Génie Logiciel

Dominique Revuz, Philippe Finkel, Philippe Cluzeau

ESIPE

avril 2021

Génie Logiciel : Enfonçons le clou !

Pourquoi le Génie Logiciel ?

parce qu'il y a beaucoup d'échecs et de demi-échecs !

Extraits (1) : Étude sur 8 380 projets

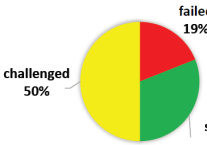
Standish Group, 1995

- Succès : 16 %
- Problématique : 53 % (budget ou délais non respectés, défaut de fonctionnalités)
- Échec : 31 % (abandonné)

Extraits (2) : Standish, 2020

Project Success
Quick Reference Card

Based on CHAOS 2020: Beyond Infinity Overview. January'2021, QRC by Henny Portman



Modern measurement
(software projects)

Good Sponsor, Good Team, and Good Place are the only things we need to improve and build on to improve project performance.

The Good Place is where the sponsor and team work to create the product. It's made up of the people who support both sponsor and team. These people can be helpful or destructive. It's imperative that the organization work to improve their skills if a project is to succeed. This area is the hardest to mitigate, since each project is touched by so many people. Principles for a Good Place are:

- The Decision Latency Principle
- The Emotional Maturity Principle
- The Communication Principle
- The User Involvement Principle
- The Five Deadly Sins Principle
- The Negotiation Principle
- The Competency Principle
- The Optimization Principle
- The Rapid Execution Principle

The Good Team is the project's workhorse. They do the heavy lifting. The sponsor breathes life into the project, but the team takes that breath and uses it to create a viable product that the organization can use and from which it derives value. Since we recommend small teams, this is the second easiest area to improve. Principles for a Good Team are:

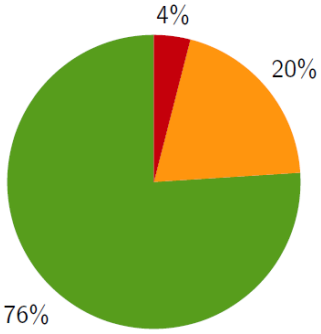
- The Influential Principle
- The Mindfulness Principle
- The Five Deadly Sins Principle
- The Problem-Solver Principle
- The Communication Principle
- The Acceptance Principle
- The Respectfulness Principle
- The Confrontationist Principle
- The Civility Principle
- The Driven Principle

The Good Sponsor is the soul of the project. The sponsor breathes life into a project, and without the sponsor there is no project. Improving the skills of the project sponsor is the number-one factor of success – and also the easiest to improve upon, since each project has only one.
Principles for a Good Sponsor are:

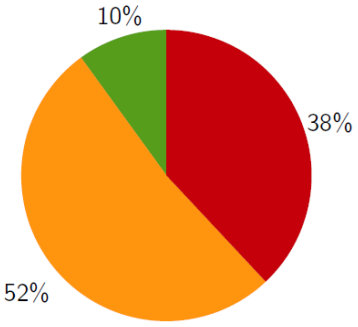
- The Decision Latency principle
- The Vision Principle
- The Work Smart Principle
- The Daydream Principle
- The Influence Principle
- The Passionate Principle
- The People Principle
- The Tension Principle
- The Torque Principle
- The Progress Principle

source: https://hennyportman.files.wordpress.com/2021/01/project-success-qrc-standish-group-chaos-report-2020.jpg

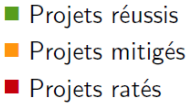
Extraits (3) : Petits vs Gros projets



Petits projets
budget ≤ $1 million

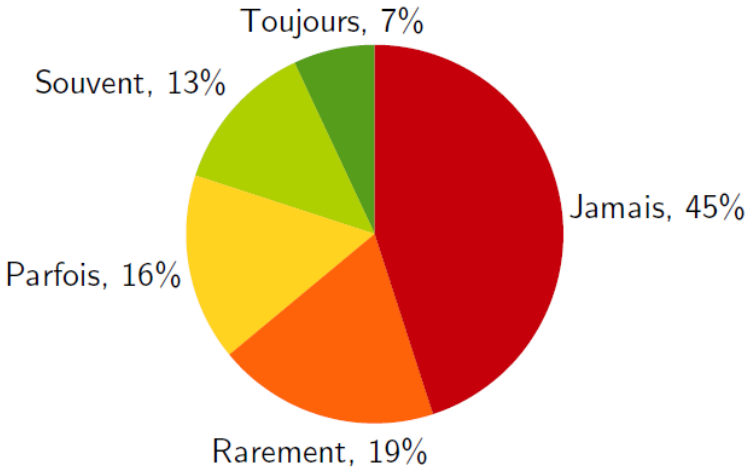


Grands projets
budget ≥ $10 millions



Standish group, *Chaos Manifesto 2013 - Think Big, Act Small*, 2013

Extraits (4) : Utilisation ?



Standish group, *Chaos Manifesto 2002*, 2002

“La satisfaction du client et la valeur du produit sont plus grandes lorsque les fonctionnalités livrées sont bien moins nombreuses que demandé et ne remplissent que les besoins évidents” (Standish group, Chaos Report 2015)

Extraits (5) : Facilité de maintenance ?

Zeltovitz, Tom De Marco (19??)

	Répartition effort dév.	Origine des erreurs	Coût de la maintenan
Définitions des besoins	6%	**56%**	**82%**
Conception	5%	27%	13%
Codage	7%	7%	1%
Intégration Tests	15%	5%	2%
Maintenance	**67%**	5%	2%

En résumé

Constat du développement logiciel dès la fin des années 60 :

- délais de livraison non respectés
- budgets non respectés
- **ne répond pas aux besoins de l'utilisateur ou du client**
- difficile à utiliser, maintenir, et faire évoluer

“Bon sens” et formalisme UML

- l’essentiel :
  - comprendre les besoins,
  - se comprendre à l’intérieur de l’équipe (expliciter sans ambiguité),
  - se faire comprendre du client (reformuler)
- le formalisme UML, qui sert comme :
  - check list
  - moyen d’expression synthétique
  - moyen d’expression interne à l’équipe
  - moyen d’expression formel auprès du client; qui complète, illustre, synthétise un discours
  - outil indispensable d’homogénéité pour les gros projets ou les grandes organisations
  - Modélisation graphique : très utile ... seulement s’il est compris par tous de la même manière (merci UML)
  - Langage standard

pour les INFO2 ? Les 2 sont importants, l’essentiel est essentiel, UML est un outil.

Introduction à l'élaboration

Comprendre les besoins du client pour rédiger le cahier des charges fonctionnel.

La démarche générale est *simple* et peut s'exprimer en quelques questions :

- Quelles sont les limites du système à construire (Frontière / **Périmètre** / Scope)
- Quels sont les utilisateurs du système (**Acteurs**) et que cherchent-t-ils à faire (**cas d'usage/UC**) ?
- Quels sont les objets (du monde réel) et leur association dans le contexte du projet (**MOD**) ?
- Quels objets utilisés par chaque cas d'utilisation ?
- Comment gère t-on les problématique de qualité (**FQM**) ?
- Comment construire effectivement le système avec du code ? (architecture, études techniques, maquettes IHM, ...)

Si une de ces questions est ignorée le risque d'un échec est fortement augmenté.

La compréhension du **contexte** du projet et la formalisation du **glossaire** sont des prérequis indispensables.

L'ensemble de ces éléments doit être complété au fur et à mesure, et la cohérence de l'ensemble sans cesse remise en question.

Rêvons !

L'équipe a réfléchi aux objets du domaine manipulés

L'équipe a identifié et décrit tous les scénarios d'utilisation du système qu'elle est sur le point de construire, et aux acteurs concernés

L'équipe a vérifié que toutes les exigences fonctionnelles du système sont prises en compte dans l'analyse et qu'elles sont satisfaites.

L'équipe est capable de décrire le logiciel à l'aide de 4 schémas principaux :

- un diagramme des acteurs qui liste les acteurs et leur relation,
- de manière synthétique au travers d'un diagramme d'interactions générale,
- de manière détaillée avec un diagramme exhaustif des cas d'utilisations,
- de manière détaillée avec un diagramme des objets du domaine, de leurs relations et leurs attributs

Processus - formalisme

Au-delà de ces étapes, un processus formel doit permettre de répondre à d'autres exigences communes à tout projet :

1. Il doit être suffisamment souple pour accueillir différents styles et types de problèmes.
2. Il doit soutenir la manière dont les gens travaillent vraiment (y compris le prototypage et développement itératif / incrémental).
3. Il doit servir de guide aux membres moins expérimentés de l'équipe, en les aidant à être aussi productifs que possible sans menotter davantage les membres expérimentés.

Exemple de définition d'un **Processus de développement logiciel** : *Ensemble d'activités successives, organisées en vue de la production d'un logiciel.*

En pratique :

- Pas de processus idéal
- Choix du processus en fonction des contraintes (taille des équipes, temps, qualité...)
- Adaptation de « processus types » aux besoins réels

Première étape : construire le contexte

Le contexte d'un logiciel est l'ensemble des concepts manipulés par les utilisateurs et par le logiciel.

La construction du contexte est fortement liée à la première version du glossaire, au périmètre du projet, et implicitement à la liste des acteurs.

Exemple pour un Logiciel de gestion des réservations d'un spectacle. Les concepts à clarifier : billets, horaire, spectateur, spectacle, réservation, paiement, place, confort, prix, remboursement, balcon, poulailler, cintres, VIP, etc.

Construire le contexte = cela permet de parler la même langue que les experts ou utilisateurs du domaine :

- expert : spécialiste du domaine *métier*
- utilisateur : quand il n'y a pas d'expert, les utilisateurs et leurs besoins est ce qui s'en rapproche le plus.
  et donc de se rendre compte VITE quand on n'a pas compris. Le fameux ***"contexte et enjeux"*** *reste d'actualité !*

Bien entendu, si vos experts ne sont pas bons (ou juste pas assez matures sur le sujet) ... il faut donc toujours avoir un panel d'utilisateurs pour valider les cas d'utilisation.

Pour construire le contexte, le premier outil est le vocabulaire (glossaire) du logiciel à réaliser.

Le logiciel à réaliser, le système visé : l'usage est de l'appeler **le système**.

Le glossaire

Notion importante dans la gestion d'un projet, le glossaire est l'outil qui a le meilleur rapport coût / intérêt de toutes les techniques de génie logiciel.

Il permet de construire une base de communication simplifiée avec vos clients et équipiers.

Un glossaire est un dictionnaire

Un glossaire est un dictionnaire mais qui se limite aux sens spécifiques utilisés dans le contexte du projet.
Comme tout dictionnaire, il est une liste ordonnée de définitions.
Une définition étant composée d'un mot (ou terme ou expression) et de sa définition en intention et en extension.
La définition en **intention**, c'est le sens que vous souhaitez donner au mot.
La définition en **extension**, c'est une définition par l'exemple, vous fournissez un certain nombre (souvent un seul) de phrases dans lesquelles le mot/terme/expression est utilisé.
Exemple :

- Utilisateur : Les utilisateurs sont des clients de la banque ayant au moins un compte et donc un identifiant internet et un protocole de connexion sécurisé. "Un visiteur qui a validé sa demande de compte deviendra un utilisateur une fois que le contrôleur aura validé sa demande".

Glossaire ou pas ?

Les définitions suivantes sont trop génériques pour un projet informatique classique :

- Fake: a class that implements an interface but contains fixed data and no logic. Simply returns “good” or “bad” data depending on the implementation.
- Mock: a class that implements an interface and allows the ability to dynamically set the values to return/exceptions to throw from particular methods and provides the ability to check if particular methods have been called/not called.
- Stub: Like a mock class, except that it doesn’t provide the ability to verify that methods have been called/not called.
- base de données clients : une base de données clients vous aide à améliorer la fidélisation des clients, à augmenter les revenus et à augmenter la valeur de chaque client. Lisez la suite pour tout savoir sur la gestion des données clients.
- VP : Vice Président.

Les acteurs “naissent” dans le glossaire

Tous les acteurs et patries prenantes doivent être identifiés dès le recueil des besoins, et doivent explicités dans le glossaire.

Exemple :

- Utilisateur : Les utilisateurs sont des clients de la banque ayant au moins un compte et donc un identifiant internet et un protocole de connexion sécurisé. “Un visiteur qui à validé sa demande de compte deviendra un utilisateur une fois que le contrôleur aura validé sa demande.”

Les acteurs initialement oubliés seront ajoutés lors des vérifications de cohérence ultérieures.

Les objets du domaine “naissent” dans le glossaire

Les objets du domaine initialement oubliés seront ajoutés lors des vérifications de cohérence ultérieures.

Dedans ou Dehors ?

La définition suivante est limite, doit-elle être dans le glossaire ? ou c'est un "truc" de la réalisation, qui n'est utile qu'aux codeurs ? Si elle est compréhensible pour un non informaticien, ET peut être utile à la lecture, alors OK.

- data structure : In computer science, a data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. (wikipedia)

La suivante n'a rien a faire dans le glossaire et doit être dans la doc technique du design technique, et/ou de la classe en question, **"une place pour chaque chose, chaque chose à sa place"**.

- Notification observer manager: The notification observer manager is responsible for the handling of the list of observers reponding to notification informations.

Les définitions du dictionnaire n'ont pas leur place dans un glossaire (ou vous ajoutez un lexique séparé pour vos amis non francophones)

Le glossaire du cours de Génie Log !

- Besoin
- Exigence
- Acteur
- Glossaire :-)
- Objet du domaine
- User Story
- Use Case
- Fonctionnalité

Acteurs

Comment modéliser les personnes qui interagissent avec le logiciel ?

Doit on modéliser seulement les personnes qui touchent l'ordinateur (les interfaces physiques) ?

Qui est "impacté" par le logiciel ?

Quelles sont les interfaces du logiciel (système) avec l'extérieur ?

Doit-on modéliser autre chose que des utilisateurs ?

A la recherche des Acteurs

Définition

Les acteurs d'un système sont les entités externes à ce système qui interagissent (saisie de données, réception d'information, ...) avec lui. Les acteurs sont donc à l'extérieur du système et dialoguent avec lui. Ces acteurs permettent de cerner l'interface que le système va devoir offrir à son environnement.
Oublier des acteurs ou en identifier de faux conduit donc nécessairement à se tromper sur l'interface et donc la définition du système à produire.

Acteurs Humains et les autres (Non Humains)

les acteurs incluent les utilisateurs humains, mais aussi les autres systèmes informatiques ou hardware qui vont communiquer avec le système.

Le Rôle

Un utilisateur (qui n'est pas un acteur) peut avoir plusieurs rôles pour le système : il donne des information et il reçoit des informations ou déclenche des opérations dans le système ou sur d'autres acteurs.
La bonne modélisation est celle qui identifie les différents rôles et donc permet de cibler directement le besoin en se limitant à chaque rôle l'un après l'aute sans avoir une solution générique pour tous qui n'est pas toujours appropriée.
Ainsi, plusieurs utilisateurs peuvent avoir le même rôle, et donc correspondre à un même acteur, et une même personne physique peut jouer des rôles différents vis-à-vis du système, et donc correspondre à plusieurs acteurs.
Chaque acteur doit être nommé. Ce nom doit refléter son rôle, car un acteur représente un rôle joué vis-à-vis du système.
A RETENIR : pour nous, ACTEUR = ROLE.

Pour trouver les acteurs d'un système, il faut identifier quels sont les différents rôles que vont devoir jouer ses utilisateurs (ex. : responsable clientèle, responsable d'agence, administrateur, approbateur, ...). Il faut également s'intéresser aux autres systèmes avec lesquels le système va devoir communiquer, comme :

- les périphériques manipulés par le système (imprimantes, hardware d'un distributeur de billets...) ;
- des systèmes informatiques externes au système, mais qui interagissent avec lui, etc.

Frontière

Pour faciliter la recherche des acteurs, on peut imaginer les frontières du système. Tout ce qui est à l'extérieur et qui interagit avec le système est un acteur, tout ce qui est à l'intérieur est une fonctionnalité à réaliser.

Attention à la frontière : ceci doit être un point de discussion important avec le client en effet vous allez dimensionner le projet avec le choix de la frontière.
Exemple, une caisse de supermarché :

- Frontière 1: lecteur de carte, caissier, caisse (contient l'argent), lecteur de code à barre etc ...
- Frontière 2: les clients, les fournisseurs (le caissier est un élément du système).

Acteurs indirects

j'ai trouvé ce conseil sur internet :
"*Vérifiez que les acteurs communiquent bien directement avec le système par émission ou réception de messages. Une erreur fréquente consiste à répertorier en tant qu'acteur des entités externes qui n'interagissent pas directement avec le système, mais uniquement par le biais d'un des véritables acteurs. Par exemple, l'hôtesse de caisse d'un magasin de grande distribution est un acteur pour la caisse enregistreuse, par contre, les clients du magasin ne correspondent pas à un acteur, car ils n'interagissent pas directement avec la caisse.*"

Attention effectivement mais l'erreur peut aussi avoir lieu dans la non prise en compte de cet acteur indirect : le client d'une banque, il n'interagit pas avec le logiciel de gestion mais s'il ne peut obtenir la simulation qu'il souhaite il va changer de banque.
Il faut que les besoins des acteurs indirects de réaliser des tâches soient satisfaits sinon le logiciel sera défectueux.

Mais formellement ce n'est pas des acteurs au sens de la méthode mais des parties prenantes. Les modéliser comme acteurs indirects est une bonne chose.

Qu'implique l'existence d'un acteur ?

Une interface API pour un acteur non-humain.
Une interface IHM (ou matériel) pour un acteur humain.
La conception d'une interface est une des missions des développeurs/concepteurs.
Le critère de qualité d'une interface est quelle est adaptée aux **capacités de l'acteur**, ce qui implique que votre définition d'acteurs doit contenir le "niveau" de compétence de l'acteur, et s'il n'est pas homogène, il faudra en tenir compte.
Pour les acteurs humains, le niveau de compétence permet de définir le niveau d'ergonomie et la formation (et documentation) nécessaire pour les futurs utilisateurs ayant ce rôle.
Enfin une interface doit s'adapter au niveau de répétition des actions à faire.
Quand les cas d'utilisation sont peu fréquents, l'interface doit être formative, (autoformation en interagissant).
Quand le cas d'utilisation implique de réaliser un même traitement sur plusieurs centaines de données, il faut que l'interface se fasse la plus économique possible.
Votre bon sens doit s'appliquer ici. Et une bonne donnée est la fréquence/densité d'utilisation.

Héritage

Pour simplifier l'écriture des acteurs il est parfois judicieux d'utiliser un héritage simple :

- A est un B signifie que A peut prendre la place d'un B devant l'ordinateur.
- L'exemple typique est l'administrateur qui hérite en général d'autres rôles ... ou du directeur commercial qui est aussi un commercial, ou du responsable de formation qui est aussi un enseignant (?), ...

Vous pourrez illustrer les relations d'héritage entre les acteurs dans les ***diagramme de cas d'utilisation****. Parfois, afin d'alléger les schémas, on s'autorise à faire un diagramme des acteurs (pas UML), qui représente les relations d'héritage mais sans les détails des Use Cases.*

Acteurs Principaux vs Acteurs Secondaires

Attention : 2 axes de définition

1. au niveau du système :
   - Les acteurs principaux : les acteurs qui utilisent les fonctions principales du système. Normalement les acteurs principaux sont à la gauche dans le cadre du diagramme de contexte.
     - Le système est construit pour répondre aux besoins / problèmes des acteurs principaux.
   - Les acteurs secondaires : par exemple, les acteurs qui effectuent les tâches de maintenance, administration et paramétrage du système.
     - Attention : il peut y avoir des use Case destinés à des Acteurs secondaires mais qui sont indispensables à la réussite du projet.
   - Une façon de les identifier et de se poser la question : “le système existe-t-il sans eux ?”
2. au niveau des UC (*ce sera notre axe !*)
   - L’acteur est dit principal pour un cas d’utilisation lorsque l’acteur est à l’initiative des échanges nécessaires pour réaliser le cas d’utilisation.
   - Les acteurs secondaires sont sollicités par le système alors que le plus souvent, les acteurs principaux ont l’initiative des interactions

Pourquoi les acteurs ?

IL est important de bien décrire le rôle de l'acteur.
En effet on distribue l'écriture des cas d'utilisation sur l'équipe de développement.
IL faut que le document de description de l'acteur permette de synchroniser l'équipe de dev et d'orienter l'écriture des cas d'utilisation.
Mettre en avant les objectifs d'utilisation du système est une bonne idée.
Explicitez les difficultés que peut avoir l'acteur à utiliser le système :

- trop de répétition dans un temps court
- complexité de la procédure
- problème de lisibilité (mont Saint Odile, 030 Meters == 0,30 Feet/second ?)

Parties prenantes (StakeHolders)

Souvent les parties prenantes ne sont pas définies !
Il s'agit formellement de toutes les personnes impliquées de près ou de loin.

Une définition :

- Utilisateurs: Ceux qui utilisent le logiciel.
- Clients: Ceux qui paient pour le logiciel.
- Développeurs: Ceux qui conçoivent et construisent le logiciel.
- Managers: Ceux qui gèrent (manage) l'organisation développant le logiciel.

Une autre : “Il s’agit de l’ensemble des personnes et des organisations qui ont quelque chose à voir avec le projet. Soit elles sont directement impliquées dans la conduite des opérations, soit elles sont impactées par la problématique de départ, par le choix ou la mise en œuvre des solutions. Certaines encore peuvent exercer une influence à différents niveaux.”

- Le commanditaire (ou demandeur, ou encore client interne) : c’est le premier concerné par le projet
- Les utilisateurs, les services impactés : ceux qui sont concernés directement par les livrables
- La direction : représente le pouvoir décisionnel et de contrôle ultime
- L’équipe projet : comprenant le chef de projet ainsi que les autres membres de l’équipe
- Les services supports impliqués : la comptabilité, la logistique, les ressources humaines, l’informatique...
- Les autres experts : apportant leurs conseils ponctuellement (directeurs fonctionnels...)
- Les clients : les premiers concernés en externe, impactés directement si leur rôle s’inscrit dans l’utilisation du produit ou service livré par le projet
- Les fournisseurs : de matière, de prestation, de main d’ œuvre
- Les organismes publics : dans le cas où votre projet doit s’inscrire dans un cadre

Pourquoi on s'intéresse aux parties prenantes

- parce que les acteurs sont dedans
- parce que les acteurs oubliés sont dedans
- parce que les acteurs indirects sont dedans
- parce qu'au travers de leur objectifs, des enjeux essentiels du projet peuvent être identifiés

On parlera abusivement de *Partie Prenante* alors qu'on pensera aux *Autres parties Prenantes*, hors acteurs.

Ex: Les acteurs indirects sont des parties prenantes qui interagissent avec le système par l'intermédiaire d'un autre acteur.

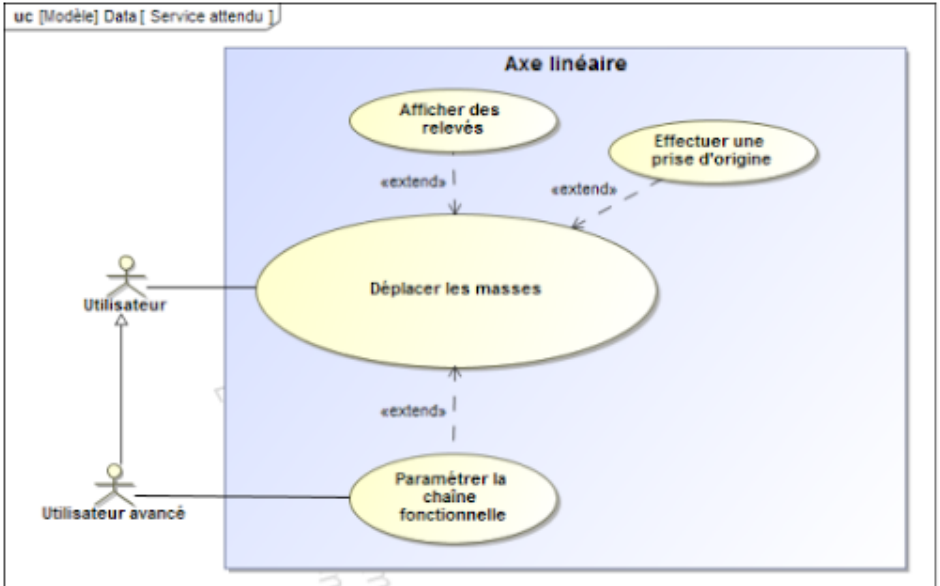
Diagramme des acteurs

Contient les liens d'héritages entre acteurs

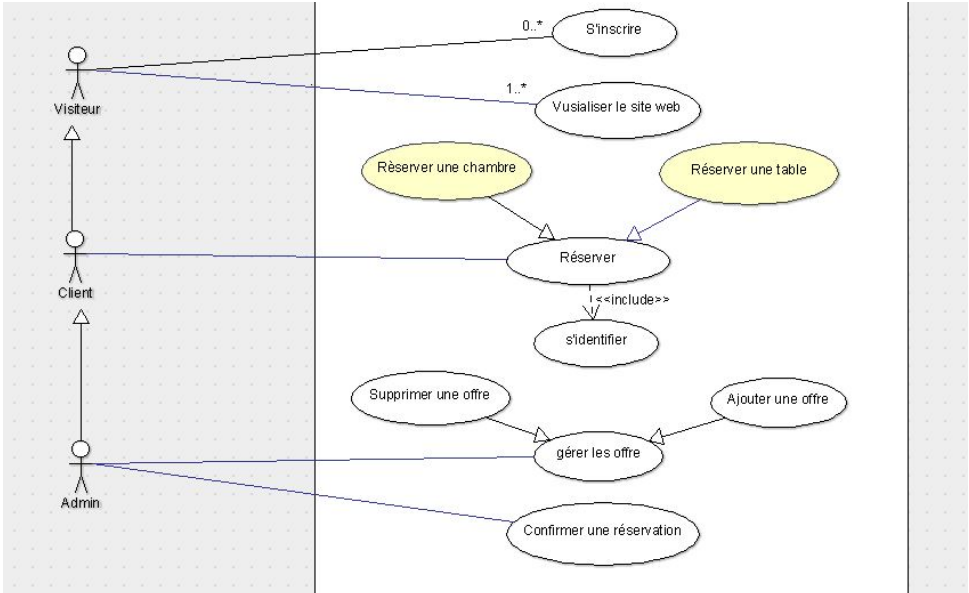
Acteur matérialisé par Bonhomme en bâton

Les non-humains ont un chapeau

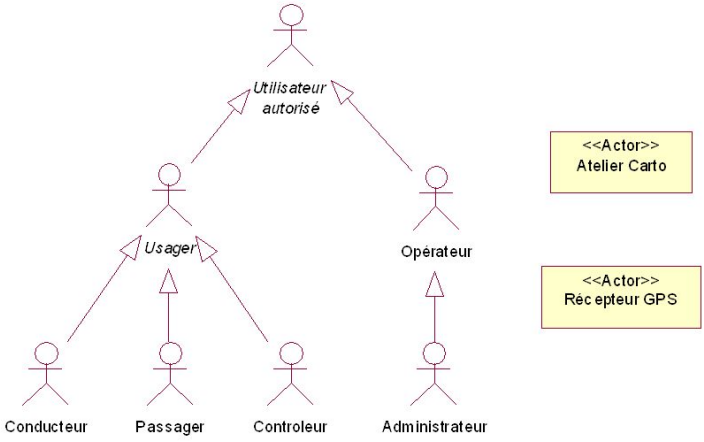
En général lié aux UC

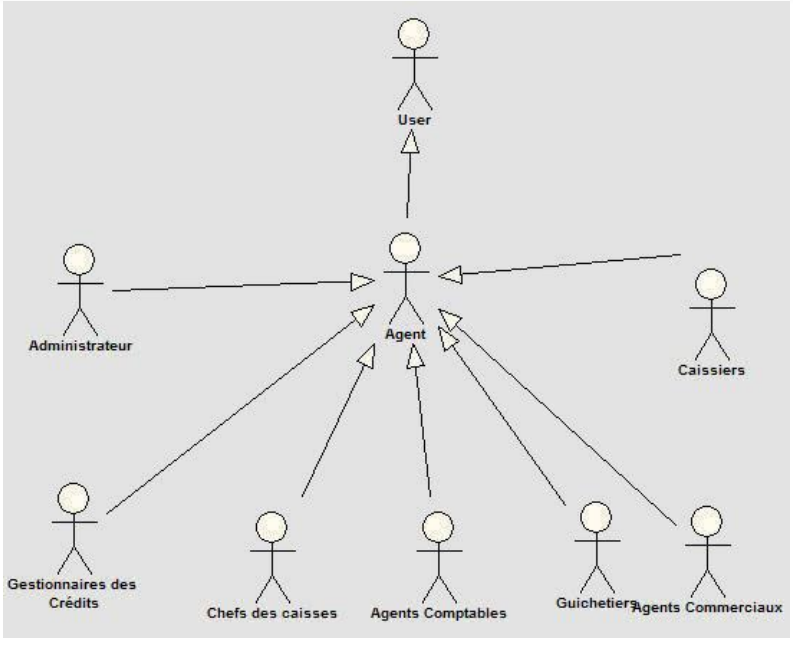


Ça peut devenir compliqué



DIAGRAMMES DE CAS D 'UTILISATION
Exemple de modélisation des acteurs




User
Administrateur
Agent
Caissiers
Gestionnaires des Crédits
Chefs des caisses
Agents Comptables
Guichetiers
Agents Commerciaux

Synthèse

- Un Acteur = un rôle
- Acteur du système : à ne pas confondre avec les acteurs du projet de construction du système
- Bien différencier Acteurs Humains vs Acteurs Non Humains
- Bien différencier Acteurs (implicitement Directs) vs Acteurs Indirects
- Choix GL INFO2 : différencier Acteurs Indirects et Parties prenantes. Les Acteurs Indirects seront liés à des Use Cases et auront des Objectifs par rapport au système. Les Parties prenantes sont concernées / impactées mais le lien peut être plus global.
- Parties Prenantes = implicitement les *Autres* Parties Prenantes
- Un utilisateur (humain) peut avoir plusieurs rôles

Cas d'usage / Cas d'utilisation (Use Case)

Les objectifs des Acteurs.

L'outil le plus puissant et le plus difficile à utiliser dans la méthode Objectory est le cas d'usage.

User Story

- Format : En tant que je veux afin de
- Les user stories sont indépendantes les une des autres
- User Story vs Use Case :
  - Les Users stories sont plus simples à comprendre que les Use Cases.
  - niveau de détail faible dans les US. élevé dans les UC
  - suivant le contexte, on peut utiliser des US, ou des UC, ou d'abord des US puis des UC

Use Case

Un Use Case (ou cas d'utilisation) correspond à une liste d'étapes, définissant les interactions entre un acteur et le système, c'est-à-dire toutes les fonctionnalités que doit fournir le système.
Il peut être représenté sous 2 formes : la description textuelle et le diagramme.
Le template textuel n'est pas universel et dépend du style du concepteur, mais en règle générale, il se compose des éléments suivants : le nom, l'objectif, l'acteur principal, les acteurs secondaires, les pré-conditions nécessaires au déclenchement de l'action, la description de la séquence nominale, des éventuelles séquences alternatives, des séquences d'exceptions et des post-conditions.

Fonctionnalité

Les fonctionnalités se situent au niveau au dessus : par exemple “gérer les utilisateurs”. Une fonctionnalité va se découper en plusieurs US/UC.

Question : pourquoi plutôt des US que des UC dans un contexte Agile ?

Identifier les objectifs

Un cas d'utilisation permet à un acteur de réaliser un objectif en interagissant avec le système.

La première étape est donc de recenser les objectif de chaque acteur !!

Diagramme d'interaction général

Sur ce sujet, peu de consensus ! Et la difficulté réside aussi dans le type d'utilisation des diagrammes (de macro à micro, de fonctionnel à technique).

Pour ce cours, on va faire des diagrammes à cheval entre le diagramme de contexte et le diagramme de package.

Les principes sont très simples :

- une boîte représente le système
- les acteurs sont à l'extérieur
- à l'intérieur du système, on va faire apparaître (graphiquement sous forme de package ou pas !) les grandes fonctionnalités que le système doit fournir.
- et les liens permettront de visualiser quels acteurs utiliseront quelles fonctionnalités
- détailler les fonctionnalités suffisamment pour donner beaucoup d'informations ... mais pas trop pour rester lisible

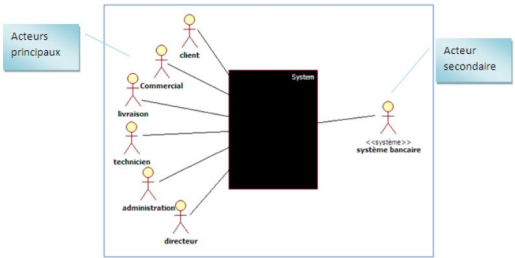


Figure 3 – Diagramme de contexte

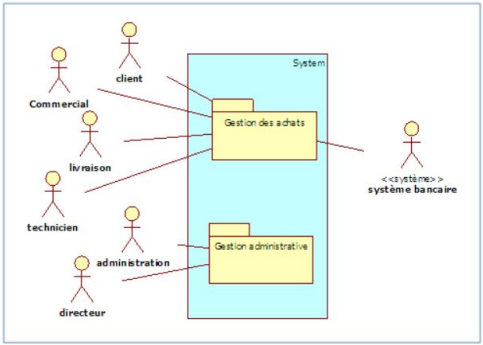
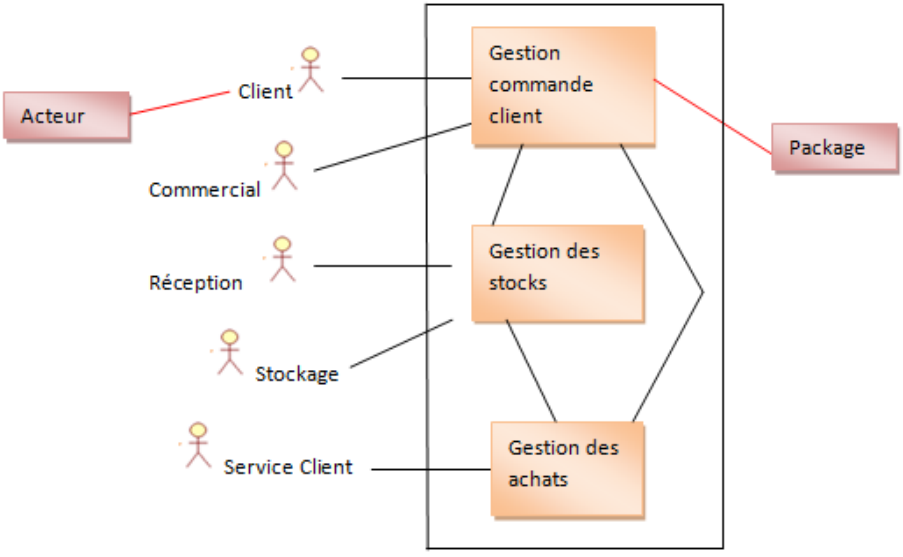


Figure 4 – Diagramme de contexte


Acteur
Client
Commercial
Réception
Stockage
Service Client
Gestion commande client
Gestion des stocks
Gestion des achats
Package

Retours TD3 : sur le contexte

Assez pauvre :

- *L'université gère les données sur les heures travaillées de ses enseignants sur papier nécessitant la participation de ces derniers chaque fin d'année avec le remplissage d'informations. L'idée du projet est de remplacer ce mécanisme en automatisant ce traitement d'informations via l'informatique.*
- *Université Paris Est Marne la Vallée veut améliorer sa gestion de paie des enseignants vacataire, soit passer au numérique.*

Limite faux : *L'Université Paris Est Marne la Vallée est demandeuse d'un logiciel de gestion des heures avec pour objectif d'automatiser la gestion des paies des enseignants*

Un début : *Création d'un logiciel de gestion d'heures qui permettrait aux enseignants d'indiquer les heures qu'ils ont fait dans l'établissement. Ce logiciel est utile pour la gestion des heures et de paye des enseignants.*

A retravailler mais bien plus riche : *Conception d'un logiciel de gestion d'heures pour des enseignants. Pour le moment, utilisation de document papier pour compter les heures de formations –> perte de temps est importantes. Les heures sont déclarées, validées (responsable de formation), mise en paiement (DRH), payées (PAYE). Doit permettre de trier les déclarants par an. Gestion des vacataires differentes : le service du perssonnel crée les vacataires, le responsable administratif de formation et composotante declare les fiches vacataires et le secretaire ou responsable administratif manipule le dossier du vacataire.*

Retours TD3 : sur les Acteurs

Choix arbitraire : différencions Acteurs indirects et Parties prenante.

- Acteurs Humains
- Acteurs Non Humains
- Acteurs Indirects
- Parties Prenantes

Erreurs :

- Confusions acteurs du projets et acteurs du système à construire
- Mauvaise compréhension du terme MOA

Manque fréquent : Description de l'acteur dans le glossaire

A venir : description du ou des objectifs pour l'acteur

Retours TD3 : sur le glossaire

Souvent, des erreurs, des manques de précisions, ou approximations ... pas grave si on les *chasse* régulièrement.

- “demande de paiement. Équivalent à facture”
- “CRI : Centre de Recherche Informatique”
- “Enseignants vacataires : Intervenants n’étant pas présent une année complete au sein d’un etablissement.”
- “Composante : école, faculté, , faisant partie de l’université”
- “Déclarants : Individu fournissant une déclaration”

Quelques exemples de définitions plus intéressantes :

- “Composantes : L’université est décomposée en sous unités administrative et organisationnelles, les composantes. Celles-ci regroupent des formations et des laboratoires (qui ne nous intéressent pas ici). Par exemple l’ESIPE (qui n’a pas de laboratoire), et l’IGM où les enseignants d’informatique font leur autre cours.”
- “Vacataire : Enseignant non titulaire qui reste pour une durée déterminé”
- “CA: Conseil d’administration, organe élu qui dirige l’université.”

Retours TD3 : sur le périmètre

- beaucoup de manques et aussi quelques contre-sens :
  - “Spéfication fonctionnel détaillé et conception du projet.”
  - confusion périmètre et acteurs non humains

Simplifiez-vous la vie en vous posant la question : quelles fonctionnalités seront implémentées ?

Certains ont commencé à le faire :

```
- Création d'une déclaration
- Validation des déclarations par la hiérarchie
- Consultation de la déclaration de chaque personne par RH et VP finance
- Création de demandes pour le service finance
- Signalisation des erreurs dans une demande
- Liaison avec le logiciel planning Ypareo
- Interfaçage avec SHIBOLETH pour auth
```

ça va devenir plus clair (j’espère !) avec le diagramme d’interactions général...

Retours TD3 : sur les exigences

- globalement très inégal
- attention aux périmètre : “Edition de fiche de paye”

On ne va pas insister beaucoup sur le sujet, on se concentrera pour la suite du cours sur fonctionnalités et use cases.

Retours . . . sur les retours TD3

Logistique :

- utilisez le milestone
- fermez l'issue quand c'est fini. "pingez" les membres du groupe quand vous fermez l'issue
- **travail individuel . . . et 11 issues** !!

Contenu :

- parfois assez juste :
  - “C’est une bonne idée d’avoir séparé les acteurs indirect et direct, cependant les acteurs qui participent au développement de le logiciel et les acteurs qui utilisent le logiciel sont mélangés”
  - “Quelques acteurs pourraient être ajouter comme ‘service du personnel ‘, ‘DRH’. Il y a bien la séparation acteurs et parties prenantes. On pourrait préciser les acteurs non humain comme les logiciels.”
- parfois trop creux, manque de “contenu” :
  - “Une plutôt bonne compréhension du contexte et un périmètre assez bien définie”
  - “Glossaire plutôt complet” .
  - “Les rendus dans leur globalité sont clairs.”
  - . . .
  - Dans ce cas, **pointez ce qui peut être amélioré ! Sinon pas très utile.**