L'analyse et la modélisation, suivies de l'implantation par SPRINTs successifs, sont des étapes cruciales dans le processus de développement logiciel agile. Elles permettent de transformer les besoins et les idées en un logiciel fonctionnel et de haute qualité. Voici une explication détaillée de chaque étape :

Obtention d'un mandat :

À cette étape, vous obtenez un mandat qui spécifie ce que le logiciel doit accomplir.

Le mandat peut provenir d'un client externe (une entreprise ou un utilisateur final), d'un client interne (un autre département ou une équipe de votre entreprise) ou même de vous-même si vous initiez le projet.

Le mandat clarifie les attentes en termes de livrables, ce que le logiciel doit réaliser pour répondre aux besoins et aux objectifs.

Analyse textuelle :

L'analyse textuelle consiste à examiner en détail le mandat pour comprendre pleinement les besoins et les objectifs.

Il s'agit d'effectuer une lecture formelle du document du mandat, de l'analyser ligne par ligne pour s'assurer que rien n'est négligé.

Cette étape est essentielle pour éviter les malentendus et les erreurs de compréhension.

Identification des cas d'usage et des rôles :

Vous identifiez les cas d'usage, c'est-à-dire les différentes manières dont le logiciel sera utilisé par les acteurs (utilisateurs, systèmes externes, etc.).

Les rôles définissent qui sont les acteurs impliqués dans chaque cas d'usage.

Cette étape aide à établir une compréhension claire des fonctionnalités attendues du logiciel du point de vue des utilisateurs.

Scénarios d'utilisation :

Les scénarios d'utilisation décrivent séquentiellement les étapes nécessaires pour accomplir chaque cas d'usage.

Ils mettent en lumière le flux de travail du logiciel, montrant comment les utilisateurs interagiront avec le système pour atteindre leurs objectifs.

Cela permet de visualiser le comportement attendu du logiciel.

L’explication de la manipulation humaine dans l’interface et les opérations effectuées par l’ordinateur

Maquettes graphiques :

À cette étape, vous créez des maquettes graphiques qui représentent le design visuel du logiciel, y compris les pages et les écrans que les utilisateurs verront.

Ces maquettes servent de base pour le design de l'interface utilisateur (UI) et permettent de visualiser à quoi ressemblera le logiciel une fois développé.

Définition des classes et des méthodes : CRC

Vous identifiez les classes (objets) et les méthodes (opérations) qui seront nécessaires pour implémenter les fonctionnalités du logiciel.

Cette étape concerne la conception orientée objet et aide à définir la structure du code source.

Modélisation de données :

Vous définissez la structure des tables de données ou d'autres modèles de données nécessaires pour stocker les informations du logiciel.

Cela garantit que la gestion des données est correctement planifiée, ce qui est essentiel pour assurer la cohérence des données. CSV

Planification globale :

La planification globale consiste à établir un cadre temporel pour le projet.

Vous déterminez l'ordre de réalisation des différentes parties du logiciel et découpez le projet en itérations appelées SPRINTs.

Implantation par SPRINTs successifs :

Après la planification globale, le développement réel commence avec les SPRINTs.

Chaque SPRINT est une période courte (généralement de 2 à 4 semaines) au cours de laquelle une partie spécifique du logiciel est développée, testée et livrée.

SCRUMs quotidiens :

Les SCRUMs sont des réunions quotidiennes de l'équipe de développement.

Chaque membre de l'équipe partage les tâches qu'il a complétées, celles en cours et signale les problèmes rencontrés.

Debriefing/Ajustement :

À la fin de chaque SPRINT, une réunion de débriefing est organisée pour évaluer l'avancement du projet.

Des ajustements sont faits au plan en fonction des retours d'expérience et des problèmes rencontrés.

Livraison partielle :

Le logiciel est livré partiellement au fur et à mesure de l'avancement des SPRINTs.

Cela permet au client de commencer à utiliser des parties fonctionnelles du logiciel dès les premières itérations, ce qui est bénéfique pour obtenir des retours rapidement.

En résumé, ce processus permet de développer un logiciel de manière itérative, en se concentrant sur les besoins du client, en planifiant et en ajustant continuellement le projet pour assurer la livraison de fonctionnalités de haute qualité.

Un cas d'utilisation, également appelé bloc fonctionnel ou cas d'usage en génie logiciel et ingénierie des systèmes, est un concept qui définit comment un système doit être utilisé pour apporter de la valeur ou de l'utilité aux acteurs impliqués. Il s'agit d'une manière de décrire comment un système interagit avec ses utilisateurs pour atteindre un objectif précis. Les cas d'utilisation sont une partie importante de la collecte des exigences d'un système, et ils adoptent le langage et le point de vue des utilisateurs finaux.

Historiquement, les cas d'utilisation ont été introduits par Ivar Jacobson en 1987 et ont évolué au fil des années pour devenir une méthode puissante de capture des besoins et de modélisation dans le développement de logiciels et d'ingénierie des systèmes. Ils ont été initialement utilisés dans le cadre de méthodes orientées objet, mais leur utilisation s'est étendue à d'autres paradigmes de développement.

Principes clés des cas d'utilisation :

**Finalité** : Chaque cas d'utilisation est associé à une finalité ou un objectif spécifique pour un acteur du système. L'acteur primaire est généralement un utilisateur humain ou un autre système, et il peut y avoir des acteurs secondaires.

**Scénarios** : Les cas d'utilisation sont décrits à travers des scénarios qui détaillent comment le système interagit avec les utilisateurs pour accomplir l'objectif. Il y a souvent un scénario principal, mais il peut également y avoir des variantes et des exceptions.

**Langage utilisateur** : Les cas d'utilisation sont exprimés dans le langage de l'utilisateur final ou de l'expert du domaine, évitant ainsi le jargon technique.

**Types de cas d'utilisation** : Il existe plusieurs types de cas d'utilisation, notamment les cas d'utilisation concrets (décrivant en détail les interactions), les cas d'utilisation paramétrés (regroupant des cas similaires avec des paramètres), les cas d'utilisation essentiels (décrivant les intentions de l'utilisateur sans préjuger de l'implémentation), et les cas d'utilisation métier (dans le contexte des modèles d'affaires).

Les cas d'utilisation sont utilisés pour définir les besoins fonctionnels d'un système, guider le développement logiciel, et servir de base pour la conception et les tests. Ils sont également adaptés à l'approche agile, où ils sont découpés en tranches pour une mise en œuvre itérative. Les cas d'utilisation jouent un rôle essentiel dans la communication entre les parties prenantes du projet et les équipes de développement, assurant ainsi que le système réponde aux besoins réels des utilisateurs.

Cas d'usage (Use Case) :

Un cas d'usage est une description abstraite d'une interaction entre un système (tel qu'un logiciel) et un acteur (un utilisateur humain, un autre système, etc.). Il décrit ce que le système doit accomplir pour satisfaire un besoin spécifique de l'acteur.

Les cas d'usage identifient les fonctionnalités ou les services que le système doit offrir à ses utilisateurs. Ils décrivent les interactions entre les acteurs et le système d'une manière globale, en mettant en avant les objectifs ou les finalités que chaque acteur cherche à atteindre.

Les cas d'usage sont généralement des descriptions de haut niveau, qui ne détaillent pas nécessairement la séquence d'actions spécifiques.

Scénario d'utilisation (Use Case Scenario) :

Un scénario d'utilisation est une description spécifique et détaillée d'une séquence d'actions qui se produisent lorsque l'utilisateur interagit avec le système pour accomplir un cas d'usage particulier.

Les scénarios d'utilisation sont des exemples concrets qui montrent comment un cas d'usage spécifique est mis en œuvre dans le système. Ils détaillent les étapes précises que l'utilisateur suit pour atteindre un objectif.

Chaque cas d'usage peut avoir plusieurs scénarios d'utilisation associés, couvrant différentes situations ou variantes du même cas d'usage.

En résumé, un cas d'usage est une description générale d'une interaction entre un système et un acteur, tandis qu'un scénario d'utilisation est une instance spécifique et détaillée de cette interaction. Les cas d'usage sont utiles pour identifier les fonctionnalités globales du système, tandis que les scénarios d'utilisation fournissent des exemples concrets de la manière dont ces fonctionnalités sont utilisées.

Quant à la "planification globale" dans le contexte du développement logiciel agile, elle est cruciale pour déterminer le cadre temporel du projet et l'ordre de réalisation des différentes parties du logiciel. Voici ce à quoi elle sert :

Cadre Temporel : La planification globale permet de définir une échelle de temps pour le projet, en identifiant les grandes étapes, les jalons et les dates importantes. Cela aide à organiser le travail sur une période donnée.

Ordre de Réalisation : Elle définit l'ordre dans lequel les fonctionnalités et les composants du logiciel seront développés. Cela peut inclure la priorisation des fonctionnalités en fonction de leur importance ou de leur complexité.

Découpage en Itérations (SPRINTs) : La planification globale découpe le projet en itérations (comme les SPRINTs en méthodologie agile), ce qui permet de travailler de manière itérative et d'obtenir des livraisons partielles de fonctionnalités fonctionnelles tout au long du projet.

Coordination d'Équipe : Elle aide à coordonner les efforts de l'équipe de développement en définissant les objectifs à court et à long terme. Cela garantit que tous les membres de l'équipe travaillent vers un objectif commun.

En somme, la planification globale offre une vue d'ensemble du projet et aide à l'organiser de manière efficace pour atteindre les objectifs fixés dans le cadre du développement logiciel. Elle est essentielle pour garantir que le projet se déroule de manière ordonnée et réussie.

Le modèle-vue-contrôleur (MVC) est un modèle de conception logicielle couramment utilisé dans le développement d'applications. Il vise à séparer les différentes responsabilités d'une application pour la rendre plus modulaire, évolutive et facile à maintenir. Voici ce que font généralement chaque composant (Modèle, Vue et Contrôleur) dans le modèle MVC et comment ils interagissent :

**Modèle (Model)** :

Le modèle représente la logique métier ou les données de l'application. Il contient généralement les structures de données, les algorithmes, les calculs et l'état global de l'application.

Le modèle ne connaît pas l'interface utilisateur (UI) ni la manière dont les données sont affichées. Il est indépendant de la vue.

Il offre des méthodes pour accéder et manipuler les données.

Le modèle peut notifier la vue lorsqu'il subit des modifications importantes (par exemple, en utilisant le modèle d'observateur) pour que la vue puisse se mettre à jour en conséquence.

**Vue (View)** :

La vue représente l'interface utilisateur de l'application. Elle est responsable de l'affichage des données et de l'interaction avec l'utilisateur.

La vue observe le modèle et réagit aux modifications en mettant à jour l'interface utilisateur en conséquence.

Elle ne contient généralement aucune logique métier. Au lieu de cela, elle se contente d'afficher les données et de transmettre les actions de l'utilisateur au contrôleur.

La vue peut également contenir des éléments d'interface utilisateur, tels que des boutons et des champs de texte.

**Contrôleur (Controller)** :

Le contrôleur agit comme un intermédiaire entre le modèle et la vue. Il reçoit les interactions de l'utilisateur depuis la vue et déclenche des actions appropriées sur le modèle.

Il contient généralement la logique de gestion des événements utilisateur, de la validation des données et de la coordination entre la vue et le modèle.

Le contrôleur peut également être responsable de la gestion de la navigation entre différentes vues (dans le cas d'applications plus complexes).

Il met à jour la vue en fonction des changements dans le modèle.

En résumé, le modèle gère les données et la logique, la vue gère l'interface utilisateur, et le contrôleur gère la coordination entre les deux. Ils communiquent principalement de manière unidirectionnelle : la vue observe le modèle pour afficher les données, le contrôleur réagit aux actions de l'utilisateur pour mettre à jour le modèle, et le modèle peut notifier la vue de ses changements. Cette séparation des responsabilités rend l'application plus facile à maintenir, à tester et à étendre, car chaque composant a un rôle bien défini et peut être modifié indépendamment des autres.

Design des algorithmes -> Scénarios d'utilisation ( La pièce maîtresse ) Peut être son propre client. Un mandat, est donné à une équipe pour la réalisation. Exercice de lecture, première étape : L'analyse textuelle. Sortir les expressions et les mots du document pour laisser les choses moins importantes. "Gros" "Vite" "Il en aura 12" (Adjectifs / Attributs) - Qu'est ce qui est essentiel. Implicite : Pourquoi? - Dans le document, mais pourquoi est-il là? L'objet est là, mais sert à quoi? Comment enrichir le programme. Supplémentaires : Peut-être qu'on pourrait faire plus. Baser sur notre expertise. Comment un utilisateur du logiciel va utiliser le logiciel ? - Les cas d'usage Caractéristiques des cas d'usage : "J'veux démarrer le logiciel" : Donc qui va afficher un screen, etc. "Jouer une partie", "Regarder les highscores", "Effacer les highscores" À quoi sert le logiciel pour la personne, l'utilisateur, en une phrase. Un début et une fin. "Faire une tarte" - Cas d'usage, mais -> 3e partie, scénarios d'utilisations : "Préparer les pommes" : Les épluchers Permettre à celui qui va coder : qu'est-ce que je vais mettre dans cette fenêtre ? : C'est déjà déterminer. En équipe on le détermine. - C'est le plus important. La partie maquette : manière dont le système sera représenter par l'usager. Représente le logiciel à travers des fenêtres. Les fenêtres ont les organises, en fonction de l'ergonomie. (Ne pas tout mélanger par dessus l'autre) La partie humaine des scénarios d'utilisations sont représentés par les fenêtres. La page de sauvegarde n'est pas la même que celle de jeu. (Par exemple lol) Les éléments en jeu seront nos objets. (Dans une class) - On définit à quoi va ressembler l'objet, comme écrire une recette. Ensuite, quand on va appliquer la recette, on pourra créer une vraie tarte. Comme créer l'objet grâce à la class qui a permit à le définir. Classe | Responsabilités de la classe (ses méthodes) | Colaboration [colaborateurs de la classe] (travailler avec quel autres objets) - CRC CRC : "à quoi est responsable cet objet là ?" Ex: Les pions, leur responsabilité c'est quoi ? : Bah de se déplacer. Les classes doivent avoir un propriétaire, attribuer à des personnes pour être sûre que cette personne est au courant de tout ce qui ce passe à propos de cette classe là. (Un extra) Modèle de donné : Préserver. Une structure de donnée Ex: Garder les noms, les scores et la date d'une partie. (Dans un fichier texte CSV) Qd on ouvre les fichiers en Python : le nom du fichier et 3 possibilités : 1. Mode lecture (Read) 2. W pour Write (Overwrite) 3. A pour ajout (Append) fichier.write(), fichier.close() Le planning global : Important - "Mon projet commence ici, et se termine ici". Ça se représente simplement. - C'est une liste de colonnes (pas très importantes les col). Important : 1 colonne avec des tâches qu'on peut lire verticalement dans le temps. Dans l'ordre dans lequel on va l'exécuter. 1. Le plus important pour le client en premier!!! 2. Des fois pour faire le plus important pour le client il faut faire des choses pour le client. "J'veux voir mon état de vente tous les mois" - Il te faut quelque chose pour saisir des données. 2. Des choses requises pour réaliser ces tâches là ?

Examen : Partie premiÃ¨re : analyse, design, planif Mandat : essentiel pour developper un logiciel, si il est pas Ã©rit notre responsabilitÃ© de l'Ã©crire (En notant ce que le client veut) On commence par l'analyse du texte en lecteur intelligente. CrÃ©er un logiciel pour 1 usager ou des usagers. Comment on sait si bonne lecture? En passant pas la petite grille pour sortir les mots clefs. On veut pas des phrases etc, il faut juste identifier ce qu'on demande dans le mandant. Ex: " 100 insectes " - on le met 1 fois on le repÃ¨te pas 15 fois. Y'a aussi une partie implicite "sous-entendu". Comment ? Bah : si on identie des noms qui represente de objets, mais on est pas capale de trouver d'actions, alors cmt est-il important si on sait pas Ã quoi il sert? Je trouve pas les caractÃ©ristiques. Donc "Implicite" : trouver Ã quoi Ã§a sert. On peut pas envoyer de couriel si on a pas d'adresse de couriel. C'Ã©tait pas Ã©crit mais c'est requis. Clarifier ce qui est lÃ pour que tout ce qui est implicite soit claire. SupplÃ©mentaire : en manipulant, on regarde des affaires, et on se dit "si on ramasse toutes ces informations lÃ , on fait un rapport trimestriel". On montre notre professionalisme Ã travers le supplÃ©mentaire. Ã€ quoi sert le logiciel et Ã qui ? - Cas d'usage. "Entrer des transactions?", etc. Ex: Action : "Transaction de vente", "Inscrire un client". Un dÃ©but et une fin - Cas d'usage. Des fois y'a des plus grands et des plus spÃ©cifiques. C'est pas forcÃ©ment absolu comme dÃ©finition. Quand on a fait la liste des tous les cas d'usage, on a du remarquer dansle mandant que des usages sont relatifs Ã certains usagers dans la hierarchie. Ex: "Le boss veut gÃ©rer Ã§a". On devrait avoir dÃ©terminer les rÃ´les des cas d'usage. Ex: Highscore, un admin peut sauvegarder le dernier mois des rÃ©sultats quelque part dans une base de donnÃ©es et pas n'importe qui. On sait faire Ã§a, mais comment? L'humain, et le CPU. L'humain veut ouvrir la page pr inscrire un client. Que va faire le systÃ¨me? - Fournir la page pour sauvegarder, annuler. Et quand le client va cliquer sur sauvegarder, l'ordinateur va aller lire la valeur des champs et t'afficher un msg pour te dire que la personne a bien Ã©tÃ© enregistrÃ©. Donc les scÃ©narios d'utilisation : On dÃ©termine les algortihmes de notre logiciel (Comment se fait Ã§a) !!! LA CHOSE LA PLUS IMPORTANT CAR ELLE VA GUIDER LE DESGIN DE NOS CLASSES !!! Maquettes : 2 caractÃ©ristiques importantes - 1. La question de regroupement des choses qui vont ensemble, si j'ai besoin de rentrer un nouv client, je vais garder les choses qui vont ensemble ensemble et les choses qui sauvegarde ailleurs. 2. La question d'esthÃ©tique, que ce soit facile et plaisant Ã retrouver les choses pour le client - CRC : On connaÃ®t les tÃ¢ches : Qui devrait Ãªtre responsable ? Les gens ont des tÃ¢ches spÃ©cialisÃ©es, mÃªme chose en prog objet, les objets ont des tÃ¢ches spÃ©cialisÃ©es. Avantage des objets, on sait ce que va faire cet objet lÃ et pas ce que tout le reste va faire au moment donnÃ©. Quand on a des logiciels plus sofistiquÃ©, on aura besoin de base de donnÃ©es (modÃ©lisation des donnÃ©es). On est en mesure de penser Ã notre planif global - On fait une liste. La seule vraie colonne qui compte c'est la premiÃ¨re, la colonne verticale des tÃ¢ches qu'on doit faire du haut vers le bas. 2 RÃˆGLES IMPORTANTES POUR PLANIF GLOBALE. 1. Ce qui est le plus important pour le client, on est tjrs Ã son service. 2. Il veut voir son rapport de vente mensuel, mais il peut pas si on inscrit pas les transactions. Pour faire les choses les plus importantes pour le client, on doit faire des choses requises avant. <! LA PLUS IMPORTANTE C'EST LA DÃ‰FINITION LA PLUS DÃ‰TAILLÃ‰E DE NOS ALGORTIHMES DANS NOS SCÃ‰NARIOS D'UTILISATIONS. !>

**Identifiez et expliquez les principes derrières la planification globale? (2) \***

En premier point la planification globale découpe le projet en itérations, ce qui permet d'envoyer des versions plus ou moins finis aux client (C'est le client le roi, donc on devrait priorisé ces besoins). Deuxièment, dans un cadre temporel elle permet de définir un ordre de réalisation du projet du plus important, pour que le client reçoit une application qui marchera même si tout n'est pas fini.

**Les "cas d'usage" ? Expliquez à partir de quoi on les identifie et expliquez les caractéristiques génériques de ce qui constitue un "cas d'usage". (2) \* (au moins deux aspects à préciser)**

Les cas d'usage on les identifies à partir de l'analyse textuelle. Se sont des fonctionnalités que le système doit offrir à ses utilisateurs ou son utilisateur. Ils décrivent l'action que le programme devrait faire par exemple, supprimer\_score ou ajouter\_score. Un cas d'usage peut être plus global comme lancer le jeux, mais il peut avoir des sous-parties comme, afficher le button. En bref, chaque cas d'usage peut avoir plusieurs scénarios d'utilisation associés.

**Relativement à l'architecture Modèle-Vue-Contrôleur, décrivez succinctement le rôle de chacun. (3)**

\* Modèle : Il contient la structure de données, les algorithmes, les calculs et l'état global de l'application. Important, il ne contient pas la manière dont l'interface graphique est afficher. Et cela est indépendant à la vue. Le modèle peut aussi notifier la vue si cela subit des changements. Vue : La vue affiche tout ce qui est pour l'interface graphique de l'application. Elle obsèrve le modèle et réagit en faisant des modifications pour afficher les bonnes choses. Le plus important c'est qu'elle ne fasse aucun calcul car ce n'est pas son rôle. En bref, elle permet d'afficher le jeux, les buttons, les figures, les couleurs etc. Contrôleur : Le contrôleur est un intermédiaire entre le modèle et la vue. Généralement il reçoit les intéractions nécéssaire pour faire marcher le code. Le contrôleur fait la validation de données, comme par exemple si l'utilisateur touche la lettre J pour jouer, alors on lancera le jeux .Il est aussi résposable de la gestion des événements entre différentes vues dans des applications plus complèxes.

**Les Scénarii d'utilisation constituent la pièce maîtresse de la modélisation de l'application envisagée - expliquez pourquoi. (3) \***

Les scénarios d'utilisation décrivent séquentiellement les étapes nécessaires pour accomplir chaque cas d'usage. Cela est important, car grâce au cas d'usage on pourrait fournir des exemples concret sur la façon dont le programme sera coddé. Plus qu'on est précis moins on aura des problèmes à construire le code. C'est surtout une instance spécifique et détaillé. C'est ici qu'on designera le logiciel et qu'on ferra la logique des choses. Exemple, l'utilisateur clique sur le button, donc la machine devra faire une action avec se button peut être, afficher un carré ou le bougé. Donc ce sont les algorythmes qui vont nous aider car elle vas nous guider a designer nos classes. Un manque de scénario d'utilisation est comme si on partait a codder sans rien savoir, sans aucune logique..