

Scrum

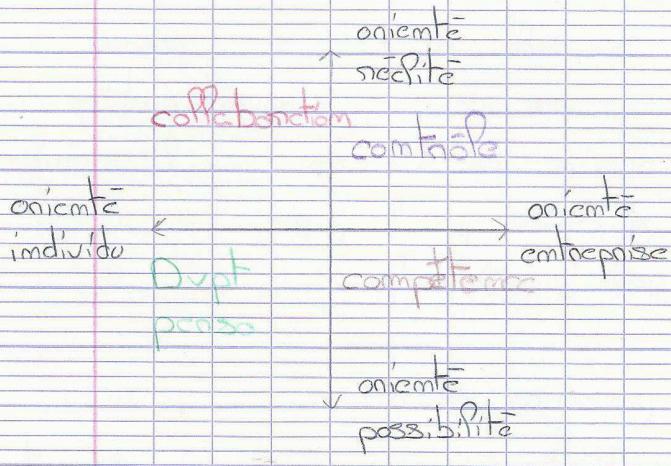
GL-1

Agilité: Se base sur le manifeste agile (12 règles) et sur 4 principes:

- + Des individus et leurs interactions
- + des logiciels opérationnels
- + la collaboration avec le client
- + l'adaptation au changement

Les cultures de l'entrepreneuriat:

culture: "La Scrum dont nous faisons les choses ici pour réussir"



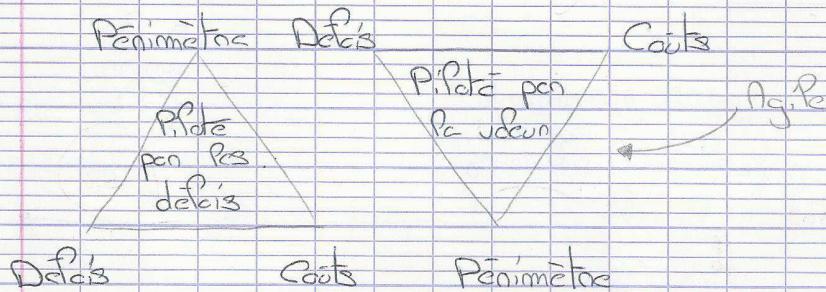
L'amélioration continue:

1. Lean startup

Ideas → build → product
→ measure → data → learn
→ ideas

2. Scrum: devloping circle
plan → do → check → act

Pilotage par valeur



Qualité

Visibilité

Épanouissement

Scrum

- Product owner : + définit les caractéristiques + priorise les fonctionnalités
+ décide de la date de livraison et du contenu + ajuste les priorités pour chaque sprint
+ négocie avec son investissement du produit + accepte ou rejette les résultats

- Équipe : + 5-9 personnes + dédiées au projet
+ pluridisciplinaires + auto-organisée

- Scrum master : + responsable de la mise en œuvre des valeurs scrum qui protège l'équipe de l'extérieur + élimine les obstacles + assure que l'équipe est fonctionnelle

Penties permanentes : personnes, ou groupe de personnes qui a des intérêts sur un projet et qui est concerné par les résultats obtenus

Cérémonies



INSPECTION et ADAPTATION

Sprint planning

Product Backlog → Sprint backlog

DoPy
scrum



working increment

↓
product
nouvelles

SPRINT PLANNING

Input

Product backlog

Latest increment

Velocity / capacity
of dev team

output \Rightarrow sprint goal
sprint backlog

Sprint planning

What will we do?

How will we do it?

RÉUNION QUOTIDIENNE

Qu'ci je fais bien ?

Que vais je faire ce jour d'aujourd'hui ?

Est ce que je rencontrerai des obstacles

Revues (5 min fin de chaque sprint)

- + toutes les personnes concernées
- + écrement du produit utilisable
- + chacun présente ce qu'il a fait
- + le po accepte ou rejette les résultats et met les notes

RETROSPECTIVE

- + mise en condition
- + revues des précédentes actions
- + noter ~~oublier~~ les données
- + chercher des idées
- + plan d'action

Vision d'un produit

Pour

Qui souhaitent

notre produit est

qui

à la différence de
permet de

Pilotage par le ROI RETURN ON INVESTMENT

- + développement itératif et incrémental
- + $ROI = \frac{\text{Benefit}}{\text{Initial investment}} / \text{gain net}$
- + induction des time to market

Négociation

- + pour coacter les développements
- + pour prioriser les fonctionnalités
- + un bon PO doit mettre le bon niveau de pression
- + un bon PO peut démontrer que le client ne sait pas ce qu'il veut

Fonction user stories

as <personae> I want <what> so
that <why>

Construction du backlog

claire écriture des stories

→ générer un maximum de stories, peu importe la taille / priorité

Estimations agiles

On estime la taille et non la durée

Utilisation d'échelles non linéaires

estimation en équipe (triangulation / poker picm)

estimation plusieurs fois moins avec une finesse différente

GL - Overview d'un projet informatique

GLS

Génie Logiciel : Maintenir des pratiques d'ingénierie et des méthodes basées sur des standards internationaux reconnus formellement même devant faire et celui de la communauté internationale.

Les delivery process

- + cycle en V : l'inclure avec effet tunnel
- + agilité (cf cours dédié)
- + itératif et incrémental : apport de valeur ajoutée par des cycles court et de l'accomplissement continue
 - prendre des décisions informées au fur et à mesure de l'avancement ; l'humain est itératif
 - itération est un mini projet
- + timeboxed (2-6 semaines) : objectifs vérifiables, résultats tangibles
- + aligne : définir et protéger les objectifs priorisés par les risques, la valeur métier
- + exécute : réaliser les objectifs et le pilotage au quotidien
- + assess : évaluer objectivement les résultats, comprendre leur impact pour la suite et effectuer une rétrospective

Démarrage

La vision est définie

Élaboration

Le niveau de spécification est suffisant

Construction et tests

La version complète de la méthode est disponible

Transition
Le client valide
le produit.

Modélisation métier

Capture des exigences

Analyse et conception

Implémentation

Tests

Déploiement

Gestion de config et chang.

Gestion de projet

Environnement

Les rôles sur un projet

- Chef de projet

- ↳ pilotage des équipes
- ↳ gestion du planning
- ↳ rédaction des spes
- ↳ reporting hiérarchie client

- Architecte

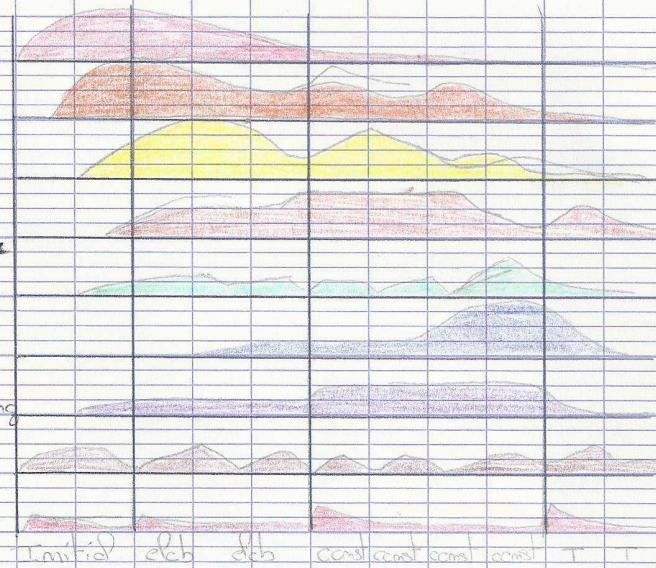
- ↳ concrétise l'architecture technique de la solution

- Responsable technique, ux et dev

- ↳ conception technique et implémentation
- ↳ support équipe technique
- ↳ gestion des configs, env, déploiements

- Responsable fonctionnel / testeur

- ↳ modélisation du besoin métier, des exigences
- ↳ analyse et conception fonctionnelle
- ↳ rédaction des spes
- ↳ rédaction et exécution des plans de tests



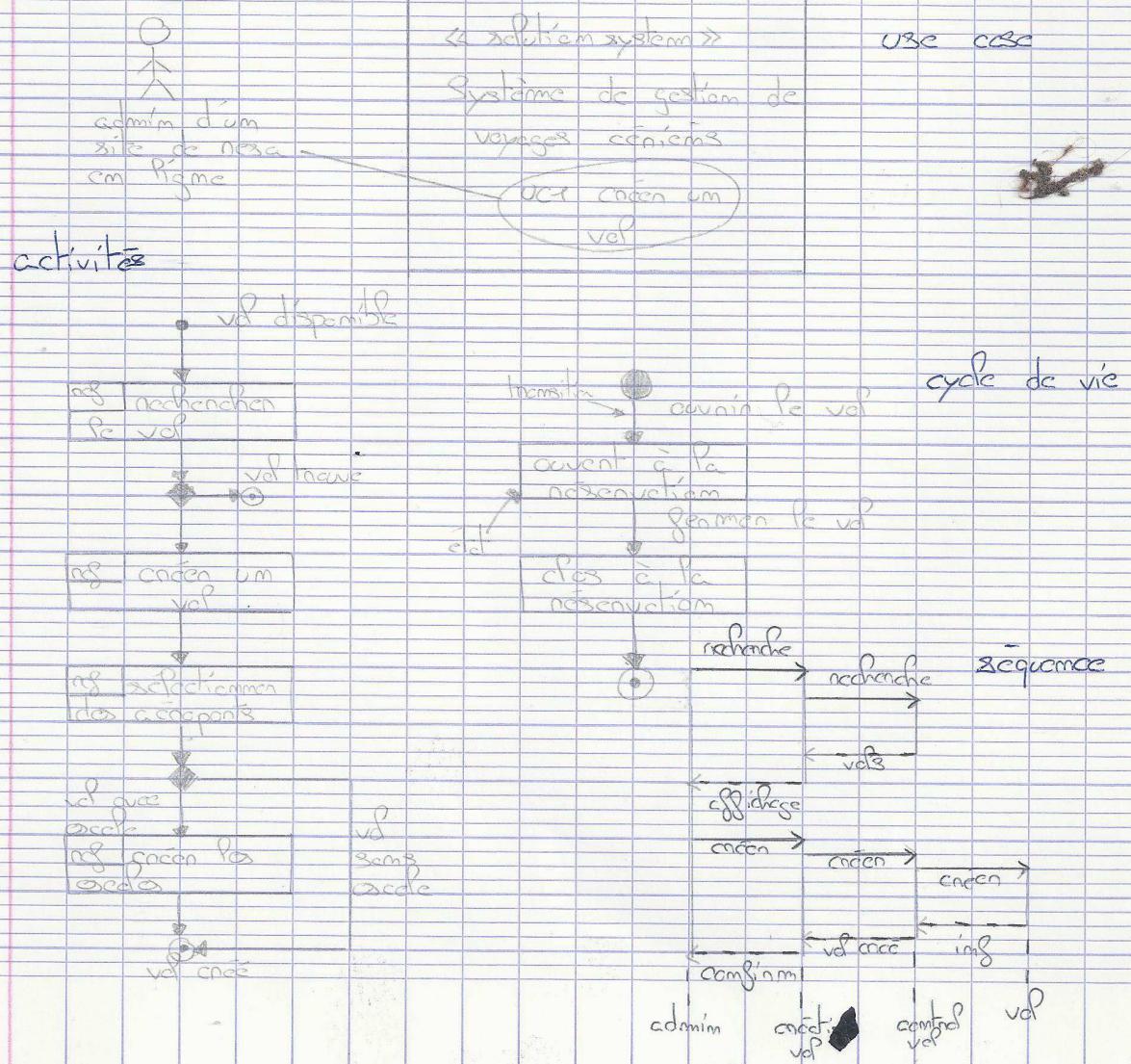
Le modélisation

GL4

Modélisation au déroulement du projet pour comprendre le besoin métier et tout au long du projet.

Les diagrammes UML :

- classe : pour partager le vocabulaire, les métiers / objets manipulés
- use case : pour identifier les utilisateurs et l'utilisation des grandes fonctionnalités
- activité : pour faire des workflow / des scénarios
- cycle de vie : pour présenter le cycle de vie d'un objet
- séquence : pour les présentations, les intentions entre les classes du système



Un objet métier est une abstraction d'éléments réels qui partagent les mêmes caractéristiques, un objet métier est créé dans un modèle

Pour dégager une association est bidirectionnelle mapping

Le nom de la relation et son sens de lecture sont importants uniquement mais permettent de dégager du vocabulaire

Personne employé travail pour complexe Société
0..* 0..*

Construire un référentiel neutreisible des composants du projet

Formaliser le modèle

Capitaliser le modèle

Générer des spécifications

Prémiser le modèle

Générer du code à partir des modèles

GL - Spécification et architecture technique de la solution GLS

Spécification fonctionnelle de la solution

↳ Capture des exigences + analyse et conception

Entrents

- Quoi ?

↳ Cliché des charges

↳ Expression des besoins

- Qui ?

↳ MoA - expert métier

- Pourquoi ?

↳ Expression des besoins utilisateurs

↳ Expression des contraintes non fonctionnelles

Sorties

Spec func gen (SFG)

Spec func détaillée (SFD)

Spec techniques détaillées (STD)

Rôle fonctionnel

Retraçage de la besoin dans la solution informatique

EXIGENCE: une exigence définit une caractéristique que doit posséder la solution pour répondre aux besoins de ses utilisateurs ou se conformer à une contrainte imposée

- unitaire : concerne un seul sujet

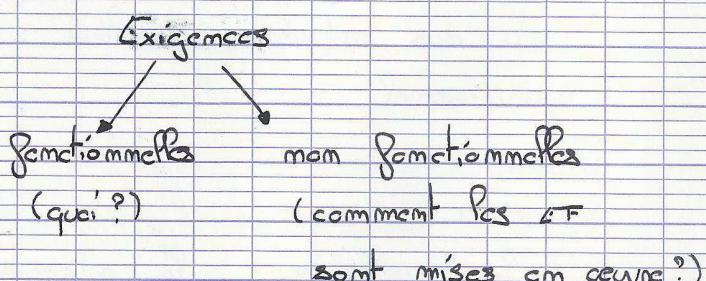
- cohérente

- complète

- faisable non ambiguë

- vérifiable

- vérifiable



Plus la détection d'une exigence fonctionnelle se trouve en aval du cycle de vie d'un projet, plus elle sera difficile à implémenter.

70% des défauts logiciels proviennent de l'expression des besoins et de leur spécification

Grande forme : une case

Forme élémentaire : une ligne

Décrire les cases atomiques

+ texte + maquette

+ classe + tests

Architecture technique de la solution

- analyse et conception + déploiement + gestion config et change + environnement

OBJECTIF :

- choisir et mettre en place les briques d'infrastructure technique
- expliquer à l'équipe les moyens de s'en servir

Éléments clés

L'architecte technique - Le référentiel technique est responsable de la cohérence entre l'architecte technique et les plateformes du projet.

- + ciblement défini
- + issu des exigences du client
- + opérationnel
- + expliqué à tous
- + partagé par tous

DEVOPS ET PIPELINE

1. Build et packaging : build automatiquement l'application et exécute les tests unitaires (compilation, tests auto, review, packaging)
microm
mpm junit gitlab gitlab docker

2. Analysis : assurer la qualité du code

(analyse statique code, tests statiques, tests statiques, gestion de flux thermis)
sonarqube sonarqube clair thermis
cppcheck cppcheck conteneurs

3. acceptation : effectuer des tests automatisés (mom negociaçao, peng cucumber, gatling, QCA) (secu dynamique, traceabilité).

4. déploiement : automatiser le build de l'image, installer les conf., déployer les appli (image, config, déploiement) (jenkins, ansible, docker)

GL - Construction de la solution

ENTRANTS

- Qui ?

- ↳ exigences, SFD, STD
- ↳ architecture technique opérationnelle
- ↳ nœuds et mondes de codage

- Qui ?

- ↳ Règlement fonctionnel
- ↳ Architecture technique

- Pourquoi ?

- ↳ conception du système
- ↳ dev du système
- ↳ tester

LIVRAISONS

- Quoi ?

- ↳ relâche prête à être testée

- Qui ?

- ↳ dev
- ↳ règlement technique
- ↳ règlement fonctionnel

Modularité dans l'architecture applicative

1. Architecture applicative

2. Architecture technique

Module : un module est une entité unité autonome qui procure des services au travers d'interfaces spécifiques pour un contexte.

Modularité :

1. Décomposable

2. Composable

3. Compréhensible

- couvert / gérable

- faible couplage

- fort cohésion

Sommes pratiques d'implémentation

- + debug book et format → tracer les exigences
- + maintenir le fonctionnel → sécurité
- + tracer les informations → identifier les codes smells
- + refactoring → éviter le rework
- + suivre la DT

Tests unitaires

- joue un rôle dans la doc
- plus rapide qu'un test menu
- peut être réduit à l'infini
- automatisé
- partagé avec toute l'équipe
- assure la bonne régularisation

Un bon test unitaire

- identifie les défaillances
- utile au dev
- mesure objective de la conformité
- indépendant des autres tests
- compréhensible & facile à maintenir
- cible ce qui est risqué

⇒ Le coût d'une correction augmente avec le temps

Donc : définit avant chaque projet, permet de définir si une tâche est terminée.

- correspond fonctionnellement à la usc story
- répond aux exigences architecturales
- niveau peu ou autre membre
- pas d'erreur bloquante
- couvre une TU complète.

GL4 - Tests de la solution.

GC7

Niveaux de tests

- Tests unitaires (beaucoup, précis, faible coûts)
- Tests intégration
- Tests qualification : (par équipe de tests)
 - Vérifier le respect des exigences
 - Valider la solution
 - Tests fonctionnels sur les cas d'utilisation, exigences func., processus métiers
 - Tests techniques sur les exigences non fonctionnelles
- Test accepte (peu, pas précis, très coûteux) (par le client)
 - solution prête à être déployée ?

TDD Méthode qui consiste à implémenter une partie de l'appli en se reposant sur les tests pour affiner le design et sécuriser l'implémentation

Les avantages :

- comprendre
- concevoir
- compléter
- stabiliser
- identifier les bugs
- corriger les bugs
- sécuriser
- séparer

Tests d'intégrations

Bouchon : module permettant de simuler un module inexistant / inaccessible.

1. Intégrer le bouchon : les interactions offrent des composants nécessaires (pas facile impossible)
2. Occupier complètement : toutes les interactions sont simulées (risque pour l'intégration)

L'intégration continue :

- valider fréquemment
- diminuer l'effet tunnel
- build auto
- imposer les tests dev
- discipliner l'équipe.

Vérification

Qualification

Besoins → Exigences → reflexe

Vérification : confirmation par l'examen Validation et la fourniture de preuves objectives que des exigences spécifiques ont été remplies.

Validation : confirmation par l'examen et la fourniture de preuves objectives que les exigences, pour un usage ou une application voulue, ont été remplies.

Conduire des tests permet de mesurer la couverture des tests fournis des indicateurs d'avancement de la qualification et de la qualité de la solution.

Recette

L'entreprise donne

- package de déploiement + doc
- tests
- méthode qualité
- périmètre l'un & réserves

Pour le client

- déploie
 - utilise ses données
- ⇒ connu ce qu'il faut pour M&P

M&P

- env principal (souvent)
- peut demander autre appli
- entreprise support