INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

PROJET INDUSTRIEL 5e année

présenté par
Marine LIEPCHITZ
Antoine BUAN
Alexis JUVIN
Tudy GOURMELEN
Spécialité SRC - EII

Année universitaire 2019 - 2020

Projet GITUS

Entreprise partenaire
La Ruche - Thalès (Rennes)

Tuteur en entreprise
Jean-François COUTTEAU

Correspondante pédagogique INSA Rennes
Merium OUTTAS

LA RUCHE THALES

Remerciements

Nous souhaiterions remercier Jean-François COUTEAU, responsable technique de La Ruche - Thalès, pour nous avoir encadré tout au long du projet. Nous remercions l'ensemble de l'équipe de La Ruche - Thalès pour leur implication personnelle dans ce projet.

Nous remercions également Erwan BOULAIN pour nous avoir accueilli au sein de l'entreprise. Nous remercions sincèrement Xavier HOCHART pour avoir pris le temps de nous former à la méthode agile et ainsi que Jean-Baptiste LE STANG ; Adrien AUBRY, Cedric CIVEIT, Jerome GIRARD et Thierry LEMOINE pour s'être rendu disponibles pour répondre à nos questions techniques.

Nous tenons à remercier Meriem OUTTAS, notre tutrice INSA, pour son support et son investissement dans le projet. Nous remercions également Mickaël DARDAILLON et Wassim HAMIDOUCHE, responsables INSA de ce module de projets industriels, pour la réalisation d'un projet en partenariat avec une entreprise en lien avec nos secteurs d'application.

Pour finir, nous remercions chaque membre de l'équipe qui a su s'investir pleinement pour mener à bien la mission confiée.

Introduction

Dans le cadre d'un projet industriel s'adressant aux étudiants en 5e année des filières Système et Réseau de Communication (SRC) et Électronique et Informatique Industrielle (EII) de l'INSA Rennes, l'entreprise La Ruche - Thalès a soumis un projet intitulé GITUS. Ce projet a émergé au sein du groupe suite à une problématique de partage et suivi de données liée à la gestion de projets en méthodologie Agile. Notre groupe a pris à la charge de trouver une solution à ce problème. Pour se faire, notre groupe est composé de quatre étudiants, tous issus de parcours différents :

- Marine Liepchitz, étudiant en filière SRC classique
- Tudy Gourmelen, étudiant en filière SRC Media Network
- Antoine Buan, étudiant en filière EII classique
- Alexis Juvin, étudiant en filière EII en double diplôme à l'IGR.

Notre mission consiste donc en le développement d'un outil de gestion de projet pour améliorer la communication entre les développeurs, appelés Product Owner (PO) et les personnes en lien direct avec le client, appelés Business Owner (BO). Notre rôle sera donc de proposer un système qui puisse synchroniser et tracer les modifications apportées par différents collaborateurs tout en liant ces modifications aux tâches qui découpent le projet en plusieurs fonctionnalités, appelées user stories. Les user stories sont amenées à évoluer en permanence au cours d'un projet aussi bien d'un point de vue développeur que d'un point de vue client. Thalès utilise des outils de gestion tels que les logiciel Taiga et Jira pour gérer cette problématique. Toutefois, l'utilisation de ces outils est limitée par les consignes de sécurité liées au secteur de la défense. En effet, par soucis de sécurité, les personnes de La Ruche-Thalès en collaboration avec un client externe ne peuvent pas accéder à internet. Cela rend la transmission des informations entre PO et PO compliquée étant donné qu'ils travaillent sur deux réseaux différents avec pour seule passerelle, une clé usb. Nous pouvons nous aider de tout le matériel dont dispose l'entreprise pour déployer notre solution. Cependant, nous devons respecter les contraintes de sécurité imposées par l'entreprise afin que le prototype s'intègre facilement dans leur système. Ainsi, l'objectif vers lequel nous devons tendre est un prototype open source qui puisse faciliter les interactions des deux environnements déconnectés et sécurisés.

Les deux principaux enjeux du projet sont le travail en équipe ainsi que la collaboration avec une entreprise partenaire dans le but de réaliser un prototype final correspondant aux attentes de l'entreprise. Cette démarche pédagogique vise à nous former à la gestion de projet. La communication et la planification seront des éléments clés pour mener à bien cette mission. Il est attendu de nous que nous soyons capables de monter en compétences de manière autonome

à partir des supports fournis par l'entreprise. Cet exercice a pour but de nous familiariser un peu plus avec le monde de l'entreprise en nous baignant dans des conditions très proches de celles que nous serons amenées à trouver dans notre future carrière d'ingénieur.

Sommaire

Table des figures

1 Environnement du projet

1.1 Le groupe Thalès

Thalès est un groupe d'électronique spécialisé dans différents domaines :

- Aéronautique
- Espace
- Transport terrestre
- La sécurité informatique
- La défense

Lors de ces dernières années, Thalès a investi dans le domaine de la cybersécurité notamment avec le rachat d'entreprises spécialisées. Cet intérêt pour la cybersécurité est aussi marqué par des financements en R&D et par la création de sites spécialisés comme celui de Rennes.

FIGURE 1.1 – L'entreprise Thalès en quelques chiffres

1.2 La Ruche de Thalès (Rennes)

Inauguré en 2019, ce nouveau site du groupe Thalès accueille 60 employés. Le dynamisme de la métropole de Rennes pour le domaine de la cybersécurité est croissant avec de nombreuses PME/Startups mais aussi de grands groupes comme Airbus. La Ruche a pour principal client la Direction Générale de l'Armement implantée à Bruz mais est aussi amenée à travailler avec divers collaborateurs présents localement.

L'objectif de la Ruche de Thalès avec ces méthodes innovantes et agiles est de permettre de livrer des logiciels répondant aux critères de leurs clients plus rapidement. Alors que des projets de développement "classique" peuvent demander plusieurs années de travail, la Ruche est capable de réaliser un logiciel fonctionnel en 2-3 mois. Cette version peut ainsi être améliorée ou modifiée lors des mois suivants grâce aux feedbacks fournis par le client lors de la livraison de la première version.

Les équipes du site travaillent sur des projets de cyberdéfense spécialisés dans les domaines aérien et numérique. Tous ces travaux sont réalisés en collaboration avec le ministère des Armées et la Direction Générale de l'Armement. La Ruche doit alors disposer de différentes compétences pour mener à bien ces projets ; intelligence artificielle, développement d'outils informatiques ou encore du suivi et de la gestion des projets.

1.3 Méthodes de travail de l'entreprise

Actuellement, la société La Ruche - Thalès gère ses différents projets à l'aide de la méthodologie Agile qui est fortement utilisée depuis quelques années car elle a montré un taux de réussite des projets plus élevé que la méthode de développement classique du cycle en V.

C'est une approche de gestion de projet qui considère que le besoin du client ne peut être figé. La méthode agile part du principe qu'il est contre-productif de planifier en détails la totalité du projet avant de développer. Le projet est divisé en plusieurs sous projets afin de fixer des objectifs à court terme dans le but d'avancer pas à pas vers l'objectif final. Cette approche plus flexible laisse place aux imprévus et aux changements. La satisfaction du client étant la priorité c'est pourquoi le client est impliqué du début à la fin du projet. Le client est chargé de valider chaque étape du projet. Cela permet de suivre efficacement l'évolution de ses besoins dans le but d'effectuer les ajustements en conséquence.

1.3.1 Implémentation de la méthodologie scrum

L'approche Scrum est la mise en application choisie par Thalès pour appliquer la méthodologie Agile. Le développement d'un projet au sein de l'entreprise se déroule selon une organisation prédéfinie (cf figure 1.2) et demande l'acquisition d'un vocabulaire bien spécifique.

Un projet est découpé en sprint, d'une durée fixe comprise entre 2 à 4 semaines. Pendant cette période, une version du produit est réalisée pour répondre aux objectifs fixés en début de sprints comprenant les fonctionnalités à implémenter. Dès qu'un sprint se termine, de nouveaux objectifs sont définis. Une réunion de planification, appelée sprint planning, organisée par le Scrum Master, a lieu avant chaque sprint, pour élaborer ces objectifs. Une revue de sprint appelée sprint review a lieu en chaque fin de sprint pour faire le bilan du sprint passé et faire le point sur les objectifs pour déterminer s'ils ont été atteints ou non. C'est aussi l'occasion pour l'équipe de s'exprimer sur ce qui a fonctionné ou non durant ce sprint. La mêlée est une réunion quotidienne entre tous les membres de l'équipe au cours de laquelle chacun prend la parole et répond aux questions suivantes : "Qu'est ce que j'ai fait la vieille ? Que vais-je faire

aujourd'hui ? Quels sont les problèmes que j'ai rencontré ?". Elle dure 15 minutes. Elle très importante pour se tenir au courant de la progression de l'équipe dans le sprint. Le product backlog est une liste des fonctionnalités à implémenter, déterminées par le client. Ce document est amené au cours du temps en fonction de l'évolution du besoin du client. Le sprint backlog est l'ensemble des objectifs à atteindre pour un sprint. L'équipe est chargée de mettre à jour l'état des objectifs au cours du sprint. L'incrément est le résultat de l'ensemble des éléments terminés du sprint backlog en fin de sprint. C'est une version du produit qui se doit d'être exploitable.

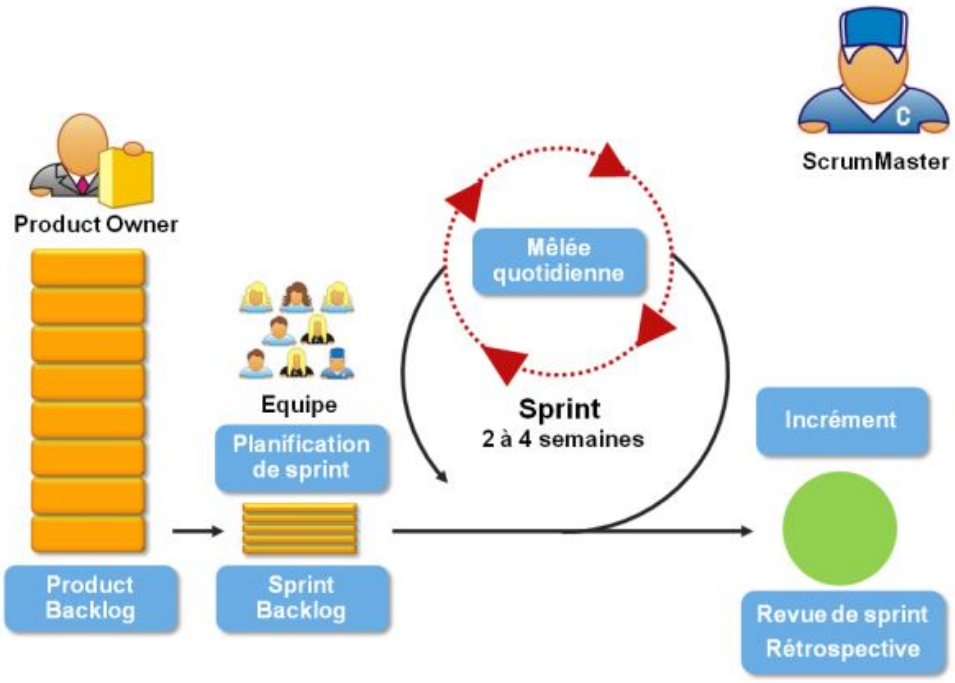

FIGURE 1.2 – Schéma de fonctionnement de la méthodologie Scrum

La méthodologie Scrum comprend des rôles bien définis. Nous rappelons ici, les principaux acteurs mis en jeux. En effet, l'équipe associée à un projet, appelé équipe Scrum, différencie plusieurs rôles distincts tels qu'un scrum Master, un Product Owner, Business Owner et une équipe de développement.

- Le Scrum Master s'assure que la méthodologie Scrum est respectée au sein du projet. Il se rend disponible pour aider les membres de l'équipe à sa mise en oeuvre.
- Le Business Owner (BO) a pour rôle de faire le lien entre le client et l'équipe. Il est le principal interlocuteur des clients. Il est chargé de retranscrire le besoin du client au reste de l'équipe.
- Un Product Owner (PO) crée des user stories techniques à partir des user stories fonctionnelles définies par le BO qui permettent de construire le "Results Backlog". Il organise le travail des développeurs afin d'être en mesure de respecter les dates des livrables fixées en amont. De plus, son travail consiste à estimer de l'effort de chaque tâche via

des story points selon la suite de Fibonacci. L'effort représente l'importance de la tâche à réaliser pendant le sprint.

Le modèle de données de la figure 1.3 montre comment est divisé un projet au sein de l'entreprise Thalès en impliquant les différents acteurs de ce découpage.

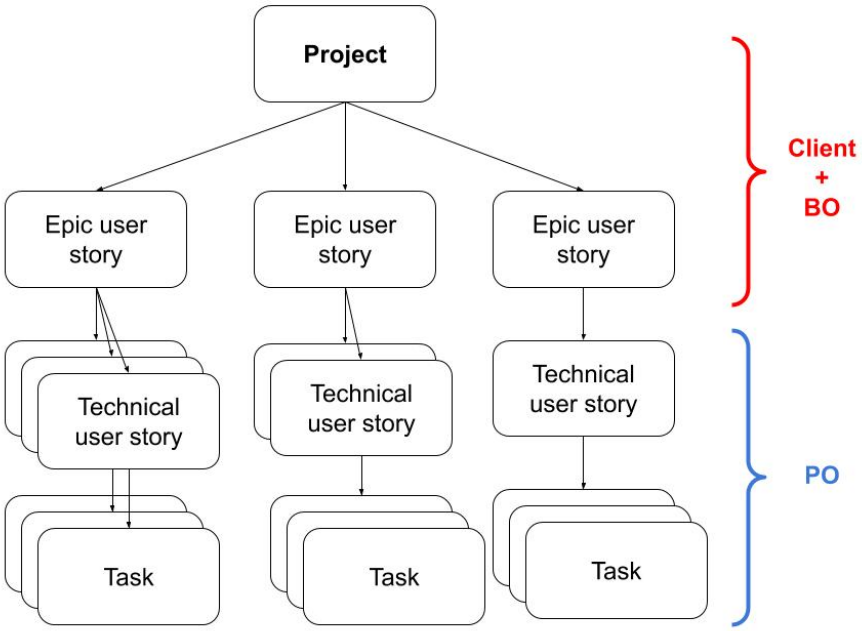

FIGURE 1.3 – Schéma de modèle de données utilisé pour décrire un projet

1.3.2 Description de l'environnement de travail

Nous allons voir plus spécifiquement quels moyens ont été développés par La Ruche -Thalès pour l'implémentation de cette méthode dans leur environnement de travail.

Comme expliqué dans la figure 1.4 chaque membre de l'équipe de développement dispose d'un PC portable fonctionnant sous windows 7 lui permettant d'accéder au réseau de l'entreprise. Le BO utilise un fichier excel pour synthétiser sa conversation avec le client et décliner les besoins du clients en user stories fonctionnelles, appelés épics. Via le répertoire partagé entre le BO situé dans le réseau de l'entreprise, les BO peuvent communiquer et échanger des données. Le transfert des données entre PO et BO se fait via clé usb ou par voie orale.

Le PO, quant à lui, doit dans un premier temps retranscrire le contenu d'un fichier excel contenant les user stories fonctionnelles vers Taiga manuellement. Il décline ensuite ces épics en user stories techniques sur lesquels vont se baser les développeurs pour orienter leur travail

(Windows 7). Le PO dispose aussi d'un PC fixe sous Window 10 pour développer. Pour que ses deux PC communiquer entre eux, une clée usb est nécessaire. Il accède au répertoire partagé entre POs via ce PC fixe.

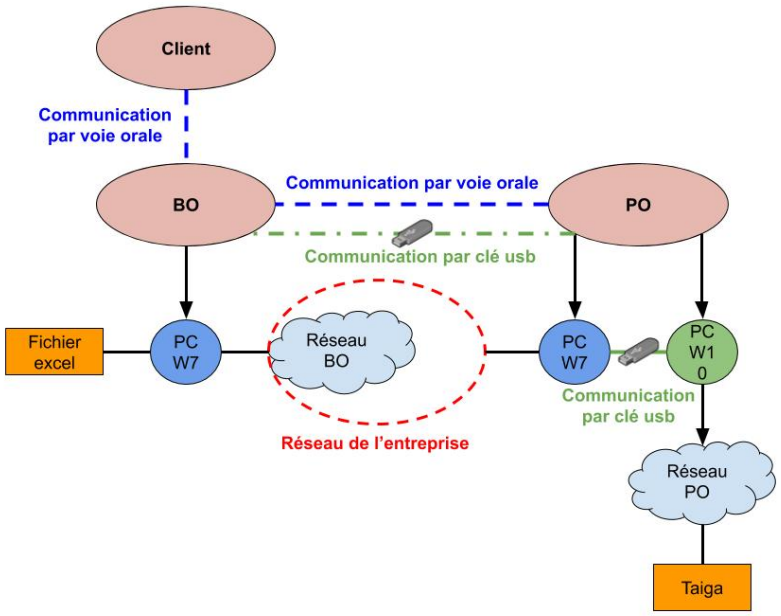

FIGURE 1.4 – Schéma du processus actuel de communication entre BO et PO

1.3.3 Présentation de la problématique

Le secteur de la défense exige un niveau de sécurité très élevé pour garantir la protection des données sensibles. L'entreprise doit donc se plier à des règles de cybersécurité très spécifiques. Les BOs, régulièrement amenés à collaborer avec des clients extérieurs, n'ont pas l'autorisation d'utiliser des logiciels exigeants une connexion internet pour relayer des informations propres à un projet. Ainsi, tout logiciel de gestion de projet tel que Taiga ou Jira est donc prescrit lors de ces travaux à l'extérieur. Une connexion à distance sur une machine n'est pas non plus envisageable dû à l'impossibilité d'ouvrir un port même local sur leur machine. La difficulté du projet est de faire communiquer les réseaux distants distants des POs et des BOs tout en prenant en compte les contraintes liées à la sécurité, imposées par l'entreprise.

Ainsi, l'objectif principal du projet est de faciliter la communication interne de l'entreprise et plus spécifiquement celle entre les POs et les BOs. Il s'agit de développer une solution basée sur la mise en commun des user stories de manière synchronisée tout en respectant les consignes de cybersécurité de l'entreprise. L'idée est d'avoir un outil qui permettant de :

— Rendre rapide et efficace la modification des user stories propres au BO et propres au PO tout en gérant les problèmes de conflits
— Permettre aux POs une visibilité permanente sur l'état des user stories des BOs et

inversement
— Instancier le modèle de données décomposant un projet en plusieurs entités (user stories, tâches, etc) comme vu précédemment dans la figure 1.4
— Mettre en lien les user stories des POs avec celles des BOs tout en gérant leur synchronisation de manière fiable.

2 Méthodologie de développement

2.1 Utilisation des méthodes agiles

La Ruche - Thalès utilisant la méthodologie Agile pour le développement de leurs projets, il nous a été demandé d'aborder la résolution de cette problématique en adoptant le même fonctionnement. Cela à donc été pour nous une opportunité d'appliquer cette méthode de travail, qui a été vue par les élèves de EII, dans un contexte professionnel. Nous avions donc en support Xavier Hochard, responsable de l'équipe DevOps de La Ruche, pour nous guider dans les différentes étapes et réalisations liées à cette méthodologie.

Nous avions la chance que notre entreprise en charge de ce projet se situe à Rennes. Nous avons décidé de nous déplacer pour les dates des sprint review suivantes : 4 novembre, 16 décembre, 12 janvier et 3 février. Ces sprints d'une durée de plus ou moins 3 semaines nous ont permis de faire des points régulièrement sur l'avancée de notre travail avec l'entreprise et d'orienter le projet dans la direction souhaitée. A chaque sprint review, une présentation avait lieu devant le personnel de Thalès impliqué dans notre projet (environ une vingtaine de personnes) au cours de laquelle nous rappelions les objectifs du sprint, nous évoquions les problèmes rencontrés et les solutions apportées et proposions une démonstration de notre travail. Les retours de l'équipe sur notre travail lors de ces réunions ont été très bénéfiques pour notre projet. En effet, cela permettait de soulever de nouveaux problèmes auxquels nous n'aurions peut être pas pensé, de répondre à nos questions et de nous débloquer sur des points appartenant au domaine de compétence de l'équipe. Par ailleurs, l'élaboration d'objectifs par sprint nous a permis de répartir efficacement le travail sur le temps imparti.

Nous avons utilisé l'outil Trello pour suivre le backlog du projet, le backlog du sprint en cours, et l'état d'avancement de chaque tâche (bloquée, en cours, à vérifier, finie) avec l'effort qui lui a été affecté et la personne en charge. L'attribution des efforts a été calculé à partir du travail que nous étions capables de fournir sur la durée d'un sprint, de nos disponibilités sur le projet et de la priorité de la fonctionnalité à développer. Grâce à cet outil, nous avons pu répartir les tâches du projet entre chaque membre de l'équipe. De plus, avoir un suivi régulier du gestionnaire de tâches nous a permis de ne pas exécuter une même tâche en double et d'avoir une connaissance des réalisations de chacun. Nous avons pu avoir une vue d'ensemble sur l'avancement du projet comme nous le montre la figure 2.1.

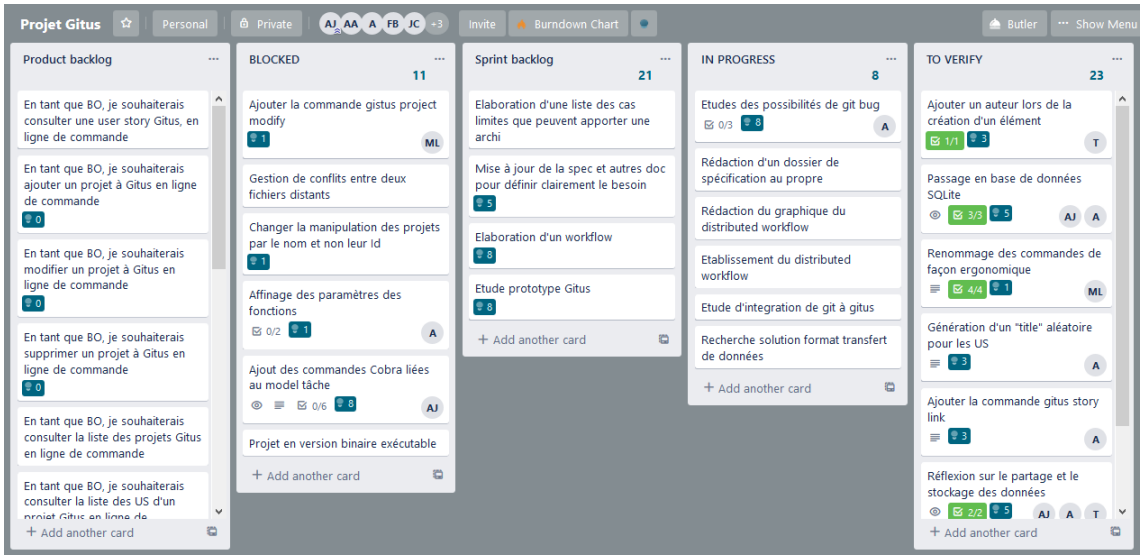

FIGURE 2.1 – Capture d'écran du Trello utilisé lors du projet Gitus

2.2 Les outils utilisés

2.2.1 Langage de programmation GO

Le langage de programmation open source Go est utilisé par les développeurs de la Ruche -Thalès. Il a été créé par les développeurs de Google, ayant pour objectif d'être optimisé et simplifié. C'est un langage compilé considéré comme simple à prendre en main. Il permet un processus de développement court en autorisant la création de gros fichiers exécutables. La syntaxe du langage est basée sur celle du C.
Pour apprendre à manipuler ce langage, nouveau pour chacun d'entre nous, nous avons réalisé le Tour de Go, un outil qui décrit les différents aspects de Go en se basant sur des exemples de programmes simples qu'il est possible de modifier et compiler. Le Tour de Go est idéal pour débuter et acquérir les bases du langage. Nous avons choisi Visual Studio Code comme éditeur de code et fait les réglages nécessaires pour programmer en Go.
L'utilisation de ce langage de programmation nous a été demandé pour être en cohérence avec les projets réalisés entièrement en Go au sein de Thalès. L'équipe de développeurs a ainsi pu nous orienter vers les outils et librairies à utiliser lors de la réalisation du logiciel. Notamment le choix de la librairie pour la gestion de l'affichage en ligne de commandes grâce à Cobra. La fonctionnalité de créer des exécutables facilement est très utile pour le projet car cela permet de simplifier l'utilisation du logiciel pour les BOs qui ont de nombreuses contraintes concernant l'installation de logiciels sur leurs machines.

2.2.2 Utilisation de Git (GitHub)

La gestion de version du logiciel a été réalisée grâce à Git. Le projet est stocké sur GitHub dans le but d'avoir un code open source. Il est alors visualisable par la communauté et pour des

potentiels futurs développeurs au sein de Thalès souhaitant améliorer notre solution. Nous avons utilisé l'outil Git Extension, un gestionnaire de dépôts qui dispose d'une interface graphique simple d'utilisation pour faciliter le travail du développeur lors de la synchronisation de son dépôt local au dépôt distant et inversement. Un Readme détaillant les étapes à suivre a été élaboré pour que chaque membre du groupe puisse avoir un environnement de travail fonctionnel et identique le plus rapidement possible.

2.2.3 Intégration continue avec Git Action

L'intégration continue est un enjeu majeur dans le développement informatique, permettant l'exécution des tests unitaires et la compilation du projet de façon automatique. Lié à un dépôt distant, il permet à chaque ajout de code d'un développeur de réaliser les étapes précédemment explicitées permettant de lever une alerte rapidement si des erreurs sont repérées afin de limiter la propagation de ces dernières et conserver un code stable. L'outil Jenkins est la référence en terme de plateforme d'intégration continue, cependant nous nous sommes orientés sur la solution GitHub Action car elle est intégrée à GitHub ce qui limitait notre travail de mise en place.

2.2.4 Merge request

Lors du développement de ce projet nous avons également utilisé le système de Merge Request proposé sur la plateforme GitHub. Une Merge Request correspond à une demande de fusion du code développé par un des développeurs de l'équipe dans le code global du projet suite à la finalisation d'une fonctionnalité. La demande doit être validée par au moins un autre membre de l'équipe qui se charge de relire le code, vérifier le respect des standards et écrire des commentaires pour d'éventuelles questions sur les techniques et technologies utilisées.
Ce système permet de prendre connaissance de ce qui est réalisé par les autres membres de l'équipe afin d'avoir une vision globale sur l'avancement du projet et également de partager ses connaissances avec les membres de l'équipe. Cela nous a permis de monter assez rapidement en compétence sur le langage Go étant donné que nous pouvions apprendre des autres en relisant leur production.

2.3 Stratégie de développement

Notre projet s'est découpé en quatre sprints ne prenant pas en compte le sprint de prise en main en début de projet. Le sprint 0 nous a permis de nous former sur les nouveaux outils dans le but d'être capable de mettre en place notre environnement de travail. Une majorité du travail au cours de ce sprint a consisté en de la documentation et de l'initiation aux logiciels fondamentaux pour la réalisation du projet par le biais d'exercices pratiques.

Au cours du sprint 1, nous avons spécifié les objets à manipuler, les fonctions à réaliser et les limites du projet. Nous sommes ensuite passés à la partie développement de cette première ébauche. C'était le meilleur moyen de mettre en application les connaissances acquises lors du sprint précédent tout en ayant une première approche plus concrète du sujet à traiter. Suite à ce premier sprint, nous avons évalué de façon plus juste la quantité de travail que nous étions en mesure de fournir au cours de cette période. Par conséquent, nous avons opté pour une

gestion de projet adaptée à nos disponibilités et à celles de l'entreprise.

Durant le sprint 2, nous avons poursuivi et amélioré notre travail entamé au sprint 1. En rentrant plus en profondeur dans le sujet, nous avons compris les réels enjeux de l'entreprise derrière ce projet. Nous avons rencontré quelques soucis à ce moment là dû à un manque de spécifications du projet du côté de l'entreprise. Nous ne disposions pas de toutes les informations nécessaires pour avancer. Nous avons donc été bloqués dans notre productivité. Pour pallier à ce manque d'informations, nous avons programmé une réunion avec l'entreprise dans le but d'éclaircir ensemble les points du projet qui nous paraissaient flous. Une spécification de l'environnement de l'entreprise et une formulation du problème écrite a été établie par l'entreprise afin d'éviter des représentations divergentes de la problématique comme cela a pu être le cas en début de projet.

Grâce à cette clarification du cahier des charges, nous avons pu approfondir notre travail au cours du sprint 3. Dans un premier temps, nous avons établi un schéma de l'environnement de travail mettant en valeur les contraintes matérielles de l'entreprise. A partir de cette base, nous avons donc eu une étape de réflexion afin d'élaborer des workflows distribués susceptibles de répondre aux attentes de l'entreprise. Pour se faire, nous nous sommes renseignés sur l'exploitation de solutions déjà existantes présentés par Git. Chaque solution a été sujette à une étude de faisabilité et d'intégration avec le schéma de contraintes matérielles nous permettant d'aboutir à un workflow distribué final. Nous avons étudié la faisabilité de chaque solution.

Le sprint 4 était une étape de rendu de nos résultats afin qu'ils soient exploitables par l'entreprise. Nous avons fixé avec l'entreprise le prototype à mettre en oeuvre, la solution finale à adopter et la mise en forme de la trace écrite : une étape indispensable pour fluidifier au maximum la transmission de savoir.

3 Le travail réalisé

3.1 Réalisation du premier prototype Gitus

L'objectif de réaliser un prototype est de pouvoir modéliser un environnement de travail pour se rapprocher au mieux de la réalité. Cet exercice est fondamental car il a pour but de mettre en lumière des problèmes afin de les corriger avant le développement d'un produit.

Notre solution étant un logiciel informatique possédant une interface graphique, nous allions le développer le code en utilisant le patron de conception Modèle - Vue -Contrôleur. L'objectif de ce premier prototype était donc de mettre au point la partie Vue de notre application tout en utilisant une première ébauche de la partie Modèle. Cette étape de prototypage allait donc, dans notre cas, nous permettre de nous familiariser avec le Modèle et en comprendre les différents enjeux vis à vis des contraintes tout en implémentant une base de notre solution finale.

3.1.1 Elaboration du modèle de donnée

La figure 3.1 est un diagramme UML présentant le modèle de donnée que nous souhaitions implémenter dans notre prototype. La construction de ce modèle a été fait en se basant sur la façon dont les ingénieurs de Thales avait l'habitude de fonctionner d'un point de vue gestion de projet. Des attributs supplémentaires étaient utilisés au sein de La Ruche cependant il n'est pas pertinent dans cette première étape de les implémenter.

FIGURE 3.1 – Schéma du modèle de données instancié

A partir de ce diagramme nous avons pu créer les différents package en Go décrivant la représentation et les fonctions de chacun des éléments du modèle. Le langage Go n'étant pas un langage orienté objet cela a été fait à l'aide de structures comme il est possible de le faire en C.

3.1.2 Stockage du modèle

Pour stocker les données, nous nous sommes naturellement orientés vers une base de données SQL. En effet, c'est un format très répandu dans les entreprises. Certains d'entre nous avaient eu l'occasion de l'étudier au cours de leur formation et de l'utiliser lors de stages. Dans un premier temps nous nous sommes orientés vers une base MySQL qui permet de créer une base de données locale sur la machine à l'aide d'une connexion TCP/IP à un port local. Cependant les BO ne pouvant ouvrir de telles connexions nous avons du changer de support. Pour pallier à cette contrainte, nous avons fait le choix d'utiliser une base de donnée SQLite qui contrairement aux autres base SQL utilise un fichier .db qui simule le comportement d'une base. L'utilisation d'une telle solution de stockage nécessitait l'ajout d'une interface de requêtes en base de données pour récupérer les informations contenues dans le fichier en question, les modifier et les mettre à jour.

3.1.3 Développement de l'interface

Pour qu'un utilisateur puisse entrer, récupérer, modifier des informations stockées dans la base de données, nous avons mis en place une interface en ligne de commandes. Pour le développement de cette interface, nous avons suivi les conseils de l'entreprise et nous avons utilisé le package Cobra, dont le code source est disponible sur github. Ce package a pour avantage d'être haut niveau ce qui facilite l'ajout d'utilisateur, la création et l'implémentation de commandes. Ce logiciel est utilisé par des applications assez connues, écrites en Golang, telles que Dockers ou Kubernetes.

La figure 3.2 montre un exemple de commande nous avons implémenté : la commande gitus userstory create pour ajouter une nouvelle user story à un projet. Cette commande prend en paramètre le nom de la story. Il est possible d'ajouter en paramètre optionnel une description et un effort associé.

```
gitus userstory create [<name>]
                        --description Description
                        --effort (0,1,3,5,8,13)
```

FIGURE 3.2 – Ligne de commande pour la création d'une usertory

3.2 Mise au point du workflow distribué

Une fois que nous avions structuré les données à partager, l'objectif de la mission était de répondre à la mise en collaboration de ces données au sein de l'environnement de travail de

l'entreprise, défini précédemment. Nous rappelons que la difficulté est qu'il faut faire communiquer deux réseaux isolés n'ayant comme seule passerelle une clé usb. Nous ne pouvons donc pas créer un dépôt git unique qui centralise tout, comme dans la majorité des workflows mis en place actuellement.

En se basant sur la documentation Git à propos des workflows distribués, nous avons proposé deux workflows à Thales ayant chacun leur avantages et leur inconvénients. Le workflow numéro 1 est représenté sur la figure 3.3.

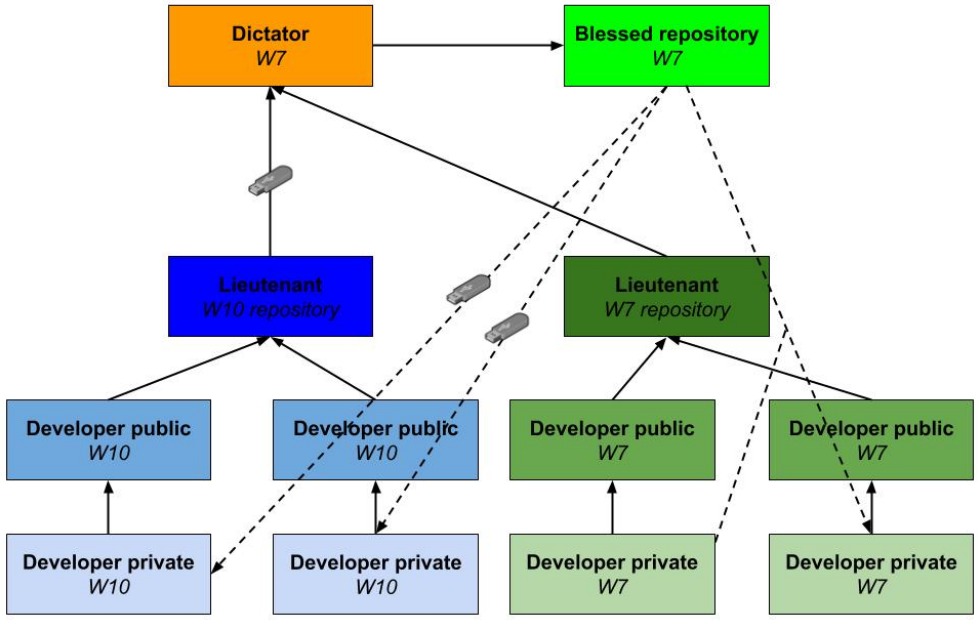

FIGURE 3.3 – Le workflow 1

Dans l'intérêt de simplifier le travail des BO, soit des personnes pas nécessairement à l'aise avec les différentes technologies, nous avons décidé de limiter leur actions en positionnant le dépôt principal, appelé blessed repository, sur le réseau des PC W7. En effet, étant donné qu'un BO dispose d'un PC W7, la mise à jour avec le blessed repository se fait en un nombre d'actions assez réduit avec de simple commandes de pull (pour copier à l'identique un répertoire). En ce qui concerne le PO, étant donné que son PC W10 n'est pas sur le réseau W7, il est obligé de se mettre à jour avec le blessed repository via une clef USB. Lorsqu'un PO ou un BO souhaite partager son travail (finalisation d'une fonctionnalité par exemple), il envoie à l'aide d'un push sa modification pour qu'elle soit intégrée dans un document situé sur son dépôt lieutenant respectif (W10 pour les PO et W7 pour les BO). Le dépôt lieutenant a pour rôle de synchroniser et de gérer les conflits du travail des PO et des BO respectivement. Les données du dépôt lieutenant sont ensuite envoyées sur le dépôt dictateur. Le dépôt lieutenant des PO,

étant situé sur le réseau W7, nécessite l'emploi d'une clé USB pour que les modifications soient apportées au dépôt dictateur. Le dépôt dictateur est un dépôt intermédiaire qui fait le lien entre le dépôt lieutenants des PO et celui des BO. Enfin, une fois toutes les fonctionnalités intégrées entre elles dans le dépôt dictateur, elles peuvent être ajoutées sur le blessed repository.

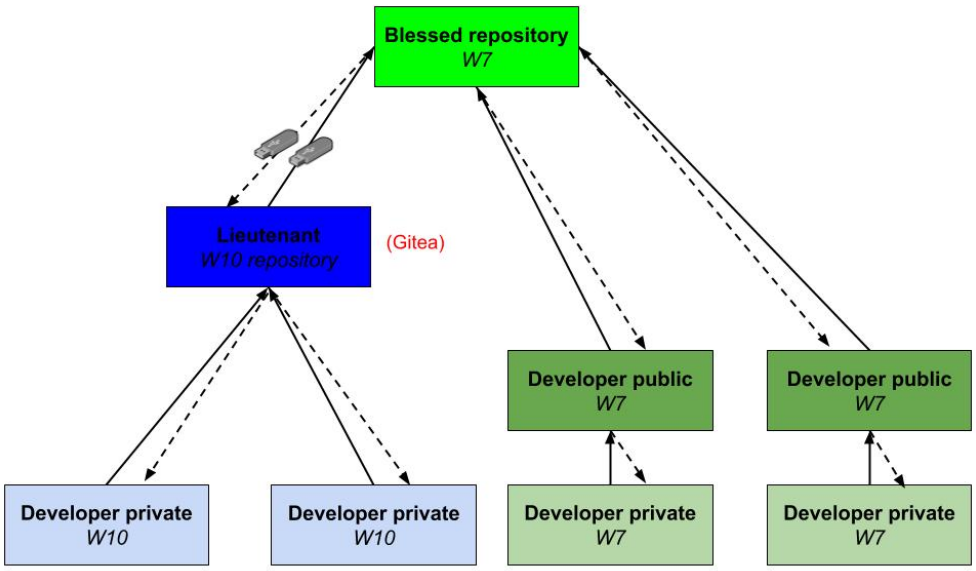

FIGURE 3.4 – Le workflow 2

Nous avons élaboré un worklfow 2, décrit sur la figure 3.4 pour être en mesure de comparer deux architectures différentes. Nous avons choisi dans cette deuxième solution d'alléger l'utilisation de clé usb. Le lien des BO avec le blessed repository reste identique. Les PO utilisent un dépôt distant hébergé à l'aide de Gitea sur le réseau W10, logiciel déjà utilisé actuellement à La Ruche-Thalès pour la synchronisation des données internes aux PO, pour se synchroniser au blessed repository.

L'inconvénient des deux solutions est l'emploi régulier de clés USB qui peut être contraignant pour l'utilisateur. Toutefois, nous n'avons pas trouvé de moyen de s'en séparer respectant les consignes de sécurité de l'entreprise. Dans l'architecture 1, il faut associer une distribution des droits par utilisateur à l'architecture. En effet, il faut limiter le nombre de personnes qui ont des droits administrateurs sur les dépôts dictateurs, lieutenants et blessed pour éviter de se retrouver avec des conflits ingérables. La solution 2 nécessite qu'une personne soit désignée pour mettre à jour quotidiennement le dépôt sur le réseau W10 avec le blessed repository. Cette action doit être faite avec rigueur pour ne pas créer de trop forts décalages entre les différentes versions du produits. L'avantage de cette solution est que chaque PO puisse se mettre à jour avec le

blessed repository à chaque instant sans dépendre d'une autre personne. Bien que l'architecture 1 soit plus complexe que l'architecture 2, elle garantit une meilleure protection des données tout en accordant une indépendance aux développeurs dans leur travail. C'est donc pour ces raisons que nous avons trouvé la solution 1 plus judicieuse.

3.3 Etude de deux solutions de stockage de données

Une fois le workflow distribué mis au point, l'objectif suivant était de déterminer la solution de stockage du modèle de données s'intégrant de la façon la plus optimale avec celui-ci. Cela intègre la procédure permettant de partager les données stockées dans les différents dépôts Git présentés précédemment ainsi que la gestion des conflits une fois le partage réalisé. Lors de cette étude nous nous sommes penché sur notre prototype Gitus et le stockage à l'aide d'une base de données SQLite ainsi que sur une solution déjà développée Git-Bug.

3.3.1 Etat de l'art prototype Gitus

Comme expliqué auparavant, la base de données a été instanciée à l'aide de SQLite qui utilise un fichier .db. La première idée était de placer une base sur chaque réseau, une sur celui de PO et une autre sur celui des BO. Cependant, cette approche nous confrontait à la synchronisation de deux bases ensemble du fait que les deux réseau soit séparées physiquement. De plus la synchronisation des deux réseaux complexifierait notre solution finale Gitus du fait que nous aurions du implémenter des fonctionnalités propres à cette tâche.

Une seconde idée a été pensée, consistant à partager ce fichier .db directement à travers les différents dépôts, seulement le fichier étant encodé nous savions pas comment procéder pour gérer les conflits. C'est pourquoi une autre solution plus portable a été abordée, consistant à exporter l'ensemble de la base de données dans un fichier JSON. Ce fichier allait ensuite pouvoir être partagé à travers les dépôts via Git permettant à l'utilisateur de gérer les conflits sur le fichier comme dans le cas du partage de code. Cependant cette solution n'est pas complètement intégrée dans notre application du fait quelle nécessite l'utilisation de façon externe de Git qui peut être une gêne à long terme. Enfin l'export au format JSON, l'import des modifications et la gestion des conflits à l'aide d'un outil à l'origine dédié pour du code ont de fortes chances de conduire à des erreurs dans les données

3.3.2 Etat de l'art de Git-Bug

Lors de la mise en place du projet Gitus en interne, Thalès, a procédé à des recherches afin de déterminer s'il existait des solutions déjà développées et Open Source pouvant répondre à leur problématique. Lors de cette étape, ils n'ont pas trouvé de solution clée en main mais une application permettant de travailler en collaboration et échanger des informations sur des "bugs". Cette solution, Git-Bug, réalisée par Michael Mure est Open Source et disponible sur la plateforme GitHub. Selon Thalès, l'application pouvait nous servir de base d'information sur les technologies à utiliser ou même une base pour notre solution finale. Dans cette étape d'état de l'art, nous avons donc procédé à une étude de l'application Git-Bug afin de découvrir son

fonctionnement, son architecture ainsi que les technologies utilisées afin de déterminer si nous pouvions nous en inspirer dans le but de répondre à notre problématique.

Git-Bug est un bug tracker, un système de suivi de bugs utilisé dans le cadre de projets de développement logiciel, distribué et intégré dans Git, l'outil de gestion de version. Les bugs signalés sont stockés sous forme d'objets Git évitant ainsi de polluer le workspace et les branches du dépôt Git de l'utilisateur. L'application a été développée avec le langage Go et est une interface en ligne de commande, mise en place à l'aide du package Cobra que nous avions également utilisé pour notre prototype.

Stockage des données dans Git-Bug

Git-Bug à la particularité d'utiliser les objets Git afin de stocker ses données, les bugs, et non pas une base de données SQL comme nous avons fait pour notre premier prototype. Cette méthodologie de stockage est appelée une base de données NoSQL pour Not only SQL. Cette forme est une alternative aux bases de données traditionnelles où les données sont placées dans des tables et le schéma des données est soigneusement établi avant que la base de données soit créée. Les bases NoSQL sont bien adaptées pour des gros volumes de données distribués qui peuvent être amenés à changer dans le temps. Il existe différentes formes de bases de données NoSQL, graph stores, key-value stores, document et wide-column stores. Git étant un système de fichier adressable par contenu, la forme implémentée est key-value, qui est la forme la plus simple, associant à un objet (value) un nom (key), par la suite l'objet est accessible en indiquant son nom.

Fonctionnement de git comme base de données NoSQL

La mise en place d'une base de données NoSQL nécessite plusieurs étapes utilisant des objets internes à Git dont nous avons peu l'habitude de manipuler tel que les Git Blops, Git trees, Git Commits et Git References.

Le première étape consiste à stocker notre objet dans un Git Blop, qui est une zone de stockage d'une valeur. Pour se faire il faut décrire l'objet à l'aide du langage JSON, JavaScript Object Notation, puis utiliser la commande hash-object, qui prend le fichier et le stocke dans le répertoire .git, puis retourne la clé sous laquelle le fichier est stockée. Cette clé est une empreinte SHA-1 contenant 40 caractères. Une fois notre Blop créé, nous pouvons par la suite accéder à son contenu via la clé, notre base de données key-value est donc établie. Cependant si nous modifions le contenu de notre objet la clé va changer, ce qui implique que nous devons connaître pour chaque version la valeur de la clé. Pour pallier à ce problème il faut donc combiner à ce Blop les Git trees, Git Commits et Git References.

Le Git tree va nous permettre de stocker plusieurs Blop ainsi que de s'affranchir du besoin de connaître le SHA-1 de chaque Blop. En effet, les Git trees, peuvent être vues comme des dossiers, et possèdent également un SHA-1. Ainsi avec Git tree nous pouvons référencer plusieurs fichiers. On utilise après un Git Commit qui permet de pointer vers un unique Git tree et qui peut être lié à un Git Commit plus ancien. Cela nous permet donc de créer un historique de notre objet en liant les commits des versions précédentes au commit de la nouvelle version.

Cependant tout comme les Git Blop, les Git Commit ont un SHA-1 qui leur sert de clé et qui est changé à chaque nouveau commit. La problématique liée au fait de devoir se souvenir d'un nouveau SHA-1 à chaque modification n'est donc toujours pas résolue.

Pour répondre à cette problématique il faut donc utiliser le dernier objet mentionné, la Git Reference. En effet, cet objet est un fichier contenant des empreintes SHA-1, possédant un nom simple choisi par l'utilisateur, dans notre cas l'identifiant de l'objet, qui ne change pas au cours du temps. De plus, la Git Reference permet de pointer vers un SHA-1 parmi l'ensemble qui la compose et ce dernier peut être mis à jour. Ainsi à chaque nouveau commit de modification, on ajoute son SHA-1 dans la Git Reference puis on change le SHA-1 pointé sur ce dernier, qui pointera donc vers la version de notre objet la plus récente. Les Git References sont simplement des branches dans Git pointant le dernier état d'une suite de travaux.

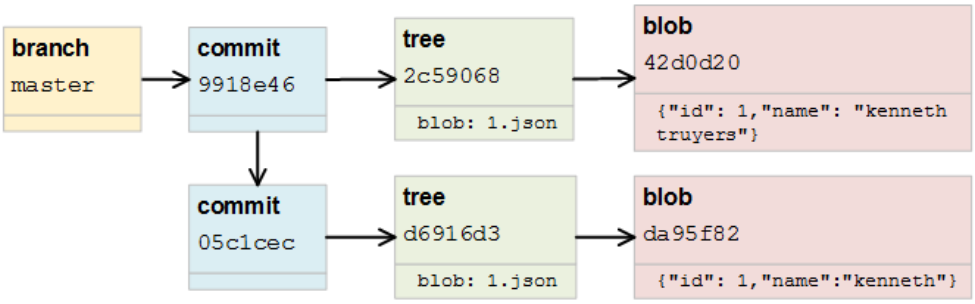

FIGURE 3.5 – Schéma décrivant les relations entre objets Git

Le figure 3.5 résume les explications précédentes avec le stockage d'un fichier JSON ayant subit une modification. L'ensemble du processus permettant de créer cette base de données est détaillé dans la documentation officielle de Git (cf Annexes).

Bilan base de données NosSQL avec Git utilisée par Git-Bug

Cette base de données avec Git présente donc de nombreux avantages permettant de résoudre notre problématique du stockage et partage de nos données. En effet, cela nous permet de gérer de façon implicite le versionnage de nos objets grâce à l'historique des commits tout en étant partageable et adaptable à notre workflow distribué du fait que cela soit stocké dans un dépôt git. L'aspect unicité de nos objets est également assuré étant donné que nous pouvons utiliser le SHA-1 du premier Blop créé comme identifiant de notre objet. Cependant, la gestion des conflits sur les modifications des données inhérent au fait qu'elles soient distribuées n'est pas assuré.

Pour résoudre le problème des conflits, Git-Bug stocke des opérations éditions simples effectuées sur le bug plutôt que l'état brut de l'objet. Ainsi, quand une fusion doit avoir lieu entre deux versions, lors d'un git pull d'un dépôt distant vers un dépôt local par exemple, l'une des deux versions voit ses nouvelles opérations placées à la fin de la chaîne, à la manière d'un rebase. On a donc la garantie d'avoir un état final cohérent puisque c'est Git-Bug qui dérive l'état final de

la chaîne d'opérations. Dans le cas où deux personnes suppriment la même étiquette, elle sera simplement supprimée une fois et l'action des deux personnes sera visible dans l'historique.

3.3.3 Bilan état de l'art des deux solutions

A la suite de ces deux études de cas, nous avons pu en conclure que l'utilisation de Git comme base de données NoSQL était la meilleure solution à notre problème. En effet, comme nous l'avons expliqué, cette solution de stockage permet de gérer additionnellement et de façon implicite la gestion de conflits, la distribution des données et le versionnage des données. Alors que notre première solution avec Gitus nécessitait, pour le même fonctionnement, deux outils externes et n'assurait pas une gestion des conflits totale et sans erreurs.
Lors de la sprint review, notre choix de solution a été validé avec La Ruche Thalès. L'objectif du sprint suivant (le dernier du projet) était donc de créer un prototype utilisant ce système de stockage en gérant seulement les User Story. Le choix de ne pas intégrer le modèle Projet et Tâche a été fait car nous savions que la durée du sprint ne nous permettrait pas de réaliser l'ensemble des fonctionnalités pour les trois types de données.

3.4 Prototype final

Le prototype final se présente de la même forme que notre première réalisation, une interface en ligne de commande développée avec le package Cobra. Nous nous sommes également basés sur le logiciel Git-Bug pour réaliser cette solution finale étant donné qu'il était codé à l'aide du langage Go. En effet, nous avons importé des packages de ce logiciel dans notre code afin de ne pas avoir à développer du code déjà existant et donc gagner en efficacité. De ce fait, nous avons adapté une architecture interne du code semblable à Git-Bug. Étant peu expérimentés dans le développement informatique, notamment en Go, nous avons préféré s'appuyer sur l'architecture d'une solution déjà testée et éprouvée plutôt que d'implémenter la nôtre dans le but de fournir à La Ruche une base de code solide respectant les normes et les standards du langage. Cependant, afin d'adapter cette architecture, un travail conséquent de compréhension du code était nécessaire.

3.4.1 Architecture de la solution

Comme mentionné précédemment, nous avons utilisé des packages de Git-Bug dans notre solution finale. La figure 3.6 représente l'architecture interne, en terme de package, du plus bas niveau au plus haut des solutions Git-Bug et Gitus. Chaque couche du schéma utilise la couche sous-jacente afin d'accéder aux données et d'interagir avec elles. Dans l'architecture de Gitus les packages en gris sont les packages importés de Git-Bug, en orange ceux qui ont été forks de Git-Bug et en vert ceux implémenté par notre groupe.

3.4.2 Packages importés

Nous avons fait le choix d'importer directement les packages Repository et Identity car leur contenu était utile dans notre solution et ne nécessitait pas d'être modifié pour s'y adapter.
En effet, le package Repository implémente les différentes interactions avec le dépôt Git présent sur le disque, afin de l'utiliser comme base de données NoSQL. Un ensemble d'interfaces est

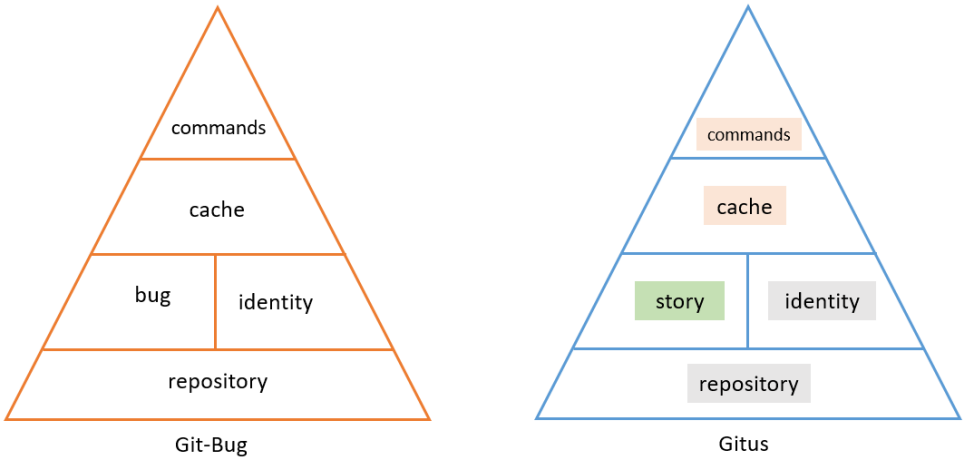

FIGURE 3.6 – Architecture interne des solutions Git-Bug et Gitus

disponible définissant un accès pratique aux méthodes d'accès et de manipulation des données stockées dans git. Parmis ces méthodes, on retrouve celles permettant la création d'objets internes à Git (Blop / Tree / References / Commit) ainsi que les fonctions liées au partage du dépôt (push / merge / fetch).
Le package Identity quant à lui, contient le modèle de données identité des utilisateurs ainsi que les fonctions de bas niveaux associées. Il permet de caractériser les utilisateurs via leurs login git de leur PC et est complètement intégré dans la base de données. Pour notre solution, nous souhaitions avoir la possibilité de gérer et d'identifier les différents utilisateurs et leurs actions sur les données. Cette solution, correspondant complètement à nos attentes et s'interfaçant parfaitement avec le package Repository, nous avons fait le choix de l'importer également.

3.4.3 Package Story

Le package Story a été complètement implémenté par notre groupe. il contient le modèle de données des story et les fonctions bas niveaux associées.
Nous l'avons réalisé à partir du modèle déjà détaillé dans notre premier prototype et en y ajoutant des attributs supplémentaires comme un statut. Ce paramètre a été ajouté car il n'est pas possible de supprimer une donnée de notre base. De ce fait, pour différencier une story implémentable d'une qui ne l'est plus, il est possible de lui attribuer un statut Open ou Close. Ces statuts ont été préférés aux statuts Todo, InProgress, Done, généralement utilisés, car nous ne savions pas si c'était le cas pour La Ruche. Nous avons donc préféré être succincts tout en implémentant cet attribut de façon à ce qu'ils puissent détailler de nouveaux statuts facilement a posteriori.

Opérations

Comme expliqué lors de l'étude de Git-Bug, ce dernier ne stocke pas les données de façons brut mais à l'aide d'un système d'opérations de modification qui une fois dérivées de l'état initial

nous donne l'état courant de la donnée. Nous avons donc repris ce principe afin de gérer les conflits sur les données de façon implicite aux yeux de l'utilisateur. Une opération de modification correspond à un fichier JSON indiquant pour chacune son auteur, la date, la valeur précédente de l'attribut modifié et sa nouvelle valeur. Chaque opération est définie par un type indiquant quel attribut est modifié. Ce fichier est ensuite transformé en Git Blop et ajouté à la base de données en suivant la logique expliquée en partie 3.3.2.A chaque nouvelle modification un Git Commit est réalisé qui remplace donc le contenu du fichier opération de modification du Git Tree comme on peut le remarquer sur la figure 3.5.

Ainsi, à l'ajout d'une nouvelle story dans la base de données, une opération de création est effectuée. Cette opération diffère des autres du fait qu'elle indique pour l'ensemble des attributs de la story leur état initial et est stockée de façon permanente dans la base. En effet, dans le Git Tree d'une story ce fichier n'est jamais amené à changer lors des différents commit contrairement au fichier d'opération de modification, du fait que l'opération de création pour une même story n'est réalisable qu'une seule fois. Puis un ensemble d'opérations de modifications lui sont appliquées lorsque l'utilisateur souhaite changer un attribut, changer son statut à Close par exemple. Lorsque l'utilisateur souhaite afficher une story afin de visualiser son contenu, l'état actuelle de la story est réalisé en récupérant le contenu du fichier d'opération de modification dans l'ensemble des Git Commit de la Git Reference associée à la story et en appliquant toutes ces opérations à l'état initial de la story.

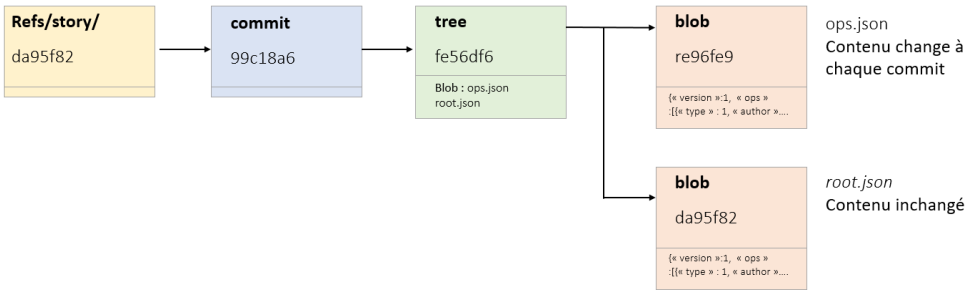

FIGURE 3.7 – Schéma de stockage des story dans le dépôt Git

3.4.4 Package cache

Le package cache implémente un niveau de caching au dessus des package de bas niveau story et identity pour fournir des fonctionnalités de requête, filtrage et tri efficaces. L'objectif du cache est de permettre un accès aux données rapide et efficace en évitant d'aller chercher dans le dépôt Git les informations à chaque actions.
Pour se faire la cache maintient en mémoire et sur le disque les infos primaires d'une story permettant des requêtes rapides à l'ensemble des stories sans avoir à les recharger à chaque fois. En pratique, la cache des stories est constitué de 2 tableaux, le premier, StoryCache, contenant l'ensemble des stories sous un format compressé snapshot mettant à disposition une API simplifiée. Le second, StoryExcerpt, qui contient un petit sous ensemble de données

pour chaque story permettant un filtrage, indexage et triage rapide. Les mêmes tableaux sont disponibles pour les identités. Ce système permet également d'assurer le maintien d'une copie unique de chaque données sur le disque.
Le package cache était existant dans la solution Git-Bug cependant étant donné qu'il devait interagir avec les modèle de données story nous devions faire des modifications sur son contenu. Pour se faire nous avons procédé à un fork de ce package, c'est à dire que nous avons récupéré la totalité du code et créé les fonctions nécessaires pour communiquer avec le package story.

3.4.5 Package commands

Enfin le dernier package implémenté est celui correspondant au plus haut niveau, le package commands. Celui-ci met à disposition de l'utilisateur un ensemble de commande lui permettant d'interagir avec les données. En terme informatique, ce package joue le rôle d'interface entre l'homme et la machine et correspond à la vue dans le patron de conception Modèle - Vue - Contrôleur.
La mise au points de ce système de commande a été fait comme pour le premier prototype avec le package Cobra. Les différentes commandes permettent de modifier les stories, comme leur statut à l'aide la commande "gitus status edit pour" laquelle on spécifie l'Id de la story à modifier et en indiquant à l'aide de flag la nouvelle valeur souhaitée. Les commandes "gitus push" et "gitus pull" permettent quant à elles la distribution des données dans le workflow mettant à disposition les modifications effectuées.

3.5 Tests unitaires et intégration de la solution

Soucieux de la qualité du code que nous allions fournir à Thalès, nous avons développé une batterie de tests unitaires et d'intégrations sur les packages story et cache. L'objectif était de tester les bons fonctionnement du système de modification par les opérations ainsi que les fonctionnalités de distribution de la base de données.
Ces tests permettent également lors de la poursuite du projet par Thalès de garder une stabilité de la solution vis à vis des modifications qu'ils pourront apporter. Cependant, le score de couverture du code par les tests est seulement de 60% des lignes produites. Ce score est correct mais pas optimal, en effet, la majeure partie des tests manquants concerne les cas d'erreurs lors des opérations, c'est à dire les tests cherchant à vérifier que la fonction est bien censée échouer dans ce cas.

4 Conclusion et perspectives

4.1 Bilan de notre mission

Pour conclure, nous avons connu des débuts difficiles lors de la prise en main du projet. Il a été nécessaire de définir plus clairement le projet et de réadapter nos compétences techniques à la durée imposée. Bien que la prise en main du projet ait pris un certain temps, nous avons su atteindre les objectifs fixés par l'entreprise.
Nous avons proposé un workflow distribué théorique qui répond aux problèmes de communication existants entre les différents acteurs de l'entreprise dans leur travail. Cette réalisation permet la synchronisation et traçabilité des user stories au sein de tous les membres du projet. Nous avons défini un modèle de données qui s'applique à leur problème. A partir de cela, nous avons créé un prototype de Gitus qui permet la manipulation des user stories. Le partage de ces données via notre prototype a été implémenté et est fonctionnel, mais quelques modifications restent à apporter afin de correspondre complètement aux attentes de l'entreprise. De plus, nous n'avons pas pu tester notre prototype dans l'environnement de l'entreprise en utilisant le workflow établi dû à un manque de temps. Des tests ont été réalisés sur une échelle plus petite dû à une difficulté de reproduire l'architecture de l'entreprise à l'identique. Nous avons fourni un rapport détaillé de l'avancée de notre travail, du point de vue technique, afin que le sujet puisse être repris et poursuivi au sein de l'entreprise.

En terme de travail qu'il reste à faire, la validation du workflow théorique proposé consiste en une première tâche. D'autre part, il sera nécessaire d'intégrer le modèle dans sa globalité au prototype dans la continuité du travail réalisé sur le modèle user story. Des pistes ont été proposées pour la création des liens existants entre les différentes types de données, toutefois celles-ci n'ont pu être mises en oeuvre. L'entreprise partenaire est globalement satisfaite du travail que nous avons pu fournir et des résultats obtenus.

4.2 Apprentissage

D'un point de vue technique, ce projet a été très enrichissant. Nous avons pu manipuler les outils fondamentaux liés au développement en collaboration comme Git, une platefrome de dépôts distants, l'intégration continue ainsi que les bonnes pratiques associées. Nous avons également découvert le fonctionnement des base de données SQL : nous maîtrisons maintenant l'exploitation de plusieurs systèmes de gestion de base de données tels que MySQL et SQLite. Cette compétence est particulièrement intéressante si nous souhaitons nous spécialiser dans le développement informatique. Nous avons élargi notre panel de langages de programmation

avec l'apprentissage du langage de programmation Go, nouveau pour chacun d'entre nous. Nous avons vu que notre bagage informatique acquis lors de notre formation nous a été fortement utile pour se former rapidement sur un langage inconnu. L'une des plus grosses difficultés de cette mission a été d'être capable de comprendre et d'exploiter des codes complexes déjà existants dans le but de les adapter à notre solution. Enfin, L'étude et l'implémentation d'une base de donnée NoSQL à l'aide de Git nous a permis de découvrir un nouveau type de base de données ainsi que de se plonger dans les rouages de Git, comprendre son fonctionnement interne et donc mieux saisir les différentes fonctionnalités misent à disposition pour le versionnage de code. Nous nous ne sommes pas des étudiants issus d'une filière informatique à la base, néanmoins, nous avons su combler le manque de connaissances que nous avions. Ce projet a fait ressortir notre polyvalence.

Grâce à ce projet nous avons compris les enjeux de la communication en entreprise. Le contact régulier avec l'entreprise sur lequel la méthode agile met l'accent a été indispensable dans ce projet. Cela nous a permis de recentrer notre travail dans la bonne voie. Toutefois, nous y avons vu aussi la difficulté de l'application de la méthode agile lorsque certaines étapes ne sont pas respectées. En effet, la définition claire du projet, de ses limites et de ses objectifs finaux sont des éléments clés. Il est essentiel pour le développeur d'avoir les clés en main pour bien cerner les attentes du client, sans quoi l'avancement peut se faire de manière efficace.

Par ailleurs, cette expérience nous a donné un aperçu des relations en entreprise. Malgré que nous ayons une position particulière par rapport à l'entreprise, nous avons été agréablement surpris de l'implication de l'entreprise dans notre projet. Nous avons été toujours très bien accueillis. Nous nous sommes sentis pris au sérieux par l'entreprise ce qui nous a permis de nous donner confiance en nous et d'exprimer notre opinion dès que nécessaire. Nous avons aussi appris à prendre en compte les remarques bienveillantes de l'entreprise au profit de notre travail. Notre relation avec l'entreprise a été basée sur le respect d'autrui ce qui nous a donné l'envie de s'investir.

Nous avons aussi développer des compétences en terme de travail en équipe. C'était un challenge de travailler avec des personnes provenant de filières différentes. Il s'est avéré que la diversité du savoir a été bénéfique pour notre projet. En effet, la répartition des tâches s'est faite de manière naturelle. Par ailleurs, la compétence pointue de membres du groupe dans certains domaines a permis aux autres membres du groupe de monter rapidement en compétence. Cette mission a été un réel partage de savoir. La bonne entente et l'écoute au sein du groupe a régné tout au long du projet : ces deux éléments ont eu un impact direct sur notre productivité. Nous pouvons dire que ce projet a été agréable à mener pour tous.

Bibliographie

https ://www.ionos.fr/digitalguide/serveur/know-how/golang/
https ://agiliste.fr/introduction-methodes-agiles/
https ://fr.wikipedia.org/wiki/SQLite
https ://www.sqlitetutorial.net/download-install-sqlite/
https ://www.sqlitetutorial.net/download-install-sqlite/
https ://medium.com/@yaravind/go-sqlite-on-windows-f91ef2dacfe
https ://www.planzone.fr/blog/quest-ce-que-la-methodologie-scrum
https ://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows
https ://git-scm.com/book/fr/v2/Les-tripes-de-Git-Les-objets-de-Git
https ://www.kenneth-truyers.net/2016/10/13/git-nosql-database/
https ://www.mongodb.com/nosql-explained

Annexes

Annexe 1 - Git-Bug

Lien vers le dépôt GitHub du logiciel Git-Bug qui a servi lors de ce projet, de sujet d'étude d'une base de données NoSQL et ainsi que d'une base de code à notre prototype final :
https ://github.com/MichaelMure/git-bug

Annexe 2 - Documentation des objets internes Git

Lien vers la documentation officielle de Git détaillant les différents objets utilisés pour la mise en place d'une base de données NoSQL dans Git détaillée dans la partie 3.3.2.
https ://git-scm.com/book/fr/v2/Les-tripes-de-Git-Les-objets-de-Git