Rapport d’audit
Journée du 26 Février 2024

Table des matières

Parcs applicatifs / Types d’application

Applications à destination des agents BCEAO (8 pays), quelques unes ouvertes sur l’extérieur en particulier les banques

- **Applications Web monolithique**, construites au dessus d’un socle technique maison, déployées sous forme de *war* sur des serveurs Tomcat
- **Jobs schédulés** par cron construit au dessus de *Talend/Mule*
  - Synchronisation entre applications (BD vers BD)
  - Importation de données externes
- **Application mobile** en cours de définition et développement

Parc applicatif / Socle technique

Construit au dessus des framework **Spring4** et **ExtJs** et d'une **BD** relationnelle (Oracle, MySQL, Postgresql)

Langage **Java** : 1.7, 1.8, 11

2 versions du socle

Parc applicatif / Apports du socle

- **Composant d'authentification** compatible BD, LDAP et Keycloak
  Keycloak est en cours d'intégration dans l'intégralité des applications
- **Modèle d'autorisation RBAC**
  - Approbations des rôles utilisateurs via RAFIA
  - Synchronisation régulière des applications avec RAFIA pour mettre à jour et adapter leur propre modèle
- **Classes du domaine partagé** (Établissements, agents)
- Support pour le **reporting** via la librairie Jasper
- Support pour le ***workflow des dossiers*** via la définition d'une simple machine à état
- **Génération d'interface** ExtJs, intégrant la gestion des batchs

Parc applicatif / Retours sur le socle

- Pour certains projet, **nécessité de surcharger** le modèle de domaine
- Pour certains projet, certaines fonctionnalités sont **inutilisées**, le modèle de domaine n’est pas repris dans son intégralité
- **Difficile de faire évoluer** le socle car cela touche tous les projets en production

=> A l’heure actuelle, l’utilisation de ce socle est en grand frein à la migration vers des technologies plus modernes

Organisation des équipes / Les pôles

Le service est divisé en 5 pôles :

- Statistiques
- SIB (Système d’informations bancaires)
- Comptabilité et administration
- RH
- DevOps : Équipe transverse qui s’occupe de l’environnement d’intégration et des déploiements

Les membres de chaque pôle sont des profils développeurs sans plus de distinction

Organisation des équipes / Portefeuille d’applications

Chaque pôle, autre que DevOps, est responsable d’un portefeuille d’applications.

En général, un **seul développeur** est responsable et a la connaissance de une ou plusieurs applications !!

Pour chaque application, il y a également 1 responsable métier, 1 responsable admin pour la mise en production.

Les développeurs sont évalués avec un indicateur de performance

Organisation des équipes / Retours développeur

- Le travail de développeur est souvent **interrompu** par des urgences :
  - Correction de bug
  - Spécifications fonctionnelles peu précises (Retours sur les développements)
  - Décisions politiques urgentes

=> Difficile de rester concentrer sur de longues tâches comme des grosses évolutions ou des migrations de socle technique

- La connaissance d'une application est **peu partagée**.
  - L'absence d'un développeur peut générer des retards car lui seul connaît les applications dont il a la charge
  - Peu de documentation fonctionnelle.
  - Le cahier de charges initial évolue et les évolutions ne sont pas tracées

Conduite de Projet / Démarrage

- Des réunions d’échange avec le métier permettent de définir le cahier de charge initial au format bureautique
- Pas ou peu de documentation technique qui pourrait inclure :
  - Architecture choisie
  - Modèle de données
  - User Story et Tests d’acceptation

Conduite de Projet / Méthodologie Agile

- Aucune méthodologie n’est formalisée
  - Scrum, Backlog, Sprint, Jalons
- Aucune cérémonie n’est formalisée :
  - Stand Up Meetings
  - Review de Sprint
- Aucun outil n’est utilisé
  - Juste outils bureautique Ad Hoc non formalisé

Conduite de projet / Dépôt des sources

- Les sources sont publiés dans des dépôts Git gérés par Git EA mais les fonctionnalités de collaboration de l'outil sont inutilisées
  - L'équipe a une bonne pratique de *git* et est capable de faire des opérations avancées (merge, rebase, cherry-pick)

- Aucun workflow de collaboration n'est défini autour de notions comme les Merge Request ou Pull Request.
  - Chaque projet a une utilisation des branches Git différentes

Environnement de développement

- Principalement 2 IDEs utilisés :
  - Eclipse
  - IntelliJ, version ultimate
- Plugins de pair-programming
  - Copilot en cours d’acquisition
  - SonarLint abandonné car trop de dette qualité
- Services de support (Base de données)
  - Installation locale manuelle par chaque développeur
    => Risque de disparité entre les postes

Usine logicielle / Périmètre et environnements

- La responsabilité de l’équipe s’arrête à l’environnement d’intégration (décliné en 2 environnements dév et test)
- Les environnements Dev et Test sont gérés par le pôle DevOps
  - Composés principalement de serveur Tomcat pré-paramétrés qui hébergent plusieurs applicatifs et de bases de données
- Les déploiements sont effectués par des scripts démarrés manuellement

Usine logicielle / Cycle de vie

- Chaque évolution suit le cycle suivant :
  - Le code est développé localement
  - Il est ensuite déployé sur un environnement de validation/intégration provisionné par l'équipe DevOps.
  - Des tests de validation sont effectués manuellement par le développeur
  - Un script démarré via Jenkins publie l'artefact dans le dépôt Archiva. Un version SNAPSHOT s'ajoute aux précédents artefacts
  - L'équipe Infra récupère le dernier SNAPSHOT et le déploie sur un environnement de recette accédé et validé par les utilisateurs finaux

Usine Logicielle / Remarques

- Pas d’approche continue : Les jobs ne sont pas démarrés automatiquement à chaque commit
- Aucun test automatisé :
  - Pas de test unitaires ou d’intégration

    => Peu de confiance dans le code produit
  - Pas de formalisation des tests d’acceptation
    => Recette floue
- Pas d’approche qualité : Pas d’analyse statique ni de revue de code

Usine Logicielle / Remarques

- Serveur mutable : Les différents déploiements transforment les environnements.
  => On ne connaît plus très bien l'état des serveurd l'intégration, de la recette et de la production
- Pas de processus de release identifié.
  Mauvaise identification des versions déployées. Mauvaise utilisation des versions SNAPSHOTS
- Interdépendance des applications
  - Problèmes lors de déploiement
  - Problème en production pour le partage des ressources. Un bug dans une appli peut avoir des conséquences sur d'autres

Usine logicielle / Configuration applicative

- La configuration applicative est externalisée dans des fichiers *properties* fournis par les développeurs
- Les fichiers sont ensuite adaptés à l'environnement
  - Par l'équipe DevOps pour l'environnement d'intégration
  - Par l'équipe d'infra pour les environnements de recette et production.

- En contradiction avec un principe DevOps :
  Le dépôt est l'unique source de vérité
  Dans l'absolu, les configurations de recette et de production devraient être dans le même dépôt que les sources

Production / Observabilité

- L’équipe n’a aucune vision de l’environnement de production.
- Aucun métriques (utilisation / performance …) ne lui est remonté
- En cas d’incident, les fichiers de traces leur sont fournies pour analyse
  - L’outil EasyVista est utilisé pour la gestion d’incidents
  - Les bugs fonctionnels ou demande d’évolution sont souvent remonté par de simple mails.

Recommandations

Augmenter la qualité

- Beaucoup de bugs relevés ou d’incidents en prod (application en rade, BD surchargée, …), comment diminuer ce type d’incidents ?
  - Augmenter les tests
  - Instaurer des processus de vérification de la qualité
  - Avoir des environnements d’intégration plus proche de la production. (Volumétrie et modèle de charge)

Augment la qualité / Les tests

- L’équipe de développement doit adopter l’approche TDD. Et être convaincu de l’apport des tests :
  - Ils n’apportent pas plus de charge de travail
  - Cela leur permet d’avoir confiance dans le code produit
  - Cela leur permet d’approfondir l’analyse de la problématique
- Les tests d’acceptation doivent être formalisées avec le métier
  - Production de User Story et de scénario d’usage validés par le métier
- Les tests doivent être automatisés et donc démarrés à chaque commit dans le référentiel Git

Augmenter la qualité / La revue de code

- La revue de code est un des moyens pour augmenter la qualité. Sur chaque applicatif, au minimum 2 personnes doivent être affectés
  - 1 développeur qui produit les évolutions
  - 1 mainteneur de projet responsable de la branche principale et des release. Il revoit le code et s'assure de sa qualité technique et de sa justesse fonctionnelle
- La revue de code favorisera également le partage de connaissance

Augmenter la qualité / Analyse statique

- En compléments des tests, une analyse statique doit être systématiquement effectuée. Les résultats de l'analyse doivent être revus conjointement avec le développeur et le mainteneur.
- Les métriques suivis sont :
  - La couverture des tests
  - Les bugs (code dangereux affectant la fiabilité)
  - Les vulnérabilités (code dangereux affectant la sécurité)
  - Les code smells (pratique diminuant la maintenabilité du code)
- Ces indicateurs doivent progressivement s'améliorer, le code existant n'étant pas pris en compte.

Augmenter la qualité / Dev Prod Parity

- Les tests actuellement effectués ne reflètent pas la situation en production, en termes de
  - Volumétrie de données (Base de données)
  - Charge utilisateur
  - Concurrence des applications
- L’équipe DevOps devrait fournir une environnement au plus proche des conditions de production
- Des scripts de simulation de charge devrait être mis au point et appliqués sur les environnements de production

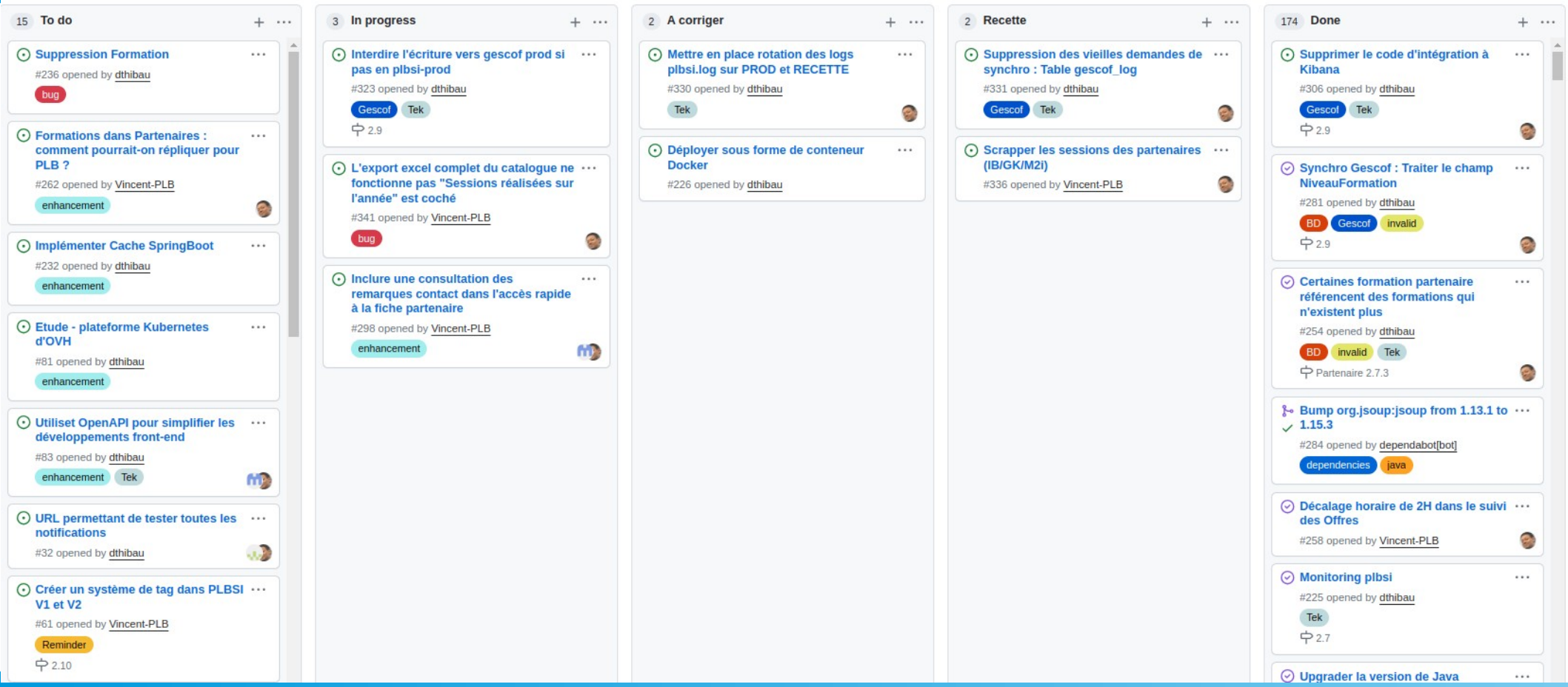
Augmenter la qualité / Outils

- Tests unitaires : Les développeurs devraient avoir une meilleure connaissance des librairies de tests comme JUnit5, spring-test, Mockito, AssertJ, Hamcrest, JsonPath, …
- Tests d'acceptation : Démarrer avec de simples tests formalisés dans des documents bureautiques pour essayer qu'un jour on puisse automatiser avec des outils comme Cucumber
- Revue de code : Appliqué un workflow de MergeRequest ou PullRequest qui sont joués dans des plateforme DevOps comme Gitlab CI
- Analyse statique : Sonarqube, Code Climate intégré à GitlabCI

Augmenter la collaboration

- La collaboration concerne tous les acteurs :
  - Mainteneur / Développeur
  - Métier
  - Infra
- Un outil est nécessaire pour tracer les échanges, planifier le travail, visualiser les travaux en cours, gérer le backlog
- Définir un workflows de collaboration de standard utilisé par tous les projets. Exemple : le Merge Request
- Outil largement utilisé en entreprise Gitlab

Augmenter la collaboration / Exemple



Augmenter la collaboration / Les cérémonies

- Les cérémonies peuvent être un bon moyen pour augmenter le partage de connaissance entre les équipes.
  - **Stand-up meetings** : Par exemple à l'échelle d'un pôle, réunion informelle quotidienne sur ce que l'on a fait hier et ce que l'on va faire aujourd'hui
  - **Revue de sprint** : Démonstration devant les partenaires projets des évolutions fonctionnelles
  - **Présentation technique** : Un sujet technique est présenté par un collaborateurs à toute l'équipe

Augmenter la collaboration / La documentation

- Formaliser les documents nécessaire au démarrage d’un projet
- Tracer les évolutions fonctionnelles via des Release Note.
  - La production de Release inclut ces Release Notes

Adopter l’approche continue

- Les étapes pour progresser dans l’approche continue :
  - Choisir l’outil Jenkins ou Gitlab CI ou autre
  - Définir un script pipeline Jenkinsfile ou .gitlab-ci.yml adossé au dépôt de source
  - Intégrer progressivement les étapes
    - Packaging et tests unitaires, publication des résultats de tests
    - Analyse de code
    - Analyse des dépendances et détection de vulnérabilités
    - Publication de SNAPSHOTs dans Archiva
    - Déploiement automatique dans l’environnement de test
    - Release : contrôle de version et Release Notes

Approche continue / La release

- Les n° de versions ne sont à ce jour pas contrôlés
- Inclure dans la pipeline une phase manuelle permettant l'automatisation de la release :
  - Fixer le n° de version dans le fichier pom.xml ou build.gradle
  - Effectuer le packaging et les tests automatisés
  - Committer et tagger le dépôt de source
  - Publier l'artefact et les release notes dans le dépôt Archiva avec le n° de version
  - Incrémenter le n° de version pour la prochaine évolution

Casser les dépendances

- Actuellement, les applications ont trop de dépendances entre elles :
  - Dépendance sur le modèle de données apportées par le socle
  - Partage des ressources serveurs de Tomcat (CPU, mémoire, pool de connexions BD)
- Ces dépendances ont de lourds impacts :
  - Nécessité de process batch de synchronisation
  - Difficulté d’évolution des technologies
  - Complexité du modèle de domaine de chaque application

Casser les dépendances / les pistes

- Progressivement, évoluer vers une architecture micro-service
  - Identifier les fonctions métier de la BCEAO et définir leurs APIs
  - Les intégrer progressivement dans les évolutions ou nouveau développement
- Se doter d’un bus d’évènement temps-réel comme Kafka
- Implémenter les ACLs avec oAuth2. Définir les rôles métier dans Keycloak à partir du modèle RAFIA existant

Casse les dépendances / l'infra

- Le déploiement des applications sur un serveur partagé Tomcat présente des désavantages
  - La version du serveur impose des restrictions sur les choix techniques. Pas de modèle réactif, version de Java imposé
  - Les déploiements ne sont pas immuables
  - Une application ne sont pas isolées et une application en erreur peut avoir des impacts sur les autres
- Pistes :
  - Dans un premier temps, migrer vers un packaging Jar embarquant le serveur
  - A terme, utiliser les orchestrateur de conteneur

Augmenter l’observabilité

- Fournir des endpoints de surveillance dans les applications déployées.
  Typiquement starter actuator dans SpringBoot
- Adopter une solution de centralisation des logs. Donner un accès aux équipes de développement
  Par exemple ELK
- Se doter d’une infrastructure de visualisation et de création d’alerte.
  Exemple : Prometheus / Grafana
- Implémenter des métriques métier dans les applications permettant une meilleure compréhension de l’usage