MIAGE

LES ENJEUX DE LA MISE EN PLACE DE L'INTÉGRATION CONTINUE DANS DE PROJETS DE DÉVELOPPEMENT EXISTANTS

30 juillet 2018

Francisco Javier Martínez Lago
Université Paris-Dauphine

Table des matières

Introduction

La maximisation de la productivité est un problème omniprésent dans l'entreprise. la productivité peut permettre à une entreprise de réussir alors que son concurrent fait faillite. Il peut faire la différence entre le respect d'une échéance et la livraison tardive d'un projet. Il n'est pas étonnant qu'il existe de nombreux articles et documents de recherche consacrés à ce sujet.

Dans le développement de logiciels, la productivité est souvent recherchée par l'automatisation. L'élimination des processus répétitifs et manuels minimise les erreurs humaines et permet aux développeurs de concentrer leur énergie sur des tâches plus importantes. Cependant, la mise en œuvre de l'automatisation n'est pas toujours triviale.

Il y a deux cas où nous pouvons nous poser la question de savoir s'il faut mettre en œuvre l'automatisation dans notre projet. D'abord, quand nous commençons un nouveau projet. Deuxièmement, lorsque nous avons un projet existant que nous souhaitons rendre plus efficace par l'automatisation. Dans le présent document, nous examinerons le deuxième cas, en particulier les difficultés d'une telle entreprise et quelques exemples de mise en œuvre dans le monde réel.

Ce document est composé de trois sections. Tout d'abord, je donnerai un aperçu général de ce qu'est l'intégration continue (IC) et parlerai de certaines des technologies les plus connues dans ce domaine. Deuxièmement, je vais aborder les différents aspects qui doivent être pris en compte lors de la mise en place de l'IC dans un projet de logiciel afin d'augmenter les chances d'une mise en œuvre sans heurt. Troisièmement, je parlerai de l'IC et de la façon dont elle est gérée dans l'entreprise où j'ai fait mon alternance.

CHAPITRE 1

La Mission

Dans cette section, je donnerai brièvement des détails sur le contexte dans lequel ce mémoire a été écrit et sur mes motivations.

1.1 L'entreprise

Natixis est une banque de financement, de gestion et de services financiers, créée en 2006, filiale du groupe BPCE[1]. Elle est une banque internationale qui compte plus de 21 000 experts présents dans 38 pays.[2]. Elle propose des expertises autour des thèmes suivantes :

- Gestion d'actifs et de fortune
- Banque de grande clientèle
- Assurance
- Services financiers spécialisés

Ils développent leurs activités principalement dans trois zones géographiques :

- Amérique
- Asie-Pacifique
- EMOA (Europe, Moyen-Orient, Afrique)

1.2 La mission

J'ai travaillé au sein de la Direction des opérations et des systèmes d'information. Parmi mes responsabilités, il y avait l'intégration d'un outil de reporting open-source avec le serveur JSP existant de l'entreprise ainsi que la mise en place d'un tableau de bord pour afficher différents graphiques.

Au cours de la mission, j'ai dû communiquer avec des fournisseurs externes, rechercher de la documentation existante et en écrire une nouvelle, coder en Java et JavaScript, créer des requêtes SQL pour une base de données Oracle et apprendre à utiliser et à comprendre les solutions logicielles existantes de l'entreprise.

1.3 Les motivations et la démarche

Même si ma mission n'impliquait pas de processus d'IC, j'ai eu l'idée d'étudier ce sujet à partir de mon expérience précédente dans l'entreprise où j'ai travaillé l'année dernière. Là-bas, j'ai été directement impliqué dans la mise en place d'un flux de travail d'intégration continu. Maintenant que j'étais dans une entreprise avec une plus grande taille, j'étais curieux de savoir comment on traitait ici avec IC en termes de logiciels, de processus et de mise à l'échelle.

J'ai pris contact avec les employés d'autres secteurs de l'entreprise. J'ai parlé à des personnes chargées de créer des flux de travail d'intégration continus que d'autres départements pourraient ensuite mettre en œuvre. Je les ai interviewés avec une série de questions qui m'ont donné une meilleure perspective de l'architecture en usage. A partir de ces entretiens ainsi que de quelques informations librement disponibles en ligne, j'ai pu créer une image que je partagerai dans les chapitres suivants.

CHAPITRE 2

L'intégration continue

2.1 Principe

Dans les premiers jours du développement de logiciels, le travail des développeurs était basé sur le monde du hardware où il était important de corriger les bogues avant de sortir un produit [3]. En effet, si vous deviez produire quelque chose en masse, vous feriez mieux de vous assurer que vous avez obtenu le bon produit *avant* production. Ce modèle de développement logiciel centré sur le hardware a été incarné dans un article de Royce (1970), dans lequel il propose une approche séquentielle de la création de programmes qui comprend plusieurs phases. Chaque phase doit être complétée dans son intégralité avant de passer à la phase suivante, un peu comme si l'eau coule sans effort le long d'une chute d'eau (modèle Waterfall). Bien sûr, tout comme une chute d'eau, il est difficile et coûteux d'essayer de remonter. Par conséquent, le modèle est basé sur l'idée que chaque phase doit être bien faite pour qu'il n'y ait pas besoin de revenir aux phases précédentes. Pour cette raison, de longues périodes sont consacrées à la création de la documentation et des spécifications. Lorsque le produit final arrive, il correspond parfaitement aux exigences originales, mais il se peut qu'il ne soit plus pertinent.

Dans un monde où les besoins évoluent, l'approche rigide par phases du modèle Waterfall ne pouvait pas rester pertinente. De nouvelles méthodologies de développement sont apparues pour présenter leur solution au problème d'essayer d'être plus dynamique. Le monde a vu l'essor du Rapid Application Development, du Dynamic Systems Development, du Scrum et de Extreme Programming, entre autres. L'air du temps de ces années était un désir de vitesse et une meilleure capacité d'évolution et d'adaptation. Un exemple des pratiques mises de l'avant à cette époque était l'accent mis sur les courtes boucles de rétroaction. Plus un programmeur reçoit rapidement de la rétroaction sur un code qu'il écrit, plus il a de chances de faire la bonne chose.

Lorsque les principaux experts de ces méthodologies se sont réunis en Utah en 2001, ils ont créé le manifeste Agile. Il s'agissait d'une nouvelle approche audacieuse en matière de développement de logiciels. Il a encouragé les développeurs à se concentrer sur le software qui marche et à réagir au changement plutôt que de s'appuyer sur une documentation complète et de suivre un plan. Il se trouve que l'IC est naturellement liée à cette nouvelle

façon de faire les choses.

2.2 Qu'est-ce que c'est ?

Le terme "Intégration Continue" a été utilisé à l'origine par Grady Booch (1991) pour décrire une technique de développement de logiciels orientés objet qui impliquait que chaque développeur fasse des intégrations régulières à une branche principale et intègre leur code avec celui des autres développeurs plutôt qu'il n'y ait qu'un seul événement d'intégration "big-bang"[4]. Cette stratégie avait pour but de prévenir les problèmes d'intégration et d'aider à trouver les problèmes au début du processus de développement. Le terme a ensuite été développé dans la méthodologie Extreme Programming qui a introduit le concept d'intégration de plusieurs jours par jour[5]. Plus tard, Martin Fowler a publié un article sur son site web [6] qui décrivait l'approche en détail et suggérait les meilleures pratiques telles que la maintenance d'un référentiel unique, l'automatisation de la compilation et la réparation immédiate des builds cassés.

Si nous considérons tous ces changements au concept original d'intégration continue, nous pouvons dire que c'est maintenant compris comme étant le processus d'intégration régulière dans une branche principale et le build automatique, le test et la validation du code à chaque fois que cela se produit.

L'IC est la réponse à la question des développeurs travaillant de manière isolée. Quand ils prennent trop de temps pour intégrer leurs changements dans la branche principale, il peut y avoir des problèmes tels que des conflits de fusion, des stratégies de code divergentes, des efforts en double et des bogues. Comme CI exige que l'équipe intègre continuellement ses changements à une branche partagée, cela permet d'atténuer ces problèmes.[7].

2.3 Intégration continue dans l'entreprise aujourd'hui

L'IC a bénéficié de taux d'adoption élevés[8][9] et est utilisée par certaines des plus grandes entreprises du monde telles que Google[10] et Netflix[11].

Cependant, il y a des défis associés au passage à des pratiques continues[12]. Parmi les défis les plus courants dans l'adoption de pratiques d'intégration continue, il y a le manque d'outils et de technologies appropriés, la résistance au changement, le scepticisme quant à la valeur ajoutée des pratiques continues, l'absence d'une stratégie de test appropriée, la mauvaise qualité des tests et la résolution des conflits dans le code[13].

Malgré ces défis, l'intégration continue reste une pratique populaire.

2.4 Les plateformes d'intégration continue

Parce qu'une seule solution ne convient pas à tous, de multiples plates-formes d'intégration continue ont été créées pour répondre aux différents besoins des entreprises. Dans cette section, nous verrons quelques-unes des options les plus populaires.

Jenkins

Jenkins est une application CI open-source qui permet de construire et de tester en continu des projets logiciels.[14] C'est l'un des outils d'intégration continue les plus populaires [15] et il y a plus d'un millier de plugins qui étendent ses fonctionnalités de base. Jenkins fonctionne comme un serveur Java autonome et nécessite donc l'environnement d'exécution Java. Parce qu'il est écrit en Java, il est capable de fonctionner sur n'importe quel système d'exploitation où Java fonctionne. En outre, il prend en charge plusieurs systèmes de contrôle de version prêts à l'emploi et peut exécuter des projets Maven et Ant ainsi que des scripts shell, des commandes batch de Windows et bien d'autres. En outre, il prend en charge l'envoi de notifications par courrier électronique.

Jenkins a été créé par Kohsuke Kawaguchi en 2004 et s'appelait initialement Hudson, mais a été rebaptisé Jenkins en 2011 en raison de différends avec Oracle[16].

Un workflow de base de Jenkins pourrait ressembler à ceci :

1. Un développeur soumet son code à un référentiel de contrôle de version.
2. Jenkins prend les changements et construit et teste la nouvelle version.
3. Si la compilation échoue, Jenkins envoie des notifications aux développeurs.

L'état du build sera disponible sur le tableau de bord Jenkins.

Pour commencer avec Jenkins, un job doit être créé. Un job se compose de trois aspects :

- **Localisation des fichiers à construire**. Il peut s'agir d'un répertoire local ou d'un référentiel de contrôle de version.
- **Un ensemble d'instructions à suivre**. Cela peut être exécuter un script ou exécuter une commande batch de Windows.
-
- **Un déclencheur**. C'est à cette condition que Jenkins commencera à faire le build. Les options principales sont basées sur le temps (ex : toutes les 20 minutes) et sur les événements (ex : chaque fois qu'il y a un commit sur la branche principale). Il peut aussi être plus avancé, comme lier l'exécution d'un build à la réussite d'un autre travail.

L'interface de Jenkins est accessible par le biais d'un navigateur web. L'interface principale de Jenkins est montrée dans la figure 2.1.

L'une des principales forces de Jenkins est sa polyvalence. Il y a plus d'un millier de plugins disponibles pour étendre ses fonctionnalités. De plus, puisqu'il s'agit d'une application open-source, n'importe qui peut écrire des plugins pour cette application. C'est un avantage énorme pour les entreprises, qui doivent souvent se conformer à des réglementations spécifiques et sont donc intéressées par le développement de solutions personnalisées.

Travis-CI

Travis-CI a été créé en 2011 par la société CB Rank. C'est un service d'IC, distribué et hébergé, utilisé pour construire et tester des projets logiciels hébergés chez GitHub. Travis-CI surveille en permanence les changements dans un référentiel de contrôle de version, construit et teste automatiquement les changements de code et fournit une rétroaction

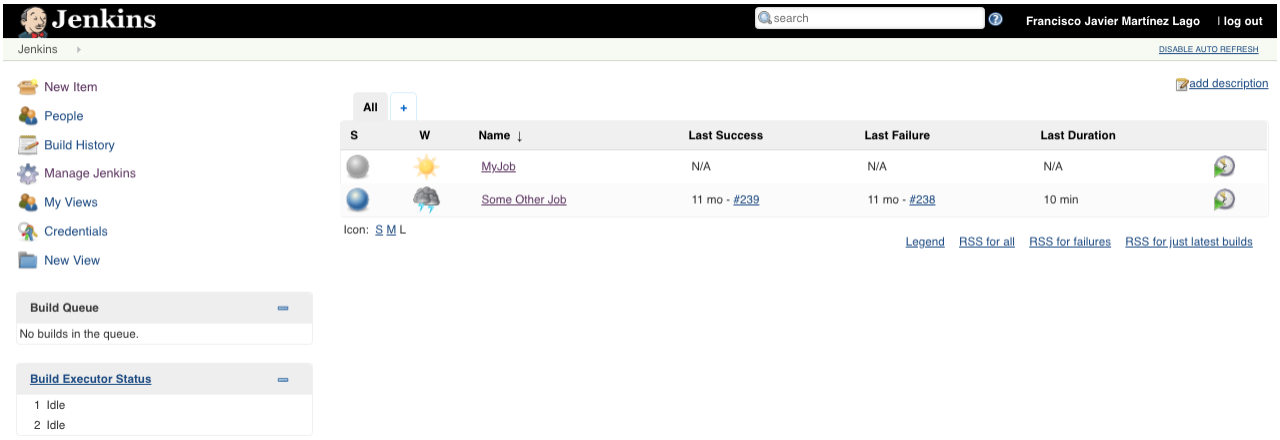

Figure 2.1 – Interface principale de Jenkins

immédiate sur le succès ou l'échec du build[17]. Les builds peuvent être personnalisés en utilisant un fichier spécial appelé « .travis.yml » qui doit être inclus dans le dossier racine du projet.

Un workflow Travis-CI de base peut ressembler à ceci :

1. Un développeur soumet son code
2. Travis-CI recueille les modifications et lit le fichier travis.yml pour prendre en compte les paramètres spécifiques au projet.
3. Travis-CI clone le référentiel de contrôle de version dans un nouvel environnement virtuel et construit et teste le code.
4. Les notifications sur le résultat de la compilation sont envoyées aux développeurs concernés.

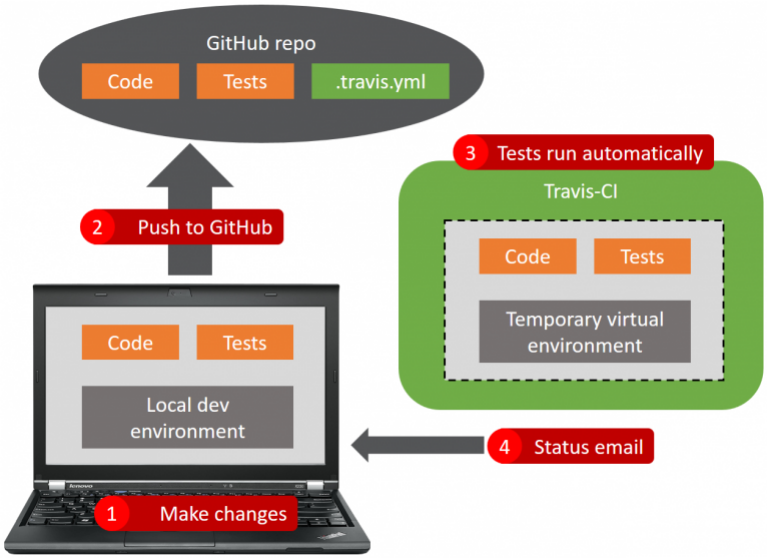

Figure 2.2 – Architecture de Travis [18]

Dans Travis-CI il y a quelques concepts fondamentaux :

- **job** : un processus automatisé qui clone votre référentiel dans un environnement virtuel et exécute une série de phases telles que la compilation de votre code, l'exécution de tests, etc. Un job échoue si le code de retour de la phase de script est différent de zéro.
-
- **phase** : les étapes séquentielles d'un job. Par exemple, la phase d'installation précède la phase de script, qui précède la phase de déploiement optionnel.
-
- **build** : un groupe de jobs. Par exemple, une compilation peut avoir deux jobs, dont chacun teste un projet avec une version différente d'un langage de programmation. Un build se termine lorsque tous ses jobs sont terminés.
- **stage** : un groupe de jobs exécutés en parallèle dans le cadre d'un processus de construction séquentielle composé de plusieurs étapes.

Un avantage de Travis-CI est qu'il n'y a pas de serveur à configurer. Au lieu de cela, il suffit d'inclure un fichier de configuration dans un référentiel de contrôle de version et de lier le compte Travis-CI avec GitHub. De plus, Travis-CI facilite le test du code dans de multiples environnements en utilisant ce qu'ils appellent une matrice de construction. Cela permet de mettre en place et de construire rapidement différents scénarios de cas pour l'application (ex : tester le logiciel avec différentes versions d'une bibliothèque spécifique).

Travis est gratuit pour les projets open-source et a des plans payants pour d'autres types.

BuildBot

Buildbot est un système de planification des tâches basé sur Python et publié sous licence GPL. Il met en file d'attente les jobs, exécute les jobs lorsque les ressources nécessaires sont disponibles et rapporte les résultats[19].

Une installation BuildBot se compose de machines agissant en tant que maître (ou maître) et d'esclaves. Le maître vérifie périodiquement les changements dans le code source d'un référentiel de contrôle de version, coordonne les activités des travailleurs et renvoie les résultats des travaux aux développeurs. Les travailleurs peuvent faire son boulot sur une variété de plates-formes.

BuildBot est configuré par un script de configuration Python sur le maître.

Voici à quoi pourrait ressembler un workflow BuildBot typique :

1. Une machine maître interroge le référentiel de contrôle des versions et tire les dernières modifications.
2. La machine maître envoie des commandes aux machines esclaves pour construire et tester le code.
3. Le maître reçoit les résultats des travaux et envoie les rapports aux développeurs par e-mail.

Les esclaves s'exécutent généralement sur des machines séparées, qui peuvent correspondre chacune à une plate-forme d'intérêt. Ils envoient une demande au maître d'œuvre

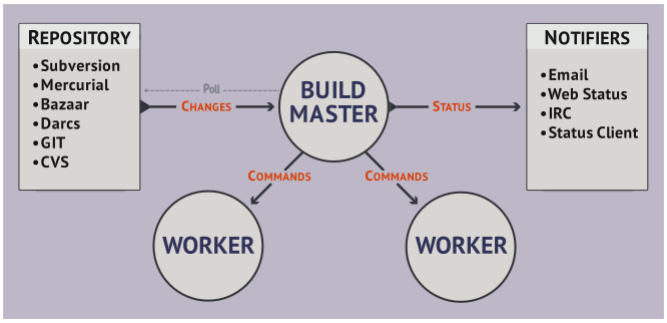

Figure 2.3 – Architecture de BuildBot [20]

pour initier une connexion TCP. Si la connexion réussit, le maître de construction l'utilise pour envoyer des commandes et les ouvriers l'utilisent pour envoyer les résultats de ces commandes.

Le travailleur doit pouvoir accéder au référentiel de contrôle de version car il n'obtient pas le code source du maître de construction.

Étant écrit en Python, BuildBot peut s'intégrer avec d'autres outils et bibliothèques écrites dans le langage. De plus, il peut être fortement personnalisé.

Figure 2.3 montre l'architecture de base de BuildBot.

TeamCity

TeamCity est un serveur de gestion de builds et d'IC basé sur Java et crée par JetBrains. Il a été publié pour la première fois le 2 octobre 2006.

Selon la documentation officielle, les fonctionnalités suivantes sont proposées par TeamCity[21] :

- Exécuter des builds en parallèle sur de plateformes et environnements différentes.
- Optimisez le cycle d'intégration du code et assurez-vous de ne jamais avoir du code cassé dans le référentiel.
- Examiner les résultats des tests à la volée avec des rapports de tests intelligents et le réordonnancement des tests intelligents.
- Personnaliser les statistiques sur la durée de build, le taux de succès, la qualité du code et des mesures personnalisées.

Un simple workflow TeamCity pourrait ressembler à ceci :

1. Le serveur TeamCity détecte un changement dans le référentiel de contrôle de version et enregistre les changements dans une base de données.
2. Le déclencheur des builds voit le changement dans la base de données et ajoute un build à la file d'attente.
3. Le serveur trouve un agent de compilation compatible pas occupé et assigne la construction en file d'attente à cet agent.
4. L'agent exécute les étapes de construction. Pendant que les étapes de construction sont en cours d'exécution, l'agent de construction rapporte la progression de la construction au serveur TeamCity en envoyant tous les messages de log, les rapports de test, les résultats de couverture de code, etc. au serveur à la volée.

5. Après avoir terminé la compilation, l'agent de compilation envoie Build Artifacts au serveur.

La figure 2.4 montre une des interfaces disponibles dans Teamcity. Celui-ci liste toutes les builds en cours d'exécution avec quelques méta-informations.

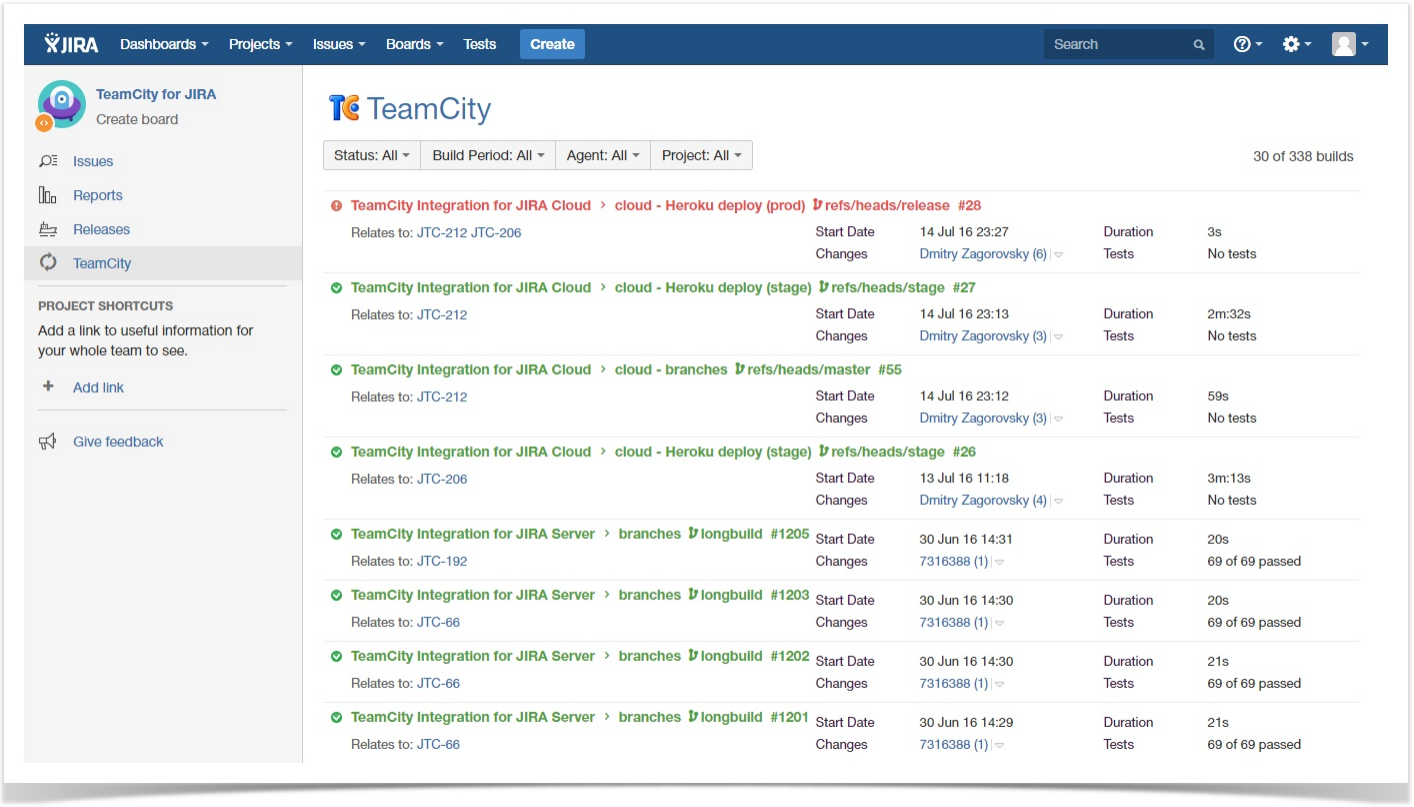

Figure 2.4 – Builds sur TeamCity [22]

CHAPITRE 3

Mise en œuvre de l'IC dans un projet existant

Il y a deux scénarios qui peuvent se produire lorsque l'on pense à mettre en œuvre l'IC dans un projet.

Le premier cas est l'intégration de CI dans un projet complètement nouveau. C'est l'étape la plus flexible car aucun code n'a été engagé et il y a de la place pour la mise en œuvre de nouvelles idées avant que les objectifs ne soient fixés.

La deuxième option consiste à mettre en œuvre une intégration continue dans un projet existant. Par projet existant, j'entends un projet qui est passé de la phase de conception à la phase de développement. En effet, l'orientation du projet a été fixée et le développement est en cours.

Dans cette section, je vais explorer la deuxième option.

La mise en œuvre de l'IC dans une infrastructure existante pourrait s'avérer une entreprise non négligeable. En effet, il faut tenir compte non seulement du travail de mise en place de la plateforme choisie mais aussi de la culture de codage en place dans l'environnement de développement. Si les codeurs sont habitués à coder de manière isolée, par exemple, il faudra du temps et des efforts pour changer leurs habitudes de codage.

Phases dans un projet de développement

Regardons les phases possibles que nous pouvons rencontrer dans un projet. [23] :

1. **Pas de serveur de build** Il n'y a pas de serveur central de compilation. Les builds sont exécutés sur les machines des développeurs, soit manuellement, soit par le biais d'un script comme Ant. Le code source peut être stocké dans un référentiel de contrôle de version mais les développeurs n'ont pas l'habitude de commettre régulièrement des changements au code. Avant la livraison, un développeur intègre les changements manuellement.
2. **Builds quotidiens** Maintenant il y a un serveur web qui fonctionne régulièrement (généralement la nuit). Le serveur compile le code, mais il n'y a pas de test automatisé.

Même s'il y a des tests, ils ne font pas partie du processus de construction. Les développeurs apportent des changements plus régulièrement, même si ce n'est qu'à la fin de la journée. S'il y a des conflits de code, le serveur de compilation avertira l'équipe le matin.

3. **Builds quotidiens et tests automatisés basiques** L'équipe prend l'IC un peu plus au sérieux maintenant. Le serveur de compilation s'exécute maintenant chaque fois qu'il y a un nouveau code sur le référentiel de contrôle de version. Le script de compilation ne se contente pas de compiler mais exécute également des tests unitaires et/ou d'intégration automatisés. Le serveur de compilation envoie également des notifications instantanées en cas de problème, non seulement par émail mais aussi par messagerie instantanée. Les échecs de build sont traités rapidement par l'équipe.
4. **Arrivée des métriques** Des métriques de qualité automatisées sont maintenant utilisées pour aider à évaluer la qualité du code et la couverture du test. Le build mesurant la qualité du code a également créé la documentation de l'API de l'application. Un tableau de bord est également à la disposition de tous les membres de l'équipe pour voir l'état d'avancement du projet en un coup d'œil.
5. **Prendre les tests au sérieux** Des tests plus approfondis sont effectués après les tests de base, tels que les tests de bout en bout et les tests de performance.
6. **Tests d'acceptation automatisés et un déploiement plus automatisé** Acceptance Test Driven Development est utilisé tout au long du projet pour faire avancer le développement et générer des rapports de haut niveau sur l'état du projet.
7. **Déploiement Continu** La confiance dans le système d'intégration continue est telle que l'équipe est confiante de laisser les modifications être automatiquement déployées en production.

Cette liste n'est pas exhaustive et la progression entre les phases peut ne pas correspondre exactement aux situations du monde réel, mais elle donne une idée globale sur la stratégie d'intégration continue à mettre en œuvre dans une organisation du monde réel.

Dans la section suivante, nous verrons les contextes spécifiques qui pourraient survenir lors de la mise en œuvre de l'IC et nous verrons comment ils ont été abordés dans la pratique.

3.1 Adaptation des applications monolithiques

Dans cette section, nous allons examiner les défis uniques de la mise en œuvre de processus d'IC dans des applications monolithiques et quelques solutions possibles.

Qu'est-ce qu'une application monolithique ?

Un monolithe est essentiellement un gros bloc de pierre. De même, une application monolithique est un logiciel construit sans modularité, un gros bloc de code qui est censé traiter de nombreuses requêtes différentes par lui-même.

Les applications monolithiques sont généralement de grande taille et de grande envergure. De plus, ils ne sont pas faciles à déboguer. Comme les différents composants sont étroitement liés, un changement dans une partie du code pourrait facilement avoir des répercussions ailleurs.

Quelques solutions essayées à des problèmes monolithiques

Nous verrons des entreprises qui ont dû faire face à la question de l'intégration continue dans des applications monolithiques. Nous décrirons également les solutions qu'ils ont trouvées.

Nextdoor Engineering[24]

Nextdoor est une société de services techniques. Ils avaient une équipe de plus de soixante-dix ingénieurs travaillant sur une application monolithique basé sur Django. Ils voulaient simplifier leur flux de travail de déploiement continu, mais ils étaient confrontés au problème que leurs constructions prendraient environ 25 minutes chacune à exécuter. Si chaque construction ne comportait qu'un seul changement, cela signifie que les constructions feraient la queue et pourraient prendre des heures jusqu'à ce qu'elles arrivent en production.

Pour résoudre ce problème, ils ont introduit le concept d'un train. Un train est simplement un lot de changements libérables. Pour déterminer ce qu'il y a à bord d'un train, ils ont un processus logique simple. Lorsqu'un changement atterrit sur le maître, s'il y a déjà un train existant, il est mis en file d'attente pour le train suivant. Si un tel train n'existe pas, un nouveau train est créé à partir de la file d'attente des changements en attente.

Les trains passent par trois phases : la livraison, la vérification et le déploiement.

1. **Première phase - livraison :** cela inclut la construction et le déploiement des changements dans l'environnement de staging.
2. **Deuxième phase - la vérification :** cela inclut la vérification automatisée comme les tests et la vérification humaine avec JIRA.
3. **Troisième phase - déploiement automatique en production**

La fonctionnalité des trains a permis à Nextdoor de regrouper plusieurs changements, ce qui résout à son tour la question des constructions à changement unique qui ralentissent l'ensemble du système d'intégration continue.

Conductor[25]

Searchlight est une plateforme permettant d'obtenir des informations sur les habitudes de recherche des clients et de fournir des recommandations personnalisées. Parce que l'application elle-même a été construite comme une grande application monolithique, ils ont trouvé des problèmes. Ils ont constaté que lorsqu'un développeur changeait une partie de l'application, il fallait tester l'ensemble de l'application pour s'assurer que tout fonctionnait.

De plus, s'ils voulaient améliorer leur performance dans un domaine, ils étaient obligés de considérer l'application dans son ensemble.

Pour résoudre ce problème, ils ont décidé d'utiliser les microservices. Ils l'ont fait en se concentrant sur trois grands principes :

1. **Définir les normes organisationnelles pour la conception, le développement et le déploiement des microservices.** Des normes communes pour la création de microservices simplifient le développement à long terme.
2. **Construisez de nouvelles fonctionnalités en tant que microservices, retirez les fonctionnalités existantes au besoin** Ne construisez que de nouvelles fonctionnalités et des fonctionnalités existantes (là où cela a du sens) en tant que microservices.
3.
4. **Utiliser des solutions prêtes à l'emploi autant que possible**.

Pour construire et tester ces microservices, ils ont décidé d'utiliser l'application CI Jenkins.

Avec cette nouvelle infrastructure de microservices en place, ils ont réussi à simplifier la construction et les tests, puis ils ont pu déployer quotidiennement du code monolithique et déployer des microservices plusieurs fois par jour.

3.2 Intégration avec les méthodologies de développement logiciel

La mise en œuvre de l'IC exige un changement de paradigme. Les anciennes méthodologies logicielles de V-Model ou Waterfall ne peuvent plus soutenir cette nouvelle façon de faire les choses. Bien qu'il soit possible d'adopter l'IC avec ces méthodologies, ce n'est ni pratique ni optimal.

La seule façon de tirer pleinement parti de l'IC est d'adopter une nouvelle mentalité et d'utiliser des pratiques de développement modernes. Ces pratiques se caractérisent par des intégrations fréquentes et des constructions et tests automatisés de logiciels.

Heureusement, de telles pratiques existent. En effet, plusieurs des méthodologies basées sur l'approche Agile sont bien adaptées. C'est pourquoi, dans cette section, nous allons les explorer. Tout d'abord, je vais définir ce qu'est une méthodologie de développement logiciel. Ensuite, j'explorerai plusieurs des méthodologies Agiles actuellement disponibles qui peuvent bénéficier de l'IC.

3.2.1 Principe

Une définition possible de la méthodologie de développement de logiciels est qu'il s'agit « d'un cadre utilisé pour structurer, planifier et contrôler le processus de développement d'un système d'information » [26].

Le dictionnaire Larousse dit qu'un cadre est « Ce qui borne, limite l'action de quelqu'un, de quelque chose ; ce qui circonscrit un sujet ». Si nous appliquons cette définition, nous

pouvons dire qu'une méthodologie de développement logiciel délimite la planification, la structuration et le contrôle du processus de développement logiciel.

Aujourd'hui, dans l'entreprise, il existe de nombreuses méthodologies différentes pour le développement de logiciels. Nous entendons des termes tels que Scrum, Kanban, XP, Lean Development et plus encore. Ceux-ci sont tous basés sur les principes de l'Agile. Mais qu'est-ce qu'Agile exactement ?

3.2.2 Agile

Agile est un terme générique qui englobe une série de méthodes et de pratiques basées sur les valeurs et les principes exprimés dans le Manifeste Agile[27]. Le Manifeste, créé en 2001 est le résultat d'un libre échange d'idées par les représentants de diverses méthodologies de développement de logiciels qui voulaient trouver une alternative aux processus de développement de logiciels lourds et axés sur les documents[28].

Le manifeste agile établit ce qui suit :

- « *Les individus et leurs interactions* plus que les processus et les outils »
- « *Des logiciels opérationnels* plus qu'une documentation exhaustive »
- « *La collaboration avec les clients* plus que la négociation contractuelle »
- « *L'adaptation au changement* plus que le suivi d'un plan »

Et à propos de ces valeurs, ils disent : « Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers éléments. ». En gros, ils donnent la priorité aux éléments de gauche tout en n'ignorant pas ceux de droite.

Nous avons parlé dans le chapitre 2 que les développeurs de l'époque cherchaient des moyens d'améliorer leur vitesse et leur adaptabilité. Agile embrasse cela. Alors que les approches traditionnelles ont essayé de tout prévoir et donc de réduire les coûts en éliminant le changement, Agile accepte que les changements sont inévitables et prend donc l'approche alternative d'essayer de réduire leurs coûts[29].

Agile se concentre sur des cycles itératifs courts qui ont une durée recommandée de deux à six semaines. Dans ce laps de temps, les développeurs travaillent sur des fonctionnalités dont ils ont discuté avec le ou les clients. Les développeurs, individuellement ou en binôme, travaillent ensuite sur les tâches qui les conduiront à atteindre leur objectif. Un produit livrable est construit par l'intégration continue du code de tous les développeurs. A la fin du cycle, les développeurs présentent un programme qui marche aux clients. Ensuite, une nouvelle direction est définie pour le cycle suivant.

Comme indiqué dans l'article « What is Agile Software Development. » (Highsmith, 2002), le cadre agile brille lorsque « Projects peut avoir une mission relativement claire, mais les exigences spécifiques peuvent être volatiles et évolutives à mesure que les clients et les équipes de développement explorent l'inconnu.

Examinons maintenant certaines des méthodologies qui embrassent le cadre Agile.

3.2.3 Kanban

Le Kanban est une méthodologie qui découle des plans de fabrication de Toyota. Ils ont été inspirés par l'efficacité des supermarchés à stocker juste assez de produits pour répondre

à la demande des consommateurs. Pour Toyota, l'objectif était de faire correspondre les niveaux de stocks avec la consommation réelle. Pour communiquer les niveaux de capacité en temps réel, les travailleurs passeraient une carte, ou « kanban », entre les équipes[30].

De nos jours, le concept a évolué, mais les principes restent les mêmes. Les équipes agiles utilisent le Kanban pour faire correspondre la quantité de travail en cours à la capacité de l'équipe. Le travail de l'équipe est affiché sur un tableau Kanban accessible à tous les membres de l'équipe. Une tableau physique ou numérique peut être utilisée à cet effet.

Qu'il s'agisse d'un tableau physique ou numérique, un tableau a un flux de travail composé de 3 colonnes : « À faire », « En cours », « Fait » Les développeurs ajoutent des tâches sous ces colonnes. Ils peuvent aussi ajouter d'autres colonnes au besoin.

La mise en œuvre de cette méthodologie est rapide et bon marché pour les équipes de développement logiciel. Il y a peu ou pas de frais et les concepts sont faciles à apprendre.

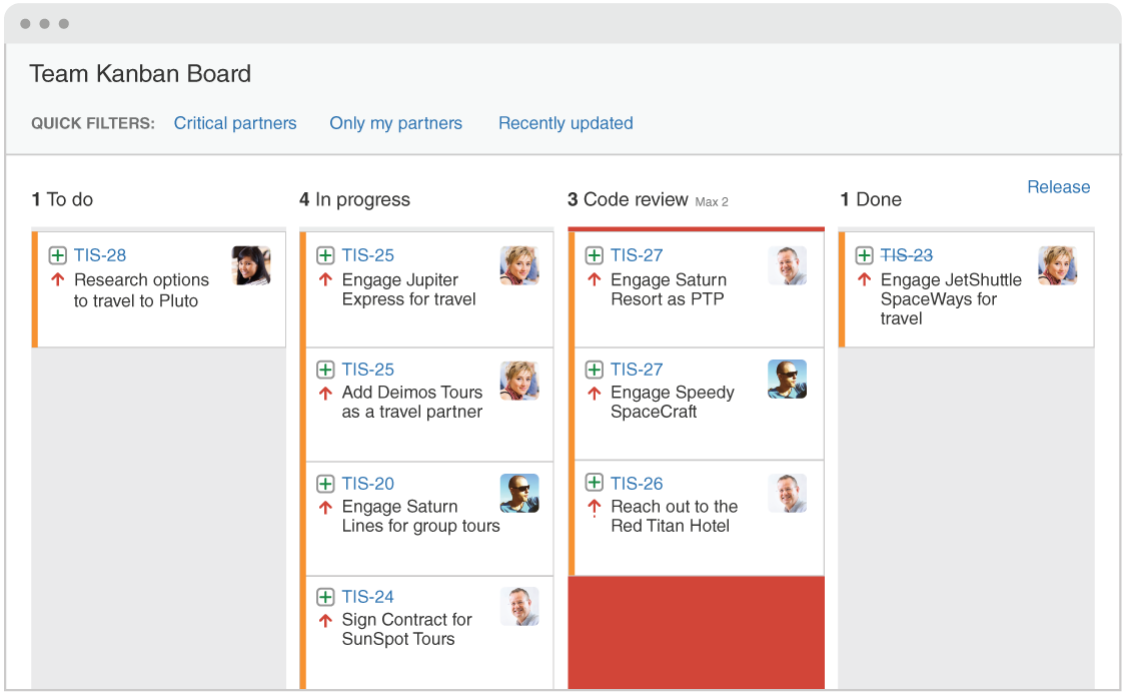

Figure 3.1 – Un tableau Kanban [30]

Le Kanban et l'intégration continue partagent les mêmes objectifs : accélérer le développement en optimisant le flux de travail. Il n'est donc pas surprenant qu'ils vont ensemble de manière transparente.

Dans le Kanban, les développeurs sont encouragés à travailler sur des tâches spécifiques. Comme ces tâches sont généralement de faible envergure, il est plus facile de les intégrer à la branche principale dans le cadre du modèle d'intégration continue. De plus, les développeurs utilisant le Kanban peuvent facilement voir s'il y a quelque chose qui ralentit le processus de développement. Si c'est le cas, ils peuvent choisir d'automatiser ce processus avec CI.

3.2.4 Scrum

Hirotaka Takeuchi et Ikujiro Nonaka ont introduit le terme scrum dans le contexte du développement de produits dans leur article de Harvard Business Review de 1986, « New Product Development Game »[31].

Scrum est une méthodologie ou un cadre pour fournir un logiciel de travail en cycles appelés sprints[32]. Il y a trois rôles dans Scrum : *Product Owner*, *Scrum Master* et *Team Member*. De plus, Scrum utilise trois artefacts : *Product Backlog*, *Sprint Backlog* et *Burndown Chart*. Ceux-ci aident à guider l'équipe pendant les sprints.

Le Product Owner s'occupe du Product Backlog. Il s'agit d'une liste de tous les résultats souhaitables que les utilisateurs attendent du produit. Fondamentalement, c'est une liste de choses à faire qui est triée par importance. Le Burndown Chart nous permet de voir le travail qu'il nous reste à faire par rapport au temps qu'il nous reste. Il est utilisé pour prédire quand tous les travaux seront terminés.

Scrum définit trois cérémonies : *sprint planning*, *daily standup* et *sprint review and retrospective*. Le Scrum Master est un facilitateur qui aide les membres de l'équipe à tenir les cérémonies et à utiliser les artefacts.

Sprint planning est le moment où l'équipe prend les objectifs énumérés dans le Product Backlog et commence à les assigner au Sprint Backlog. Il s'agit d'une conversation sur ce qui doit être construit dans le prochain sprint. Chaque jour, l'équipe fait un standup et chaque membre rend compte de trois choses : ce que le membre de l'équipe a fait le dernier jour, ce qu'il fera aujourd'hui et ce qui, s'il y a lieu, les bloque. La rétrospective du sprint est généralement la dernière chose faite dans un sprint. L'ensemble des équipes participe à ce processus. L'idée est de trouver ce qui peut être amélioré à l'avenir. Un exemple d'approche consiste à demander à chaque membre des choses précises que l'équipe devrait :

- Commence à faire/
- Arrêter de faire.
- Continuer à faire

Un sprint a une date de début et une date de fin. Pendant le sprint, l'équipe va travailler sur les tâches et les compléter. Pendant le sprint, l'état des tâches passera de « À Faire » à « En Cours » et enfin à « Fait ».

3.2.5 Rapid Application Development

Rapid Application Development (RAD) a été introduit dans le livre de James Martin en 1991 [34].

RAD décrit une méthode de développement logiciel qui met l'accent sur le prototypage rapide et la livraison itérative.

Le RAD fonctionne par cycles, chacun d'entre eux étant composé de 4 étapes :

1. **La planification des besoins.** Les développeurs, les concepteurs et les utilisateurs se réunissent pour s'entendre sur la portée et les exigences du projet.
2. **Design utilisateur.** La rétroaction des utilisateurs est recueillie dans le but d'établir une architecture de système. Cette étape est répétée aussi souvent que nécessaire.

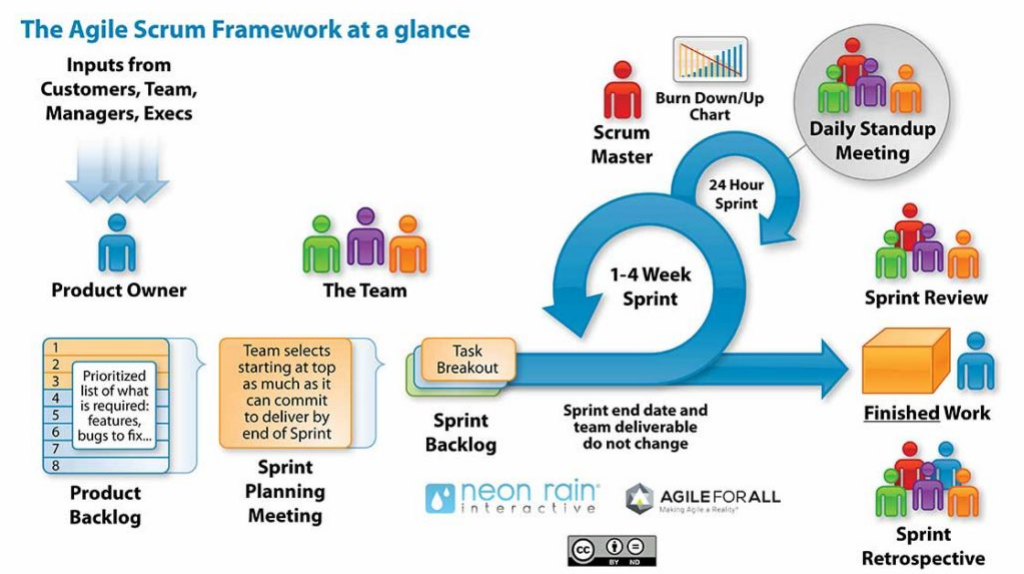

Figure 3.2 – Le framework Scrum [33]

3. **Construction rapide.** La plupart des activités de codage, de test et d'intégration se déroulent à cette étape. Cette étape peut également être répétée si, par exemple, des modifications sont apportées aux besoins du projet.
4. **Transition.** L'équipe de développement déplace le développement dans un environnement de production en direct.

Examinons brièvement certaines des caractéristiques de RAD :

- **Prototypage.** Un prototype rapide permet aux utilisateurs d'utiliser le logiciel et de fournir une rétroaction sur un système en direct plutôt que d'essayer d'évaluer les spécifications écrites sur un document de conception.
- **Développement interactif**. Chaque version est revue avec le client pour produire des exigences qui alimentent la version suivante. Le processus est répété jusqu'à ce que toutes les fonctionnalités aient été développées.
- **Time-boxing.** Le système à développer est divisé en un certain nombre de composants ou de « time boxes » qui sont développés séparément.

3.3 Barriers to adoption

Dans cette section, nous allons examiner certaines des complications les plus courantes constatées par les entreprises lorsqu'elles tentent de promouvoir l'adoption de processus d'intégration continue à l'interne.

Premièrement, il y a le coût de l'adoption d'une nouvelle technologie.

L'intégration continue nécessite l'utilisation de logiciels spécialisés. Cela a un coût monétaire, mais plus important encore, un coût en heures-homme. Les développeurs doivent

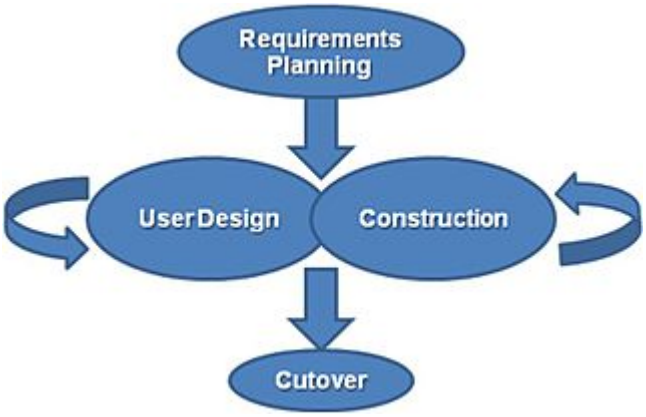

Figure 3.3 – Les 4 étapes de Rapid Application Development [35]

consacrer du temps à la formation pour apprendre à utiliser, intégrer et mettre en œuvre les nouveaux logiciels.

Deuxièmement, il y a le coût phychologique[36].

L'utilisation de pratiques d'intégration continue exige un changement de mentalité. Les développeurs travaillant de manière isolée doivent exposer leur travail dès le début et s'engager dans la collaboration. La perspective d'avoir à le faire peut les amener à montrer une certaine résistance aux changements.

Ways in which developers can show resistance to adopting a CI mindset

- **Le scepticisme.** Pour une équipe qui n'a jamais travaillé avec CI, les avantages de la mise en œuvre d'une telle pratique pourraient ne pas être immédiatement évidents. Ils pourraient se demander pourquoi ils doivent consacrer leur temps et leurs efforts à la mise en œuvre d'un nouveau système alors que l'ancien fonctionne très bien. Il est important de pouvoir expliquer clairement les avantages de l'intégration de l'IC et de répondre aux questions des développeurs.
- **Changement de vieilles habitudes.** L'intégration continue exige que les développeurs intègrent leur code fréquemment. Si un développeur est habitué à développer du code de manière isolée pendant longtemps, il peut avoir du mal à changer sa façon de travailler.
- **Exposer le travail.** L'intégration continue exige que les développeurs intègrent leur code rapidement. Les développeurs habitués à intégrer leur code dans un événement big-bang peuvent préférer travailler sur leur branche pendant un certain temps, en corrigeant les bogues qui pourraient survenir. Avec l'IC, ils sont tenus de publier leur travail tôt, que le code soit de haute qualité ou non. Cela peut être décourageant car un développeur peut se sentir anxieux à l'idée que son code soit examiné par ses pairs.
- **Objectifs peu clairs.** Une façon possible pour une entreprise de présenter CI à ses employés est de demander à une équipe pilote de le faire d'abord. Cette équipe pilote peut aider les autres équipes à mettre en œuvre le processus. Par conséquent, ces autres équipes qui mettent en œuvre le CI pourraient se plaindre qu'il n'y a pas d'objectif clair. Au lieu de cette sorte de liberté qui facilite la vie des développeurs, il peut plutôt leur faire préférer des instructions plus spécifiques sur la façon de mettre

en œuvre l'IC.

- **Pression accrue.** C'est une augmentation de la pression. La pression de la direction ainsi qu'un sentiment de manque d'expérience pour s'adapter à de nouvelles exigences peuvent démoraliser une équipe. Par conséquent, il serait dans le meilleur intérêt de mettre en œuvre des changements graduels plutôt que d'essayer de tout faire en même temps.

CHAPITRE 4

Intégration continue dans l'entreprise

Introduction

Chez Natixis, l'intégration continue est au cœur des efforts de développement de nombreux départements. En effet, en tant qu'institution financière qui doit suivre des normes strictes de sécurité et d'utilisabilité, la nécessité d'effectuer de multiples contrôles et tests sur chaque build est d'une importance primordiale. Il serait impensable pour une si grande entreprise d'avoir besoin d'effectuer ces tests manuellement. Par conséquent, une solution sous la forme d'une intégration continue est naturellement mise en place.

4.1 Une plate-forme d'intégration continue pour tous

Afin de standardiser la façon dont les équipes utilisent l'IC, une plateforme basée sur Jenkins a été mise à disposition. Toute équipe peut demander l'accès à cette plateforme et, ce faisant, elle aura accès à une variété d'outils (voir figure 4.1).

Lorsque leur demande est acceptée, les équipes peuvent bénéficier de Jenkins, la principale plateforme CI, ainsi que de nombreux outils et applications qu'elles peuvent intégrer dans leurs scripts de construction. Il est à noter que les équipes n'ont pas besoin d'utiliser toutes les fonctionnalités disponibles. En fait, chaque équipe peut personnaliser le serveur de construction Jenkins à leurs propres besoins.

Nous allons maintenant examiner les différentes étapes d'une éventuelle construction d'un CI chez Natixis.

Étape 1 : Récuperation des dépendances

Cette phase consiste à télécharger les dépendances des paquets utilisés dans le projet de développement.

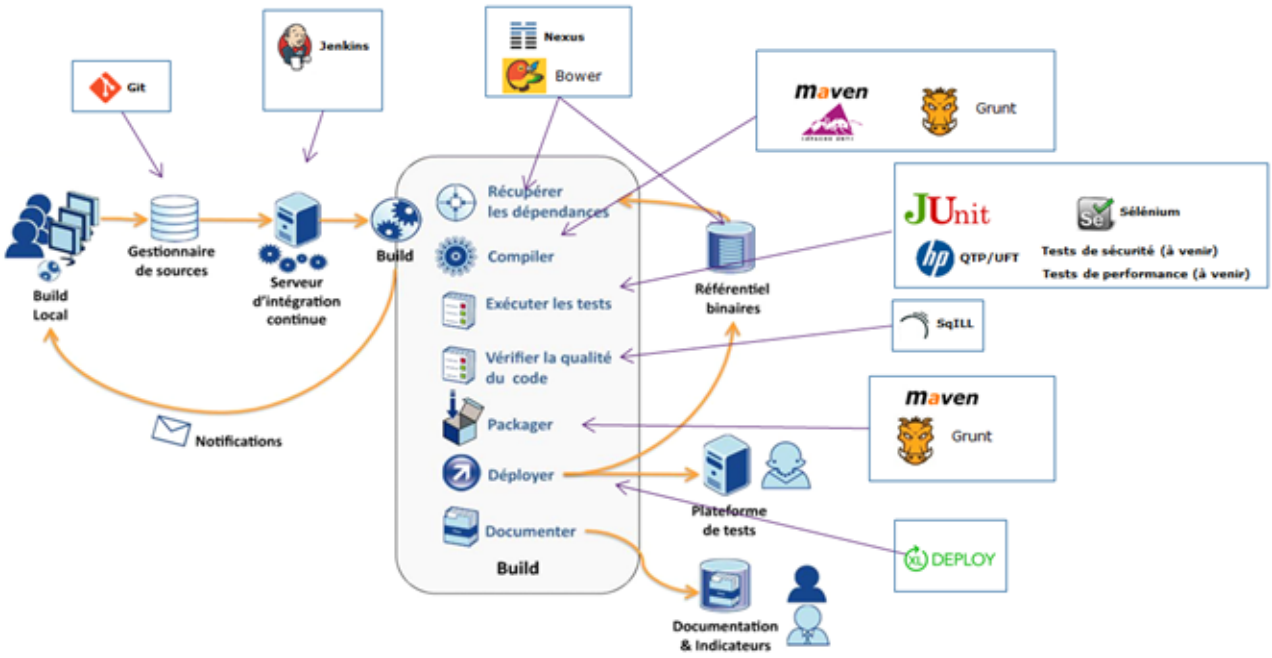

Figure 4.1 – La plateforme d'integration continue chez Natixis

Étape 2 : Compilation

Le code est compilé.

Étape 3 : Exécution des tests

On vérifie le bon fonctionnement du logiciel avec plusieurs types de tests.

Étape 4 : Vérification de la qualité du code

On vérifie la qualité du code (code smells, bugs, vulnérabilités)

Étape 4 : Packaging du code

Le code est mis ensemble dans un format qui peut être distribué.

Étape 5 : deploiement du code

Transfert du package créé à l'étape 4 vers une instance de XLDeploy.

Étape 6 : documentation

Création de la documentation par les développeurs si besoin.

Logiciels mis à disposition

- **Maven** : construction et résolution des dépendances.
- **Grunt** : automatisation de JavaScript.
- **JUnit** : tests unitaires.
- **QTP/UFT** :test fonctionnels et de régression en simulant des événements souris ou clavier sur des GUIs.
- **Selenium** : tests en simulant des actions sur un navigateur web.
- **SonarQube (SqILL en interne)** : trouve les bugs, les code smells et les vulnérabilités dans le code.
- **Nexus** : stockage centralisé et organisé de données.
- **Bower** : gestionnaire de paquets.

4.2 Projet Fenergo

Introduction

Fenergo est un logiciel de gestion du cycle de vie du client (CLM), sélectionné par Natixis pour offrir un point d'entrée unique pour le processus KYC (Know Your Client), la Due Diligence réglementaire, les Credit Limit et les Master Agreement Requests. Ces processus étaient auparavant pris en charge par plusieurs applications.

Le projet de mise en œuvre de Fenergo a débuté en août 2017, avec un objectif de mise en service d'un produit minimum viable (MVP) au début du deuxième trimestre 2018. Le projet est toujours en cours de développement en juillet 2018.

Mon implication dans le projet

J'ai pris conscience de Fenergo lorsque je regardais des projets qui utilisaient Jenkins et l'intégration continue. J'ai rapidement identifié un processus où je pouvais offrir des améliorations (voir Figure **??**). Le processus implique une mise à jour vers une nouvelle version du logiciel Fenergo. Chaque fois qu'une nouvelle version de Fenergo est disponible, une série d'étapes doit avoir lieu. Ces étapes comprennent le téléchargement du paquet, la lecture d'une version et d'un numéro d'itération, son décompression dans des dossiers spécifiques, le téléchargement des sources de Fenergo dans un dépôt Artifactory et quelques étapes optionnelles.

Quand j'ai vu que les étapes devaient être exécutées manuellement chaque fois qu'une nouvelle version de Fenergo était disponible, j'ai pensé que l'ensemble du processus pouvait être un bon candidat pour faire partie du processus d'intégration continue. En effet, si le processus pouvait être automatisé, non seulement il éliminerait l'inefficacité et l'erreur humaine, mais il augmenterait aussi la productivité en libérant du temps pour d'autres tâches.

— To Work On —

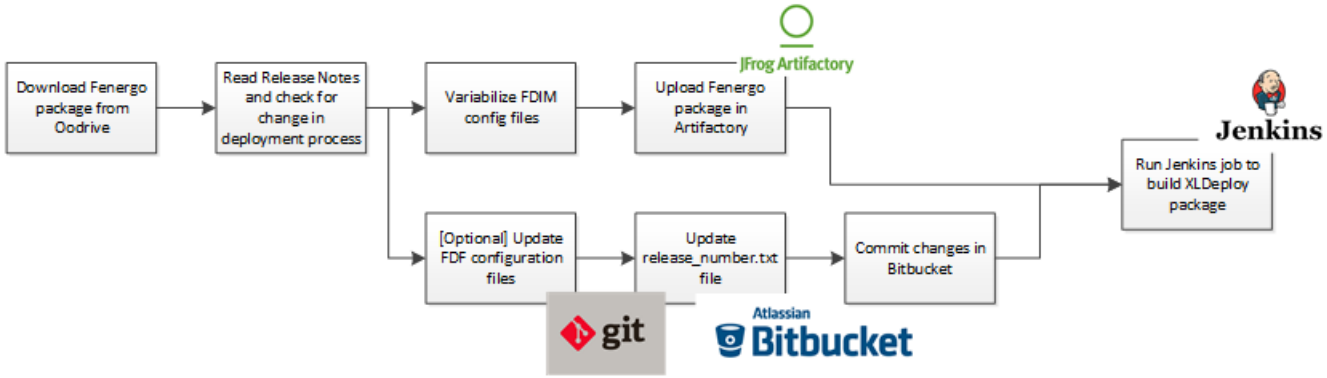

Figure 4.2 – La procédure de déploiement de Fenergo

Bibliographie

[1] Wikipédia. Natixis — wikipédia, l'encyclopédie libre. http://fr.wikipedia.org/w/index.php?title=Natixis&oldid=150673472, 2018. Consulté le 27-juillet-2018.

[2] Natixis. https://www.natixis.com/natixis/jcms/ala_5361/fr/profil. Consulté le 27-juillet-2018.

[3] Alek Sharma. A brief history of devops. https://circleci.com/blog/a-brief-history-of-devops-part-i-waterfall/, 2018. Consulté le 27-juillet-2018.

[4] Grady Booch. *Object Oriented Design with Applications*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1991.

[5] Kent Beck and Cynthia Andres. *Extreme Programming Explained : Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[6] Martin Fowler. Continuous integration. https://www.martinfowler.com/articles/continuousIntegration.html, 2006. Consulté le 27-juillet-2018.

[7] Sam Guckenheimer. What is continuous integration? https://www.visualstudio.com/learn/what-is-continuous-integration/, 2017. Consulté le 27-juillet-2018.

[8] Perforce. Continuous delivery : The new normal for software development. https://www.perforce.com/sites/default/files/files/2017-07/continuous-delivery-migration.pdf, 2017. Consulté le 27-juillet-2018.

[9] Ambysoft. Agile practices and principles survey. http://www.ambysoft.com/downloads/surveys/PracticesPrinciples2008.pdf, 2008. Consulté le 27-juillet-2018.

[10] Brett Slatkin. Continuous delivery at google. https://air.mozilla.org/continuous-delivery-at-google/, 2013. Consulté le 27-juillet-2018.

[11] Ben Schmaus. Deploying the netflix api. https://medium.com/netflix-techblog/deploying-the-netflix-api-79b6176cc3f0, 2013. Consulté le 27-juillet-2018.

[12] H. H. Olsson, H. Alahyari, and J. Bosch. Climbing the "stairway to heaven" – a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 392–399, 2012.

[13] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment : A systematic review on approaches, tools, challenges and practices. *CoRR*, abs/1703.07019, 2017.

[14] Meet jenkins. https://wiki.jenkins.io/display/JENKINS/Meet+Jenkins. Consulté le 27-juillet-2018.

[15] Best continuous integration software. https://www.g2crowd.com/categories/continuous-integration. Consulté le 27-juillet-2018.

[16] Hudson's future. https://jenkins.io/blog/2011/01/11/hudsons-future/. Consulté le 27-juillet-2018.

[17] plaindocs et al. Core concepts for beginners. https://docs.travis-ci.com/user/for-beginners/, 2015. Consulté le 27-juillet-2018.

[18] DMAHUGH. http://mahugh.com/2016/09/02/travis-ci-for-test-automation/, 2016.

[19] Buildbot : the continuous integration framework. https://buildbot.net/. Consulté le 27-juillet-2018.

[20] http://docs.buildbot.net/0.8.3/System-Architecture.html.

[21] Pavel Sher et Julia Alexandrova. Continuous integration with teamcity. https://confluence.jetbrains.com/display/TCD10/Continuous+Integration+with+TeamCity, 2015. Consulté le 27-juillet-2018.

[22] https://docs.stiltsoft.com/display/public/JTC/Viewing+TeamCity+Builds.

[23] John Ferguson Smart. *Jenkins : The Definitive Guide*. O'Reilly Media, Inc., 2011.

[24] Niall O'Higgins. 3 hard lessons from scaling continuous deployment to a monolith with 70+ engineers. https://bit.ly/2mLJ7L9, 2017. Consulté le 27-juillet-2018.

[25] Niall O'Higgins. Revamping continuous integration and delivery at conductor. https://www.conductor.com/nightlight/revamping-continuous-integration-delivery-conductor/. Consulté le 27-juillet-2018.

[26] ITinfo. Software development methodologies. http://www.itinfo.am/eng/software-development-methodologies/. Consulté le 27-juillet-2018.

[27] Agile Alliance. What is agile software development? https://www.agilealliance.org/agile101/, 2018. Consulté le 27-juillet-2018.

[28] Jim Highsmith. History : The agile manifesto. http://agilemanifesto.org/history.html, 2001. Consulté le 27-juillet-2018.

[29] J. Highsmith and A. Cockburn. Agile software development : the business of innovation. *Computer*, 34(9) :120–127, Sep 2001.

[30] Dan Radigan. How the kanban methodology applies to software development. https://www.atlassian.com/agile/kanban. Consulté le 27-juillet-2018.

[31] Hirotaka Takeuchi et Ikujiro Nonaka. The new new product development game. https://hbr.org/1986/01/the-new-new-product-development-game, 1986. Consulté le 27-juillet-2018.

[32] Scrum. https://www.scrum.org/. Consulté le 27-juillet-2018.

[33] Oleksandr Mandryk. The agile : Scrum framework at a glance. https://icoderman.wordpress.com/2016/02/23/the-agile-scrum-framework-at-a-glance/.

[34] Andrew Powell-Morse. Rapid application development (rad) : What is it and how do you use it? https://airbrake.io/blog/sdlc/rapid-application-development, 2016. Consulté le 27-juillet-2018.

[35] Karenjoy toletol. https://en.wikipedia.org/wiki/Rapid_application_development, 2001.

[36] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. Challenges when adopting continuous integration : A case study. In Andreas Jedlitschka, Pasi Kuvaja, Marco Kuhrmann, Tomi Männistö, Jürgen Münch, and Mikko Raatikainen, editors, *Product-Focused Software Process Improvement*, pages 17–32, Cham, 2014. Springer International Publishing.