Master 2 MIAGE

Informatique pour la Finance

Université Paris-Dauphine

COMMUNICATION ET COMPREHENSION AU SEIN DES PROJETS INFORMATIQUES

LYXOR ASSET MANAGEMENT – PROJET OMX

Etienne DOYEN

Année universitaire 2014 - 2015

Tuteur d'apprentissage: Cyrille COMBES

Tuteur universitaire: Fabio FURINI

SOCIETE GENERALE
Corporate & Investment Banking

CONFIDENTIALITE

Le corps de ce dossier et ses annexes peuvent contenir des informations confidentielles sur les activités de Lyxor Asset Management. C'est pourquoi, par respect des personnes avec qui j'ai travaillé, il est important que ce mémoire reste accessible dans le seul et unique cadre de ma formation.

REMERCIEMENTS

Je remercie en premier lieu Cyrille Combes, qui m'a accueilli à la table de négociation de Lyxor Asset Mangement et accompagné tout au long de cet apprentissage.

Je remercie également Stéphane Ravetier pour son soutien et ses cours particuliers, l'ensemble des membres de l'équipe la table de négociation pour leur accueil et Bertrand Langeac pour son aide pour rédiger ce mémoire.

Enfin, je remercie Fabio Furini, mon tuteur universitaire.

SOMMAIRE

INTRODUCTION

*" I know that you believe you understand what you think I said, but I'm not sure you realize that what you heard is not what I meant."** Des échos de cette citation de l'américain Robert McCloskey, écrivain et illustrateur de livre pour enfants dans les années 50, raisonnent souvent de nos jours entre les murs des directions informatiques des entreprises. Deux personnes ont une conversation. L'une a des compétences informatiques pour automatiser une solution au besoin de l'autre. L'autre personne, elle, connait parfaitement son métier mais n'a aucune notion de conception d'application. Ces deux personnes doivent collaborer ensemble à la fabrication de cette solution, et pour cela elles doivent communiquer. La communication au sein des projets informatiques est essentielle car elle est vectrice de collaboration et intimement liée à la compréhension du projet par tous les acteurs concernés. Construire un système d'information est une large entreprise qui rassemble nombre de métiers différents. Malgré ce constat une bonne communication a toujours du mal à s'établir entre les acteurs d'un projet informatique et cela nuit à leur compréhension du projet. Le manque de temps ou le manque d'implication sont des raisons parmi d'autres mais ce qu'il manque le plus c'est l'utilisation de méthodes et un environnement adapté. Qu'entendons-nous par environnement adapté ? Quelles sont ces méthodes ? Comment vont-elles améliorer la communication au sein d'un projet informatique et la compréhension du projet par les acteurs ? C'est lors de mon apprentissage chez Lyxor Asset Management (Lyxor AM) que j'ai tenté de répondre à ces questions. Les éléments de réponse vont comprendre entres autres l'implication de l'utilisateur final dans le projet et l'adoption par le développeur du langage des utilisateurs.

Lyxor Asset Management est une société de gestion d'actifs, filiale du groupe Société Générale, gérant plus de 100 milliards d'euros d'actifs. En début d'année 2015 a débuté le projet OMX. Ce projet est le développement d'un order management system (OMS). Un OMS est un logiciel permettant principalement de collecter électroniquement les ordres et les instructions des gérants de fond d'investissement au niveau de la table de négociation, qui est en charge de l'exécution des ordres. Lyxor AM possède déjà une solution achetée à un éditeur dans laquelle elle voulait intégrer toutes les activités de la société et les anciens systèmes d'information, impliquant alors en même temps tous les corps de métiers dans ce projet. Mais ce fut un échec et aujourd'hui le logiciel n'est utilisé que partiellement, obligeant un retour aux anciens systèmes d'information. Lyxor AM a donc décidé de créer son propre OMS. J'ai pu suivre une partie de ce projet qui implique cette fois-ci seulement la table de négociation et

une petite partie de la gestion. J'ai procédé à une sélection de méthodes, notamment vues pendant le cours de Méthodes Agiles d'Ingénierie Logicielle, qui tendent à améliorer la communication entre les acteurs d'un projet informatique mais aussi leur compréhension de ce projet. Ces méthodes ont été appliquées à l'équipe en charge du projet OMX. Celle-ci est composée d'un chef de projet, de deux développeurs et d'un expert fonctionnel ou maître d'ouvrage. De plus des utilisateurs finaux, c'est-à-dire deux gérants et une personne de la table de négociation ont participé au projet grâce aux méthodes que nous allons décrire dans ce mémoire. Je me suis basé ensuite sur les impressions et les retours de toutes ces personnes pour connaître la véritable efficacité de ces méthodes et pour élaborer la solution la plus adaptée à ce type de projet.

Ce mémoire débutera par le constat de l'échec d'un grand nombre de projets informatiques et cela à cause du manque de communication et du manque d'une réelle méthode pour stimuler les échanges au sein de ces projets. Puis nous décrypterons les enjeux de cette communication et comment elle prend forme. Dans la seconde moitié de ce mémoire nous décrirons d'abord les méthodes de conception, de développement, de conduite et les environnements qui encourageront la communication puis nous en ferons un bilan et donnerons des indications permettant de mieux les adapter.

** « Je sais que vous croyez avoir compris ce que vous pensez que j'ai dit, mais je ne suis pas sûr que vous réalisiez que ce que vous avez entendu n'est pas ce que j'ai voulu dire »*

1 L'échec des projets informatiques et la communication

Dans cette première partie nous allons exposer ce qu'est un projet informatique ainsi que ce qui nous permet d'évaluer si celui-ci est un succès ou un échec. Puis nous nous intéresserons aux données de ces dernières années ainsi qu'aux raisons des échecs, qui sont fortement liées à la communication au sein des projets.

1.1 Le triangle du projet

Selon la norme ISO 10006, « Un projet est un ensemble d'activités coordonnées et maîtrisées comportant des dates de début et de fin, entrepris dans le but d'atteindre un objectif conforme à des exigences spécifiques ». Un projet a besoin d'organisation et d'encadrement pour se dérouler selon les décisions qui ont été prises en amont du projet. Ces décisions concernent principalement la planification et la budgétisation mais aussi la maîtrise et le pilotage des risques liés au projet ou l'intervention des parties prenantes du projet. La gestion d'un projet informatique doit donc atteindre les objectifs fixés par les décideurs du projet tout en respectant des contraintes de temps, de ressources et d'objectifs.

Ces trois contraintes sont présentables sous la forme d'un triangle. Les côtés de ce triangle sont le **temps**, l'argent ou **les ressources** et l'objectif ou **les fonctionnalités**. La qualité est comme un sommet caché. Elle dépend des choix qui sont faits et de la gestion en ce qui concerne les trois côtés du triangle. La qualité englobe également la satisfaction des utilisateurs qui ont exprimé les besoins qui ont donné naissance à ce projet. Le développement des fonctionnalités doit prendre en compte l'objectif visé, c'est-à-dire ce que nous attendons d'elles qu'elles fassent, mais aussi leur optimisation, qui peut comprendre le temps d'exécution, la mémoire du poste occupée par cette exécution ou encore la consommation d'énergie. Ces paramètres peuvent également rentrer dans le scope de la qualité. Il est donc difficile de la quantifier. D'après le triangle du projet elle est représentative de l'équilibre entre le temps, l'argent et les objectifs visés mais son évaluation est aussi propre à chaque gestionnaire et chaque organisation. Ce triangle se présente de la manière suivante :

Le principe de ce triangle est que nous ne pouvons pas modifier les délais, les ressources ou le nombre de fonctionnalités d'un projet sans affecter au moins l'un des deux autres paramètres, et donc la qualité.

- Pour économiser du temps nous pouvons :
  - augmenter les ressources du projet, c'est-à-dire augmenter le budget, ou
  - revoir à la baisse les objectifs en termes de fonctionnalités, c'est-à-dire en supprimer.
- Pour baisser les coûts d'un projet nous pouvons :
  - reporter l'échéance du projet, c'est-à-dire reporter la date de fin du projet ou
  - supprimer des fonctionnalités.
- Pour ajouter des fonctionnalités nous pouvons :
  - allonger les délais ou
  - augmenter le budget.

Dans la plupart des projets au moins un des côtés du triangle n'est pas modifiable, il est immuable. Il est très un important de clarifier dès le début du projet de qui ces éléments immuables sont le ressort. Lorsque les problèmes surgissent, et ils finissent toujours par surgir, quelqu'un doit décider quel élément est le plus important pour la réussite du projet.

Le respect de ces contraintes est l'unité de mesure traditionnelle de la réussite d'un projet. Le projet sera réussi s'il est mené dans les temps, sans dépasser le budget et qu'il offre toutes les fonctionnalités attendues. Si l'une des contraintes n'est pas respectée il ne peut pas non plus être considéré comme un échec. De plus d'autres indicateurs peuvent être pris en compte pour mesurer la réussite du projet, comme la satisfaction du client et des utilisateurs finaux, la rentabilité du projet, ou le fonctionnement de l'application depuis son déploiement. Nous

allons discuter de ces paramètres de mesure de la réussite d'un projet informatique utilisé notamment par le Standish Group.

1.2 Succès et échec selon le Standish Group

Créé en 1994, le *Chaos Report* du Standish Group fourni chaque année des informations sur la réussite des projets informatiques. Ces informations sont fréquemment utilisées pour améliorer la réussite des projets. D'abord le Standish Group classe les projets selon 3 catégories :

- Résolution de type 1, ou « **succès** » : Le projet a été complété en temps et en heure, dans le respect du budget et a livré tous les services et fonctionnalités initialement spécifiés.
- Résolution de type 2, ou « **challengé** » : le projet a été complété, est opérationnel mais a dépassé les estimations de budget et/ou de temps et/ou a livré moins de services et de fonctionnalités qu'initialement spécifié.
- Résolution de type 3, ou « **échec** » : le projet a été annulé à un moment du cycle de développement.

Avant d'exposer les chiffres nous allons discuter de cette méthode de classement afin de bien prendre conscience de ce qu'elle représente mais aussi des questions qu'elle soulève.

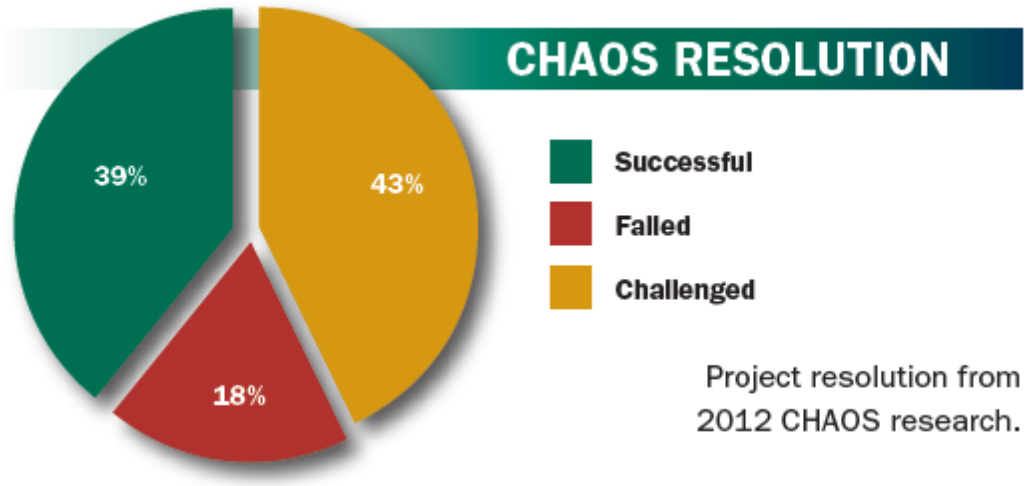
Le succès d'un projet est défini sur la base du respect des estimations initiales de coût, délais et fonctionnalités. Seulement cette notion de succès doit être modérée car pendant le projet des éléments de l'environnement peuvent varier et conduire à des demandes de changement. Il faut par exemple prendre en compte qu'au cours d'un projet, des fonctionnalités a priori indispensables dans l'esprit des utilisateurs apparaissent inutiles à automatiser, car trop complexes pour peu de valeur ajoutée. Le projet peut donc être livré en respectant les délais et le budget mais avec moins de fonctionnalités que prévues sans que ce soit un point négatif. Des variations naturelles de coûts, délais et fonctionnalités peuvent arriver plusieurs fois pendant un projet complexe et de longue durée. Nous pouvons donc supposer que la catégorie « challengé » englobe des projets qui, suivant un autre point de vue, pourraient être des succès. Le succès s'établie par la différence entre estimation initiale et réel constaté, mais le poids de cette différence est fortement lié au contexte (comme par exemple le secteur de l'entreprise ou la taille du projet). En effet dans certains contextes, 20 % d'erreurs de prévisions ne prêtent pas à conséquence et n'ont pas d'impact. Dans d'autres contextes, seulement 2% de dépassement peuvent causer beaucoup plus de mal. En conclusion il faut comprendre que les chiffres du Standish Group ne reflètent pas une réalité mais une

perception du monde des projets informatiques. Néanmoins nous allons pouvoir faire quelques constats à partir de ces données.

1.3 En 2012, près d'un projet informatique sur cinq est un échec…

Le Chaos Manifesto de 2013 est une étude portant sur une base de 50 000 projets de 2004 à 2012. Ces projets sont essentiellement basés aux Etats-Unis et en Europe (60% aux Etats-Unis, 25% en Europe et 15% dans le reste du monde). Les entreprises portant les projets de cette étude sont pour plus de la moitié des multinationales, 30% sont des entreprises de taille moyenne et 20% sont de petites entreprises. Cette diversité permet d'avoir une vue globale des environnements où naissent les projets informatiques. En 2012 le pourcentage de projets considérés comme des succès se situe à 39%. 43% sont challengés et 18% sont des échecs.

Le Standish Group calcule son taux de succès en comptant le nombre de projets qui ont eu une prévision initiale supérieure au réel pour les coûts et les délais et ceux qui ont une couverture prévisionnelle inférieure pour les fonctionnalités puis en divisant cela par le nombre total de projets, toutes entreprises confondues.



RESOLUTION

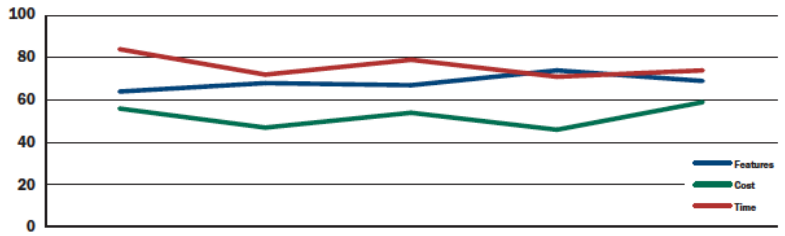
	2004	2006	2008	2010	2012
Successful	29%	35%	32%	37%	39%
Failed	18%	19%	24%	21%	18%
Challenged	53%	46%	44%	42%	43%

Project resolution results from CHAOS research for years 2004 to 2012.

Nous pouvons constater sur l'ensemble de la période allant de 2004 à 2012 que globalement le pourcentage de succès augmente tandis que le pourcentage d'échecs a tendance à stagner. La pire année était celle de 2004 où seulement 29% des objets étaient des succès. Le nombre de projets challengés a significativement baissé de 2004 à 2006 mais ensuite il représente toujours presque la moitié des projets jusqu'en 2012. Nous allons nous intéresser de plus près à ces projets challengés et aux statistiques de dépassement de budget ou de délai et de fonctionnalités développées.

OVERRUNS AND FEATURES

Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012.



	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%
FEATURES	64%	68%	67%	74%	69%

L'année 2012 montre une légère augmentation des dépassements de budget et de délai. Les dépassements de budget sont passés de 56% en 2004 à 59% en 2012. Les dépassements de budget sont aussi à la hausse entre 2010 et 2012, passant de 71% à 74%. Les dépassements de délai ont atteint 84% des projets en 2004. Le taux de fonctionnalités développées a baissé, avec 74% des exigences qui ont été satisfaites en 2010 à 69% en 2012. Il est possible d'interpréter positivement ces derniers chiffres car les entreprises ont tendance à se concentrer sur les fonctionnalités qui ont le plus de valeur ajoutée plutôt que de développer 100% des exigences exprimées en début de projet.

L'étude démontre qu'une fois l'application en route, 20% des fonctionnalités sont souvent utilisées alors que 50% ne le sont que peu, voir jamais. 30% d'entre elles sont utilisées avec des fréquences irrégulières. Ces chiffres montrent la difficulté qu'il y a à se concerter, sélectionner et implémenter les fonctionnalités du projet, c'est-à-dire bien définir les objectifs de celui-ci. Se concentrer sur les 20% de fonctionnalités qui représentent 80% de la valeur ajoutée du projet va donner une meilleure rentabilité et améliorer la satisfaction de l'utilisateur. Mais il est aussi des situations où la hiérarchie ou la politique, de la société qui développe le projet ou du client, exigent que 100% des objectifs soient complétés, au détriment d'une stratégie plus prudente comme celle évoquée précédemment. Nous commençons à distinguer comment le manque de communication et d'échanges, en particuliers quand il s'agit d'exprimer les besoins qui vont fixer les objectifs du projet, peut nuire à sa réussite.

1.4 … et l'une des premières raisons est le manque de communication

Les facteurs d'un échec sont multiples mais plusieurs études ont permis de lister ceux qui reviennent le plus souvent. Voici une sélection de ceux en lien plus ou moins direct avec la communication au sein des projets informatiques :

- manque d'engagement des principaux acteurs,
- manque d'implication des utilisateurs finaux,
- mauvaise compréhension par les différentes parties prenantes de ce qui est attendu,
- ignorer les problèmes qui se posent.

L'engagement des acteurs, qu'ils soient impliqués dans le projet ou seulement concernés, techniques ou fonctionnels, est essentiel à la réussite du projet. La communication est clé dans la responsabilisation et l'engagement de ces acteurs. Un exemple flagrant et qui revient très régulièrement est le désintérêt marqué des utilisateurs pour le projet. La gestion de projet doit souligner l'importance de la participation des utilisateurs finaux dans les projets. Seulement il est souvent difficile d'impliquer les utilisateurs et cela est encore plus vrai lorsque le projet va mal. De plus le chef de projet peut considérer que l'intervention des utilisateurs est effectivement une perte de temps et il n'est pas prêt à entamer un processus de concertation qui risque d'être beaucoup plus long que ce qu'il a prévu au départ. Peut donc aboutir une entente ou une complicité entre un responsable de projet pressé et des utilisateurs indifférents. Certains chefs de projet sont même convaincus de la non participation des utilisateurs, qu'ils considèrent comme trop « incompétents » pour pouvoir participer à des projets « trop innovants pour eux ». Ces utilisateurs de leur côté sont très souvent occupés à

des tâches opérationnelles récurrentes qui ne leur permettent pas facilement de fonctionner en mode projet. Les expériences passées des utilisateurs peuvent les amener à considérer qu'une solution existe déjà ailleurs et qu'il suffit de l'appliquer ici (attitude dite du « y'a qu'à faut qu'on »). Enfin la phase de partage des responsabilités en cas d'échec tend à éloigner les acteurs du projet, personne ne voulant être considéré comme le responsable de l'échec.

La mauvaise compréhension des acteurs vient généralement d'une mauvaise définition du problème ou du besoin. Le périmètre et la portée de celui-ci doivent être clairs pour tout le monde et cela ne se fera qu'au prix d'un réel investissement et d'une réelle communication des acteurs concernés.

Il n'existe pas une méthode ni une solution précise qui maximise la communication au sein des projets. Pourtant comme nous avons pu le constater l'amélioration de la communication ne peut être que bénéfique pour le projet.

Le Chaos Report montre tout de même une augmentation du taux de réussite des projets. Il liste quelques facteurs en leur attribuant des points qui leur donnent un poids dans ce succès. Deux de ces facteurs ont retenus mon attention pour leur lien avec la communication. D'abord l'implication de l'utilisateur, pour les raisons données précédemment, et l'apport des méthodes agiles, dont certaines tendent à améliorer cette communication.

Factors of Success	Points
Executive management support	20
User involvement	15
Optimization	15
Skilled resources	13
Project management expertise	12
Agile process	10
Clear business objectives	6
Emotional maturity	5
Execution	3
Tools and infrastructure	1

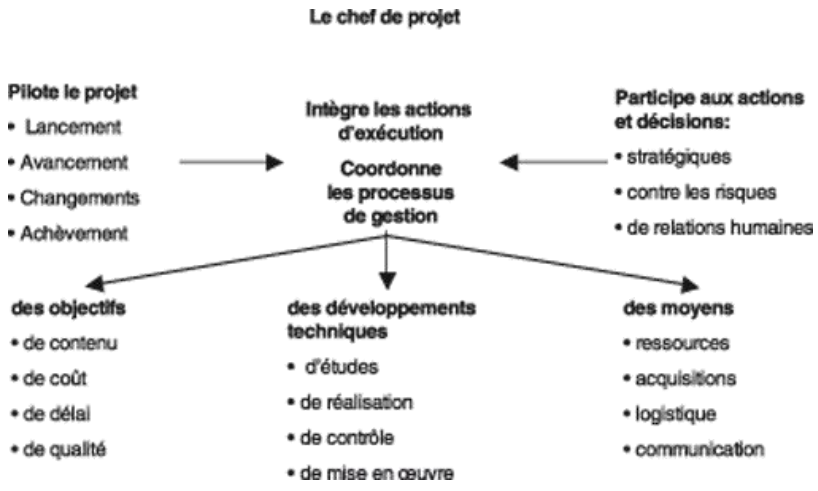
Nous allons dans la partie suivante expliciter cette communication en donnant les principaux acteurs des projets informatiques qu'elle concerne et en décrivant plus en profondeur l'importance qu'elle a.

2 Un projet, des communications

Un projet informatique concerne plusieurs acteurs : développeurs, chefs de projet, experts fonctionnels, utilisateurs, etc. Chacun de ces acteurs tient un langage spécifique et la communication entre chacun de ces interlocuteurs doit être adaptée à ces spécificités. En effet, transmettre des informations via une seule communication revient à perdre l'attention des collaborateurs non rompus à cette communication. Par exemple, les utilisateurs n'ont pas besoin d'informations détaillées et les développeurs ne recherchent pas de données au niveau macro. En utilisant une seule communication et donc en fournissant toutes ces différentes données en une seule fois, personne ne se sentira concerné et les informations importantes ne parviendront pas (ou mal) à leurs destinataires initiaux. Nous allons d'abord décrire le rôle et les apports des principaux acteurs d'un projet informatique afin de mieux comprendre quelle place ils ont au sein du projet et comment chacun peut alimenter les échanges d'informations au cours du projet.

2.1 Le chef de projet

Le chef de projet a pour but l'organisation et le bon déroulement du projet. Il planifie la mise en œuvre d'un projet ou d'un ensemble de projets de la conception à la réalisation en s'appuyant sur les ressources à sa disposition, internes comme externes. Il doit donc animer les équipes dont il a la charge. Le chef de projet a également pour mission le suivi et le reporting de l'avancement du projet en termes d'objectifs, de coûts et de délais. En plus de ses compétences en méthodes d'organisation le chef de projet doit donc faire preuve d'un bon relationnel que ce soit avec la maîtrise d'ouvrage, qui s'assure que ses demandes sont bien prises en compte, ou avec les équipes du projet en charge de son développement. Il est donc un véritable moteur de communication au sein du projet mais il doit veiller à utiliser le bon langage avec le bon interlocuteur.



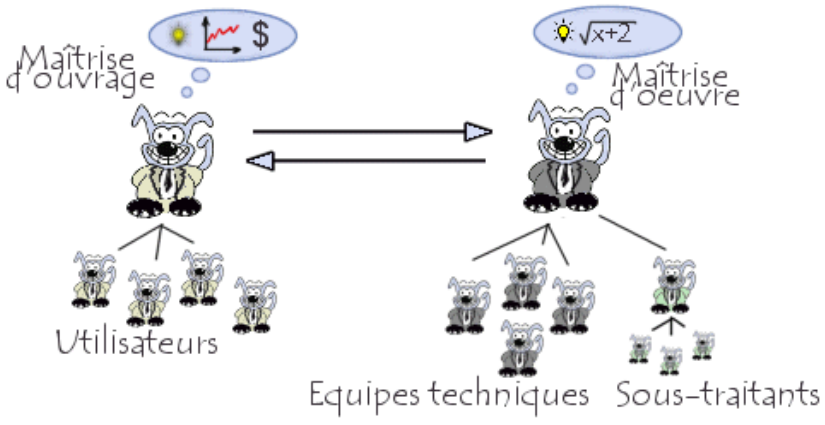
2.2 Le développeur

Également appelé analyste-programmeur, le développeur doit concevoir et développer une application informatique en réponse aux besoins exprimés par la maîtrise d'ouvrage. Sous la direction du chef de projet c'est lui qui possède les compétences en langages de programmation nécessaire à ce développement. Il utilise donc un langage très orienté technique c'est-à-dire composé de terme et d'expression proche du langage de programmation utilisé et, plus généralement, de l'informatique. Nous sommes donc loin du langage des utilisateurs qui lui se rapporte au métier qu'ils exercent.

2.3 La maîtrise d'ouvrage

La maîtrise d'ouvrage exprime le besoin. Elle définit également les objectifs, l'échéancier et le budget du projet. Elle est représentative des utilisateurs finaux de l'application car elle composée d'experts métier. Elle a la responsabilité de la compréhension et de l'expression des besoins des utilisateurs et donc de l'ajout ou la suppression de fonctionnalités. La maîtrise d'ouvrage n'a pas de connaissances techniques, elle délègue à l'équipe du projet des choix d'ordre fonctionnel qui seront implémentés par les développeurs. De ce fait elle utilise un langage très orienté vers le métier et compréhensible surtout par les utilisateurs de l'application.

Nous pouvons voir la nécessité qu'il y a à trouver un langage commun entre la MOA et le chef de projet et l'équipe de développeurs, également nommée maîtrise d'œuvre. La MOA doit faire face à un environnement technique et un métier qui peut être d'une grande complexité. Elle peine à formaliser clairement les besoins.



2.4 Autres acteurs récurrents

Le poids de ses actions est souvent sous-estimé et pourtant il est le garant de la qualité et de la viabilité du projet : c'est le **testeur**. Il s'assure que l'application réponde aux exigences du client sans erreurs ni défauts. Il élabore et planifie les tests avant leur exécution et vieille à la mise en place des corrections nécessaires. Pour cela le testeur doit se familiariser avec les enjeux du projet pour le client. De plus il doit comprendre les usages finaux et les besoins du client et donc s'intéresser à ce qu'il y a autour de l'application.

Lorsqu'il ne possède pas en interne les ressources nécessaires pour la réalisation de certaines tâches du projet, le chef de projet peut faire appel à une ou plusieurs entreprises externes, on parle alors de **sous-traitance**. Chaque sous-traitant réalise un sous-ensemble du projet mais n'a aucune responsabilité directe. Néanmoins cela en fait un interlocuteur supplémentaire au sein du projet qui doit comprendre les tenants et aboutissants de celui-ci.

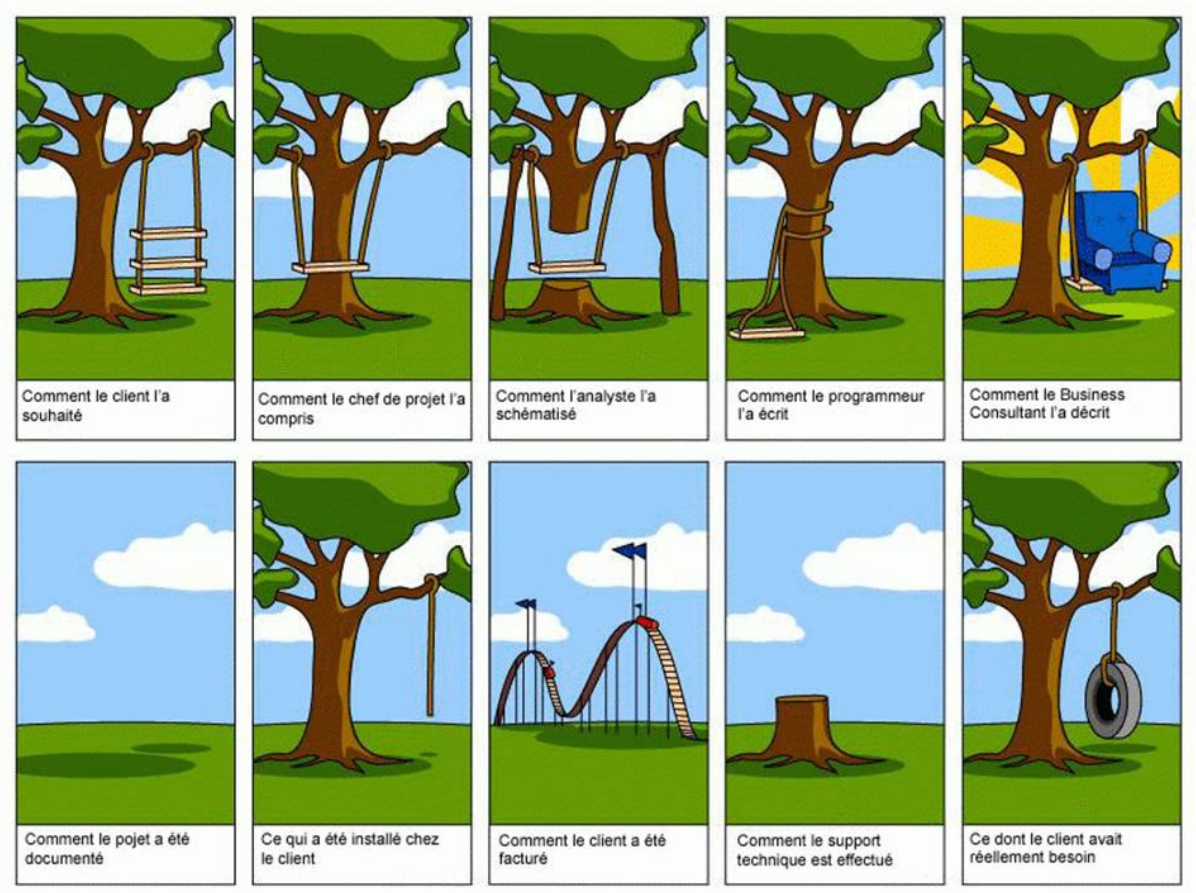
D'autres acteurs comme l'administrateur de base de données, le comité directeur ou les fournisseurs font partie de la vie des projets informatiques mais il n'est pas nécessaire de préciser leur rôle pour aller plus en avant dans ce mémoire.

Nous pouvons voir que de par leur rôle et leur apport au projet, chacun des acteurs à besoin d'informations qui lui sont spécifiques mais également d'en partager d'autres. Pour les obtenir et les transmettre ils doivent communiquer entre eux.

2.5 Partager les décisions

Les décisions autour d'un projet informatique engagent des choix fonctionnels, techniques et organisationnels. Il est donc indispensable qu'elles soient partagées entre tous les acteurs du projet. Le partage de l'information et sa traçabilité permet également de responsabiliser tous ces acteurs. De plus le partage des mises à jour majeures du projet fera remonter plus rapidement les possibles erreurs ou incohérences et informera tout le monde sur les nouveaux éléments à intégrer dans la conception et la réalisation du projet. La validation par chacun des acteurs des nouvelles orientations prises est motrice de leur implication dans le projet. Les décisions sont donc une des premières sources de communication au sein du projet, quelque soit le niveau auquel elles sont prises. Nous savons donc l'importance que l'information circule mais ce n'est qu'une partie de l'enjeu de la communication au sein du projet car il faut également penser à son interprétation et à sa compréhension par chacun des acteurs.

L'illustration suivante représente bien les incompréhensions que nous retrouvons tout au long du projet. Les interprétations des informations que chacun des acteurs reçoit peuvent conduire à de gros écarts de compréhension. Une personne va automatiquement se servir de son point de vue pour comprendre ce qu'on lui dit. Ce point de vue est biaisé par sa fonction au sein du projet et le métier qu'elle exerce. Elle est donc particulièrement sensible aux ambiguïtés qu'il pourrait y avoir dans le flot d'informations qu'elle reçoit d'autre personne. Nous pouvons donc identifier deux sources d'incompréhension : la formulation de l'information et l'interprétation personnelle. En effet la case « Comment le client l'a souhaité » montre déjà des points qui mériteraient une clarification (une balançoire à trois sièges...). La compréhension du chef de projet et ce que le développeur a codé ne font que plus s'éloigner de l'objectif. La case « Comment le Business Consultant l'a écrit » montre l'image parfaite mais éloignée de la réalité que peut avoir un expert fonctionnel. Les cases inférieures sont moins en rapport avec la communication mais nous montre le problème récurrent du manque de documentation, voire de son inexistence, des projets informatiques. Enfin la dernière case est ce dont avait réellement besoin le client mais qui n'a été exprimé ni compris par personne.



Nous comprenons mieux l'importance du langage au sein du projet car c'est ce langage avec lequel les acteurs doivent communiquer et se comprendre.

2.6 L'importance d'un langage commun

Nous pouvons constater que les problèmes de communication découlent pour une grande partie du manque de rigueur dans la définition de ce que les acteurs souhaitent exprimer. Pour remédier à ce problème de compréhension dans la vie de tous les jours nous avons le **dictionnaire** (ou n'importe quelle autre document imposé). Chacun a une vision différente des choses liée à nombre de paramètres tels que l'éducation, la formation, l'expérience. Sans ce type de document, impossible de se mettre d'accord sur "quoi est quoi". Pour nous rapprocher de nos projets informatiques il suffit d'écouter les termes employés par des intervenants techniques comme les développeurs et ceux qu'emploient les experts fonctionnels. Les développeurs parlent dans un langage où les mots *classes*, *méthodes*, *algorithmes*, *héritage* ou *polymorphisme* reviennent très régulièrement. Pour eux ces termes prennent un sens qui est entièrement lié à l'activité de programmation de logiciel. Des problèmes supplémentaires surviennent avec la tendance qu'ils ont à vouloir faire correspondre les éléments du métier qu'ils implémentent à des artefacts de programmation. De leur côté les experts fonctionnels utilisent le jargon en lien avec leur métier. Les besoins qu'ils expriment auprès de l'équipe projet sont de complexité variable. Certains peuvent être abordables et faciles à comprendre tandis que d'autres peuvent nécessiter une formation afin d'assimiler des règles très spécifiques au métier. De plus ce jargon peut aussi être spécifique à l'entreprise ou à l'équipe. Il n'est pas rare de voir des personnes travaillant depuis plusieurs années ensemble au sein de la même équipe se constituer une « langue » qui leur permettra de s'exprimer et de se comprendre plus rapidement. Mais cette « langue » ne sera compréhensible que par les membres de cette équipe. C'est cette barrière entre environnement technique et environnement fonctionnel que la communication doit franchir.

Nous savons maintenant le bénéfice qu'il y a à établir une bonne communication au sein du projet, ne serait-ce que pour qu'il réponde vraiment aux attentes et surtout aux besoins des utilisateurs finaux. Mais nous allons voir que bien se comprendre et bien communiquer c'est aussi de la valeur ajoutée pour le projet.

2.7 No narrative = no business value

Nous allons d'abord nous intéresser au titre de cette sous-partie et à sa traduction. Celui-ci concerne le besoin des utilisateurs. Il signifie que si on ne peut pas en faire le récit, alors il n'a pas de valeur pour le métier. Autrement dit la valeur ajoutée que procurera au projet la solution à un besoin provient directement de l'expression de ce besoin. Nous avons remarqué précédemment que de l'expression d'un besoin peut découler beaucoup d'incompréhensions et si ces incompréhensions ne sont pas clarifiées elles se retrouveront dans l'application. Celle-ci ne fera pas ce que les utilisateurs attendent et ne leur servira donc à rien, elle n'aura aucune valeur pour eux. C'est pour cela que nous parlons de valeur ajoutée. Le premier rôle d'une application est d'automatiser des tâches qui sont réalisées tous les jours par des personnes. Bien plus que le nombre de lignes de code c'est le temps gagné et la simplification de ces tâches qui constituent la valeur de l'application. C'est pourquoi un besoin doit être parfaitement défini et l'exprimer n'est pas une mince affaire, même pour les experts fonctionnels. L'identification elle-même demande de la coopération entre les différents acteurs du projet. Il s'agit non seulement de comprendre le besoin mais également tous les concepts qui l'entourent. C'est de cette démarche d'échange et de communication que la valeur du besoin va se distinguer. Un autre avantage qu'il y a à connaître la valeur des besoins, et donc des futures fonctionnalités, est de pouvoir les comparer et les prioriser les uns par rapport aux autres. Améliorer la communication et favoriser les échanges en rapport avec l'expression d'un besoin c'est s'assurer de la pertinence de ce besoin auprès de toutes les parties.

Nous connaissons désormais l'importance et l'impact de la communication et de la compréhension dans un projet informatique. Nous allons dans la partie suivante nous imprégner des méthodes qui on été sélectionnées et des environnements que nous avons retenus pour améliorer ces deux points.

3 Améliorer la communication et la compréhension

Dans cette troisième partie nous allons décrire des méthodes de développement, de conception, de pilotage de projet et des environnements qui vont faciliter et encourager les interactions entre les acteurs du projet et améliorer leur compréhension de chacun des aspects du projet. L'un des principaux changements pour la plupart des intervenants sur le projet OMX était d'évoluer dans un environnement agile.

L'agilité a été ma principale source de recherche de méthodes car deux des quatre valeurs fondamentales des méthodes agiles sont en adéquation avec ma problématique. Celles-ci prônent l'équipe et la collaboration. Elles sont décrites dans le manifeste des méthodes agiles par les citations suivantes :

« Les individus et leurs interactions, plus que les processus et les outils »

« La collaboration avec les clients, plus que la négociation contractuelle »

Dans un environnement agile l'équipe est bien plus importante que les procédures de fonctionnement ou les outils. L'équipe est composée de développeurs soudés qui communiquent plutôt que de profils plus spécialisés qui travail de manière isolée. De plus le client doit être impliqué dans le développement et pas seulement dans la négociation des contrats. Ces demandes ne doivent pas être négligées. La communication et la collaboration sont donc des notions fondamentales. Néanmoins il ne suffit pas de dire que nous avons appliqué l'agilité pour répondre au manque de communication dans les projets informatiques. Les méthodes et prescriptions retenues devaient se cantonner à l'amélioration de la communication entre les acteurs du projet et l'amélioration de leur compréhension du projet. Nous allons préciser notre solution en décrivant les méthodes que nous avons sélectionnées et utilisées lors du projet OMX.

3.1 Le backlog

Le backlog est la liste des fonctionnalités qui vont devoir être implémentées et des tâches qui vont devoir être accomplies pour la réalisation du projet. Cette liste est établie lors de rencontres entre les différents acteurs, techniques comme fonctionnels, qui participent au projet. Ils doivent réfléchir ensemble à tout ce qui leur semble indispensable ou nécessaire au projet. Cette liste n'est absolument pas définitive et est amené à être modifiée tout au long du projet. Si, en cours de projet, une tâche contenue dans le backlog s'éloigne de l'objectif du projet, elle sera retirée. Au contraire si de nouvelles tâches apparaissent nécessaires elles viendront s'ajouter au backlog. L'état du backlog est donc lié à l'état des connaissances de

l'équipe à un instant donné du projet et il sert de référentiel en matière d'avancement et d'exigences. En effet une telle liste permet de prioriser ces tâches les unes par rapport aux autres et d'établir un planning à court terme.

Le backlog peut prendre n'importe quelle forme et nous avons choisi celle qui nous paraissait apporter le plus de visibilité et aiderait à rassembler les équipes pour le consulter ensemble : un tableau (voir annexe 1). Tout au long du projet un tableau accroché à côté des bureaux de l'équipe projet a servi de backlog. Les tâches et fonctionnalités sont écrites sur de petits bouts de papier manipulables sur le tableau grâce à des aimants. L'intérêt d'un tel outil est d'offrir à n'importe qui passant devant ce tableau d'avoir une vision globale du projet, de ce qui est passé, de ce qui est en cours et de ce qui est à venir. A cette forte communication visuelle s'ajoute la capacité que le backlog a à encourager la collaboration et la conversation entre tous les acteurs du projet.

Il est donc possible de booster la communication avec un simple tableau, nous allons maintenant nous intéresser à la communication au niveau de la conception de l'application.

3.2 Le domain-driven design

Le domain-driven design (DDD) est une approche de conception introduite par Eric Evans en 2003. Par approche de conception nous n'entendons pas une méthode ou une technique mais une manière de penser la conception autour du code. C'est une technique de collaboration entre les experts fonctionnels du projet et l'équipe technique en charge de son développement. Précisons d'abord ce qu'est la conception.

La conception est une manière de concevoir et d'implémenter un ensemble de fonctionnalités en prenant en compte un ensemble de considérations. Ces considérations peuvent concerner l'adaptabilité, les coûts, la performance, la qualité, la sécurité, ... La conception permet d'avoir un plan d'actions de développement qui va prendre en compte la complexité de l'application. Elle est considérée par certains comme un art car elle ne peut pas être appliquée comme une science précise faite de formule et de théorème. La conception et le développement d'une application contiennent toujours la touche personnelle de ceux qui l'ont conçue et développée. Cela renforce l'idée de l'importance du poids de l'humain dans un projet informatique. Après cette brève introduction à la conception de logiciel nous allons détailler en quoi consiste l'approche DDD.

Le DDD se concentre sur deux éléments du projet : le métier et le code source. L'objectif est de mettre en avant les préoccupations du domaine, c'est-à-dire du métier, et sa logique

ainsi que de refléter sa complexité sur un modèle. Ce modèle est une représentation du domaine ciblée. Il permet d'organiser l'information et de synthétiser la mécanique de raisonnement sur le domaine. Ce modèle n'est pas un diagramme en particuliers mais c'est l'idée que l'on cherche à transmettre par ce diagramme. Cette idée est une abstraction rigoureusement sélective et organisée de la connaissance de l'expert métier. La création de ce modèle est donc source de collaboration et de discussion entre équipe technique et expert fonctionnel. Il s'agit d'introduire l'intelligence du métier dans le code. Bien sûr ce modèle ne sera pas complet dès le début et nécessite des itérations. Plus nous avançons dans ce processus cyclique, plus le savoir métier est divisé et organisé en sous éléments, ce qui permettra leur traitement l'un après l'autre. Ici notre modèle est centré sur l'*Order*. Il se base en grande partie sur les différentes étapes dans le processus de gestion d'un ordre, de la décision du gérant à l'exécution et la terminaison de l'ordre (voir annexe 2). Le DDD va former des ponts entre équipe technique et équipe fonctionnelle car ce modèle doit être constitué d'un langage de communication commun aux deux équipes. Ce langage commun, pierre angulaire du DDD, forment ce qu'on appelle l'*Ubiquitous Language*. Ces termes se retrouveront tels quels dans le code source et dans toute communication quelle soit orale ou écrite. Pour les développeurs, il n'y a pas de méthode particulière pour s'approprier ces termes et acquérir les compétences métier. L'interview d'expert fonctionnel peut être une solution. De simples discussions vont permettre de dédramatiser la complexité du domaine. L'appropriation du vocabulaire par les personnes qui n'y sont pas habituées doit se faire en douceur. Utilisateurs et membres de l'équipe projet ont par exemple travaillé à la définition de *Order Sate* et *Order Status*, c'est-à-dire l'état et le statut d'un ordre, en créant un schéma (voir annexe 3).

Après avoir vu cette manière de concevoir une application il est temps de se rapprocher de son développement avec la méthode de behavior driven development.

3.3 Le behavior driven development

Le Behavior Driven Development (BDD) est une méthode agile encourageant elle aussi la collaboration entre les différents acteurs d'un projet informatique, particulièrement les développeurs, les testeurs et les intervenants non-techniques comme les utilisateurs. C'est un concept défini en 2003 par Dan North comme une réponse au Test Driven Development (TDD) qui est une technique de développement informatique préconisant d'écrire les tests unitaires avant d'écrire le code source d'une application. Grossièrement le développeur commence par écrire le test qui échouera dans un premier temps car il n'y a aucun code source. Puis il va développer le programme qui fera passer le test au vert. Le BDD vise principalement à

simplifier la compréhension de la finalité du TDD. En effet le développeur veut savoir par où commencer, savoir ce qu'il faut tester et ne pas tester, comprendre les échecs, et il ne peut pas répondre à ces questions tout seul. Le BDD permet de rendre plus cohérents ces tests car il donne à l'utilisateur un rôle prépondérant.

Le BDD met en avant les notions de comportement et de fonctionnalité d'une application. Il permet de lier les demandes de l'utilisateur au travail du développeur et de vérifier son bon fonctionnement. En effet le BDD intervient lors de l'écriture des cas de tests. Il va permettre à tous les intervenants du projet, qu'ils soient fonctionnels ou techniques, de se comprendre via un langage non technique, omniprésent et utilisé par tous.

Concrètement cela se matérialise dans un premier temps par une demande exprimée par le métier, formalisée de la façon suivante :

En tant que *< Gérant, Trader, ...>*
Je souhaite *< consulter la NAV des fonds, calculer mon bonus, ...>*
Pour *< faire les opérations d'ajustement nécessaire, connaitre mes performances, ...>*

De cette manière sont posés : le contexte, la fonctionnalité demandée et la finalité de ce besoin. Cela permet de bien exposer la valeur de cette fonctionnalité car nous devons nous poser la question de savoir pourquoi nous voulons faire ça, ce que ça nous apporte. Ainsi la demande est mise à l'épreuve et le besoin détaillé. Le métier a la tâche de réfléchir à des cas de test correspondant à sa demande.

Puis dans un second temps le développeur, le testeur et le représentant du métier vont discuter et établir les cas de tests. Nous constatons ici que le BDD tend à rapprocher les acteurs du projet afin qu'ils échangent leur avis et prennent une décision ensemble. De la même façon que la demande du métier, les cas de tests sont formalisés avec un langage composé de mots clés. Ce langage est appelé **Gherkin**. Il est spécifique au domaine c'est-à-dire au métier et donc compréhensible par l'utilisateur de l'application. Il décrit le comportement de la fonctionnalité sans détailler son implémentation. Il se présente comme suit :

Given *<Le contexte>*
When *<L'action exécutée, l'évènement, ...>*
Then *<La vérification>*

En formulant ainsi le test, son implémentation est pilotée par le besoin de l'utilisateur et il est compréhensible par tous.

Un détail important dans la rédaction du nom des méthodes de test est l'emploi du mot *'should'* (conditionnel de doit). Celui-ci permet deux choses : d'abord de ne pas se disperser au sujet de ce que doit faire la classe pour laquelle nous rédigeons le test, puis de poser un débat et un questionnement constant. En effet, le modèle de phrase « la classe doit/devrait faire quelque chose » signifie que nous ne pouvons définir un test, correspondant à un comportement, que pour la classe actuelle. Cela permet de se demander si le comportement est bien à sa place, si c'est bien cette classe qui doit avoir cette fonctionnalité.

En plus de sélectionner des méthodes plutôt techniques nous avons mis en place d'autres méthodes plus en lien avec la conduite et la gestion de projet.

3.4 Le daily stand-up meeting

Il s'agit de mettre en place une réunion quotidienne, rapide (généralement pas plus de 15 minutes) rassemblant équipe technique et équipe fonctionnelle. L'objectif principal de cette réunion est de faire le point sur l'avancement du projet mais aussi sur les difficultés rencontrées jusqu'à présent. Chacun leur tour les acteurs du projet prennent la parole et informent les autres sur les tâches accomplies depuis la dernière réunion. Cela parait très simple à première vue mais nous allons préciser la méthode car cette initiative peut vite s'avérer inutile si elle n'est pas claire pour tous les intervenants. Pour commencer la rapidité et un certain niveau d'énergie concentreront les équipes vers la bonne direction. Trop de longueur et un manque de dynamisme pourrait jusqu'à pénaliser la journée. Les participants doivent faire preuve de solidarité, notamment dans la remontée des problèmes. La résolution des obstacles est une chose mais pour cela il faut que ces obstacles soient partagés. Aucune des personnes qui les rencontrent ne doit craindre d'en faire part aux autres. Le discernement des personnes engagées et des personnes impliquées est primordial. Les personnes engagées sont celles qui contribuent à l'avancement du projet comme les développeurs, alors que les personnes impliquées ne sont là qu'en tant qu'observateurs de cet avancement. Cela va permettre de bien délimiter le périmètre des sujets abordés lors de la réunion. Un sujet ne doit pas être trop large, au risque d'allonger la réunion et de perdre l'attention de la plupart des participants, mais pas trop précis non plus, car nous aurions un sujet trop technique ne concernant que quelques participants. La contribution des personnes engagées doit tourner autour de trois axes :

- **Qu'ai-je accompli hier ?**
- **Que vais-je faire aujourd'hui ?**
- **Quels sont les obstacles qui m'empêchent d'avancer ?**

C'est le minimum abordable lors de ces réunions et c'est souvent amplement suffisant. En donnant ces informations les différentes parties s'engagent les unes envers les autres et communiquent sur la respectabilité de leurs engagements. Tenir ces réunions devant le backlog que nous avons décrit plus tôt est un excellent moyen de captiver l'attention des participants en leur offrant un véritable visuel.

Dans notre cas, les daily stand-up meetings se tenaient tous les jours à 10h pour l'équipe projet à leur bureau. Un jour sur deux les futurs utilisateurs, soient deux gérants et une personne de la table de négociation, étaient présents. Il en a été décidé ainsi car comme nous nous situions en tout début de projet il n'était pas possible de montrer des points d'avancement autre que purement technique, ce qui n'intéresse pas les utilisateurs finaux.

Enfin nous allons décrire ce qu'est un environnement propice aux échanges et à la communication entre les acteurs d'un projet informatique.

3.5 Privilégier les développements à proximité et à temps plein

Nous allons décrire maintenant un environnement organisationnel qui va faciliter non seulement la mise en place des méthodes décrites précédemment mais également aider la communication en générale au sein du projet. Ces préconisations sont inspirées de la méthode *crystal clear* d'Alistair Cockburn. *Crystal clear* concerne les petites équipes et se veut fortement adaptable aux spécificités de chaque projet.

D'abord les équipes fonctionnelles et les équipes techniques doivent être à proximité physique l'une de l'autre. La meilleure des situations seraient de les avoir dans la même pièce ou le même open space mais c'est impossible car cela signifierai que les équipes devraient déménager à chaque fois qu'elles passent d'un projet à un autre. Ce qui compte c'est qu'elles puissent se rencontrer facilement et que le laps de temps qu'elles réservent à se concerter ne soit pas gaspiller par le trajet jusqu'au lieu de rencontre. Les méthodes décrites précédemment réclament que les rencontres entre les acteurs du projet aient lieu à fréquence régulière mais aussi qu'ils puissent se voir de manière moins conventionnelle, un peu dans l'urgence. Les développements offshores sont à proscrire totalement car la communication s'estompe avec la distance et malgré les outils de vidéoconférence le décalage horaire sera toujours présent.

Une seconde préconisation concerne le temps que les développeurs peuvent consacrer au projet. Dans un monde idéal, tous les membres de l'équipe projet pourraient consacrer tout son temps à un unique projet. Mais ils doivent en général répondre à d'autres demandes et réaliser en plus des tâches quotidiennes qui n'ont pas de rapport avec le projet. La gestion de

projet doit avoir pour but qu'ils se dispersent et s'éloignent le moins possible du projet. En cas de retard il est préférable de plus concentrer ses ressources sur la raison du retard plutôt que d'ajouter de nouvelles ressources. En effet la *loi de Brooks* nous prédit que d' « ajouter des personnes à un projet en retard accroît son retard ». La mobilisation des ressources à disposition doit donc se faire le plus possible, le mieux étant totalement, sur le projet. L'implication des personnes dans le projet ira de pair avec leur faculté à communiquer sur le projet.

Un dernier point concerne la taille de l'équipe projet et la limitation des intermédiaires. *Crystal clear* conseille d'avoir une équipe de 2 à 7 développeurs. Cette restriction permet de mieux souder l'équipe. Si les personnes sont plus à même de tisser des liens dans leur mission elles communiqueront plus facilement ensemble. Je me suis intéressé au cas particulier d'intermédiaire que l'on trouve régulièrement dans les projets informatiques : l'assistance à maîtrise d'ouvrage (AMOA). L'AMOA ne doit pas être mis en place pour supprimer toute difficulté de dialogue entre MOA et équipe projet en ne les faisant plus parler ensemble. Au contraire il est très important de continuer à les faire communiquer. L'AMOA doit aider à formaliser les demandes MOA de sorte que l'ensemble des intervenants comprenne les services demandés. Il est très important de juger de l'utilité de ce type d'intervenant. Si la MOA et l'équipe projet se comprenne en faisant force de communication l'AMOA n'est pas nécessaire et sa présence sera au contraire un frein à une bonne communication au sein du projet.

Nous avons désormais fait la liste des préconisations apportées au projet OMX afin d'améliorer la communication entre tous les acteurs du projet. Dans une dernière partie nous allons faire le bilan le bilan de ces préconisations avant de conclure.

4 Retour d'expérience et bilan

A ce stade il est nécessaire de décrire plus précisément le contexte dans lequel évolue le projet OMX. Pour rappel l'équipe projet est composée d'un chef de projet, de deux développeurs à temps-plein sur le projet et d'un expert fonctionnel. Il est important de remarquer le profil d'un des développeurs. Celui-ci est particulièrement familiarisé avec le métier car il a déjà développé un OMS et possède une grande expérience en finance. Des utilisateurs finaux participent également au projet : une personne de la table de négociation et deux gérants.

4.1 Le backlog, pas si simple

L'efficacité du backlog en termes de vision du projet est indéniable. Le format tableau est vraiment préféré par les participants au projet à tout autre format numérique comme un fichier Excel par exemple, et il est bien plus consulté. Il offre un vrai support aux présentations sur l'avancement du projet et l'interactivité qu'il offre en pouvant déplacer les tâches inscrites sur les bouts de papier motive les conversations entre acteurs.

Néanmoins se servir d'un backlog n'est pas sans difficulté. La première de ces difficultés est le découpage des éléments du backlog et des tâches à accomplir. Ces tâches doivent avoir un aspect « atomique » c'est-à-dire qu'une tâche ne doit correspondre qu'à une et une seule exigence. Il ne faut pas qu'elles puissent être subdivisées en plusieurs petites tâches. Cela demande une vraie réflexion entre tous les acteurs du projet. Ils doivent trouver le bon niveau de granularité pour que le backlog ait un sens pour tout le monde. La deuxième difficulté est dans la projection des tâches futures à accomplir. Celles-ci ne doivent pas trop être éloignées dans le futur car plus c'est le cas, plus l'objectif va devenir flou et vague pour les acteurs du projet et plus ces prédictions seront fausses car elles ne prennent pas en compte les évènements exceptionnelles qui arrivent dans un projet. Cette difficulté a eu plus de mal à être surmonté pour le projet OMX. En effet pour obtenir un budget dans une structure comme Lyxor AM ou n'importe quelle grande société, les porteurs du projet doivent donner un planning précis s'étendant jusqu'à la fin du projet. Ce type de prédiction va à l'encontre de l'agilité. D'abord parce que la véracité de ces prédictions est toute relative et ensuite parce qu'elles vont disperser l'équipe projet qui va oublier les objectifs à court terme et les exigences prioritaires.

4.2 Le DDD, un bon moyen de partager le savoir

Le DDD apporte à la compréhension par les intervenants techniques du métier des utilisateurs finaux. L'utilisation des termes qu'ils emploient dans le code source peut même offrir une monté en compétences fonctionnelles des équipes techniques. Le code source de l'application n'est plus simplement du code mais exprime réellement ce que fait l'application pour les utilisateurs. La création d'un modèle ainsi que de l'*Ubiquitous Language* oblige les pôles techniques et fonctionnels à collaborer et se mettre d'accord sur les termes qu'ils utiliseront l'un comme l'autre, un arbitrage est nécessaire. Cette méthode apporte donc communication et compréhension aux actuels acteurs du projet mais aussi aux possibles futurs intervenants car ceux-ci arriveront dans un environnement où chaque élément a bien été défini. Le DDD développe donc également la réutilisabilité, la maintenabilité et la simplicité de l'application.

Dans le cadre du projet OMX le gros avantage est le niveau des compétences fonctionnelles d'un des développeurs qui permet de gagner beaucoup de temps. Son niveau est tel que c'est parfois l'utilisateur qui a besoin d'éclaircissement (voir annexe 4). De plus ce développeur a un profil d'architecte et donc un très bon niveau en conception logiciel. Dans des cas plus classiques le développeur n'a pas ou peu de connaissances fonctionnelles. Il faut quand même privilégier un profil expérimenté en conception logiciel pour appliquer le DDD. L'appropriation du langage des utilisateurs par le développeur ne se fera qu'au prix d'une grande implication des deux parties et l'aide des experts fonctionnels.

4.3 Le BDD, trop onéreux

Dans notre cas le bilan de l'utilisation du BDD est plutôt mitigé et ce pour plusieurs raisons. Nous allons exposer les points positifs et les points négatifs de cette méthode.

Le BDD aide et encourage significativement la collaboration entre les représentants du métier et l'équipe qui va développer le projet. Ils sont poussés à réfléchir ensemble sur ce qui est le plus important et le plus essentiel que l'application fasse. En effet le plus grand bénéfice du BDD est la recherche de valeur ajoutée pour les fonctionnalités et donc l'application. Cette méthode est donc très bénéfique pour le projet mais nous avons constaté qu'elle est très coûteuse.

Le BDD est une méthode qui demande à ceux qui l'utilisent plusieurs années d'expériences avant de s'en servir convenablement. Il faut techniquement se familiariser avec la syntaxe Gherkin mais également bien comprendre la philosophie du BDD et des tests en informatique.

La rédaction de bonnes demandes et des cas de test qui lui sont associés exige beaucoup de temps qui est généralement une ressource rare dans un projet informatique. L'avantage dans notre cas est que l'équipe projet est particulièrement rompue au métier. Certains éléments connaissent autant le métier que les utilisateurs. Malgré cela seuls les cas nominaux de test, c'est-à-dire les situations normales, où tout se passe bien, ont pu être rédigés car ce sont les plus simples. S'intéresser aux cas « bizarres » aurait demandé plus de ressources en temps et/ou effectifs car cela demande de réaliser des simulations.

Le BDD est donc une méthode de développement qui pousse à la collaboration et offre une véritable plus-value à l'application mais qui demande beaucoup de ressources. En fin de compte pour le projet OMX le BDD aurait pu ne pas être employé sans perturber les échanges et la compréhension au sein du projet.

4.4 Le daily stand-up meeting, très apprécié

D'après l'ensemble des retours c'est le daily stand-up meeting qui a récolté le plus d'avis positifs. Les participants ont bien compris l'intérêt d'un tel dispositif et il est effectué de la bonne manière. Si des sujets méritant plus d'approfondissement sont abordés pendant le daily stand-up meeting, une réunion ultérieure est organisée avec les acteurs concernés. Les deux gérants et la personne de la table de négociation sont bien impliqués alors que nous n'en sommes qu'au tout début du projet. Ils ont dû tout de même faire face à de nombreuses abstractions vu que les développeurs n'avaient pas toujours quelque chose à leur montrer. Les participants ont exprimés qu'ils ont pu vraiment suivre globalement l'avancement du projet grâce à ces réunions mais aussi prendre du recul par rapport ce qui est fait par chacun des acteurs.

4.5 Un environnement plutôt favorable

L'environnement est plutôt favorable au projet OMX. En effet les utilisateurs finaux et l'équipe projet ne sont séparés que de deux étages ce qui permet même aux développeurs d'intervenir directement sur les postes des utilisateurs. Les deux développeurs ont été recrutés à temps-plein sur le projet OMX. Cette petite équipe a prouvé son efficacité mais de l'avis général un développeur supplémentaire aurait été le bienvenu afin d'accélérer les développements.

CONCLUSION

La problématique de ce mémoire était de trouver une solution aux problèmes de communication entre les acteurs d'un projet informatique en vue d'améliorer leur compréhension de ce projet. Grâce au backlog, un simple tableau auquel nous accrochons de petits bouts de papier, la visibilité de l'avancement du projet est étendue et nous avons un point de rencontres et de conversations entre ces acteurs. Grâce au daily stand-up meeting, ces rencontres se font tous les jours et sont de véritables moments de partage des réussites comme des difficultés de chacun. Grâce au domain-driven design, les développeurs et les utilisateurs emploient le même langage et se comprennent. Les développeurs acquièrent des compétences fonctionnelles et place l'intelligence du métier dans le code de l'application. Grâce au behavior driven development, intervenants techniques et intervenants fonctionnels réfléchissent ensemble aux fonctionnalités les plus importantes de l'application par rapport à ce qu'elles réclament en termes d'implémentation. Enfin, grâce à un environnement adapté englobant proximité entre les acteurs du projets et temps à consacrer au projet, les interactions les uns avec les autres sont démultipliées. Seulement l'application de ces prescriptions dépend pour une grande partie de l'implication personnelle de chacun des acteurs. De plus de telles initiatives doivent être encouragées par la hiérarchie qui doit mettre à disposition des moyens, car j'ai pu constater que de telles méthodes, et même juste le fait que les gens communiquent plus, est couteux. Le temps consacrer aux échanges est forcément imputé à celui consacré aux tâches opérationnelles et pourtant il en résulte un vrai bénéfice pour le projet. Nous pouvons conclure que le véritable moteur de communication dans un projet est, avant toutes méthodes et préconisations, l'humain.

D'un point de vue personnel la rédaction de ce mémoire m'a permis de découvrir de nouvelles méthodes qu'on ne voit pas partout mais aussi de découvrir une réalité des projets informatiques que je n'avais pas eu l'occasion de voir lors de mes précédentes expériences. Mais d'abord j'aimerai revenir sur la place que j'occupai moi-même au sein de ce projet. N'ayant pas les compétences techniques suffisantes pour participer au développement du projet (et de toutes façons n'ayant pas été recruté chez Lyxor AM en tant que développeur) et n'ayant pas les compétences fonctionnelles pour intervenir sur le projet, je me trouvais là qu'en tant qu'observateur des usages de toutes les méthodes qui ont été décrites dans ce mémoire. Observation que je pouvais seulement réaliser quand j'étais présent en entreprise. Je pense

que cet éloignement au projet se ressent dans mon mémoire et que la solution proposée reste assez impersonnelle. Je suis tout de même satisfait d'avoir vu ces méthodes sur un tel projet et j'espère avoir l'occasion dans ma vie professionnelle de les revoir, que ce soit du côté technique comme du côté fonctionnel (car moi-même je ne sais pas encore vers quoi aller) et je réalise le véritable intérêt qu'elles ont. J'ai pu mieux comprendre ce que pouvais représenter un projet informatique pour tous les acteurs grâce aux conversations que j'ai pu avoir avec chacun d'eux. Une telle expérience m'a rapproché des processus que l'on trouve dans une société de gestion d'actifs et plus particulièrement du processus de gestion d'un ordre. Je compte d'ailleurs continuer à évoluer dans un environnement Front-Office pour la suite de mon projet professionnel.

Enfin, je garde à l'esprit que d'autres sujets et problématiques auraient pu être intéressants pour ce mémoire notamment dans la gestion des petits développements que j'ai réalisé à la table de négociation tout au long de cette année et que l'on peut considérer comme de petits projets informatiques. N'y auraient-ils pas des méthodes à appliquer à ce type de mission ?

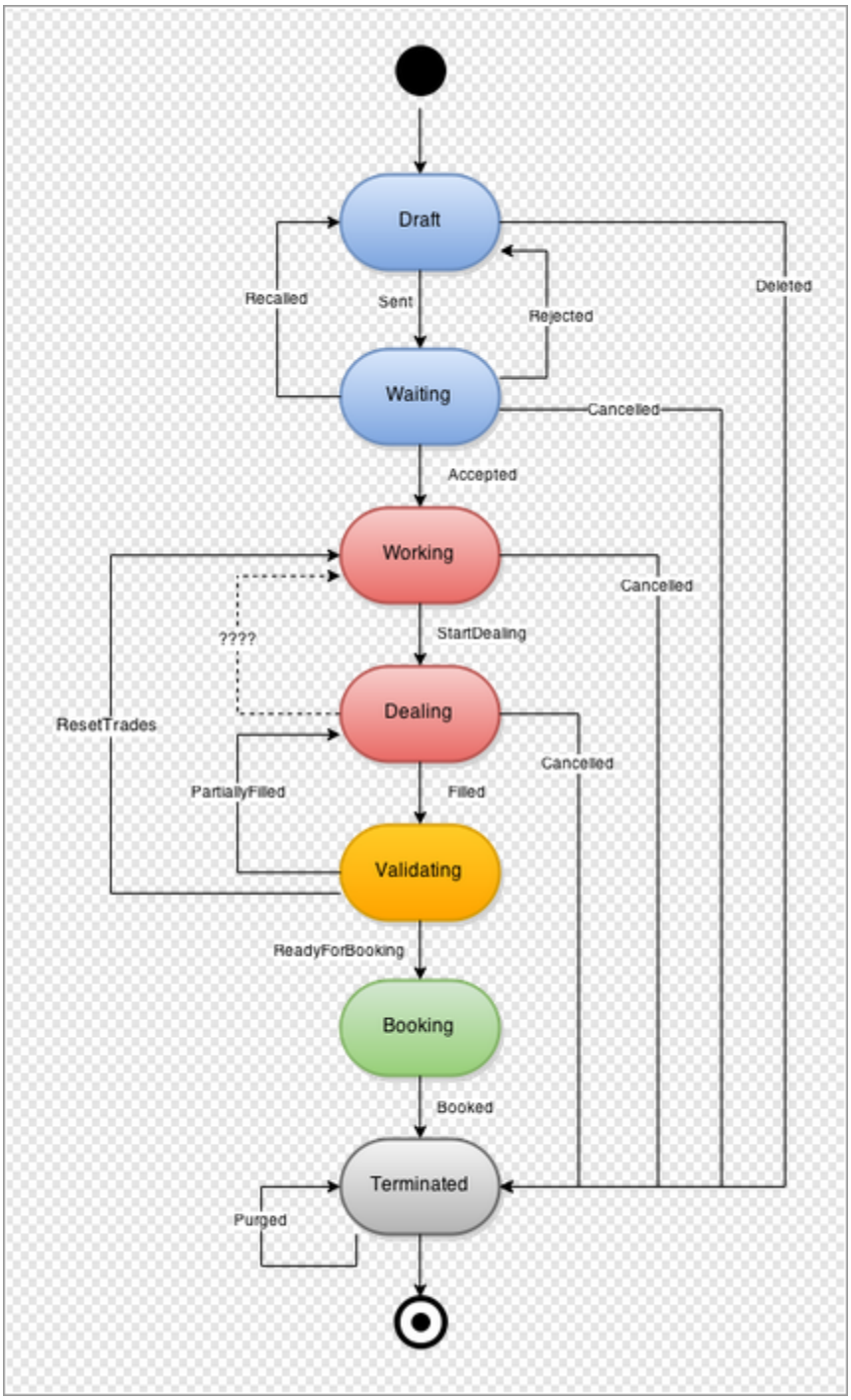
BIBLIOGRAHIE

- *Chaos Manifesto 2013*, Standish Group
- *Top 10 Reasons Why Systems Projects Fail*, Paul Dorsey
- *A Guide to the Business Analysis Body of Knowledge (BABOK)*, Kevin Brennan
- *The Mythical Man-Month*, Frederick Brooks
- *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Eric Evans
- *BDD in Action: Behavior-driven development for the whole software lifecycle*, Dan North

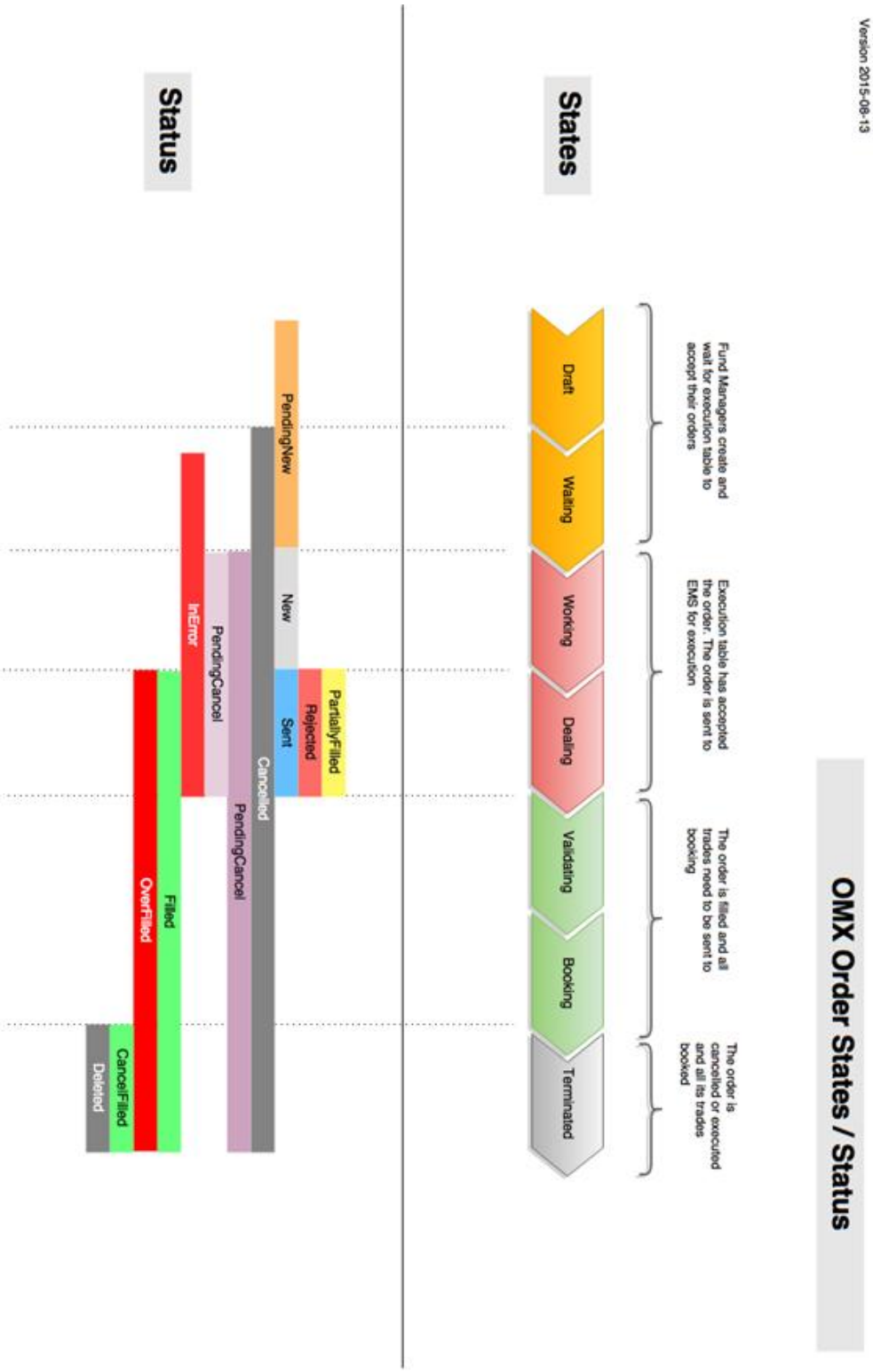
ANNEXES

Table des annexes :

Annexe 1 : Le blacklog

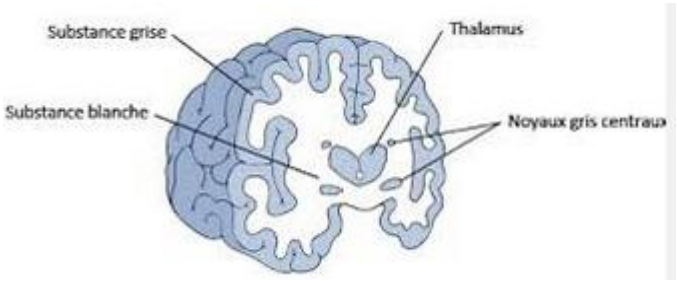


Annexe 2 : Le modèle du domain-driven design



Annexe 3 : Schéma *Order Sates / Order Status*

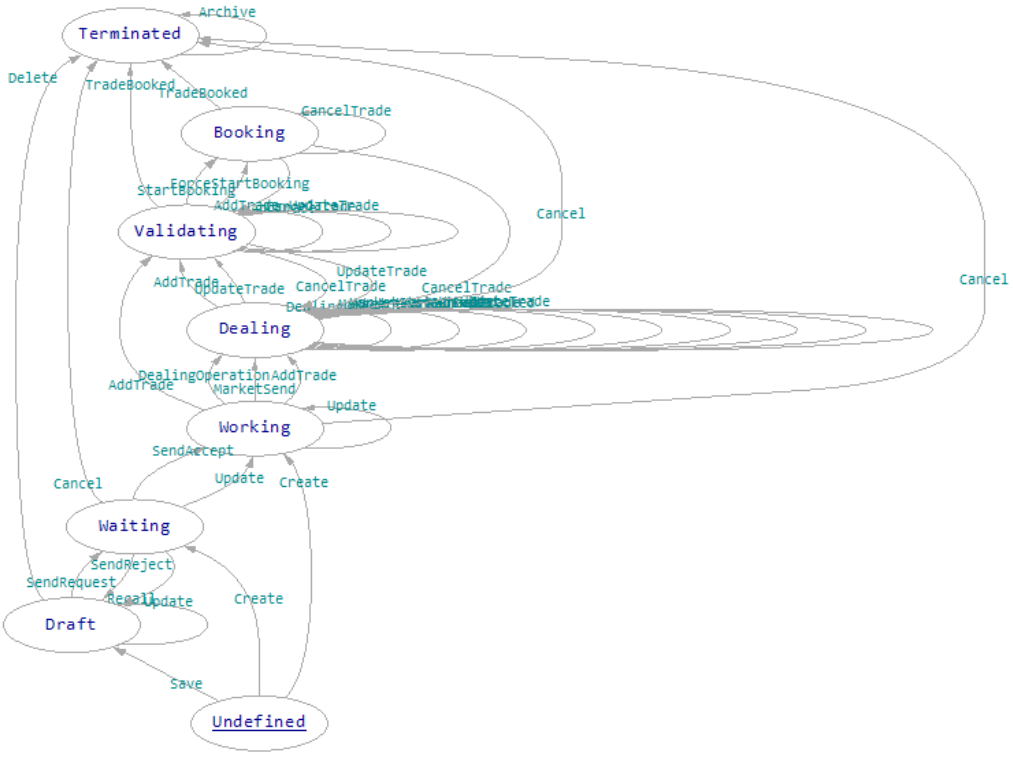
From: RAVETIER Stephane LyxrNeg
Sent: Thursday 13 August 2015 17:52
To: LANGEAC Bertrand (EXT) ItecCttLyx; COMBES Cyrille LyxrNeg
Cc: KHADRAOUI Taher (EXT) ItecCttLyx
Subject: RE: [OMX] Un schema pour comprendre comment travaillent les utilisateurs



Désolé je n'ai pas trouvé la bonne zone du cerveau pour comprendre… mais merci quand même ! ☺

From: LANGEAC Bertrand (EXT) ItecCttLyx
Sent: Thursday 13 August 2015 17:46
To: RAVETIER Stephane LyxrNeg; COMBES Cyrille LyxrNeg
Cc: KHADRAOUI Taher (EXT) ItecCttLyx
Subject: RE: [OMX] Un schema pour comprendre comment travaillent les utilisateurs

En fait ca se résume en terme de machine à états par (sans trucage) :



Annexe 4: Des problèmes de compréhension