "What You See Is What You Get", dont le sens est assez évident. La suite logicielle WYSIWYG la plus connue est probablement Microsoft Office, où les documents traités sont pour l'essentiel des documents papiers. 16, 22, 33, 47, 141

XML XML (Extensible Markup Language, "langage de balisage extensible") est un langage informatique de balisage générique. Cette syntaxe est dite extensible car elle permet de définir différents espaces de noms, c'est à dire des langages avec chacun leur vocabulaire et leur grammaire. (source : Wikipédia). 23, 39

Glossaire

apprentissage supervisé L'apprentissage supervisé est une technique d'apprentissage automatique où l'on cherche à produire automatiquement des règles à partir d'une base de données d'apprentissage contenant des "exemples". (source : Wikipedia). 24, 47, 48

branchement On parle de branchement d'un projet informatique lorsque celui-ci, à une version précise, se sépare en sous branche. Chaque sous branche est une évolution indépendante du logiciel, et peut elle aussi être sujette à un sous branchement. Le branchement permet donc à un développeur de modifier profondément le code sans avoir à se soucier de l'intégrabilité ou de la compatibilité des autres branches d'un projet.. 52

eclipse Eclipse est un environnement de développement open source. Son objectif est de définir un environnement qui puisse être utilisé du début à la fin du cycle de développement d'un logiciel. Eclipse est très largement utilisé et dispose de nombreux plugins maintenu par une très grande communauté. 39, 42, 43

edit-distance L'edit-distance entre deux chaînes de caractères permet d'évaluer la distance relative entre celles-ci. Elle représente le nombre d'opération atomiques (insertion, effacement, remplacement d'un caractère) minimum à faire pour passer d'une chaîne à l'autre. 50

human-readable Un langage est dit $human - readable$ s'il est possible pour un humain de lire le fichier et d'en comprendre la majorité du contenu. La majorité des langages de programmation sont $human - readables$, de même que XML et ses extensions. A contrario, le code source d'une image ou d'un fichier pdf n'est pas $human - readable$. 22

librairie Une librairie est, en jargon informatique, un morceau de programme qui n'est pas exécutable tel quel, mais qui peut être intégré à un autre logiciel pour en étendre les fonctions. Les librairies sont très utilisées car elles permettent un développement plus

rapide et permettent aux développeurs de réutiliser du code déjà écrit et testé par d'autres développeurs. 131

modalité d'interaction Du point de vues des interfaces Homme-Machine, la modalité d'interaction se définit comme le mode d'interaction utilisé lors d'une interaction, en entrée comme en sortie. Typiquement, il s'agit en entrée de la voix, du clavier, de la souris... alors qu'en sortie, il s'agira essentiellement de l'écran, d'un signal sonore, d'une vibration... 13, 23

niveau de fidélité Le niveau de fidélité d'un prototype d'Interface Utilisateur (UI)* est une évaluation de la correspondance visuelle entre ce prototype et l'interface réelle. Un haut niveau de fidélité signifie que le prototype reflète très fidèlement l'interface réelle, tandis qu'un bas niveau signifie que le prototype ne représente que sommairement l'interface réelle. 14, 15, 17, 21, 32, 39, 41, 47, 66, 74, 75, 138, 143

open source La désignation open source s'applique aux logiciels dont la licence respecte des critères précisément établis par l'Open Source Initiative, c'est-à-dire la possibilité de libre redistribution, d'accès au code source et de travaux dérivés. (source : Wikipédia). 128, 129, 131

parsing Le terme anglais *parsing* est utilisé en informatique pour décrire la décomposition analytique d'un fichier. Il consiste en la traduction d'un fichier en une représentation utilisable par le logiciel. 39

requirement Le terme "requirement" tel qu'utilisé en informatique n'a pas d'équivalent parfait en français. Il pourrait être approximativement traduit par "requis", "besoin" ou "exigence". 10, 14, 30–37, 65, 66, 68–70, 75

thread Un thread est un sous programme lancé en parallèle de l'exécution d'un logiciel. Il permet d'exécuter plusieurs tâches en même temps. 44

version alpha La version alpha est une dénomination largement utilisée en développement logiciel. C'est la première phase de développement concret du logiciel après le codage de l'application. Une version alpha n'est pas censée être accessible à un large public : c'est une version interne. Généralement, un produit en test alpha - on utilise couramment le terme anglais alpha-test - n'a pas toutes les fonctionnalités prévues dans le produit final, contrairement à un produit en bêta-test qui devrait être complet (mais toujours

sujet à certains bugs). L'alpha est donc dépourvue de certaines fonctionnalités, et contient un nombre de bugs encore important (source : Wikipédia). 20

widget Un widget est un élément d'interface utilisateur. Il peut s'agir d'un bouton, de texte, d'un champ de texte, d'une liste, etc. 29, 35, 41, 45, 47, 59, 60, 74, 75, 144

Table des figures

Première partie

Introduction

Chapitre 1

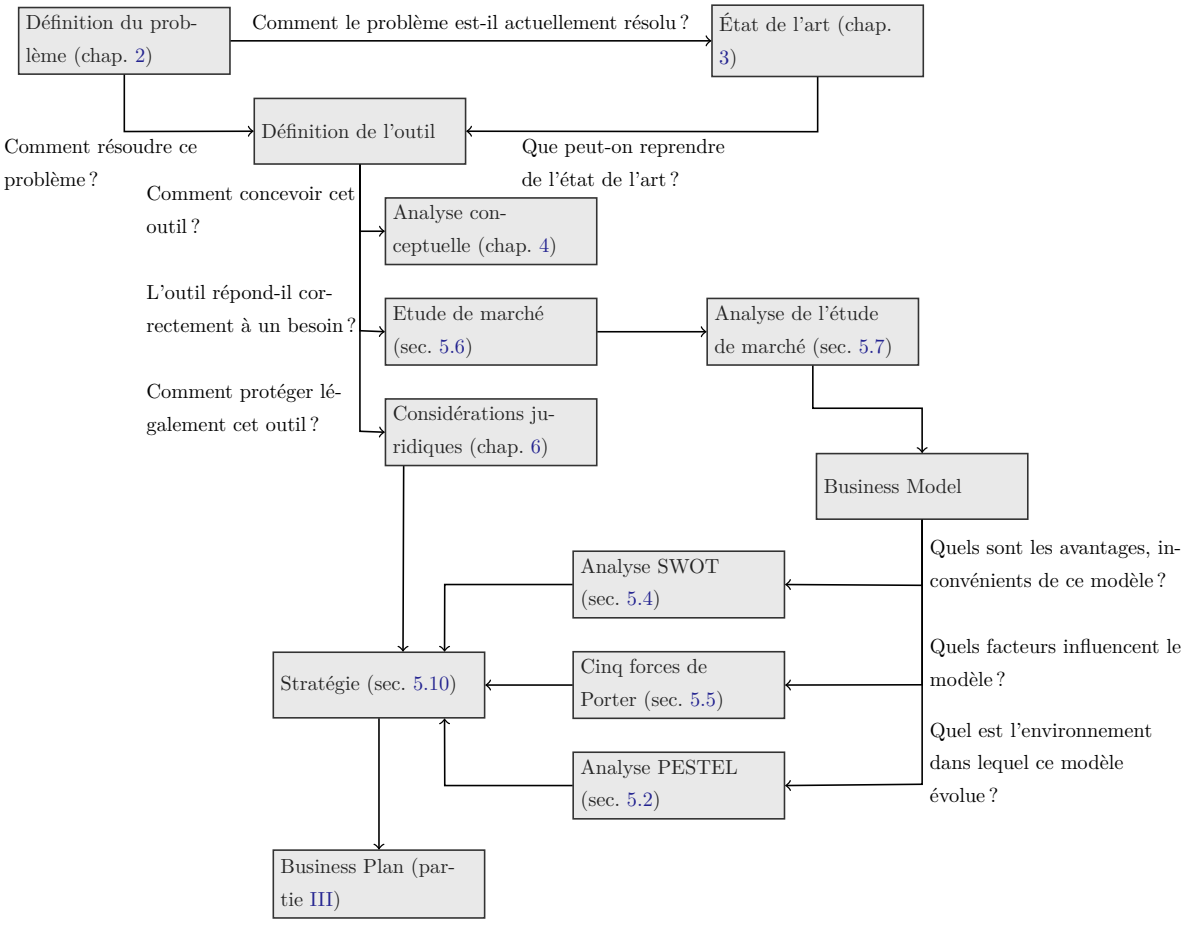
Préface

étoffer préface

Ce mémoire CPME traite de l'élaboration d'un outil de prototypage d'UI* basé sur l'esquisse, et d'une étude marketing en vue de sa mise en valeur.

1.1 Plan de lecture

Pour faciliter la compréhension du lecteur, un glossaire et une liste des abréviations ont été définis. Les mots et abréviations repris dans ceux-ci sont annoté d'une étoile. La figure ci-dessous illustre le plan de lecture du présent document.



Chapitre 2

Problème à résoudre

Ce chapitre est structuré en quatre parties. Tout d'abord, nous définirons les concepts de base utilisés dans ce mémoire. Nous enchaînerons par le problème lié au prototypage lui-même. Ensuite, nous définirons l'objectif central de ce mémoire. Enfin, nous présenterons le plan de lecture de ce document.

2.1 Définitions

2.1.1 L'Interaction Homme-Machine (IHM) et les Interfaces Utilisateur (UI)

L'interface homme-machine (IHM) définit les moyens et outils mis en oeuvre afin qu'un humain puisse contrôler et communiquer avec une machine. Les ingénieurs en ce domaine étudient la façon dont les humains interagissent avec les ordinateurs ou entre eux à l'aide d'ordinateurs, ainsi que la façon de concevoir des systèmes qui soient ergonomiques, efficaces, faciles à utiliser ou plus généralement adaptés à leur contexte d'utilisation. [...] L'interaction entre l'utilisateur et la machine se déroule dans l'Interface Utilisateur (UI).

Wikipédia (2011a)

En développement logiciel, une application interactive est typiquement décomposée en deux parties :

1. le noyau fonctionnel, qui regroupe les fonctions sémantiques de l'application ;
2. l'IHM.

Dans la suite de ce mémoire, le prototypage se centrera sur l'IHM.

2.1.2 Le prototypage d'UI

Ambler (2009) définit le prototypage d'UI* comme une technique d'analyse itérative durant laquelle les utilisateurs sont impliqués activement dans sa conception d'une UI* pour un système. Un prototype d'UI* a plusieurs utilités :

- explorer les problèmes posés par le système à concevoir.
- déceler les requirements* demandés par le système à concevoir.
- concevoir les solutions possibles pour le système à concevoir.
- communiquer les différentes UIs* possibles pour le système.
- construire les fondations à partir desquelles le système peut être construit.

Prototypage vertical et horizontal from Louis

2.2 Problème

2.2.1 Le prototypage d'UI, pour quoi faire ?

Le prototypage d'UI*, bien qu'utile, est accessoire dans le développement logiciel : le prototype en lui-même n'est pas utilisé dans le produit final, il ne sert que d'intermédiaire entre l'idée et sa concrétisation. Dans, ce cas, pourquoi gaspiller du temps et de l'énergie à en produire un ? Nous tenterons d'expliquer ici les raisons qui poussent les équipe de développement à effectuer du prototypage d'UI*. Nous commencerons par définir certains concepts largement utilisés en développement logiciel, ensuite nous étudierons le prototypage d'UI* et l'intérêt qu'il représente.

Cycle de développement logiciel ou Software Development Life Cycle

Généralement, le développement d'un logiciel se passe selon un schéma préétabli par le gestionnaire de projet. Bien que ce schéma puisse différer d'une équipe de développement à l'autre, et d'un projet à l'autre, la logique sous-jacente est généralement la même. Le Cycle de développement logiciel (SDLC)* est une représentation théorique de ces schémas de développement. Il existe beaucoup de représentations différentes d'un SDLC*. Il arrive souvent qu'il ne soit pas respecté, cependant il est très utile en tant que "feuille de route" du développement logiciel, et son utilisation est très répandue.

Deux exemples de diagramme SDLC* sont présentés aux Figure 2.1 et 2.2.

Détection et correction des erreurs

Durant chaque étape d'un développement logiciel, des erreurs s'immiscent dans le système. Plus ces erreurs sont détectées et corrigées tard dans le SDLC*, plus le coût de cor-

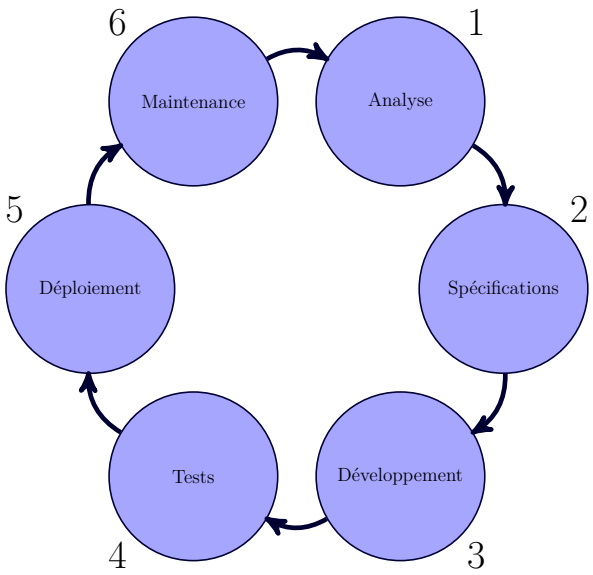


FIGURE 2.1 – Exemple de diagramme SDLC circulaire

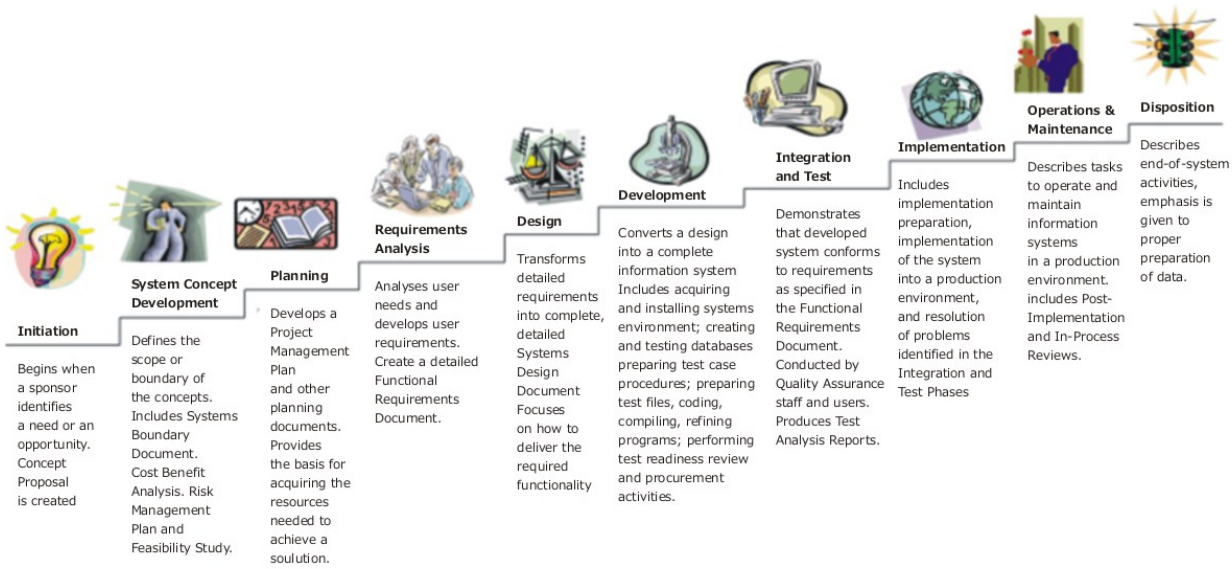


FIGURE 2.2 – Exemple de diagramme SDLC en cascade ; source : US Department of Justice (2003)

rection est élevé (King et Marasco (2008)). Il est donc d'un intérêt vital pour une équipe de développement de détecter ces erreurs le plus rapidement possible. La figure Figure 2.3 illustre le coût relatif de correction des erreurs en fonction du stade de développement de l'application.

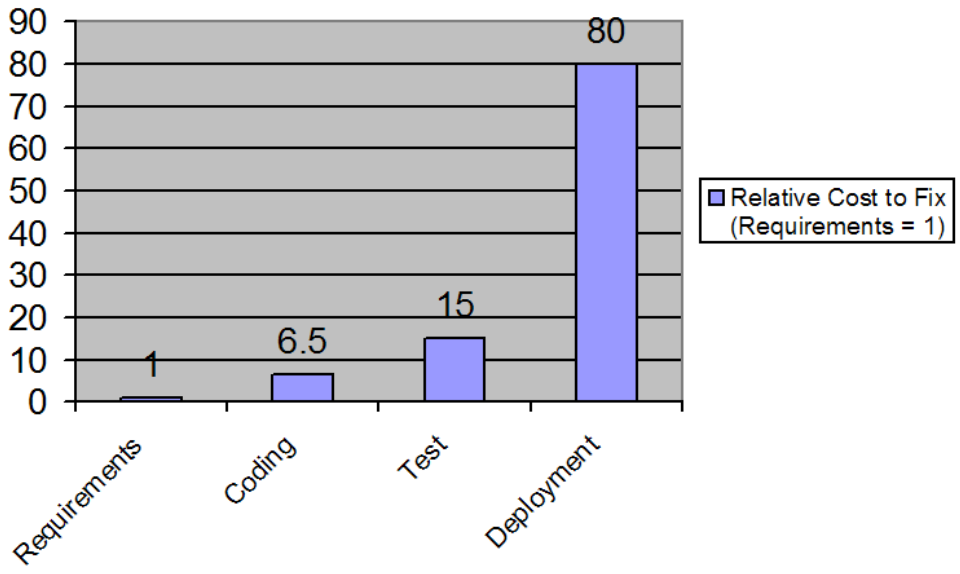


FIGURE 2.3 – Coût relatif de correction d'erreur selon le stade de développement ; source : King et Marasco (2008)

La Figure 2.4 est une illustration de tableau reprenant le nombre d'erreurs introduites et corrigées pour un logiciel donné. Par exemple, 23 erreurs furent introduites durant l'étape de spécification et corrigées lors des tests par les développeurs. Pour chaque cellule, plus la couleur tend vers le rouge, plus le coût de correction est élevé (car corrigé durant les dernières phases de développement). Un logiciel développé de façon peut coûteuse doit donc voir un maximum d'erreurs dans les cellules en haut à gauche de la figure.

Le prototypage d'UI* est une des premières étapes dans le développement logiciel. Il est principalement effectué durant l'étape de design sur la Figure 2.4. Il peut cependant être effectué plus tôt, selon le logiciel à développer. Toutefois, il l'est toujours avant les phases de test.

La principale raison pour laquelle un prototypage est fait dans de gros projets de développement logiciel est tout simplement celle-ci : prototyper l'interface à venir, c'est détecter les problèmes d'ergonomies qui vont se présenter, avant même que la première ligne de code ne soit produite. Comme nous venons de de le voir, détecter une erreur tôt réduit considérablement son coût de correction. Par conséquent - et paradoxalement - prototyper une UI*, c'est gagner du temps et de l'argent.

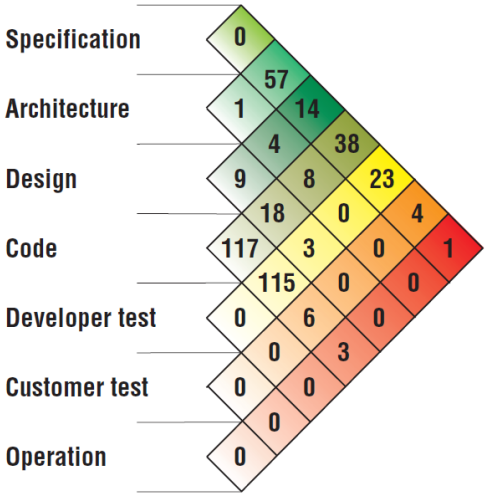


FIGURE 2.4 – Étapes de développement où les erreurs sont introduites et où elles sont corrigées ; source : Hall et Chapman (2002)

Autres avantages

Bien entendu, d'autres avantages, précités à la section 2.1.2, sont à considérer.
C'est un précieux outil de communication : le fonctionnement d'un outil peut être décrit à un utilisateur non-informaticien avec une plus grande facilité. De plus, le client peut apporter un feedback précieux dès les premières étapes de développement. Sans prototypage d'UI*, il doit attendre la phase de tests, qui se trouve tard dans le SDLC*. Encore une fois, prototyper permet de réduire les coûts.
C'est aussi un outil de comparaison : utilisé en interne dans une équipe de développement, le prototypage d'UI* peut servir pour comparer deux solutions, et prendre ainsi la bonne décision dès le départ.
Enfin, c'est occasionnellement un outil de production : si le prototypage est fait via un outil logiciel, il peut être utilisé comme base pour la conception de l'UI* réelle, qui se situe durant la phase de développement.

2.2.2 Le prototypage d'UI, problème intrinsèquement complexe

De nos jours, il existe de nombreuses modalités d'interaction* moyen d'interagir avec un ordinateur. La plus ancienne, l'Interface Utilisateur par Caractère (CUI), fonctionne exclusivement par texte et utilise le clavier comme unique interface physique d'entrée. Son successeur, l'Interface Utilisateur Graphique (GUI)*, est le type d'UI* le plus répandu de

nos jours : la souris est utilisée en complément du clavier et des éléments graphiques (boutons, fenêtres, images...) sont utilisés en complément du texte. Un nouveau type d'interface commence aussi à émerger : les Interfaces Utilisateur Naturelles (NUI)*. La plus connue aujourd'hui est lez module Kinect pour XBOX 360, de Microsoft, où le corps de l'utilisateur fait office d'interface d'entrée.

Ces différents types d'UI* rendent les outils de prototypage extrêmement difficiles à généraliser. Prenons l'exemple des GUI* contre les NUI*. Il est possible de représenter sur papier une GUI* avec un niveau de fidélité* correct. Mais comment le faire pour une interface basée sur le mouvement du corps ou sur la voix ? C'est tout simplement irréalisable.

Même en se limitant uniquement aux GUIs*, définir un outil capable de toutes les prototyper est extrêmement complexe. En effet, un des principaux requirements* d'un outil de prototypage est sa capacité à simuler le comportement de l'interface. Mais que faire lorsque le projet de développement utilise des contrôles rarissimes, voire uniques ?

Par exemple, imaginons une interface possédant un menu circulaire, apparaissant au centre de l'écran lors de l'appui d'une touche du clavier. Ce type de contrôle est très rare. Par conséquent, les outils de prototypage informatiques ne disposent pas de ce type de contrôle nativement. Deux options se présentent alors au concepteur :

- Créer son prototype sur papier, mais au prix d'une réduction significative de la simulabilité du prototype.
- Si le logiciel le permet, créer le contrôle lui-même. Cependant, cela va à l'encontre de l'objectif du prototypage d'UI* : créer rapidement une interface simple sans avoir à écrire de code. Il va perdre du temps pour créer ce contrôle, qu'il n'utilisera probablement qu'une fois dans sa vie.

Pour toutes les raisons évoquées ci-dessus, le prototypage d'UI* est intrinsèquement complexe : il est virtuellement impossible de créer un outil de prototypage capable de représenter et simuler tous les prototypes imaginables. Nous verrons au chapitre 3 différents outils de prototypage représentatifs et les comparerons.

2.3 Objectif central

Maintenant que nous venons de présenter les concepts de base, ainsi que les problèmes liés au prototypage, il est temps d'énoncer l'objectif central de ce mémoire :

Notre objectif est la recherche d'opportunité, l'analyse conceptuelle détaillée, la mise en valeur et une analyse juridique succincte d'un logiciel de prototypage basé sur l'esquisse, permettant un prototypage rapide, intuitif, réutilisable, et d'un niveau de fidélité* bas. Nous synthétiserons cette recherche par l'élaboration d'un plan d'affaires concernant la création d'un entreprise valorisant cet outil.

Chapitre 3

Etat de l'art

Le logiciel usiSketch tire parti de différentes technologies, tant au niveau software que hardware. En effet, le logiciel en lui-même est un outil de prototypage par esquisse, ce qui implique une technologie de reconnaissance d'image ainsi que celle des outils de prototypage. Il repose aussi sur usiXML, qui est un langage de description d'interfaces graphiques. Enfin, il faudra un outil matériel de dessin assisté par ordinateur pour utiliser le logiciel - l'outil le plus typique étant une tablette graphique -. Certaines de ces technologies existent déjà depuis longtemps et sont largement utilisées, d'autres sont apparues plus récemment.

Je trouve que ce paragraphe arrive trop tôt : usiSketch n'a pas encore été clairement défini. Pourtant, il introduit très bien ce chapitre en reprenant chaque technologie présentée. Symétriquement, définir usiSketch avant semblerait artificiel, puisque l'état de l'art conduit à sa définition. Qu'en pensez-vous ?

Ce chapitre traite de chacune de ces technologies individuellement. L'objectif est d'établir une synthèse de toutes les solutions existantes, non seulement pour se positionner correctement sur le marché, mais aussi pour avoir une idée claire de ce qui est aujourd'hui techniquement possible et de ce qui ne l'est pas.

Nous commencerons l'analyse avec les outils de prototypage et les éditeurs d'interfaces. Ensuite, nous analyserons les différents langages de description d'UI* existants. Nous poursuivrons avec l'analyse des différentes techniques de reconnaissance d'image, pour terminer avec les solutions matérielles.

3.1 Les outils de prototypage

Un outil de prototypage a pour but de faciliter le prototypage d'UI* tel que définit à la section 2.1.2. Rappelons que durant le prototypage, le concepteur doit produire une représentation plus ou moins fidèle du logiciel final, afin de valider son ergonomie, les scénarios d'utilisation typique, etc. L'important dans cette étape n'est donc pas la création de

l'interface finale, mais bien une représentation suffisamment fidèle pour en tirer les conclusions voulues.
Pour cette raison, les outils de prototypage sont tous (ou presque) des outils de type What You See Is What You Get (WYSIWYG)*.

Détailler davantage la définition d'un outil de prototypage, ainsi que les types de prototypes possibles. Voir article ci-joint dont vous pouvez reprendre les figures si vous le voulez

Nous allons présenter ici quelques outils de prototypage connus, et évaluerons plusieurs de leurs attributs :

- intuitivité : facilité d'accès à l'outil, sans apprentissage préalable
- simulabilité : capacité de l'outil à simuler le comportement de l'interface finale
- complétude : capacité de l'outil à représenter le plus de prototypes différents possible
- niveau de fidélité* : capacité de l'outil à fournir un prototype représentant le plus fidèlement possible l'interface finale
- exportabilité : capacité de l'outil à fournir du matériel directement réutilisable dans la suite du processus de développement
- collaborativité : capacité de l'outil à permettre un travail de plusieurs personnes sur le même projet
- temps d'apprentissage : temps nécessaire à un nouvel utilisateur pour apprendre à utiliser le logiciel efficacement
- temps de production : temps relatif nécessaire à un concepteur pour construire un prototype
- prix

3.1.1 Papier-Crayon

Aujourd'hui, le prototypage se fait encore actuellement sur un support papier. Beaucoup de concepteurs considèrent encore le prototypage papier comme une des méthodes les plus efficaces(Bailey et Konstan (2003); Landay et Myers (2001); Newman et al. (2003)). En effet, rien n'est plus naturel, lorsque l'on parle de prototypage, de prendre une feuille de papier et de dessiner ce que l'on s'attend à voir à l'écran. C'est une méthode quasiment gratuite, intuitive et à la portée de tous, ce qui en fait un outil de choix dans le processus de prototypage. C'est aussi un outil très complet : l'ordinateur analysant et simulant le comportement du prototype (ici, la feuille de papier) est le cerveau humain.

Toutefois, son niveau de fidélité et sa simulabilité sont très mauvais. Le dessin est généralement très différent du résultat final, et simuler une interface avec uniquement du papier est

très difficile (bien que réalisable). Il existe des techniques visant à utiliser des post-its et/ou des feuilles partiellement découpées . La simulation se fait alors en superposant ces différentes feuilles. Cette technique est coûteuse en temps et en effectif humain : il faut, en plus de l'utilisateur, un humain (généralement le designer) chargé de simuler le comportement de l'interface. Un exemple de prototype par post-its est présenté Figure 3.1.

citation livre sur le prototypage

L'exportabilité, quant à elle, est tout simplement nulle : il faut réécrire électroniquement le prototype papier après validation, ce qui entraîne un surcoût supplémentaire.

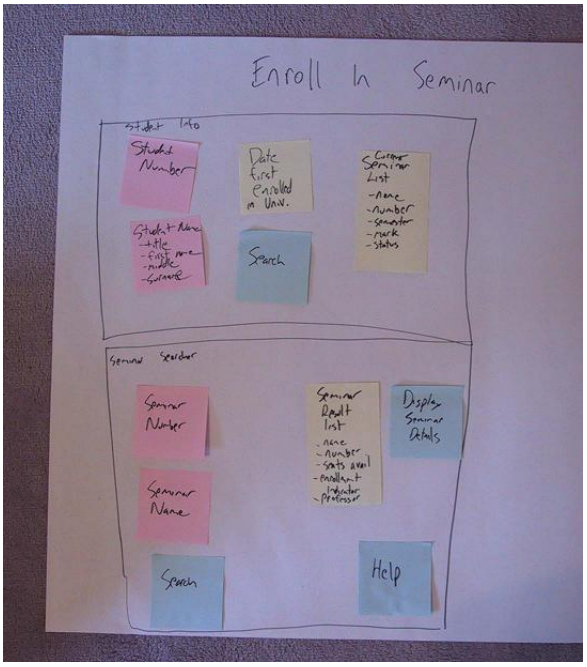


FIGURE 3.1 – Exemple de prototypage papier par post-its ; source : Ambler (2009)

3.1.2 iRise

iRise est une entreprise de développement spécialisée dans le prototypage destiné aux entreprises. Leur logiciel est un puissant outil de définition de plate formes, principalement web. Il est destiné et utilisé par des spécialistes et permet des simulations d'applications logicielles via navigateur. La version professionnelle d'iRise est vendue à 6.995$.

Ce logiciel est intéressant car il permet un travail collaboratif : le créateur peut publier son interface sur un serveur géré par iRise, afin que d'autres collaborateurs puissent tester l'interface en ligne et donner un feedback au designer.

iRise est uniquement utilisable sous Windows, ce qui lui ferme une part de marché de plus en plus importante .

trouver citation pour corroborer

Cet outil est un bon représentant des outils de prototypage logiciel actuels : il est très complet, parfaitement simulable et offre un niveau de fidélité presque parfait. Toutefois, ces outils sont peu intuitifs et demandent un apprentissage préalable pour être pleinement utilisés.

L'exportabilité d'iRise est par contre très mauvaise : le format de fichier utilisé par iRise est propriétaire, et l'outil n'offre que très peu de fonctionnalités d'exportation. Remarquons cependant que certains autres outils de prototypage ont une excellente exportabilité.

Ensuite, le logiciel n'est pas intuitif. Une étude menée par Olivier Bourdoux, Antoine Dubois et Akonkwa Mubagwa (annexe E) a montré que le logiciel était très difficile à utiliser par des personnes peu compétentes en informatique.

Enfin, le prix du logiciel est élevé. Par conséquent, seules les entreprises suffisamment grandes et dont le budget et les besoins justifient cet achat seront en mesure de s'offrir un tel logiciel.

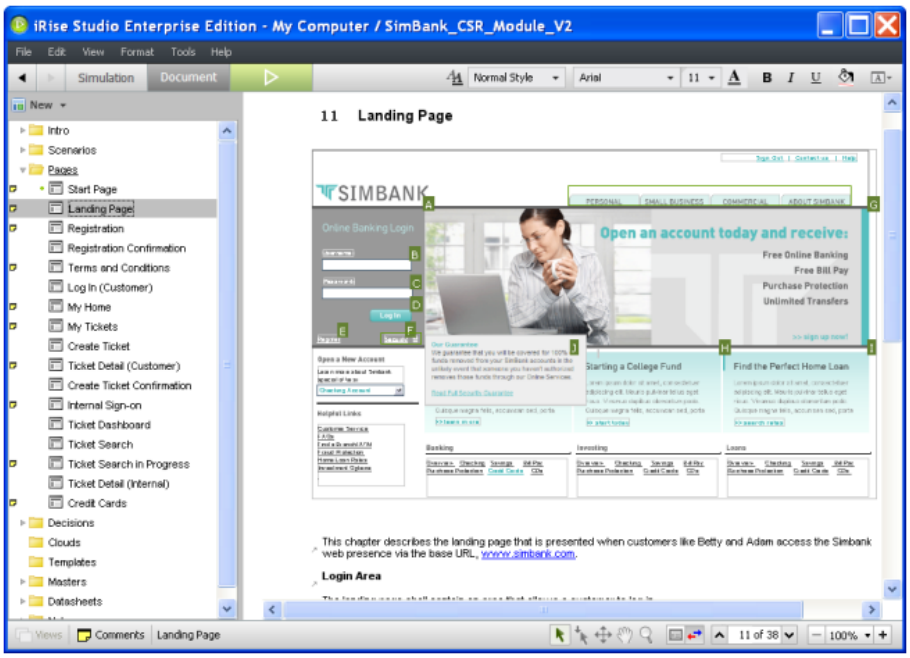


FIGURE 3.2 – Prototypage sous iRise

3.1.3 SketchFlow

SketchFlow fait partie de la suite Microsoft Expression Studio. C'est un outil de prototypage pour entreprises comparable à iRise. Cependant, il fournit d'avantage d'outils d'exportation et son niveau de fidélité est légèrement inférieur (le widgets sont affichés avec un "sketchy style", comme s'ils avaient été dessinés à la main). Le prix officiel annoncé de Microsoft Expression Studio est de 599$.

SketchFlow apporte la possibilité d'annoter les prototypes à la main. C'est une fonctionnalité intéressante car imitant la méthode papier/crayon, permettant ainsi une plus grande complétude. Cela permet aussi de créer des widgets non standards, mais ne permet pas d'en simuler le comportement.

Remarquons aussi le prix nettement inférieur de la suite Microsoft, ce qui le rend plus abordable pour de plus petites structures.

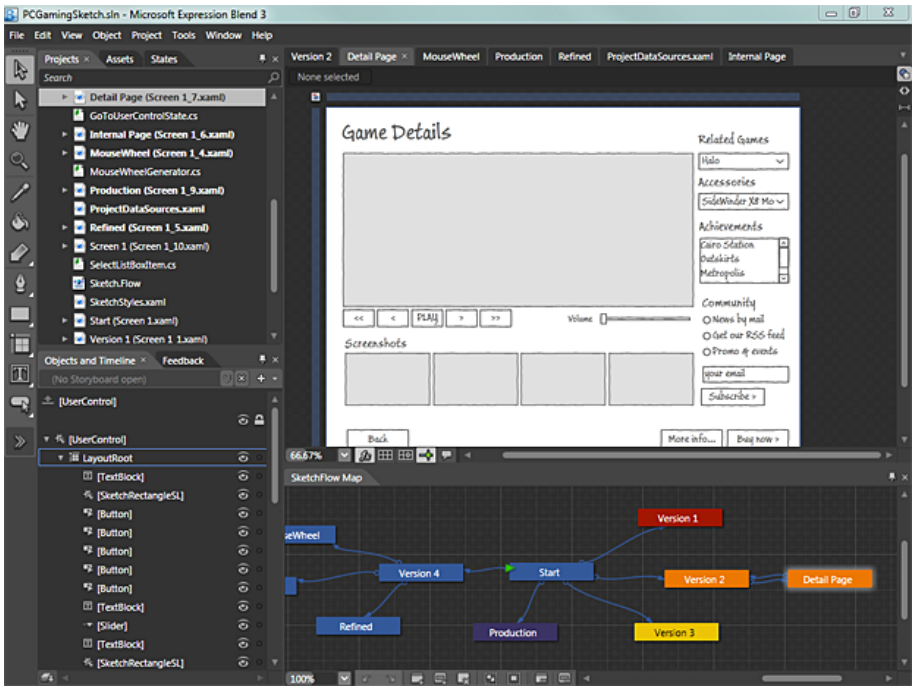


FIGURE 3.3 – Prototypage sous SketchFlow

3.1.4 SketchiXML

SketchiXML est l'ancêtre d'usiSketch. Plus exactement, il s'agit de la version alpha*, difficilement distribuable bien que fonctionnelle. Il a été principalement développé par Adrien Coyette et est disponible gratuitement sur le site d'usixml usixml.org (2010).

La mission de SketchiXML est la même que celle d'usiSketch : combiner le dessin à la main et le prototypage informatique. Les principales fonctionnalités offertes par SketchiXML sont :

- la reconnaissance de formes simples (ligne, triangle, rectangle, cercle, croix, flèche),
- la combinaison de ces formes simples en widgets usuels (bouton, label, champ de texte...),
- l'assignation d'actions typiques sur certains widgets ; cela reste toutefois très restreint,
- l'exportation de l'interface sous le format usiXML.

C'est un outil ayant une fidélité relativement basse : les widgets dessinés n'ont pas tous la même taille, ne sont pas alignés alors qu'ils le devraient... Cependant, l'exportation en usiXML permet de réutiliser ce travail préalable dans un logiciel d'édition ayant un plus haut niveau de fidélité*.

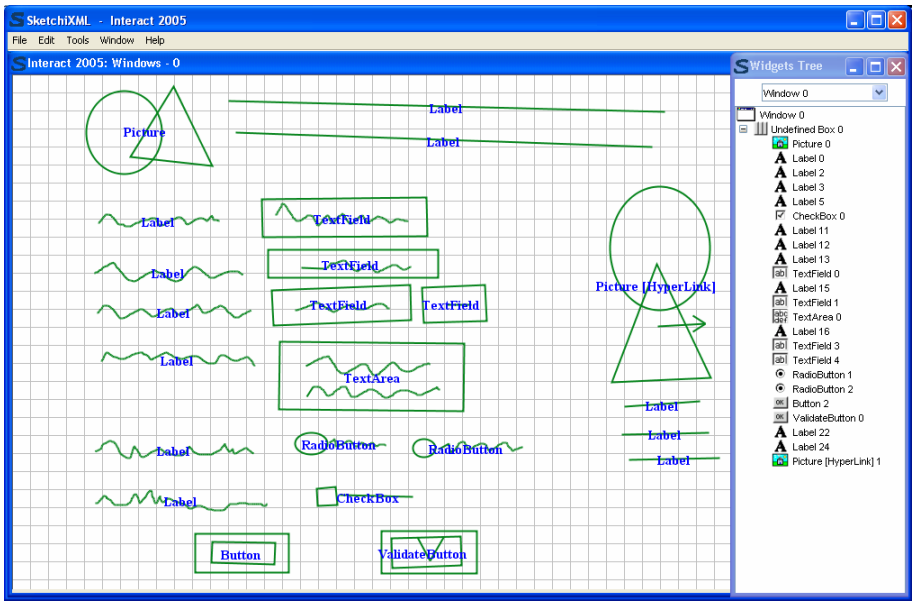


FIGURE 3.4 – L'interface de SketchiXML, avec différentes représentations de widgets

3.1.5 Comparaison des outils analysés

La Table 3.1 compare chaque outil présenté ci-dessus, avec une estimation de chaque attribut décrit au début de cette section.

Attribut	Papier-Crayon	iRise	SketchFlow	SketchiXML
intuitivité	élevée	faible	faible	moyenne
simulabilité	très faible	élevée	élevée	moyenne
complétude	très élevée	moyenne	élevée	moyenne
niveau de fidélité	très bas	haut	moyen	bas
exportabilité	nulle	nulle	faible	moyenne
collaborativité	basse	haute	moyenne	moyenne
apprentissage	très court	long	long	court
temps de conception	très court	long	long	court
prix	gratuit	$\tilde{7}$000$	$\tilde{6}$00$	gratuit

TABLE 3.1 – Comparaison des outils de prototypage analysés

3.2 Les éditeurs d'interfaces

À l'issue de la phase de prototypage, deux types de prototype sont généralement produits : des prototypes jetables (on a terminé le prototype, on le jette et on redéveloppe le tout) ou non jetables (on récupère tout ou partie du prototype pour poursuivre le développement).

Contrairement aux outils de prototypage purs, les éditeurs d'interfaces n'ont pas pour objectif de créer/simuler un prototype, mais de définir l'interface finale utilisée dans un logiciel. A ce titre, leur capacité d'exportation est généralement très bonne, de même que leur niveau de fidélité. Ce sont aussi des logiciels de type WYSIWYG*, mais ils proposent souvent la possibilité d'éditer le code source directement (pour peu qu'il soit human-readable*).

Les techniques utilisées par les éditeurs d'interfaces et celles des outils de prototypage sont généralement similaires. La différence fondamentale entre ces deux classes de logiciels est leur mission : l'une regroupe les logiciels aidant au prototypage pur, alors que l'autre sert à créer des interfaces qui seront réellement utilisées.

3.3 Les langages de description d'interface

VDK : Sections 2.3 et 2.4. Citez bien la référence suivante comme base : http ://cite-seerx.ist.psu.edu

D'après García et al. (2009), un langage de description d'interface, ou UIDL, se définit comme tel (traduction de l'anglais) :
Un UIDL est un langage de haut niveau dont l'objectif est la description des caractéristiques d'intérêt d'une UI [...] afin d'être utilisées durant certaines étapes du développement de celle-ci.*

Le principal intérêt d'un UIDL est un découplage parfait entre Vue et Modèle-Contrôleur dans une architecture typique Modèle-Vue-Contrôleur (MVC)*. Ce découplage apporte plusieurs avantages. Parmi ceux-ci :

- séparer l'UI* des aspects fonctionnels d'un logiciel, évitant la réécriture complète de celle-ci en cas de migration vers un autre langage,
- abstraire certains contrôles de sorte que la modalité d'interaction* utilisée n'ai pas d'impact sur les fonctionnalités du logiciel (exemple : entrer son nom : par clavier ou par voix. L'UI* s'adaptera selon la(les) modalité d'interaction*(s) présente(s)),
- éviter la redondance de code si le logiciel est utilisé sur des plate-formes physiques différentes,
- définir une interface pour un non-programmeur (typiquement : un designer),
- prototyper l'interface graphique en parallèle à son développement fonctionnel.

Une comparaison systématique de ces langages sort du cadre de ce mémoire. Le lecteur intéressé pourra se référer à García et al. (2009) pour une comparaison plus exhaustive.
Nous discuterons donc uniquement d'usiXML, un langage dérivé de l'Extensible Markup Language (XML)*. C'est ce langage de description d'interfaces qui sera utilisé pour l'exportation des prototypes dans usiSketch.

La définition d'usiXML, reprise de son site officiel usixml.org (2010) et traduite, est la suivante :
"UsiXML est un langage compatible-XML* qui décrit une UI* pour de multiples contextes d'utilisation, tels que les interfaces par texte, les interfaces graphiques, les interfaces auditives, et les interfaces multimodales.
En d'autres termes, des applications avec différents types d'interactions, de modalité d'utilisation, de plate formes de calcul, peuvent être décrites indépendamment des caractéristiques de la plate forme physique utilisée."

UsiXML a été validé en mars 2011 par le World Wide Web consortium (W3C)*, ce qui lui donne une bonne visibilité dans le monde de l'informatique, et assure un gage de qualité et une bonne standardisation du langage.

3.4 La reconnaissance de forme

L'objectif de la reconnaissance de forme est de permettre à un ordinateur d'identifier la forme représentée par un dessin à la main. Étant donné qu'un dessin est toujours soumis à interprétation, ces algorithmes ne sont jamais parfaitement précis. Néanmoins, ces derniers présentent aujourd'hui un précision de l'ordre de 80% lorsqu'ils sont adaptés à l'utilisateur (source : (Beuvens et Dullier, 2008, p. 78)). Ce champ de recherche est étroitement lié à l'intelligence artificielle, et plus précisément aux algorithmes de classification.

Il existe de nombreux algorithmes, mais aucun d'entre eux n'est suffisamment générique pour couvrir l'ensemble de la reconnaissance de forme tout en maintenant une précision de reconnaissance acceptable. La plupart du temps, ils sont spécifiques à un certain champ d'application : reconnaissance de symboles, reconnaissance de formes géométriques, reconnaissance de signatures... Il faut donc sélectionner un algorithme qui soit adéquat pour le type de forme à reconnaître.

La plupart des algorithmes sont basés sur un apprentissage supervisé*. Ce sont donc des algorithmes capables de s'adapter selon les exemples qui lui sont fournis au préalable. Deux approches sont généralement utilisées :

- les exemples sont tirés de plusieurs utilisateurs, pour représenter le plus généralement l'espace des formes dessinées ;
- ou les exemples sont tirés d'un seul utilisateur, pour représenter l'espace des formes typiquement dessinées par ce dernier.

Les performances de ces algorithmes sont souvent très supérieures lorsque l'apprentissage est dépendant de l'utilisateur.

Le section 4.3 sera consacré à la reconnaissance de forme. Nous y présenterons quelques algorithmes, ainsi que les algorithmes utilisés dans le logiciel.

3.5 Le matériel de pointage

Bien qu'usiSketch soit utilisable avec un dispositif de pointage traditionnel (souris, trackpad), cet outil est principalement conçu pour être utilisé avec un outil de pointage de type crayon.

Nous séparerons ces dispositifs en trois classes : les tablettes graphiques, les écrans virtuels, et les tablettes PC. Nous comparerons leurs avantages et inconvénients. Cette séparation est faite car l'utilisation d'une classe plutôt qu'une autre dépend fortement du contexte de travail. L'utilisateur seul utilisera de préférence une tablette graphique, alors qu'un groupe de personnes sera plus à l'aise avec un écran virtuel, car l'écran est projeté, donc de taille variable et plus facilement lisible par plusieurs personnes. Un utilisateur nomade (par exemple, un démarcheur commercial) aura par contre une préférence pour une tablette PC, plus compact et légère.

3.5.1 Les tablettes graphiques

Ce type de dispositif, principalement utilisé dans des domaines artistiques, est très performant. Il en existe pour toutes les bourses et de toutes les tailles : de la simple tablette de quelques pouces pour moins de 100 euros à la tablette professionnelle de 21 pouces pouvant atteindre 2000 euros.

Ce type d'outil, initialement destinés au graphistes, existe depuis plusieurs dizaines d'années. Ils sont très performants et ont de nombreuses capteurs : de position, d'inclinaison, de pression... Cependant, d'après Beuvens et Dullier (2008), les tests effectués montrent que les algorithmes de reconnaissance de formes les plus connus ne bénéficient que très peu des capteurs autres que le capteur de position. Dans certains cas, ils ont parfois même un effet néfaste.

Les tablettes haut-de-gamme ont aussi un écran intégré, permettant de travailler directement sur l'écran et donnant une meilleure immersion. Ce type de tablettes est très intéressant pour favoriser l'intuitivité d'usiSketch.

Elles sont aussi relativement facile à transporter. Néanmoins, ces dispositifs ne sont, à la base, pas destinés à être déplacés, mais plutôt à servir comme espace de travail pour un graphiste

Leur principal défaut est la difficulté à travailler à plusieurs sur une même tablette. Il n'existe - à notre connaissance - pas de tablette permettant d'utiliser plusieurs stylets en même temps, et leur taille ne permet pas à plusieurs personnes de travailler efficacement ensemble sur le même écran.

3.5.2 Les écran virtuels

Ce type de dispositif est apparu plus récemment. Concrètement, il s'agit d'un projecteur affichant l'image sur un mur ou une table, associée à un capteur et un stylet. Ils sont difficilement déplaçables et généralement plus chers : ce genre de dispositif coûte environ 5000 euros.

Contrairement aux tablettes graphiques, ces dispositifs permettent une taille d'écran virtuellement infinie. Ils permettent aussi d'utiliser plusieurs stylet en même temps. Ces deux avantages leur permettent un travail collaboratif bien plus efficace qu'avec une tablette graphique.

Malheureusement, ces dispositifs souffrent d'un problème important, lié à la façon dont l'image est affichée. Comme l'écran est projeté, tout objet entre le projecteur et la surface d'affichage y crée une zone d'ombre. Comme l'utilisateur doit dessiner sur cet écran, il en résulte qu'une partie de celui-ci est masquée par sa main et le stylet au mieux. Dans le cas d'une projection sur mur, la silhouette du dessinateur s'affichera elle aussi. C'est la principale raison pour laquelle un utilisateur seul sera plus à l'aise avec une tablette graphique qu'avec un écran virtuel.

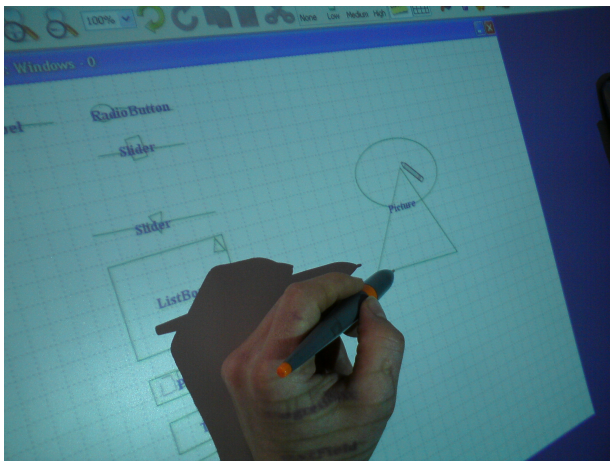


FIGURE 3.5 – Illustration de l'effet d'ombre avec un écran virtuel projeté

3.5.3 Les tablettes PC

Bien qu'existant depuis des années, le marché des tablettes PC s'est étendu au grand public ces dernières années, en particulier grâce à l'iPad d'Apple. Ce type de dispositif pourrait lui aussi être utilisé dans un contexte de consultance nomade.

Remarquons cependant qu'une tablette PC grand public est d'abord prévue pour fonctionner d'une manière tactile, et même s'il existe des styles adaptés, cela peut être un obstacle au déploiement d'usiSketch sur ce genre de dispositif, car la précision des capteurs de ces dispositifs est bien inférieure à celle d'une tablette graphique.

terminer par une conclusion qui résume les limites des familles d'outils analysés et identifie quelques lignes directrices basées sur cet état de l'art.

Deuxième partie

Questions de recherche et Analyse conceptuelle

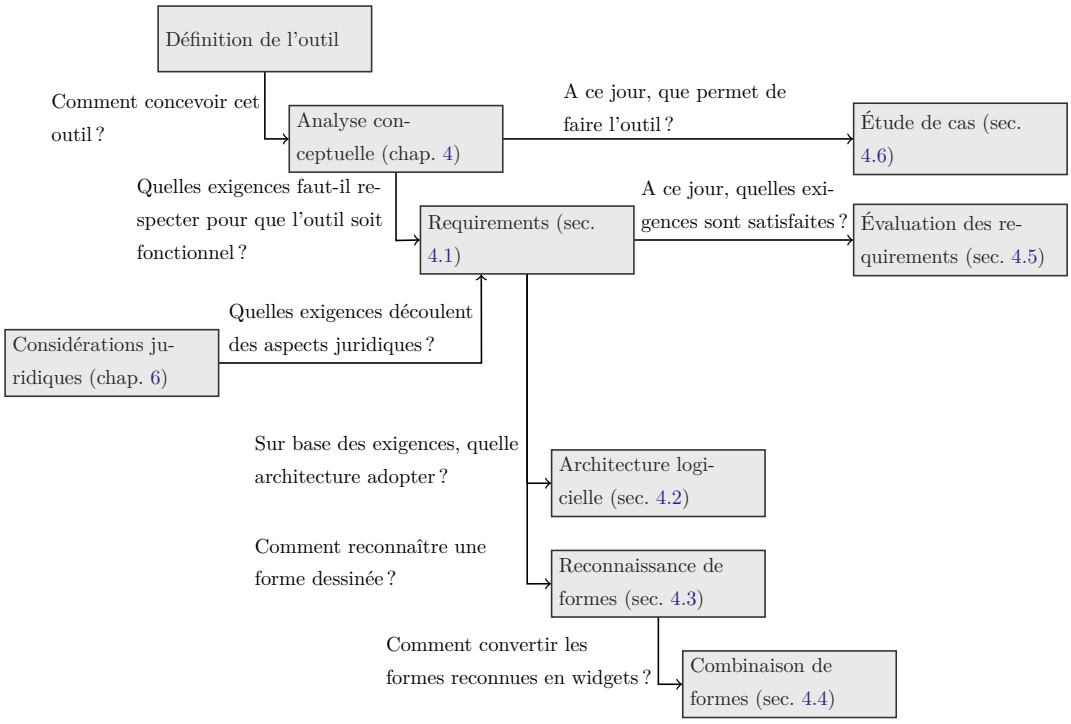
Chapitre 4

Analyse conceptuelle

En tant que logiciel à but commercial, et au vu de sa complexité, usiSketch ne peut être développé à l'aveuglette. En effet, ce logiciel est innovant d'un point de vue technique et requiers une bonne connaissance des UIs* pour être développé. Par conséquent, une analyse conceptuelle rigoureuse est requise pour partir sur de bonnes bases.

Dans ce chapitre, nous traiterons de ces aspects techniques, ainsi que du développement du logiciel. Nous commencerons par y décrire les différentes exigences liées à l'objectif central de ce mémoire. Ensuite, nous analyserons l'architecture utilisée pour développer le logiciel. Nous décrirons en outre plus en détails les aspects liés à la reconnaissance de formes et à leur traduction en widgets*. Nous analyserons enfin l'état d'avancement du développement et nous terminerons par une étude de cas afin d'évaluer l'expérience utilisateur que la version actuelle propose.

La figure ci-dessous illustre le plan de lecture de ce chapitre.



4.1 Requirements

VDK : expliquez la méthodologie suivie pour récolter ces besoins, quelles sources, quel gabarit (inspiré de RADIUS, VOLERE), pourquoi ?

Cette section regroupe les principaux requirements* identifiés pour usiSketch. Le terme "requirement" tel qu'utilisé en informatique n'a pas d'équivalent parfait en français. Il pourrait être approximativement traduit par "requis", "besoin" ou "exigence". Pour cette raison, nous garderons le terme anglais.

Pour une liste complète des requirements*, référez-vous à l'annexe A.

4.1.1 Méthodologie

ajouter req de vérif de la licence

L'élaboration des requirements* fut commencée très tôt, et poursuivie tout le long du développement du logiciel. Nous avons commencé par définir les requirements* principaux, puis nous avons progressivement ajouté les requirements* moins importants. Pour plus de clarté dans l'élaboration de ceux-ci, nous avons scindé cette liste en plusieurs sous-listes.

détaillez toutes les raisons pour lesquelles la récolte des besoins est ici fondamentale.

Chaque requirement* dispose d'une priorité. Cette dernière indique le niveau d'importance du requirement* : vital pour une priorité Haute, important pour une priorité Moyenne, accessoire pour une priorité Basse. Cette structuration par importance nous a permis de définir nos priorités durant le développement du logiciel, afin de développer d'abord les fonctionnalités vitales.

reqs : metodo

4.1.2 Définition d'un requirement

Le modèle de requirement* utilisé s'inspire du modèle VOLERE (2010). Chaque requirement* est défini selon la table suivante :

Numéro	0	**Priorité**	Priorité du requirement : Haut-Moyen-Bas
Type	Type du requirement : fonctionnel, non-fonctionnel	**Responsable**	Personne(s) actuellement responsable(s) du requirement et de sa réalisation
Description	Description du requirement.		
Motivation	Les raisons pour lesquelles le requirement doit être réalisé		
Scénario	Situation durant laquelle le requirement est utile ou souhaité		
Bénéficiaire	Personne bénéficiant de la réalisation du requirement		
Prérequis	Tout requirement nécessaire pour que ce requirement soit réalisable.		

Les bénéficiaires définis sont les suivants :

- Dessinateur : la ou les personnes chargées de dessiner l'interface via usiSketch. Il y a en général un seul dessinateur par prototype.
- Testeur : toute personne ayant accès à l'interface dessinée et capable de la simuler. Il y a en général plusieurs testeurs.
- Concepteur : toute personne chargée de concevoir les scénarios d'utilisation d'un prototype. Il y a en général un seul concepteur par prototype.

réécrire concepteur

4.1.3 Structure des requirements

Nous avons regroupé les requirements* en plusieurs sous-sections. Chacune est responsable d'un aspect logiciel. Pour chaque sous-section, Nous décrirons cet aspect et en donnerons les requirements* principaux.

Les sous-sections définies sont les suivantes :

- Reconnaissance
- Dessin-Rendu
- Prototypage
- Données
- Simulation
- Ergonomie
- Architecture

Reconnaissance

Dans cette section sont regroupés les requirements* liés à la reconnaissance de forme et leur combinaison en formes plus complexes ou en widgets.

Nombre de requirements* fonctionnels identifiés : **6**

Numéro	1	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir reconnaître une forme géométrique dessinée et la convertir sous une forme lissée. Les formes à reconnaître sont : – ligne – ligne en vaguelettes (wavy line) – triangle – rectangle – ellipse – croix		
Motivation	Une forme lissée a un plus haut niveau de fidélité* que son équivalent dessiné à main levée. C'est aussi nécessaire pour la combinaison de formes.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	2	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir combiner plusieurs formes simples en une forme plus complexe ou un widget, selon des règles pré-établies. A chaque fois qu'une nouvelle forme est ajoutée dans une fenêtre, le logiciel doit vérifier s'il est possible de la combiner avec d'autres formes.		
Motivation	Permet de limiter l'ensemble des formes à reconnaître, donc à minimiser l'apprentissage pour la reconnaissance.		
Scénario	Durant le dessin de l'interface, lorsqu'une nouvelle forme est reconnue		
Bénéficiaire	Dessinateur		
Prérequis	requirement 1		

Numéro	4	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Si deux combinaisons mutuellement exclusives sont possibles sur base d'une liste de formes, la combinaison la plus contrainte doit être choisie.		
Motivation	Pour décider quelle est la combinaison à choisir, et avoir le même comportement à tout moment. La combinaison la plus contrainte étant par définition plus difficile à valider, c'est elle qui doit avoir la priorité.		
Scénario	Durant le dessin de l'interface, lorsqu'une nouvelle forme est reconnue et que plusieurs combinaisons sont possibles mais s'excluent mutuellement		
Bénéficiaire	Dessinateur		
Prérequis	requirement 2		

Dessin-Rendu

Dans cette section sont regroupées les fonctionnalités disponibles durant le dessin d'une fenêtre.

Nombre de requirements* fonctionnels identifiés : **8**

Numéro	8	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit être WYSIWYG*.		
Motivation	Le logiciel se veut intuitif, donc la vue doit toujours être en adéquation avec le modèle en cours de dessin.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	9	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit disposer d'un mode de dessin par reconnaissance. Après chaque dessin, le logiciel doit l'analyser et retourner la forme vectorielle reconnue.		
Motivation	Certains widgets sont uniques et spécifiques à une interface, donc tous les représenter est impossible. Il faut donc laisser un espace plus libre pour le dessinateur. Le testeur peut aussi vouloir donner un feedback visuel, via les annotations.		
Scénario	Durant le dessin de l'interface et son test		
Bénéficiaire	Dessinateur		
Prérequis	requirement 1		

Numéro	11	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit disposer d'un mode annotation, où aucune reconnaissance n'est faite.		
Motivation	Certains widgets sont uniques et spécifiques à une interface, donc tous les représenter est impossible. Il faut donc laisser un espace plus libre pour le dessinateur. Le testeur peut aussi vouloir donner un feedback visuel, via les annotations.		
Scénario	Durant le dessin de l'interface et son test		
Bénéficiaire	Dessinateur, Testeur		
Prérequis			

Numéro	12	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit permettre l'effacement de formes préalablement dessinées.		
Motivation	Le dessinateur doit pouvoir effacer une forme placée par erreur ou devenue obsolète.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Navigation

Cette section regroupe les requirements* liés aux interactions entre widgets et fenêtres, la description de scénarios, la structure des widgets, etc.

Nombre de requirements* fonctionnels identifiés : **4**

Numéro	17	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Des actions typiques sur un widget* ou une fenêtre (show, hide, minimize, maximize, setText, ...) doivent pouvoir être définies.		
Motivation	Permettre une simulabilité plus fidèle qu'un simple affichage des fenêtres de façon statique.		
Scénario	Conception de l'interface		
Bénéficiaire	Concepteur, Testeur		
Prérequis			

Données

Cette section regroupe les requirements* concernant les fonctionnalités d'importation/-exportation et de sauvegarde.

Nombre de requirements* fonctionnels identifiés : **4**

Numéro	21	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir exporter un projet usiSketch en fichier usiXML.		
Motivation	L'interface dessinée doit être exportée sous un format standardisé, pour permettre d'éditer le résultat avec un éditeur de plus haute fidélité, choisi par le concepteur.		
Scénario	A la fin du prototypage via usiSketch		
Bénéficiaire	Concepteur		
Prérequis			

Numéro	23	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir sauver/charger un prototype sous un format de fichier propre à usiSketch.		
Motivation	Certaines informations peuvent être pertinentes uniquement pour usiSketch en interne, mais inutiles lors de son exportation. Un format standardisé ne pourra probablement pas sauver ce type de données.		
Scénario			
Bénéficiaire	Tous		
Prérequis			

Simulation

Cette section contient les requirements* concernant la simulation de prototype dans usiSketch et par les testeurs.

Nombre de requirements* fonctionnels identifiés : **4**

Numéro	25	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Une interface définie dans usiSketch doit pouvoir y être simulée sommairement.		
Motivation	Durant la conception, le dessinateur tire avantage de pouvoir simuler son interface sans avoir à l'exporter en usiXML à chaque fois.		
Scénario	Durant la conception d'interface, lorsque le dessinateur veut tester son interface.		
Bénéficiaire	Dessinateur		
Prérequis			

Ergonomie

Cette section contient les requirements* de type ergonomique. La plupart d'entre eux sont non-fonctionnels.

Nombre de requirements* fonctionnels identifiés : **3**

Numéro	30	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Un maximum d'opérations de prototypage doivent être accessible via le pointeur.		
Motivation	Le logiciel étant destiné à être utilisé avec une tablette graphique, il faut limiter les échanges stylet-clavier ou stylet-souris.		
Scénario	Durant le dessin d'une interface		
Bénéficiaire	Desinnateur		
Prérequis			

Numéro	33	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le dessinateur doit pouvoir annuler/refaire une action contenue dans l'historique de conception du prototype.		
Motivation	La même que pour tout logiciel de création de contenu : parce que c'est extrêmement pratique. Dans ce cas précis, cela permet de modifier une interface sans avoir à se soucier si le résultat obtenu après modification est accepté ou non : s'il ne l'est pas, il suffit de revenir en arrière.		
Scénario	Durant la description de l'interface, si le dessinateur veut revenir en arrière		
Bénéficiaire	Dessinateur		
Prérequis	requirement 32		

Architecture

Cette section contient les requirements* relatifs à l'architecture logicielle à utiliser.

Aucun requirement fonctionnel n'est défini ici. Il s'agit de règles de développement à respecter pour faciliter le développement et la maintenance du logiciel.

Numéro	35	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel de dessin/conception est développé sous la forme d'un plugin eclipse.		
Motivation	Eclipse est un outil bien connu des développeurs, ce qui facilite l'apprentissage d'usiSketch. De plus, cet outil dispose déjà de vues utiles pour usiSketch. Le développement s'en voit accéléré.		
Scénario	Changement de l'algorithme de reconnaissance		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	36	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit être utilisable quel que soit l'Operating System (OS)* utilisé.		
Motivation	S'assurer que le logiciel puisse fonctionner sur n'importe quel ordinateur évite de devoir se cantonner à un OS* précis, et donc de perdre d'éventuelles parts de marché.		
Scénario	Utilisation sur différents OS*		
Bénéficiaire	Tous		
Prérequis			

Numéro	37	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	L'outil de dessin/conception et l'outil de simulation doivent être construits comme deux programmes utilisables séparéments.		
Motivation	Le testeur n'a pas le droit de modifier l'interface, il n'a donc pas besoin de l'outil de dessin, mais uniquement d'avoir accès à l'interface et la simuler.		
Scénario	Utilisation collaborative		
Bénéficiaire	N/D		
Prérequis			

Après ces requirements, donner une présentation générale dans laquelle vous faites ressortir ce qui vous semble important, sensible, critique, etc. Il ne faut pas laisser le lecteur simplement avec une liste de requis.

4.2 Architecture logicielle

Partant du requirement 37 : "L'outil de dessin/conception et l'outil de simulation doivent être construits comme deux programmes utilisables séparéments", nous ne nous sommes concentrés que sur l'outil de dessin/conception, et non celui de simulation à distance. Le développement de l'interface de simulation à distance est un autre logiciel à part entière, possiblement dans un langage de programmation différent. De plus, il devrait être construit sur base d'usiXML et non d'usiSketch, et ce pour plusieurs raisons :

- Les données internes au format usiSketch ne sont pas utiles lors de la simulation. Ces données sont les formes dessinées et l'historique de conception.
- Le prototype développé sous usiSketch n'a pas un bon niveau de fidélité*. Il sera sans doute nécessaire de modifier le fichier usiXML exporté pour améliorer ce niveau. Or, si le logiciel de test est développé sous usiSketch et non usiXML, il n'est pas possible de modifier le prototype avec un logiciel présentant un meilleur niveau de fidélité* et de le re-simuler ensuite.

Concernant le développement du logiciel, nous devons respecter le requirement 35 : "Le logiciel de dessin/conception est développé sous la forme d'un plugin eclipse". Le langage utilisé est donc Java, qui est un langage indépendant de l'OS* utilisé. Cette particularité valide le requirement 36 : "Le logiciel doit être utilisable quel que soit l'OS* utilisé", pour autant qu'aucune librairie dépendante de l'OS* ne soit utilisée.

4.2.1 Librairies

usiSketch utilise les librairies suivantes :

- eclipse Sketch : cette librairie, développée dans le laboratoire de M. Vanderdonckt sous la forme d'un plugin eclipse*, contient les algorithmes de reconnaissance de forme utilisés dans usiSketch.
- usiXML : cette librairie fut générée grâce à Castor, c'est elle qui contient les classes Java permettant la liaison entre Java et usiXML. Le script de génération de cette librairie se trouve en annexe B.
- Castor : cette librairie assure la liaison entre XML* et Java. Elle est utilisée conjointement à la librairie usiXML pour le parsing* et l'exportation d'un fichier usiXML. Elle dépend des deux librairies Commons-Logging et JDOM.

– Commons-logging : cette librairie est utilisée pour la journalisation d'événements lors de l'exportation vers usiXML.
– JDOM : cette librairie est utilisée par la librairie usiXML. Elle est là par souci de rétro compatibilité.

4.2.2 Principaux modules

L'architecture générale s'inspire de l'architecture MVC*. La Figure 4.1 regroupe tous les principaux modules d'usiSketch.

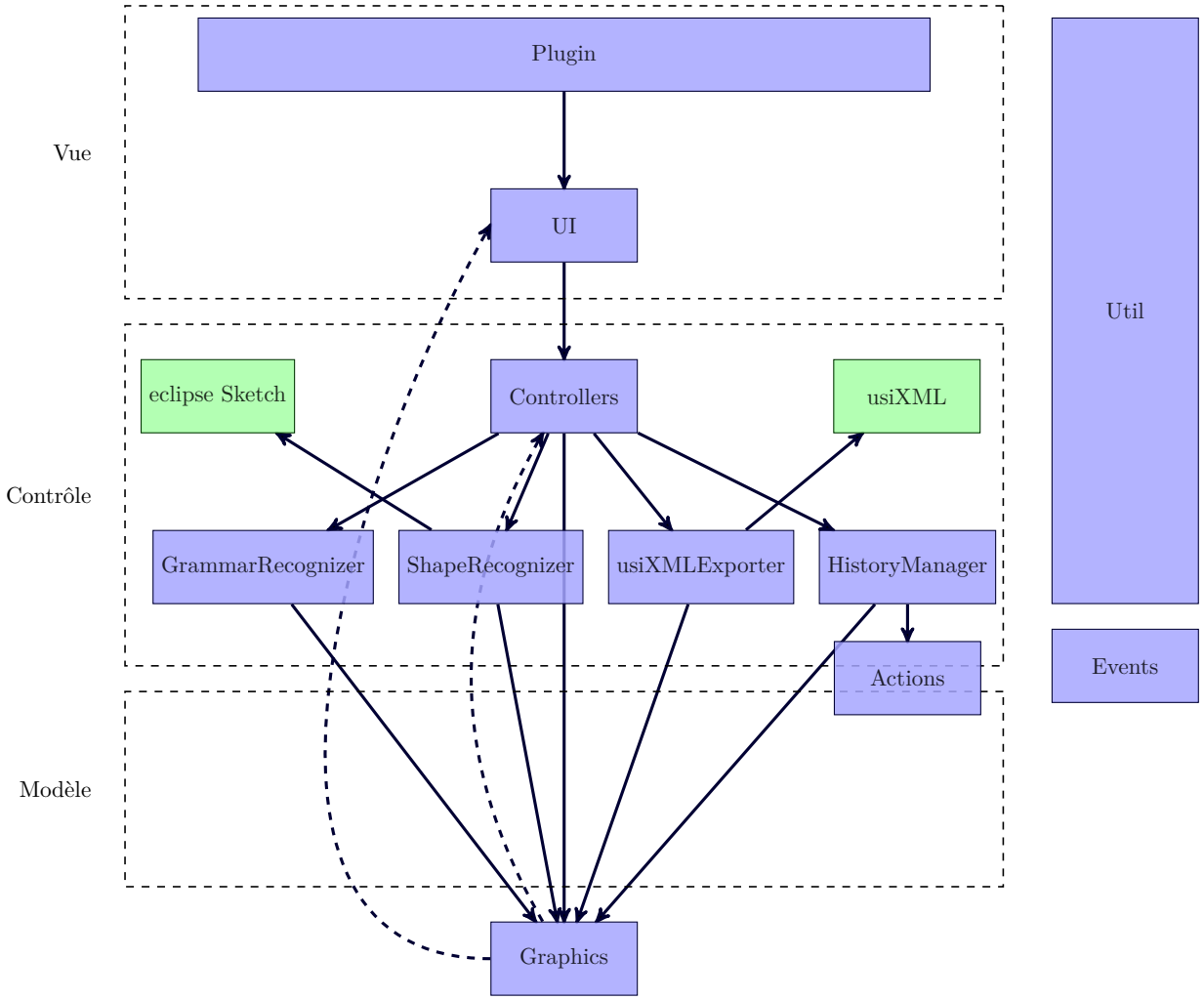


FIGURE 4.1 – Représentation en couches des modules d'usiSketch, côté conception
Légende : En bleu, les modules ; en vert, les librairies principales. Les flèches non-pointillées sont des appels directs d'un module vers un autre ; les pointillées sont des envois d'événements d'un module, captés par un autre. Les appels vers les modules Util et Events ne sont pas représentées.

finir cette figure

finir de décrire chaque module

Modèle :

Ces modules forment l'ensemble des données manipulées par usiSketch. Il regroupe les éléments graphiques et l'historique.

Graphics

Ce module contient l'ensemble des éléments graphiques utilisés par le logiciel, ainsi que des structures pour les stocker. Il ne s'agit pas ici des composants affichés à l'écran, mais bien des données nécessaires à leur dessin (position, taille, points...).

Ces éléments sont décomposés en quatre catégories, dont les trois premières contiennent les éléments graphiques héritant de la classe *Graphic* :

1. *DotsSet* : représente un dessin à la main. Il s'agit d'un ensemble de points reliés entre eux sous forme d'une courbe.
2. *VectorialShape* : représente une forme vectorielle. Plusieurs classes héritent de *VectorialS* une par type de forme définie dans le requirement 1 : "Le logiciel doit pouvoir reconnaître une forme géométrique dessinée et la convertir sous une forme lissée".
3. *Widget* : représente un widget*. Plusieurs classes héritent de *Widget*.

La quatrième catégorie reprends les structures de stockage des éléments.

figure représentant la structure des éléments graphiques

Pour stocker les éléments graphiques autres que *DotsSet*, une structure spécialisée a été développée. Elle est conçue pour remplir le requirement 16 : "Le système doit permettre plusieurs niveaux de fidélité*, changeables à tout moment de la conception". La Figure 4.2 est une représentation de cette structure de données.

Il s'agit d'un arbre inversé, dont les noeuds sont des éléments graphiques. En particulier, les feuilles sont des formes vectorielles reconnues par le moteur de reconnaissance de formes. L'ajout de forme dessinée se fait toujours au niveau des feuilles.
La structure dispose aussi d'une méthode *compose*(), qui effectue la composition de formes en une autre forme. La forme composée devient alors le parent des noeuds qui le composent. Cette méthode est exclusivement appelée par le moteur de composition de formes.

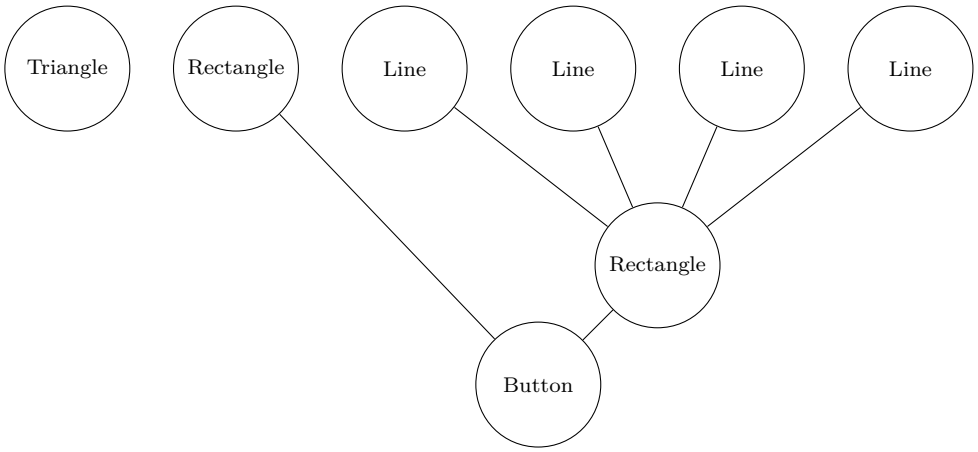


FIGURE 4.2 – Structure de données des éléments graphiques - exemple

Lors du rendu, le parcours de cette structure se comporte différemment selon le niveau de fidélité.

- niveau “fidélité widgets” et supérieur : seuls les noeuds racines sont parcourus.
- niveau “fidélité formes” : les noeuds enfants sont dessinés si et seulement si le noeud parent est un widget. Dans ce cas le noeud parent n’est pas dessiné.
- niveau “fidélité dessin” : seules les feuilles sont dessinées, sous leur forme dessinée réellement.

remettre l’exemple ci-dessus et colorer les noeuds qui sont affichés, par niveau de fidélité

Actions

Ce module regroupe toutes les actions annulables. Ce module est à la frontière du modèle et du contrôle. En effet, les classes qu’il contient définissent les méthodes *undo*() et *redo*(), qui implémentent l’action à effectuer. Elles sont aussi utilisées lors de la sauvegarde du fichier, pour remplir le requirement 34 : “Un prototype sauvé sous format usiSketch doit contenir son historique de conception et de dessin, possiblement compressé”.

Cette double appartenance fut faite pour éviter la redondance de code : la méthode *redo*() implémentant l’action à effectuer, et *undo*() son annulation, il était inutile de réécrire le même code dans les contrôleurs, alors qu’il était déjà défini dans ces actions. Cela assure un comportement cohérent et une maintenance facilitée.

Ces classes sont créées par $HistoryManager$, et stockées dans un historique, implémenté par la classe $HistoryList$ contenue dans ce module.

Vue :

Ces modules forment l'ensemble des aspects visuels d'usiSketch. Il contiennent les vues implémentées dans eclipse* ainsi que les branchements faits en tant que plugin.

Plugin

Ce module contient l'ensemble des classes responsables du branchement d'usiSketch à eclipse*. Il est responsable du chargement des vues, de l'ouverture de fichiers, etc.

UI

Ce module contient l'ensemble des vues de l'application. Chaque vue est chargée de gérer les événements qui surviennent chez elle, et transmettent l'information aux contrôleurs concerné, selon l'événement.

Quatre vues ont été définies :

1. windowview : cette vue est chargée de gérer le dessin de formes et de widgets.
2. conceptionview : cette vue est chargée de gérer les interactions entre widgets.
3. historyview : cette vue est chargée de lister l'historique de conception, et de permettre à l'utilisateur de revenir à un état précédent par simple clic.
4. grammarview : cette vue est chargée de lister l'ensemble des grammaires pour la composition de widgets. Elle a pour but de remplir le requirement 6 : "Le dessinateur doit pouvoir spécifier les règles de composition à l'aide d'une interface simple et intuitive".

Contrôle :

Les modules de contrôle sont responsables des opérations fonctionnelles faites dans usiSketch. Il peut s'agir d'opérations de reconnaissance, de combinaison, d'actions utilisateurs, etc.

Controllers

Ce module contient tous les contrôleurs de l'application. Ce sont eux qui sont chargés de router les actions aux différents modules de reconnaissance. Les principales classes de ce modules sont :

- SketchingController : ce contrôleur gère toute action faite sous forme de dessin. Il peut s'agir de l'ajout d'une esquisse, d'une commande d'effacement ou de sélection.
- ActionController : ce contrôleur gère les actions faites sur une interface. Plus concrètement, il s'agit des boutons de contrôle présents dans la vue windowview.
- HistoryController : ce contrôleur gère les actions faites dans la vue de l'historique.

ShapeRecognizer

Ce module est la réalisation du requirement 1 : "Le logiciel doit pouvoir reconnaître une forme géométrique dessinée et la convertir sous une forme lissée". Il s'agit d'un thread* chargé de reconnaître une forme sur base d'un ensemble de points (la class *DotsSet* du module *Graphics*). Ce module utilise la libraire eclipse Sketch. Son implémentation est discutée à la section 4.3.

GrammarRecognizer

Ce module est la réalisation du requirement 2 : "Le logiciel doit pouvoir combiner plusieurs formes simples en une forme plus complexe ou un widget, selon des règles pré-établies". Il s'agit d'un thread* chargé de combiner plusieurs éléments graphiques en un élément plus complexe : forme ou widget. Son implémentation est discutée à la section 4.4.
Ce module est fortement lié à la structure de donnée des éléments graphiques. C'est ce module qui appelle la méthode *compose*() de la structure de données décrite plus haut.

HistoryManager

Ce module est la réalisation du requirement 32 : "Le logiciel doit conserver un historique de conception et de dessin d'un prototype à tout moment". Il est chargé de tenir à jour l'historique de conception et gère les actions de retour arrière et de retour avant.

Autres :

Les modules qui ne sont dans aucune de ces catégories servent au bon fonctionnement de l'architecture. Il s'agit des utilitaires et des événements.

Util

Ce module regroupe différentes classes utilisées un peu partout dans le code. Les classes de ce module sont conçues pour être génériques, donc réutilisables dans n'importe quel projet Java sans réécriture.

Events

L'architecture d'usiSketch utilise un système d'événements pour informer un changement des couches inférieures aux couches supérieures. Par exemple, lorsqu'une forme est ajoutée dans la liste des formes d'une fenêtre, le conteneur envoie un événement aux vues et contrôleurs concernés. Le contrôleur demandera alors au module *GrammarRecognizer* de rechercher de nouvelles correspondances, dans l'espoir d'identifier un nouveau widget*.

La gestion des événements se trouve dans ce module. Les classes responsables de l'envoi et la réception d'événements sont *EventTriggerer* et *EventListener*. Les autres classes sont des événements.
Il s'agit d'un design pattern typique d'Observateur-Observable.

4.3 La reconnaissance de formes

Cette section traite de la reconnaissance de formes, définie par le requirement 1 : "Le logiciel doit pouvoir reconnaître une forme géométrique dessinée et la convertir sous une forme lissée".

Une des principale fonctionnalités d'usiSketch est sa capacité à reconnaître et traduire un geste manuscrits en formes géométriques. Ces formes sont ensuite traduites en widgets grâce à la combinaison de formes (voir section 4.4).

Nous ferons une rapide description des algorithmes les plus intéressants pour usiSketch, pour ensuite sélectionner celui ou ceux qui semblent le(s) plus adéquat(s). Nous discuterons ensuite de l'implémentation de l'algorithme et des modifications qui y ont été apportées pour en améliorer les performances.

4.3.1 Les types de gestes manuscrits

Un geste manuscrit est une suite temporalisée d'un ou plusieurs trait. Le mot trait utilisé ici signifie "une ligne continue sur la surface de dessin".

Beuvens et Dullier (2008) définissent les différents types de gestes de la façon suivante : "Les gestes peuvent être définis comme des mouvements. [...] Il existe donc différents types de gestes qui sont potentiellement étudiables et reconnaissables algorithmiquement. [...]

Dans la catégorie des gestes manuscrits il est possible de faire des distinctions entre différents types des symboles :

- lettres et chiffres : les lettres de l'alphabet en majuscule ou minuscule, avec ou sans accent, etc. ;
- gestes d'action : principalement des segments de droits, des arcs etc., mais cela peut aussi être des symboles comme par exemple l'arobas @ ;
- formes géométriques : différentes formes géométriques, doublées ou non, etc. ;
- signature : la signature d'une personne étant sensée être relativement difficile à imiter, il est intéressant de pouvoir la reconnaître, de manière à authentifier la personne l'exécutant.

De plus, les gestes peuvent être faits d'un seul ou plusieurs traits."

4.3.2 Les types de reconnaissance de forme

Il existe deux grands catégories de reconnaissance de forme :

- la reconnaissance “online” : la reconnaissance se fait durant la construction de l’image.
- la reconnaissance “offline” : la reconnaissance ne se fait que lorsque l’utilisateur la demande explicitement, généralement à la fin de la construction de l’image.

La reconnaissance online est généralement plus directe et intuitive, mais elle nécessite que le résultat soit dans un état cohérent à tout moment, et le système doit parfois revenir sur ses décisions précédentes. De plus, la reconnaissance doit être suffisamment rapide pour ne pas gêner le dessinateur durant celle-ci, et est donc sensible du point de vue calculatoire.
La reconnaissance offline est plus simple et moins restreinte du point de vue calculatoire (une fois le dessin fini, si la reconnaissance prends une minute ou plus, ce n’est pas grave). Elle est toutefois moins intuitive, car le niveau de fidélité* reste bas jusqu’à la reconnaissance elle-même. De plus, cette reconnaissance étant non-déterministe par essence, le résultat peut être erroné. L’utilisateur ne pourra s’en rendre compte que tard durant la conception, ce qui est un désavantage.

4.3.3 Contexte d’utilisation d’usiSketch

Comme nous l’avons mentionné à la section 3.4, les algorithmes de reconnaissance de formes ne sont pas tous adaptés au même contexte d’utilisation. Par conséquent, il est important de définir celui d’usiSketch, afin de choisir le ou les algorithmes utilisés par ce dernier.

Le requirement 8 : “Le système doit être WYSIWYG*” implique que la reconnaissance doit se faire au moment du dessin, et non a la fin de celui-ci. La reconnaissance “online” est donc bien plus adéquate que la reconnaissance “offline”. Cette dernière est néanmoins utile pour remplir le requirement 7 : “Le logiciel doit pouvoir importer une image de prototype basse fidélité dans projet usiSketch”. Ce dernier étant moins prioritaire, nous nous concentrerons dans cette section sur la reconnaissance “online”.

Les éléments de haute fidélité d’usiSketch sont des widgets*. Ces derniers sont très nombreux, et de nouveaux apparaissent fréquemment. Par conséquent, et puisque les

algorithmes de reconnaissance sont basés sur un apprentissage supervisé*, vouloir reconnaître tous ces widgets individuellement nécessiterait que l'utilisateur entraîne l'algorithme pour chacun d'entre eux. Un tel processus serait fastidieux, et ne serait jamais terminé : pour chaque nouveau widget*, il faudrait réitérer le processus.
Néanmoins, tous ces widgets* ont un point commun : leur représentation basse fidélité est une composition de formes géométriques simples. Pour cette raison, nous avons décidé d'implémenter la reconnaissance de forme usiSketch de sorte qu'elle ne reconnaisse que ces formes géométriques, réduisant considérablement le nombre de formes à reconnaître, et *a fortiori* le temps requis pour entraîner l'algorithme. Ces formes sont par la suite combinées selon des grammaires prédéfinies. Cette combinaison est décrite à la section 4.4.

Le requirement 19 : "Si présent, le texte contenu dans les widgets doit pouvoir être défini, idéalement par reconnaissance de texte", demande aussi une implémentation du logiciel avec la reconnaissance de texte. Cependant, ce type de reconnaissance utilise d'autres algorithmes. Faute de temps, nous n'avons pas implémenté ce type de reconnaissance, aussi nous ne traiterons pas ce sujet.

4.3.4 Les algorithmes retenus

Il existe de nombreux algorithmes de reconnaissance de forme, et la recherche dans le domaine est actuellement très intensive. Toutes les présenter ici sort du cadre de ce mémoire. Nous ne nous concentrerons donc que sur les algorithmes suffisamment documentés et dignes d'intérêt pour usiSketch.

Nous présenterons quatre algorithmes génériques : le filtrage par motif, Rubine, One-Dollar, et la reconnaissance par distance de Levenshtein. Ces descriptions sont extraites de Beuvens et Dullier (2008), nous invitons le lecteur à s'y référer pour plus de détails.

Tous les algorithmes représentés ici, à l'exception du filtrage par motif, sont des algorithmes basés sur l'apprentissage supervisé*. Ces algorithmes fonctionnent sur deux phases : une phase d'apprentissage et une phase de classification.

Durant la phase d'apprentissage, l'utilisateur fournit à l'algorithme plusieurs exemples représentatifs d'une classe (ici, il dessines plusieurs formes géométriques et indique à l'algorithme de quelle forme il s'agit). L'algorithme construit alors un modèle, utilisé dans la phase de classification.

Durant la phase de classification, l'utilisateur fournit de nouveaux éléments à l'algorithme (ici, il dessine des formes sans spécifier leur classe). L'algorithme utilise alors le modèle préalablement construit pour sélectionner la forme correspondante.

Rubine

Cet algorithme est, historiquement, un des premiers algorithmes de reconnaissance de forme spécifié. Il a été décrit la première fois dans Rubine (1991), et permet d'analyser les gestes faits d'un seul trait.

Pour chaque geste, un vecteur de caractéristiques est extrait. Selon la phase en cours (entraînement ou classification), ce vecteur est soit ajouté au modèle, ainsi que la classe de la forme, soit comparé avec les caractéristiques du modèle. Dans l'implémentation initiale, 13 caractéristiques sont extraites.

Chaque classe est définie par un ensemble de gestes-exemples, définis durant l'étape d'apprentissage. La classification se fait par une séparation linéaire du geste entré avec les gestes-exemples. Pour chaque classe, un score est calculé. La classe ayant le score le plus élevé est la classe assignée au geste.

One-Dollar

L'algorithme One-Dollar se base sur la comparaison de l'ensemble des points constituant le geste à reconnaître avec les gestes établis durant l'apprentissage. Pour ce faire, chaque ensemble de points (geste) est transformé suivant quatre phases consécutives :

1. rééchantillonnage : les points sont placés sur une grille, pour éliminer les problèmes d'échantillonnage différents entre deux gestes,
2. rotation : le geste subit une rotation de sorte que l'angle formé par son barycentre et son premier point égale 0,
3. redimensionnement : le geste est redimensionné suivant une fenêtre de référence de taille fixée,
4. translation : le centroïde du geste est translaté vers un point de référence, typiquement $(0, 0)$.

Une fois ces transformations effectuées, le geste est comparé avec chaque geste-exemple donné durant l'entraînement. La classe du geste-exemple le plus proche est sélectionnée comme classe du geste à classifier.

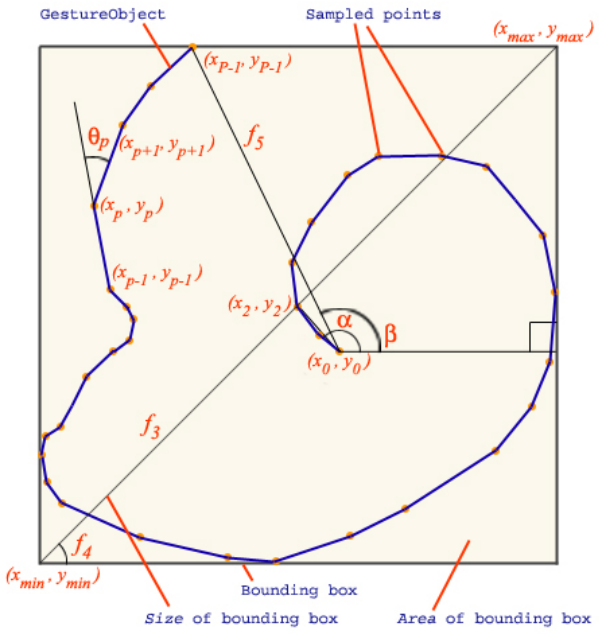


FIGURE 4.3 – Rubine : exemple de traduction d'un dessin en caractéristiques ; source : Rubine (1991)

Le filtrage par motif

Cette technique utilise l'image finale comme donnée. Une liste de formes de base est définie au préalable.

Durant le processus de reconnaissance, une image vierge est utilisée comme point de départ. La reconnaissance est effectuée comme une reconstruction de l'image par une composition de formes de base, à une ou plusieurs transformations près (typiquement : rotation, translation, échelle). Une fois que l'image reconstruite est suffisamment proche de l'image originale, le processus de reconnaissance s'arrête, et la liste des formes la composant est retournée.

Cette technique requiert uniquement une image comme donnée d'entrée, ce qui la rend très utile pour interpréter une image préétablie (par exemple, un dessin papier scanné). Elle est néanmoins coûteuse en temps, ce qui la rend peu praticable dans le cas d'une reconnaissance "online".

La distance de Levenshtein

La distance de Levenshtein est une technique permettant de calculer l'edit-distance* entre deux chaînes de caractères. Cette méthode, présentée par Coyette et al. (2007), consiste en la traduction d'une esquisse en une chaîne de caractères, chacun de ceux-ci représentant un des huit points cardinaux. La Figure 4.4 illustre cette traduction.

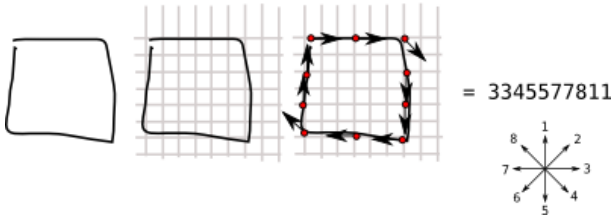


FIGURE 4.4 – Exemple de traduction d'un dessin en chaîne ; source : Sangiorgi (2010b)

Une fois la chaîne calculée, il suffit d'utiliser l'algorithme de Levenshtein pour comparer un geste avec tous les gestes référencés dans le set d'entraînement. Le type du geste d'entraînement le plus proche (selon la distance de Levenshtein) est retenu.

Cette technique nécessite de capturer le mouvement du pointeur lors du dessin. Elle n'est donc pas adaptée pour analyser une image préétablie, mais convient parfaitement pour un usage "online".

4.3.5 Comparaison des algorithmes

Cette comparaison est tirée de (Beuvens et Dullier, 2008, pp.81-86). Comme nous pouvons le voir à la Figure 4.5, l'algorithme de Levenstein et celui De One-Dollar sont globalement équivalents. L'algorithme de Rubine est moins efficace lorsque peu de formes sont données pour entraîner l'algorithme, mais atteint les même performances que les deux autres algorithmes lorsque l'échantillon d'entraînement est plus grand. De plus, la version classique de Rubine est incompatible avec un dessin multi-trait. Par le requirement 10 : "Le système de reconnaissance de forme doit pouvoir gérer le multi-traits", cet algorithme ne semble pas être un bon candidat.
La précision du filtrage par motif est bien plus délicate à évaluer. En effet, elle sera d'autant plus grande que le nombre de formes de base (prédéfinies dans le code) et de transformations applicables. Néanmoins, comme nous le verrons plus loin, la complexité calculatoire étant très élevée, cet algorithme n'est pas un bon candidat non plus pour

usiSketch.

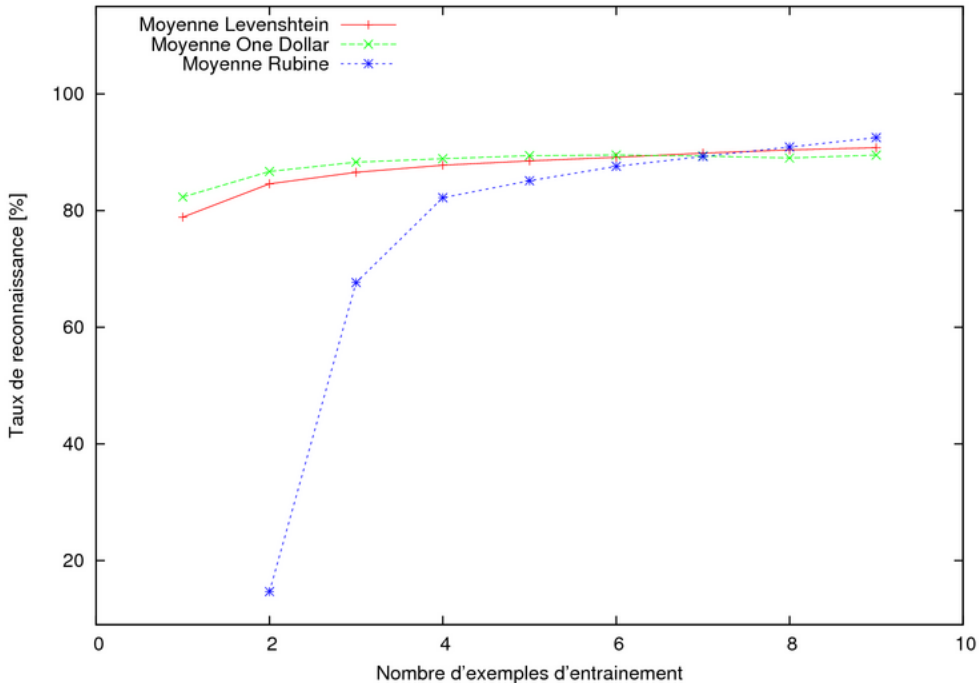


FIGURE 4.5 – Précision des algorithme pour des formes géométriques (source : Beuvens et Dullier (2008))

Il nous reste donc les algorithme de Levenstein et celui de One-Dollar, dont la précision et la rapidité d'exécution en font de bons candidats. Comme l'algorithme de Levenshtein fut élaboré et est toujours en cours d'amélioration dans le laboratoire de M. Vanderdonckt, nous avons opté pour ce dernier. Ainsi nous avions accès à une excellente expertise concernant l'algorithme, ainsi qu'à son implémentation originale. Nous avons finalement repris l'implémentation du projet eclipse Sketch (Sangiorgi (2010a)), que nous avons modifié selon nos besoins.

4.3.6 Notre implémentation

Comme mentionné ci-dessus, nous utilisons l'algorithme de Levenshtein comme base. Nous appliquons un pré-traitement sur chaque geste avant d'appeler l'algorithme, pour en améliorer la précision. Ce pré-traitement est générique et peut en principe être réutilisé pour un autre logiciel, donc le contexte peut varier. Pour cette raison, nous avons

implémenté cet algorithme dans un branchement* du projet eclipse Sketch. Ce branchement* se trouve à Olivier Bourdoux (2011).

Ce pré traitement s'effectue en trois étapes : Rééchantillonnage, Conversion en chaîne, Mise à l'échelle, Rotation.

Rééchantillonnage

D'après Beuvens et Dullier (2008) : "Comparer deux gestes dans leur forme initiale point à point ne serait toutefois pas pertinent. Pour deux gestes qui sont similaires, la vitesse de l'utilisateur, couplée a la sensibilité hardware aussi bien que software, peut faire en sorte que la fréquence d'apparition des points soit différente. [...] Considérons a titre d'exemple la Figure 4.6 :

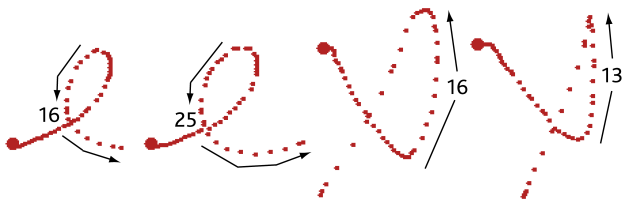


FIGURE 4.6 – Exemple de disparités entre gestes similaires
(source : Beuvens et Dullier (2008))

On peut se rendre compte que les deux premiers ensembles de points définissent le même geste malgré une fréquence d'apparition de ces points différente."

Nous effectuons donc un rééchantillonnage du geste afin de régler ce problème. Contrairement à Beuvens et Dullier (2008), nous ne définissons pas à l'avance le nombre de points à obtenir, mais la distance entre chacun d'entre eux. En effet, cela permet de conserver une ressemblance entre la forme rééchantillonnée et l'originale, quelle que soit la longueur du geste. L'algorithme de Levenshtein permet cette flexibilité, alors que One-Dollar nécessite un nombre de points pré-établi. La Figure 4.7 illustre le processus.

Conversion en chaîne

La conversion en chaîne fonctionne comme illustré à la Figure 4.4. Cela fonctionne bien pour un geste d'un trait, mais cela doit être implémenté avec un geste multi-traits avec plus de précautions.

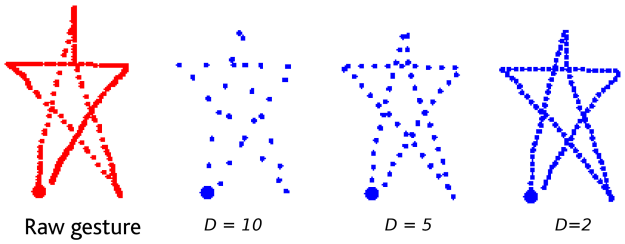


FIGURE 4.7 – Exemple de rééchantillonnage
(source : Beuvens et Dullier (2008))

L'algorithme de Levenshtein fonctionne grâce à la direction relative entre deux points. Par conséquent, le multi-traits est gérable nativement : il suffit de considérer un geste comme une suite temporalisé de points. Le dernier point d'un trait et le premier du suivant sont donc mis bout-à-bout et la conversion en chaîne peut fonctionner. Cependant, cette chaîne ne représente pas bien le geste réel. Comme illustré à la Figure 4.8, si l'on considère uniquement un geste comme un ensemble de points rééchantillonnés, la chaîne produite varie trop peu. Dans l'exemple de cette figure, le premier geste est un rectangle effectué en deux traits. La deuxième représente une simple ligne brisée. Les chaînes retournées sont très similaires : seul le '8' en rouge est ajoute au geste multi-traits. La distance de Levenshtein entre ces deux chaînes est de 1 : il suffit d'ajouter/enlever le '8' pour passer d'une chaîne à l'autre. Pourtant, ces formes ne se ressemblent pas.

Pour remédier à ce problème, nous adoptons une approche différente : au lieu de simplement mettre deux traits bout-à-bout, une ligne imaginaire est tracée entre le dernier point d'un trait et le premier du suivant. Cette ligne imaginaire représente le chemin le plus court que doit faire le pointeur pour atteindre la position du trait suivant. De cette manière, la conversion en chaîne conserve une trace du déplacement effectué sans que la pointe du stylet ne touche la surface de dessin. La Figure 4.9 illustre ce type de conversion. Comme nous pouvons le voir, le '8' unique est remplacé par cinq '8' consécutifs. La distance de Levenshtein retourne alors 5. Les chaînes obtenues en multi-traits sont donc plus réalistes et plus simples à distinguer avec cette méthode.

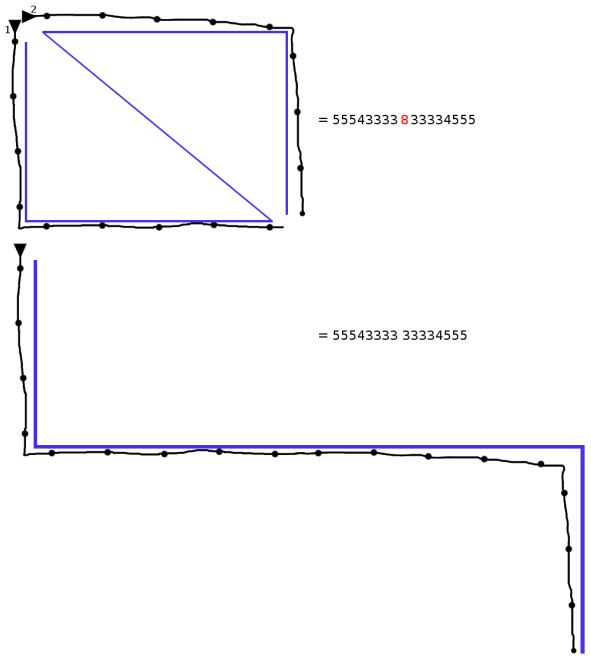


FIGURE 4.8 – Illustration de l'ambiguïté en multi-traits

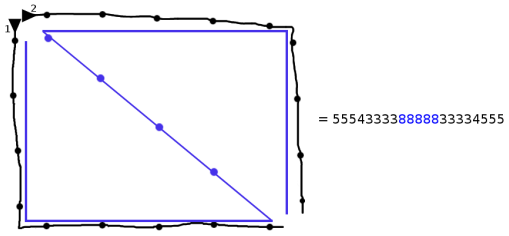


FIGURE 4.9 – Illustration de la ligne imaginaire en multi-traits

Mise à l'échelle

Notre méthode d'échantillonnage retourne un nombre de points variable selon la longueur du geste. Par conséquent, deux formes identiques mais à une échelle différente auront un nombre de points rééchantillonnés différent, et donc des chaînes de longueur différentes. Une propriété de la distance de Levenshtein stipule que la borne minimale de la distance entre deux chaînes est égale à la différence de longueur entre ces deux chaînes :

$$\min(Levenstein(\sigma_1, \sigma_2)) = |\text{length}(\sigma_1) - \text{length}(\sigma_2)|$$

En effet, la modification la plus courte entre deux chaînes est la suppression d'autant de caractères que la différence de leur longueur. Si après ces suppressions, les chaînes sont identiques, la distance de Levenshtein vaut cette différence.

Sur base de cette propriété, lorsque deux gestes de longueur très différente sont comparés, la distance de Levenshtein sera toujours grande, et ce quelle que soit la ressemblance des formes à l'échelle près. La Figure 4.10 illustre ce problème : les deux formes sont très semblables, pourtant la distance de Levenshtein minimale est de $60 = 72 - 12$.

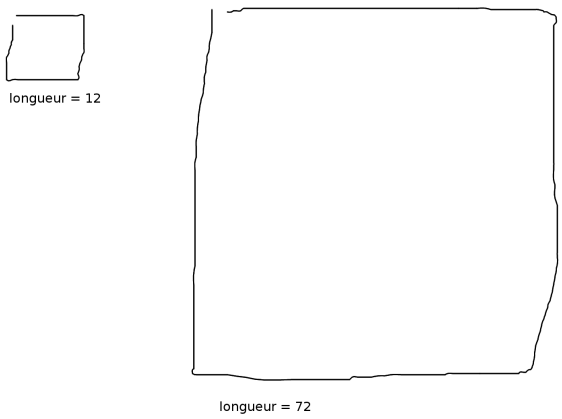


FIGURE 4.10 – Illustration du problème de l'échelle

Pour remédier à ce problème, nous utilisons une approche qui permet d'agrandir artificiellement une chaîne tout en conservant l'aspect général du geste. Avant de comparer deux chaînes, la plus courte des deux est étendue à la longueur de l'autre. De cette façon, $\min(Levenstein(\sigma_1, \sigma_2)) = 0$ et tous les gestes de l'ensemble d'entraînement ont la même probabilité d'être retenus.

Pour étendre une chaîne de longueur l en une chaîne de longueur l', nous utilisons l'algorithme décrit à la Table 4.1. En substance, il duplique un caractère tous les *step* caractères de la chaîne, jusqu'à ce que la nouvelle chaîne aie la longueur voulue. Par exemple, la chaîne 111222333444 (de longueur 12) étendue à la longueur 20 devient 11111222223333344444. Comme cette duplication de caractère s'effectue toujours à la même fréquence, la forme générale d'un geste est conservée.

L'étirement de chaîne permet en outre de normaliser la valeur de similarité entre deux formes. En effet, une autre propriété de la distance de Levenshtein est que la plus

longue distance possible entre deux chaînes est la longueur de la plus longue des deux chaînes :

$$\max(Levenstein(\sigma_1, \sigma_2)) = \max(\text{length}(\sigma_1), \text{length}(\sigma_2))$$

Cette propriété permet d'estimer la ressemblance entre deux chaînes de même longueur l comme le rapport $1 - \frac{Levenstein(\sigma_1, \sigma_2)}{l}$. C'est cette valeur qui est utilisée pour déterminer le geste d'entraînement le plus proche du geste à classifier.

Entrée :
- σ_1 : la chaîne a étirer
- l' : la longueur de la chaîne à obtenir

Sortie :
- σ_2 : la chaîne étirée

Pré-condition :
- $\text{length}(\sigma_1) \leq l'$

Post-condition :
- $\text{length}(\sigma_2) = l'$

function stretch(σ_1, l') :
  $l = \text{length}(\sigma_1)$
  IF $l = l'$: **return** σ_1
  $step = l/(l' - l)$
  $\sigma_2 =$“”
  FOR $i = 0$ TO l STEP $step$:
    IF $i + step > l$: $\sigma_2 = \sigma_2 + \text{substring}(\sigma_1, i)$
    ELSE :
      $c = \text{charAt}(\sigma_1, i + step - 1)$
      $\sigma_2 = \sigma_2 + \text{substring}(\sigma_1, i, i + step) + c$
  return σ_2

TABLE 4.1 – Algorithme d'étirement d'une chaîne

Rotation

Une rotation à 90, 180 et 270 degrés est effectuée sur le geste à classifier et l'algorithme de Levenshtein est appelé à chaque rotation pour tous les gestes d'entraînement.

Cette opération est effectuée directement sur la chaîne de caractère correspondant à un geste. Elle permet de diminuer le nombre de gestes d'entraînement nécessaires pour atteindre une précision adéquate, et évite d'avoir à refaire chaque geste dans toutes les directions pour qu'il soit effectivement reconnu.

4.4 La combinaison de formes

Dans cette section, nous présenterons le fonctionnement système dont la tâche est d'implémenter le requirement 2 : "Le logiciel doit pouvoir combiner plusieurs formes simples en une forme plus complexe ou un widget, selon des règles pré-établies".

Une fois les formes reconnues, il faut encore définir un moyen de les traduire en widgets. Ce problème est complexe, car une forme peut représenter différents types de widgets. Par exemple, un rectangle représente-t-il un bouton, un champ de texte, un conteneur, ... ? De plus, certaines formes (en particulier : le triangle, le rectangle, la croix) peuvent être vues comme une combinaison de lignes. Permettre ce type de combinaison est une fonctionnalité intéressante à implémenter.

Pour résoudre cette ambiguïté, nous utilisons le concept de grammaires contextuelles.

Nous commencerons par définir les grammaires contextuelles. Ensuite, nous présenterons plusieurs types d'implémentation du moteur de grammaire contextuelles. Nous en choisirons une, et nous discuterons de ses performances. Nous discuterons ensuite des résultats obtenus et des extension possible du moteur.

4.4.1 Grammaires contextuelles

Les grammaires contextuelles seules ne permettent pas de reconnaître une forme, mais elle peut être intégrée dans n'importe quel technique de reconnaissance. Elle permet de définir des formes comme une composition de formes plus simples. Ces formes peuvent être pré-dessinées et ajoutées à la souris, ou dessinées et reconnues par une technique de reconnaissance de forme.

Chaque forme complexe est accompagnée d'une grammaire, elle-même composée de formes et de contraintes (la plus typique étant "<forme1> dans <forme2>". Si toutes les contraintes de la grammaire sont respectées, la forme complexe est reconnue.

Avant de poursuivre, définissons une grammaire représentant un widget* typique, mais néanmoins complexe : le *listBox* (liste de sélection). Ce widget est généralement représenté par un rectangle contenant un triangle en haut à droite, un autre en bas à droite, et plusieurs lignes à l'intérieur du rectangle (dans cet exemple, nous en prendrons trois). Cette description est traduite sous forme de grammaire contextuelle à la

Figure 4.11. Cette grammaire sera utilisée à titre d'illustration dans la discussion qui suit.

ListBox :=
- Forme : rectangle1
- Forme : ligne1
- Forme : ligne2
- Forme : ligne3
- Forme : triangle1
- Forme : triangle2

(a) Contrainte : ligne1 est horizontale
(b) Contrainte : ligne2 est horizontale
(c) Contrainte : ligne3 est horizontale
(d) Contrainte : ligne1 dans rectangle1
(e) Contrainte : ligne2 dans rectangle1
(f) Contrainte : ligne3 dans rectangle1
(g) Contrainte : triangle1 dans rectangle1 en haut à droite
(h) Contrainte : triangle2 dans rectangle1 en bas à droite

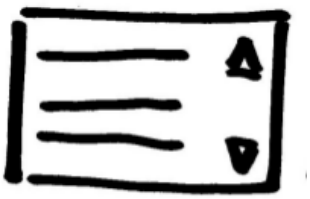


FIGURE 4.11 – Exemple de grammaire contextuelle pour un *ListBox*

4.4.2 Types d'implémentation

Deux types d'implémentations ont été envisagées :

1. Une méthode dépendante de l'ordre de dessin : le moteur enregistre l'ordre dans lequel les formes ont été dessinées. A chaque nouvelle forme, les grammaires sont testées uniquement sur les dernières formes dessinées.
2. Une méthode indépendante de l'ordre de dessin : à chaque nouvelle forme, les grammaires sont testées sur toutes les combinaisons de formes possibles.

Le premier type est très rapide, mais manque d'efficacité : il arrive souvent qu'un dessinateur recycle une forme ou y ajoute une nouvelle, de sorte qu'une autre grammaire soit valide. Cependant, si les autres formes ont été dessinées préalablement, le système

ne teste pas la grammaire sur ce groupe de formes, par conséquent la grammaire n'est pas appliquée, donc le widget* n'est pas identifié.

image représentant le problème du premier type d'implé

Le second type n'a pas ce problème. Par contre, c'est un problème NP-difficile : il faut créer toutes les combinaisons possibles et les tester une à une.

Une implémentation naïve coûte très cher en temps de calcul : le nombre de combinaisons à tester pour une grammaire composée seulement de formes de même types est $\frac{n!}{k!(n-k)!}$, où n est le nombre de formes identiques d'un certain type dans une fenêtre, et k étant le nombre de formes de ce type spécifiées dans la grammaire.
Si plusieurs types sont présents dans une grammaire, le nombre de combinaison est le produit des combinaisons pour chaque type.

Par exemple, la grammaire du *listBox* définie précédemment est composée d'1 rectangle, 2 triangles et 3 lignes. Sur une fenêtre composée de 15 rectangles, 10 triangles et 25 lignes (voir Figure 4.12), le nombre de cas à tester avec une implémentation naïve est

$$\frac{15!}{1!(15-1)!} \cdot \frac{10!}{2!(10-2)!} \cdot \frac{25!}{3!(25-3)!} = 15.45.2300 = 1\ 552\ 500$$

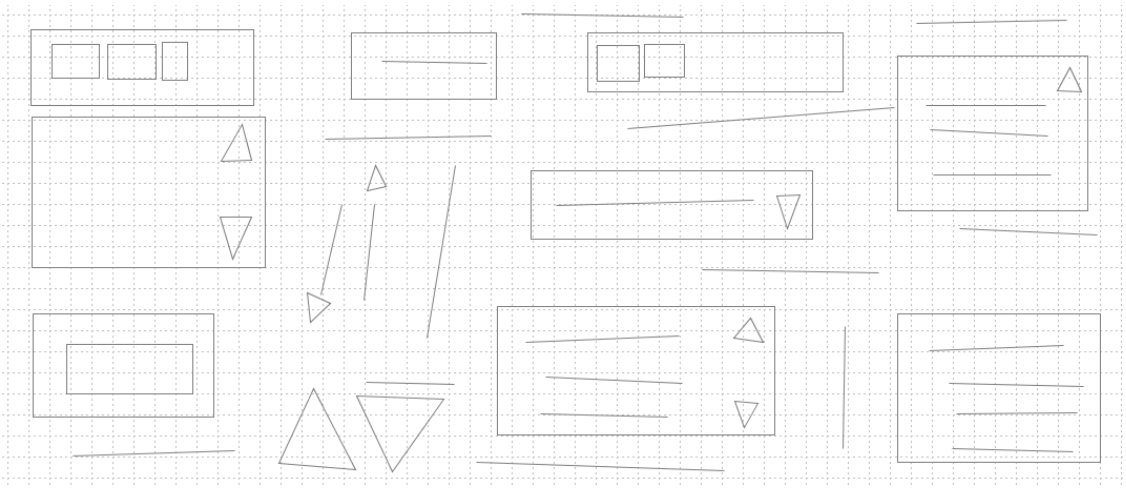


FIGURE 4.12 – Fenêtre utilisée pour l'exemple du *ListBox*

Ce nombre de cas, bien que grand, reste suffisamment petit pour être testés en un temps raisonnablement court avec un ordinateur récent. Rappelons aussi qu'il s'agit d'une implémentation naïve. La section suivante présentera certaines optimisations permettant de réduire drastiquement le nombre de combinaisons effectivement testées.

4.4.3 Notre implémentation

Comme nous venons de le voir, le deuxième type d'implémentation est coûteux, mais réalisable dans ce contexte. Pour implémenter cette technique, nous utilisons un moteur de programmation par contraintes.

Wikipédia (2011b) définit la programmation par contrainte comme suit :
La programmation par contraintes est un paradigme de programmation [...] permettant de résoudre des problèmes combinatoires de grandes tailles tels que les problèmes de planification et d'ordonnancement. Elle sépare la partie modélisation à l'aide de problème de satisfaction de contraintes (ou CSP pour Constraint Satisfaction Problem*), de la partie résolution dont la particularité réside dans l'utilisation active des contraintes du problème pour réduire la taille de l'espace des solutions à parcourir (on parle de propagation de contraintes).*

Chaque grammaire est définie comme suit : chaque forme utilisée est une variable dont le domaine initial est l'ensemble des formes contenues dans la fenêtre en cours d'analyse. Ces domaines sont réduits au fur et à mesure jusqu'à ce qu'ils ne contiennent plus qu'un élément ; la variable est alors considérée comme liée. Une fois que toutes les variables (*ie* toutes les formes impliquées dans la grammaire) sont liées, la grammaire est testée. Si elle est valide, la solution est conservée dans une liste de solutions acceptables. Le moteur continue ensuite sa recherche en liant les variables autrement, jusqu'à ce que l'ensemble des solutions possibles ai été testé.
L'avantage d'utiliser un moteur de programmation par contraintes est sa capacité à élaguer l'arbre de recherche par la propagation de contraintes : chaque contrainte est traitée indépendamment, et si certaines formes ne respectent pas une contrainte, elles peuvent être retirées.

Exemple de propagation

Pour illustrer la propagation, reprenons l'exemple du *listBox* et la Figure 4.12 : 1 552 500 combinaisons sont possibles avec une implémentation naïve (*ie* on teste toutes les combinaisons une à une). Avec un moteur de programmation par contraintes, un grand nombre des formes sont retirées des domaines dès le départ.

– Les contraintes (a),(b) et (c) retirent 4 lignes (non horizontales) du domaine de ligne1, ligne2 et ligne3 ; il reste 21 lignes dans ces domaines.

– Les contraintes (d),(e) et (f) retirent 8 lignes (horizontales qui ne sont pas dans un rectangle) du domaine de ligne1, ligne2 et ligne3 ; il reste 13 lignes dans ces domaines.
– La contrainte (g) retire 7 triangles (qui ne sont pas dans un rectangle en haut à droite) du domaine de triangle1, ainsi que 12 rectangles (qui ne contiennent pas de triangle en haut à droite) du domaine de rectangle1 ; il reste 3 rectangles dans le domaine de rectangle1, et 3 triangles dans le domaine de triangle1.
– Enfin, la contrainte (h) retire 7 triangles (qui ne sont pas dans un rectangle en bas à droite) du domaine de triangle2, ainsi qu'1 rectangle (qui ne contient pas de triangle en bas à droite) du domaine de rectangle1 ; il reste 2 rectangles dans le domaine de rectangle1, et 3 triangles dans le domaine de triangle2.

A l'issue de cet élagage préalable, les tailles des domaines sont réduites à

– **13** lignes pour ligne1,ligne2 et ligne3
– **3** triangles pour triangle1 et triangle2
– **2** rectangles pour rectangle1

Il reste donc $13^3.3.3.2 = 39546$ combinaisons à tester au plus, c'est-à-dire moins de 3% du nombre de combinaisons qui auraient été testées avec une approche naïve. C'est une réduction plus que considérable ! En outre, cet élagage se poursuit durant toute la recherche, aussi ce nombre continuera de décroître tout au long de celle-ci.

Contraintes floues

Certaines contraintes sont parfaitement booléennes (*ie* valide ou non valide), d'autres non. Nous les appellerons contraintes floues. Par exemple, la contrainte "<ligne> est verticale" est floue, car le système ne peut pas exiger du dessinateur de dessiner une ligne parfaitement verticale. Par conséquent, la contrainte permet qu'une droite soit "presque verticale".

Le test d'une contrainte floue retourne un nombre compris entre 0 et 1, où 1 signifie "correspondance parfaite" et 0 "aucune correspondance". Pour permettre la validation de ces contraintes, une valeur de seuil est définie pour chacune. La contrainte est considérée comme valide si la valeur retournée est supérieure à la valeur de seuil.

Grammaires mutuellement exclusives

Il peut arriver que le test d'une grammaire soit valide sur un groupe de formes, mais qu'il existe une autre combinaison, de meilleure qualité (*ie.* dont la valeur retournée lors du test est plus proche de 1). La valeur de ces contraintes floues est utilisée comme $tie - breaker$ si une grammaire est correcte pour plusieurs combinaisons mutuellement exclusives. La figure Figure 4.13 illustre cette fonctionnalité.

(a) (b) (c)

FIGURE 4.13 – Illustration du $tie - break$ entre deux combinaisons mutuellement exclusives.
(a) Huit lignes faites à la main pour représenter deux rectangles
(b) Combinaison retenue, mais très floue
(c) Combinaison possible, de meilleure qualité ; c'est celle-ci qui sera choisie

Les contraintes linéaires permettent de valider partiellement le requirement 4 : "Si deux combinaisons mutuellement exclusives sont possibles sur base d'une liste de formes, la combinaison la plus contrainte doit être choisie". Dans le cas de deux grammaires différentes mutuellement exclusives, c'est la grammaire contenant le plus de formes qui est choisie. Si le nombre de formes est le même, c'est le nombre de contraintes qui joue le rôle de $tie - breaker$.

4.4.4 Discussion, Extensions possibles

finir discussion

L'implémentation choisie permet de modifier d'anciens widgets sans avoir à les redessiner intégralement. Il reste cependant quelques problèmes avec cette implémentation :

– La fenêtre doit être intégralement reconstruite si

4.5 Évaluation des requirements

usiSketch est un logiciel en cours de développement. Aussi, certains requirements ne sont pas encore complètement remplis. Ce chapitre regroupe l'ensemble des requirements et leur état d'avancement. Pour chaque catégorie telle que défini à la section 4.1, nous évaluerons l'état d'avancement du développement. Pour chacun des requirements, un pourcentage d'état d'avancement sera donné.

Les requirements non-fonctionnels ne sont pas repris dans cette évaluation, car leur évaluation est subjective.

Nous ne traiterons pas des requirements de type Architecture, car ce type ne comprend que des requirements non-fonctionnels.

4.5.1 Reconnaissance

Dans l'ensemble, la reconnaissance et la composition de formes sont implémentées. Étant donné qu'il s'agit du coeur même d'usiSketch, nous avons passé l'essentiel du temps de développement sur cette fonctionnalité, qui a été totalement réimplémentée par rapport à SketchiXML.

Étant donné qu'il s'agit d'une fonctionnalité basée sur l'intelligence artificielle, il n'existe pas de solution déterministe parfaite à ce type de requirement*. Ces fonctionnalités sont donc perfectibles. Néanmoins, les résultats actuellement obtenus sont très satisfaisants.

La fonctionnalité majeure manquant encore actuellement est la possibilité d'importer une image et de la convertir en fichier usiSketch. L'algorithme de reconnaissance de forme le permettant doit encore être implémenté. De plus, les règles de composition d'un widget ne sont pas encore éditables via une interface. Cependant, étant définies dans un fichier XML, un utilisateur suffisament expérimenté peut déjà créer ou modifier ces règles.

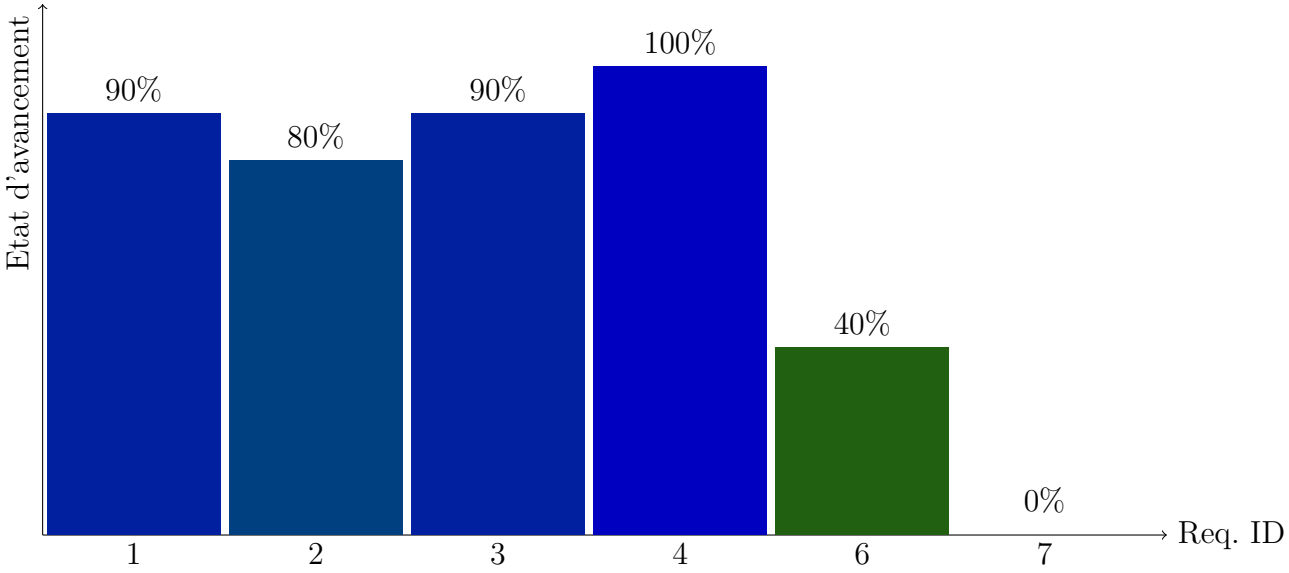


FIGURE 4.14 – États d'avancements des requirements de type Reconnaissance

4.5.2 Dessin-Rendu

Toutes les fonctionnalités prévues lors de l'élaboration des requirements* sont implémentées, à l'exception de la sélection et du déplacement de formes préalablement dessinées. Cependant, la fonctionnalité de sélection est en cours de développement.

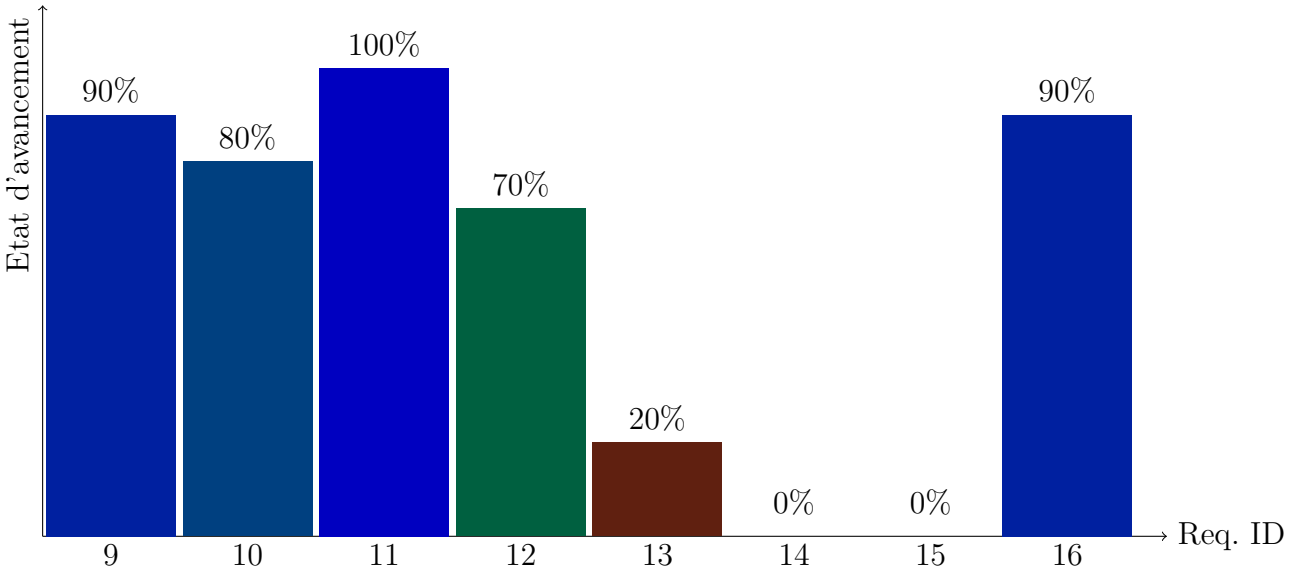


FIGURE 4.15 – États d'avancements des requirements de type Dessin-Rendu

4.5.3 Navigation

La navigation est fonctionnelle mais reste encore très perfectible. En effet, les seules actions possibles se font sur les fenêtres lors d'un clic sur un widget cliquable (*ie* un bouton). De plus, le seul événements reconnu est le clic sur un élément.
Remarquons cependant que le niveau de fidélité* attendu d'usiSketch est faible. Aussi, il est normal que ces fonctionnalités soient implémentées plus tard durant le développement, voire pas du tout si les utilisateurs finaux n'y trouvent pas un intérêt.

Une fonctionnalité importante selon les utilisateurs manquant encore actuellement est le requirement 20 : "Le système doit être capable de structurer les éléments graphiques sous la forme d'un arbre". Cependant, comme nous le verrons à la section 4.6, une solution intéressante et facile à implémenter a été proposée par un testeur du logiciel.

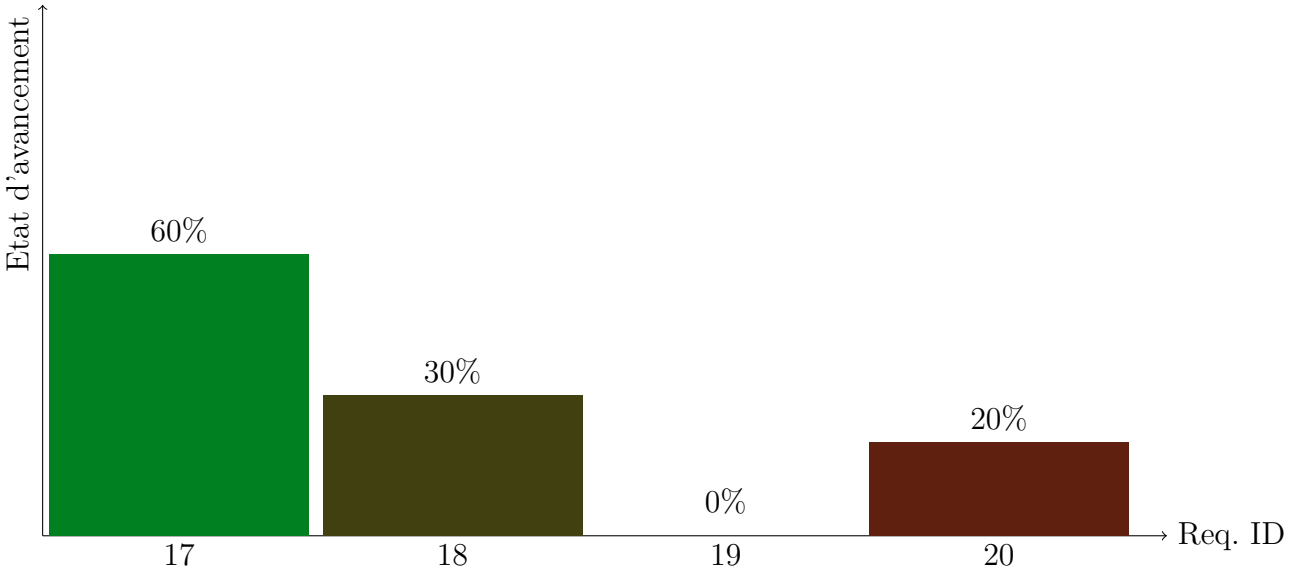


FIGURE 4.16 – États d'avancements des requirements de type Navigation

4.5.4 Données

Un projet usiSketch peut être sauvé sous la forme d'un fichier usk.
L'exportation d'un projet usiSketch et fichier usiXML est elle aussi implémentée, à l'exception des règles de navigation d'une page à l'autre qui ne sont pas encore exportées. Cependant, il fut difficile de tester cette fonctionnalité du fait du manque d'interpréteur

ou d'éditeur de code usiXML suffisamment abouti pour vérifier que les fichiers usiXML produits sont corrects.

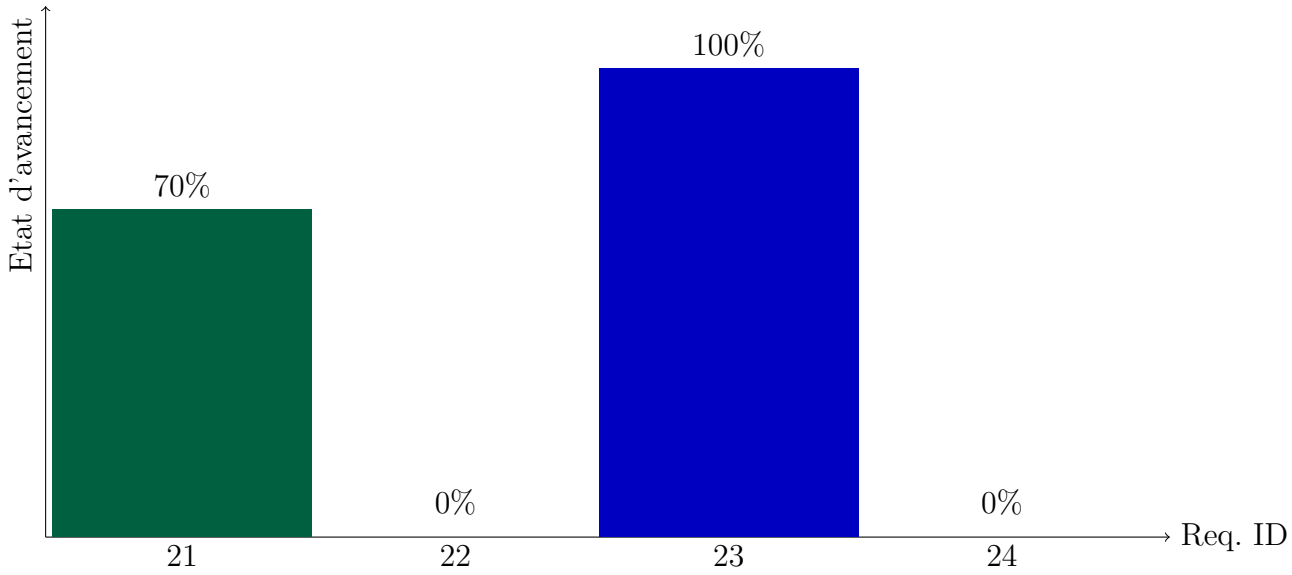


FIGURE 4.17 – États d'avancements des requirements de type Données

4.5.5 Simulation

Comme nous l'avons déjà mentionné dans la section 4.2, nous ne nous sommes peu attardés sur cette section, car elle requiers la présence du logiciel de test pour la plupart de ces requirements*.

Le seul requirement* implémenté à ce jour est le requirement 25 : "Une interface définie dans usiSketch doit pouvoir y être simulée sommairement". Il reste cependant perfectible : actuellement, le seul comportement dynamique proposé est le clic sur un bouton. Il n'est pas possible de changer le texte d'un champ, de cocher une case ou tout autre comportement dynamique. Bien que cela ne soit pas vital pour l'utilisation du logiciel, une amélioration de cette fonctionnalité serait appréciable.

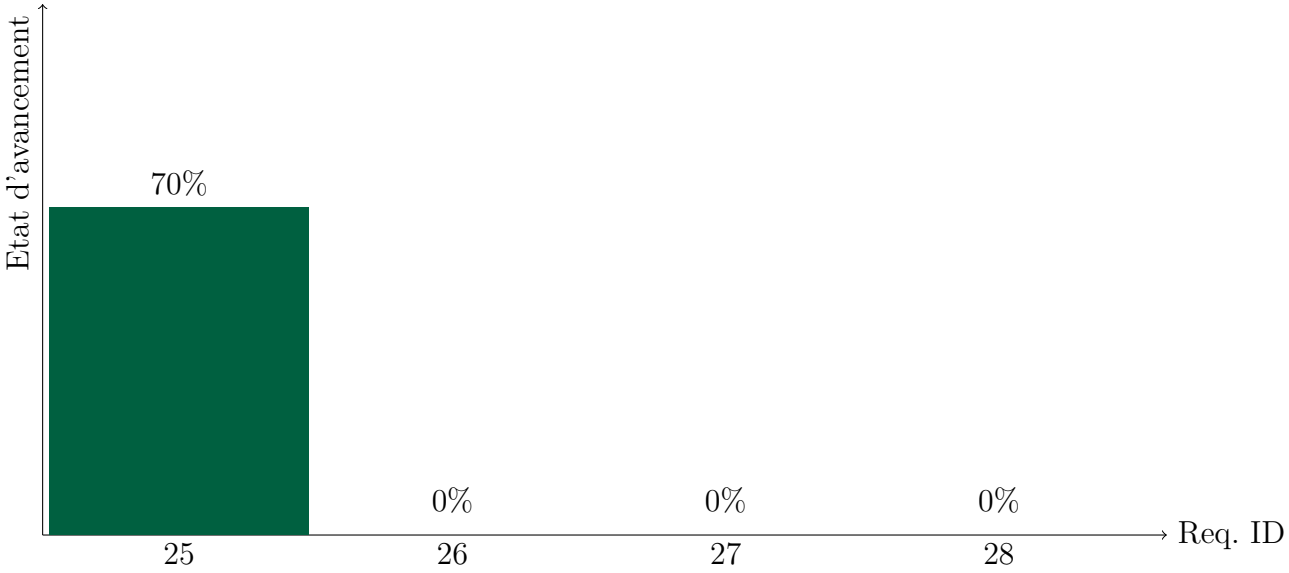


FIGURE 4.18 – États d'avancements des requirements de type Simulation

4.5.6 Ergonomie

Les requirements* fonctionnels de cette section sont tous liés à l'historique de conception. Cette fonctionnalité étant très importante, elle fut intégralement développée.

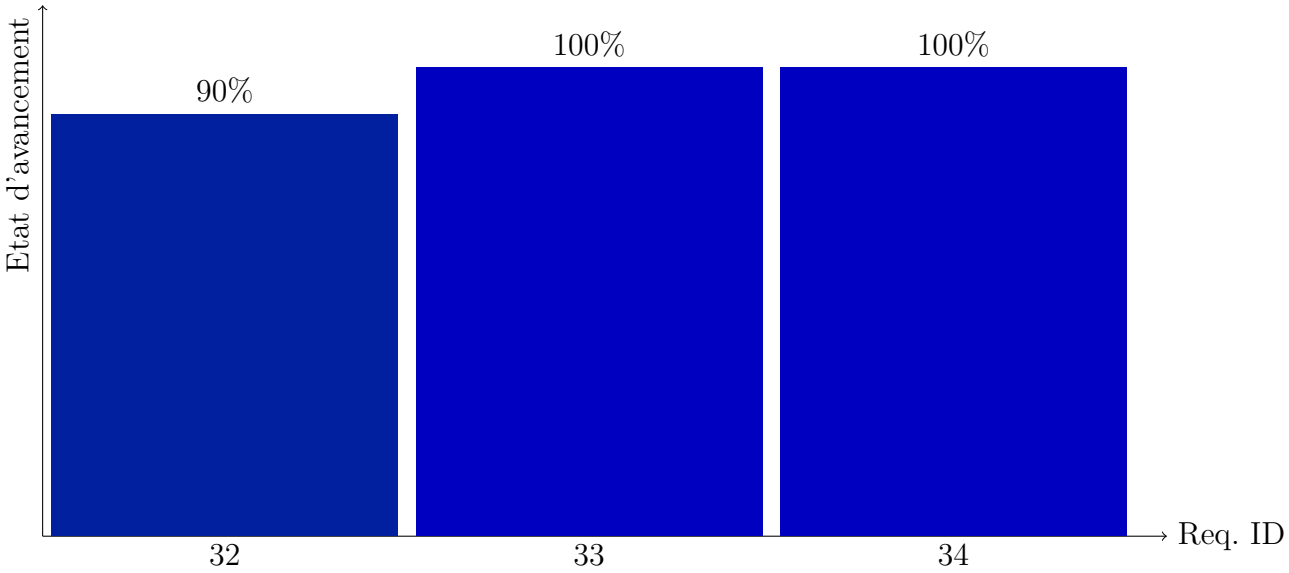


FIGURE 4.19 – États d'avancements des requirements de type Ergonomie

4.5.7 Charge de travail

Comme nous venons de le voir, une partie des requirements ne sont pas encore tous remplis. Cependant, le coeur du logiciel est prêt : les fonctionnalités manquantes ne demandent pas de recherche intensive.

mettre nombre de lignes de code

Le code source d'usiSketch comprends actuellement xxxx lignes de code. Nous estimons qu'au rythme de travail actuel, un développeur seul pourrait remplir tous les requirements* présentés dans ce mémoire en une année civile. Dans cette estimation n'est pas repris le logiciel de test, qui est un logiciel à part entière. Ce dernier devrait être construit sur base d'usiXML plutôt qu'usiSketch.

Utiliser CO-COMO pour plus de précision

4.6 Etude de cas

Cette section illustre le fonctionnement d'usiSketch via l'élaboration d'interfaces typiques. Nous évaluerons ensuite l'expérience utilisateur. Pour se faire, nous avons demandé à plusieurs volontaires, issus de différents milieux, d'élaborer les prototypes d'interfaces et de nous donner leur retour d'expérience concernant le logiciel.

Le logiciel étant encore au stade de développement, nous avons adopté une approche qualificative plutôt que quantificative quant aux retours d'expérience des volontaires. En effet, ceux-ci avaient souvent des remarques ou idées intéressantes à présenter. Ces idées pourront être utilisées durant la suite du développement d'usiSketch.

4.6.1 Définition des scénarios

Nous avons défini trois scénarios, croissant dans leur complexité. Nous avons volontairement laissé ces définitions floues, de sorte que les volontaires aient le plus de liberté possible quant à la façon dont l'interface doit se comporter, afin d'obtenir le retour d'expérience le moins influencé possible.

Le premier scénario est un formulaire de login typique. Il est composé de trois fenêtres : une fenêtre formulaire, une fenêtre de login réussi, et une fenêtre de login échoué. Dans le cas du login échoué, l'utilisateur doit pouvoir revenir sur la page de formulaire pour recommencer.

Le second scénario est inspiré d'un site de vente en ligne : il doit comprendre une page reprenant la liste des produits en ventes, au moins un des produits dans cette liste doit disposer d'un lien vers sa description, et un lien vers le panier doit être présent dans la page de liste et dans la page de description.

Le troisième scénario consiste en la création d'un prototype pouvant représenter l'outil WalkAware, situé à l'adresse :http ://www.walkaware.com/geo/Edit et représenté à la Figure 4.20

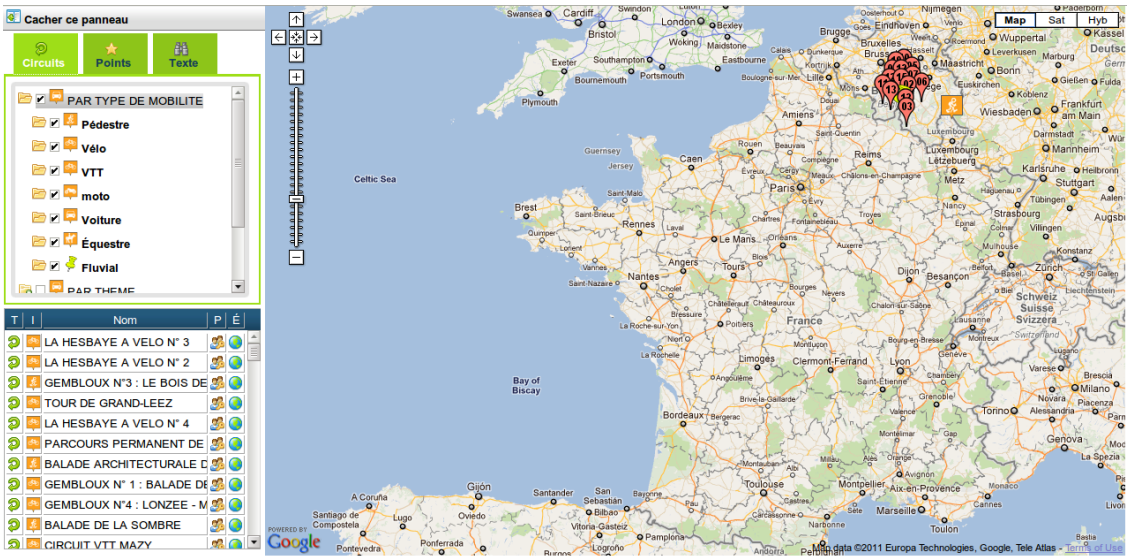


FIGURE 4.20 – Interface de WalkAware

4.6.2 Volontaires

Les volontaires ayant testé l'interface sont des personnes ayant toutes des compétences en design et ergonomie. Nous avons demandé à un testeur de logiciel, deux développeurs et un designer de tester l'interface d'usiSketch.

4.6.3 Analyse des résultats

La première étude de cas a donné de très bon résultats : les volontaires n'ont eu aucune difficultés à dessiner les pages et à les faire interagir. De même, la simulation s'est bien déroulée, et le résultat est assez fidèle par rapport à l'interface envisagée. Le temps d'exécution de la tâche (simulation mise à part) varie de 4 à 7 minutes.

La deuxième étude de cas fut plus long, mais n'a pas présenté de problème majeur. En effet, le temps d'exécution de la tâche varie de 5 à 12 minutes. Il faut cependant remarquer le nombre plus important de formes à dessiner durant cette étude de cas.
Les volontaires ont par contre eu plus de remarques à faire durant celle-ci. Nou en reparler

Les volontaires ont par contre eu beaucoup de mal à prototyper la troisième étude de cas. En effet, la présence d'un widget très complexe (google Maps) dans l'interface finale fut déroutante, et l'interface prototypée est très peu représentative Le temps

rédiger walka-ware

d'exécution de la tâche varie de minutes.

mettre temps

4.6.4 Retours d'expérience

Outre certains bugs identifiés par les volontaires, et corrigés depuis, nous avons demandé un retour d'expérience qualitatif à chaque volontaire. Nous avons préféré un retour d'expérience qualitatif à un retour d'expérience quantitatif, car le nombre de volontaires était trop faible pour obtenir un échantillon quantitatif suffisamment représentatif. De plus, notre échantillon était très biaisé (uniquement des personnes compétentes en ergonomie et en design), les résultats que nous aurions obtenus auraient été peu fiables.

Prise en main

Le premier contact avec le logiciel s'est très bien passé. Les volontaires n'ont eu aucun problème à dessiner et ont très vite compris le fonctionnement du système. C'est très encourageant, vis-à-vis du requirement 29 : "Le temps d'apprentissage de l'utilisation du logiciel doit être le plus rapide possible".

L'un des premier problèmes auquel tous les volontaires ont été confrontés est l'absence d'une liste des widgets existants et de leur grammaire. C'est un obstacle important pour l'intuitivité du logiciel et sa rapide prise en main. Néanmoins, ce problème sera partiellement réglé lorsque la fenêtre d'édition des grammaires sera disponible (requirement 6 : "Le dessinateur doit pouvoir spécifier les règles de composition à l'aide d'une interface simple et intuitive"). Cependant, selon certains, ce n'est pas suffisant. Il faudrait un affichage visuel direct de la fenêtre (sans possibilité de modification) en complément de la vue d'édition de grammaires. Cette liste serait accessible à tout moment (par exemple via un bouton) et afficherait tous les widgets existants et leur grammaire.
Cette fonctionnalité demande un effort d'implémentation relativement important : sur base d'une grammaire générique, il faut pouvoir construire une image représentative. C'est un problème assez complexe, qui demande sans doute un effort de recherche relativement important.

Critiques, Observations

Il fut intéressant de remarquer que certains volontaires utilisaient la fonctionnalité multi-traits, alors que d'autres ne l'utilisaient pas. Pire, cette fonctionnalité les handi-

capait : ils n'étaient pas en mesure de dessiner à la vitesse qu'ils souhaitaient.
La façon dont le multi-traits est gérée actuellement est assez grossière : après un certain temps sans trait (environ 300ms), le logiciel considère que le geste est terminé. Il serait probablement plus efficace d'utiliser une fenêtre de temps minimum et maximum : après chaque trait, la reconnaissance est effectuée sur ce dernier. Si un trait est dessiné avant un certain temps (moins d'une seconde), le reconnaissance tente une reconnaissance avec les deux traits. Si, et seulement si, cette seconde reconnaissance retourne une forme plus ressemblante, alors le geste est considéré comme multi-traits.
Une autre solution, plus simple, serait de donner la possibilité à l'utilisateur de choisir entre multi-traits et mono-trait. Cette solution, plus simple, est aussi plus radicale : certains volontaires alternaient entre mono-trait et multi-traits. Avec cette solution, cela n'est plus possible.

Il est intéressant de voir aussi que pour l'écriture de texte, certains volontaires ont préféré utiliser le mode annotation plutôt que de dessiner des lignes représentant des labels, mais seulement pour du texte statique. De même, ils ont aussi ajouté le texte des boutons via l'annotation.
Comme les volontaires n'ont pas eu à exporter le prototype en usiXML, ils n'ont pas remarqué que les annotations ne sont pas exportées. Cela nous pousse donc à réévaluer l'importance d'un système de reconnaissance de texte. Une solution possible serait d'ajouter un autre mode de dessin, le mode texte, qui reconnaîtrait tout geste comme du texte et le convertirait en label avec le contenu souhaité. Ce système permettrait une exportation usiXML et semble plus intuitif pour l'utilisateur. Remarquons cependant que le widget* "label" servirait alors de champs de texte dynamique : ont sait qu'il faut en mettre un, mais on ne sait pas ce qu'il contiendra à l'avance. Par exemple, dans la seconde étude de cas : les champs dans le panier, ou la liste des produits en ventes.

Les volontaires ont en outre très peu utilisé la possibilité de changer de niveau de fidélité*. Ils se sont visiblement senti suffisamment à l'aise avec le niveau dit "Low". Certains nous ont répondu que les miniatures affichées sur la gauche de la zone de dessin étaient suffisantes pour se faire une idée du rendu général. S'ils avaient besoin d'un aperçu plus détaillé, ils utilisaient la prévisualisation directement.

La suppression de lien dans la vue de navigation n'est, pour certains, pas intuitive : devoir redéfinir le lien, pour obtenir le "delete" du menu des liens n'est pas pratique.

L'un d'entre eux a suggéré d'utiliser le même système que la gomme utilisée dans la vue de dessin.

L'absence de frontières sur la zone de dessin a aussi perturbé certains volontaires. Ces derniers préféreraient avoir une vision claire de ces dernières, pour avoir une idée plus précise du rendu final.

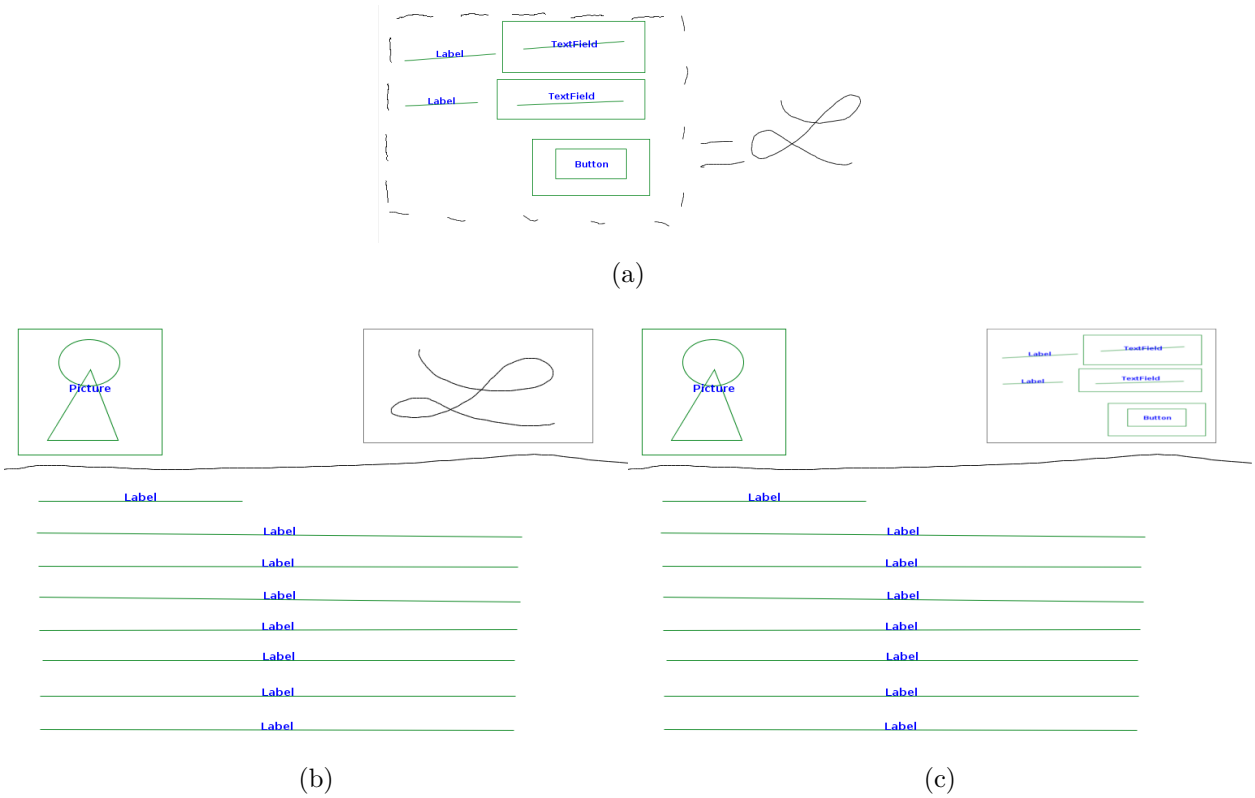
Suggestions de fonctionnalités additionnelles

Le designer volontaire nous a suggéré d'ajouter la possibilité de dupliquer le dernier widget dessiné, afin de le placer plusieurs fois plus rapidement. Cette fonctionnalité, cousine du copier/coller, fut suggérée durant la seconde étude de cas.

Le testeur volontaire nous a proposé une idée très intéressante concernant le requirement 20 : "Le système doit être capable de structurer les éléments graphiques sous la forme d'un arbre" : la possibilité de définir des "blocs". Chaque bloc aurait son contenu spécifique et serait identifié par un geste choisi par l'utilisateur. L'utilisateur pourra alors ajouter un bloc sur n'importe quelle fenêtre en dessinant le geste représentatif du bloc et en encadrant ce geste par un rectangle (pour spécifier les frontières du bloc).
Le scénario de création d'un bloc serait le suivant :

1. L'utilisateur informe le logiciel qu'il veut créer un bloc, sans doute par le clic sur un bouton
2. Le logiciel lui demande d'entrer le geste représentant ce bloc. Le geste est alors ajouté comme geste d'entraînement au module de reconnaissance de forme.
3. Une nouvelle zone de dessin est créée et l'utilisateur peut commencer à décrie le contenu du bloc.
4. Une fois le bloc défini, il lui suffit de choisir la/les fenêtre(s) devant le contenir et de dessiner le geste du bloc encadré d'un rectangle

Remarquons que ce scénario respecte le requirement 30 : "Un maximum d'opérations de prototypage doivent être accessible via le pointeur". De plus, il permet une plus grande productivité (puisqu'un bloc peut être utilisé plusieurs fois) et permet aussi de structurer les widgets* entre eux : un bloc pourra être perçu comme un élément de type "box" dans la syntaxe d'usiXML. Il faudra peut-être insérer un niveau de fidélité* intermédiaire entre "Low" et "Medium", où les formes représentatives d'un bloc seraient remplacées par leur contenu. La Figure 4.21 illustre l'utilisation d'un bloc.



(a)

(b)

(c)

FIGURE 4.21 – Illustration de l'utilisation de blocs

(a) Définition du bloc : le contenu entre pointillés est dessiné, la lettre est le symbole représentatif du bloc

(b) Une fenêtre simple. Le rectangle de droite est le bloc défini en (a)

(c) Résultat obtenu avec le niveau de fidélité* intermédiaire : le symbole du bloc est remplacé par son contenu

4.6.5 Discussion

Comme nous venons de le voir, le logiciel permet à ce jour de faire ce pour quoi il est conçu : du prototypage rapide et basse fidélité. Cependant, il reste encore du travail pour déployer tout son potentiel. Le retour d'expérience fourni par nos volontaires est une première base de travail pour étoffer les requirements* du logiciel, et améliorer le logiciel.

Chapitre 5

Questions marketing

L ?étude marketing est dé ?nie comme l ?identi ?cation systématique et objective, la collecte, l ?analyse, l ?interprétation et l ?utilisation d ?informations dans le but d ?améliorer les prises de décisions liées à la reconnaissance et la résolution de problèmes et d ?opportunité dans le domaine du marketing. Malhotra et al. (2007)

jouter page

L ?étude vise à développer des outils dans le but de déterminer la manière la plus efficace d ?optimiser les ventes et le positionnement lors de l ?élaboration de la stratégie de vente.section 5.10.

Pour ce faire, nous commencerons par déterminer le besoin de base et les possibles besoins périphériques. Par la suite, nous appliquerons les outils marketing nécessaires, nous émettrons les hypothèses de travail qui vont pouvoir nous guider vers l'établissement d ?un questionnaire répondant aux hypothèses à tester. Après une conclusion des résultats des tris à plat ainsi qu ?une analyse de tri croisés nous interpréterons ces informations pour déterminer notre panier final d ?attributs, notre stratégie ainsi que notre business model. .

Les sections sont les suivantes :

- L'analyse du besoin
- L'analyse PESTEL
- L'analyse SWOT
- L'analyse de Porter
- L'analyse du terrain
- L'élaboration des Hypothèses
- L'analyse des données
- Le choix du panier d'attribut
- Le choix du business model
- La stratégie

5.1 L'analyse du besoin

Dans cette section nous mettrons l ?accent sur le besoin de base auquel nous comptons répondre. S ?en suivront alors une énumération et une explication des différents besoins périphériques possibles à ajouter au besoin de base. L ?application des différents outils marketing nous permettra de déterminer ultérieurement le choix de notre panier d ?attribut.

5.1.1 Le besoin de base

Nous illustrerons le besoin de base par l ?utilisation du concept de besoin générique et dérivé.

Le Marketing Stratégique voit les besoins génériques comme des problèmes auxquels sont confrontés les clients potentiels qui recherchent des solutions à ces problèmes par l'acquisition de produits ou de services.(Lambin et de Moerloose, 2008, p.70)

La dé ?nition suivante permet de mieux comprendre la nuance entre besoin générique et besoin dérivé par l ?explication plus précise de ce dernier.

Le besoin dérivé est la réponse technologique particulière apportée au besoin générique et est aussi l'objet du désir. L'automobile par exemple, est un besoin dérivé par rapport au besoin générique de transport (Lambin et de Moerloose, 2008, p.70)

Dans notre cas, le besoin générique auquel répond le prototypage d ?User interface s ?inscrit dans le besoin de créer une maquette ou prototype lors de la création d ?une interface graphique. Suite au développement de la sphère informatique, le besoin dérivé s ?est avéré être la constitution d ?outils qui permettront de répondre au besoin de maquettage ou de prototypage lors de la création d ?un interface graphique. Le besoin de base de notre produit est donc le besoin de créer une maquette ou prototype lors de la création d ?une interface graphique.

Nous énumérerons et expliquerons chacun des attributs périphériques qui peuvent être ajoutés au besoin de base. Une fois toute l ?analyse marketing effectuée, le panier d ?attribut complet sera alors choisi.

- Attribut 1 : Utilisation du langage informatique de programmation usixml : L'utilisation de celui-ci entraîne une facilité d ?exportation du langage vers plusieurs plateformes d ?utilisation différentes. Il est alors possible de l ?employer sur n ?importe quel OS (Mac, windows, li- nux,..) et type de machine (PC, tablette graphique, téléphone portable (de type PDA)). L ?inconvénient de cet attribut est que ce langage reste neuf, et n ?a donc pas encore fait ses preuves en matière d ?outil informatique possible ; il n ?est donc pas ? à priori - ?able.
- Attribut 2 : L ?utilisation d ?une tablette graphique pour l ?utilisation du logiciel usiSKetch. Cette dernière entraine une facilité d ?utilisation et une amélioration du caractère pédagogique nécessaire pour transmettre l ?information.
- Attribut 3 : La vente d ?un service de consultance accompagnant notre outil usiSketch. Ce service peut s ?avérer utile pour les entreprises de développement existantes car en combinant notre produit et notre service de consultance, ces entreprises auront l ?occasion de sous-traiter la totalité de la partie de consultance et de consacrer leur temps à toutes les autres phases de développement d ?une interface graphique.
- Attribut 4 : L ?utilisation du programme usiSKetch : Celui-ci permet de dessiner instantanément la maquette ou le prototype de l ?interface graphique souhaité sur la plateforme utilisée (pc, tablette graphique, téléphone portable). Ce logiciel possède la particularité de pouvoir, une fois les différentes maquettes d ?interface dessinées, de les tester (« que se passe-t-il si je clique ici), et de les exporter ensuite dans le langage informatique choisi. .

check si on peu choisir le langage ou non

- Attribut 5 : La proposition d ?un cours d ?utilisation de notre logiciel. La réponse au besoin auquel nous répondons peut être accompagnée d ?un cours d ?utilisation dans le cas où nous ne nous occuperions pas de la consultance auprès du client ?nal. Avec ce service, nous offrons de former des personnes à mieux utiliser usiSketch, et à obtenir pour ce faire un comportement plus « pédagogique ». Comportement pédagogique dès lors, dans le but de rapidement cibler le besoin du consommateur et de lui faire comprendre l ?importance de chaque technique utilisée lors de la création du prototype d ?interface graphique voulu.
- Attribut 6 : L ?utilisation d ?un site web pour exposer notre produit en ligne. Cet attribut nous permet de faire de la publicité sur la manière de fonctionner de notre outil et des différents possibles de ce dernier.

Chaque attribut apporte certaines caractéristiques au produit final. Ces caractéristiques vont une fois déterminée, être testée auprès du marché des consommateur pour

estimer leur sensibilité sur l'importance qu'il accordent à ceux-ci, la performance attendue et observée si ils sont déjà client de service IT chez un fournisseur donné. Pour ce faire chaque attribut cité ci-dessus va être subdivisé en caractéristiques apportées au produit par l'utilisation de cet attribut.

- Attribut 1 : (+) améliorer la rapidité de prototypage, (+)l'utilisation sur plusieurs plateformes, (-)pas encore fiable car nouveau, (-) le client est dépendant de notre produit.
- Attribut 2 : (+) Améliorer la pédagogie, (+) améliorer la rapidité (+) démonstrabilité immédiate (+) améliorer l'intéractivité.
- Attribut 3 : (+) Baisse la charge de travail pour une entreprise de développement de logiciels, (-) pas de contact direct entre l'entreprise responsable du projet et celle qui consulte le client pour l'élaboration du prototype.
- Attribut 4 : (+) Améliorer la rapidité, (+)la pédagogie, (+) baisser le prix, (+) diminuer le nombre de rendez-vous nécessaire
- Attribut 5 : (+) améliorer la pédagogie, (+) faciliter le contact avec le client pour le consultant.
- Attribut 6 : (+) améliorer la notoriété du produit, (+) faire connaitre le produit.

Chaque attribut est à présent subdivisé en adjectifs caractérisant ces attribut. Plusieurs outils sont dés à présent utilisé pour détermine les caractéristiques extérieur et interne au projet qui peuvent également aider à déterminer les hypothèses à tester et à priori.

5.2 PESTEL

Nous avons choisi d ?élaborer cette analyse a ?n d ?optimiser notre stratégie de positionnement par rapport aux éléments macro-environnementaux entourant notre entreprise. L ?analyse Pestel permet de repérer les effets extérieur susceptible d ?in ?uencer ou non d ?une certaine manière le marché auquel nous voulons nous adresser. Les aspects qui peuvent in ?uencer le marché sont les aspect politiques, économiques, socioculturels, technologiques, écologiques et législatifs.

Facteurs politiques et légaux

Le développement d ?interfaces graphiques ne cesse d ?augmenter depuis 10 ans. Chaque entreprise, organisation ou même événement dispose de nos jours d ?un site internet, d ?un ERP pour améliorer l ?efficacité de son travail et de son organisation. Concernant les autorités, elles prennent doucement des mesures adéquates à ce domaine plutôt nouveau. Plus particulièrement dans le domaine du développement des interfaces graphiques, les facteurs politiques concernent surtout l ?établissement d ?un base légale d ?exploitation de ses logiciels. Par bases légales, nous entendons des lois, licences, brevets, entourant l ?utilisation de notre logiciel usiSketch et les outils nécessaires au bon fonctionnement de ce dernier. Ces bases légales seront plus amplement discutées dans la partie juridique de ce business plan.

Les gouvernements, et plus localement les régions de notre pays prennent ce genre d ?initiatives. Plus particulièrement, des efforts en matière de directives et de standardisation sont mis en ½uvre pour encourager les développeurs à générer des outils similaires et facilement utilisables par monsieur tout le monde. C ?est de la sorte qu ?un label Anysurfer, d ?abord lancé par des particuliers et par la suite soutenu par le gouvernement wallon, eut pour but de stimuler la création de "site accessible si il est utilisable et consultable dans son intégralité par tous ses visiteurs" [1]

Ce brevet né en 2000 et soutenu par la région wallonne dès 2006 a pour objectif de rendre accessible à tous l ?utilisation des sites internet. Plus récent et au niveau international, nous pouvons aussi évoquer que les grandes communautés internationales se soucient également de ce la réglementation de l ?outil internet. Récemment le rassemblement de l ?e-G8 à permis au principaux acteurs mondiaux de la sphère informatique

1. . (source : http ://www.anysurfer.be/fr/apropos-de-anysurfer/un-site-accessible/index.html)

(google, facebook, etc..) de se rassembler avec des dirigeants du G8 pour discuter la manière de réglementer au mieux la gestion de cet outil et les conséquences gouvernementales qui pourraient en découler. En conclusion, nous pouvons admettre que notre domaine d ?activité est loin d ?être sans régulation et directives à respecter. Nous nous devons bien au contraire de suivre attentivement la manière dont se développe les règles/lois pour mieux répondre à notre besoin sur la base légale.

Facteurs économiques

Nous pensons à l'heure actuelle qu'il n'y a pas vraiment de facteurs économiques externe à notre organisation qui pourraient nous influencer d'une mauvaise manière. Le seul facteur qui peut nous influencer est l'existence de la crise et ses effets sur les états membres. Tant que nos clients et leurs investissements ne seront pas affectés par ces événements notre marché ne sera pas en danger. De plus la possession d'un site internet ou d'un ERP devenant de plus en plus courant et nécessaire pour tout type d'organisation nous fait aller encore plus dans cette direction. Sur le graphique suivant nous voyons le pourcentage d'entreprises disposant d'un site internet dans les pays de l'union Européenne en 2008. Sachant que cette proportion est en constante augmentation ceci nous amène à dire que les facteurs économiques influencent positivement nos services proposés.

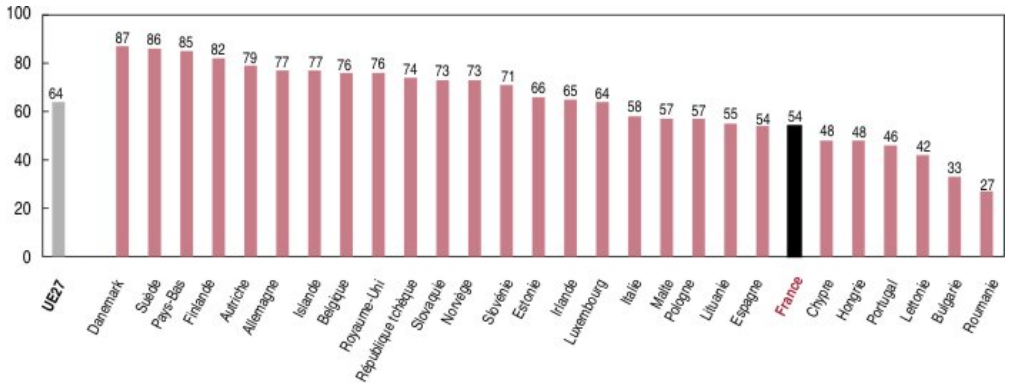


FIGURE 5.1 – Pourcentage d'entreprises disposant d'un site Internet par pays en 2008 (source : www.insee.fr/fr/themes/document.asp ?ref_id=ip1227&page=graph#graphique1)

Facteurs technologiques

Vu que notre projet d'entreprise est basé sur un nouvel objet de précision technique, les facteurs technologiques influenceront énormément la capacité à réussir de notre produit. En effet les clés de notre réussite sont l'utilisation d'une part du logiciel usiSketch que nous avons développé et d'autre part l'utilisation du langage Usixml qui permet de nous différencier de nos concurrents en utilisant une tablette graphique pour faciliter la conception durant les premières étapes de développement.

Le premier facteur technologique influençant positivement est la certification du langage usixml au près du W3C. Ceci entraine une plus grand notoriété et une plus grande visibilité internationale de ce langage et de notre entreprise.

Le second facteur technologique est le fait que notre logiciel usiSketch soit utilisable sur n'importe quel OS (windows, macosx, linux,...) il peut être utilisé sur d'autres tablettes graphique (example : l'iPad) que celle que nous utilisons au sein de notre entreprise.

Passons maintenant aux facteurs technologiques influençant négativement l'existence de notre organisation. Le principal et unique facteur est que nous nous trouvons dans un secteur de pointe et que celui-ci évolue très rapidement. La menace d'autres entrant reprise dans l'analyse swot est donc le facteur important qui influencera notre prise de décision lors du choix de la stratégie.

Facteurs socio-culturels

Comme principal facteur socio-culturel nous pouvons noter que la société d'aujourd'hui se développe de plus en plus au tour de la sphère internet. Pour soutenir cette clause nous savons que le nombre de connexion internet a augmenté de 2009 à 2010 de 3,5% respectivement pour les entreprise et de 6,5% pour les particuliers affichant un total de 3.286.201 connections internet sur tout le royaume [2]. De ce fait nous pouvons conclure qu'internet est un acteur plus qu'important au yeux de la société de nos jours et que c'est un facteur positif pour l'accomplissement de notre projet

2. Source :Statbel :http ://statbel.fgov.be/fr/statistiques/chiffres/travailvie/temps/connexion_internet/

5.3 L'analyse DOCoCliDiF

L'analyse dococlidif à pour but de déterminer plus précisément les Opportunités/-menaces liées au marché. Cette analyse est divisée en 6 partie différentes.

Demande

Le demande de prototypage de UI est parallèle à celle de la demande de UI ou de logiciel. Aujourd'hui celle-ci ne fait qu'augmenter. Le graphique suivant reprend l'évolution de celui-ci pour la période de 2006 à 2009.

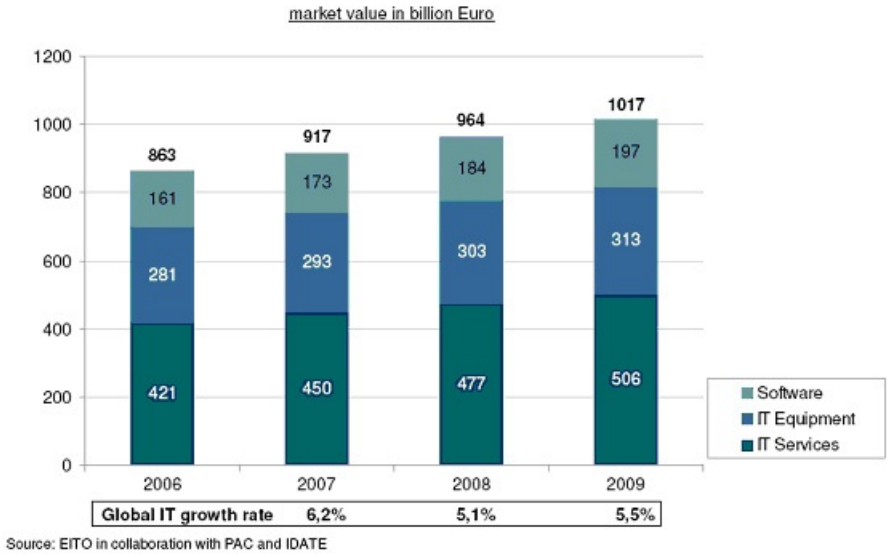


FIGURE 5.2 – Marché informatique mondial 2006/2009 par sous-segments (source :itrnews.com (2008))

5.3.1 Offre et Concurrence

Pour ce qui est de cette section nus invitons le lecteur à reparcourir la section chapitre 3 expliquant toute l'offre disponible à ce jour pour répondre au besoin de prototypage.

5.3.2 Client

Nous distinguerons deux client type que nous estimons pouvoir visr dans l'etablissement de cette société. Pr

5.4 Analyse SWOT

L'analyse SWOT à pour but de déterminer quatre caractéristiques en lien avec le produit et d'établir en conclusion, une stratégie déduite ou construite pour faire face et déterminer la manière de prendre en compte toutes ces données : L'analyse SWOT détermine les points forts du produit, les faiblesses du produits, les opportunités liés au produit, les menaces. L'analyse SWOT est un outil permettant de mieux se rendre compte de tous les aspects pouvant influencer le produit lors de sa mise en vente.

5.4.1 Strengths

Premièrement : le gain de temps
Nous comptons vendre un outil permettant de faciliter le prototypage lors de la création d'une interface graphique. Nos clients seront donc des entreprises ayant déjà leurs clients et faisant appel à nous pour faciliter cette section qui est le prototypage. Nos avantages sont donc exprimés en fonction des autres outils nommés dans la partie chapitre 3.

Il y a gain de temps pour le consommateur final car l'utilisation de la palette graphique combinée à un service pédagogique permet de réduire le nombre de rendez-vous nécessaires afin de satisfaire la demande du client. Mettre le point sur l'aspect pédagogique du consultant permet de faire comprendre au consommateur final toutes les subtilités liées à la création de chaque interface graphique. Ceci réduit également le temps nécessaire car tous les points sont repris et expliqués en langage courant pour ne pas se tromper sur le projet.

Deuxièmement : facilité de modification et maintenance du code
Lors de l'ajout d'une nouvelle fonctionnalité, il n'est plus nécessaire de modifier chaque interface séparément, en lien avec le logiciel. Il suffit de modifier le code usiXML. Le format usiXML fonctionne comme un tronc commun. Comme application concrète de cet avantage, prenons comme exemple une entreprise développant un logiciel sous iPhone. Si cette entreprise veut étendre son marché aux téléphones Android, et pour peu que l'interface iPhone aie été développée sous usiXML, l'entreprise peut réutiliser cette interface sans aucune modification. Il ne faut donc pas, comme cela se fait de nos jours,

recréer un tout nouveau code de logiciel. Cela signifie donc, exprimé en avantages concurrentiel, une facilité d'adaptation à des codes existants et un gain de temps énorme lors de la modification d'une interface existante.

Troisièmement : le temps d'apprentissage bas à l'utilisation du logiciel
la possibilité d'assigner la création d'interfaces graphiques à des personnes sans compétences informatique. Par exemple, cette personne pourra exprimer ses idées de logiciel sous forme de dessin (dessin, boutons) et voir le résultats immédiatement après l'avoir dessiné.

5.4.2 Weaknesses

Premièrement : Utilisation d'un langage informatique nouveau et donc peu courant. L'inconvénient réside donc dans le caractère non commun de ce format. Les informaticiens dans les entreprises ne sont pas familiarisés avec ce dernier et peuvent influencer la direction d'acheter ou non les services chez un autre fournisseurs IT. En effet, lorsqu'un nouveau système informatique est utilisé dans une organisation, et qu'un service informatique doit s'occuper de la maintenance en interne de ces outils, il est plus facile d'utiliser un langage connu.

Deuxièmement : Le prix d'une tablette graphique.
Le prix de la tablette va bien entendu influencer le prix facturé au client lors des différents services effectués. Ceci entrainera donc, qu'en utilisant notre produit il coutera plus cher au final à son client à lui.

Troisièmement : Logiciel Usisketch est inachevé.
Le logiciel usiSketch est toujours en développement. Bien que l'outil soit déjà fonctionnel, sa finalisation est prévue dans environ un an si un seul informaticien se charge de son développement à plein temps.

5.4.3 Opportunities

Premièrement : Le marché mondial des services IT, de la ventes hardware et du marché en matière d'interfaces graphiques ne cesse d'augmenter et d'affluer. voir Figure 5.3

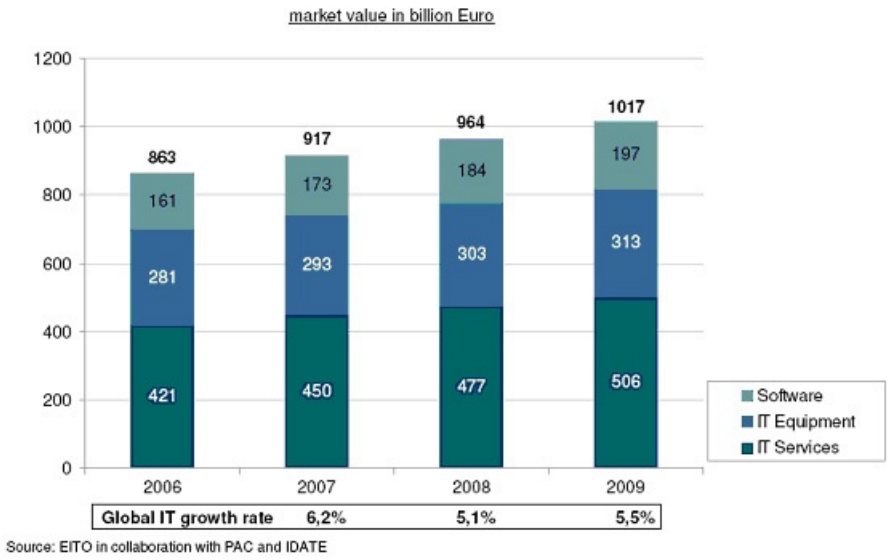


FIGURE 5.3 – Marché informatique mondial 2006/2009 par sous-segments (source :itrnews.com (2008))

La demande en Software et en services IT ne cesse d'augmenter sur le marché mondial. Ces deux indicateurs nous permettent de dire que le marché IT est en constante augmentation et qu'on peut le considérer comme une opportunité globale pour notre secteur.

Deuxiemement : Le marché de prototypage d'interface est varié.
Le fait qu'usiXML soit un format ouvert entraîne que n'importe quelle autre utilisateur de ce dernier pourra comprendre ce qui à été écrit sans avoir le logiciel qui l'a écrit. La variété du marché de prototypage est donc une opportunité pour nous car notre langage est compatible avec tous les outils de prototypage présents sur le marché. D'un autre point de vue, plusieurs outils ayant des utilités et spécialités différentes, peuvent être utilisés afin de créer une même interface. Par exemple, le concepteur peut utiliser usiSketch pour programmer les bases de son interface, puis modifier le code usiXML via un autre éditeur ayant une plus grande précision. Cela entraîne que si un de nos clients utilise ce format et utilise un outil différent du nôtre, il sera capable de lire et travailler sur le code qui à été écrit par notre outil.

Troisièmement :Le langage usiXML à été certifié par le W3C
Comme cité précédemment dans ce travail le W3C régule tous les standards de la sphère

informatique. Le fait que notre langage soit standardisé par le W3C entraine une reconnaissance internationale. Le langage reste neuf auprès du consommateur mais à fait ses preuves qu'il fonctionne.

Quatrièmement : L'arrivée massive des tablettes graphiques et la diminution de leurs prix sur le marché. Sur la Figure 5.4, nous pouvons observer les prévisions de ventes pour les années à venir.

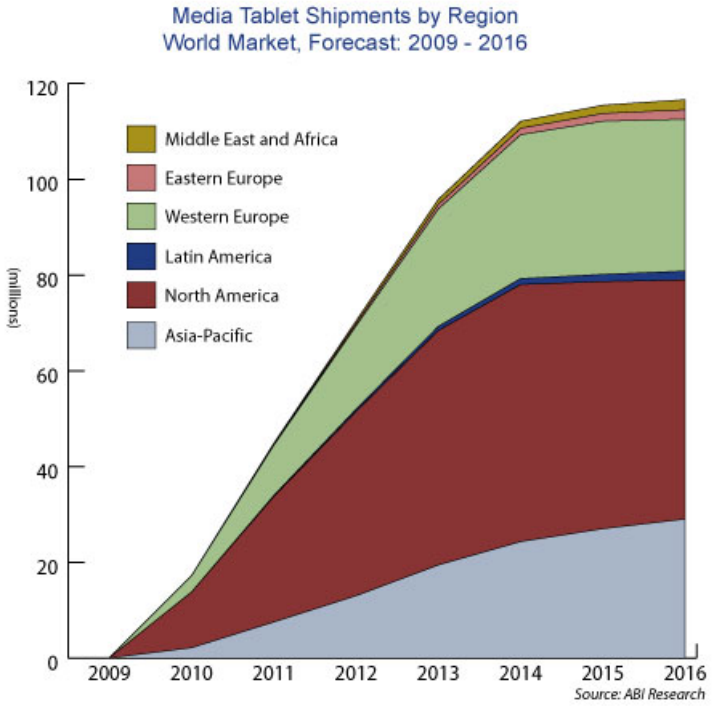


FIGURE 5.4 – Croissance du marché des tablettes PC et projections (source :abiresearch.com (2011))

Ceci va permettre, dans notre cas de vente du logiciel usiSketch à des entreprises de développement, d'avoir une opportunité de plus facilement convaincre l'acheteur en lui expliquant qu'usiSketch est utilisable sur n'importe tablette, quel que soit son OS*.

5.4.4 Threats

Une première menace est l'existence d'une concurrence élevée proposant de répondre au même besoin de base mais d'une manière différente. Ceci est une menace mais n'est

pas trop importante car nos attributs nécessaires et supplémentaires sont différents de ceux de nos concurrents.

produit nouveau

Une autre menace plus importante est le cas d'une grosse entreprise, comme Microsoft, qui dispose d'équipes de développement considérables et de moyens importants. Si une telle entreprise décide de se lancer sur le même segment, nous pourrions être en danger. Par exemple : Microsoft dispose déjà du logiciel de prototypage SketchFlow, dont un outil d'annotation par dessin est déjà présent. Il pourraient donc l'étendre pour permettre la reconnaissance de forme. Les possesseurs de SketchFlow disposeraient alors du même type de technologie et pourraient donc s'implanter sur notre marché.

5.4.5 Conclusion

Strengths	**Weaknesses**
- technologie innovantes - utilisation d'un format commun pour toutes les plateformes - conception possible pour un non-informaticien - Produit protégé par un brevet	- usiXML neuf, peu connu - prix des tablettes graphiques - usiSketch toujours en développement
Opportunities	**Threats**
- Marché de l'IT en croissance - Variété des produits présent sur le marché - Démocratisation des tablettes graphiques	- Offre de produits répondants au même besoin élevé - Arrivée d'une grosse entreprise sur le marché

FIGURE 5.5 – Résumé de l'analyse SWOT

Pour émettre notre conclusion de cette analyse nous devons d'abord déterminer si nous optons pour une stratégie déduite ou construite. La stratégie déduite estime si les caractéristiques internes à l'entreprise (forces/faiblesses) sont aptes à faire face aux caractéristiques extérieurs à l'entreprise (menaces/opportunités). La stratégie construite

estime si certaines opportunités peuvent être crée afin de mieux profiter des avantages propres à l'entreprise (strengths, weaknesses) . Nous optons dans notre cas pour une citation stratégie déduite. En effet, dans notre cas ce sont les aspects extérieurs à l'entreprise qui sont à la base de l'idée de créer un meilleur outil de prototypage d'interface graphique. Nous prendrons un à un les caractéristique externes à l'entreprise (opportunités et menaces) et nous justifierons chacun de ces points par une caractéristique interne à l'entreprise (forces et faiblesses) justifiant ainsi notre produit sur l@e marché existant.

Les opportunités :

- Le marché de l'IT en croissance : usiSketch est un outil de prototypage d'UI. Les UI font partie du marché de l'IT.
- La variété des produits présent sur le marché : Il n'y a pas un seul outil qui détient le monopole pour satisfaire le besoin. La multitude des produits permet d'avoir des consommateur plus ouvert à d'autres possibilités d'outillage pour ce besoin. Les entreprises de développement ne sont donc pas fixées à un seul outil de prototypage et sont plus aptes à découvrir usisketch.
- La démocratisation des tablettes graphiques : Celle-ci permet de réduire le prix vendu au consommateur et ainsi démocratiser le prix de l'ensemble de notre produit.

Les menaces :

- Offre de produits répondants au même besoin élevé : Notre produit se différencie fortement des autres produits répondant à ce besoin. Il utilise une technologie basée sur l'esquisse et utilise le langage usixml pour se différencier. Notre avantage concurrentiel est estimé comme fort vu l'innovation technique que constitue notre produit.
- Arrivée d'une grosse entreprise sur le marché : Notre produit sera protégé par des licences appropriée reprises dans le partie considérations juridiques.

Nous concluons que pour ce qui est de l'analyse SWOT notre produit est viable et répond au Opportunités et menaces du marché.

5.5 Les 5 forces de porter

Nous avons décidé d'établir une analyse de porter car celle-ci permet de cibler les éléments influençant les performances de l'entreprise. Dans notre cas les 5 forces peuvent être détaillées suivant le schéma suivant.

5.5.1 Le pouvoir de négociation des clients

Dans notre cas les clients sont les entreprises de développement. Sur le marché d'outil de prototypage le pouvoir de négocier est très varié en fonction de l'entreprise choisie. Les aspects qui pourraient nuire à l'entreprise et augmenter le pouvoir de négociations du clients sont les suivants : Premièrement, le fait que la manière dont nous voulons solutionner le même problème que nos concurrents est différente et nouvelle. Ceci entraine donc un manque d'habitude et de connaissance du client envers notre business procès. Deuxièmement, le fait que l'entreprise est nouvelle entraine également un manque de clarté au niveau de la fiabilité de l'entreprise. Même si nous nous adressons à des clients B2B qui s'y connaissent et se rendent compte de l'avancée technologique les clients ne savent pas ce qui à déjà été accomplis par l'entreprise cela entraine donc un plus grand pouvoir de négociations. On considère donc en conclusion que celui-ci est élevé voir modéré.

5.5.2 Le pouvoir de négociation des fournisseurs

Le seul fournisseur pouvant avoir de l'influence sur nous est celui des tablettes graphiques. Comme fournisseurs de tablette graphique nous avons choisi deux marques ou entreprises différentes.

La première est l'entreprise LUCID basée à Liège que nous avons été visiter. Celle-ci conçoit des tablettes fixes ou mobiles à usage professionnel.

La seconde marque est Wacom, qui fournit principalement des tablettes portable ou transportable à destination d'utilisation professionnelle et grand-publique. Leurs pouvoir est mitigé car le prix et les produits substituts à leurs solutions peuvent être facilement trouvé sur le marché. A coté de ces deux fournisseurs de tablette graphique un autre fournisseur peut être cités qui est le fournisseur d'emballage carton de nos produits.

Étant donné de l'existence d'une vaste offre sur ce produit le pouvoir de négociation est considéré comme faible. D'autres fournisseur ne sont pas notable. Notre produit se situe principalement dans ce que nous produisons en interne à savoir le logiciel usiSketch. Nous considérons que le pouvoir de négociations des fournisseurs est faible.

5.5.3 La menace de produits de substitutions

L'ensemble des produits de substitution peu être retrouvé dans la partie State of the art. Étant donné l'avantage concurrentielle que constitue notre produit nous concluons que cette menace est faible. En effet notre outil de prototypage est bien plus performant et la manière d'innover lui permet de se distinguer fortement des autres solutions présetnes sur le marché. Le seul point qui peu susciter une menace considérable est que le prix de notre produit se situe au dessus de l'outil le plus employé qui est le prototypage par dessin.

5.5.4 L'intensité de la concurrence intra-sectorielle

Celle-ci est inexistante car notre produit est totalement neuf et qu'aucune autre entreprise ne s'est encore positionné de cette manière sur le marché d'outil de prototypage.

5.5.5 La menace d'entrants potentiels

Comme nous le disions dans l'analyse SWOT dans la partie des menaces venant de l'extérieur de l'entreprise nous estimons modérée l'arrivée de nouveau entrant dans notre secteurs d'activité. En effet l'entrée nécessite un certain investissements de la part de ceux qui veulent se lancer dans le même domaine avec le même business process. La crainte majeure réside donc comme cité plus haut, dans les grand groupes qui auraient les moyens de lancer une grosse équipe sur le projet durant quelques mois. Ces entreprises, ayant une structure d'employés conséquentes pourraient en effet rapidement créer un service similaires au notre.

5.5.6 Conclusion

Nous observons que les pouvoir des acteurs externe à notre entreprise est modérée vu de tous les facteurs pris en considération dans cette analyse de porter. Il sera judicieux de tenir en compte de chaque partie de cette analyse séparément pour ce qui est du positionnement et de l'effort à délivrer pour dévelloper certains attributs.

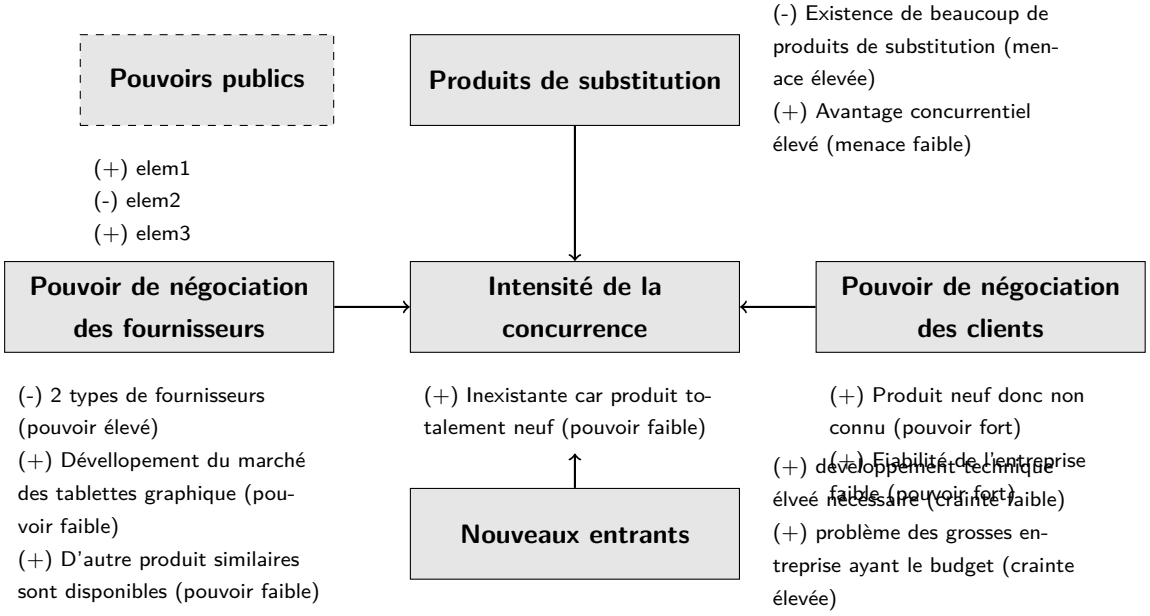


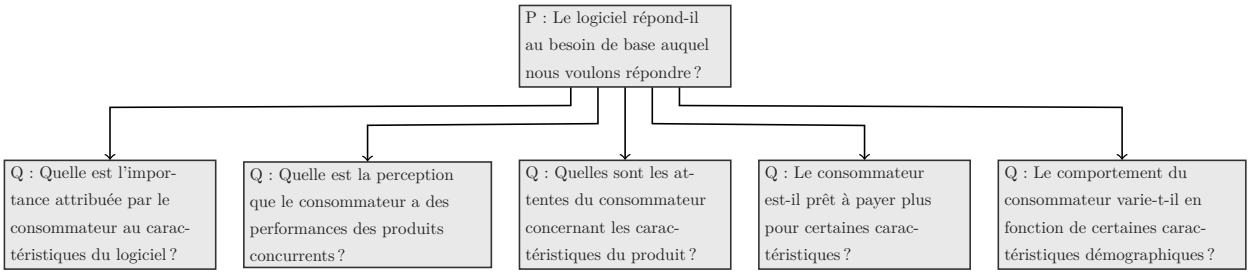
FIGURE 5.6 – Analyse de Porter

5.6 L'élaboration des hypothèses *a priori* et à tester

Cette partie à pour but de poser les hypothèses à prioiri et a tester qui vont pouvoir éclairer les choix à prendre lors du positionnement du projet à lancer.

5.6.1 Méthodologie

Dans cette section, nous reprendrons les concepts théoriques utilisés ainsi qu'une explication de la raison d'utilisation de ceux-ci. Le raisonnement suivi dans l'élaboration de l'étude de marché est inspiré par la reference :section 5.6. Il part d'une question de recherche principale subdivisée ensuite en plusieurs question de recherche. Chaque question de recherche est elle même divisée en plusieurs Hypothèses à tester et à priori qui seront expliquée plus largement dans cette section. Le plan suivant reprend le raisonnement pour l'élaboration des hypothèses à tester et à priori.



Les origines des hypothèses

Commençons par déterminer la manière dont nous avons posé les hypothèses pour l'élaboration de ce mémoire. Plusieurs manières sont possibles pour les trouver : L'observation courante, la découverte fortuite, l'élaboration théorique, l'imagination, le travail de défrichage et de pré-enquête.Aktouf et Omar (1987). Dans notre cas, les méthodologies suivantes ont étés abordées :

La découverte fortuite

Celle-ci décrit le fait de réfléchir par accident à certains aspects dans le cours d'élaboration de l'enquête. Cette manière nous à permis de révéler avec Mme de Moerloose plusieurs hypothèses qui n'avaient à la base pas été citées. En discutant de certaines

hypothèses , il à été possible de se rendre compte, de diviser, ou d'ajouter d'autres hypothèses. Comme exemple nous pouvons citer l'émission de deux hypothèses qui ont par la suite été transformée en une seule hypothèse :

Hypothèse 1 : "Le client est sensible à l'age du fournisseurs en interface de logiciels"
Hypothèse 2 : "Le client est sensible à la renommée du fournisseurs en interface de logiciels"

Transformé en :

Hypothèse : "Le client est sensible à la fiabilité du fournisseur en interface de logiciels"

Le travail de déchiffrage et de pré-enquête

Cette manière d'élaboration d'hypothèses est celle qui est la plus laborieuse car elle nécessite des recherches dans des ouvrages et des rencontres avec des personnes en lien avec le sujet de la recherche. Lors de nos différentes recherches nous avons eu l'occasion de rencontrer Mr Ivanov, directeur général de l'entreprise Pulsar consulting basée à Wavre, Monsieur Thomas employé de l'entreprise Tecteo, l'entreprise Lucid qui développe des écrans tactiles par rétro projection ou encore Monsieur Laurent qui nous à éclairé sur les questions juridiques. Lors des rencontres avec ces personnes, plusieurs questions et hypothèses ont pu être établies afin de créer un questionnaire plus pertinent pour notre enquête.

5.6.2 Le choix des échelles et le type de test effectués

Durant l'élaboration du questionnaire, nous avons du choisir le type d'échelle et test que nous allions utiliser afin de recueillir de la plus juste manière les résultats souhaités. Dans cette section un descriptif du choix des échelles et des tests sera justifié et expliqué. La section suivante expliquera en détails les faiblesses et forces de chacun des test choisis et les limites d'interprétation des donnée ressortant de l'analyse statistiques. Ces types d'échelles ont été trouvés à la référence suivanteAktouf et Omar (1987).

L'échelle nominale

Celle-ci nous à été utile pour répertorier des répondants au questionnaire dans des catégories précises. La seule propriété d'une échelle nominale est qu'elles sont différentes

chacune des autres. Les différentes échelles n'ont aucun lien entre elles,il n'y a pas d'ordre. Typiquement les questions utilisant ce type de classification permettent de refléter si un individu fait partie d'un groupe d'individu ou non.

L'échelle Métrique

Cette échelle est utile pour les variables portant sur l'identification, le classement et le rapport de différences entre les objets testés. Cette échelle à principalement été utilisée pour déterminer l'attitude du répondant et l'importance qu'il accorde face à son choix de fournisseurs et de ce qu'il est prêt à payer plus ou non, pour certains attributs.

5.6.3 Le choix des test effectués

Une fois le choix des échelles effectués pour chacune des variables, une analyse statistique s'est relevé judicieuse pour limiter les erreurs d'observation et de disposer d'outils objectif de prise de décision. Les tests statistiques qui ont été utilisés sont le test de Student ainsi que le test Chi-carré. Pour chaque test une hypothèse nulle à tester est déterminée. Les deux tests effectués sont univarié pour le test de student et Bi-varié pour celui de Chi-carré.

Le test-t de student

Le test de student est utilisé pour déterminer la p-valeur de la statistique de Student calculée de l'échantillon observé pour les variables métrique ou la moyenne peut être utilisée pour représenter la tendance centrale de la variable. Ce test à du être utilisé au détriment du théorème central-limit étant donné que notre échantillon est petit et n'excède pas 30 personnes. Ce test nous donne après calcul de la statistique de Student, la p-valeur équivalente à notre échantillon qui détermine si oui ou non l'hypothèse nulle peut être rejetée ou non. Si la p-valeur observée est plus petite que 5% alors il y a rejet de l'hypothèse nulle. Si la p-valeur observée est plus grande que 5% alors il y a non rejet de l'hypothèse nulle.

Le test Chi-carré

Le test chi-carré est un test bivarié qui permet de déceler si il y a un lien entre deux variable différentes. Il permet donc de savoir si des personnes répondant à l'étude de marché une réponse A à une question précise ont tendance à répondre B à une autre question complètement différente.

5.6.4 Hypothèses à Priori :

Les hypothèses à priori sont celles qui ne sont pas à questionner. Nous les justifierons brièvement ces hypothèses avant de les poser.

Tablette graphique :

Les hypothèses à priori qui concernent l'utilisation de la tablette graphique sont cités ci-dessous. Ils reprennent les principales caractéristiques de la tablette. Ces hypothèses sont donc considérées de fait et ne sont donc pas remises en question par notre enquête. Elles ont été établies par questionnement des avantages et inconvénients liées à l'utilisation d'un tel outil.

- Ergonomique
- Illustratif
- Facile à transporter
- Fait le lien entre le dessin et le langage
- Précision du trait dessiné grâce à l'outil
- Coût élevé d'une tablette
- Apprentissage d'utilisation
- Dépendant du langage usiXML travaillant avec Usisketch.

Concurrents :

Les produits proposant une solution au même besoin sont les suivants : Sketchflow, Irise, Papier-crayon, Sketchixml. Ils ont été plus longuement reprise dans le Chapitre : chapitre 3.

5.6.5 Hypothèses à Tester :

Les hypothèses à tester sont toutes les hypothèses émises lors de nos questions de recherche qui n'ont pas pu être élucidées, ou qui doivent l'être d'avantage à travers une étude de marché sur le comportement du consommateur final. Pour chacune des hypothèse à tester une échelle de mesure et le choix d'un test statistique sera déterminé. Un tableau récapitulatif reprendra toutes les questions du questionnaire relative au hypothèses testées ainsi que leurs échelle et le choix du test statistique effectué.

- Quelle est l'importance attribuée par le consommateur au caractéristiques du logiciel ?
- Quelle est la perception que le consommateur a des performances des produits concurrents ?
- Quelles sont les attentes du consommateur concernant les caractéristiques du produit ?
- Le consommateur est-il prêt à payer plus pour certaines caractéristiques ?
- Le comportement du consommateur varie-t-il en fonction de certaines caractéristiques démographiques ?

Quelle est l'importance attribuée par le consommateur au caractéristiques du logiciel ?

Ce paragraphe émet les hypothèses à tester en rapport avec l'importance que chaque individu accorde au attribut de notre produit.

Hypothèse 1 : L'acheteur accorde une importance différente aux attributs suivants décrivant la création d'un site internet et d'un ERP lors du choix de son fournisseur.

- A : La capacité à changer de fournisseur de site internet ou d'ERP.
- B : La fiabilité de l'entreprise.
- C : Le prix du produit.
- D : Le prix du service après vente.
- E : La connaissance par son membre du personnel informatique du langage informatique utilisé.
- F : Le nombre de jours d'attente entre le premier contact avec le créateur et la mise en place du site internet ou de l'ERP

Choix de l'echelle : ordinale univarié
Choix du test : Kolmogrov-Smirnov

Hypothèse 2 : L'acheteur accorde une importance différente aux attributs suivants décrivant la création d'un site internet et d'un ERP lors du processus de création.

- A : Le nombre de rendez-vous nécessaire pour finaliser le produit.
- B : La pédagogie utilisée par le concepteur.
- C : L'interactivité avec le concepteur lors du processus de création. (facilité de modification, illustration en temps réel, le dialogue avec le créateur).
- D : La démonstrabilité immédiate du produit (le client voit tout de suite le résultat final à l'aide d'une tablette graphique).

Choix de l'echelle : ordinale univarié
Choix du test :Kolmogrov-Smirnov

Hypothèse 3 : L'acheteur accorde une importance différente aux attributs suivants décrivant la création d'un site internet et d'un ERP lors de la mise en place.

- A : La rapidité du service après vente en cas de panne.
- B : La rapidité de mise à jour du site internet.

Choix de l'echelle : ordinale univarié
Choix du test :Kolmogrov-Smirnov

Quelle est la perception que le consommateur a des performances des produit concurrents ?

Hypothèse 4 : Si le consommateur est déjà client chez un fournisseur d'UI il n'est pas assez satisfait des caractéristiques suivantes d'un outil de prototypage.

- A Le client est déjà client chez un concurrent
- B La dépendance de son fournisseur actuel de service IT est trop élevée.
- C La fiabilité du fournisseur est trop faible.
- D Le prix du produit est trop faible.
- E Le prix du service après vente est trop élevé.
- F Le langage informatique utilisé par la concurrence est mieux connu par son service informatique.
- G La rapidité de livraison du produit final est trop faible.
- H Le nombre de rendez-vous chez les concurrents est trop élevé.
- I Le concepteur n'a pas de pédagogie facile pour expliquer des concepts informatiques précis.
- J La démonstrabilité du produit en cours de conception du concurrent est trop faible.
- K Le service après vente est trop lent chez les concurrents
- L La mise à jour du produit livré est trop lente.

Choix de l'échelle pour l'hypothèse 4.1 : Nominale Choix du test pour l'hypothèse 4.1 : Proportion Choix de l'échelle pour les autres hypothèses : ordinale univarié
Choix du test pour les autres hypothèses :Kolmogrov-Smirnov

Quelles sont les attentes du consommateur concernant les caractéristiques du produit ?

Hypothèse 5 : Le consommateur à des attentes spécifiques pour ce qui est des attributs suivants lors du choix de son fournisseur d'UI.

- A Au prix du Produit.
- B Au prix du service après vente.
- C La rapidité de livraison du produit final. (en nombre de jours)
- D Au nombre de rendez-vous nécessaire avec le créateur pour déterminer le produit en détails.
- E A la rapidité du service après vente.
- F A la rapidité de mise à jour du produit.

Choix de l'échelle : métrique univarié
Choix du test : student

Le consommateur est-il prêt à payer plus pour certaines caractéristiques ?

Hypothèse 6 : L'acheteur est prêt à payer plus/moins pour les caractéristiques suivantes lors du processus de création du logiciel.

- A Améliorer la rapidité.
- B Une meilleure interactivité.
- C Avoir le choix du langage utilisé pour le langage.
- D Baisser le nombre de rendez- vous .
- E Baisser la dépendance à un fournisseur particulier de service informatique.
- F Améliorer la fiabilité du fournisseur.

Choix de l'échelle : métrique univarié

Choix du test : student

Le comportement du consommateur varie-t-il en fonction de certaines caractéristiques démographiques ?

Hypothèse 7 : Le comportement du consommateur sur toutes les hypothèses précédentes varie en fonction des caractéristiques démographiques suivantes.

- A Le répondant à un lien quelconque avec le service IT dans l'entreprise dans laquelle il travaille.
- B Le chiffre d'affaires de l'organisation.
- C Le nombre d'employés dans l'organisation.
- D La langue principale utilisée dans l'organisation.
- E Le client dispose déjà d'un service IT au sein de l'entreprise.

Choix de l'échelle : métrique univarié ou ordinale univarié

Choix du test : test de comparaison deux moyennes ou le test U de Mann-Whitney.

5.7 La réponse à l'étude de marché et les conclusions à en tirer

Suite à l'ensemble des hypothèses à tester un questionnaire à été élaboré pour pouvoir questionner le marché sur la véracité des Hypothèses émises. Les individus ont été questionnés sur les hypothèses tentant de répondre aux 5 questions de recherche.

- Quelle est l'importance attribuée par le consommateur aux caractéristiques du logiciel ?
- Quelle est la perception que le consommateur a des performances des produits concurrents ?
- Quelles sont les attentes du consommateur concernant les caractéristiques du produit ?
- Le consommateur est-il prêt à payer plus pour certaines caractéristiques ?
- Le comportement du consommateur varie-t-il en fonction de certaines caractéristiques démographiques ?

Nous allons parcourir chaque réponse aux questions de recherches, divisée en plusieurs sous-questions, pour emmètre des conclusions pour chacune d'entre elles basée sur l'-analyse de la moyenne et de l'écart-type de chaque statistique. Ensuite, après avoir appliqué un tri-croisé entre la dernière question de recherche triant les répondant sur des caractéristiques démographiques, d'autres conclusions pourront être tirée afin de mieux cerner la stratégie à adopter.

18 personnes ont répondues à notre étude de marché. Ce nombre peu élevé est la conséquence du caractère précis du domaine dans lequel nous effectuons notre recherche. Le nombre de répondants peut varier par question vu qu'il y a des question qui n'ont pas du être répondue par certains répondants ne rentrant pas dans certains critères. Dans le mot d'introduction du questionnaire nous avons mentionné qu'il était requis d'avoir des compétences en informatique pour pouvoir répondre justement à chacune des questions. Néanmoins nous estimons suffisant le nombre de répondant vu la précision et la complétude du questionnaire.

Chaque tableau reprend chacun des attributs testé et reflète la capacité à utiliser les données par les résultats obtenus dans les différents tests.

référence annexe

5.7.1 Analyse primaire

Quelle est l'importance attribuée par le consommateur aux caractéristiques du logiciel ?

La première question en rapport avec l'importance est la suivante :
Précisez, sur une échelle de 1 (très peu important) - 4 (très important) l'importance que vous consacrez au attributs suivants lors du choix de votre fournisseur de site Internet ou ERP.

$H_0 : U < 2,5$
$H_1 : U \geq 2,5$
Échelle de la variable : Métrique
Test statistique : Student

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q1.1. Capacité à changer de fournisseur	23	2.522	1.016	0.1038	$\alpha > 0.05$	H_0 non rejeté
Q1.2. Fiabilité	25	3.48	0.7	7	$\alpha \leq 0.05$	H_0 rejeté
Q1.3. Prix du produit	25	3.56	0.637	8.3203	$\alpha \leq 0.05$	H_0 rejeté
Q1.4. Prix du SAV	25	3.28	0.776	5.0258	$\alpha \leq 0.05$	H_0 rejeté
Q1.5. Langage utilisé	21	2.571	0.849	0.3832	$\alpha > 0.05$	H_0 non rejeté
Q1.6. Rapidité de livraison	24	3.208	0.763	4.5458	$\alpha \leq 0.05$	H_0 rejeté

Légende :
1. très peu important
2. peu important
3. important
4. très important

Suite à l'analyse des données et au choix du degré de 90% minimum requis de significativité pour les données seuls les questions 1.2, 1.3, 1.4, 1.6 rejettent l'hypothèse nulle.

Nous observons que le répondant est très sensible à la fiabilité, au prix du produit et SAV ainsi qua la rapidité de livraison. Ceci est favorable sur les trois derniers attributs car notre outil y est très compétitif. La sensibilité du répondant au prix est positive pour nous car l'achat de notre produit vise à réduire le temps de travail, et donc le prix nécessaire afin d'effectuer le prototypage d'UI. L'importance accordée à la rapidité de livraison est également très bonne pour nous car le principal attribut de notre produit est le gain de temps par la diminution du nombre de rendez-vous nécessaire pour cibler les désidératas du client. Par contre, la fiabilité est un point faible pour nous car si nous nous lancons comme entreprise de développement, notre ancienneté ne pourra pas prouver notre fiabilité lors de la vente de nos produits.

La deuxième question portant sur l'importance testée les attributs du produit concernant le processus de création de votre site internet / ERP par une entreprise externe à votre organisation. La question est la suivante :

Précisez, sur une échelle de 1 (très peu important) - 4 (très important) l'importance que vous consacrez aux attributs suivants lors du processus de création de votre site internet / ERP par une entreprise externe à votre organisation.

Ho : L'importance attribuée par les répondants ne peut pas être reflété par une tendance centrale.
H1 : L'importance peut être reflétée par une tendance centrale.
Échelle de la variable : Ordinale
Test statistique : Kolmogrov-Smirnov

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q2.1. Nombre de rendez-vous	24	2.833	0.943	1.73	$\alpha \leq 0.05$	H_0 rejeté
Q2.2. Pédagogie utilisée	25	3.28	0.722	5.4017	$\alpha \leq 0.05$	H_0 rejeté
Q2.3. Interactivité	23	3.043	0.751	3.4676	$\alpha \leq 0.05$	H_0 rejeté
Q2.4. Démonstrabilité	24	3.25	0.661	5.5586	$\alpha \leq 0.05$	H_0 rejeté

Légende :
1. très peu important
2. peu important
3. important
4. très important

Suite à l'analyse des données nous retiendrons les questions 2.2, 2.3, 2.4.

Sur ces données nous observons que la pédagogie, l'interactivité et la démonstrabilité sont importants au yeux du répondant. Ceci est positif pour le développement de notre produit car ses attributs décrivent les performances clefs de notre produit. Il est alors primordial lors de la vente de notre produit de mettre ces atouts en avant. Nous observons également que le répondant accorde une importance moyenne au nombre de rendez-vous nécessaire à l'élaboration de notre produit.

La troisième question portant sur l'importance teste les attributs du produit lors de la mise en place (livraison) de l'ERP / Site internet. La question est la suivante :

Précisez, sur une échelle de 1 (très peu important) - 4 (très important) l'importance que vous consacrez aux attributs suivants lors de la mise en place (livraison) de votre ERP / Site internet par une entreprise externe à votre organisation.

Ho : L'importance attribuée par les répondants ne peut pas être reflété par une tendance centrale.
H1 : L'imortance peut être reflété par une tendance centrale.
Échelle de la variable : Ordinale
Test statistique : Kolmogrov-Smirnov

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q3.1. Rapidité du SAV	25	3.36	0.742	5.7951	$\alpha \leq 0.05$	H_0 rejeté
Q3.2. Rapidité de mise à jour	24	3.25	0.829	4.4321	$\alpha \leq 0.05$	H_0 rejeté

Légende :

1. très peu important
2. peu important
3. important
4. très important

Nous pouvons tenir compte du premier attribut testé.

Cet attribut est important aux yeux du répondant. Ceci est positif car, comme cité précédemment, notre produit vise, grâce à l'utilisation d'une tablette graphique combiné au logiciel usiSketch à réduire le temps de travail nécessaire à la création mais également à la modification d'une UI*. La médiane de la mise à jour n'est représentative qu'a 80%. Ceci est probablement à l'image différente que peut avoir un répondant d'une entreprise à l'autre sur la rapidité de la mise à jours d'une interface graphique.

Quelle est la perception que le consommateur a des performances des produits concurrents ?

Cette question est en réalité la 6ième question du questionnaire. Les questions 4 et 5 reprennent des caractéristiques démographiques. Ce sont des variable à échelle nominale. La question 4 informe si le répondant est déjà client ou non d'un développeur et la question 5 demande le nom de ce dernier. La réponse à cette question est que 61% des répondants disposent aujourd'hui d'un fournisseur en IT. Ces informations seront reprises et utilisées lors de la dernière question de recherche qui concerne croisera certaines informations afin d'établir des liens de comportements entre l'information démographique et les résultats des réponses précédentes. La réponse à la question 5 reprenant le nom de fournisseurs n'a eu que 3 réponses et est donc considérée comme non traitable. La question 6 est formulée comme ceci :

Sur une échelle de 1 (pas du tout d'accord) à 4 (tout à fait d'accord) êtes vous plus ou moins d'accord avec les propos suivants ? Ho : L'importance attribuée par les

répondants ne peut pas être reflété par une tendance centrale.

H1 : La tendance centrale peut être reflété par une tendance centrale.

Échelle de la variable : Ordinale

Test statistique : Kolmogrov-Smirnov

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q6.1. Trop dépendant du fournisseur actuel	15	2.733	0.929	0.9714	$\alpha > 0.05$	H_0 non rejeté
Q6.2. Fournisseur pas assez fiable	17	2	0.686	3.0052	$\alpha \leq 0.05$	H_0 rejeté
Q6.3. Prix du produit élevé	15	3.067	0.68	3.2294	$\alpha \leq 0.05$	H_0 rejeté
Q6.4. Prix du SAV élevé	15	3.2	0.748	3.6244	$\alpha \leq 0.05$	H_0 rejeté
Q6.5. Langage pas assez connu	10	1.9	0.7	2.7105	$\alpha \leq 0.05$	H_0 rejeté
Q6.6. Livraison trop lente	16	3.063	0.899	2.505	$\alpha \leq 0.05$	H_0 rejeté
Q6.7. Nombre de rendez-vous trop élevé	15	3	0.966	2.0046	$\alpha \leq 0.05$	H_0 rejeté
Q6.8. Pas assez pédagogique	14	2.714	0.7	1.1439	$\alpha > 0.05$	H_0 non rejeté
Q6.9. Mauvaise vision du résultat durant la conception	17	2.765	1.002	1.0904	$\alpha > 0.05$	H_0 non rejeté
Q6.10. SAV trop lent	17	2.824	0.785	1.7018	$\alpha > 0.05$	H_0 non rejeté
Q6.11. Vitesse de mise à jour top faible	15	2.667	0.789	0.8198	$\alpha > 0.05$	H_0 non rejeté

Légende :

1. pas du tout d'accord
2. pas d'accord
3. d'accord
4. tout à fait d'accord
5. non répondu

Cette analyse ne nous apporte que peu d'information. Ceci est principalement du au petit nombre de répondants ayant complété cette réponse. En effet avant de commencer cette réponse il était demandé au répondant de déterminer si oui ou non il disposait au sein de son organisation d'un fournisseur de service IT ou non. Si le répondant répondait

négativement il passait à la question suivante. La seule information interpretable dans ces données est la question 6.10 relative au caractère lent du service après vente.

Le consommateur est-il prêt à payer plus pour certaines caractéristiques ?

La première question sur cette section est la suivante :

Lors de l'acquisition d'un site internet ou d'un ERP précisez le budget que vous estimez raisonnable de payer pour un site internet d'une part et d'un ERP d'autre part

Ho : L'intervalle de confiance est trop large que pour représenter une tendance centrale dans la valeur estimée.
H1 : La valeur estimée de l'intervalle de confiance est concentrée.
Échelle de la variable : métrique
Test statistique : Student

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q7.1. Budget site internet [€]	25	6500	3741.657	8.6827	$\alpha \leq 0.05$	H_0 rejeté
Q7.2. Budget ERP [€]	18	11388.889	5414.886	8.9214	$\alpha \leq 0.05$	H_0 rejeté

Légende :

2500. $<$ 5000 €
7500. 5000 - 10000 €
12500. 10000 - 15000 €
17500. 15000- 20000 €

Dans cette question, la valeur attribuée à un site internet et d'un ERP .

Celle-ci est estimée à une moyenne de 0 à 10000 euros. Cette valeur est difficile et trop vaste à interpréter. Une moyenne combinée à un écart-type plus précis nous aurait

permis d'être plus rigoureux sur le prix estimé d'un site internet. La valeur estimée d'un ERP ne peut pas être prise en compte. La raison pour laquelle cette valeur est trop dispersée au yeux du consommateur est probablement du à la vaste possibilité entourant la création d'un ERP. Un ERP pour une petite PME sera tout à fait diffèrent, vu le nombre de données à traiter d'un ERP d'une grande multinationale. Le site sera lui aussi différent mais la différence de prix sera moins notable que lors de la création d'un ERP.

La deuxième question concernant cette partie est la suivante :
Précisez le budget que vous estimez raisonnable de payer pour une service après vente (Deux rendez-vous d'une heure avec livraison du service)

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q8. Budget SAV (RDV 2h+service) [€]	25	315	177.2	8.8177	$\alpha \leq 0.05$	H_0 rejeté

Légende :

125. $<$ 250 €
375. 250 - 500 €
625. 500- 750 €
1500. $>$ 1000 €

L'écart type étant suffisamment petit, nous observons que la moyenne reflète un prix majoritairement situé entre 250€et 500€. Ce chiffre sera interprété plus tard lors de l'établissement de notre stratégie de prix et du business plan à la fin de ce mémoire.

Les 4 questions suivantes portent sur le délai d'attente pour plusieurs services nécessaires au produit. Nous avons regroupé toutes les questions reflétant un délai et organisé une seule conclusion à la fin de ces 4 questions.

Quel délai de livraison du site internet estimez vous raisonnable entre le jour du premier contact avec le fournisseur IT et la livraison finale de ce dernier ?

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q9. Délai de livraison site internet [jours]	25	35.7	20.598	8.059	$\alpha \leq 0.05$	H_0 rejeté

Légende :

7.5. $<$ 15 Jours

22.5. 15Jours - 1 mois

45. 1- 2 mois

75. 2-3 mois

135. 3-6 mois

270. $>$ 6mois

Quel délai de livraison d'un ERP estimez vous raisonnable entre le jour du premier contact avec le fournisseur IT et la livraison finale de ce dernier ?

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q10. Délai de livraison ERP [jours]	25	98.1	52.817	9.0501	$\alpha \leq 0.05$	H_0 rejeté

Légende :

7.5. $<$ 15 jours

22.5. 15jours - 1 mois

45. 1 - 2 mois

75. 2 - 3 mois

135. 3 - 6 mois

270. $>$ 6 mois

Quel délai de service après vente (Site internet/ ERP confondu) estimez vous raisonnable entre la prise de contact avec le fournisseur et la livraison du travail effectué ?

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q11. Délai SAV [jours]	25	4.28	2.298	3.8729	$\alpha \leq 0.05$	H_0 rejeté

Légende :

1. $<$ 2jours
4.5. 2 jours - 1 semaine
10.5. 1 semaine - 2 semaines
21. 2 semaines - 1 mois
45. $>$ 1 mois

Quel délai de mise à jour (Site internet/ ERP confondu) estimez vous raisonnable entre la prise de contact avec le fournisseur et la livraison du travail effectué ?

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q12. Délai mise à jour [jours]	25	4.92	5.023	2.4089	$\alpha \leq 0.05$	H_0 rejeté

Légende :

1. $<$ 2jours
4.5. 2 jours - 1 semaine
10.5. 1 semaine - 2 semaines
21. 2 semaines - 1 mois
45. $>$ 1 mois

Les données utilisables et assez précises sont les données illustrant le délai de livraison d'un site internet et le délai de livraison d'un service après vente.

Le délai de livraison de site internet moyen se situe entre 15 jours et un mois. Le délai de service après vente se situe entre deux jours et une semaine et dispose d'un écart-type très petit. Les autres données ont probablement un écart-type trop grand dû à la diversité des produits rentrant dans la description du produit cité. C'est la même remarque à effectuer lors du prix estimé pour la création d'un ERP qui est diffèrent d'une multinationale à une PME.

La dernière question relative aux intentions d'achat illustre le pourcentage en prix que les répondants sont prêt à payer en plus pour améliorer cerains attributs ; la question

est la suivante :

Décrivez la différence de prix en pourcentage que vous êtes prêts à payer pour améliorer les attributs suivants

Question	Nb	Moyenne	E-Type	V. table	p-Valeur	Décision
Q13.1. Différence de prix - rapidité d'élaboration [%]	25	9.8	4.946	7.3797	$\alpha \leq 0.05$	H_0 rejeté
Q13.2. Différence de prix - meilleure interactivité avec tablette [%]	23	9.674	7.383	4.6601	$\alpha \leq 0.05$	H_0 rejeté
Q13.3. Différence de prix - choix du langage informatique [%]	18	5.833	3.997	3.5378	$\alpha \leq 0.05$	H_0 rejeté
Q13.4. Différence de prix - baisser nombre de rendez-vous [%]	22	12.614	9.06	5.2361	$\alpha \leq 0.05$	H_0 rejeté
Q13.5. Différence de prix - réduire dépendance fournisseur IT [%]	21	6.429	6.527	2.7585	$\alpha \leq 0.05$	H_0 rejeté
Q13.6. Différence de prix - augmenter fiabilité du fournisseur [%]	24	13.646	7.464	7.3157	$\alpha \leq 0.05$	H_0 rejeté

Légende :

2.5. < 5

7.5. 5-10

15. 10-20

30. > 20

Les données utilisables sont les suivantes : 13.1, 13.3, 13.4 et 13.5. Nous observons que le répondant est prêt a augmenter son prix de 5 à 10% pour améliorer la rapidité d'élaboration et de baisser le nombre rendez-vous nécessaire. Pour ce qui est du choix du langage informatique et la réduction de la dépendance au fournisseur, il n'est en moyenne que prêt à payer 0 à 5% de plus. Ces données sont importante en ce qui concerne notre stratégie de prix à adopter et la manière de vendre notre produit.

Le comportement du consommateur varie-t-il en fonction de certaines caractéristiques démographiques ?

Toutes les questions suivantes répondent aux caractéristiques démographiques du répondant. Ces données seront par la suite croisées une à une avec les questions précédentes pour déceler une différence de moyenne et d'écart-type aux questions précédentes en fonction du statut démographique du répondant.

Avez vous un lien quelconque (de pouvoir, de conseil, de travail) avec la gestion des avoirs informatique de l'entreprise/organisation pour laquelle vous travaillez ?

Question	Moyenne	Ecart Type
Q14. Lien avec gestion informatique	1.167	0.373

1. OUI
2. NON

Quel est le chiffre d'affaires de l'entreprise pour laquelle vous travaillez ? (en milliers d'euros/an) ?

Question	Moyenne	Ecart Type
Q15. Chiffre d'affaire (k¤/an)	2.647	1.643

1. < 250
2. 250- 500
3. 500-1000
4. 1000-5000
5. 5000-10000
6. > 10000

Quel est le nombre d'employés présent au sein de l'entreprise dans laquelle vous travaillez ?

Question	Moyenne	Ecart Type
Q16. Nombre d'employés	2.412	0.974

1. < 10
2. 10-20
3. 20-50
4. 50-100

L'entreprise dans laquelle vous travaillez dispose-t-elle d ?un service IT ?

Question	Moyenne	Ecart Type
Q17. Service IT	1.412	0.492

1. OUI
2. NON

Quelle langue est la plus utilisée dans votre entreprise / organisation ?

Question	Moyenne	Ecart Type
Q18. Langue parlée	1.889	1.048

1. Francais
2. Néerlandais
3. Anglais
4. Espagnol

5.7.2 L'analyse des tri-croisés

Cette section à pour but de croiser les résultats obtenu durant la dernière partie avec les quatre premières question de recherche afin de déceler un lien quelconque avec la situation démographique de l'individu répondant au questionnaire. La totalité des tri-croisés sont ajoutés en annexe. Nous reprendrons ici uniquement les tri-croisés pertinent et ajoutant une information pouvant être utilise lors du choix de vente du produit. Nous avons testé chaque hypothèses une à une avec les 6 caractéristique démographiques reprises ci-dessus.

tri-croisé 1

tri-croisé 2

Ce tri-croisé croise les deux variables suivantes :
Importance attribuée à la capacité à changer de fournisseur
L'entreprise dispose d'un service IT.

Réponse	Moyenne	Ecart Type
ne sait pas	0	0
OUI	2.3	0.9
NON	3	0.816

1. très peu important
2. peu important
3. important
4. très important

Nous observons que les personnes répondant qui ne disposent pas d'un service IT dans leur entreprise ont tendance à attribuer plus d'importance à la capacité à changer de fournisseur. Ceci comme cité dans le tri-croisé 1 nous permet de mieux cibler la stratégie de vente à effectuer lors du lancement du produit.

tri-croisé 3

Ce tri-croisé croise les deux variables suivantes :
L'importance attribué à la fiabilité du fournisseur d'IT
La taille du chiffre d'affaire effectué par l'entreprise.

Réponse	Moyenne	Ecart Type
Ne sait pas	4	0
< 250	3.857	0.35
250- 500		
500-1000	3	0.577
1000-5000	3.5	0.5
5000-10000		
> 10000	3.5	0.5

1. très peu important
2. peu important
3. important
4. très important

Dans ce tri-croisé nous observons généralement que toute entreprise apporte un importance significative au caractère fiable du fournisseur d'IT. Plus particulièrement nous observons que l'écart-type le plus petit est combiné avec la plus grande moyenne pour les entreprise ayant le plus petit CA. Vu que notre produit est neuf et que notre entreprise n'a pas encore fait ses preuve elle ne peut pas encore être considéré comme fiable par le consommateur final. Cet aspect très marqué peut être reflète dans notre choix du business model. Ce fait nous amène de plus en plus à croire qu'un solution en B2B serait plus avantageuse afin de limiter le risque liés à cette réticence émise par le consommateur final.

tri-croisé 4

Ce tri-croisé croise les deux variables suivantes :
L'importance du prix du service après vente.

La taille du CA des entreprise consommatrice de développement de logiciels.

Réponse	Moyenne	Ecart Type
Ne sait pas	3	0
< 250	3.429	0.728
250- 500		
500-1000	3.5	0.764
1000-5000	2.5	0.5
5000-10000		
> 10000	2.5	0.5

1. très peu important
2. peu important
3. important
4. très important

Nous voyons que la moyenne de l'importance accordée au prix du service après vente est plus élevée pour les entreprise ayant un CA plus petit que 1 million d'euros par an. Ceci est important car nous voyons que ces petites entreprise sont plus sensible au prix du service après vente que les plus grandes structure. Notre outil, qui permet de réduire le temps nécessaire au prototypage d'UI mais aussi au SAV permet de réduire le prix facturé au client si cet outils est employé par une entreprise de développement.

tri-croisé 5

Ce tri-criosé croise les deux variables suivantes :
L'importance du prix du service après vente
Le nombre d'employés présent dans l'entreprise du répondant

Réponse	Moyenne	Ecart Type
Ne sait pas	3	0
< 10	3.4	0.8
10-20	4	0
20-50	3.1	0.831
50-100	3	0

1. très peu important
2. peu important
3. important
4. très important

La conclusion à ce tri-croisé va renforcer la conclusion par le tri-croisé précédente. On voit également que les petites structures accordent plus d'importance au prix du service après vente.

tri-croisé 6

Ce tri-croisé croise les deux variables suivantes :
Le délai de livraison estimé d'un ERP.
Le lien du répondant avec la gestion de l'informatique au sein de l'entreprise dans laquelle il travaille.

Réponse	Moyenne	Ecart Type
OUI	4.533	0.718
NON	2.333	0.943

1. $<$ 15 jours
2. 15jours - 1 mois
3. 1 - 2 mois
4. 2 - 3 mois
5. 3 - 6 mois
6. $>$ 6 mois

Nous observons que les répondants ayant un lien avec la gestion informatique de l'entreprise pour laquelle ils travaillent estiment le temps de livraison d'un ERP bien plus long que les personnes n'ayant aucun lien avec celui-ci. Nous observons que les répondants dans cette question, ayant un lien avec la gestion informatique sont plus réaliste par rapport au temps moyen de l'élaboration d'un ERP qui est de 3 mois. Cette information est utile car elle permet de cibler en cas de recours à la publicité le public type se rendant compte de l'utilité technologique de notre produit.

5.7.3 Conclusion générale

Suite à cette analyse approfondie de notre étude de marché nous avons décidé la manière de vendre notre produit. Notre produit sera vendu sous forme de package comprenant la tablette graphique, le logiciel et un manuel d'explication. Ce produit sera donc à destination d'entreprise de développement de logiciel. Ceci va nous permettre de ne pas devoir s'adresser au consommateur final de développement de logiciel. Celui-ci est en effet sensible au caractère nouveau et non fiable d'un nouveau fournisseur d'IT. De plus étant donné que nous sommes spécialisé dans la partie du prototypage nous ne devrons pas nous soucier des autres aspects entrant en compte lors cycle de développement de logiciel.

5.8 Le choix du panier d'attributs

Passons maintenant, après analyse des données au choix et à la vision de notre produit au travers d'un panier d'attributs. L'utilisation du panier d'attributs permet de décider le besoin de base auquel nous décidons de répondre ainsi que des fonctionnalités périphériques ajoutées et nécessaires choisies pour ce dernier. Le panier est graphiquement représenté par un cercle concentrique allant de la fonctionnalité de base situé au centre, aux fonctionnalités périphériques situées dans les cercles excentriques.

5.8.1 le centre du panier

Le centre du panier représente le fonctionnalité de base du produit. Il définit le besoin principal auquel l'entreprise répond en vendant son produit. Le besoin de base auquel nous répondons est le suivant :

Un outil au prototypage d'UI

5.8.2 les fonctionnalités périphériques

Ce sont des attributs nécessaires au produit de base pour qu'il soit vendable. On distingue deux types d'attributs secondaires.

Les fonctionnalités nécessaires : Ce sont des utilités liées aux fonctionnalités de bases. Il s'agit de tout attribut de vente qui est nécessaire pour que la vente du besoin de base puisse être accomplie.

Les fonctionnalités ajoutées : ce sont des utilités qui ne sont pas nécessaires à la vente de la fonctionnalité de base mais qui peuvent être utilisée comme élément de différentiation lors de la vente du produit.

comme expliqué dans la conclusion de l'analyse des données, nous avons décidé de vendre un produit et non un service auprès d'entreprise de développement de logiciel. Ceci limitera notre risque de se lancer sur un marché déjà très prisé comptant 3200 entreprises de développement.Nous proposons dès lors un produit à acheter permettant de faciliter la phase de prototypage nécessaire au développement des UI.

Les fonctionnalités nécessaires pour notre produit sont :

- Une outil de visualisation du prototype qui est la tablette graphique
- Un logiciel d'exploitation pour effectuer le prototypage qui est l'utilisation du logiciel usiSketch

La tablette graphique est nécessaire car sans elle le logiciel usisketch ne peut être utilisé. Le logiciel usiSketch est également nécessaire car sans celui-ci le dessin effectué sur la tablette ne peut être interpreté et utilisé.

Les fonctionnalités ajoutées pour notre produit sont :

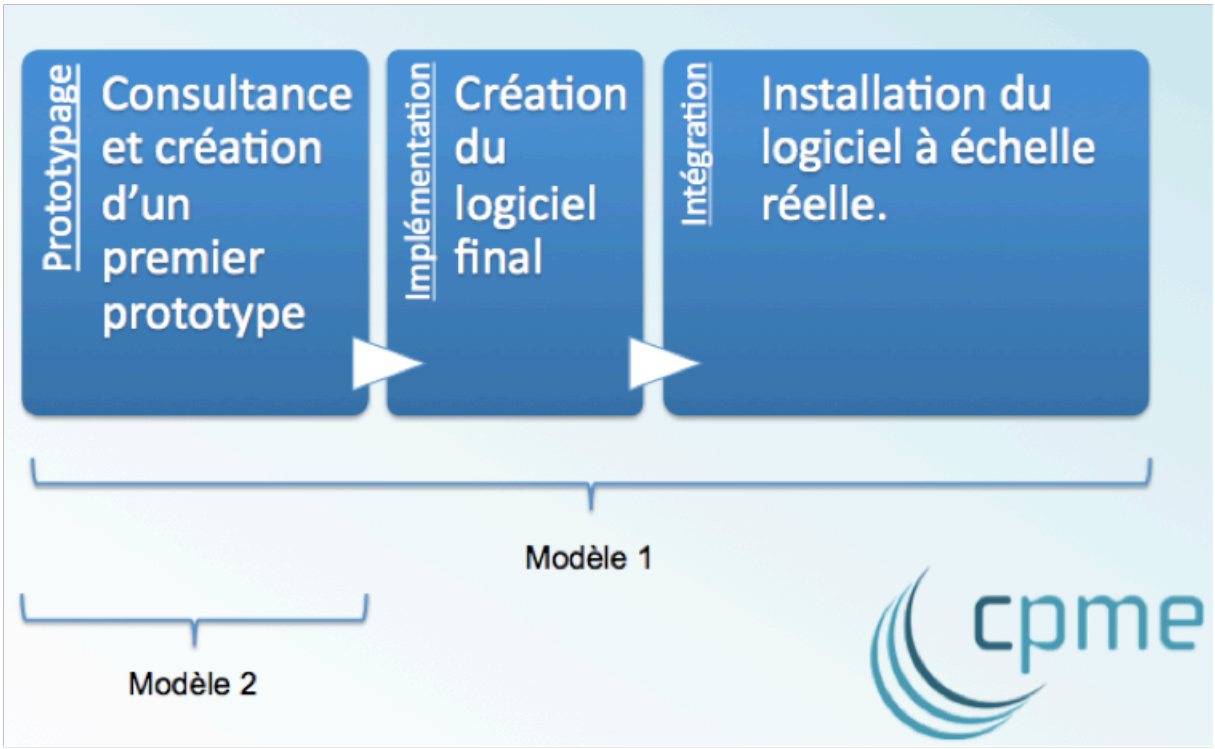
- Un langage informatique qui permet d'exporter le prototype et d'améliorer la rapidité qui est le langage usixml.
- Un service d'éducation au consultant qui rencontre le client final.
- Un site internet permettant une vente en ligne

Le choix d'usixml comme langage informatique améliore la vitesse de développement d'un logiciel vu que l'utilisation de celui-ci permet d'exporter le prototype dessiné et évite donc de devoir écrire la totalité du code. L'outil usisketch combiné à la tablette graphique permet de dessiner un prototype et de le tester sur la tablette graphique. L'utilisation d'usisketch avec le langage usixml permet de rendre utilisable ce qui à été dessiné sur la tablette graphique comme prototype au niveau du code informatique écrit. Cela signifie qu'une fois que le client est fixé sur le prototype dessiné il est possible de transformer grace à usixml ce dessin en langage informatique. Celui-ci sert alors de base pour effectuer les ajouts, tel que le design de l'UI ,des outils de calculs à ajouter dans le logiciel,etc..

5.9 Choix du business model

Dans cette section nous allons expliquer notre choix du business model. Le processus d'élaboration d'une interface graphique se divise en trois partie qui sont les suivantes : Le prototypage, l'implémentation et l'intégration. Ces trois parties peuvent encore être subdivisée en plusieurs sou parties expliques dans le SDLC. Cette subdivision ne nous concerne pas car il n'y a que trois partie qui peuvent être indépendamment effectuée l'une de l'autre.

Ces trois parties ainsi que les deux modèles que nous comptons proposer sont illustrées sur le tableau suivant :



5.9.1 Le prototypage

La première partie, est le prototypage. Elle consiste à concevoir des versions non complètes ou provisoire d'une interface, logiciel ou site internet. Cette partie consiste à consulter le client pour prendre note de ses désidératas et dessiner les premières esquisses du projet. Cette phase résulte généralement de multiples allez retours entre le client et le fournisseur d'IT et donc d'une perte de temps et d'argent. Le prototypage permet

donc de déterminer une esquisse non codée illustrant une première idée de ce que le client recherche.

Lors de cette phase le client détermine les onglets, la structure, le plan et finalement le contenu. L'étape la plus importante dans ce processus est l'établissement d'un prototype de la structure ou du squelette désignant tous les différents cheminements possible lorsqu'une personne naviguera sur l'interface graphique. Que se passe-t-il si on pousse sur cet onglet la ? Comment retourner à la page précédente ? L'étape d'implémentation suit cette phase. La différence entre le prototypage et l'implémentation est le degré de fidélité représenté par ce qui à été crée. Le prototypage consiste à émettre une idée, une esquisse qui permet de simuler le jeux d'interface sans avoir implémenté les détails entourant le jeux d'interfaces. Lors de l'implémentation le code sera tapé et retravaillé afin de rajouter tous les détails nécessaire à la finalisation du logiciel commandé. (Nielsen,J. (1994) Usability Engineering AP Professional, AP cambridge concsulté le 13 Mai 2011) google scholar.

Le prototypage peut être divisé en deux parties complémentaires :

Le prototypage horizontal et le prototypage vertical.

Le prototypage horizontal permet de déterminer les aspects fonctionnels d'une interface. On détermine par exemple, les boutons à mettre l'emplacement de ceux-ci, sans se soucier de ce qu'il se passe quand on pousse sur le bouton et la suite d'événements qui s'en suit.
Le prototypage vertical quant à lui permet de concevoir la suite logique du site internet. On conçois le squelette et la structure du programme ou site web. Que se passe-t-il quand on pousse sur ce bouton ? Cette méthodologie va permettre de déterminer les besoins d'interactions entre les différentes interfaces.

5.9.2 L'implémentation

Une fois le modèle proposé et approuvé par le client, l'implémentation peut être effectuée. L'implémentation permet de coder à un degré de fidélité précis les désidératas des clients. Dans le cadre de l'utilisation d'usiSketch un code usiXML représentant

l'esquisse de l'interface graphique pourra être sorti du prototype crée par usiSketch. Ce code usiXML représentant le squelette de l'interface graphique devra être finalisé et amélioré afin d'ajouter les différents outils et design exigé par le client final. C'est la que se situe la grande plus value de notre produit : Rendre le premier prototype utilisable sous forme de code usiXML qui doit alors uniqement être retouché lors de l'implémentation pour ajouter les options précises.

5.9.3 L'intégration

L'intégration est l'étape finale qui consiste à intégrer le logiciel dans l'entreprise ou sur le net. Il est l'étape finale qui finalise le produit. Une fois intégré le logiciel pourra alors être totalement utilisé.

5.9.4 Notre choix

Nous avons décidé de limiter les risques et d'optimiser nos chance de réussites. Pour cela nous avons décidé de vendre un outil permettant de faciliter le prototypage. Nous avons donc décidé de ne pas créer d'entreprise de développement de logiciel mais une entreprise vendant et développant un outil aidant à la phase de prototypage. En effet notre réelle plus-value se situe dans cette partie du business model type du développement de logiciel. Nous vendrons donc un produit composé d'une tablette graphique, et d'un logiciel d'utilisation. A coté de cella, vu le caractère nouveau du produit nous proposerons des séances de tutoring destinées au consultant rencontrant le client demandeur d'UI afin d'améliorer sa pédagogie lors de la séance de prototypage. Le choix de ce business model est aussi préféré vu le risque limité que nous prenons en se lançant d'abord dans cette section du business model. Si nous voyons que les ventes marchent bien, nous pourrons par la suite, décider de nous lancer à notre propre compte dans le business model complet propre au entreprise de développement. Nous avons donc opté en référence à la figure exprimant les modèle choisi le modèle 2 reprenant uniquement la phase de prototypage.

5.10 Stratégie

Dans cette section nous appliquerons les conclusions tirées lors de l'analyse marketing effectuée précédemment. Nous avons divisé l'élaboration de la stratégie en différentes parties parcourant les différents aspect d'une stratégie.

5.10.1 Le choix d'une couverture de segments

5.10.2 Stratégie de base

Le choix de la stratégie de base détermine, après avoir ciblé le besoin et le business process, la manière dont nous allons conquérir le Marché. Selon le livre (Lambin et de Moerloose, 2008, p.321) deux options de base sont possibles en ce qui concerne le marché à attaquer. La première consiste à attaquer un marché existant et le second à attaquer un marché futur. Dans notre cas, nous devons trouver un produit-marché qui nous convient afin de déterminer les facteurs clés, les forces et les faiblesses et trouver les concurrents les plus dangereux. Nous avons décidé de nous attaquer au marché existant de prototypage d'UI* qui est déjà réparti entre les différents outils cité dans la chapitre 3. Pour bien s'attaquer au marché existant, plusieurs stratégies en lien avec la taille de la cible stratégique, l'avantage concurrentiel et la stratégie de coûts doivent être pris en compte.

Sur la figure 5.7 illustrant la combinaison de la cible stratégique avec l'avantage concurrentiel notre entreprise optera pour une stratégie de concentration avec différentiation du produit. En effet notre produit étant différent des autres produit vu sa technicité, l'utilisation du langage usiXML ainsi que la tablette graphique il répondra différemment au besoin de prototypage d'UI*. Nous n'optons donc pas pour une stratégie de domination par les coûts vu que notre produit à l'achat est plus cher que les autres

check si plus cher ou non avec plan financier.

Dans les conclusions de notre étude de marché nous avons vu que les répondants apportaient beaucoup d'importance aux attributs sur lequel notre produit est fort et sont prêt à payer plus pour les . Il s'agit, de la baisse du nombre de rendez-vous, de la rapidité de mise en ½uvre, du prix du service après vente et de la phase de prototypage. Nous nous différencions sur ces aspects par l'utilisation d'une technologie innovante par l'annexion d'une tablette graphique rendant le dessin de l'UI* possible par l'esquisse.

En ce qui concerne la cible stratégique, nous nous focaliserons comme expliqué lors du ciblage du produit et du choix du business model sur les entreprise de développement à la recherche de nouveaux outils pour améliorer leurs service lors du développement de logiciels et du prototypage d'UI*. Notre cible ne sera donc pas, comme cité précédemment toute entreprise/organisation cherchant à développer un site ou une UI*, mais bien toutes les entreprises de développement qui ont déjà leur carnet d'adresse, leurs clientèle et qui auront, vu leur degré de fiabilité plus de facilité à faire passer une nouvelle technologique pour le consommateur final. Notre cible stratégique est donc de type étroite.

5.10.3 Stratégie de croissance

Comme cité précédemment, nous nous attaquerons en premier temps aux entreprises de développement qui disposent de leurs clientèle et qui ont une bonne connaissance du marché du développement d'UI*. Si nous arrivons à bien vendre notre produit et à atteindre nos objectifs de vente nous opterons donc pour une stratégie de croissance intensive suivie d'une stratégie intégrative.

On parle de croissance intensive si l'objectif de croissance se situe au sein du marché de référence dans lequel on opère(Lambin et de Moerloose, 2008, pg.327)

On parle de stratégie de croissance intégrative lorsque l'objectif de croissance se situe au sein de la même filière industrielle, par une extension latéral, en amont ou en aval de son activité de base. (Lambin et de Moerloose, 2008, pg.327).
Pourquoi choisir une telle stratégie de croissance ?

Le choix d'une stratégie de croissance intensive

Étant donné que notre outil a un avantage compétitif fort en comparaison avec les autres outils disponible sur le marché visé nous avons décidé d'opter pour la croissance intensive par la croissance des marchés afin de le développer cet avantage un maximum. Cette objectif de croissance sera exercée sur le marché Belge de 3200 entreprises de développement de logiciel avant de s'étendre sur le marché international. Le développement de notre marché à l'échelle internationale est facilité car notre outil est utilisable sous différentes langues (Français, Anglais, Néerlandais).

citation pages dor.

Le choix d'une stratégie de croissance intégrative

Ce choix est le résultat du temps et de l'effort consacré à développer cette technologie. De plus vu que nous nous lançons au départ dans le domaine du développement de logiciel nous apprendrons à connaître le métier et ses difficultés. Notre stratégie de croissance intégrative sera de ce fait effectuée en amont et en aval. Nous prendrons donc en charge la gestion des clients et le suivi de ceci ainsi que toute la partie en aval qui concerne l'implémentation et l'intégration du logiciel développé. La raison pour laquelle nous nous lançons pas tout de suite sur le marché des logiciels de développement est que comme reflété par notre enquête nous pouvons observer que le répondant est très sensible au caractère fiable d'une entreprise.

5.10.4 Stratégie concurrentielles

Pour ce qui est de la stratégie concurrentielle que nous avons décidé d'adopter il s'agit de la stratégie du Spécialiste. La raison pour laquelle nous avons décidé d'adopter cette stratégie est simple. Comme expliqué nous nous situerons uniquement sur le segment du prototypage. Les concurrents utilisent d'autres techniques et n'utilisent aucunement une tablette graphique. Notre entreprise à pour but de se situer sur un segment précis ou nous développons une technologies nouvelle

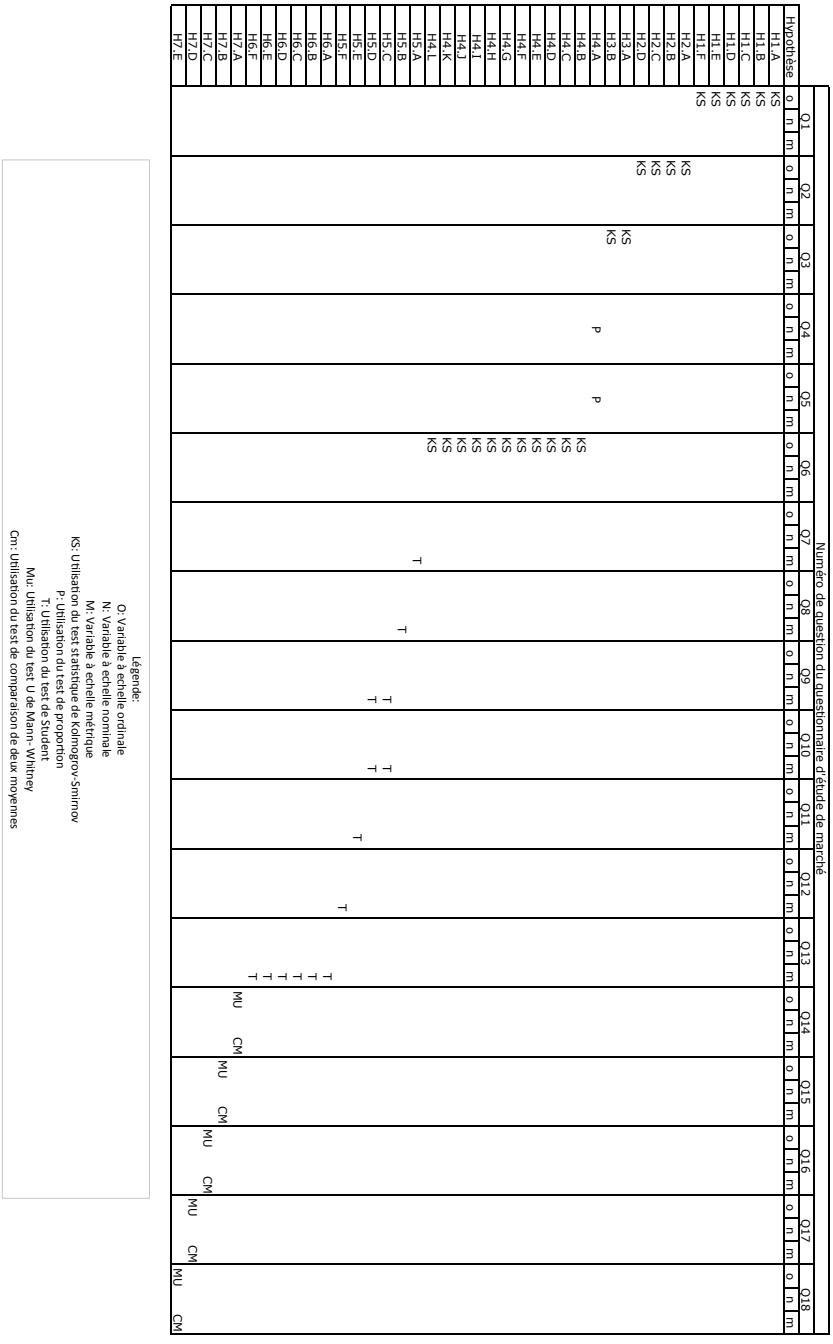


FIGURE 5.7 – Tableau récapitulatif des Hypothèses testée sous formes de questions dans l'étude de marché

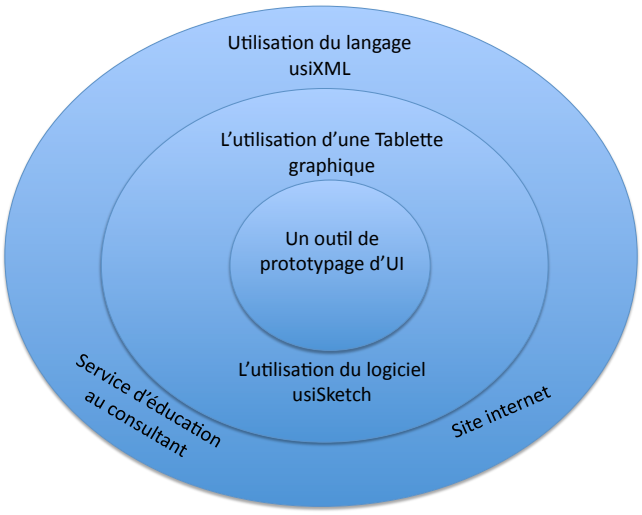


FIGURE 5.8 – Panier d'attribut du produit

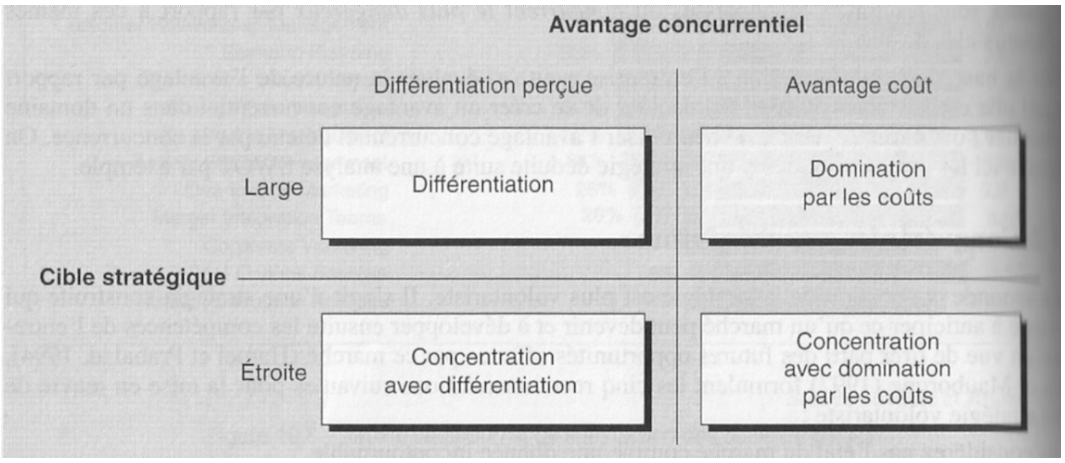


FIGURE 5.9 – Stratégie de base ; source : (Lambin et de Moerloose, 2008, p. 322)

Chapitre 6

Questions juridiques

Dans ce chapitre, nous traiterons de certaines questions juridiques importantes pour la création d'une entreprise en Technologie de l'Information (IT)*.
Notre équipe ne contenant pas d'étudiant en droit, nous nous sommes concentrés sur deux questions importantes : Comment protéger le produit, et quels librairies et logiciels externes sont utilisables vis-à-vis de la solution de protection retenue.

Pour répondre à ces deux questions, nous avons rencontré plusieurs experts en droit de l'informatique et de la propriété intellectuelle. Parmi eux :

- Mme Poupaert : Conseillère juridique à l'UCL,
- M Laurent : avocat au barreau de Bruxelles (Marx Van Ranst Vermeersch & Partners) spécialisé dans le droit des logiciels open source*,
- M Remiche : Président du Département de Droit Economique et Social (DESO) de la Faculté de Droit de l'UCL.

Pour une étude juridique plus complète, nous invitons le lecteur à se référer au mémoire de Firket et Stoefs (2008).

6.1 Comment se protéger

La protection du logiciel contre le piratage et l'espionnage industriel est un point important, en particulier lorsque le logiciel est innovant.

6.1.1 La propriété intellectuelle

Tout logiciel est par défaut protégé par la propriété intellectuelle, pour peu qu'il soit original. Cette propriété intellectuelle protège d'emblée contre l'espionnage industriel. Par contre, l'utilisation d'un logiciel ou la copie de celui-ci (par une réécriture complète) n'est pas protégé par cette dernière. Le licence d'utilisation permet de se protéger contre l'utilisation abusive (piratage, intégration dans d'autre logiciels...), alors que le brevet permet de se protéger contre la copie par réécriture.

6.1.2 Les brevets

Le brevet permet de protéger une solution technique à un problème technique. Dans le cas du développement logiciel, la jurisprudence est très floue et très complexe. En effet, nos interlocuteurs nous ont mentionnés qu'il était difficile d'obtenir un brevet pour un logiciel, car il n'est pas considéré tel quel comme une solution technique. Néanmoins, il est possible d'obtenir un brevet pour un logiciel, mais la demande de ce dernier requiert des compétences en droit très pointues. De plus, peu de logiciel sont effectivement protégés par un brevet, car cela n'est pas fondamentalement nécessaire. Aussi, nous n'avons pas exploré cette solution davantage.

6.1.3 Les licences d'utilisation

La licence d'utilisation spécifie les droits qui sont cédés à l'utilisateur lors de l'obtention du logiciel. Le choix de celle-ci est sujet complexe. En effet, selon la licence choisie, les droits cédés vont différer. Les trois principales options sont :

1. Distribuer usiSketch sous une licence propriétaire,
2. Distribuer usiSketch sous une licence open source*,
3. Ne pas distribuer usiSketch du tout, et l'utiliser uniquement en interne (protection par le secret).

Étant donné le business model choisi pour valoriser usiSketch (voir section 5.9), la troisième option n'est pas possible. Un business model de type service pourrait permettre que le code critique du logiciel puisse être protégé par le secret. Cette solution demanderait une réécriture partielle du logiciel, de sorte de le code critique soit hébergé sur un serveur. Nous n'avons pas retenu cette solution, compliquée à implémenter.

La licence open source* pose comme seule et unique contrainte la diffusion complète du code source du logiciel. Son principal avantage est que le développement d'usiSketch pourra être fait par des personnes tierces, permettant ainsi de l'accélérer à des frais virtuellement nuls. Cependant, pour atteindre cet état de choses, il faut une communauté de développeurs prêts à travailler sur le code d'usiSketch. De par la nature innovante de notre logiciel, cette communauté n'existe pas encore. Distribuer le logiciel sous cette licence n'apportera donc que des inconvénients sur le court terme, mais pourrait être envisagée sur le moyen terme, si le logiciel a suffisamment de succès.

La solution la plus intéressante est la licence propriétaire. Ses principaux avantages sont la protection contre la copie illégale, l'interdiction de modifier le logiciel sans une permission préalable, et la possibilité de ne pas divulguer le code source du logiciel. Comme pour les brevets, la rédaction de celle-ci est complexe est requiert des compétences en droits que nous ne possédons pas. Nous avons donc simplement choisi de distribuer usiSketch sous licence propriétaire, mais celle-ci reste encore à être rédigée.

6.2 Logiciels externes

Nous venons de voir qu'usiSketch sera distribué sous une licence propriétaire. Par conséquent, certaines précautions doivent être prises concernant les librairies* sous licence open source* utilisées.

UsiSketch plusieurs librairies*. Certaines de ces licences sont dites "fortement copyleft", c'est-à-dire qu'un maximum de droits sont cédés à l'utilisateur (dont celui de modifier et redistribuer), mais que toute extension, modification, ou intégration du logiciel dans un autre implique qu'il doit être distribué sous cette même licence, ou une licence cédant encore plus de droits. Par conséquent, un logiciel utilisant une librairie* sous licence fortement copyleft est lui aussi tenu d'être open source*. Ce n'est cependant pas le cas de toutes les licences open source*. Nous avons identifié les licences fortement copyleft les plus répandues afin de ne pas s'en servir durant le développement.

D'après l'annexe D, qui compare les six licences open source* les plus populaires et répond à cinq questions fréquentes, la licence la plus permissive est la licence Apache, qui n'est pas copyleft, et qui permet de distribuer un logiciel sous n'importe quelle autre licence. *A contrario*, les licences GPL et LGPL sont trop peu permissives et fortement copyleft. Les licence BSD et MPL permettent aussi de distribuer un logiciel sous une autre licence.

Nous avons posé comme contrainte lors du développement que les librairies* open source* utilisées devront être des librairies* sous licence Apache, BSD, MPL ou toute autre licence suffisamment permissive pour pouvoir redistribuer notre logiciel sous n'importe quelle autre licence. Ce choix nous interdit donc l'utilisation de librairie* sous licence GPL ou LGPL, quelle que soit sa version.

Une copie de la licence Apache se trouve à l'annexe C. Une liste exhaustive des licences open source* reconnues peut être trouvée à l'adresse suivante :
http ://opensource.org/licenses/category

Compatibilité avec la licence d'eclipse

Chapitre 7

Conclusion

conclusion

Troisième partie

Plan d'Affaires

Quatrième partie

Conclusion générale

Bibliographie

ABIRESEARCH.COM (2011). Apple ipad held 85 http://www.abiresearch.com/press/3665-Apple+iPad+Held+85%25+Media+Tablet+Market+Share+in+2010 (date d'accès : 10/05/11).

AKTOUF et OMAR (1987). Méthodologie des sciences sociales et approche qualitative des organisations.

AMBLER, S. W. (2009). Introduction to User Interface Prototypes. http://www.agilemodeling.com/artifacts/uiPrototype.htm (date d'accès : 28/05/11).

BAILEY, B. P. et KONSTAN, J. A. (2003). Are informal tools better ? : comparing demais, pencil and paper, and authorware for early multimedia design. In Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '03, pages 313–320, New York, NY, USA. ACM.

BEUVENS, F. et DULLIER, T. (2008). Vers une plate-forme multi-algorithmes de reconnaissance de gestes : conception, developpement et experimentation.

COYETTE, A., SCHIMKE, S., VANDERDONCKT, J. et VIELHAUER, C. (2007). Trainable sketch recognizer for graphical user interface design. In BARANAUSKAS, M. C. C., PALANQUE, P. A., ABASCAL, J. et BARBOSA, S. D. J., éditeurs : INTERACT (1), volume 4662 de Lecture Notes in Computer Science, pages 124–135. Springer.

FIRKET, F. et STOEFS, J. (2008). Commercialisation d'outils de développement d'interfaces homme-machine et d'évaluation ergonomique.

GARCÍA, J. G., GONZÁLEZ-CALLEROS, J. M., VANDERDONCKT, J. et ARTEAGA, J. M. (2009). A theoretical survey of user interface description languages : Preliminary results. In CHÁVEZ, E. et abd ALBERTO L. MORÁN, E. F., éditeurs : LA-WEB/CLIHC, pages 36–43. IEEE Computer Society.

HALL, A. et CHAPMAN, R. (2002). Correctness by construction :developing a commercial secure system. IEEE Software.

ITRNEWS.COM (2008). La croissance du marché européen des TIC devrait ralentir en 2008. http://www.itrnews.com/articles/77793/croissance-marche-europeen-tic-devrait-ralentir-2008.html (date d'accès : 10/05/11).

KING, T. et MARASCO, J. (2008). What is the cost of a requirement error ? stickyminds.com. http://www.stickyminds.com/sitewide.asp?ObjectId=12529&Function=edetail&ObjectType=ART (visité le 29/05/11).

LAMBIN, J.-J. et de MOERLOOSE, C. (2008). Marketing stratégique et opérationnel (French edition). Dunod, 7e édition édition.

LANDAY, J. A. et MYERS, B. A. (2001). Sketching interfaces : Toward more human interface design. pages 56–64.

MALHOTRA, N., DÉCAUDIN, J.-M. et BOUGUERRA, A. (2007). Etudes marketing avec SPSS. Pearson education, Paris, 4e édition édition.

NEWMAN, M. W., LIN, J., HONG, J. I. et LANDAY, J. A. (2003). Denim : an informal web site design tool inspired by observations of practice. Hum.-Comput. Interact., 18:259–324.

OLIVIER BOURDOUX, U. S. (2011). Eclipse Sketch Plugin - usiSketch version. https://github.com/xurei/sketch/tree/usisketch (date d'accès : 23/06/11).

RUBINE, D. (1991). Specifying gestures by example. SIGGRAPH Computer Graphics, pages 329–337.

SANGIORGI, U. (2010a). Eclipse Sketch Plugin. http://www.eclipse.org/sketch/ (date d'accès : 18/04/11).

SANGIORGI, U. (2010b). Using string distance to compare sketches. http://ugosan.org/using-string-distance-to-compare-sketches/ (date d'accès : 22/02/11).

US DEPARTMENT OF JUSTICE (2003). DOJ Systems Development Life Cycle Guidance, chapter 1. http://www.justice.gov/jmd/irm/lifecycle/ch1.htm (date d'accès : 15/05/11).

USIXML.ORG (2010). USIXML. USer Interface eXtensible Markup Language. http://www.usixml.org (date d'accès : 22/11/10).

VOLERE (2010). Requirements specifications template. http://www.volere.co.uk/template.htm (date d'accès : 02/03/11).

Wikipédia (2011a). Interaction homme-machine. http://fr.wikipedia.org/wiki/Interaction_Homme-Machine (date d'accès : 11/05/11).

Wikipédia (2011b). Programmation par contraintes. http://fr.wikipedia.org/wiki/Programmation_par_contraintes (date d'accès : 25/05/11).

Cinquième partie

Annexes

Annexe A

Liste des requirements

compléter et structurer les requirements techniques ; en rouge veut dire à compléter

Cette liste regroupe l'essentiel des requirements définis pour usiSketch. Elle se veut exhaustive pour les priorités moyenne et haute. Les requirements de faible priorité ne sont pas tous présents.

La structuration des requirements est la même que celle du section 4.1

Reconnaissance

Numéro	1	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir reconnaître une forme géométrique dessinée et la convertir sous une forme lissée. Les formes à reconnaître sont : – ligne – ligne en vaguelettes (wavy line) – triangle – rectangle – ellipse – croix		
Motivation	Une forme lissée a un plus haut niveau de fidélité* que son équivalent dessiné à main levée. C'est aussi nécessaire pour la combinaison de formes.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	2	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir combiner plusieurs formes simples en une forme plus complexe ou un widget, selon des règles pré-établies. A chaque fois qu'une nouvelle forme est ajoutée dans une fenêtre, le logiciel doit vérifier s'il est possible de la combiner avec d'autres formes.		
Motivation	Permet de limiter l'ensemble des formes à reconnaître, donc à minimiser l'apprentissage pour la reconnaissance.		
Scénario	Durant le dessin de l'interface, lorsqu'une nouvelle forme est reconnue		
Bénéficiaire	Dessinateur		
Prérequis	requirement 1		

Numéro	3	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	En cas d'erreur de reconnaissance, le dessinateur doit pouvoir indiquer au logiciel qu'il s'est trompé.		
Motivation	Permet à l'algorithme de reconnaissance d'apprendre de ses erreurs et de s'adapter à son utilisateur.		
Scénario	Utilisateur dont le set d'entraînement de base n'est pas adéquat		
Bénéficiaire	Dessinateur		
Prérequis	requirement 1		

Numéro	4	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Si deux combinaisons mutuellement exclusives sont possibles sur base d'une liste de formes, la combinaison la plus contrainte doit être choisie.		
Motivation	Pour décider quelle est la combinaison à choisir, et avoir le même comportement à tout moment. La combinaison la plus contrainte étant par définition plus difficile à valider, c'est elle qui doit avoir la priorité.		
Scénario	Durant le dessin de l'interface, lorsqu'une nouvelle forme est reconnue et que plusieurs combinaisons sont possibles mais s'excluent mutuellement		
Bénéficiaire	Dessinateur		
Prérequis	requirement 2		

Numéro	5	**Priorité**	Moyen
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Les règles de composition doivent être spécifiées en dehors du code logiciel.		
Motivation	Permet de définir de nouvelles règles sans toucher au code source ; permet une grande flexibilité de l'outil dans la définition des compositions ; évite au dessinateur de devoir s'adapter à des règles de composition qui ne lui conviennent pas.		
Scénario	Nouveau widget à définir ; changement d'une règle de composition plus claire pour le dessinateur		
Bénéficiaire	Dessinateur		
Prérequis	requirement 2		

Numéro	6	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le dessinateur doit pouvoir spécifier les règles de composition à l'aide d'une interface simple et intuitive.		
Motivation	Faciliter l'édition des règles de composition par le dessinateur, même sans compétence en programmation.		
Scénario	Le dessinateur veut ajouter ou modifier une règle de composition		
Bénéficiaire	Dessinateur		
Prérequis	requirement 5		

Numéro	7	**Priorité**	Bas
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir importer une image de prototype basse fidélité dans projet usiSketch.		
Motivation	Permet de convertir un prototype papier scanné, ou une image dessinée avec un logiciel de dessin		
Scénario	Le dessinateur a déjà un prototype papier et veut l'utiliser dans usiSketch		
Bénéficiaire	Dessinateur		
Prérequis			

Dessin-Rendu

Numéro	8	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit être WYSIWYG*.		
Motivation	Le logiciel se veut intuitif, donc la vue doit toujours être en adéquation avec le modèle en cours de dessin.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	9	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit disposer d'un mode de dessin par reconnaissance. Après chaque dessin, le logiciel doit l'analyser et retourner la forme vectorielle reconnue.		
Motivation	Certains widgets sont uniques et spécifiques à une interface, donc tous les représenter est impossible. Il faut donc laisser un espace plus libre pour le dessinateur. Le testeur peut aussi vouloir donner un feedback visuel, via les annotations.		
Scénario	Durant le dessin de l'interface et son test		
Bénéficiaire	Dessinateur		
Prérequis	requirement 1		

Numéro	10	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système de reconnaissance de forme doit pouvoir gérer le multi-traits.		
Motivation	L'expérience utilisateur recherchée se doit d'être la plus naturelle possible. Or, beaucoup de personnes dessinent certaines formes avec plusieurs traits (l'exemple le plus frappant est sans doute la croix). Il faut donc que l'algorithme de reconnaissance puisse être capable de le gérer.		
Scénario	Durant le dessin de l'interface et son test		
Bénéficiaire	Dessinateur		
Prérequis	requirement 1		

Numéro	11	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit disposer d'un mode annotation, où aucune reconnaissance n'est faite.		
Motivation	Certains widgets sont uniques et spécifiques à une interface, donc tous les représenter est impossible. Il faut donc laisser un espace plus libre pour le dessinateur. Le testeur peut aussi vouloir donner un feedback visuel, via les annotations.		
Scénario	Durant le dessin de l'interface et son test		
Bénéficiaire	Dessinateur, Testeur		
Prérequis			

Numéro	12	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit permettre l'effacement de formes préalablement dessinées.		
Motivation	Le dessinateur doit pouvoir effacer une forme placée par erreur ou devenue obsolète.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	13	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit disposer d'un moyen de sélectionner un élément ou groupe d'éléments graphiques.		
Motivation	Sélectionner un ensemble de widgets est nécessaire pour pouvoir les déplacer, les copier ou les effacer.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	14	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Il doit être possible de déplacer une sélection.		
Motivation	Pour aligner les widgets, ou les déplacer vers une zone différente de la fenêtre sans avoir à les redessiner.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis	requirement 13		

Numéro	15	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit disposer d'un moyen de copier/couper/coller une sélection.		
Motivation	Pour des raisons évidentes de facilitation du travail : si le dessinateur veut déplacer un widget ou ensemble de widget, il faut que cela lui soit possible sans avoir à effacer puis redessiner l'ensemble.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur		
Prérequis	requirement 13		

Numéro	16	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit permettre plusieurs niveaux de fidélité*, changeables à tout moment de la conception. – fidélité dessin : les dessins sont affichés tels qu'il ont été dessinés, rien d'autre – fidélité formes : les formes reconnues sont affichées lissées, les widgets ne sont pas affichés – fidélité widgets : les widgets sont affichés sous forme d'images représentatives, les formes restantes sous forme lissée – fidélité simulation : les widgets sont affichés sous forme de widgets réels, les formes restantes sous forme lissée		
Motivation	Un niveau de fidélité plus bas est plus agréable durant le dessin. A contrario, un niveau de fidélité haut est plus agréable durant la création des interactions entre fenêtres/widgets.		
Scénario	Durant le dessin de l'interface		
Bénéficiaire	Dessinateur, Concepteur, Testeur		
Prérequis			

Navigation

Numéro	17	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Des actions typiques sur un widget* ou une fenêtre (show, hide, minimize, maximize, setText, ...) doivent pouvoir être définies.		
Motivation	Permettre une simulabilité plus fidèle qu'un simple affichage des fenêtres de façon statique.		
Scénario	Conception de l'interface		
Bénéficiaire	Concepteur, Testeur		
Prérequis			

Numéro	18	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Les événements typiques pour un widget* (onclick, onchange, ...) doivent pouvoir être définis.		
Motivation	Nécessaire comme point d'entrée pour effectuer certaines actions d'un widget* (le même ou un autre).		
Scénario	Conception des interactions entre widgets, simulation du prototype		
Bénéficiaire	Concepteur, Testeur		
Prérequis	requirement 17		

Numéro	19	**Priorité**	Bas
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Si présent, le texte contenu dans les widgets doit pouvoir être défini, idéalement par reconnaissance de texte.		
Motivation	Augmenter la fidélité du prototype.		
Scénario	Conception de l'interface		
Bénéficiaire	Concepteur, Testeur		
Prérequis			

Numéro	20	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le système doit être capable de structurer les éléments graphiques sous la forme d'un arbre. Autrement dit, les widgets doivent pouvoir être ajoutés dans un widget de type conteneur, automatiquement ou manuellement.		
Motivation	Permettre d'avoir un prototype structuré, où les conteneurs contiennent réellement leurs widgets, plutôt qu'une vue "plate", ou les conteneurs n'ont qu'une fonction visuelle. Cela réduit le temps de travail du concepteur lors de l'utilisation d'un projet exporté en usiXML : les widgets sont déjà structurés, il ne doit plus le faire		
Scénario	Conception de l'interface		
Bénéficiaire	Concepteur, Testeur		
Prérequis			

Données

Numéro	21	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir exporter un projet usiSketch en fichier usiXML.		
Motivation	L'interface dessinée doit être exportée sous un format standardisé, pour permettre d'éditer le résultat avec un éditeur de plus haute fidélité, choisi par le concepteur.		
Scénario	A la fin du prototypage via usiSketch		
Bénéficiaire	Concepteur		
Prérequis			

Numéro	22	**Priorité**	Bas
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir exporter un projet usiSketch en fichier autre qu'usiXML.		
Motivation	Toucher plus de marché, en n'obligeant pas le client à utiliser usiXML s'il ne le veut pas. Ce requirement est de priorité plus basse, car l'exportation sous plusieurs format est moins importante que sous au moins un format.		
Scénario	A la fin du prototypage via usiSketch		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	23	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir sauver/charger un prototype sous un format de fichier propre à usiSketch.		
Motivation	Certaines informations peuvent être pertinentes uniquement pour usiSketch en interne, mais inutiles lors de son exportation. Un format standardisé ne pourra probablement pas sauver ce type de données.		
Scénario			
Bénéficiaire	Tous		
Prérequis			

Numéro	24	**Priorité**	Bas
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit pouvoir importer un fichier usiXML et le convertir en projet usiSketch.		
Motivation	Il peut être pratique de recycler une ancienne interface et de la modifier, ou de simplement l'annoter. Cela évite à un dessinateur de redessiner une interface complexe si une interface similaire a déjà été définie.		
Scénario	Lorsqu'une partie du prototype a été définie via un autre logiciel compatible usiXML, ou lorsque son fichier source au format usiSketch n'est pas disponible		
Bénéficiaire	Dessinateur, Concepteur		
Prérequis			

Simulation

Numéro	25	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Une interface définie dans usiSketch doit pouvoir y être simulée sommairement.		
Motivation	Durant la conception, le dessinateur tire avantage de pouvoir simuler son interface sans avoir à l'exporter en usiXML à chaque fois.		
Scénario	Durant la conception d'interface, lorsque le dessinateur veut tester son interface.		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	26	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Une interface définie sous usiSketch doit pouvoir être envoyée aux testeurs pour demande de simulation et de feedback.		
Motivation	Pour permettre la simulation online, il faut que le concepteur valide son prototype et l'envoie pour simulation et demande de feedback		
Scénario	Durant la conception d'interface, lorsque le concepteur veut un avis sur le prototype en cours de conception.		
Bénéficiaire	Concepteur, Testeur		
Prérequis			

Numéro	27	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Chaque testeur doit pouvoir donner un ou plusieurs feedbacks lorsqu'il teste un prototype.		
Motivation			
Scénario	Lors du test online d'un prototype		
Bénéficiaire	Testeur		
Prérequis	requirement 28		

Numéro	28	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le dessinateur et le concepteur doivent avoir accès aux feedbacks fournis par le(s) testeur(s).		
Motivation	Les feedbacks fournis servent à améliorer le prototype. Il faut donc que les concepteurs puissent y avoir accès		
Scénario	Lors du test online d'un prototype		
Bénéficiaire	Testeur		
Prérequis			

Ergonomie

Numéro	29	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le temps d'apprentissage de l'utilisation du logiciel doit être le plus rapide possible.		
Motivation	Le logiciel voulant être en compétition avec l'approche papier et les logiciels de dessin simple, il faut qu'usiSketch soit aussi simple à apprendre et à utiliser que ces derniers.		
Scénario	Premier contact avec le logiciel		
Bénéficiaire	Desinnateur		
Prérequis			

Numéro	30	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Un maximum d'opérations de prototypage doivent être accessible via le pointeur.		
Motivation	Le logiciel étant destiné à être utilisé avec une tablette graphique, il faut limiter les échanges stylet-clavier ou stylet-souris.		
Scénario	Durant le dessin d'une interface		
Bénéficiaire	Desinnateur		
Prérequis			

Numéro	31	**Priorité**	Moyen
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Les boutons des barres d'outils doivent être suffisamment grands pour être faciles à cliquer. Une taille de bouton de 50x50 pixels est une taille acceptable.		
Motivation	Des boutons de taille inférieure à 50x50 pixels sont désagréables à l'utilisation avec un stylet.		
Scénario	Durant le dessin d'une interface		
Bénéficiaire	Dessinateur		
Prérequis	requirement 30		

Numéro	32	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit conserver un historique de conception et de dessin d'un prototype à tout moment.		
Motivation	Pour permettre de voir l'évolution d'une interface, et aussi pour permettre le retour arrière.		
Scénario	Durant la description de l'interface		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	33	**Priorité**	Haut
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le dessinateur doit pouvoir annuler/refaire une action contenue dans l'historique de conception du prototype.		
Motivation	La même que pour tout logiciel de création de contenu : parce que c'est extrêmement pratique. Dans ce cas précis, cela permet de modifier une interface sans avoir à se soucier si le résultat obtenu après modification est accepté ou non : s'il ne l'est pas, il suffit de revenir en arrière.		
Scénario	Durant la description de l'interface, si le dessinateur veut revenir en arrière		
Bénéficiaire	Dessinateur		
Prérequis	requirement 32		

Numéro	34	**Priorité**	Moyen
Type	Fonctionnel	**Responsable**	Olivier Bourdoux
Description	Un prototype sauvé sous format usiSketch doit contenir son historique de conception et de dessin, possiblement compressé.		
Motivation	Conserver cet historique permet de revenir en arrière si nécessaire, même après fermeture. De plus, il assure une bonne reconstruction de l'interface. Il peut être compressé pour limiter la taille du prototype et éviter que sa reconstruction soit trop longue.		
Scénario	Durant la description de l'interface		
Bénéficiaire	Dessinateur		
Prérequis	requirement 23,requirement 32		

Architecture

Numéro	35	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel de dessin/conception est développé sous la forme d'un plugin eclipse.		
Motivation	Eclipse est un outil bien connu des développeurs, ce qui facilite l'apprentissage d'usiSketch. De plus, cet outil dispose déjà de vues utiles pour usiSketch. Le développement s'en voit accéléré.		
Scénario	Changement de l'algorithme de reconnaissance		
Bénéficiaire	Dessinateur		
Prérequis			

Numéro	36	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel doit être utilisable quel que soit l'OS* utilisé.		
Motivation	S'assurer que le logiciel puisse fonctionner sur n'importe quel ordinateur évite de devoir se cantonner à un OS* précis, et donc de perdre d'éventuelles parts de marché.		
Scénario	Utilisation sur différents OS*		
Bénéficiaire	Tous		
Prérequis			

Numéro	37	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	L'outil de dessin/conception et l'outil de simulation doivent être construits comme deux programmes utilisables séparéments.		
Motivation	Le testeur n'a pas le droit de modifier l'interface, il n'a donc pas besoin de l'outil de dessin, mais uniquement d'avoir accès à l'interface et la simuler.		
Scénario	Utilisation collaborative		
Bénéficiaire	N/D		
Prérequis			

Numéro	38	**Priorité**	Haut
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel ne doit pas intégrer de code ou de librairie sous licence GPL ou LGPL.		
Motivation	Les licences GPL ou LGPL sont fortement copyleft, ce qui implique que leur intégration demande que le logiciel soit distribué sous la même licence, ce qui va à l'encontre des décision prises au chapitre 6		
Scénario	Distribution du logiciel sous licence propriétaire		
Bénéficiaire	N/D		
Prérequis			

Numéro	39	**Priorité**	Moyen
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	L'architecture doit être conçue pour faciliter l'ajout de nouveaux type de widgets.		
Motivation	De nouveaux widgets pourraient être définis à l'avenir, aussi faut-il prévoir leur ajout d'avance.		
Scénario	Si un nouveau widget apparaît et est largement utilisé		
Bénéficiaire	Tous		
Prérequis			

Numéro	40	**Priorité**	Moyen
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le module de reconnaissance doit pouvoir être facilement remplaçable par un module implémentant un autre algorithme.		
Motivation	La reconnaissance de forme étant un champs de recherche encore très actif, il est possible qu'une méthode plus efficace soit trouvée dans le futur. Il faut donc prévoir une éventuelle migration vers cette méthode à moindre coûts. Un module facilement remplaçable permet aussi de tester différents algorithmes plus aisément.		
Scénario	Changement de l'algorithme de reconnaissance		
Bénéficiaire	N/D		
Prérequis			

Numéro	41	**Priorité**	Moyen
Type	Non-fonctionnel	**Responsable**	Olivier Bourdoux
Description	Le logiciel de test devrait être accessible en ligne, et idéalement sans besoin d'installation (typiquement : une interface web).		
Motivation	Le testeur est généralement une personne pas ou peu compétente en informatique. Il faut donc lui faciliter la vie au maximum, en lui évitant toute installation d'outil, et en utilisant un mode d'utilisation déjà connu.		
Scénario	Utilisation du logiciel de test		
Bénéficiaire	Testeur		
Prérequis			

Annexe B

Génération de la librairie usiXML Java

Vous trouverez dans cette annexe le script qui a été utilisé pour générer la librairie usiXML, sur base de ses spécifications. Le résultat final est un fichier jar contenant la librairie et

B.1 Fichier de bindings

Ce fichier permet d'éviter les collisions entre certains tags. Ce fichier doit être nommé $UsiXML_bind.xml$ et placé dans le même dossier que le script de génération.

```
<binding xmlns="http://www.castor.org/SourceGenerator/Binding"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.castor.org/SourceGenerator/Binding
      C:\\Castor\\xsd\\binding.xsd" defaultBinding="element">

   <elementBinding name="/menuItem">
      <java-class name="MenuItemElement" />
   </elementBinding>

   <elementBinding
      name="/uiModel{http://www.usixml.org}/developmentLibrary/pathLibrary/developmentStep">
      <java-class name="DevelopmentLibrary.DevelopmentStep" />
   </elementBinding>
   <elementBinding
      name="/uiModel{http://www.usixml.org}/transformationModel/developmentPath/
developmentStep">
      <java-class name="TransformationModel.DevelopmentStep" />
   </elementBinding>

   <elementBinding
      name="/uiModel{http://www.usixml.org}/developmentLibrary/pathLibrary/
developmentSubStep">
      <java-class name="DevelopmentLibrary.DevelopmentSubStep" />
   </elementBinding>
   <elementBinding
      name="/uiModel{http://www.usixml.org}/transformationModel/developmentPath/
developmentSubStep">
      <java-class name="TransformationModel.DevelopmentSubStep" />
   </elementBinding>

   <elementBinding
      name="/uiModel{http://www.usixml.org}/developmentLibrary/pathLibrary/developmentPath">
      <java-class name="DevelopmentLibrary.DevelopmentPath" />
   </elementBinding>
   <elementBinding
      name="/uiModel{http://www.usixml.org}/transformationModel/developmentPath/
developmentPath">
      <java-class name="TransformationModel.DevelopmentPath" />
   </elementBinding>
```

```
   <elementBinding name="/declaration{http://www.usixml.org}/method">
      <java-class name="Declaration.Method" />
   </elementBinding>
   <elementBinding name="/domainClass{http://www.usixml.org}/method">
      <java-class name="DomainClass.Method" />
   </elementBinding>

   <elementBinding
      name="/uiModel{http://www.usixml.org}/developmentLibrary/pathLibrary/developmentStep/
developmentSubStep">
      <java-class name="DevelopmentLibrary.DevelopmentStep.DevelopmentSubStep" />
   </elementBinding>
   <elementBinding
      name="/uiModel{http://www.usixml.org}/transformationModel/developmentPath/
developmentStep/developmentSubStep">
      <java-class name="TransformationModel.DevelopmentStep.DevelopmentSubStep" />
   </elementBinding>

   <elementBinding
      name="/uiModel{http://www.usixml.org}/developmentLibrary/pathLibrary/developmentStep/
developmentSubStep">
      <java-class name="DevelopmentLibrary.DevelopmentStep.DevelopmentSubStep" />
   </elementBinding>
   <elementBinding
      name="/uiModel{http://www.usixml.org}/transformationModel/developmentPath/
developmentStep/developmentSubStep">
      <java-class name="TransformationModel.DevelopmentStep.DevelopmentSubStep" />
   </elementBinding>

</binding>
```

B.2 Script de génération

Ce script utilise le générateur de code de Castor pour générer les classes Java responsables de la liaison XML-Java. C'est un fichier bash nommé *buildusiXMLparser.sh*.

```
#!/bin/bash

#Ce script génère la librairie usiXML sur grâce au générateur Castor
#Auteur : Olivier Bourdoux <olivier.bourdoux@gmail.com>

#Paramètres : changer ces variables selon la configuration
OUTPUTDIR=usixml #dossier de destination du code généré
PACKAGE=org.usixml #Nom du package a créer
UTILS=usixml_util #dossier contenant les classes non générées par castor, mais utiles dans
la lib
CASTORDIR=castor-1.3.1 #Dossier contenant Castor
CASTORVERSION=1.3.1 #Version de Castor
################################################################################

# A partir de cette ligne, ne rien changer

CP=$CASTORDIR/castor-$CASTORVERSION-codegen.jar
CP=$CP:$CASTORDIR/castor-$CASTORVERSION-xml.jar
CP=$CP:$CASTORDIR/castor-$CASTORVERSION-xml-schema.jar
CP=$CP:$CASTORDIR/castor-$CASTORVERSION-core.jar
CP=$CP:commons-logging-1.1.1.jar
CP=$CP:jdom.jar

DIR=$OUTPUTDIR/${PACKAGE//./"/"}

echo "Exportation..."
rm -R $OUTPUTDIR

java -cp $CP org.exolab.castor.builder.SourceGeneratorMain -i UsiXML.xsd -binding-file
UsiXML_bind.xml -dest $OUTPUTDIR -package $PACKAGE

cp -R $UTILS $DIR
mv $DIR/$UTILS $DIR/util
echo "Exportation terminée"

echo "Compilation..."
javac -cp $CP:$OUTPUTDIR $DIR/*.java; rm $DIR/*.java
javac -cp $CP:$OUTPUTDIR $DIR/descriptors/*.java; rm $DIR/descriptors/*.java
javac -cp $CP:$OUTPUTDIR $DIR/types/*.java; rm $DIR/types/*.java
javac -cp $CP:$OUTPUTDIR $DIR/types/descriptors/*.java; rm $DIR/types/descriptors/*.java
javac -cp $CP:$OUTPUTDIR $DIR/util/*.java; rm $DIR/util/*.java
echo "Compilation terminée"

echo "Jar'isation..."
rm org.usixml.jar
jar cf $PACKAGE.jar -C $OUTPUTDIR .
echo "Jar'isation terminée"
```

Annexe C

La licence Apache

Apache License

Version 2.0, January 2004

http ://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

C.1 Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

C.2 Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

C.3 Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

C.4 Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions :

You must give any other recipients of the Work or Derivative Works a copy of this License ; and

You must cause any modified files to carry prominent notices stating that You changed the files ; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works ; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places : within a NOTICE text file distributed as part of the Derivative Works ; within the Source form

or documentation, if provided along with the Derivative Works ; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

C.5 Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

C.6 Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

C.7 Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT

WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

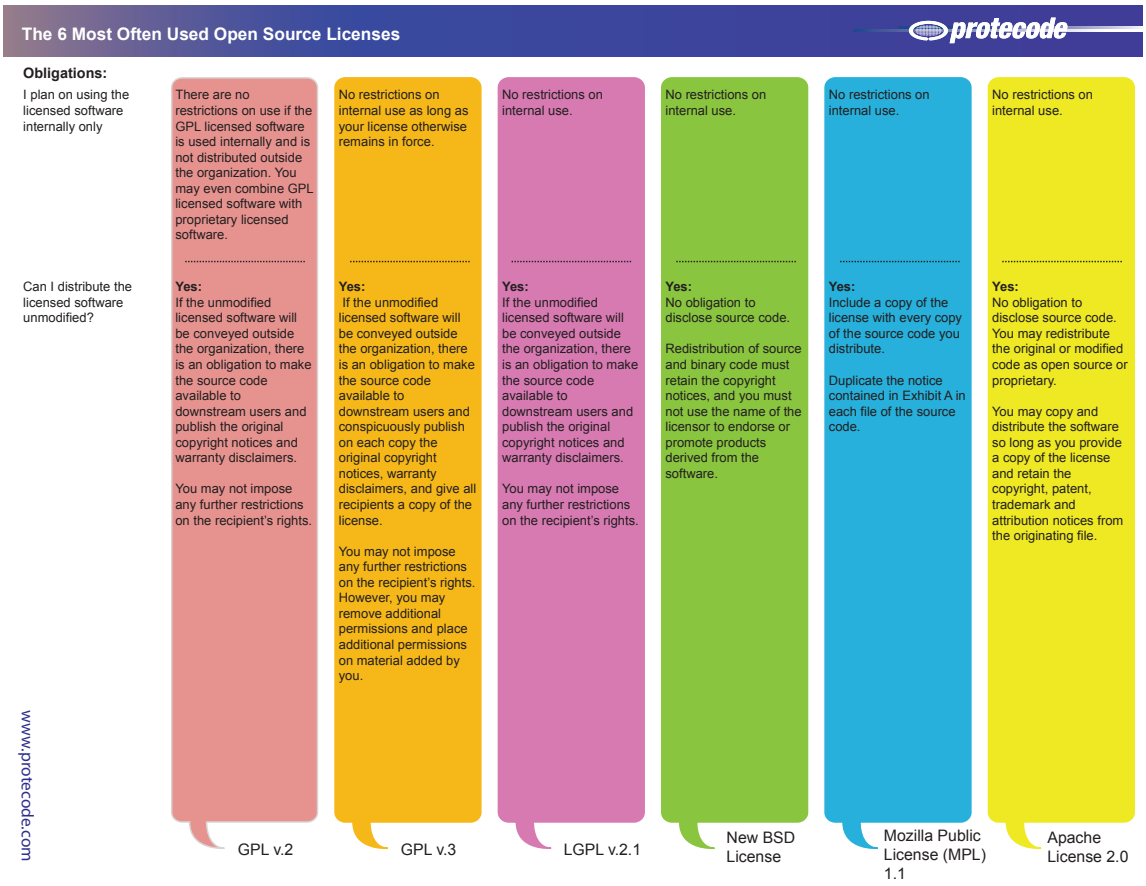
C.8 Limitation of Liability.

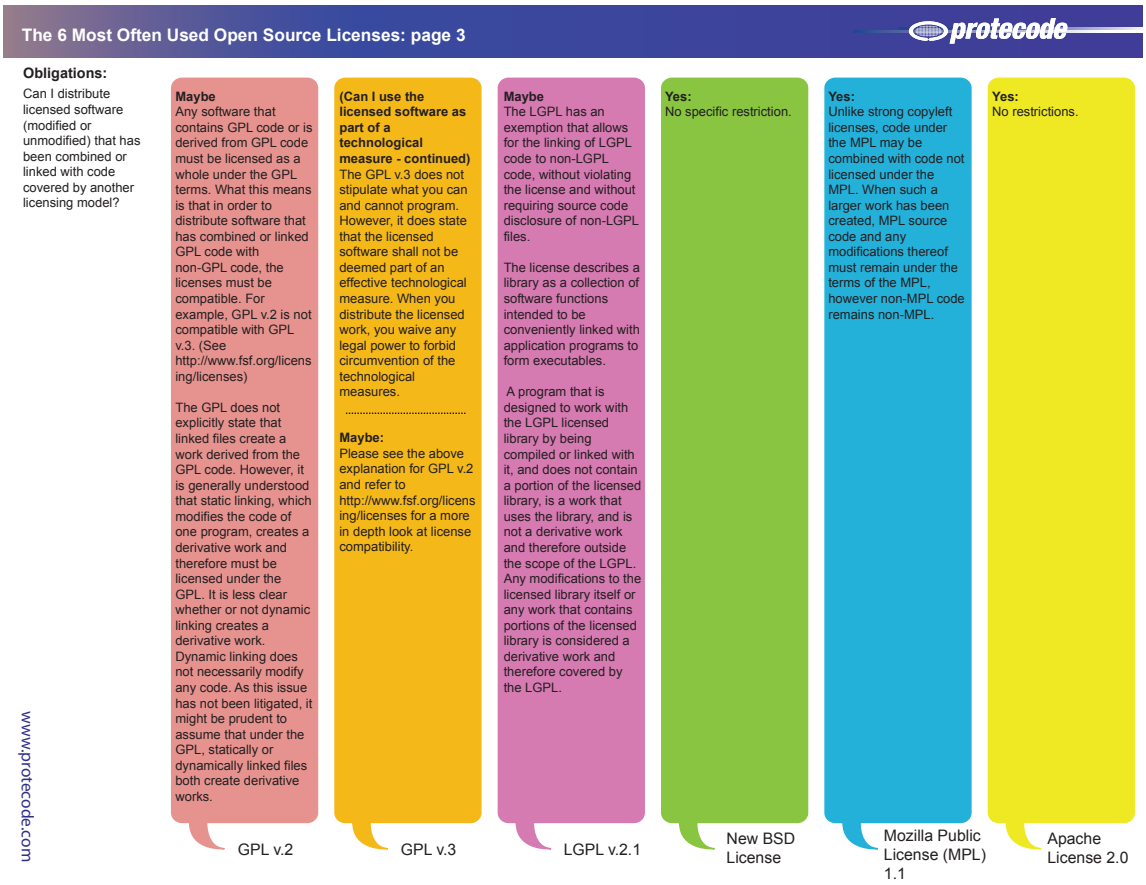
In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

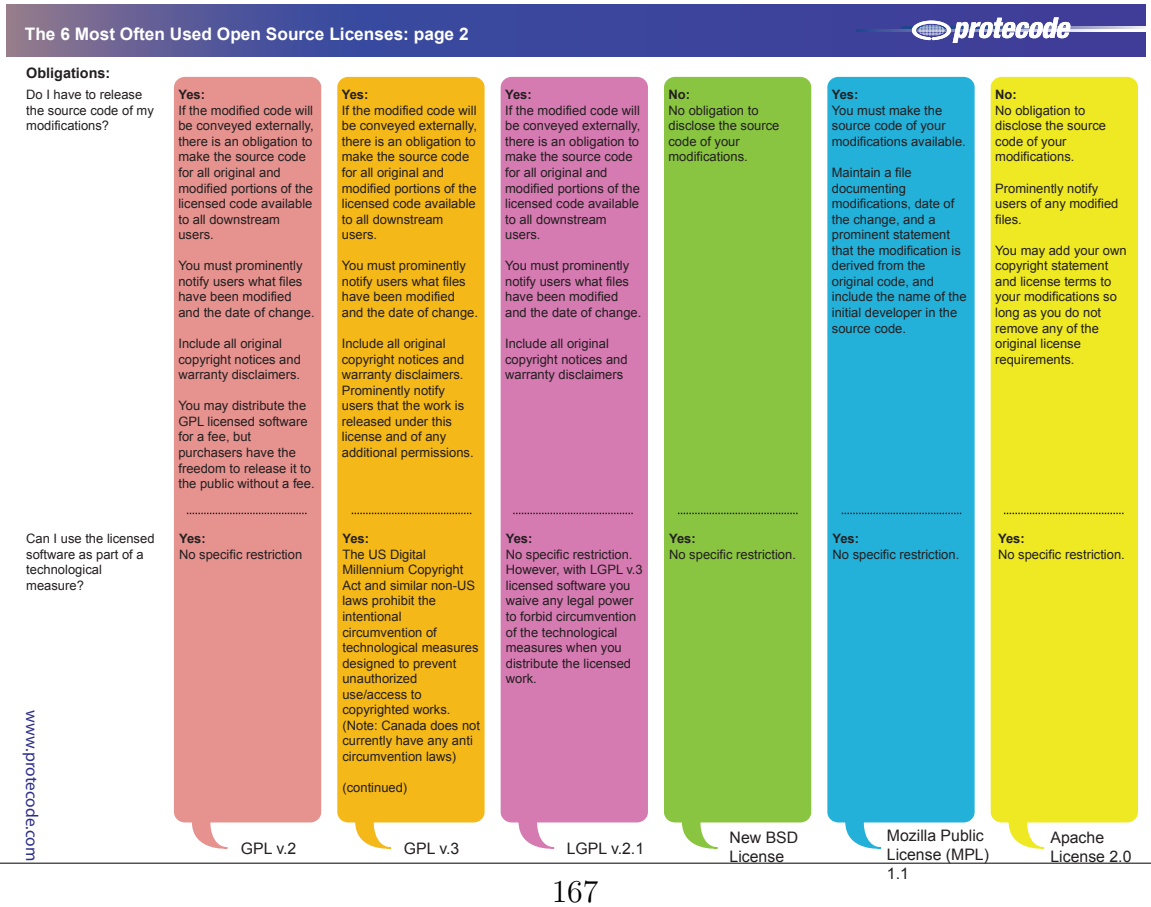
C.9 Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Comparaison des 6 licences open source les plus populaires




The 6 Most Often Used Open Source Licenses: page 3
protecode
Obligations:
Can I distribute licensed software (modified or unmodified) that has been combined or linked with code covered by another licensing model?
Maybe
Any software that contains GPL code or is derived from GPL code must be licensed as a whole under the GPL terms. What this means is that in order to distribute software that has combined or linked GPL code with non-GPL code, the licenses must be compatible. For example, GPL v.2 is not compatible with GPL v.3. (See http://www.fsf.org/licensing/licenses)
The GPL does not explicitly state that linked files create a work derived from the GPL code. However, it is generally understood that static linking, which modifies the code of one program, creates a derivative work and therefore must be licensed under the GPL. It is less clear whether or not dynamic linking creates a derivative work. Dynamic linking does not necessarily modify any code. As this issue has not been litigated, it might be prudent to assume that under the GPL, statically or dynamically linked files both create derivative works.
GPL v.2
(Can I use the licensed software as part of a technological measure - continued)
The GPL v.3 does not stipulate what you can and cannot program. However, it does state that the licensed software shall not be deemed part of an effective technological measure. When you distribute the licensed work, you waive any legal power to forbid circumvention of the technological measures.
Maybe:
Please see the above explanation for GPL v.2 and refer to http://www.fsf.org/licensing/licenses for a more in depth look at license compatibility.
GPL v.3
Maybe
The LGPL has an exemption that allows for the linking of LGPL code to non-LGPL code, without violating the license and without requiring source code disclosure of non-LGPL files.
The license describes a library as a collection of software functions intended to be conveniently linked with application programs to form executables.
A program that is designed to work with the LGPL licensed library by being compiled or linked with it, and does not contain a portion of the licensed library, is a work that uses the library, and is not a derivative work and therefore outside the scope of the LGPL. Any modifications to the licensed library itself or any work that contains portions of the licensed library is considered a derivative work and therefore covered by the LGPL.
LGPL v.2.1
Yes:
No specific restriction.
New BSD License
Yes:
Unlike strong copyleft licenses, code under the MPL may be combined with code not licensed under the MPL. When such a larger work has been created, MPL source code and any modifications thereof must remain under the terms of the MPL, however non-MPL code remains non-MPL.
Mozilla Public License (MPL) 1.1
Yes:
No restrictions.
Apache License 2.0
www.protecode.com


The 6 Most Often Used Open Source Licenses: page 2
protecode
Obligations:
Do I have to release the source code of my modifications?
Yes:
If the modified code will be conveyed externally, there is an obligation to make the source code for all original and modified portions of the licensed code available to all downstream users.
You must prominently notify users what files have been modified and the date of change.
Include all original copyright notices and warranty disclaimers.
You may distribute the GPL licensed software for a fee, but purchasers have the freedom to release it to the public without a fee.
Yes:
If the modified code will be conveyed externally, there is an obligation to make the source code for all original and modified portions of the licensed code available to all downstream users.
You must prominently notify users what files have been modified and the date of change.
Include all original copyright notices and warranty disclaimers. Prominently notify users that the work is released under this license and of any additional permissions.
Yes:
If the modified code will be conveyed externally, there is an obligation to make the source code for all original and modified portions of the licensed code available to all downstream users.
You must prominently notify users what files have been modified and the date of change.
Include all original copyright notices and warranty disclaimers
No:
No obligation to disclose the source code of your modifications.
Yes:
You must make the source code of your modifications available.
Maintain a file documenting modifications, date of the change, and a prominent statement that the modification is derived from the original code, and include the name of the initial developer in the source code.
No:
No obligation to disclose the source code of your modifications.
Prominently notify users of any modified files.
You may add your own copyright statement and license terms to your modifications so long as you do not remove any of the original license requirements.
Can I use the licensed software as part of a technological measure?
Yes:
No specific restriction
Yes:
The US Digital Millennium Copyright Act and similar non-US laws prohibit the intentional circumvention of technological measures designed to prevent unauthorized use/access to copyrighted works. (Note: Canada does not currently have any anti circumvention laws)
(continued)
Yes:
No specific restriction. However, with LGPL v.3 licensed software you waive any legal power to forbid circumvention of the technological measures when you distribute the licensed work.
Yes:
No specific restriction.
Yes:
No specific restriction.
Yes:
No specific restriction.
GPL v.2
GPL v.3
LGPL v.2.1
New BSD License
Mozilla Public License (MPL) 1.1
Apache License 2.0
www.protecode.com

Annexe E

iRise

Cette annexe est un travail de recherche effectué par Olivier Bourdoux, Antoine Dubois et Akonkwa Mubagwa dans le cadre du cours “Interfaces Homme-Machine” donné par M. Vanderdonckt dans le cadre du Master INGI de l’EPL.

annexe irise

Todo list

9, étoffer préface

11, Prototypage vertical et horizontal from Louis

17, Je trouve que ce paragraphe arrive trop tôt : usiSketch n'a pas encore été clairement défini. Pourtant, il introduit très bien ce chapitre en reprenant chaque technologie présentée. Symétriquement, définir usiSketch avant semblerait artificiel, puisque l'état de l'art conduit à sa définition. Qu'en pensez-vous ?

18, Détailler davantage la définition d'un outil de prototypage, ainsi que les types de prototypes possibles. Voir article ci-joint dont vous pouvez reprendre les figures si vous le voulez

19, citation livre sur le prototypage

20, trouver citation pour corroborer

24, VDK : Sections 2.3 et 2.4. Citez bien la référence suivante comme base : http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.45.9491

28, terminer par une conclusion qui résume les limites des familles d'outils analysés et identifie quelques lignes directrices basées sur cet état de l'art.

31, VDK : expliquez la méthodologie suivie pour récolter ces besoins, quelles sources, quel gabarit (inspiré de RADIUS, VOLERE), pourquoi ?

31, ajouter req de vérif de la licence

31, détaillez toutes les raisons pour lesquelles la récolte des besoins est ici fondamentale.

31, reqs : metodo

32, réécrire concepteur

39, Après ces requirements, donner une présentation générale dans laquelle vous faites ressortir ce qui vous semble important, sensible, critique, etc. Il ne faut pas laisser le lecteur simplement avec une liste de requis.

42, finir cette figure

42, finir de décrire chaque module

42, figure représentant la structure des éléments graphiques

43, remettre l'exemple ci-dessus et colorer les noeuds qui sont affichés, par niveau de fidélité

62, image représentant le problème du premier type d'implé

65, finir discussion

66, Eval reqs : Finir les paragraphes de chaque partie

71, mettre nombre de lignes de code

71, Utiliser COCOMO pour plus de précision

73, rédiger walkaware

74, mettre temps

78, jouter page

80, check si on peu choisir le langage ou non

91, produit nouveau

92, citation

104, référence annexe

120, comme expliqué dans la conclusion de l'analyse des données, nous avons décidé de vendre un produit et non un service auprès d'entreprise de développement de logiciel. Ceci limitera notre risque de se lancer sur un marché déjà très prisé comptant 3200 entreprises de développement.Nous proposons dès lors un produit à acheter permettant de faciliter la phase de prototypage nécessaire au développement des UI.

125, check si plus cher ou non avec plan financier.

126, citation pages dor.

133, Compatibilité avec la licence d'eclipse

134, conclusion

141, compléter et structurer les requirements techniques ; en rouge veut dire à compléter

168, annexe irise

$A + B$

5.$x = 42$

Fraction : $\frac{num}{denom}$

Racine : $\sqrt{a+b+c}$

Exposant : x^{y+z}, x^2

Exemple : $\frac{\sqrt{x^2+1}}{5\mu}$

Question	Nb	Médiane	D	Pertinence
Q6.1. Trop dépendant du fournisseur actuel	10	1-2	0.3	< 80%
Q6.2. Fournisseur pas assez fiable	11	1-2	0.25	< 80%
Q6.3. Prix du produit élevé	10	2-3	0.3	< 80%
Q6.4. Prix du SAV élevé	9	2-3	0.278	< 80%
Q6.5. Langage pas assez connu	4	1-2	0.25	< 80%
Q6.6. Livraison trop lente	10	3-4	0.3	< 80%
Q6.7. Nombre de rendez-vous trop élevé	9	2-3	0.278	< 80%
Q6.8. Pas assez pédagogique	9	2-3	0.167	< 80%
Q6.9. Mauvaise vision du résultat durant la conception	11	2-3	0.25	< 80%
Q6.10. SAV trop lent	11	2-3	0.318	80%
Q6.11. Vitesse de mise à jour top faible	8	1-2	0.125	< 80%

Légende :

1. pas du tout d'accord
2. pas d'accord
3. d'accord
4. tout à fait d'accord
5. non répondu

Deg. certitude	Deg. liberté	χ^2	Val. Critique	Conclusion
0.10	3	1.103	6.251	Non rejet => données corrélées