PROJET PROGRAMMATION SYSTEME

LIVRABLE 0

FODIL Nel | MARCELLI Enzo | GOUADFEL Rayan | ARRIGHI Fabien

31 janvier 2024

Table des matières

Table des figures

I. Intro

Ce rapport a pour objectif de faire office de « Fil Rouge » durant le projet. Il sert à maintenir la cohérence et l'alignement tout au long du projet. En cas de changement ou d'ajustement nécessaire pendant le projet, ce rapport servira de référence pour évaluer l'impact de ces changements sur les objectifs globaux.

II. Contexte

ProSoft, éditeurs de logiciels, a intégré votre équipe pour gérer le projet « EasySave » sous la supervision du DSI. Ce projet vise à développer un logiciel de sauvegarde intégré à la politique tarifaire de ProSoft, avec un prix unitaire de 200€HT. Le contrat de maintenance annuel inclut mises à jour et est fixé à 12% du prix d'achat, avec une reconduction automatique basées sur l'indice SYNTEC.

Les responsabilités de notre équipe incluent le développement, la gestion des versions et la documentation. Cette documentation comprend un manuel d'utilisation pour les utilisateurs et les informations nécessaires pour le support client. Cela permettra de faciliter la compréhension du logiciel pour le client, et de faciliter le travail du support client.

Afin de garantir une reprise de notre travail par d'autres équipes, la direction de l'entreprise nous impose de travailler en suivant certaines contraintes. Notamment le développement du logiciel en C#, l'utilisation de Git mais encore une certaine lisibilité et maintenabilité du code. L'objectif est de diriger ce projet de manière à réduire les coûts de développement des futures versions. Et surtout d'être capable de proposer rapidement des mises à jour en cas d'éventuels dysfonctionnement au sein du logiciel.

III. Organisation du projet

1. Méthodologie PDCA

La gestion du projet EasySave intègre l'approche itérative PDCA, un modèle de gestion de la qualité pour un contrôle et une amélioration continue des processus et des produits. Cette méthode, également connue sous le nom de "roue de Deming", est structurée en quatre phases :

Plan (Planifier) : Identification des objectifs, analyse des processus actuels et planification des changements nécessaires. Dans cette phase, nous définirons les exigences du logiciel EasySave, établirons les spécifications et concevrons les solutions pour répondre aux besoins identifiés.

Do (Faire) : Mise en œuvre des plans, souvent à petite échelle d'abord, pour tester leur viabilité. L'équipe développera une version bêta du logiciel, en appliquant les plans définis et en intégrant les fonctionnalités essentielles.

Check (Vérifier) : Suivi et évaluation des résultats par rapport aux attentes, analyse des données pour vérifier l'efficacité des changements. Nous évaluerons le logiciel en termes de fonctionnalité, de performance et de retour utilisateur.

Act (Agir) : Action basée sur ce qui a été appris dans la phase de vérification. Si les changements n'ont pas fonctionné, ils seront revus et affinés. Si les résultats sont satisfaisants, les changements seront implémentés à plus grande échelle et le logiciel sera finalisé pour la production.

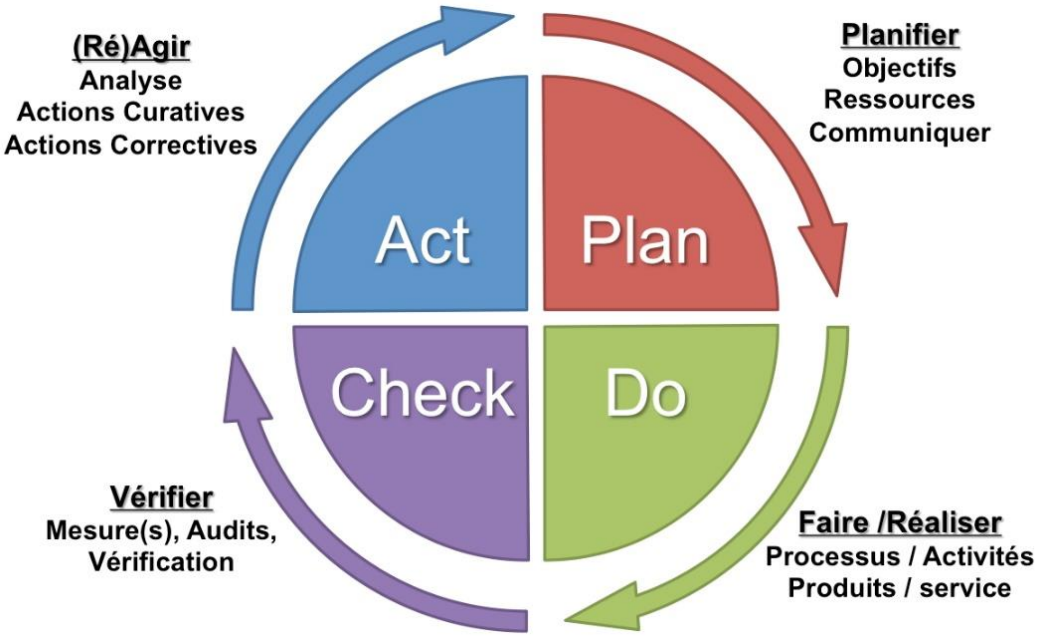

Figure 1 : PDCA

2. Rôles d'Équipe

Les rôles au sein de notre équipe ont été définis pour favoriser une collaboration optimale et efficace :

- Animateur (Fabien) : Assumera le rôle de facilitateur, s'assurant que toutes les réunions et sessions de travail collaboratif se déroulent sans heurts et que chaque membre de l'équipe est engagé et contribue pleinement.
- Gestionnaire de Temps (Nel) : Prendra en charge le suivi du planning et veillera à ce que le projet reste dans les temps. Il sera responsable de rappeler les échéances et de s'assurer que les objectifs sont atteints en temps voulu.
- Secrétaire (Enzo) : S'occupera de la documentation du projet, y compris la mise à jour et l'archivage des comptes-rendus, des décisions prises et des modifications du cahier des charges.
- Scrib (Rayan) : Aura pour mission de documenter les discussions clé, rédiger les synthèses et s'assurer que les points importants sont communiqués à toute l'équipe.

3. Planification

Le projet EasySave a été planifié en trois phases principales, avec une attention particulière portée à la gestion efficace du temps et des ressources. La première phase, Initiation et Conception, se déroule du 29 janvier au 8 février 2024. Pendant cette période, l'équipe se concentrera sur la compréhension approfondie du cahier des charges et la mise en place de l'environnement de développement, suivies par la planification initiale et le début de la programmation de la version 1.0 du logiciel, incluant la création des premiers diagrammes UML.

La seconde phase, Développement et Amélioration, du 9 au 16 février 2024, permettra l'élaboration des versions 1.1 et 2.0. Cette période sera consacrée à l'amélioration des fonctionnalités existantes et au développement de nouvelles fonctionnalités, tout en continuant à documenter le processus à travers des diagrammes UML.

La troisième et dernière phase, Finalisation et Préparation de la Soutenance, prévue du 19 février au 1er mars 2024, se concentrera sur le peaufinage du livrable final, la version 3.0, et la préparation de la présentation du projet. Cette étape cruciale comprend la finalisation du développement, la mise à jour des diagrammes UML pour le livrable 3 et les préparatifs pour la soutenance du projet.

Tout au long du projet, une approche agile sera adoptée pour garantir flexibilité et réactivité face aux changements. Cela signifie que le projet peut évoluer en fonction des retours, des révisions du cahier des charges et des demandes des parties prenantes. L'objectif est d'assurer que le logiciel EasySave non seulement répond aux besoins initiaux mais est également adaptable aux exigences futures.

4. Diagramme de GANTT

EasySave

2F Roaming

lun, 1/29/2024

1

TÂCHE	DÉBUT	FIN
Phase 1 : Initiation et Conception	29/1/24	8/2/24
Lancement du Projet : Comprendre le cahier des charges et préparer l'organisation du projet.	29/1/24	29/1/24
Préparation de l'environnement de travail (installation de Visual Studio, .NET Core, Git, ArgoUML).	29/1/24	29/1/24
Développement du Livrable 0 : Planification initiale, conception de base	29/1/24	31/1/24
Développement du Livrable 1 (Version 1.0) : Programmation, codage et documentation initiale.	1/2/24	8/2/24
Création des Diagrammes UML pour la Version 1.0: Développement des diagrammes UML.	1/2/24	8/2/24
Phase 2 : Développement et Amélioration	9/2/24	16/2/24
Début des Versions 2.0 et 1.1 : Lancement et planification du développement des nouvelles versions.	9/2/24	9/2/24
Développement Version 1.1 : Travail sur les améliorations et les ajustements	12/2/24	14/2/24
Développement Version 2.0 : Début du développement des fonctionnalité	14/2/24	16/2/24
Création des Diagrammes UML pour le livrable 2	12/2/24	16/2/24
Développement du Livrable 2 : EasySave versions 2.0 et 1.1	12/2/24	16/2/24
Phase 3 : Finalisation et Préparation de la Soutenance	19/2/24	1/3/24
Création des Diagrammes UML pour le livrable 2	19/2/24	22/2/24
Développement du Livrable 3 : Développement final et préparation pour la version 3.0.	23/2/24	29/2/24
Préparation de la Soutenance : Finalisation des préparatifs pour la présentation du projet.	1/3/24	1/3/24

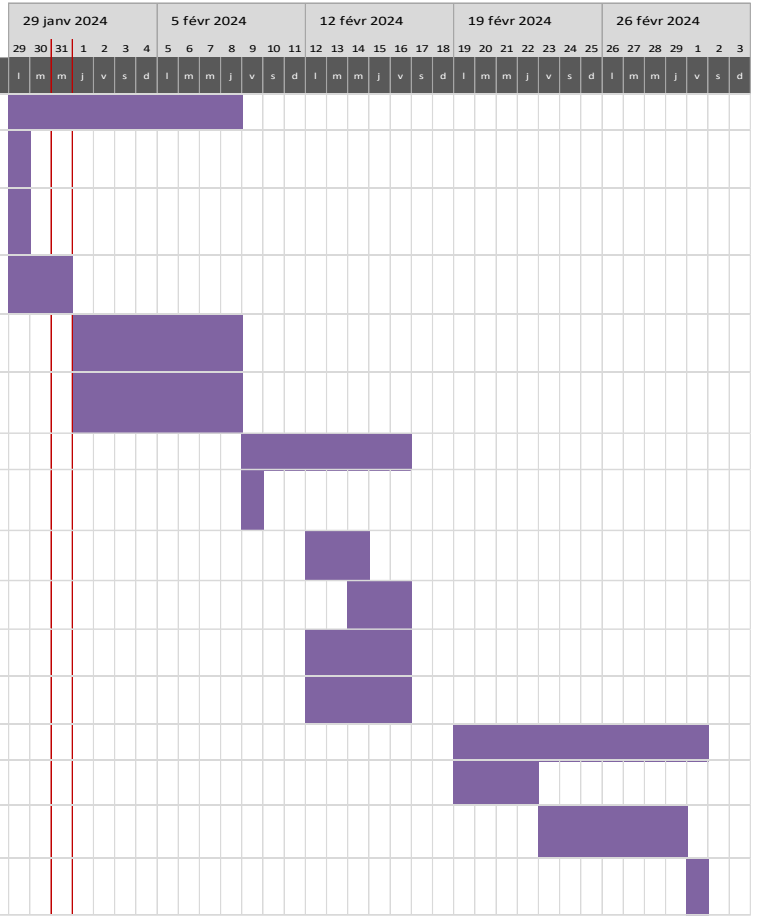

Figure 2 : GANTT

IV. Technologies utilisées

1. .NET

Pour réaliser le projet EasySave qui nous a été confié nous allons utiliser le Framework .NET sous sa dernière version .NET 8.0
Initialement .NET est sorti sous le nom de .NET Framework, ayant pour objectif d'être une solution de développement sous Windows exclusivement. Par la suite a été créé .NET Core avec le but d'offrir une meilleure interopérabilité, conçu pour être multiplateforme, il peut être utilisé sur Windows, Linux et MacOs. .NET Core a été distribué avec un ensemble de CLR compatible sur ces différents systèmes d'exploitation. De plus, Microsoft a Open-Sourcé .NET Core pour que chacun puisse y contribuer.
Finalement arrive la création de .NET STANDART, avec le but d'être un sous-ensemble du Framework comme un standard accessible aussi bien en .NET Core qu'en .NET Framework.
Il y avait alors trois Frameworks, et cela commençait à causer certaines confusions au sein de la communauté. Pour pallier ce problème Microsoft a tenté une unification pour apporter plus de cohérence. Ils ont donc créé l'unique .NET avec pour première version .NET 5.0, qui est le socle de toutes les versions de .NET à venir.
Avec .NET on retrouve de nombreuses bibliothèques et de Frameworks qui existe en .NET, par exemple WinForms qui est un ensemble de bibliothèque graphique pour le développement d'UI pour les applications Windows. Il existe le Framework ML.NET pour le machine learning, les Frameworks ASP.ET et Blazor pour le Web.

.NET prend en charge plusieurs langages de programmation tels que C#, F# et Visual Basic, permettant aux développeurs de choisir le langage qui convient le mieux à leurs compétences et à leurs besoins.
Pour fonctionner, .NET s'appuie sur son composant central, le CLR (Common Language Runtime). Le CLR est une machine virtuelle qui gère l'exécution des programmes .NET. Il offre des fonctionnalités telles que la gestion de la mémoire, la compilation à la volée, la gestion des exceptions, etc. Le CLR assure également la sécurité en exécutant le code dans un environnement isolé, appelé bac à sable.

Pour assurer la compatibilité il existe le CIL (Common Intermediate Language), c'est un langage intermédiaire utilisé par le CLR. Lorsqu'on compile un programme .NET, il se retrouve

traduit en CIL au lieu du langage machine spécifique à une plateforme. Cela permet au code d'être portable et de s'exécuter sur n'importe quelle plateforme compatible avec le CLR.

1- Plus concrètement le code source est écrit dans un langage tel que le C#
2- Le code source est compilé en code intermédiaire CIL à l'aide du compilateur de langage (compilateur C#)
3- Lors de l'exécution, le CLR charge le code intermédiaire CIL et le compile en code machine spécifique à la plateforme.

2. C#

Comme nous l'avons vu .NET propose de nombreux langages, pour ce projet nous allons utiliser C#. Ce dernier est un langage de programmation orienté objet créé en 2000, il a été conçu pour développer sur la plateforme Microsoft .NET.
C# est un langage dérivé du C++ et avec une certaine ressemblance à Java. Il propose des caractéristiques comme un typage sûr, de l'encapsulation, de l'héritage et du polymorphisme. Il comporte également un ramasse-miettes et un système de gestion d'exceptions.
C'est donc avec C# que nous allons développer notre application Windows.

3. Environnement de développement

Pour ce projet, toute l'équipe travaillera sur Visual Studio 2022. Visual Studio est un environnement de développement intégré (IDE) qui regroupe plusieurs outils permettant de créer, éditer, déboguer et gérer le code plus facilement. Dans notre cas, nous allons devoir par la suite réaliser une interface graphique, Visual Studio possède également les fonctionnalités nécessaires pour réaliser ce genre de projet.

Nous n'utiliserons aucune extension pour ce projet mais un paramétrage commun sera fait pour le formatage du code.

4. Git et Versioning

Afin d'assurer le suivi de chaque version de notre code et de faciliter la collaboration entre les développeurs, nous utiliserons Git et GitHub. Le premier outil est un logiciel de gestion de versions qui permettant d'avoir une vue complète de la base de code ainsi que de son historique sur tous les postes de travail des développeurs. GitHub quant à lui, est un service web permettant de sauvegarder un dépôt de manière distante, favorisant ainsi la collaboration au sein de l'équipe.

Nous utiliserons également GitHub afin d'appliquer des règles de sécurités à notre branche principale. En effet, il ne sera pas possible de pousser du code directement sur la branche principale. Au lieu de cela, les développeurs devront d'abord effectuer un commit sur une autre branche, puis soumettre une pull request qui devra être validée par une ou plusieurs personnes.

Notre dépôt sera constitué de deux branches principales :

- `*main*` qui contiendra le code de production stable.
- `*dev*` où les fonctionnalités sont fusionnées au fur et à mesure de leur achèvement.

D'autres branches seront créées au fur et à mesure de l'avancement du projet pour travailler sur le développement des fonctionnalités ou la correction de bug.

Une convention de nommage pour les commits sera également mise en place avec la rédaction du message de commit en anglais et avec l'utilisation de mots clés ou autres afin d'identifier facilement les modifications apportées par les commits.

Bien que la grande majorité des manipulations puissent être faites directement via ligne de commande avec Git, l'utilisation d'une interface graphique se révèlera être utile dans certains cas tels que la gestion de conflit. Pour cela, nous utiliserons l'application GitHub Desktop.

5. Tests / Déploiement

L'utilisation de Jenkins et Docker ensemble nous permettra de créer un processus de développement, de test et de déploiement robuste, automatisé et efficace, aligné avec les meilleures pratiques de DevOps. Cette configuration contribue non seulement à l'efficacité opérationnelle, mais améliore également la fiabilité et la qualité du logiciel que nous déployons pour EasySave.

5.1. Utilisation de Docker pour les Tests

Dans notre projet EasySave, Docker jouera un rôle essentiel pour créer des environnements de test isolés et reproductibles. En encapsulant notre application et son environnement dans des conteneurs Docker, nous assurons que nos tests sont effectués dans des conditions constantes, réduisant ainsi les variables et les problèmes de "ça marche sur ma machine".

Nous utiliserons des images Docker pour définir les configurations nécessaires des systèmes d'exploitation et des piles logicielles. Ceci garantit que chaque membre de l'équipe, indépendamment de son environnement de développement local, peut exécuter des tests fiables et cohérents. De plus, nous envisageons d'utiliser les extensions disponibles pour les frameworks de test, comme NUnit ou xUnit pour .NET, qui peuvent être intégrées dans nos conteneurs Docker pour automatiser les tests unitaires et d'intégration.

5.2. Mise en Place du Serveur Jenkins pour les Tests et le Déploiement

Jenkins, un serveur d'intégration continue (CI) et de déploiement continu (CD), sera au cœur de notre pipeline d'automatisation. Nous configurerons Jenkins pour qu'il surveille notre dépôt Git et déclenche des builds automatiques à chaque nouvelle soumission de code (commit).

Chaque build sera accompagné d'une série de tests automatisés qui valideront les modifications. Si les tests sont concluants, le build pourra être marqué comme prêt pour le déploiement. En cas d'échec, l'équipe sera alertée pour corriger les problèmes avant de procéder.

5.3. Configuration des Pipelines CI/CD dans Jenkins

Nos pipelines CI/CD dans Jenkins seront configurés pour effectuer les actions suivantes :

- **Extraction du Code** : À chaque commit poussé sur la branche de développement ou à chaque pull request, Jenkins extraira le code du dépôt Git pour débuter le processus de CI/CD.
- **Exécution des Tests** : Jenkins exécutera les tests unitaires et d'intégration dans les conteneurs Docker pour s'assurer que le code soumis répond aux critères de qualité définis.
- **Build et Rapports de Tests** : Après les tests, Jenkins compilera le code et générera des rapports de tests qui seront mis à disposition de l'équipe.
- **Déploiement Automatisé** : Une fois validé, le code sera automatiquement déployé dans l'environnement de staging pour des tests plus approfondis ou directement en production selon la stratégie de déploiement définie.

6. Diagramme UML

Les Diagrammes UML (Unified Modeling Language) sont un élément clé dans la planification et la documentation de notre projet EasySave. Pour la création de ces diagrammes, notre choix initial s'est porté sur ArgoUML. Cet outil open-source offre les fonctionnalités nécessaires pour créer des diagrammes UML complets.

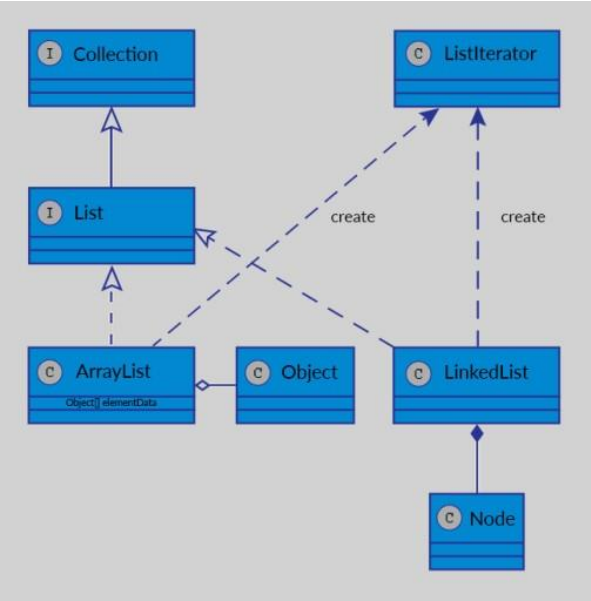

Figure 3 : Exemple diagramme UML

Ces diagrammes nous permettent de visualiser, de spécifier, de construire et de documenter les parties du système de logiciels en développement. Dans le cadre de notre projet, les UML joueront plusieurs rôles cruciaux :

- Conception de l'Architecture
- Communication entre Équipe
- Analyse des Besoins
- Documentation du Projet

V. Pipeline CI/CD

Le principal objectif d'une pipeline CI/CD (Continuous Integration/Continuous Delivery) est d'automatiser et d'optimiser le processus de développement logiciel, du moment où le code est modifié jusqu'à son déploiement en production.
Durant notre projet, nous allons donc devoir développer notre application. Afin d'assurer un environnement de travail professionnel permettant le bon développement de notre projet, nous allons devoir mettre en place certaines bonnes pratiques. Ces dernières auront pour objectif de suivre l'intégration des parties développées par chaque membre de l'équipe. Permettant ainsi d'éviter des conflits de versions, de valider le bon fonctionnement du code et en général de garantir une certaine qualité, rapidité et fiabilité tout au long du cycle de vie de notre application.

1. Intégration Continue

L'intégration continue est la première étape cruciale d'une pipeline CI/CD. Son objectif principal est d'automatiser et de rationaliser le processus d'intégration des modifications apportées au code source dans un référentiel partagé. Lorsqu'un développeur pousse des modifications, la CI déclenche automatiquement des étapes telles que la compilation du code, l'exécution de tests automatisés, et l'analyse statique du code. L'objectif principal de cette automatisation est de détecter rapidement les potentielles erreurs de développement et/ou incompatibilité, pour ainsi assurer une base stable pour les phases de développement ultérieur.

Cette pratique vise à garantir une intégration harmonieuse des contributions individuelles des membres de l'équipe au sein du projet. Ainsi, la collaboration entre les membres se retrouve simplifiée. En détectant par le biais de test les problèmes dès leur apparition, la CI continue à réduire les risques liés à l'intégration tardive, permettant une progression rapide du projet et des cycles de développement plus courts. D'un point de vue financier, résoudre un problème en phase de développement est bien moins coûteux qu'une fois l'application lancée, mais aussi bien plus simple.

2. Déploiement Continue

Le déploiement continu constitue la phase suivante, se concentrant sur la livraison automatisée et fiable du logiciel après la phrase d'intégration. Il consiste à déployer les nouvelles modifications de code dans un environnement de production dès qu'elles passent les étapes de test et de construction. Cela permet d'avoir une chaîne d'automatisation rapide et efficace pour ajouter à l'application des nouvelles fonctionnalités et des correctifs de bugs par exemple.

La CD automatisée comprend généralement des étapes telles que la création de builds déployables, la validation de ces builds à travers des tests de déploiement, et finalement le déploiement automatisé dans l'environnement de destination. L'objectif est de minimiser les interventions manuelles dans le processus de déploiement, réduisant ainsi les erreurs potentielles et accélérant le cycle de livraison.

Le déploiement continu est essentiel pour répondre au besoin croissant de livraison version dans le code, tout en garantissant une certaine fiabilité de test. Pouvoir déployer rapidement des fonctionnalités ou des correctifs est important, mais le faire en gardant la stabilité du système l'est tout aussi. C'est donc le déploiement continue qui va garantir cette certaine rapidité tout en évitant d'ajouter d'éventuels problèmes à la suite du déploiement.

3. Jenkins

Nous l'avons vu, mettre en place une pipeline CI/CD est une pratique obligatoire à mettre en place pour effectuer un projet de développement dans les meilleures conditions possibles.
Nous allons donc utiliser Jenkins qui est un outil open source d'intégration et de déploiement continu développé en Java. Jenkins se charge automatiquement de recompiler et de tester chaque modification de code. Lors de la détection d'une erreur, il alerte le développeur afin qu'il résolve le problème.
Actuellement, nous n'avons pas une vision assez grande du projet pour faire les choix les plus appropriés. En avançant dans le projet, nous serons sûrement amenés à réaliser des changements quant à nos choix. Dans le cas où Jenkins ne serait pas approprié, nous choisirons un autre outil. Cela vaut pour tous nos choix pris en ce début de projet.

VI. Conclusion

En conclusion du projet "EasySave", nous avons efficacement combiné des méthodes de gestion de projet rigoureuses, comme PDCA, et une organisation d'équipe claire, avec des rôles bien définis pour chaque membre. L'utilisation judicieuse de technologies modernes telles que .NET, C# et des outils comme Docker et Jenkins a renforcé la qualité et l'efficacité de notre processus de développement. Notre capacité à créer et à gérer un pipeline CI/CD robuste a assuré un déploiement et une intégration continue efficaces. Ce projet ne cherche pas non seulement à répondre aux exigences techniques et commerciales de ProSoft mais a également posé des bases pour l'évolution future du logiciel, démontrant notre compétence à naviguer dans des environnements technologiques complexes tout en atteignant des objectifs clairs.

VII. Webographie

https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

https://searchsoftwarequality.techtarget.com/feature/Visual-Studio-IDE-offers-many-advantages-for-developers

https://docs.microsoft.com/en-us/dotnet/framework/

https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

https://en.wikipedia.org/wiki/Release_notes

https://rogerdudler.github.io/git-guide/index.fr.html

https://tropars.github.io/downloads/lectures/DevOps/devops_11_Integration_Continue.pdf

https://www.youtube.com/watch?v=jmn1EgMYE3Q

https://www.lebigdata.fr/integration-continue-definition

https://www.lebigdata.fr/docker-definition

http://www.morere.eu/IMG/pdf/virtualisation.pdf

https://www.wavestone.com/app/uploads/2017/09/2017-SyntheseDEVOPS_VF_WEB.pdf