Le projet **DELIRE**

Développement par **E**quipe de **L**ivrables **I**nformatiques et **R**éalisation **E**ncadrée

PR1 – Organisation de la phase de réalisation

La phase de Réalisation

La phase de réalisation est une phase qui a trois caractéristiques

1. Elle est coûteuse. Comme elle a été structurée pour que chacun puisse travailler de façon indépendante, elle permet de maximiser la productivité.
2. Elle est courte. Dans le projet DELIRE elle dure approximativement un mois
3. La qualité de ce qui est produit durant la phase de réalisation est un point clé pour pouvoir ensuite mener à bon port la phase de convergence

L'objectif de cette phase est « relativement » simple : permettre à chacun d'implémenter et de tester son (ou ses) composant logiciel conformément aux spécifications, et ce de façon indépendante du travail de ses collègues.

Nous allons revenir ici sur ces quelques points

Gérer le coût

La phase de réalisation est courte mais intensive. Il est important pour chacun de monter en puissance dès le début de cette phase.
Ne pas le faire, et donc retarder le moment où les problèmes vont être mis en exergue, va engendrer des retards préjudiciables au projet par effet boule de neige :

1. Du fait de ressources limitées (difficile de consacrer plus de 24 heures par jour) vous risquez de ne pas pouvoir rattraper votre retard en fin de phase, et de prendre l'ensemble du groupe en otage, puisque la phase d'intégration et de convergence ne pourra pas démarrer, par absence de votre composant
2. Si vos spécifications sont incomplètes, incohérentes voire non implémentables, le surcoût engendré pour rectifier la situation risque de vous faire exploser le coût de l'opération et donc la durée de la phase. Il faudra au minimum redéfinir l'objectif de ce qui va être implémenté.
3. De même, si le dimensionnement des tâches se révèle irréaliste, et en général il est souvent trop optimiste, il est important de corriger au plus tôt ce problème pour refixer un objectif cohérent pour le groupe et éviter ainsi de laisser se développer des composants qui ne seront finalement pas intégrés dans le projet
4. Si vous êtes en retard, le chef de projet devra prendre la décision de confier une partie du travail à un autre membre de l'équipe. Si celui-ci découvre que ce retard est du à une flemme généralisée, l'ambiance risque de se corser dans le couple.

Commencez par les tâches prioritaires (c'est-à-dire celle qui seront nécessaires pour démarrer des premiers tests d'intégration).
Et dans ces tâches commencez par les plus complexes :

1. A priori elles vous ont été confiées parce que vous avez le savoir faire pour les réaliser, autant garder les tâches plus simples pour celui ou celle qui viendra potentiellement vous aider en cas de retard.
2. Si ces tâches se révèlent impossibles à accomplir, par exemple pour un manque de compétences techniques du à une difficulté supérieure à ce qui avait été anticipé, il sera toujours temps de modifier l'architecture retenue pour le projet

Gérer le délai
Chacun d'entre vous doit avoir son planning personnel pour la phase de réalisation
Ce planning est composé

1. De la liste ordonnée des tâches
2. De la durée (nombre d'heures de travail) de chaque tâche
3. De la date prévue de début, de la date prévue de fin

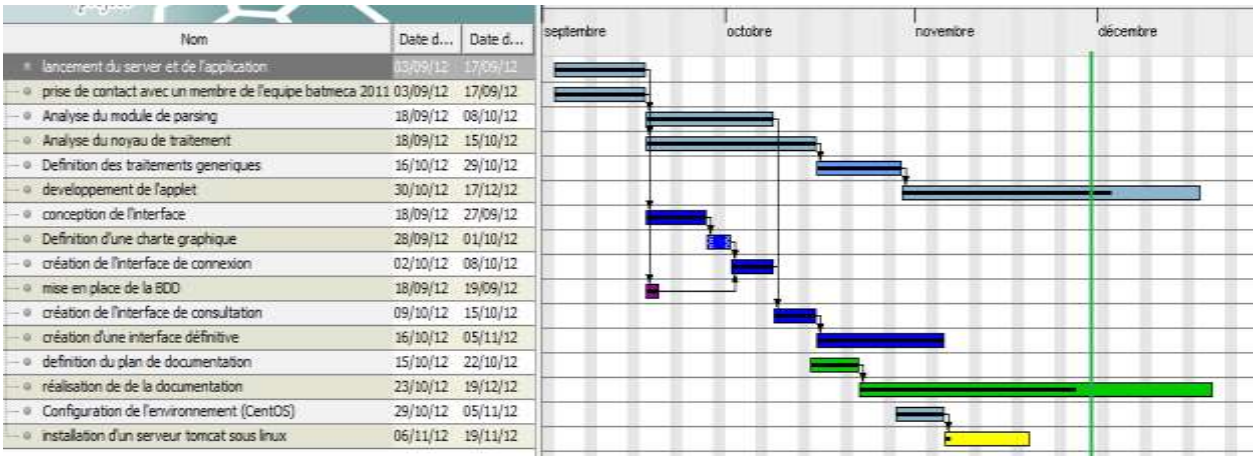
Il y a un ordre, respectez-le. Attention à ne pas papillonner de tâche en tâche, en commençant tout et en ne finissant rien : c'est la pire méthode pour pouvoir faire une estimation du travail réalisé et de ce qui reste à faire. Si une tâche non prévue apparait nécessaire pour la suite du projet, avertissez sans délai l'architecte et le chef de projet : ce peut être mineur, mais ce peut être également annonciateur de gros problèmes.

Il y a une durée : vérifiez-la. Et si la durée est notoirement différente de la prévision, remontez l'information :

1. Les prévisions sont peut -être trop optimiste : il va falloir réduire le scope du projet
2. Certaines tâches se révèlent finalement plus simple que prévu : il faut définir comment employer de façon judicieuse ce nouveau buffer

Il y a une date de début et une date de fin, respectez-les. Et si besoin, redéfinissez-vous un nouveau planning réaliste compte tenu de ce qui s'est passé jusqu'alors. Conformément à l'adage « passé les bornes, il n'y a plus de limite », je dirai que si le planning de référence est irréaliste et si un retard certain s'est déjà accumulé, alors être encore plus en retard ne pose guère de problème : de toute façon on n'y arrivera pas.

Ayez toujours votre planning à jours, c'est un outil de communication puissant et efficient.



Nom	Date d...	Date d...
lancement du server et de l'application	03/09/12	17/09/12
prise de contact avec un membre de l'equipe batmeca 2011	03/09/12	17/09/12
Analyse du module de parsing	18/09/12	08/10/12
Analyse du noyau de traitement	18/09/12	15/10/12
Definition des traitements generiques	16/10/12	29/10/12
developpement de l'applet	30/10/12	17/12/12
conception de l'interface	18/09/12	27/09/12
Definition d'une charte graphique	28/09/12	01/10/12
création de l'interface de connexion	02/10/12	08/10/12
mise en place de la BDD	18/09/12	19/09/12
création de l'interface de consultation	09/10/12	15/10/12
création d'une interface définitive	16/10/12	05/11/12
definition du plan de documentation	15/10/12	22/10/12
réalisation de de la documentation	23/10/12	19/12/12
Configuration de l'environnement (CentOS)	29/10/12	05/11/12
installation d'un serveur tomcat sous linux	06/11/12	19/11/12

Gérer la qualité

Faites tout ce qui vous incombe

Au moment où vous ferez la promotion de votre composant, il faut que vous ayez le sentiment du devoir accompli. Et bien accompli.

1. Le code fonctionne
2. Les Unit tests fonctionnent tous
3. Les règles de programmation ont été respectées.
4. Les test de performances unitaires ont été effectués
5. Les fuites mémoires ont été éradiquées,
6. Les règles d'architectures, les cas d'erreurs ont été validées

Bref, le coding n'est que la partie émergée de l'iceberg.

Mais ne faites que ce qui vous incombe

Votre objectif est de réaliser votre composant, et non pas celui des autres. Vos Unit tests ont pour objet de valider la qualité de ce que vous avez écrit, non de réaliser l'ensemble du projet. En d'autres termes faites-vous un banc de tests pour simuler les autres composants

Un banc de tests doit être facile, pas cher, et rapporter gros.

1. Une base de données est simulée par un simple tableau en mémoire
2. Un service renverra de temps en temps des erreurs, en fonction du nombre d'appel ou de l'heure d'appel.

Pensez à l'avenir.

Même s'il est peu probable que le code développé durant le projet DELIRE va être commercialisé, faites comme si cela devait être. Les commentaires que vous mettrez dans votre code ne sont pas destiné à vous aider à comprendre votre code dans les 15 jours qui suivent, mais lorsque vous, ou un de vos collègues, aurez à faire de la maintenance dans votre composant dans plusieurs mois voire plusieurs années. C'est là où vous regretterez votre astuce très certainement géniale mais imbitable parce que non documentée.

Les Unit tests que vous écrivez vous serviront :

1. Pour valider la qualité et la complétude de votre composant en phase de réalisation
2. Pour valider le système en tests d'intégration durant la phase de convergence
3. Chaque fois que vous serez intervenu pour faire une correction dans votre composant, en phase de convergence puis plus tard en maintenance sur le marché.

Ils vous serviront également :

1. Pour rechercher les fuites mémoires
2. Pour identifier les éventuelles dérives de performances
3. Pour prévenir les éventuelles régressions en cas d'évolution du composant.

En d'autres termes, l'investissement en ingénierie dans les Unit tests est un investissement à long terme.

Assurez-vous que vous testez tout : toutes les branches des algorithmes doivent être couvertes.

Assurez-vous que vous testez tous les cas d'erreurs. En informatique la loi de Murphy règne en maitre : c'est toujours el cas qu'on a pas testé qui pose problème

en clientèle. Et n'utilisez pas le fait que vous ne pouvez pas tester un cas d'erreur car le composant appelé ne renvoie ladite erreur : votre banc de tests a pour objectif de vous permettre de tout tester
Commencez par faire le design et écrire le code de vos tests avant d'attaquer le code de l'application proprement dit : cela vous permettra de mettre en évidence certaines variables qu'il sera nécessaire de pouvoir suivre.

Le banc de tests va certes vous servir pour valider la qualité et la complétude de votre composant en phase de réalisation. Mais vous serez amenés à modifier votre coding soit en phase de convergence soit plus tard dans le cadre de la maintenance de l'application. A chaque fois, la règle est de revalider le composant lui-même avant d'intégrer cette nouvelle évolution dans le système

Vous l'aurez compris, il y a 3 parties de code à réaliser en phase de réalisation :

1. Le code de l'application proprement dit
2. Le code des Unit tests
3. Le code du banc de tests

L'investissement en qualité pour chacune de ces parties de code est un très bon placement à court, moyen et long terme.
Au fait, j'ai oublié de vous demander si dans votre planning les taches de développement des Unit tests et du banc de test ont été prévu : n'oubliez pas qu'elles ne sont pas négligeables en coût vis-à-vis du code de l'application

Le 2nd principe de la thermodynamique

Ne remettez pas à demain les règles de qualité dans l'écriture de votre code :

1. Pourcentage et qualité des commentaires
2. Lisibilité du code
3. Respect des normes de programmation (nommage des variables, indentation…)
4. Programmation défensive, test des valeurs reçues, clarté des écritures en mode trace….

On remet toujours à plus tard ce qui semble un peu superflu : je le ferai quand j'aurai le temps. Dans la réalité, vous n'aurez jamais le temps, et ce n'est qu'en pleine crise de maintenance que vous prendrez conscience de l'importance de faire bien du premier coup.

Il n'y a pas que le code en phase de réalisation

Vous avez certes votre code à écrire. Mais vous avez surtout à le faire fonctionner et à le vérifier. Lorsque vous allez promouvoir votre code (promotion qui inclut les Unit tests et le banc de tests), vous devez avoir fait tout ce qui était humainement possible pour éradiquer les erreurs potentielles de votre composant.

La phase de convergence commence par une phase d'intégration où on teste la cohérence des appels entre composants. Si les erreurs sont noyées dans le bruit des erreurs intrinsèques à un composant donné, c'est peut être très agréable pour le responsable dudit composant, qui s'est évité ainsi l'écriture de Unit tests puis leur exécution, mais c'est désastreux pour l'équipe en terme de productivité.
Je dis souvent que la paresse est la première qualité d'un ingénieur : c'est-à-dire minimiser l'énergie consommée. Et dans le cas présent, les erreurs trouvées en

phase d'intégration plutôt qu'en phase de réalisation coûtent beaucoup plus cher à identifier, à comprendre et à corriger.

Pour que l'intégration se passe bien, il est nécessaire que les appels de services inter-composants soient cohérents avec les interfaces desdits services. A priori, ces interfaces ont été définies en phase d'architecture logique, et ne devraient aps être remis en cause. Si néanmoins, vous identifiez un manque dans une des interfaces, la correction doit intervenir très tôt en phase de réalisation. Ce n'est pas quand le code des autres composants sera finalisé et que les bancs de tests auront été réalisés qu'il faudra arriver avec un sourire béat voire benêt sur les lèvres et annoncer que vous avez totalement restructuré la partie publique de votre composant.

Vous êtes face à vous-même… mais dans un projet
Le projet a été découpé pour que chacun puisse faire son travail indépendamment des autres. A vous de réaliser ce qui vous incombe sans pouvoir vous reposer sur les autres. Mettez-vous à l'ouvrage sans tarder, en pensant que la part qui vous a été attribuée est équilibrée, réalisable et à votre portée.

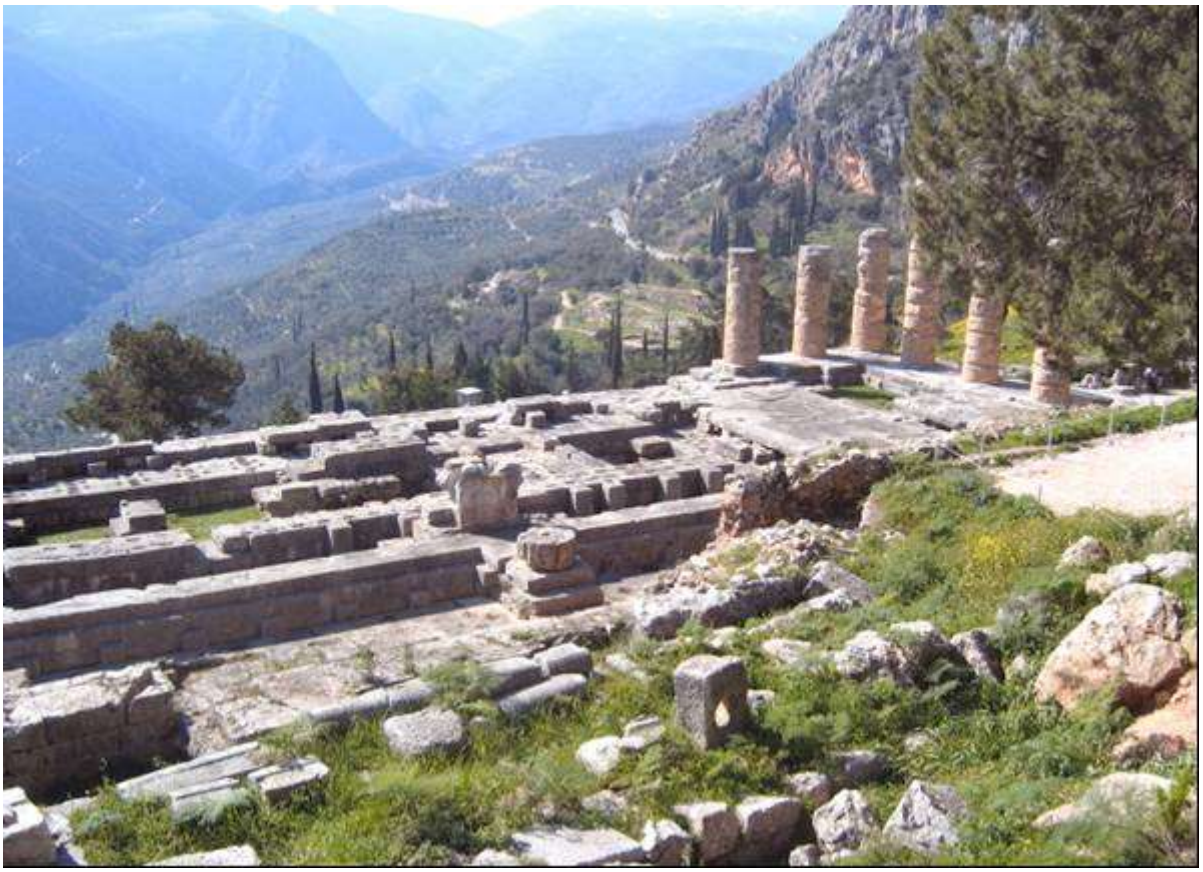
Ceci étant posé, l'objectif est de réaliser un projet commun, et non une compétition. Pas de honte si vous êtes en retard, dites-le, tout simplement. Pas de fierté si vous êtes en avance, profitez-en pour filer un coup de main aux autres. Et surtout ni jugement ni compétition : c'est en équipe qu'on gagne, et c'est dans l'échange qu'on grandit.

Vous aurez le sentiment d'en avoir fait plus que les autres : tant mieux, vous les aurez peut-être fait progresser par votre façon de travailler. De toute façon sachez que dans un groupe on a souvent l'impression d'en faire plus que les autres.

Vous aurez le sentiment que les autres vous ont plus apporté que vous ne leur avez apporté : tant mieux, vous les avez fait progresser dans leur façon de transmettre leur savoir faire, ce qui leur servira lorsqu'ils deviendront chef de projet dans leur carrière.

Connais-toi toi-même (Γνῶθι σεαυτόν)…
Chacun fonctionne comme il l'entend.
Si vous êtes capable de vous isolez dans votre chambre pour travailler 2 heures chaque soir sur le projet, c'est très bien.
Si vous ne supportez pas d'être isolé dans votre chambre 2 heures par soir, ce n'est pas non plus dramatique, vous n'êtes sans doute pas seul. Il peut être astucieux d'organiser de séances de travail collectif : non pas au sens de tout le monde touche à tout, mais où chacun travaille sur son composant en ne se sentant pas isolé. Et cela permet lorsqu'on a un doute ou une interrogation d'avoir rapidement la réponse. Mais pour ce faire, il faut que vous annonciez très tôt la couleur au chef de projet.

… Et tu connaitras l'univers et les dieux
Oui, vous allez sans doute trouver que la phase de réalisation n'est pas si enthousiasmante que ce que vous imaginiez : respect des spécifications, manque de créativité, respect du planning, suivi des coûts, contrôle de la qualité…
Mais lorsque, la réalisation étant terminée, vous entamerez la phase de convergence et que celle-ci se déroulera bien mieux que prévu, alors vous prendrez conscience de ce que le travail de spécification et de structuration, puis la rigueur dans le travail de réalisation ont été payants.



Un projet qui converge en Délai, en Coût et en Qualité est un moment suffisamment rare et jouissif dans la vie pour être gouté avec volupté.

Le saviez-vous

On a tendance à baptiser comme premier développeur informatique Grace Hopper (qui termina sa carrière comme amiral de la Navy).

Faux, la première personne à écrire du code fut Ada Lovelace, fille de Lord Byron.
Mais à l'époque où elle a écrit ce code, les ordinateurs n'existaient pas.
Pour votre information, c'est cette Ada qui a donné son nom à un langage informatique (et non une entreprise de location de voiture).
D'autre part, le code qu'elle avait écrit a été compilé il y a quelque année, et il fonctionnait le bougre. Bravo !