Gestion Agile de Projets IT

Author	Frédéric TRIBEL frederic.tribel@gmail.com
Date	16-09-2015

URL http://users.skynet.be/tribel.f/ipl/IPL-2015-2016-Gestion%20de%20Projets.pdf

Table des matières

Table des matières détaillée

0. Préliminaires

0.1. Audience

- 3ème baccalauréat en Informatique
- Participants à un projet IT
- Futur chefs de projets IT

0.2. Applicabilité

Tout projet I.T.

0.3. Objectifs

Ce cours explique les bases de la gestion Agile de projets informatiques. Il s'agit d'une introduction générale destinée aux étudiants de 3ème baccalauréat en Informatique, introduction qui n'a pas la prétention ni d'être exhaustive, ni de former des chefs de projet professionnels.

Il est conçu pour être expliqué en trois ou quatre sessions de 2 heures.

Les références mentionnées ont été choisies pour être facilement accessibles et dans leur grande majorité, gratuites.

0.4. Conventions

0.4.1. Symboles

Ce document n'utilise pas de symbole particulier.

0.4.2. Style et forme

Les exercices sont énoncés sur fond grisé.

0.5. Evaluation

L'examen sera écrit et comprendra des questions ouvertes et à choix multiples.

0.6. En cas d'erreur

Merci de signaler les coquilles ou toute autre erreur par email à
mailto:frederic.tribel@gmail.com?subject=GESPROJ3:coquille

1. Introduction

1.1. Gérer un projet

Référence: ESA Std PSS-05-08

Gérer un projet informatique c'est le planifier, organiser, pourvoir en personnel[1], suivre, piloter, diriger et contrôler[2]

Le rôle du Chef de Projet c'est de:

- Produire le plan de projet
- Définir les rôles et les pourvoir en personnel
- Piloter le projet et informer la/les équipe(s) de leur part dans le plan global
- Diriger le projet en prenant les décisions importantes et en motivant les membres du/des équipe(s)
- Contrôler le projet en mesurant le(s) progrès et les coûts
- Informer les sponsors, clients, et autres parties prenantes

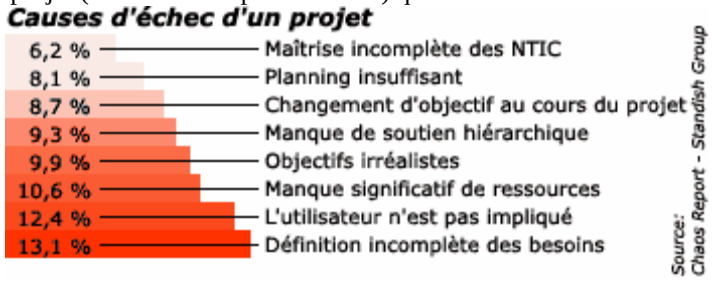
Mais avant tout, qu'est-ce qu'un projet ?

Un projet est un ensemble de tâches ou d'actions effectuées par une ou plusieurs personnes pour satisfaire un besoin d'un mandant.

Les cinq conditions suivantes sont nécessaires pour parler de projet:

- Le mandant (le client interne ou externe, le demandeur) doit exister et être clairement identifié
- Un chef de projet doit être nommé
- Le but est connu et formalisé
- Le délai est connu et formalisé
- Le budget est connu et formalisé

Il est plus facile de rater un projet (totalement ou partiellement) que de le réussir:



1.2. Ce dont nous allons parler

- Voir Table des Matières

1.3. Ce dont nous n'allons pas parler

- Comment faire la conception ("Software Design")
- Comment faire la construction ("Software Construction")
- Comment tester ("Software Testing")
- Comment faire la maintenance ("Software Maintenance")
- Comment gérer la configuration ("Software Configuration Management")
- Comment gérer la qualité ("Software Quality Management")
- Comment pourvoir en personnel
- Quels outils de gestion de projet utiliser
- Quel langage utiliser
- Quel outil de développement utiliser
- Comment gérer les changements dans les organisations ("(organizational) change management")
- Comment gérer l'évolution d'un logiciel ("software change management")

[1] "staffing"
[2] "controlling"

2. Notions de base en gestion de projets IT

2.1. Méthodes classiques

Il existe un très grand nombre de manières de gérer un projet, un très grand nombre de méthodes à la fois différentes et semblables.

- IEEE
- ESA
- Prince2 ([3])
- …

Toutes les méthodes "classiques" partagent certaines caractéristiques:

- Le projet est découpé en phases
- Les phases sont exécutées dans un ordre bien précis

2.2. Références

Software Engineering Body of Knowledge Book (SWEBOK)
http://www2.computer.org/portal/web/swebok/htmlformat (★★)
IEEE standards
http://standards.ieee.org/reading/ieee/std_public/description/se/index.html (★)
ESA standards
http://www.esa.int/TEC/Software_engineering_and_standardisation/TECBUCUXBQE_0.html (★★★)
En particulier le PSS-05-08 mais aussi Bssc962, PSS-05-0, PSS-05-01, PSS-05-02, PSS-05-03, PSS-05-04

2.3. Eléments-clefs

Pour bien gérer un projet IT et assurer son succès, il est très important d'identifier quelques éléments-clefs

- **Objectifs**
  Ce qu'on veut réaliser
- **Finalités**
  Pourquoi veut on réaliser ce projet
- **Déliverables**
  Ce qu'on s'engage concrètement à donner au "client".
- **Parties prenantes** ("stakeholders")[4]
  Les parties prenantes sont toutes les personnes ou groupes de personnes dans ou en dehors de l'organisation qui ont un intérêt à ce que le projet réussisse ou échoue, ou qui ont ou peuvent avoir une influence positive ou négative sur le déroulement du projet
- **Méthode de gestion de projet**
- **Budget(s), délai**
  Si c'est opportun, bien différencier contraintes et cible
- **Risques**
  Quels sont les risques ?
  pendant la phase projet ?
  pendant le déploiement ?
  Et que risque-t-on si on réussit[5] ?
- **La gestion du changement**

[3] http://en.wikipedia.org/wiki/PRINCE2
[4] Recherchez la définition sur wikipedia !
[5] Pensez aux conséquences négatives du projet, ou pouvant-être vue comme négatives par certains (par exemple: "si ce projet réussi, je perdrai mon boulot")

2.4. Découpes en phases

Références: ESA: PSS-05-0.pdf et PSS-050-08.pdf

Voici un exemple de la découpe en phase prévue par l'ESA – European Space Agency:

Phase		Description	Output
UR	User Requirements Definition Cf PSS-05-02	Cette phase définit le problème. On décrit (donc par écrit) les besoins des utilisateurs. Si nécessaire, on vérifie la compréhension des besoins par des prototypes	URD: User Requirements Document SPMP: Software Project Management Plan v1
SR	Software Requirements Definition Cf PSS-05-03	La phase d'analyse. On décrit un "modèle" de ce que le logiciel doit faire (le "quoi") mais pas le "comment".	SRD: Software Requirements Document SPMP: Software Project Management Plan v2
AD	Architectural Design Cf PSS-05-04	L'objectif est de décrire la structure du logiciel, ces différents composants et leurs interactions (le "comment").	ADD: Architectural Design Document SPMP v3
DD	Detailed Design and Production	L'objectif est de détailler la conception du logiciel, de coder, implémenter, documenter et tester	Code, DB, … DDD: Detailed Design Document SUM: Software User Manual
TR	TRansfer	L'objectif est de vérifier que le logiciel réponds aux besoins spécifié&s dans le URD, entre autre via des tests de qualification ("acceptance tests")	STD: Software Transfer Document
OM	Operations and Maintenance	La vie proprement dite de l'application	

Chaque phase à des activités principales, des "délivrables" et se clôture formellement ("reviews").

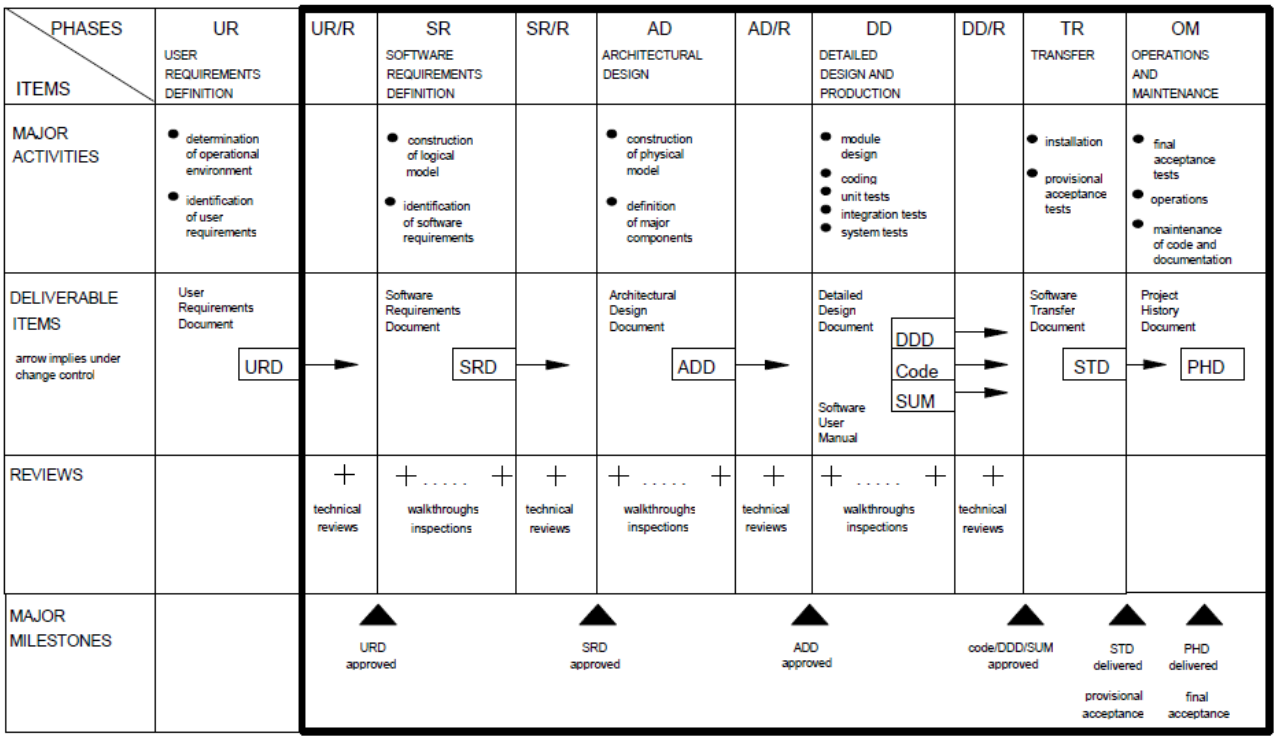
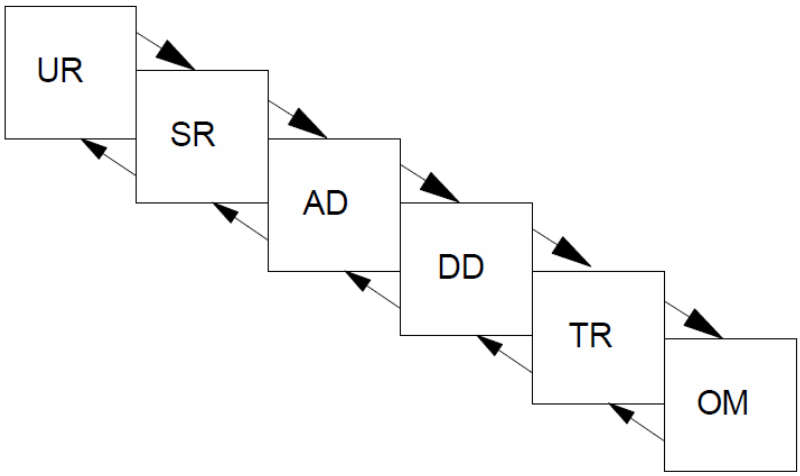


Figure 1.2: The Software Life Cycle Model

2.4.1. Approche Chute d'eau

L'approche la plus classique est dite en "chute d'eau". Les phases sont exécutées l'une après l'autre, dans l'ordre.
Si on c'est trompé, en recommence

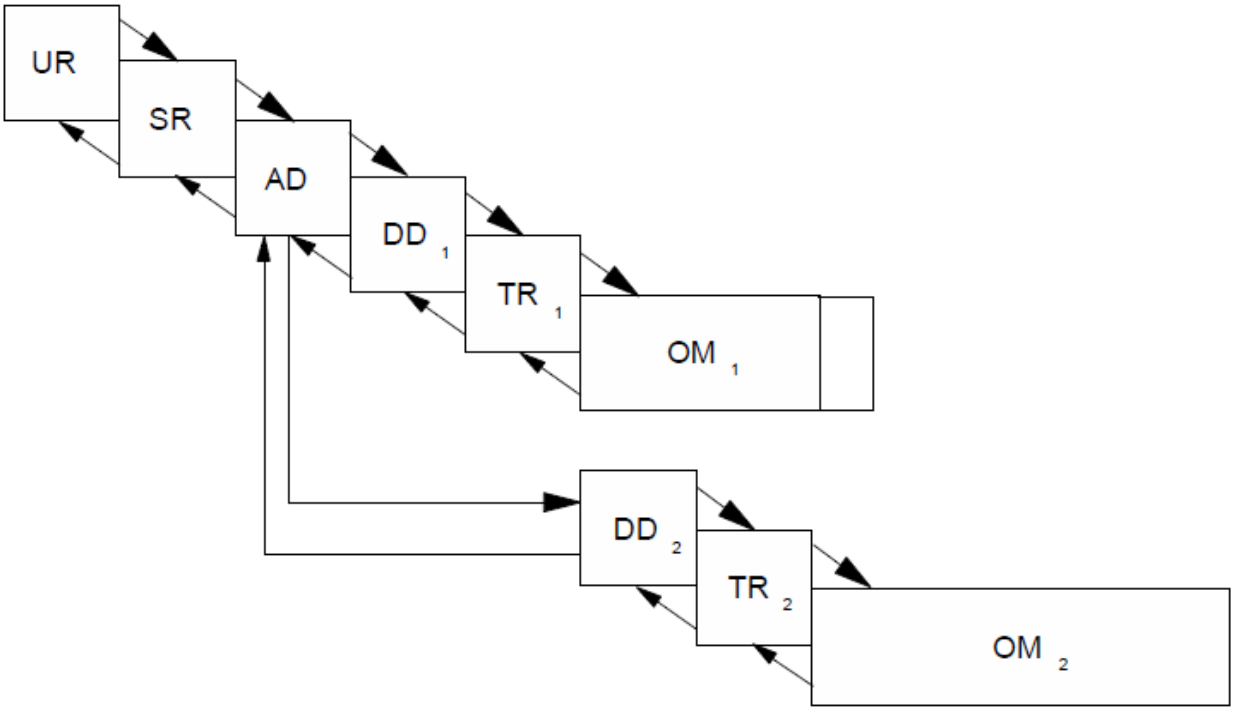


Cette approche est bien adaptée lorsque:

- Il est possible de formaliser correctement et précisément les besoins utilisateurs et ceux-ci ne changeront pas.
- Le projet est relativement court
- L'application complète doit être produite en une fois.

2.4.2. Approche incrémentale

Une autre approche classique est dite "incrémentale". Les phases DD, TR et OM sont démultipliées.

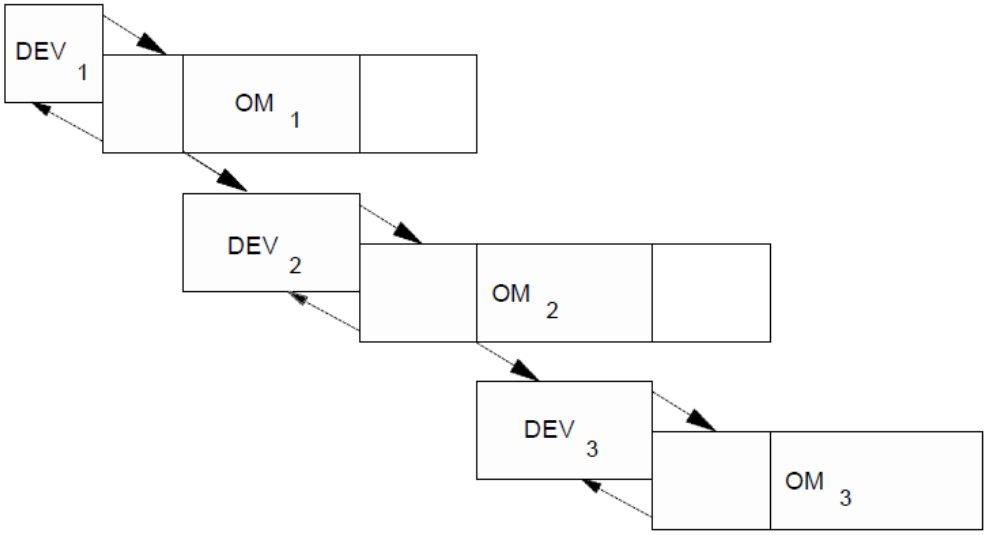


Cette approche est bien adaptée lorsque:

- L'application peut être livrée/produite en "morceaux" ou "versions" distinctes.
- Le projet complet serait trop long
- Il est nécessaire d'avoir des preuves que le logiciel sera accepté (acceptable ?) avant la fin du projet complet

2.4.3. Approche itérative

L'approche itérative ne devrait a priori pas être classée comme une approche classique. Elle sera couverte dans le chapitre du cours consacré aux méthodes dites "modernes".
Elle peut être vue comme composée en de multiples cycles de développement "en chute d'eau" avec un recouvrement entre les phases TR et OM.



2.5. Vocabulaire

2.5.1. Planning ou Planification

Référence: http://fr.wikipedia.org/wiki/Planification

La planification est la mise en œuvre d'objectifs dans le temps:

- dans un domaine identifié
- avec des objectifs précis
- avec des moyens
- sur une durée, et des étapes, précises

2.5.2. Ordonnancement

Référence: http://fr.wikipedia.org/wiki/Th%C3%A9orie_de_l%27ordonnancement

Ordonnancer consiste à organiser dans le temps la réalisation de tâches connues pour atteindre un objectif identifié, compte tenu de contraintes temporelles, telles que les durées estimées et les contraintes d'enchaînement, et des contraintes de ressources.

Les problèmes d'ordonnancement peuvent être résolus par l'intuition (s'ils sont très simples) ou par la recherche opérationnelle.

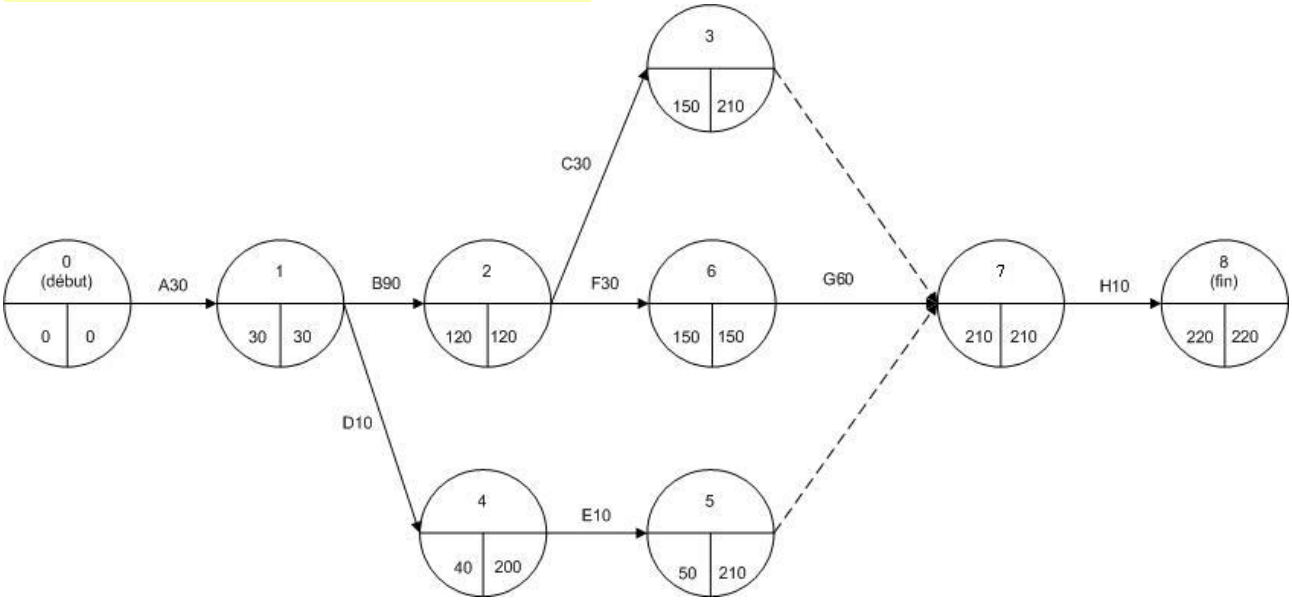
2.5.3. Contraintes d'enchaînement

		A -> B
SF	Start-to-Finish	On doit commencer B avant d'avoir fini A
SS	Start-to-Start	On ne peut commencer B que si on a déjà commencé A
FS	Finish-to-Start	On ne peut commencer B que si on a déjà complètement fini A
FF	Finish-to-Finish	On ne peut finir B qu'après avoir fini A

2.5.4. Pert

Référence: http://fr.wikipedia.org/wiki/R%C3%A9seau_PERT

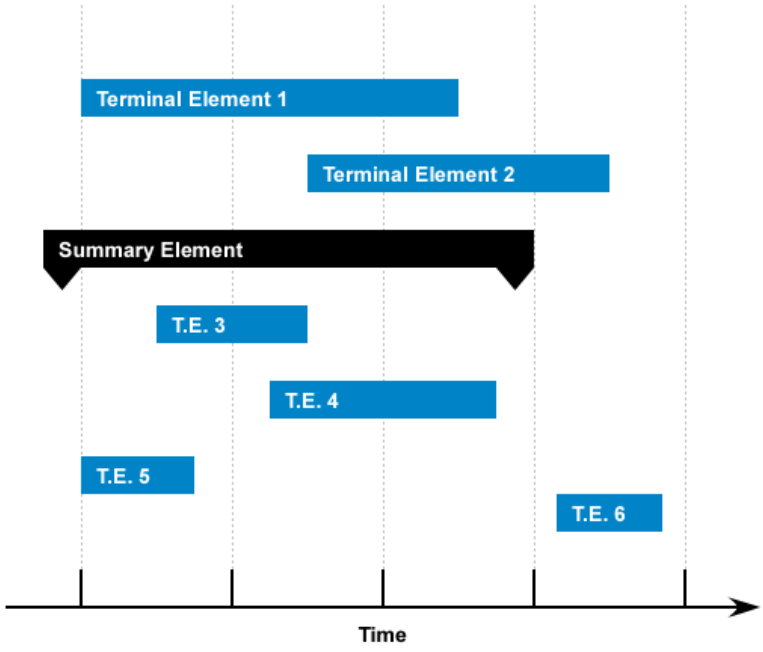
Un graphique ou diagramme de PERT permet de visualiser les relations entre une série de tâches. Chaque tâche est représentée dans un plan. Si la tâche A est représentée à gauche de la tâche B, c'est que la tâche B est commencée après la tâche A. Des flèches représentent les relations entre les tâches: si une flèche va de la tâche A à la tache B, c'est que la tâche A doit être finie avant de commencer la tâche A.



2.5.5. Gant

Référence: http://fr.wikipedia.org/wiki/Diagramme_de_Gantt

Un diagramme de Gantt permet de visualiser dans le temps une série de tâches. Chaque tâche est représentée dans un plan, typiquement par un rectangle. L'axe horizontal représente le temps, de gauche à droite (le futur). L'axe du temps doit être gradué et l'échelle explicite (absolue ou relative). La longueur de chaque rectangle est directement proportionnelle à la durée de la tâche. Les abscisses des cotés du rectangle indiquent les instants de début et de fin prévu.



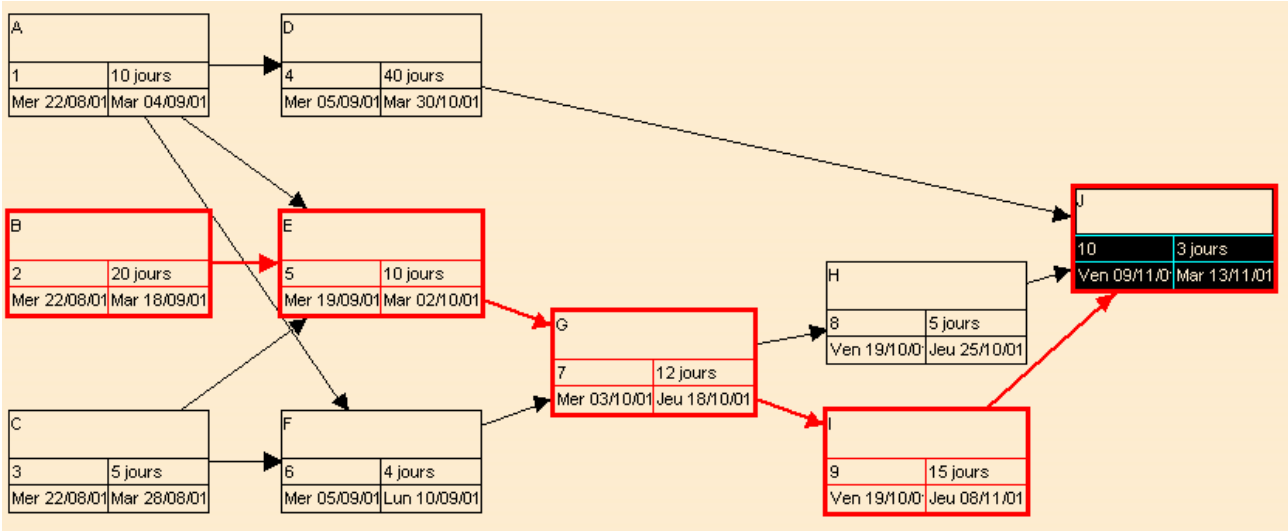
2.5.6. Nivellement de capacité

Dans les premières étapes de l'ordonnancement des tâches d'un projet, certaines ressources seront surchargées (les pics), d'autres sous-chargées (les creux). Dans beaucoup de cas, le projet n'est pas réalisable tel quel.
"Niveler" la capacité, c'est décharger les pics pour boucher les creux afin de réaliser le projet dans les temps

2.5.7. Chemin critique

Référence: http://www.e-miage.org/demos/B303/B303_4.htm#PERT

Succession des tâches se suivant l'un l'autre sans marge du début à la fin d'un projet. Tout retard sur une seule des tâches située sur le chemin critique compromet le délai final du projet.



2.5.8. RACI

Référence: http://fr.wikipedia.org/wiki/RACI

Une table RACI est utilisée pour spécifier les responsabilités des intervenants.

R	Responsible	Celui qui fait la/les action(s)
A	Accountable	Celui qui valide, qui approuve
C	Consulted	Est consulté
I	Informed	Est informé

Exemple:

Activités	Rôles: Sponsor	Chef de projet	Chef d'équipe	Développeur
Planning projet	A	R	C	I
Rédaction de la documentation technique	C	A	R	C
Développement		C	A	R
Tests unitaires		I	A	R

3. Gestion de projets « Agile »

Référence:

"**Agile Estimating and Planning**" by Mike Cohn, Prentice Hall, 2006 (★★★★)
http://fr.wikipedia.org/wiki/M%C3%A9thode_agile

Les méthodes de gestion de projet Agiles se veulent plus pragmatiques et flexibles que les méthodes traditionnelles. Il est clair que le concept de méthode Agile a démarré dans le domaine du développement d'applications informatiques mais aujourd'hui, un mouvement plus large lies les valeurs "Agiles" aux techniques de l'amélioration continue de la qualité (voir Management Agile, Lean Management, ...).

Ce chapitre est dédié plus particulièrement à la gestion de projet "Agile" telle que décrite par Mike Cohn dans son livre "Agile Estimating and Planning", livre dont je recommande tout particulièrement la lecture (disponible sur amazon.com ou safaribooksonline.com).

Seuls les aspects suivants, tous relatifs à la gestion de projet, seront couvert ci-après:

		Dans ce document
Cahier des charges, spécifications	User Stories	Oui
Budget du projet	Evaluation de la taille	Oui
	Gérer la valeur	Oui
Planifier	Ordonnancer	Oui
	Gérer les risques	Oui
Réaliser	Implémenter	Oui
	Suivre et communiquer	Oui
	Tester, Valider	Non
	Analyser, Concevoir, Coder	Non
	Documenter	Non

3.1. Introduction

Le Manifeste Agile formalise en 2001 la notion de méthode "Agile". Il a été signé par 17 personnalités sur base de leur grande expérience en génie logiciel. Le manifeste met en évidence des valeurs et des principes.

3.1.1. Valeurs

- « **Personnes et interaction plutôt que processus et outils** »: l'équipe est beaucoup plus importante que les outils (structurants ou de contrôle) ou les procédures de fonctionnement. Une équipe soudée et qui communique bien, composée uniquement de développeurs même de niveaux variables est préférable à une équipe composée d'experts travaillant chacun dans son coin (ou domaine). La communication est une notion fondamentale.
- « **Logiciel fonctionnel plutôt que documentation complète** » : L'application qui fonctionne est l'objectif premier. Le reste, et notamment la documentation technique, est une aide précieuse mais non un but en soi, même si une documentation précise est utile. La documentation représente une charge de travail importante, mais peut être dommageable si elle n'est pas à jour. Il est donc préférable de commenter abondamment le code lui-même, et surtout de transférer et partager les compétences au sein de l'équipe.
- « **Collaboration avec le client plutôt que négociation de contrat** » : (collaboration) Le client doit être impliqué[6] dans le développement. On ne peut se contenter de négocier un contrat au début du projet, puis de négliger les demandes et les réactions du client. Le client doit collaborer avec l'équipe et fournir un feed-back continu sur l'adaptation du logiciel à ses attentes.
- « **Réagir au changement plutôt que suivre un plan** » : (acceptation du changement) La planification initiale, la méthode de gestion de projet (et la structure du logiciel) doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet. Les premières releases du logiciel vont très souvent être suivie de demandes d'évolution.

En une phrase:

l'équipe collabore et accepte le changement pour une application qui fonctionne

[6] Il est facile de se rappeler la différence entre "impliqué" et "concerné" si l'on pense à une poule, à un cochon et à une omelette au lard: dans l'omelette au lard, la poule est concernée mais le cochon, lui, est impliqué !

3.1.2. Principes du Manifeste Agile

- « Notre première priorité est de satisfaire le client en livrant tôt et régulièrement des logiciels utiles ».
- « Le changement est bienvenu, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client ».
- « Livrer fréquemment une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte ».
- « Les gens de l'art et les développeurs doivent collaborer quotidiennement au projet ».
- « Bâtissez le projet autour de personnes motivées. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail ».
- « La méthode la plus efficace pour transmettre l'information est une conversation en face à face ».
- « Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet ».
- « Les processus agiles promeuvent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment ».
- « Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité ».
- « La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle ».
- « Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent ».
- « À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens ».

3.1.3. Une approche Agile de la planification de projet

L'application des principes Agile conduit à une méthode de développement de logiciel qui est itérative, incrémentale et donne au client une application fonctionnelle, testée et documentée à la fin de chaque itération. Pour réussir, une équipe Agile:

- Travaille comme une équipe
- Travaille en itérations courtes
- Délivre quelque chose à chaque itération (en principe au client)
- Se focalise sur les priorités "business" (en principe, les priorités du client)
- Mesure et s'adapte

3.1.4. Travailler comme une équipe

Une équipe Agile travaille ensemble à la réussite du projet. Différents rôles doivent être identifiés: le **responsable produit**, le **client**, les **développeurs** et le **chef de projet**.

Rôle	Description
Responsable produit	Les responsabilités du responsable produit sont • d'être certain que tous les membres de l'équipe partagent une vision commune du projet, • de décider les priorités (voir chapitre 3.5) • de prendre les décisions nécessaires pour assurer la rentabilité du projet
Client	Le client est la personne qui finance le projet (interne ou externe). Autrement dit, c'est lui qui paye. Dans les projets conduits dans une entité pour elle-même, le client et le responsable produit sont souvent la même personne. Dans les projets visant à commercialiser un produit, le client ne peut pas être représenté par le responsable produit, il faut une personne différente ayant une bonne connaissance des futurs clients.
Développeurs	Développeur désigne n'importe qui développe ou contribue à la réalisation. • Programmeurs • Analystes • Ingénieur bases de données • Rédacteurs • Architectes • Designers • Testeurs • … Les rôles sont périodiquement permutés lorsque c'est possible.
Chef de projet	Le chef de projet Agile se concentre sur le "leadership" plus que sur le contrôle de gestion. Il fait en sorte que : • Tous les membres de l'équipe œuvrent ensemble à sa réussite, en ayant la même définition du concept de réussite • Les principes Agile sont connus et respectés, en particulier en favorisant l'autonomie et l'auto-organisation

	Les rôles de chef de projet et de développeurs peuvent être périodiquement échangés, mais pas trop souvent (toutes les 2 ou 3 itérations).

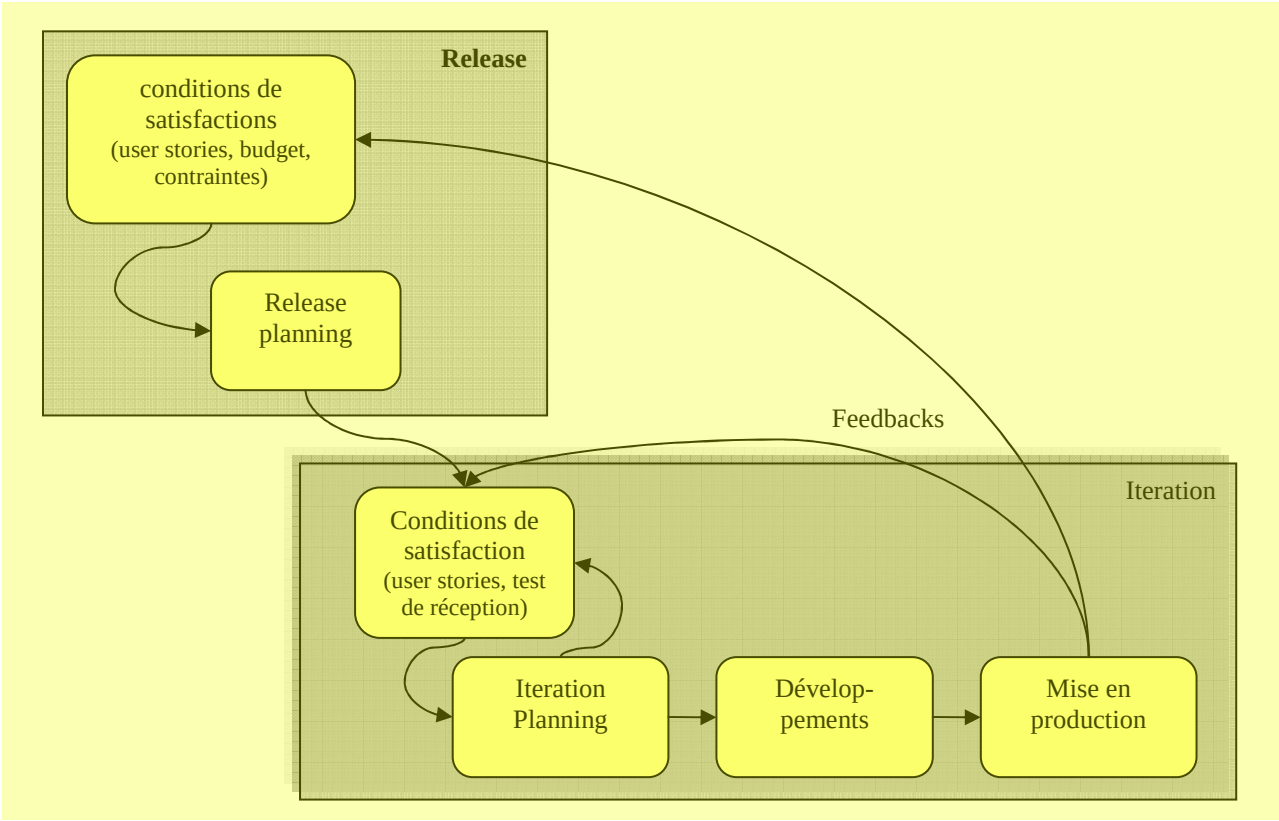
3.1.5. Travailler en itérations courtes

Un projet Agile se divise en une ou plusieurs **Releases**, divisées en plusieurs **itérations**.

Toutes les phases classiques (analyse, conception, codage, test,…) sont faites en parallèle durant chaque itération

Une itération implémente complètement plusieurs fonctionnalités. **La durée des itérations est limitée et constante.** Elle est choisie en début de projet, elle devrait toujours être comprise entre 2 semaines et 2 mois.
Une durée constante donne un rythme à l'équipe. Une durée courte évite le "syndrome de l'étudiant" ([7]).

Je conseille d'avoir au moins de 5 itérations dans une Release, adaptez la durée de l'itération à la complexité (ou plutôt la simplicité) du projet.



3.1.6. Délivrer quelque chose à chaque itération

Une itération implémente complètement plusieurs fonctionnalités afin de délivrer, à la fin de chaque itération, un logiciel qui **fonctionne**, qui **est testé**, **documenté** et **est potentiellement utilisable** par le(s) client(s)/utilisateur(s).
Chaque itération pouvant avoir un impact sur les fonctionnalités implémentées dans les itérations précédentes, il est très important de prévoir dès le tout début des tests les plus automatiques possibles (Unit Tests, tests de non régression, …).

A la fin d'une Release, le logiciel est complet, autonome, testé, documenté et utilisable. Il est envoyé au client.

[7] http://en.wikipedia.org/wiki/Student_syndrome et http://www.aubryconseil.com/post/2007/04/28/215-le-syndrome-de-l-etudiant

3.1.7. Se focaliser sur les priorités "business"

Priorité "business" = les priorités du client, pas celles du développeur.

1) Une équipe Agile délivre les fonctionnalités dans l'ordre spécifié par le responsable produit (qui l'a négocié avec le client), responsable produit qui a fixé ces priorités et combiné les fonctionnalités en Releases pour optimiser la rentabilité du projet.
2) Une équipe Agile concentre ses efforts pour compléter des fonctionnalités qui ont un sens et une valeur pour le client et les utilisateurs, plutôt que sur des tâches. La meilleure manière d'y arriver est de décrire les fonctionnalités sous forme de "User Stories" (voir chapitre 3.2).
3) Une équipe Agile implique le client dans chaque itération, et même avant. Le client est impliqué au jour le jour en particulier dans l'écriture des "user stories", dans la fixation des priorités, dans la planification du projet **et dans les tests.**

3.1.8. Mesurer et s'adapter

"Planning is everything. Plans are nothing."
Helmuth van Moltke, quoted by Mike Cohn

Le meilleur plan est souvent obsolète avant d'avoir été terminé, s'il est possible de le terminer. Beaucoup de paramètres changent continuellement, en particulier les désidératas des clients.

Une équipe Agile se concentre sur les prochaines itérations à délivrer et non sur un plan préétabli. Elle voit les changements comme des opportunités et adapte continuellement ses plans.
L'équipe Agile mesure ces progrès à la fin de chaque itération: quelles "user stories" sont complètement terminées (testées, documentées, utilisables, acceptées par l'utilisateur,…) ? Elle en calcule sa vitesse de progression.

Au début de chaque itération, l'équipe Agile tient compte des changements, de la nouvelle situation, des besoins changés du client et de sa vitesse de progression, pour planifier quelles fonctionnalités seront implémentées lors de l'itération (voir chapitre 0). Idéalement, le chef de projet fige les décisions (user stories, rôle, équipe, infrastructure) pour l'itération qui commence (itération N), tout en anticipant l'itération suivante (itération N+1) voire les 2 suivantes (N+1 et N+2). Tous les intervenants sont bien conscients que seule l'itération N est figée, les décisions concernant les itérations suivantes seront revues au début de chaque itération, tant de choses pouvant encore changer.

Si les progrès sont inférieurs à ce qui était prévu, l'impact sur les itérations suivantes et l'ensemble de la Release est calculé et annoncé formellement au client (voir chapitre 3.8.4).

3.2. User stories

Pour être compatible avec les principes Agile, en particulier pour permettre l'implication du client et des utilisateurs, les spécifications ne peuvent pas être écrites[8] n'importe comment. Bien qu'il soit possible d'utiliser de nombreuses formes, nous allons nous restreindre à les écrire sous forme de "user stories".

Références supplémentaires:

- "User Stories Applied" par Mike Cohn (édité par Addison-Wesley Professional)
- http://www.mountaingoatsoftware.com/presentations/63-an-introduction-to-user-stories
- http://www.mountaingoatsoftware.com/articles/27-advantages-of-user-stories-for-requirements

3.2.1. Définition

Une "user story" est une brève description d'une fonctionnalité, vue par un utilisateur ou un client. La forme n'importe pas trop mais, **il est** pratique et **vivement recommandé** d'utiliser la forme suivante:

> As a <type of user>, I want/must <capability> so that <business value>.

> En tant que <type d'utilisateur>, je souhaite/dois <fonctionnalité> afin de <objectif ou valeur business>.

Exemple: *en tant que professeur, je souhaite retrouver mon local via mon nom, le nom du cours et la date, afin de retrouver rapidement le local où je dois donner cours.*

Pratiquement, les user stories sont notées sur des fiches (l'idéal) ou dans une application adéquate[9]. Chaque user story a sa fiche, le recto pour l'histoire elle-même, le verso, pour y noter les "conversations".
Une "conversation" est une discussion entre l'équipe de développement et le client ou le responsable du produit, discussion au sujet de l'histoire. Elle sert à détailler l'histoire elle-même (par rapport à l'exemple ci-dessus, une conversation pour être: *que veux dire nom ? par nom, on entend nom et/ou prénom. En entier ? non, une partie de chaque mot suffit. Avec des "wildcards" ? non, pas nécessaire*).

3.2.2. Critères

Les user stories doivent vérifier les critères suivants :

Non ambiguë	Il ne doit y avoir qu'une seule interprétation. Si nécessaire, une ou plusieurs conversations permettront de lever les ambiguïtés
Concise	Une histoire doit être énoncée dans un langage qui précis, bref et, si possible, agréable à lire.
Indépendante	Une histoire doit être la plus indépendante, la plus autonome possible des autres histoires. C'est le critère le plus difficile à satisfaire. Voir chapitre 3.2.3
Négociable	Une histoire est un engagement négocié, mutuellement accepté et renégociable, entre l'équipe de développement et son client. Ce n'est ni une fonctionnalité arbitraire, ni un contrat figé.
Valorisable	Une histoire a de la valeur pour le client, pas pour les développeurs. Voir chapitre 3.5
Estimable	Une histoire doit être conçue de manière à permettre d'évaluer la charge de travail, la complexité de son implémentation. Voir chapitre 3.3
De la bonne taille	Une histoire ne doit pas être trop complexe ou trop longue à implémenter. Voir chapitres 3.2.3 et 0
Cohérentes	Une histoire ne doit pas contredire d'autres exigences établies, ni être contredite par d'autres exigences. Le vocabulaire utilisé doit être cohérent à travers tout le projet

[8] On ne fait pas des spécifications, on les écrit.
[9] Voir chapitre 3.14 ou encore un tableur

Vérifiable	Une histoire doit être formulée de manière à permettre de déterminer objectivement[10] si l'application est conforme ou non selon l'une des méthodes possibles : inspection, démonstration, ou test (en privilégiant toujours l'option "test automatique")

Truc: INVEST: Independent, Negotiable, Valuable, Estimatable, Sized appropriately, Testable

Toutes les spécifications ou pour être plus précis, toutes les exigences[11], quelles soient fonctionnelles, non-fonctionnelles, conceptuelles ou des contraintes, devraient être formalisées sous la forme de user stories. Dans tous les cas, "l'utilisateur" ou "le propriétaire" de l'histoire **doit** être indiqué. L'objectif ou la valeur "business" ne doit pas être indiqué si c'est une évidence, en tenant toutefois compte que ce qui est évident pour l'un ne l'est pas nécessairement pour l'autre.

Exigence	Description Exemple
Fonctionnelle	Une spécification fonctionnelle décrit les processus que l'application doit supporter. En France, on parle de "règles métier"
	En tant que professeur, je dois pouvoir modifier la description de mon cours. *En tant que secrétaire, je dois pouvoir modifier le local associé à un professeur, un cours et une certaine plage horaire*
Non fonctionnelle	Une spécification non fonctionnelle décrit les propriétés que le système doit avoir. Par exemple, en matière de sécurité informatique (confidentialité, intégrité,..), de performance, d'accessibilité
	En tant que professeur, je dois retrouver ma salle de cours en moins de 5 secondes, afin de ne pas être en retard au cours *En tant que professeur, je souhaite accéder à l'application via mon téléphone mobile afin d'y accéder de n'importe quel endroit, même sans ordinateur.*
Contrainte	Une contrainte est une spécification non fonctionnelle arbitraire, imposée par des règles externes ou internes. L'objectif est souvent manquant, arbitraire, caché,… ou légal
	En tant que chef de projet, je décide que nous utiliserons LINUX et que nous développerons en Eiffel. *En tant que responsable produit, je certifie que l'application doit être compatible avec le logiciel IUCLID 5 (contrainte légale).*
Conceptuelle	Une spécification conceptuelle définit un concept qui sera utilisé dans d'autres histoires. Il ne s'agit pas ici de définir un comportement de l'application, une fonctionnalité.
	En tant que responsable produit, je défini le concept de classe comme regroupant un cours, un professeur, un local, une heure de début, une heure de fin et un programme.

Veillez à prendre en compte des users-stories fonctionnelles mais aussi non-fonctionnelles, conceptuelles et ne pas oublier les contraintes !

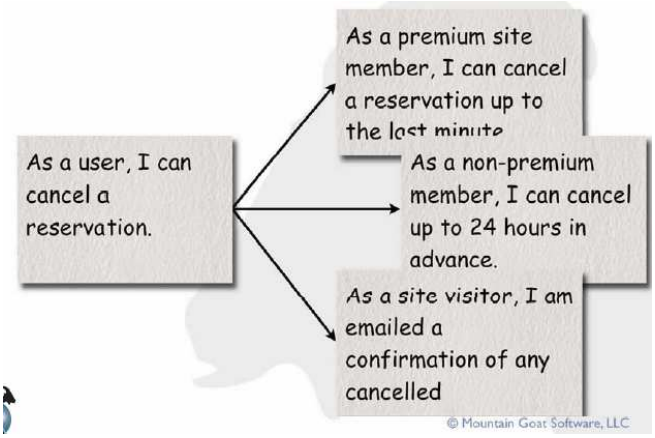
[10] Ne pas utiliser des formulations du type, rapide, assez rapide, convivial, … Toujours être mesurable objectivement.
[11] Voir sur http://fr.wikipedia.org/wiki/Exigence_(ing%C3%A9nierie)

3.2.3. Diviser une histoire

3.2.3.1. Quand diviser une histoire

On divise une histoire:

- Pour séparer des cas différents.
  Voir image ci-contre



- Parce qu'elle est trop grande pour être implémentée en une seule itération.
  Voir chapitres 3.3 et 0

- Pour isoler des fonctionnalités communes ("cross-cutting concerns").
  Par exemple, des fonctionnalités de traces, de traitement d'erreur, de sécurité,…

En tant que professeur, je dois retrouver ma classe sur base de mon nom, du cours, ou de l'heure
En tant que professeur, je dois retrouver ma classe sur base de mon nom, du cours, ou de l'heure, en moins de 5 secondes

- Pour séparer spécifications fonctionnelles et non-fonctionnelles, en particulier les spécifications de performances.

- Pour séparer des fonctionnalités de différentes priorités.

3.2.3.2. Quand ne pas diviser une histoire !

Dans aucun cas, une histoire ne doit être divisée en tâches nécessaires à son implémentation. Chaque histoire doit avoir un sens pour un utilisateur, par rapport à la finalité de l'application. Ne jamais concevoir des histoires du genre "créer la table", "créer les rapports", "tester",…

3.3. Evaluer la taille

Une équipe Agile fait la différence entre les estimations de "taille" et les estimations de durée.
Faire cette différence est extrêmement important: la "taille" est une mesure relative de la complexité d'une histoire alors que la durée est une mesure absolue qui dépend de nombreux facteurs.

Faisons l'analogie entre un projet (ou l'implémentation d'une histoire) et un voyage en voiture.
Pour planifier un voyage, tout le monde utilise les concepts suivants:

Distance totale	Km	Miles
Distance parcourue	Km	Miles
Distance restant à parcourir = distance totale – distance parcourue	Km	Miles
Temps écoulé	Jours	Days
Vitesse (moyenne) = distance parcourue / temps écoulé	Km/jour	Miles/day
Carburant consommé	Litres	Gallons
Consommation	Litre/100Km	Miles/gallon
Consommation journalière	Litres/jour	Gallons/days
Temps restant à parcourir = distance restante / vitesse	Jours	Days

Le temps et la distance sont bien 2 concepts séparés.

Un chef de projet qui ne gère que le temps ne peut pas savoir où il est ! S'il ne sait pas où il est, comment peut-il s'adapter à la situation présente. Comment dire à son client qu'il sera à temps ou pas ? Que le projet coutera plus cher que prévu ou pas ? Comment prendre la moindre décision sans savoir où l'on est ni à quelle vitesse on avance ?

Pour être agile, il faut[12] donc introduire dans la gestion de nos projets des concepts analogues à la distance et à la vitesse.

Voyage	Projet
Distance	Taille
Temps	Temps
Vitesse	Vitesse ou Vélocité

Ceci introduit 2 questions:

- Quelle unité de mesure utiliser pour la taille ?
- Comment évaluer la taille ?

N.B. La taille d'un projet est, par définition, la somme des tailles des histoires définissant le projet.

3.3.1. Unité de mesure de taille

m.d = men.days = jours.hommes

3.3.1.1. Jours.hommes

L'approche la plus classique est d'estimer la taille en jours.hommes.
S'il faut 3 jours à 2 personnes pour terminer une histoire, la taille sera 6 jours.hommes.
L'utilisation de cette mesure est aussi ancienne que discutable.
En effet, les jours.hommes sont également utilisés pour mesuré le "carburant consommé" par un projet !
Ceci conduit à des unités de mesure de vitesse bizarre, en m.d per day, même unité que la consommation journalière mais avec un tout autre sens ou encore une consommation en m.d/m.d (!?!?).

Distance totale	Km	m.d
Distance parcourue	Km	m.d
Distance restant à parcourir = distance totale – distance parcourue	Km	m.d
Temps écoulé	Jours	Days
Vitesse (moyenne) = distance parcourue / temps écoulé	Km/jour	m.d/day
Carburant consommé	Litres	m.d
Consommation	Litre/100Km	m.d/m.d
Consommation journalière	Litres/jour	m.d/day
Temps restant à parcourir = distance restante / vitesse	Jours	m.d/m.d/day = days

[12] C'est une condition nécessaire mais pas suffisante.

Le nombre de jours.hommes nécessaire dépends de la complexité de l'histoire (ou du projet) mais aussi de l'expérience de l'équipe, des outils utilisés, du rendement. Comme si la distance à parcourir dépendait de l'expérience du chauffeur, de la voiture et du rendement du moteur !

Il vaut donc mieux réserver l'utilisation des jours.hommes à la mesure de ce que le projet consomme (ou consommera) et non pas à la mesure de taille.

3.3.1.2. Jours-idéaux

Un "jour-idéal" est par définition un jour où tout ce passe bien, un jour où l'équipe travaille à l'histoire choisie du début à la fin de la journée, sans interruption ni perturbations[13], et où tout ce qui est nécessaire est prêt "sous la main".
Il est possible à une équipe expérimentée, d'utiliser les "jours-idéaux" ("Ideal Days") pour mesurer la taille de chaque histoire.

Notons ci-dessous les jours idéaux avec l'abréviation "ji"

Distance totale	Km	Ji
Distance parcourue	Km	Ji
Distance restant à parcourir = distance totale – distance parcourue	Km	Ji
Temps écoulé	Jours	Days
Vitesse (moyenne) = distance parcourue / temps écoulé	Km/jour	Ji/j
Carburant consommé	Litres	m.d
Consommation	Litre/100Km	m.d/Ji
Consommation journalière	Litres/jour	m.d/j
Temps restant à parcourir = distance restante / vitesse	Jours	J

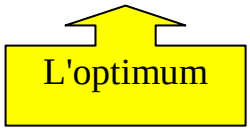
L'utilisation de jour-idéaux pour mesurer la taille évite certains inconvénients de la mesure en jours-hommes mais la confusion entre les jours (normaux) et les jours-idéaux reste possible.

3.3.1.3. Points

Un point est une mesure arbitraire et relative de la taille d'une histoire. Une histoire à 2 points est deux fois plus grande, compliquée, risquée qu'une histoire à un point. C'est lié à l'histoire, indépendamment de qui va l'implémenter, de combien de personnes vont y travailler, du rendement, des outils plus ou moins performant utilisés. Je recommande vivement de mesurer en points.

De plus, pour obtenir des ratios ayant un sens et n'étant pas trop influencés par des événements ponctuels, pour ne pas non plus passer plus de temps à mesurer et calculer qu'à produire, l'équipe Agile mesure le temps par "itération" et non par jour.

Distance totale	Km	**Points**
Distance parcourue	Km	**Points**
Distance restant à parcourir = distance totale – distance parcourue	Km	**Points**
Temps écoulé	Jours	**Itérations**
Vitesse (moyenne) = distance parcourue / temps écoulé	Km/jour	**Points/itération**
Carburant consommé	Litres	**m.d**
Consommation	Litre/100Km	**m.d/itération**
Consommation journalière	Litres/jour	**m.d/jours** **m.d/iteration**
Temps restant à parcourir = distance restante / vitesse	Jours	**Iteration**



[13] en particulier un jour sans réunion, sans appel téléphonique, sans email (!), sans rien d'autre à faire…

3.3.2. Comment évaluer la taille

Nous cherchons une **mesure arbitraire, relative et consensuelle** qui est **proportionnelle** au temps nécessaire à l'équipe pour implémenter complètement une histoire (ou une somme d'histoire).

- Arbitraire veut dire que l'on choisi l'unité qui nous arrange: des "points"
- Relative veut simplement dire qu'une histoire à cinq points est moitié moins longue à implémenter qu'une histoire à 10 points; qu'une histoire à 2 points est deux fois plus grande qu'une histoire à un point. Les mesures sont relatives d'une histoire à l'autre, exactement comme les mesures de longueur ont été longtemps relative à un mètre-étalon. L'important c'est d'être cohérent (raisonnablement cohérent).
- Consensuelle pour que toute l'équipe (idéalement y compris le client) accepte, partage et adhère collectivement à cette mesure: "*on est bien tous d'accord que cette histoire vaut x points*". Il est donc indispensable que presque toute l'équipe participe à l'évaluation.
- Proportionnelle au temps parce que nous diviserons la taille du projet ou d'une itération (somme des points des histoires à implémenter) par une vitesse[14] calculée dans l'unité de mesure adéquate (points par itération). Quelle que soit la valeur d'un point, en divisant la taille par la vitesse, on obtient un nombre d'itération qui a veut dire quelque chose.

Points	Histoire A	Vitesse (2 histoires de même taille finie en une itération)	Durée estimée
plus grands	2 points	4 points/itération	½ itération
plus petits	8 points	16 points/itération	½ itération

Pour estimer la taille d'une histoire, 4 méthodes sont fréquemment utilisées: l'expérience, l'analogie, la désagrégation et le "planning poker; méthodes qui sont expliquées ci-après.

Afin d'éviter des discussions sans fin, une équipe Agile choisira une **échelle de mesure discrète, basée sur la suite de Fibonacci**([15]): **0, 1, 2, 3, 5, 8, 13**, …

Elle s'interdira d'utiliser toute mesure intermédiaire, donc de discuter si la taille d'une histoire est plutôt 6 ou plutôt 7. La question n'est donc pas 6 ou 7 mais bien plus à peu près 5 ou à peu près 8 ? En pratique, cette règle évite beaucoup d'ergotage.

Pour estimer la taille d'un projet ou d'une itération, il suffit de sommer les tailles des histoires concernées.

> N.B.: Il est recommandé que les histoires représentant des concepts et des contraintes soient de taille nulle (ou presque[16])

3.3.2.1. L'expérience

Un expert donne une mesure sur base de son expérience et son intuition. C'est une méthode rapide mais peu consensuelle. Elle tend également à sous-estimer ou surestimer la complexité, donc la taille, dans les domaines ou activités mal connu de l'expert.

3.3.2.2. L'analogie

L'équipe procède par analogie, comparant les histoires entre elles: "*cette histoire est un peu/n fois/… plus grande que celle-ci…*". Cette une méthode relative et consensuelle, qui fonctionne bien quand le nombre d'histoires à évaluer n'est pas trop grand, sinon il est difficile de rester cohérent.

3.3.2.3. La désagrégation

Désagréger veut dire diviser une histoire en histoires plus petites, plus simples à évaluer. Cette méthode est en général combinée avec les autres méthodes, lorsque nécessaire: voir chapitre 3.2.3.

---

[14] La vitesse est donc le coefficient de proportionnalité.

[15] Chaque nombre de la suite est la somme des 2 précédents. Une échelle 0, 1, 2, 4, 8, 16,… est également possible.

[16] Mais il faudra bien vérifier formellement que le logiciel implémente bien les concepts et respecte les contraintes. Selon la complexité, on prévoira

- soit des histoires conceptuelles et de contraintes de taille >= 1;
- soit une ou plusieurs histoires spécifiques afin de vérifier formellement lesdits concepts et contraintes, autrement dit $n \times 0 >= 1$.

3.3.2.4. Le Planning Poker

Référence supplémentaire: http://fr.wikipedia.org/wiki/Planning_poker

La meilleure méthode d'évaluation est le "Planning Poker":

1. Tous les membres de l'équipe s'installent autour d'une table, placés de façon à ce que tout le monde puisse se voir, chacun muni d'un jeu de cartes[17] selon la suite de Fibonacci: 0, 1, 2, 3, 5, 8, 13, 21, …
2. Le responsable de produit explique une histoire à l'équipe.
3. Les participants posent des questions au responsable de produit, discutent du périmètre, évoquent les conditions de satisfaction qui permettront de la considérer comme "terminée".
4. Chacun des participants évalue la complexité, choisit la carte qui correspond à son estimation et la dépose, face vers le bas, sur la table devant lui.
5. Au signal, les cartes sont retournées en même temps.
6. Tant qu'il n'y a pas unanimité, la discussion reprend.
7. Le processus est répété jusqu'à l'unanimité.

Cette méthode est très rapide, efficace et certainement consensuelle.

3.3.3. Quand réévaluer

Dans certains cas, il est nécessaire de réévaluer la taille d'une histoire.

L'équipe réévaluera la taille lorsque :

- une histoire est changée (mais il n'est pas permis de changer une histoire pendant l'itération qui la concerne).

Il est interdit de réévaluer lorsque:

- une histoire est plus rapide ou plus longue que prévu (c'est la vitesse qui change)
- une histoire est plus facile ou difficile que prévu (c'est la vitesse qui change)
- une histoire n'est pas terminée à la fin d'une itération

[17] Une équipe peu douée en bricolage utilisera un jeu de cartes normal en retirant les cartes inutiles, par exemple: Joker=0, 1, 2, 3, 5, 8, Valet=13, Reine=21, Roi=44

3.4. Exercice 1: Spécifier et évaluer des user-stories

Objectif du Projet:
Réaliser une application permettant à l'école d'allouer des ordinateurs et des serveurs aux étudiants (pour différents travaux pratiques) et de communiquer ces informations aux étudiants concernés.

A) Ecrire les spécifications-utilisateurs sous la forme de User Stories. Identifier entre 15 et 20 histoires numérotées.

Veillez à définir correctement les concepts, en particulier les utilisateurs.

B) Evaluer la taille de chaque histoire en "points", en "jouant" au "Planning Poker"

Veillez à terminer complètement la première partie de l'exercice avant d'aborder la deuxième.

Travail en groupe de 4.
Utiliser uniquement la suite Open Office ou PDF
Chaque fichier doit être correctement identifié

- Nom1-Nom2-Nom3-Nom4.US

Au début du fichier, indiquez clairement les noms & prénoms de tous les membres du groupe.
Le fichier a 4 colonnes (ou plus):

- numéro
- utilisateur ("*en tant que* …")
- histoire
- conversations (si nécessaire)
- point

Eviter le plagiat, les copies, soyez créatifs, respectez les critères et les règles du jeu.

Un ou 2 groupes présenteront leurs résultats au début de la session suivante.

Si vous avez des questions, envoyez-les par email à mailto:frederic.tribel@gmail.com?subject=GESPROJ3EX1:maquestion avec le sujet GESPROJ3EX1:maquestion
(comptez sur 24 à 48h pour avoir une réponse)

3.5. Gérer la valeur

3.5.1. Introduction

Récapitulons: l'équipe Agile:

- écrira les spécifications sous forme de user-stories
- évaluera la taille de chaque histoire
- calculera la taille du projet
- décidera la durée d'une itération et la composition de l'équipe
- estimera la vitesse (sur base de l'expérience ou d'une itération d'apprentissage)
- calculera le nombre nécessaire (estimation) d'itération pour compléter le projet.

Bien souvent, un problème apparaît à ce stade: le nombre d'itérations estimé conduit soit à un dépassement du budget, soit à un dépassement des délais, soit les deux. Il est alors nécessaire

1) soit de (re)négocier avec le client
2) soit d'éliminer ou simplifier des spécifications jusqu'à ce que le projet "rentre à nouveau dans l'épure".

Comment choisir les spécifications à garder et celle à éliminer ? Sur base du risque (voir chapitre 3.11), sur base de la rentabilité et/ou sur base de la valeur pour le client.

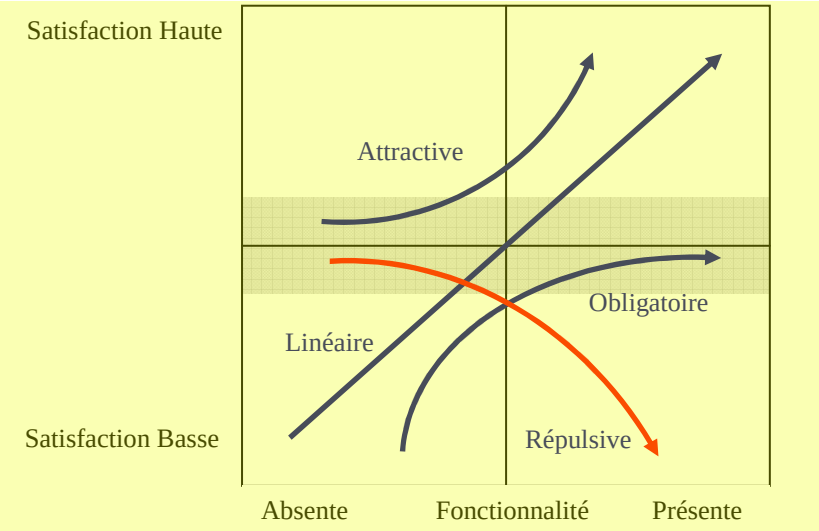
La valeur pour le client a été étudiée et formalisée par de nombreux auteurs avec des modèles plus ou moins objectifs, plus ou moins financiers, plus ou moins subjectifs. Un modèle subjectif et non-financier intéressant est le modèle de satisfaction client que le Professeur Kano, de l'Université de Tokyo, a inventé dans les années 80.

3.5.2. Le modèle Kano

Référence supplémentaire: http://www.qualitystreet.fr/2007/12/13/le-modele-de-kano-si-precieux-pour/

Le Professeur Kano classe la préférence, la satisfaction du client (par rapport à une fonctionnalité, une exigence, une histoire,…) en 5 catégories: obligatoire, linéaire, attractive, répulsive, indifférente.

Obligatoire	Must-have	Une exigence obligatoire doit être présente. Si elle ne l'est pas, c'est l'échec assuré. Améliorer une exigence obligatoire a peu d'impact sur la satisfaction du client. *Le lit dans une chambre d'hôtel.*
Linéaire	Linear	Une exigence linéaire est du type "le plus, c'est le mieux". La satisfaction du client est proportionnelle au nombre, à la qualité, au niveau de performance…. *La taille d'une chambre d'hôtel.*
Attractive	Exciters Delighters	Une exigence Attractive offre un plus qui attire ou fait plaisir au client mais dont l'absence n'est pas source d'insatisfaction. *Une bouteille d'eau gratuite dans une chambre d'hôtel.*
Répulsive	Reverse	Une exigence Répulsive fait fuir le client. *Un chambre fumeur pour un client non-fumeur.*
Indifférente	Indifferent	Le client est indifférent à la présence ou l'absence d'une exigence indifférente.



Toutes les exigences obligatoires doivent donc être implémentées. Il faut implémenter autant que possible d'exigences linéaires. Eviter les exigences répulsives et essayer d'implémenter quelques exigences attractives mais se rappeler que leur absence ne crée pas vraiment d'insatisfaction chez le client.

3.5.3. Classer les histoires avec le modèle de Kano

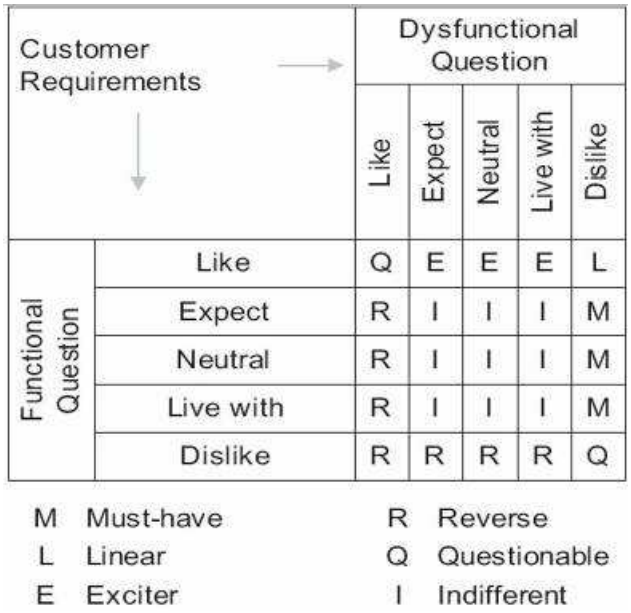
1) Constituer un panel représentatif des clients/utilisateurs, en principe entre 20 et 30 personnes
2) Pour chaque histoire, poser 2 questions à chacun: une question fonctionnelle, la même question sous la forme dysfonctionnelle.

   - Si le produit, l'application peut faire cela… qu'en penseriez-vous ?
     - Cela me fait plaisir, j'aime ("Like")
     - C'est un minimum pour moi ("Expect")
     - Cela m'est égal ("Neutral")
     - Je l'accepte, je peux vivre comme cela ("Live with")
     - Cela me dérange ("Dislike")

   - Si le produit, l'application ne possède pas la fonction … , qu'en penseriez-vous ?
     - Cela me fait plaisir, j'aime ("Like")
     - C'est un minimum pour moi ("Expect")
     - Cela m'est égal ("Neutral")
     - Je l'accepte, je peux vivre comme cela ("Live with")
     - Cela me dérange ("Dislike")

3) Classer chaque histoire dans le modèle de Kano en utilisant la grille d'analyse ci-contre

4) Analyser la distribution des réponses et décider.



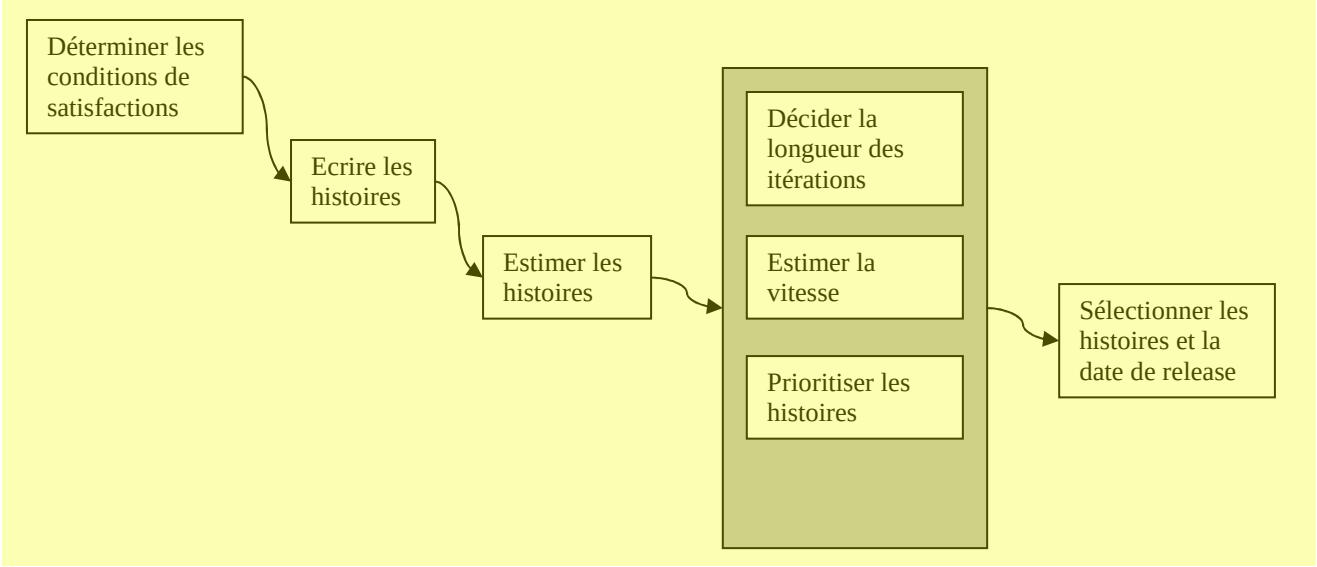
Customer Requirements → / Functional Question ↓	Dysfunctional Question: Like	Expect	Neutral	Live with	Dislike
Like	Q	E	E	E	L
Expect	R	I	I	I	M
Neutral	R	I	I	I	M
Live with	R	I	I	I	M
Dislike	R	R	R	R	Q

M Must-have
L Linear
E Exciter
R Reverse
Q Questionable
I Indifferent

Histoire	Must	Exciter	Reverse	Linear	?	Indifferent	Result
1	10%	20%	1%	74%	0%	5%	Linear
2	60%	15%	0%	10%	5%	10%	Must
…							

3.6. Planifier les Releases

Planifier une release, c'est créer un plan de haut-niveau qui couvre plusieurs itérations, typiquement 2 à 6 mois donc 5 à 12 itérations ou plus. Un projet contient donc au moins une release et dans le cas le plus simple, planifier la release = planifier le projet.

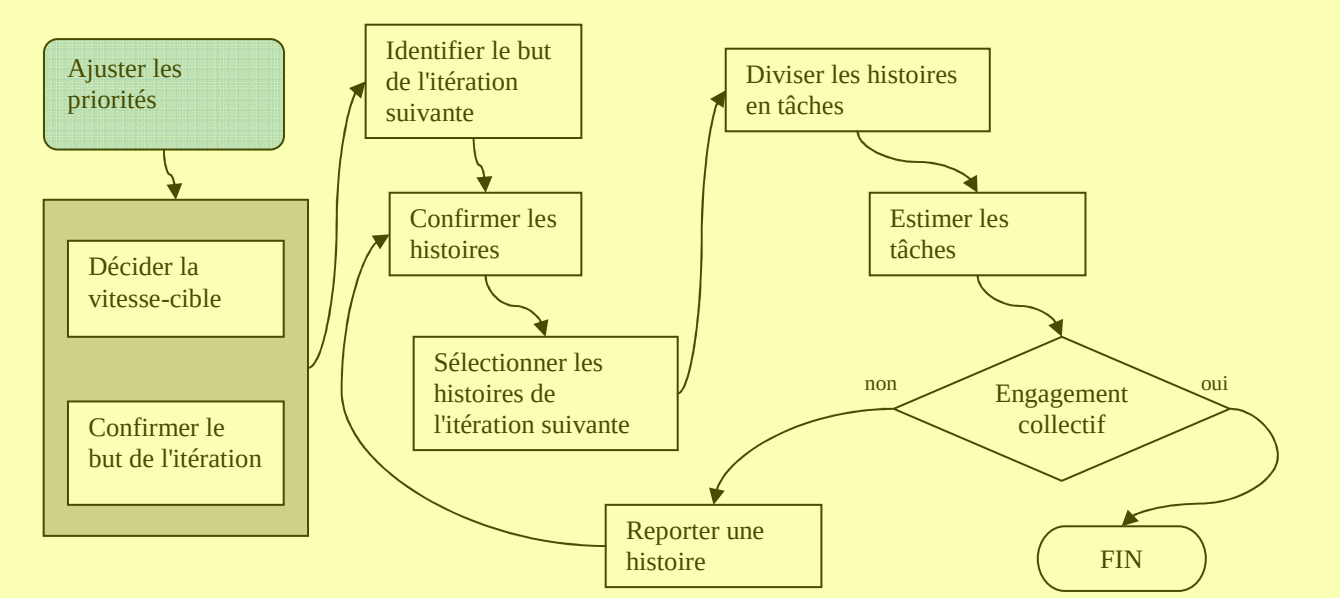


1) **Déterminer les conditions de satisfaction**
   Quel est l'objectif ? Terminer toutes les histoires ("feature-driven project") ou terminer pour une certaine date ("date-driven project"). Une vision commune, partagée de cet objectif est indispensable.
2) **Ecrire les histoires**
   Voir chapitre 3.2
3) **Estimer les tailles des histoires**
   Voir chapitre 3.3
4) **Décider la longueur des itérations**
   La plupart des équipes Agiles utilisent des itérations de 2 à 4 semaines. Rappelons que cette durée est constante pour toute une release. Voir chapitre 3.1.5
5) **Estimer la vitesse**
   Trois méthodes classiques pour estimer la vitesse avant de commencer. Cette estimation vaudra ce qu'elle vaudra, le temps d'obtenir des mesures réelles plus précises en exécutant des itérations concrètes. L'équipe Agile adaptera son planning à sa performance réelle.
   - **L'expérience**
     Le chef de projet ou dans un monde idéal l'équipe utilise sont expérience de projets semblables ou des releases précédentes pour estimer la vitesse. Si la technologie, le domaine, l'équipe, le responsable produit, les outils, l'environnement sont semblables, les chiffres d'un projet précédent pourront être utilisés, ou adapté à la hausse ou à la baisse.
   - **Faire une prévision calculée**
     Le chef de projet estime combien d'heure chaque membre de l'équipe va effectivement travailler au projet chaque jour, combien de jour par semaine peuvent être consacrés au projet, quel pourrait être le rendement effectif. En combinant tout ces nombres, il obtient une estimation de vitesse
   - **Exécuter une itération de test, d'apprentissage**
     Faire une première itération sur base d'une vitesse estimée au moyen des 2 autres méthodes. Elle mesure sa vitesse réelle et utilise cette mesure pour replanifier toute la release.

   La meilleure solution est probablement de combiner les 3 méthodes.
6) **Prioritiser les histoires**
   Voir chapitre 3.5
7) **Sélectionner les histoires et la date de release**
   - "feature-driven project": l'équipe implémentera toutes les histoires choisies et la date de release sera calculée sur base de la taille de la release (= somme des tailles des histoires retenues) et de la vitesse estimée, calcul effectué en nombre entier d'itérations.
   - "date-driven project": connaissant la date de release et la durée des itérations, le chef de projet calcule le nombre d'itérations possible, en sur base de l'estimation de vitesse, combien de points peuvent être implémenté. Les histoires sont sélectionnées en fonction de leur priorité et de ce nombre maximum de points possible pour la release.

3.7. Planifier les itérations

Chaque itération commence par une réunion de planification de l'itération: "iteration planning meeting". Cette réunion rassemble toute l'équipe de développement et ne doit pas durer plus que 3 heures. L'objectif est d'arriver à détailler raisonnablement ce que l'équipe fera et arrivera à terminer pendant l'itération.



0) Ajuster les priorités
Les priorités sont revues au début de l'"iteration planning meeting" ou avant celle-ci (voir chapitre 3.5). La révision des priorités est faite (au moins) par le responsable-produit, le(s) client(s) et le chef de projet.

1) Déterminer la vitesse-cible
La vitesse-cible est la vitesse (en points par itération) que l'équipe est confiante de pouvoir atteindre. La méthode la plus simple est de prendre la moyenne des 3 dernières itérations (et pas plus).

2) Confirmer le but de l'itération
Le but de l'itération doit être décidé et partagé. Il symbolise en une phrase ce que l'équipe essaye d'accomplir pendant l'itération. Par exemple: "*terminer tous les rapports*" ou encore "*implémenter la sécurité*". Ce but a été proposé pendant la réunion précédente. Il faut maintenant confirmer et s'engager.

3) Identifier le but de l'itération suivante
Le but probable de l'itération suivante doit être discuté. Il ne sert à rien d'en discuter trop longtemps. Ce but sera revu lors de la prochaine réunion.

4) Confirmer les histoires
L'équipe va confirmer les histoires qu'elle s'engage à terminer. Des histoires sont ajoutées ou retirées (en fait: postposées) en fonction de l'avancement, de leurs tailles, de la vitesse-cible et des priorités revues.

5) Sélectionner les histoires de l'itération suivante
L'équipe va décider les histoires qui seront probablement implémentées lors de l'itération suivante, en fonction de leurs tailles, de la vitesse-cible, des priorités et du but probable de l'itération. Il ne sert à rien d'en discuter trop longtemps. Cette liste sera revue lors de la prochaine réunion. Cette activité permet au chef de projet d'anticiper des éventuels besoins ou contraintes particuliers qui ne seraient pas facile à traiter sans un préavis suffisant.

6) Diviser les histoires en tâches
Chaque histoire sélectionnée est divisée en tâches, qui dépendent de l'histoire. Exemples de tâches:

- Ecrire les scénarios de tests pour montrer comment cela doit fonctionner
- Concevoir l'interface utilisateur
- Présenter l'interface utilisateur aux experts et/ou au client
- Implémenter l'interface utilisateur
- Ajouter les tables dans la base de données
- Automatiser les tests
- Ecrire la documentation utilisateur

7) Estimer les tâches
L'équipe va collectivement estimer la durée de chaque tâche, en heures-idéales[18] . Le total des heures-idéales nécessaire sera comparé au nombre d'heures réelles disponible et au "rendement" de l'équipe.

[18] Une heure-idéale est par définition une heure où tout ce passe bien, une heure où une personne travaille à la tâche du début à la fin, sans interruption ni perturbation. Voir chapitre 3.3.1.2

8) Obtenir l'engagement collectif
L'équipe va s'engager (ou non) à terminer toutes les histoires sélectionnées pendant l'itération. Si l'engagement n'est pas partagé, s'il est jugé irréaliste ou trop ambitieux, une ou plusieurs histoires doivent être reportées à une itération ultérieure.

Astuces, règles supplémentaires

a) Les tâches ne sont pas assignées pendant la planification d'une itération.
C'est inutile car ces assignations peuvent encore changer en fonction de l'avancement, de la disponibilité des personnes. De plus, cela nuit à l'esprit d'équipe

b) Prendre en compte les bugs
Tout bug, tout défaut relatif à une histoire détecté pendant l'itération où l'histoire est implémentée doit être résolu pendant la même itération, si possible. La différence entre les "jours-idéaux" ou "heures-idéales" et les jours ou heures réelles sert à cela. Si le problème ne peut pas être réglé avant la fin de l'itération, soit une nouvelle histoire sera créée, évaluée, prioritisée et traitée dans une itération ultérieure (pas nécessairement la suivante), soit l'histoire est simplement considérée comme non-terminée et sera traitée dans une itération ultérieure (pas nécessairement la suivante).

c) Eviter les "trop grandes" tâches
Il vaut mieux éviter les tâches qui ne peuvent être complétées en une journée normale. Rappelons que si une journée normale de travail comporte habituellement 8 heures, elle contient significativement moins d'heures-idéales. Si nécessaire, remplacer la tâche par plusieurs plus petites, lorsqu'il y a assez d'information pour le faire même pendant l'itération.

d) Ne pas ajouter ou retirer ou modifier significativement une histoire dans une itération en cours
Un projet Agile permet de modifier, d'ajouter ou de retirer presque à tout moment des histoires. Cependant, il ne faut jamais faire cela dans une itération en cours, ni sans l'accord formel du chef de projet et du responsable produit et du client. Le changement et son impact sont étudiés et approuvés lors de l'"iteration planning meeting":

- "feature-driven project": le nombre d'itérations nécessaire sera adapté.
- "date-driven project": pour garantir la date, le nombre d'itérations ne peut pas être augmenté, ou autrement dit, le nombre total de points de la release ne peut pas augmenter. Si nécessaire, des histoires seront retirées (postposées à dans une release ultérieure) ou ajoutées en fonction du nombre de points.

e) Eviter les fins de semaines
Mike Cohn recommande de ne pas terminer les itérations le vendredi et de préférer le jeudi. Cela peut paraître un détail mais une partie du vendredi, 1er jour officiel d'une itération, pourra aisément être consacré à clôturer des tâches ou activités importante qui devraient être terminées sans retard, mais également à préparer le début de la nouvelle itération, tout en préservant le week-end et en permettant, en fin d'après-midi, des activités renforçant la cohésion de l'équipe (aller boire un verre, une sortie, un bowling, ce qu'on veut).

Histoire	Tâches	
En tant que secrétaire, je peux assigner des étudiants à un cours	Déterminer les règles qui disent qui peut suivre quel cours	Spécifier les tests de réception ("acceptance tests")
	Concevoir l'interface-utilisateur	Implémenter l'interface-utilisateur
En tant que professeur, je peux changer la description de mes cours	Spécifier les tests de réception ("acceptance tests")	Changer la vue d'affichage des cours pour permettre l'édition de la description

3.8. Implémenter

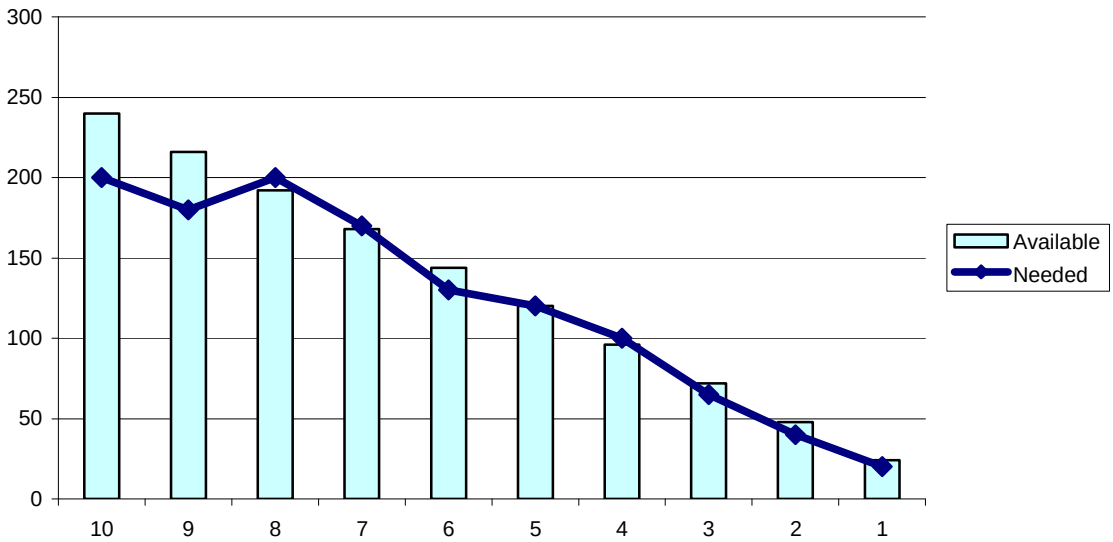
3.8.1. Suivre une itération

Une gestion de projet Agile ne demande pas d'outil sophistiqué: des fiches cartonnées, des tableaux de suivi suffisent.
Un tableau de suivi d'itération comprend habituellement différentes lignes par histoires et plusieurs colonnes pour les tâches suivant leurs statuts:

- à faire
- tests définis
- en cours
- en cours de test
- fini

Il est recommandé d'ajouter une dernière colonne qui reprend le nombre total d'heures mentionnés sur toutes les tâches non terminées.

Le nombre d'heures-idéales restant à faire et le nombre d'heures-hommes encore disponibles d'ici la fin de l'itération (en gros, nombre d'heures multiplié par le nombre de membre de l'équipe, moins les absences ou autres événements prévus) sont indiqués sur un "iteration burndown chart":



3.8.2. Complet ou pas: 0 ou 1

A la fin de chaque itération, l'équipe (ou le chef de projet) compte le nombre de point des histoires **complètement terminées.** Ce sont ces points qui entre en compte pour le calcul de la vitesse.
Les histoires non acceptées, dont les tests ont échoués, ou en cours d'implémentation ne sont pas prise en compte:

- Une histoire presque terminée est simplement terminées dans l'itération suivante, cela affecte la vitesse moyenne et c'est bien normal. Le nombre de point ne sera pas recalculé, ni partiellement affectés à deux itérations. Il est pris en compte dans sa totalité dans l'itération où l'histoire est complètement terminée.
- Dans les autres cas, l'histoire sera prioritisée à nouveau et traitée dans une itération ultérieure (pas nécessairement la suivante). Le nombre de point de l'histoire ne sera pas re-évalué.

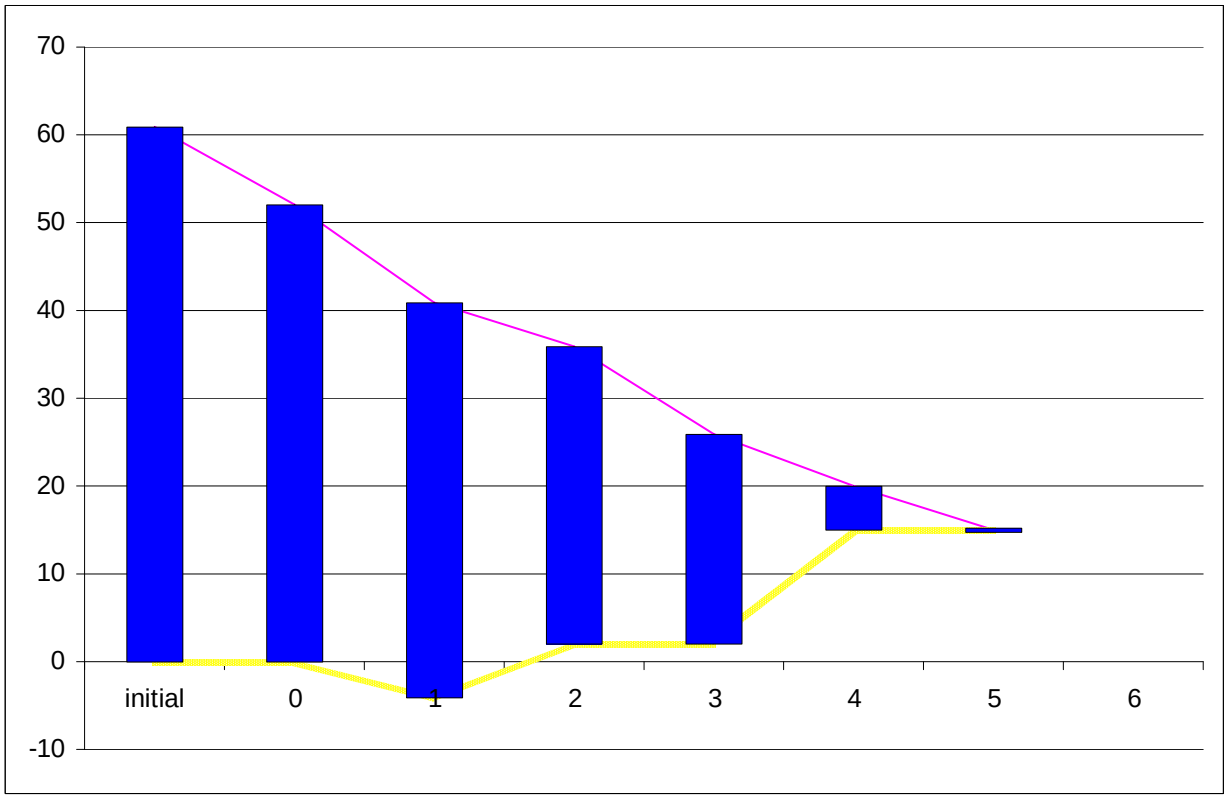
3.8.3. Prendre en charge une tâche

Le chef de projet Agile prendra bien soin de **ne pas assigner** de tâche aux membres de l'équipe. Chacun choisira une tâche selon ses compétences et sa disponibilité et la mènera à bien, le mieux possible et le plus vite possible.
Si l'équipe utilise des fiches papier sur un tableau, il est pratique d'avoir une copie du recto de chaque fiche, qui reste en permanence sur le tableau en plus de la fiche originale qui est emportée par la personne en charge. Celle-ci indiquera son nom sur le duplicata lorsqu'elle emporte la fiche originale, en n'oubliant pas de déplacer le duplicata dans la colonne adéquate. Lorsque la tâche est terminée, les deux documents sont joint à nouveau sur le tableau, jusqu'à prise en charge par la personne suivante.

3.8.4. Suivre les releases

Le chef de projet Agile suivra le progrès de son équipe au début de chaque itération. L'outil le plus utile est appellé "release burndown chart":

- L'axe horizontal donne les itérations
- L'axe vertical est gradué en points
- La première barre donne le nombre de point total tel que prévu au début de la Release
- Le haut de chaque barre indique le nombre de points restant et indique donc la progression
- Le bas de chaque barre indique les changements de "scope" (vers le bas = des points ajoutés, vers le haut, des points retirés)
- La longueur de chaque barre indique la quantité de travail restant à faire (en points)

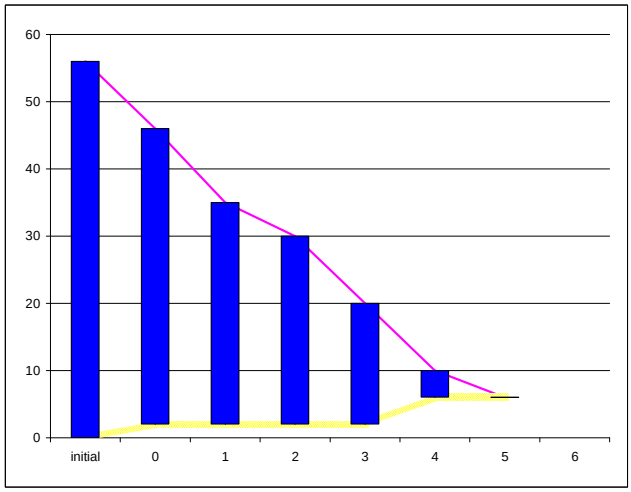


3.8.5. Communiquer

Une communication complète, transparente et honnête est indispensable.
Voici un exemple de rapport de fin d'itération:

Key-figures

Start date	1/01/2009	
Target end date	1/03/2009	
Estimated end date	16/04/2009	
delta	6,6	weeks
Iteration length	2	weeks
Initial size	56	points
Added work	0	points
Removed work	-6	points
Planned size	50	points
actual work	50	points
remaining work	0	points
Current avg velocity	7,7	points/iter.
Total estimated iterations	7,3	iterations
Estimated weeks	15	weeks
Total consumption	65	m.d
Consumption rate	1,3	m.d/point
Estimated workload	65	m.d
Budget	60	m.d
Estimated/Budget	108%	
Team members	1	



Ratios

			%in-scope		% applicable	
	#	points	#	points	#	points
Total number of stories/req	38	59				
In-scope	34	50				
Tested	19	32	56%	64%	56%	64%
Test not applicable	0	0	0%	0%		
Documented	15	25	44%	50%	50%	54%
Document.not applicable	4	4	12%	8%		
Accepted	34	50	100%	100%	100%	100%

Letter

Dear Steering Committee member, dear user/customer representative

Right now, this appears to be a 50-points project.
Based on our performance during the first phase of the project and our experience, with 1 team member and your intimate involvement in the project, a project of this size will take between 12 and 18 weeks. We estimate the project will be completed on or before the 16 avr 2009.

However, a new application release will be ready for your use every 2 weeks and we'll be ticking off these feature stories to your satisfaction.

The good news is that if you're not satisfied, you can stop.
The better news is that if you become satisfied before all the features are done, you can stop.
The bad news is that you need to work with us to make it clear just what your satisfaction means.
The best news is that whenever there are enough features working to make the application useful, you can ask us to prepare it for deployment , and we'll do that.
As we go forward, we'll see how fast we're progressing, and our estimate of the time needed will improve.
In every case, you'll see what is going on, you'll see concrete evidence of usefull application running the tests that you help to specify, and you'll know everything as soon as I know it.

3.9. Exercice 2: Gérer la valeur et planifier une release

Objectif du Projet:
Contexte: voir exercice précédent.

A) Evaluer la valeur en appliquant le modèle de Kano. Choisissez 5 (cinq) histoires et répondez chacun aux 2 formes de questions. Calculez la moyenne des réponses et classez chacune des histoires dans l'une des catégorie: Must, Exciter, Reverse, Linear, Questionable ou Indifferent

B) Considérant des itérations d'une semaine, planifier une première release d'un maximum de 5 itérations. Décrivez votre planning.

Travail en groupe de 4 (pas nécessairement les mêmes que pour le dernier exercice).
Utiliser uniquement la suite Open Office ou PDF
Chaque fichier doit être correctement identifié

- Nom1-Nom2-Nom3-Nom4.VAL (VALeur)
- Nom1-Nom2-Nom3-Nom4.RP (Release Plan)

Au début de chaque fichier, indiquez clairement les noms & prénoms de tous les membres du groupe.

Eviter le plagiat, les copies, soyez créatifs, respectez les critères et les règles du jeu.

Un ou deux groupes présenteront leurs résultats au début de la session suivante.

Si vous avez des questions, envoyez-les par email à mailto:frederic.tribel@gmail.com?subject=GESPROJ3EX2:maquestion avec le sujet GESPROJ3EX2maquestion
(comptez sur 24 à 48h pour avoir une réponse)

3.10. Exercice 3: Planifier et suivre des itérations

Objectif du Projet:
Contexte: voir exercice précédent.

A) Préparez votre première itération. Délivrables (cf § 3.7) : vitesse-cible, but n, but n+1, liste d'histoires pour itération n (minimum 3 histoires/itération svp), liste d'histoires n+1, division en tâches (diviser minimum 3 histoires en minimum 4 tâches chacunes), estimer les tâches en heures-idéales, estimer le nombre total d'heures-idéales, confirmer l'engagement collectif.

B) Terminez votre itération. Choisissez 10% de tâches non terminées (vous essayerez de les finir à l'itération suivante) et 10% d'histoires même pas entamées (vous les re-planifierez au début de la prochaine itération). Calculez votre vitesse. N'oubliez pas de mettre à jour votre "Release burndown chart". Publier un rapport (cf § 3.8.5

C) Préparez votre 2ème itération. Idem A) ci-dessus.

Travail en groupe de 4 (pas nécessairement les mêmes que pour le dernier exercice).
Utiliser uniquement la suite Open Office ou PDF
Chaque fichier doit être correctement identifié

- Nom1-Nom2-Nom3-Nom4.PIT1 (Plan ITeration 1)
- Nom1-Nom2-Nom3-Nom4.LIT1 (Log ITeration 1)
- Nom1-Nom2-Nom3-Nom4.PIT2 (Plan ITeration 2)

Au début de chaque fichier, indiquez clairement les noms & prénoms de tous les membres du groupe.

Eviter le plagiat, les copies, soyez créatifs, respectez les critères et les règles du jeu.

Un ou deux groupes présenteront leurs résultats au début de la session suivante.

Si vous avez des questions, envoyez-les par email à mailto:frederic.tribel@gmail.com?subject=GESPROJ3EX3:maquestion avec le sujet GESPROJ3EX3maquestion
(comptez sur 24 à 48h pour avoir une réponse)

3.11. Evaluer les risques

Un risque est la possibilité de la réalisation d'un évènement contraire aux attentes ou à l'intérêt d'une personne ou d'une organisation. Le terme "risque" désigne à la fois l'évènement considéré et sa probabilité.

3.11.1. Risques du projet

- **Dépassement du temps**
  Le projet prend plus de temps que prévu
- **Dépassement du budget**
  Le projet coûte plus cher que prévu
- **Insatisfaction des utilisateurs**
  Les utilisateurs ne sont pas contents du résultat
- **Insatisfaction du client**
  Les objectifs du client, qui peuvent être différents de ceux des utilisateurs, ne sont pas rencontrés
- **Échec**
  Echec total, le projet doit être arrêté

3.11.2. Risques opérationnels

- **Disfonctionnement**
  L'application ne fonctionne pas comme prévu
- **Mauvaise performance**
  La performance souhaitée n'est pas atteinte
- **Insatisfaction des utilisateurs**
  A l'utilisation, les utilisateurs ne sont pas satisfaits
- **Insatisfaction du client**
  A l'utilisation, le client n'est pas satisfait

3.12. Gérer les risques

Les **risques du projet** sont minimisés par le choix d'une méthode de gestion de projet adéquate et son application correcte. L'analyse ci-dessous est faite par rapport à la méthode Agile décrite au chapitre 3.

Risque	Mitigation
dépassement du temps	Gestion du scope (évaluer la taille, mesurer la vitesse, calculer le nombre d'itérations, éliminer des histoires si nécessaire), des itérations courtes, auto-organisation, motivation de l'équipe Prévoir des "buffers" (voir chapitre 3.12.2)
dépassement du budget	En général budget = temps * taille de l'équipe Bien gérer le temps évite les dépassements de budgets
insatisfaction des utilisateurs	Focalisation sur les histoires-utilisateurs et les priorités de l'utilisateur
insatisfaction du client	Implication du client dans la gestion, permise et facilitée par une gestion focalisée sur la "livraison" de déliverables concret focalisé sur les histoires-utilisateurs
Échec	Minimiser tous les risques ci-avant

Les risques opérationnels sont gérés suivant leur nature:

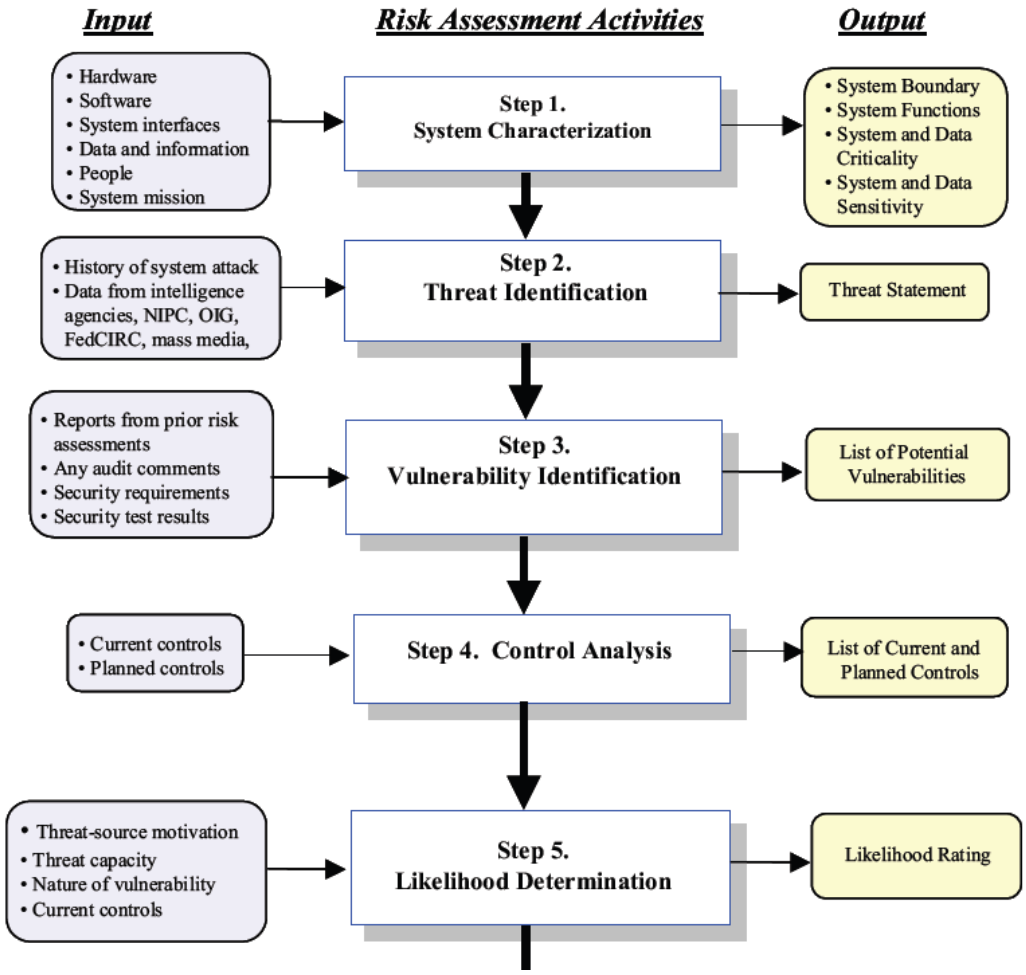
Risque	Mitigation
Disfonctionnement	Voir ci-dessous
Mauvaise performance	
Insatisfaction des utilisateurs	Ces risques sont présents lorsque le projet est considéré comme un succès mais que les opérations, l'utilisation de l'application ne réponds pas ou plus aux besoins explicites ou implicites. Il y a peu d'autre solution que de prévoir une "nouvelle release": • mieux adaptée, • en impliquant encore mieux les utilisateurs et le client, • en faisant bien attention d'identifier les besoins implicites pour qu'ils deviennent explicites
Insatisfaction du client	

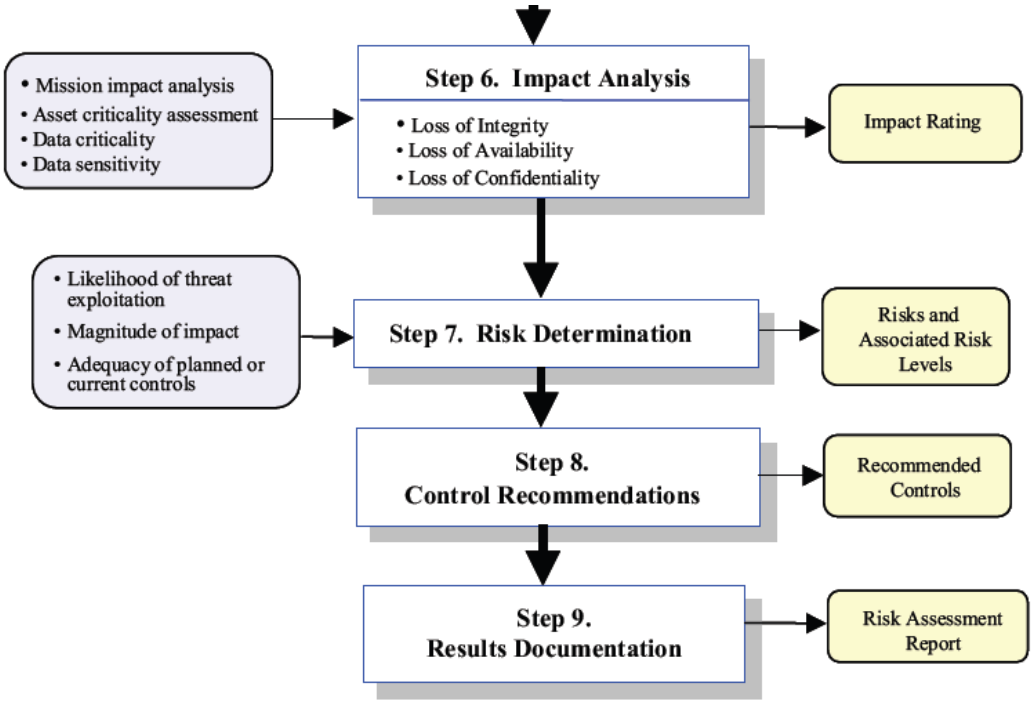
3.12.1. Principe général de gestion de risques

Une référence intéressante en gestion des risques est :

Risk Management Guide for Information Technology Systems
NIST - National Institute of Standards and Technology
Technology Administration – U.S. Department of Commerce
Special Publication 800-30

Référence d'où est extrait le schéma de principe suivant:





3.12.2. Prévoir des réserves

Le meilleur chef de projet du monde à la tête de la meilleure équipe ne peut être certain que tout ce passera exactement comme prévu. Particulièrement dans les contextes où aucun dépassement n'est admissible, il est indispensable de se protéger contre les erreurs d'estimation.

3.12.2.1. Réserve de fonctionnalités

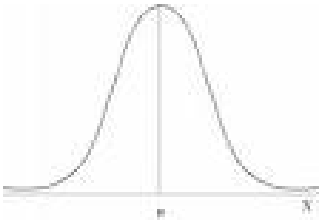
Nous avons vu que les fonctionnalités peuvent être classées, selon le modèle de Kano, en fonctionnalités obligatoires, linéaires, attractives, répulsives ou indifférentes et qu'il faut dans un projet réaliser toutes les fonctionnalités obligatoires, le plus possible de fonctionnalités linéaires, et quelques fonctionnalités attractives. En prévoyant toujours moins de 100% de fonctionnalités obligatoire, l'équipe Agile se crée une réserve de travail qui peut ne pas être fait, donc qui permet de tenir le planning sur l'essentiel au détriment de l'accessoire. Dans la méthode DSDM en particulier, la limite est fixée à un maximum de 70% en temps (ou en points ce qui revient presque à la même chose) de fonctionnalités "Must-have", donc la "réserve de fonctionnalités" est d'au moins 30%.

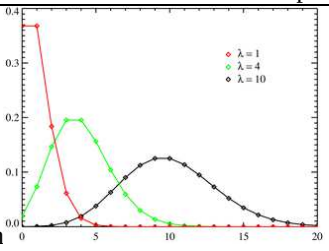
3.12.2.2. Réserve de temps

N.B.: dans ce chapitre, je confondrais temps et points d'une part, user-story et tâche de l'autre.

Nous savons tous que les estimations relatives à une user-story ne sont jamais que des estimations.et qu'elles ont donc toutes une marge d'erreur non nulle. De plus, elles sont faites par des hommes (ou des femmes) qui vont y intégrer leur expérience, leurs visions des facteurs de risques et donc, le plus souvent, surestimer le temps strictement nécessaire.

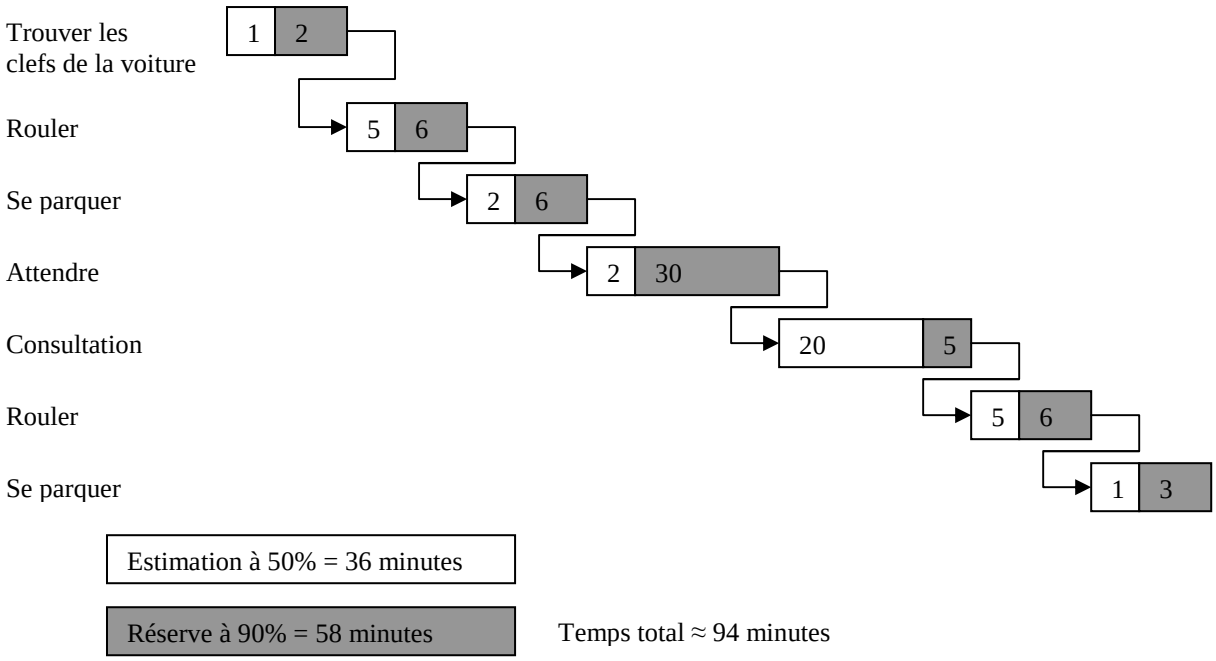
Si on demande à plusieurs personnes d'exécuter une tâche, les mesures de temps seront distribuées le plus souvent selon une distribution gaussienne (aussi appelée normale) soit encore parfois suivant une distribution de poisson.

Gauss 

Poisson 

Faisons l'hypothèse que la distribution est normale. Si l'on prend la moyenne comme estimation, on a 50% de chance que la tâche prenne plus longtemps et 50% moins longtemps. On peut aussi utiliser une estimation plus conservatrice, disons à 90%, alors par définition, on aura une probabilité de 90% que la tâche dure moins longtemps et seulement 10% de risque qu'elle dure plus longtemps.

Que ce passe-t-il si toutes nos estimations sont conservatrices ?
Considérons une visite chez le médecin:

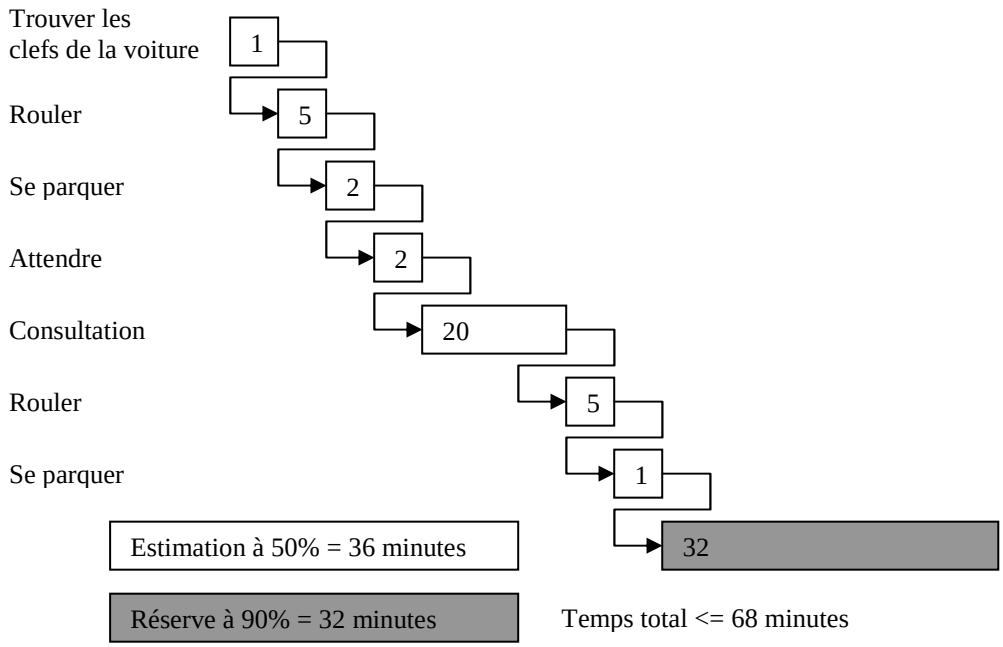


Mais faut-il réellement plus d'une heure et demie ? **Non** dans notre exemple car il faudrait que chaque tâche requière l'estimation haute, ce qui est peu probable. **Oui** dans un projet informatique car s'il l'on donne 11 heures à quelqu'un pour terminer une tâche (cf "Rouler" dans notre exemple), il le fera, en moyenne en 11 heures, mais 11 heures, c'était déjà notre estimation à 90% (il existe une « loi » qui dit qu'une tâche prend toujours tout le temps qui lui est alloué, voire plus, mais jamais moins). De plus, si jamais la tâche était effectuée en 5h (estimation à 50%), il faudrait qu'elle ait commencé à temps (ce qui est rarement le cas = syndrome de l'étudiant) et que la tâche suivante puisse commencer immédiatement (ce qui est rarement le cas = syndrome de l'étudiant) pour ne pas impacter le projet dans son ensemble.

Comment prendre en compte efficacement tous ces éléments ? **Simplement :**

1) **en ne prenant aucune sécurité au niveau individuel de chaque tâche**, autrement dit en l'estimant à 50% de probabilité,
2) **en prenant une réserve globale, au niveau de l'ensemble du projet,** réserve globale intelligemment calculée.

Notre exemple devient alors:



La réserve globale du projet est calculé comme la racine carrée de la somme des carrés des différences entre l'estimation à 90% et l'estimation à 50% ([19]). Cette réserve globale est sous la responsabilité du chef de projet, et planifiée en fin de

[19] Le lecteur intéressé par une démonstration consultera "Agile Estimating and Planning" de Mike Cohn page 195 ou encore "Critical Chain" du Dr. E. Goldratt (édité par North River Press)

projet (ou de release sous forme d'itérations additionnelles). Elle est connue de tous mais les règles du jeu doivent être claires:

- n'utiliser la réserve globale qu'en cas de réelle nécessité, elle n'est pas là pour permettre de fignoler ou de faire du superflu
- faire le maximum pour réaliser chaque tâche selon son estimation "nominale" (donc à 50%)
- commencer chaque tâche au plus tôt
- ne pas stresser si l'on dépasse sur une tâche, il y a d'autres tâches qui iront plus vite (le principe même des estimations à 50%) et il y a la réserve globale
- ne pas blâmer quelqu'un qui dépasserait l'estimation nominale d'une tâche

3.13. Synthèse: pourquoi cela marche

Dans de nombreux cas, la gestion de projet Agile peut fonctionner beaucoup mieux que les méthodes plus classiques.

- **Le client est très impliqué dans le projet**
  Le client participe à la gestion du projet, il la comprend puisque les concepts sont simples et centrés sur les histoires écrites dans son vocabulaire.
- **Le contenu du projet ne doit pas nécessairement être gelé longtemps à l'avance**
  Des histoires peuvent être changées, ajoutées, retirées à tout moment
- **Il n'y a pas de confusion possible entre les points, les jours.hommes et les jours-calendriers**
  La vitesse lie pointes et itérations.
- **Les évaluations sont indépendantes l'une de l'autre**
  Le problème de surévaluation ou sous-estimation systématique disparaît en utilisant le concept de vitesse mesurée
- **Le syndrome de l'étudiant est minimisé**
  Des itérations courtes mais constantes maintiennent une pression contrôlée, sans besoins de Pert ou Gantt détaillés. La plupart des histoires et des tâches sont complétées aussi vite et aussi tôt que possible
- **Délivrer souvent un logiciel qui fonctionne est au centre des préoccupations**
  Et non pas respecter la méthode ou le plan de projet
- **La gestion de projet Agile demande moins de temps que la gestion de projet classique**
- **Un projet Agile est plus motivant pour tous les membres de l'équipe**
- **La probabilité de satisfaction des utilisateurs et du client est maximisée**
  Evident vu leur implication dans toutes les activités, y compris la gestion du projet.

3.14. Outils

Il existe de nombreux outils facilitant la gestion de projet Agile
Parmi une offre très étendue, je mentionnerais les suivants (les étoiles attribuées étant un classement personnel et très subjectif):

Rally	http://www.rallydev.com	★★★
Pivotal Tracker	https://www.pivotaltracker.com/	★★
Mingle	http://www.thoughtworks-studios.com/mingle-agile-project-management	★★
VersionOne	http://www.versionone.com	★
Scrinch	http://sourceforge.net/projects/scrinch/	★
Agilefant	http://sourceforge.net/projects/agilefant/	
Agiletrack	http://sourceforge.net/projects/agiletrack/	
Planagile	https://www.planigle.com/demo/Planigle%20Demo.htm	

4. Annexe : Exemples de questions

Voici quelques exemples de questions. Cette liste n'est évidement pas exhaustive.

Q1. Donnez la définition d'un projet et les cinq conditions nécessaires (/10)

Q2. Définissez "Diagramme de Gantt" (/5)

Q3. Vrai/faux (/10) (correct +0.5, incorrect ou non répondu: 0)

1. Il faut que chaque tâche du chemin critique soit en retard pour que le projet soit en retard [Vrai / Faux]
2. Une phase d'un projet peut être considérée comme terminée lorsque tous les déliverables contractuellement prévus ont été livrés [Vrai / Faux]
3. La durée d'une itération est variable [Vrai / Faux]
4. La durée d'une release est variable [Vrai / Faux]
5. La durée d'une itération est comprise entre 2 mois et 6 mois [Vrai / Faux]
6. L'objectif d'une itération est de délivrer les déliverables contractuellement prévus [Vrai / Faux]
7. La vitesse est calculée en fonction des tâches terminées et entamées [Vrai / Faux]

Les questions 2015-2016 seront en grande majorité à choix multiples