AUDIT APPLICATIF

L'analyse étape par étape de votre application métier

axiocode
DÉVELOPPEMENT WEB & MOBILE

1ère édition

Les étapes et les outils pour auditer vos applications professionnelles.

www.axiocode.com

axiocode

AUDIT APPLICATIF : POINT DE DÉPART DE LA MODERNISATION

Si les derniers mois nous ont clairement montré quelque chose, c'est que les entreprises de tous les secteurs doivent **être agiles et capables de changer rapidement**. Nouveaux besoins des clients, nouvelles idées, nouvel environnement incluant des confinements et des chaînes d'approvisionnement perturbées. Les entreprises doivent être capables de s'adapter rapidement aux changements, au risque de perdre en compétitivité.

Cette adaptabilité s'étend également aux **applications web et mobiles utilisées par les entreprises** !

Trop souvent, les entreprises déploient beaucoup d'efforts pour s'en tenir au portfolio applicatif existant alors que le **changement devient nécessaire**. Ce faisant, elles échangent les coûts initiaux de changement de plateforme des applications contre une dette technique à long terme. Cette dette se traduit par :

- *des coûts d'exploitation plus élevés*
- *un développement logiciel moins agile*
- *une complexité accrue des applications*
- *Sacrifier la résilience et l'évolutivité des applications*

En se concentrant sur la **modernisation** de ses applications, une entreprise est plus à même de créer une plateforme qui évolue efficacement pour répondre spécifiquement à ses besoins. Cela peut se traduire par des **économies** substantielles, une plus grande **flexibilité** dans le processus métier ou une capacité globale à **accélérer l'innovation.**

La modernisation des applications désigne le **processus d'amélioration** des systèmes existants afin de répondre à l'évolution des besoins de l'entreprise. Au lieu d'abandonner les applications métiers vieillissantes que vous utilisez quotidiennement, vous pouvez tirer parti de leur base et les **réadapter** aux besoins de votre entreprise.

Tout n'est pas à jeter dans votre système actuel ! Vous devez être en mesure d'identifier **ce qui va, ce qui ne va pas et ce qu'il faut ajouter** à vos applications actuelles. C'est le rôle de l'audit applicatif. Pour optimiser vos performances et celles de votre entreprise, il faut nécessairement passer en revue l'existant. Si vous utilisez des systèmes obsolètes, vous constaterez que les nouveaux programmes ne s'intègrent pas correctement, voire pas du tout.

Dans un premier temps, il vous faudra réaliser **l'audit de votre portefeuille d'applications**. Cette étape vous permettra de voir où les besoins sont satisfaits et où ils peuvent être améliorés. Vous aurez alors plus de visibilité pour décider quelles applications sont les meilleures candidates à la modernisation et ce qui sera le plus judicieux pour votre entreprise.

Dans ce Livre Blanc, AxioCode vous accompagne **étape par étape** dans l'audit de vos applications métiers.

L'équipe AxioCode.

SOMMAIRE

ANALYSER LA COMPLETUDE FONCTIONNELLE

La première étape d’un audit applicatif est de répondre à la question suivante :

Quels problèmes rencontrez-vous avec vos applications professionnelles et quelles sont leurs étendues ?

Qu'elles soient web ou mobiles, les applications métiers que vous utilisez peuvent parfois poser problème, **ralentissant votre activité ou votre productivité** par exemple. Dans la plupart des cas, des indices signalent que votre application web ou mobile est **obsolète** ou sur le point de le devenir :

- *Les utilisateurs se plaignent-ils de l’application en question ?*
- *Vous êtes conscient de l’urgence à faire évoluer vos applications*
- *Les technologies sont-elles problématiques ?*
- *Les tests automatisés sont-ils absents ?*
- *Des problèmes importants de conception ?*
- *Des compétences techniques manquantes pour faire évoluer votre application ?*

Pour savoir ce qu’il en est précisément, la meilleure solution consiste à faire un état des lieux complet.

QU’EST CE QU’UNE FONCTIONNALITÉ MÉTIER ?

Une fonctionnalité métier ou exigence fonctionnelle répond à un besoin précis de votre entreprise comme par exemple :

- *Créer un devis pour un client*
- *Planifier un rendez-vous*
- *Comparer des produits dans votre catalogue*

Les services rendus par les applications web et mobiles que vous utilisez ne sont pas tous essentiels au bon fonctionnement de votre entreprise, c’est pourquoi il est important d’identifier les fonctionnalités clés sans lesquelles votre activité est fortement impactée voir arrêtée. Il vous appartient donc de **définir et d’évaluer ces fonctionnalités en premier**.

Afin de vous aider à identifier les fonctionnalités clés dont vous avez besoin pour vos applications métiers, voici les informations que vous pouvez utiliser pour faire l’inventaire des fonctionnalités clés de votre application :

- **L’expression de besoin initiale** indique les objectifs de l’application et les principales fonctionnalités de cette dernière

- Pour aller plus loin, n’hésitez pas à télécharger notre modèle de **cahier des charges** il vous permettra de **vérifier que vous n’avez rien oublié** d’important. Ce sera la même chose avec la documentation utilisateur de l’application

- Il est fréquent que les **documents initiaux** ne soient pas mis à jour. Pour vérifier s'il y a des modifications de fonctionnalités, rien de plus simple : **utilisez, inspectez et explorez l’application avec tous les profils utilisateurs**

- Appuyez-vous également sur les **demandes d'évolutions** faites par les utilisateurs de l'application métier. Cela vous permettra d' identifier les fonctionnalités métiers clés de l'application qui n'étaient pas implémentées au départ. Ces demandes sont souvent référencées dans un gestionnaire de tickets (par exemple Gitlab)

- Utilisez la **liste des tâches**. Elle peut provenir, par exemple, d'un outil de gestion de projet. Cela vous évitera d'**oublier des fonctionnalités métiers clés** dans l'inventaire

- **Réalisez des interviews avec des utilisateurs** identifiés, compétents et maîtrisant l'application. Attention, l'exercice risque d'être trompeur. Les utilisateurs ne parlent généralement que des fonctionnalités qu'ils utilisent, et plus encore, de celles qui leur posent problème. Il est possible qu'ils manquent d'une vue globale de l'application

Ce travail préliminaire ressemble généralement à un jeu de piste ! Idéalement, il doit être réalisé par le chef de projet de l'application. Il sera d'autant plus efficace et rapide.

Prenez le temps d'évaluer toutes vos fonctionnalités métiers de façon objective. Vous pouvez **associer une note à chaque critère**. Cela vous permettra d'obtenir un score pour chaque fonctionnalité.

ÉVALUER LES FONCTIONNALITÉS MÉTIERS

L'audit d'une application web ou mobile comprend une phase d'**analyse fonctionnelle**. Elle sert à vérifier si l'application remplit son rôle et répond aux fonctionnalités dont vous avez besoin.

Évaluer ces critères vous aidera à **mesurer l'obsolescence de votre application** d'un point de vue fonctionnel. Un audit applicatif efficace s'appuie sur l'**analyse détaillée et poussée des fonctions** dont vous avez besoin dans le cadre de votre activité.

L'analyse fonctionnelle de votre application comporte deux objectifs essentiels :

- **Déterminer si les fonctionnalités métiers clés** sont complètes en fonction de vos besoins métiers. Cela permet de garantir la **pérennité de l'application**.

- Vérifier si l'**équipe de développement est en mesure d'apporter les évolutions** demandées pour garantir la meilleure réponse aux utilisateurs.

Lorsqu'une application existante pose problème, l'audit applicatif permet de **débloquer des situations parfois complexes** et laborieuses. Il permet à l'entreprise de reprendre le contrôle sur l'application et ses investissements.

AxioCode vous propose une **méthode efficace** pour aborder sereinement les évolutions de vos outils digitaux. Vous trouverez, dans cette première partie, nos conseils et astuces pour identifier les fonctionnalités métiers clés de votre entreprise.

Il est nécessaire pour pérenniser votre application web ou mobile de connaître l'ensemble des fonctionnalités métiers qu'elles soient déjà implémentées ou manquantes.

Afin de vous aider à évaluer les fonctionnalités de vos applications métiers nous vous préconisons d'utiliser les **six critères** suivants :

6 critères pour évaluer vos fonctionnalités

Critère 1	Critère 2	Critère 3	Critère 4	Critère 5	Critère 6
Importance fonctionnelle	**Complexité fonctionnelle**	**Spécification fonctionnelle**	**Couverture des tests**	**Documentation utilisateur**	**Complétude fonctionnelle**
Très haute	Très difficile	Oui	Oui	Oui	Conforme
Haute	Difficile	Non	Non	Non	Mise à jour nécessaire
Moyenne	Moyenne	Mise à jour nécessaire	Mise à jour nécessaire	Mise à jour nécessaire	Non conforme
Basse	Faible				Inutile
Très basse	Très faible				

N'hésitez pas à vous reporter à l'exemple fourni dans le fichier Excel que nous vous proposons !

- **L’importance de la fonctionnalité**

Une fois les fonctionnalités métiers clés identifiées vous devrez **définir leur importance**. L’importance ou la criticité permet de définir **l’impact qu’aurait l’absence** ou l’arrêt d’une fonctionnalité sur votre activité.

Échelle de criticité : *très haute, haute, moyenne, basse, très basse.*

- **La complexité de la fonctionnalité**

Cette complexité est aussi à juger du **point de vue du métier**.

De la même manière, vous pouvez utiliser l’**échelle de notation** suivante :
très difficile, difficile, moyen, faible, très faible.

- **La présence de spécifications fonctionnelles claires et précises**

La fonctionnalité est-elle clairement décrite dans des **spécifications fonctionnelles** de votre application métier ? Cette description respecte t-elle les bonnes pratiques d’écriture d’une exigence fonctionnelle ?

La notation : *oui / non ou une mise à jour est nécessaire.*

- **La couverture des tests**

La fonctionnalité est-elle couverte par au moins un cas de test ?

Un cas de test, aussi appelé test d’acceptance, décrit la **manière dont la fonctionnalité devra être testée pour être acceptée, validée**.

Un cas de test est défini par une séquence d’étapes de test. Il faut alors préciser l’action effectuée et le résultat attendu. Si l’ensemble des résultats attendus est correct alors le cas de test est correct.
L’ensemble des tests d’acceptance doivent être consignés dans un plan de test.

Vous pouvez utiliser **la notation** : *oui / non ou une mise à jour est nécessaire.*

- **La documentation**

La fonctionnalité est-elle décrite dans la documentation utilisateur ?

Nous pouvons là encore utiliser **la notation** : *oui / non ou une mise à jour est nécessaire.*

- **Le plus important : la conformité fonctionnelle**

Cette fonctionnalité **rend-elle le service attendu** ?

Les **quatre valeurs possibles** s’approchent d’une conformité ou non :
conforme, mise à jour nécessaire, non conforme, inutile.

Vous trouverez dans le tableau ci-dessous un exemple de **classement des fonctionnalités** d’une application web ou mobile.

Classement des fonctionnalités

ID/Code	Fonctionnalité métier	Importance	Complexité	Spécifiée	Couverture des tests	Documentation	Conformité
U001	Rechercher un hôtel	Haute	Difficile	Mise à jour nécessaire	Mise à jour nécessaire	Mise à jour nécessaire	Conforme
U002	Visualiser les hôtels trouvés	Haute	Moyenne	Spécifiée	Mise à jour nécessaire	Documentée	Mise à jour nécessaire
U003	Comparer des hôtels	Moyenne	Difficile	Spécifiée	Mise à jour nécessaire	Non documentée	Conforme
U004	Obtenir des informations sur un hôtel	Haute	Moyenne	Mise à jour nécessaire	Non couverte par les tests	Mise à jour nécessaire	Conforme
U005	Localiser des hôtels sur une carte	Très haute	Moyenne	Mise à jour nécessaire	Non couverte par les tests	Documentée	Conforme
U006	Effectuer une réservation	Très haute	Très difficile	Spécifiée	Mise à jour nécessaire	Documentée	Mise à jour nécessaire
U007	Payer une réservation	Très haute	Difficile	Mise à jour nécessaire	Mise à jour nécessaire	Mise à jour nécessaire	Conforme
U008	Recevoir des notifications	Moyenne	Faible	Mise à jour nécessaire	Non couverte par les tests	Documentée	Conforme
U009	Partager son expérience	Moyenne	Faible	Non spécifiée	Non couverte par les tests	Non documentée	Conforme
U010	Evaluer un hôtel	Haute	Faible	Mise à jour nécessaire	Non couverte par les tests	Documentée	Mise à jour nécessaire
U011	Annuler une réservation	Moyenne	Moyenne	Mise à jour nécessaire	Non couverte par les tests	Mise à jour nécessaire	Conforme
U012	Créer un compte premium	Haute	Faible	Mise à jour nécessaire	Non couverte par les tests	Documentée	Conforme
U013	Suspendre ses avantages membre	Haute	Très faible	Mise à jour nécessaire	Non couverte par les tests	Documentée	Non conforme

Chacune d’entre elles est listée en indiquant son **importance**, sa **complexité**, si elle est spécifiée dans la documentation fonctionnelle, si elle est couverte par un ou des **cas de test**, si d'un point de vue utilisateur elle est **documentée** etc.

ÉVALUER L'ÉVOLUTIVITÉ DE VOTRE APPLICATION

Le second objectif de l'audit fonctionnel d'une application web ou mobile vise à déterminer si les **demandes d'évolution** que vous souhaitez apporter sont maîtrisées par l'équipe de développement. Le but est de **garantir la meilleure réponse possible** aux utilisateurs face aux changements demandés.

Il arrive que l'application ait évolué. Les demandes d'évolution ont lieu en général en cas de **dysfonctionnement** ou de **changement fonctionnel**. Pour les besoins de l'audit, la source d'information sera généralement un **outil de ticketing**/support/bugtracker. Par exemple **Github**, cet outil de ticketing sert à réaliser une demande auprès du développeur qui fera les modifications nécessaires. On peut aussi utiliser d'autres outils comme un tableur excel qui peut permettre de remonter les demandes.
Il faut établir la liste des demandes d'évolution (non clôturées). Elle comporte les informations suivantes :

- L'**intitulé de la demande**
- La **fonctionnalité** métier associée
- La **date** de la demande
- Le **type de la demande** : *nouveauté, correction, amélioration*
- **L'état** de la demande : *ouverte, acceptée, refusée, validée, en cours, résolue*
- La **date de prise en charge** de la demande par l'équipe de développement
- Et enfin la **date de résolution** de la demande

Utilisez un tableur collaboratif pour référencer les demandes, il vous permettra ensuite de faire des calculs de score.

exemple - fichier Excel de suivi des demandes d'évolution

ID/Code	Intitulé de la demande	Fonctionnalité métier	Date demande	Type	Etat	Priorité	Date prise en charge	Date résolution
324	Remplacer le titre des hôtels trouvés	Visualiser les hôtels trouvés	24/03/2021	Correction	Ouverte	Moyenne		
336	Photos en double	Obtenir des informations sur un hôtel	26/03/2021	Correction	Acceptée	Très haute	05/04/2021	
342	Ajouter le paiement via virement	Payer une réservation	31/03/2021	Nouveauté	Ouverte			

Comme pour les fonctionnalités, il faut définir les critères d'évaluation des demandes d'évolution.

Par exemple, posez-vous des questions sur :

- **Le type de demande**
  S'agit-il d'une nouveauté, d'une amélioration ou d'une correction ?

Si les **demandes de correction sont nombreuses**, cela signifie que l'application a un réel **problème**.
Par contre, si les demandes sont des demandes d'évolution, c'est tout à fait normal.

Méthode de notation : *Associer une note à chaque critère vous permettra d'obtenir un score pour chaque demande d'évolution.*

- **La priorité de la demande**

Si vous avez des priorités très hautes, cela traduit un **problème d'obsolescence** important. A l'inverse, si les demandes ont un niveau de priorité bas, il n'y a aucun souci.
.
Échelle de priorité : *très haute, haute, moyenne, basse, très basse.*

- **Le délai de prise en charge**
  C'est une **évaluation sur la capacité de l'équipe de développement** à traiter vos demandes.

Si vous avez des demandes qui s'accumulent et que l'application n'évolue pas, elle deviendra obsolète. En revanche, si le délai est **rapide**, c'est **bon signe pour la santé de l'application**.

Suivre la méthode en fonction des dates saisies dans la liste des demandes d'évolution, vous pouvez définir que le **délai de prise en charge a été** : *court (moins de 4 jours par exemple), moyen (de 5 à 9 jours), long (de 10 à 15 jours) ou très long (plus de 15 jours).*

Une fois toutes les fonctionnalités et les demandes d’évolution répertoriées et évaluées, il vous sera possible de produire des **indicateurs et des métriques** pour mettre en perspective votre audit fonctionnel.

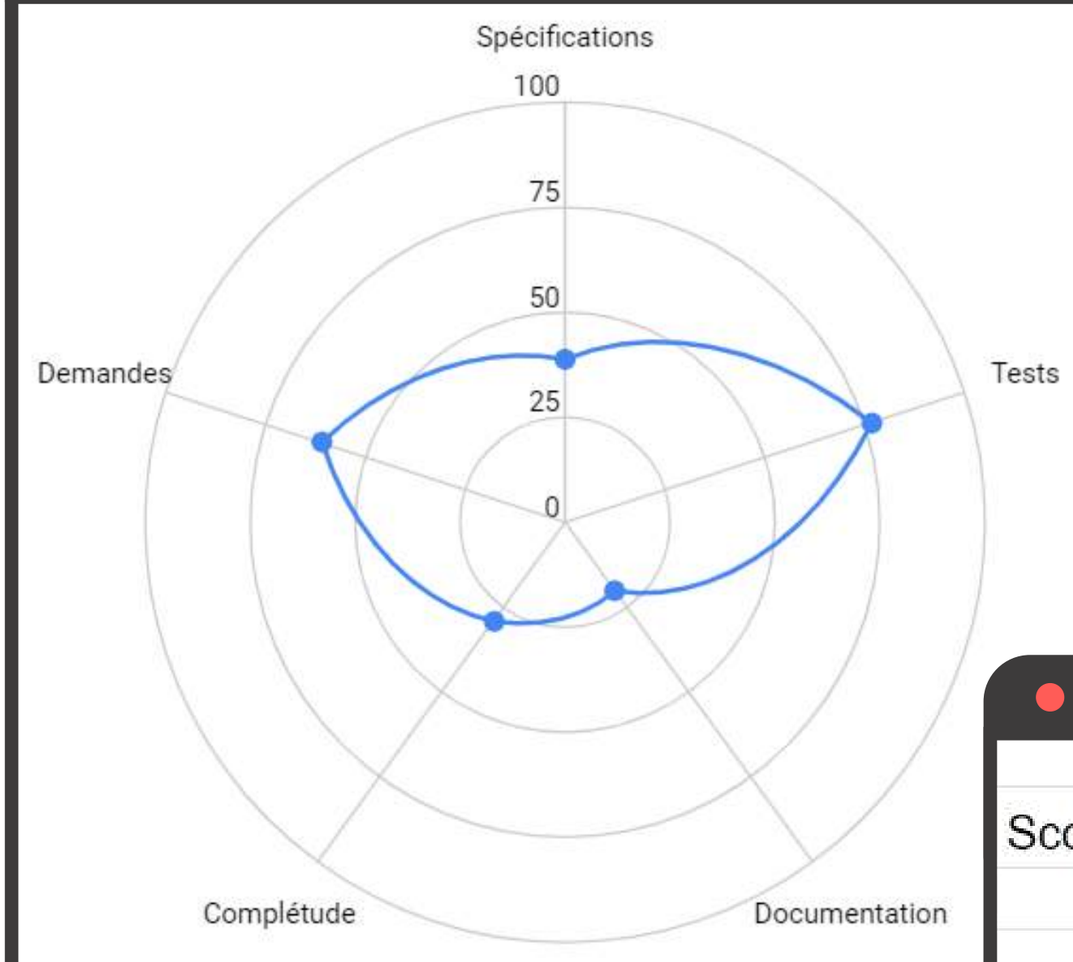
Dotez-vous d'**outils de pilotage** (indicateurs de performance, tableaux de bord), cela facilitera la **prise de décision** opérationnelle.

Nous vous présentons ci-dessous un **exemple d’indicateurs globaux** calculés à partir des différents critères. Des points sont calculés en utilisant des coefficients selon certains critères. L’importance et la complexité des fonctionnalités par exemple.

En rapportant ces points sur la base 100, on obtient un **score**. Le **diagramme** permet d’observer clairement **où les problèmes se situent**.

Dans l’exemple ci-dessous, nous observons un problème autour des tests et du traitement des demandes d’évolution de l’application.

Graphique indicateurs globaux



Indicateurs globaux

	Points	Score
Score global	8174,375	**31**
Spécifications	176,875	39
Tests	346,25	77
Documentation	54,25	20
Complétude	262	29
Demandes	525	61

Ces étapes permettent de dresser un audit fonctionnel détaillé dévoilant les forces et faiblesses de votre application métier. Elles ont aussi pour but de vous aider à définir les priorités sur lesquelles se concentrer pour transformer efficacement les faiblesses de votre application web ou mobile en forces.

REALISER L'INVENTAIRE TECHNOLOGIQUE

Après l'étude des fonctions de votre application web ou mobile, il est temps de passer à la seconde étape : l'**inventaire technologique**.

Un extrait du rapport de Synopsys : Open Source Security and Risk Analysis, 2020 indique que le code source de nombreux composants techniques utilisés dans les applications métier possède des failles de sécurité connues.

"Le code source de 9 applications métier sur 10 contient des composants open source obsolètes ou laissés à l'abandon. 75% du code source contient des composants qui présentent des failles de sécurité connues et 49% de ce code source intègre des vulnérabilités à haut risque."

RECENSER LES RESSOURCES ET LES COMPOSANTS

L'inventaire technologique de votre application consiste à **recenser les ressources et les composants** utilisés par l'application, le produit ou le système à auditer.

Dans un premier temps, il faut rechercher les **composants applicatifs** par exemple une application mobile ou une base de données. Il faut ensuite recenser les **composants techniques** de chaque composant applicatif comme le système d'exploitation ou le framework de développement par exemple.

A partir de quelles ressources pouvez-vous réaliser l'inventaire technologique ? Où trouver les informations sur les technologies utilisées ?

- **La documentation d'architecture** (spécifications techniques) est le plus souvent une mine d'or en termes d'informations. Référez-vous au **cahier des charges** de l'application.

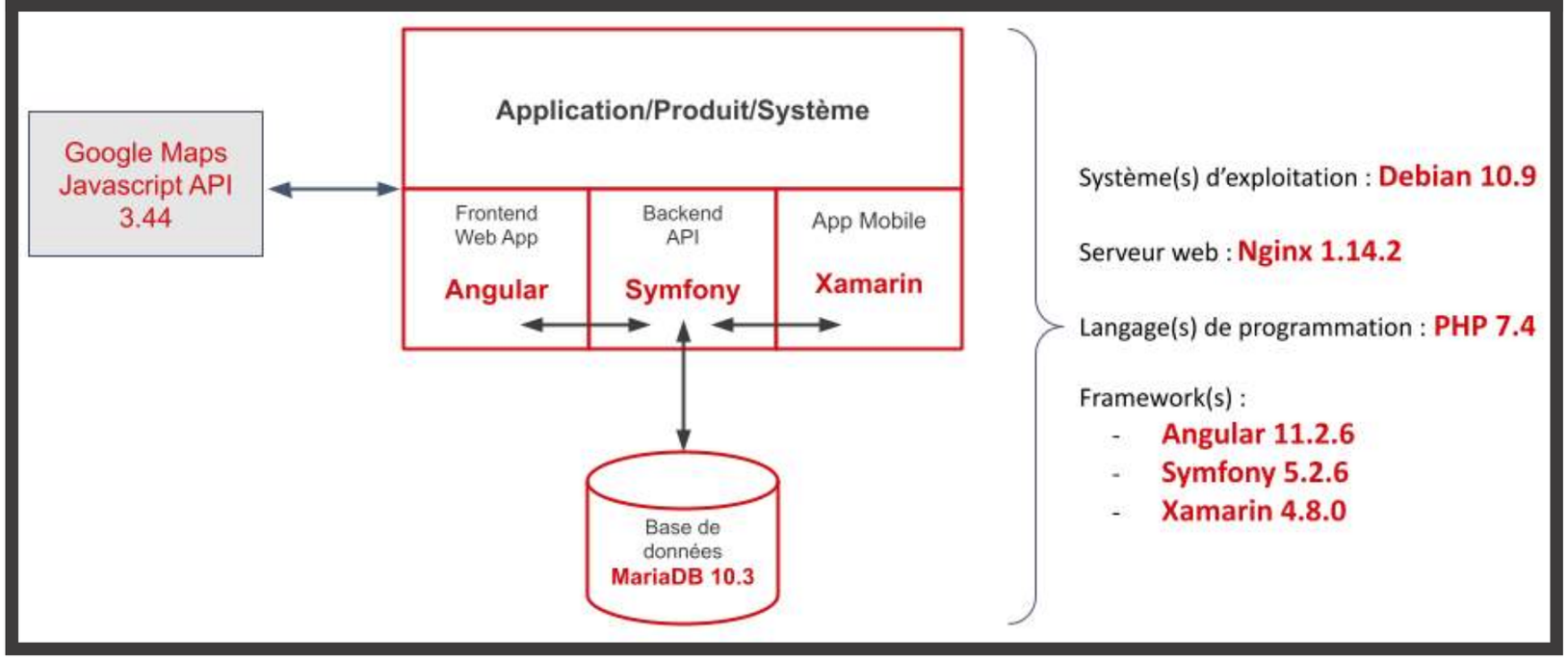
- **Le gestionnaire de sources** (Gitlab, Github…) est un outil collectant le code source, avec une gestion des versions tracées et des revues de codes. Il est utilisé pour produire des applications et gérer des tickets de support reliés à des bugs. Vous pourrez y trouver des **informations sur les technologies et les composants utilisés**.

- **Échanger avec l'équipe de développeurs**. N'hésitez pas à en parler avec un **profil technique**, le chef de projet ou un développeur, qui travaille sur votre application.

- Enfin, selon votre niveau de compétences, prenez le temps d'**explorer et d'inspecter les ressources techniques**. Vous pouvez inspecter le code source, les bases de données de l'application, les fichiers de configuration, les serveurs…

Afin de vous aider à comprendre, voici **l'exemple d'une architecture technique** que nous rencontrons régulièrement chez AxioCode :

Exemple d'architecture composant applicatif



Après avoir recenser tous les **composants, applicatifs et techniques**, il vous faudra associer le bon composant applicatif au bon composant technique.

Le tableau ci-dessous reprend l'ensemble des éléments du schéma précédent, nous y avons donc **associé chaque composant applicatif à son (ou ses) composant technique** :

Tableau des composants applicatifs et composants techniques associés

Composant applicatif	Composant technique associé
Application web en front-end *Le “front-end” correspond à ce que l’utilisateur peut voir et avec lequel il peut interagir directement.*	**Framework Angular version 11.2.6.** *Un framework est en quelque sorte une boîte à outils de développement.*
Application back-end (API) *Le “back-end” représente tout ce qui se passe en arrière-plan sans que l’utilisateur ne s'en rende compte.*	**Framework Symfony version 5.2.6** *Une API est une application qui va permettre à un système tiers d’interagir avec une entité informatique. Dans notre exemple, l’API permettra à l’application web en front-end et à l’application mobile d’interagir avec la base de données pour afficher les informations qui y sont enregistrées et les mettre à jour.*
Application mobile	**Framework Xamarin version 4.8.0**
Base de données	**MariaDB version 10.3**
Autres composants techniques	
Langage de programmation PHP 7.4	
API Google Maps version 3.44	
Système d’exploitation du serveur web Debian 10.9	
Serveur web Nginx 1.14.4	

VÉRIFIER LES VERSIONS DES COMPOSANTS

Après avoir fait l'inventaire technologique, il faut maintenant vérifier que tous les composants sont maintenus, mis à jour et surtout sans faille de sécurité. Pour vérifier si vos composants sont à jour, le plus simple est de se rendre sur **le site internet de l'éditeur**. Vous trouverez dans la page d'information les versions supportées ainsi que le numéro de la dernière version.

Les numéros de versions sont souvent construits selon la sémantique SemVer composé de trois numéros :

- Le **premier numéro** correspond au **numéro de version majeure**, il change dès que des changement non rétrocompatibles sont appliqués
- Le **second** est le numéro de **version mineure** et correspond à l'ajout de fonctionnalités rétrocompatibles
- Le **dernier numéro** marque la **correction d'anomalies rétrocompatibles**, c'est le numéro de version correctif

Par exemple, la version 5.2.6 de Symfony indique qu'il s'agit de la cinquième version majeure de ce framework, de la deuxième version mineure de cette version cinq et de la sixième version de correctifs apportés à la deuxième version mineure.

A partir de la version du composant que vous utilisez, nous vous recommandons de noter les informations suivantes :

- Le **type de dernière mise à jour** (release) :
  - Active : la version est maintenue activement par l'éditeur
  - Maintenance : seuls les bugs et problèmes de sécurité sont adressés
  - LTS (Long Term Support) : version maintenue plus longtemps que la normale par l'éditeur, ce qui laisse le temps aux développeurs de passer à la version majeure suivante.
- La **dernière date de mise à jour** (release).
- Le cas échéant, la **date de fin de LTS** (Long Term Support)

Il est important d'organiser une **veille technologique** régulière. Elle permet de suivre les **évolutions** des versions majeures des composants techniques utilisés dans vos applications. Si vous utilisez une version obsolète, cela peut **représenter un risque** pour votre organisation.

PACKAGES, BUNDLES, LIBRAIRIES ET OPEN SOURCE

Les composants techniques utilisent des sous-composants. Par exemple, les frameworks utilisent des **sous-composants techniques** appelés **packages, bundles, librairies**.

Ces sous-composants techniques sont bien souvent **Open Source**, c'est-à-dire que le code source est disponible et mis à jour par une communauté de développeurs indépendants, quelquefois soutenus par des entreprises.

Ils apportent des fonctionnalités techniques intéressantes dans les projets. Cela **évite aux développeurs de réécrire certaines fonctionnalités** comme la génération de fichier PDF, l'accès à des bases de données, etc. De la même manière, ils doivent faire l'objet d'un suivi particulier et régulier.

Pour déterminer les versions utilisées, vérifiez ou faites vérifier les fichiers de configuration **référençant les packages utilisés** par l'application. Par exemple, les fichiers : *composer.json, package.json, pom.xml.*

Utilisez les **gestionnaires de packages pour détecter les versions trop anciennes** qui posent problème : *Packagist pour PHP, NPM pour NodeJS, Nuget pour .NET ou encore Maven pour Java.*

Exemples de gestionnaires de packages

Packagist (PHP)

NPM (NodeJS)

Nuget (.NET)

Maven (Java)

Pour illustrer nos propos vous trouverez ci-dessous un modèle excel pour effectuer votre **veille technologique**, il vous faudra collecter :

- Les **versions utilisées** et leurs dates de sortie
- Les **dates de fin de support** pour anticiper la fin de vie d'une version
- La **dernière version de l'éditeur** pour mettre en lumière les composants trop anciens

Modèle Excel de veille technologique

Technologies

Composant Applicatif	Composant Technique	Version Utilisée				Support						Dernière Version		
		Numéro	Date	Age (jours)	Type	Actif	Fin dans (jours)	Sécurité	Fin dans (jours)	LTS	Fin dans (jours)	Numéro	Date	Age (jours)
Système d'exploitation	Debian	10.9	16/04/2021	5	Active					30/06/2024	1166	10.9	16/04/2021	5
Base de données	MariaDB	10.3	22/02/2021	58	Active					30/06/2024	1166	10.5.9	22/02/2021	58
Frontend Web App	Angular	11.2.6	17/03/2021	35	Active	11/5/2021	20	11/05/2022	385	11/05/2022	385	11.2.10	16/04/2021	5
Frontend Web App	Google Maps API	3.44	15/02/2021	65	Active							3.44	15/02/2021	65
Backend API	PHP	7.4.16	04/03/2021	48	Active	28/11/2021	221	22/11/2022	580			8.0.3	04/03/2021	48
Backend API	Symfony	5.2.6	29/03/2021	23	Active	31/07/2021	101					5.2.6	29/03/2021	23
Backend API	Doctrine Bundle	2.3.0	16/03/2021	36	Active							2.3.1	05/04/2021	16
						...								
App Mobile	Xamarin Forms	4.7.0 SR 6	28/08/2020	236	Maintenance							5.0.0 SR 3	12/02/2021	68
App Mobile	Xamarin iOS	13.10.0.17	13/01/2020	464	Maintenance			13/01/2022	267			14.14.1.0	09/02/2021	71
App Mobile	Xamarin Android	10.1.1.0	14/01/2020	463	Maintenance			14/01/2022	268			11.1.0.17	10/11/2020	162

Vous disposez maintenant d'un **inventaire technologique complet**. Cet inventaire en lui-même est intéressant, mais il n'est pas suffisant pour évaluer précisément **l'obsolescence** des technologies utilisées.

La prochaine étape présentée ci-après propose une méthode **d'évaluation du degré d'obsolescence** d'une application et d'identification des risques liés à l'obsolescence.

TELECHARGER LE MODELE EXCEL >

ESTIMER LE DEGRE D'OBSOLESCENCE

L'estimation du **degré d'obsolescence** d'une technologie repose sur deux questions :

Quelle est la probabilité d'obsolescence de cette technologie ?
Quel est l'impact de cette obsolescence sur l'entreprise ou l'organisation ?

De nombreux critères permettent d'évaluer la probabilité d'obsolescence d'un composant technique. Nous pouvons les classer en deux catégories :

- Les critères liés à l'écosystème du composant ou de la technologie étudiée
- Les critères techniques.

Intéressons nous dans un premier temps à l'écosystème.

ANALYSER L'ÉCOSYSTÈME APPLICATIF

L'écosystème d'une technologie se caractérise par l'ensemble des informations qui permettent de juger de sa pérennité. Nous avons défini 6 critères importants liés à l'écosystème : Vous pouvez utiliser une échelle de 1 à 9 (où 1=risque faible et 9=risque fort) pour déterminer le risque d'obsolescence à chaque critère.

Critères d'évaluation de l'écosystème applicatif

	0	1	2	3	4	5	6	7	8	9
Editeur de la technologie		Grande entreprise			Entreprise moyenne		Petite entreprise		Startups	
Communauté de développeurs		Importante			Moyenne			Confidentielle		Inexistante
Ressources, documentation		Abondantes		Quelques ressources				Très peu		Rare
Support à long terme de l'éditeur		> 2ans		< 2 ans		< 1 an		< 6 mois		Aucun support
Maturité de la technologie		Eprouvée > 10 ans			3 à < 5 ans			< 2 ans		"Vient de sortir

- L'**éditeur de la technologie** peut être évalué en fonction de son **ancienneté** et de la taille de ses équipes. Une grande entreprise présente un risque faible, tandis qu'une start-up a plus de chances de disparaître, sans solution de maintenance de la technologie.

- La **communauté des développeurs** autour de cette technologie : la communauté est-elle **importante**, **active** ? Il est possible de l'évaluer en consultant les forums et sites dédiés.

- Les **ressources et documentations** relatives à la technologie sont-elles **complètes**, régulièrement **mises à jour** et de qualité ?

- Quelle est la **disponibilité des compétences** pour cette technologie sur le marché de l'emploi ?
  Cela pose réellement **problème** pour les **technologies les plus anciennes**.

- Quelle est la **politique de support à long terme** de l'éditeur ?
  Vous pouvez l'évaluer en fonction du **nombre d'années que l'éditeur va garantir**.

- Quelle est la **maturité de la technologie** ?
  Une **technologie nouvelle représente plus de risques d'obsolescence** qu'une technologie éprouvée depuis des années.

DÉFINIR LES CRITÈRES D'OBSOLESCENCE

Passons maintenant aux **critères d'appréciation** liés à la technologie du composant.
Vous trouverez cinq critères importants permettant d'évaluer l'obsolescence :

- La **facilité de la mise à niveau** par rapport à la version utilisée par l'application que l'on audite : sera-t-il facile de passer à la version supérieure ?

- La **facilité du langage de programmation** (le cas échéant)

- La **facilité du framework de programmation** (le cas échéant)

- Les **failles de sécurité connues** de la version du composant technique utilisée

- Les **problèmes de compatibilité** avec d'autres technologies

Voici un **exemple de grille de notation** de la probabilité d'obsolescence de trois technologies utilisées par une application. Pour chacune des technologies, nous avons **attribué une note à chaque critère**, de 1 à 9. Nous obtenons ainsi une moyenne par technologie et une moyenne globale.

Grille de notation des technologies utilisées

	Technologies utilisées		
	Symfony 2.8	**PHP 5.6**	**Debian 6**
Facteurs écosystème			
Editeur de la technologie	6	1	1
Communauté de développeurs	7	1	7
Ressources, documentation	5	1	1
Disponibilité des compétences	8	1	6
Support à long terme de l'éditeur	9	9	9
Maturité de la technologie	1	1	1
Facteurs techniques			
Facilité de mise à niveau de la techno	7	5	5
Facilité du langage	2	2	
Facilité du framework	6		
Failles de sécurité connues	7	7	8
Problème de compatibilité avec d'autres techno	8	8	9
	6,0	**4,0**	**6,0**
Niveau de probabilité d'obsolesence global	**5,3**		

Dans notre exemple, la **probabilité d'obsolescence** de Symfony 2.8 (framework PHP web) et de Debian 6 (système d'exploitation) est élevée. La probabilité d'obsolescence de PHP 5.6 est moyenne.

Le score global est de 5.3, ce qui représente une **probabilité moyenne d'obsolescence**. Le score tend vers 6, ce qui signifie que l'application est en voie d'obsolescence. Néanmoins, **quelques facteurs** sont tout de même **rassurants**, Symfony 2.8, l'éditeur de ce langage (SensioLabs) est **sérieux** et met en confiance pour passer à la **version supérieure** rapidement.

COMPRENDRE L'IMPACT DE L'OBSOLESCENCE

Nous avons vu comment estimer la **probabilité d'obsolescence** d'une technologie, tant sur des critères liés à l'écosystème que sur des critères techniques. La seconde question porte sur **l'impact de l'obsolescence** des technologies sur l'entreprise ou l'organisation : quel sera l'impact du maintien d'une ou plusieurs technologies probablement obsolètes ?

Ces **critères d'impact** sont très souvent à mettre en contexte, **en fonction de l'application auditée** et de l'activité même de l'entreprise. Par exemple, la défaillance d'un module de paiement par carte bancaire aura un impact très important sur l'activité d'une entreprise de e-commerce.

Voici quelques **exemples de critères d'impact** à évaluer :

- L'**exposition aux vulnérabilités** et aux attaques informatiques,
- La **perte financière** que pourrait générer la défaillance de la technologie,
- L'**atteinte à la réputation** de l'entreprise,
- La **non conformité de l'application**,
- etc.

Nous utiliserons, là encore, une échelle de 1 à 9 *(où 1=risque faible et 9=risque fort)* pour attribuer un score de risque d'obsolescence à chaque critère.

Critères d'évaluation de l'impact de l'obsolescence

Impacts	
Exposition aux vulnérabilités et aux attaques	9
Perte financière	6
Atteinte à la réputation	3
Non conformité	7
Impact global	**6,3**

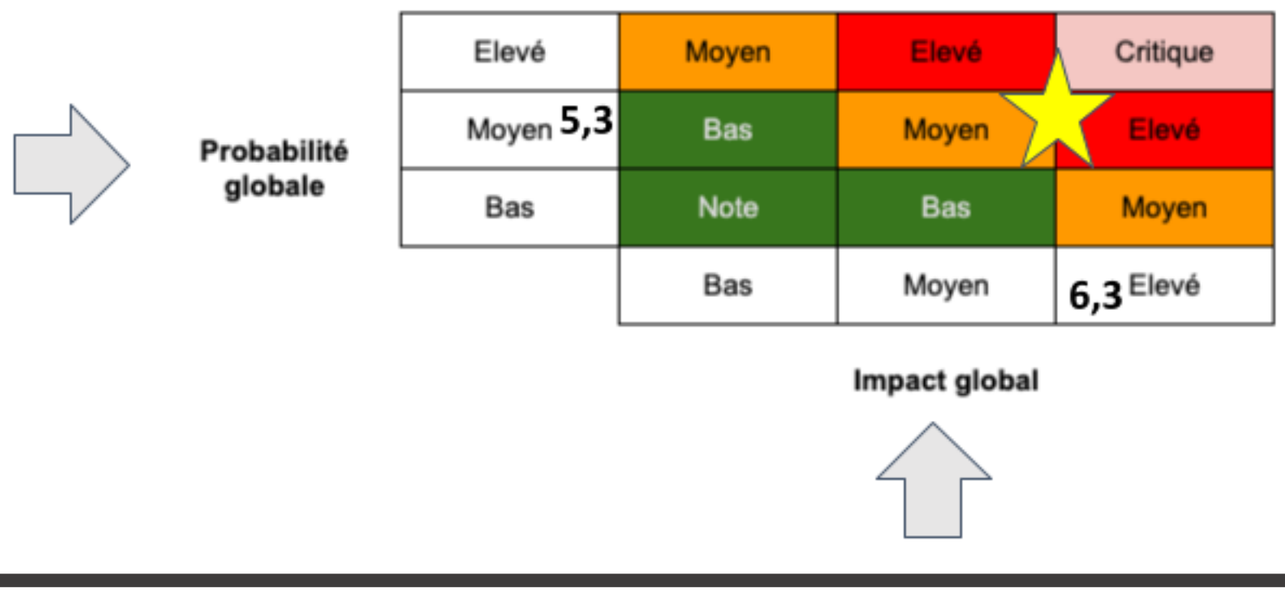
Dans cet exemple, nous avons évalué 4 impacts, l'application présenterait **un risque de vulnérabilité** et exposerait le système aux attaques.

Si l'application devient **obsolète**, il est fort probable qu'elle ne soit **plus conforme** et ne réponde plus complètement à vos besoins métiers.

Elle représente aussi un risque élevé de **perte financière** pour l'entreprise en cas de défaillance. Son score global de 6.3, est élevé. Si cette application devient obsolète, **l'impact sur l'entreprise sera fort** : elle représente un danger pour l'entreprise.

En croisant ces deux indicateurs, la probabilité d'obsolescence et l'impact de cette dernière, nous pouvons obtenir **la sévérité du risque d'obsolescence**.

Calcul de la sévérité du risque d'obsolescence



Dans notre exemple nous avons un niveau de **probabilité d'obsolescence de 5,3**. C'est un niveau de probabilité "moyen" mais tout de même proche de "élevé". L'impact de l'obsolescence est de 6,3 donc plutôt élevé.

Le croisement des 2 indicateurs nous donne une **sévérité du risque élevée**.

Dans cet exemple, nous sommes face à une application **obsolète**, à un **niveau** qui n'est pas loin d'être **critique**.

CHOISIR LES TECHNOLOGIES À METTRE À JOUR

Lorsqu'une application présente un **degré d'obsolescence très élevé**, cela peut signifier qu'il est trop tard pour l'améliorer. Il sera souvent **moins coûteux et moins risqué de développer une nouvelle application** pour la remplacer. Mais cette solution "révolutionnaire" est celle du dernier recours, fort heureusement la moins fréquente. Autant que possible, il faut privilégier une méthode d'amélioration **étape par étape**. Par où commencer ?

Il semble évident de **prioriser la mise à jour des technologies ayant la probabilité d'obsolescence la plus élevée**. Il faut toutefois tenir compte des **compétences disponibles** pour réaliser les travaux nécessaires, ainsi que de la facilité et/ou de la rapidité de mise à jour.

DRESSER L'ETAT DES RESSOURCES HUMAINES

Dans un premier temps, nous allons parler des **compétences** mettant en valeur le capital humain de l'entreprise. Elles jouent un rôle essentiel mais pas seulement !

Les compétences regroupent des **qualifications professionnelles** qui peuvent être "critiques", **longues à acquérir** ou **difficiles à transmettre** et devant être maintenues. Les **facteurs de criticité** des compétences ont donc été déterminés par rapport à ce qui les distingue des connaissances.

L'objectif de cette étape est de vous assurer de la disponibilité des compétences métiers et techniques nécessaires au maintien du bon fonctionnement de l'application.

Pour établir la **liste de ces compétences**, indiquez la criticité de chaque compétence ainsi que **le nombre de ressources humaines** minimum requises. Ensuite, évaluez les ressources **disponibles**, qu'il s'agisse de salariés de votre organisation ou de ressources externes (chez un prestataire par exemple).

IDENTIFIER LES COMPÉTENCES NÉCESSAIRES

Lors des étapes précédentes de l'audit, nous avons listé les fonctionnalités métiers relatives à l'application et nous avons fait un inventaire des technologies utilisées. D'après la lecture des tableaux remplis précédemment, nous avons d'un côté des **compétences métiers** : certaines personnes sont capables d'établir les fonctionnalités, de les expliquer et d'en demander d'autres car elles connaissent le métier associé à l'application.

Puis de l'autre, nous avons un inventaire des technologies. Certaines personnes ont des **compétences techniques** pour mettre en œuvre les fonctionnalités demandées dans l'application.

Facilitez le **recensement des compétences métiers et techniques** des personnes, en les listant dans deux tableaux, et en précisant la **disponibilité des compétences**, et l'évaluation des besoins en effectifs

Compétences métiers et techniques nécessaires

Compétences métier requises	**Compétences techniques requises**
Compétence	Compétence
Gestion commerciale	Symfony
Facturation	SQL Server
Process de réservation	Spécifications fonctionnelles
Litiges et SAV	Spécifications techniques
Juridique	Xamarin
Paiement en ligne	Gestion de projet
	Angular
	Maintenance serveur

Le tableau précédent représente les **compétences métiers et techniques** utilisées pour l'application.

Le travail d'audit va donc permettre de dresser une **liste précise des compétences métiers et des compétences techniques**.

Pour chacune de ces compétences, il faudra définir :

- La **criticité de la compétence** avec une échelle de 3 valeurs *(haute, moyenne, basse)*
- Le **nombre de ressources humaines nécessaires** par compétence

⚠ Attention : Le nombre de ressources est à évaluer en fonction des besoins au moment de l'audit.

Si la quantité des fonctionnalités est importante et que l'application est **encore en développement**, les **besoins** seront certainement **plus importants** que si l'application est passée en mode maintenance avec quelques évolutions dans l'année.

Compétences métiers et techniques requises

Compétences métier requises		
Compétence	Criticité	Nb requises
Gestion commerciale	Moyenne	1
Facturation	Moyenne	1
Process de réservation	Haute	2
Litiges et SAV	Basse	1
Juridique	Moyenne	1
Paiement en ligne	Haute	2
Compétences techniques requises		
Compétence	Criticité	Nb requises
Symfony	Haute	2
SQL Server	Haute	1
Spécifications fonctionnelles	Haute	1
Spécifications techniques	Haute	1
Xamarin	Haute	1
Gestion de projet	Moyenne	1
Angular	Moyenne	2
Maintenance serveur	Haute	1

Le tableau suivant **regroupe toutes les compétences métiers** associées à leur niveau de criticité et au nombre de personnes nécessaires pour mettre en œuvre ces compétences.

Il suffit ensuite de **répéter l'opération** pour lister les compétences techniques avec les mêmes critères.

Une fois le listing des compétences métiers et techniques effectué, **vérifiez si les besoins sont satisfaits et si le nombre de personnes est optima**l, insuffisant ou excédentaire par rapport au besoin.

IDENTIFIER LES RESSOURCES DISPONIBLES

Pour vérifier si les besoins en compétences sont satisfaisants, il faudra établir **la liste des ressources humaines**. Vous devez faire l'inventaire des ressources **internes, mais aussi externes** (comme des consultants ou des développeurs chez un prestataire).

Pour chacune de ces ressources humaines, selon vos besoins, définir :

- L'**identité de la personne**
- Sa **fonction** principale
- Ses **compétences métiers et/ou techniques**
  (une personne peut avoir plusieurs compétences)

Pour chacune des compétences listées, il faudra préciser son **niveau de maîtrise**.

⚠ Attention : Comme pour les besoins, le niveau des ressources est à évaluer au moment de l'audit.

Au moment de l'audit, s'il est soulevé des problèmes de ressources humaines, comment **évaluer ces ressources externes chez un prestataire** ?

Les équipes de développeurs associées à la maintenance peuvent évoluer. Suivre ces modifications est important pour évaluer la **capacité à maintenir les compétences et vérifier leur niveau de maîtrise** Dans le cas de ressources externes, vous pouvez demander à votre prestataire des informations (CV, compétences, présentation des personnes) sur les ressources humaines affectées à **votre application web ou mobile**

Dans cet exemple, trois tableaux mettent en valeur les ressources de l'équipe projet.

Nous définissons dans un premier tableau, une **liste de collaborateurs** avec leurs fonctions et leur provenance (interne ou externe). Deux autres tableaux définissent les **compétences métiers**, les **compétences techniques** du collaborateur et le niveau d'expertise associé à sa compétence.

Par exemple, à la lecture du tableau, nous remarquons qu'Edouard, un sénior, maîtrise 3 compétences métiers dans des domaines différents (la gestion commerciale, la facturation et le process de réservation).

Du côté technique, nous établissons une liste de collaborateurs aux diverses **compétences requises par l'application**, avec leur niveau d'expertise. Pour chaque composant technique, vous avez une équipe de séniors et juniors qui peuvent pallier aux besoins.

Collaborateurs et fonctions

Collaborateur	Fonction	Type
Arnaud	Chef de projet métier	Interne
Benjamin	Chef de projet technique	Interne
John	Lead développeur	Interne
Martin	Développeur	Interne
Samir	Développeur	Externe
Abdel	Développeur	Externe
Nadia	Développeur	Interne
Leïla	Développeur	Interne
Georges	DevOps	Externe
Edouard ...	Commercial	Interne
Magali	SAV	Interne
Arthur	Juriste	Interne

Compétences métiers associées

Compétences métier

Compétence	Collaborateur	Niveau
Gestion commerciale	Edouard ...	Senior
Facturation	Edouard ...	Senior
Process de réservation	Edouard ...	Senior
Litiges et SAV	Magali	Débutant
Litiges et SAV	Arthur	Junior
Juridique	Arthur	Junior

Compétences techniques associées

Compétences techniques

Compétence	Collaborateur	Niveau
Symfony	Martin	Senior
Symfony	Samir	Senior
Symfony	Nadia	Junior
SQL Server	Leïla	Junior
.Net	Leïla	Junior
Xamarin	Abdel	Senior
.Net	Martin	Junior
SQL Server	John	Senior
SQL Server	Benjamin	Senior
Angular	John	Senior
Angular	Nadia	Senior
Angular	Leïla	Junior

ETUDE DE CAS : ANALYSE DES RESSOURCES HUMAINES

A partir des données que vous aurez récolté, il sera possible de s'assurer que **toutes les compétences** requises **sont bien couvertes** par des ressources disponibles.

Nous avons pour vous un **exemple** où les compétences métiers sont représentées dans un tableau. On y retrouve le nombre de ressources requises au moment de l'audit ainsi que le nombre de personnes disponibles.

Pour évaluer la criticité des compétences, trois valeurs seront attribuées pour préciser leur niveau de criticité :

- **Criticité haute**, risque de défaillance si personne ne traite la tâche
- **Criticité moyenne**, risque de défaillance à moyen terme
- **Criticité basse**, risque de défaillance faible

Dans cet exemple, il est nécessaire d'avoir **deux personnes affectées** à la compétence "Paiement en ligne" sous **risque d'une forte perturbation** (voire de l'arrêt) de l'activité de l'entreprise en cas de défaillance.

On observe qu'au moment de l'audit, il n'y a plus **aucune personne compétente** (disponibilité 0) sur cette fonction critique. Il est pourtant primordial de **maintenir ces compétences** au risque de mettre le système de paiement en péril.

Nous allons produire la même analyse au sujet des compétences techniques. On observe dans le tableau ci-dessous qu'il y a **2 compétences critiques et non couvertes** par une ressource (disponibilité 0).

Vous pouvez également observer une valeur de 2,5 pour maintenir l'application.
Ce n'est pas deux personnes et demi : **un junior comptera pour 0,5**, alors qu'**un sénior comptera pour 1**. Il faut alors affecter 2 séniors et un junior pour couvrir le besoin en ressources et permettre le bon fonctionnement de votre application web ou mobile.

Bien entendu, vos formules de calcul sont **modifiables selon l'analyse personnelle** de vos besoins.
Par exemple, faire varier l'analyse en fonction de la criticité des compétences et du niveau de compétence.

Compétences métiers et techniques requises

Compétences techniques requises

Compétence	Criticité	Nb requises	Disponibles
Symfony	Haute	2	2,5
SQL Server	Haute	1	2,5
Spécifications fonctionnelles	Haute	1	0
Spécifications techniques	Haute	1	0
Xamarin	Haute	1	1
Angular	Moyenne	2	2,5
Maintenance serveur	Haute	1	1

Compétences métier requises

Compétence	Criticité	Nb requises	Disponibles
Gestion commerciale	Moyenne	1	1
Facturation	Moyenne	1	1
Process de réservation	Haute	2	1
Litiges et SAV	Basse	1	1
Juridique	Moyenne	1	1
Paiement en ligne	Haute	2	0

Après enquête et analyse des besoins humains et techniques, vous disposez de suffisamment d'informations sur **vos besoins en compétences métiers et techniques** et sur les ressources humaines qui y sont associées. L'audit a pour but de soulever les **risques de défaillance**, si vous ne disposez pas de ressources suffisantes, et d'y remédier afin de rendre votre application pleinement fonctionnelle.

La **défaillance** de l'application peut avoir des conséquences sur le personnel, l'environnement, le respect de la réglementation, et peut **entraîner un fort ralentissement de productivité et même un arrêt d'activité** ! Une raison de plus pour auditer rigoureusement votre application métier.

ANALYSER LE CODE SOURCE

Les applications métiers, qu'elles soient web ou mobile, sont des **systèmes complexes** car elles **répondent à un besoin très précis** de l'entreprise. Ce sont des applications qui évoluent avec l'organisation, elles reçoivent donc des améliorations et des modifications tout au long de leur vie. Votre application métier est aussi **soumise aux aléas du temps**, elle doit donc pouvoir s'adapter aux évolutions et à la durée de vie de ses composants.

Il faut donc procéder à des **vérifications régulières** afin de minimiser les risques d'obsolescence de votre application. Lors de l'audit d'une application, une des étapes cruciale est **l'analyse du code source**. Cette étape fortement recommandée peut parfois mettre à jour des failles de sécurité.

Bien entendu, il faudra disposer du code source de chacun des **composants** de votre application. Supposons que le système soit composé d'un back office web et d'une application mobile, il faudra **analyser le code** de ces deux composants applicatifs.

Vous pensez sûrement que l'étude du code source de votre application peut se faire **manuellement**. Mais c'est une tâche très complexe et vous passeriez certainement à côté de certains problèmes majeurs. C'est pourquoi nous vous conseillons de réaliser votre audit en vous aidant d'**outils d'analyse statique de code**.

LES OUTILS D'ANALYSE DU CODE SOURCE

Pour faciliter l'analyse du code source, des **outils** payants sont accessibles, avec des tarifs et des stratégies d'éditeurs variables. Certains sont adaptés pour de grandes entreprises et d'autres plus **abordables** pour des organisations de moindre ampleur.

Voici quelques outils d'analyse qui vous alertent lorsqu'ils rencontrent une anomalie ou une erreur potentielle :

- **Sonarqube**
- **Code Climate**
- **Codacy**
- **Coverity**
- **Checkmarx**
- etc.

Ces outils **gèrent beaucoup de langages** connus (PHP, Java, .Net, etc.), ils évaluent la **qualité** du projet et indiquent la **présence de bugs** (problèmes) dans le code source. Certains outils permettent même d'attribuer un score de dette technique et une évaluation de la charge de travail pour y remédier.

Ces outils **facilitent** grandement la gestion de la qualité du code. Ils permettent de **mettre en évidence les problèmes** liés directement au code source de votre application web ou mobile et de soutenir les développeurs dans leur travail d'analyse.

QUELLES MÉTRIQUES PRENDRE EN COMPTE ?

Lors de l'analyse du code source, il faut déterminer des **métriques**. Ces valeurs serviront de **référence** à l'analyse de la qualité du code source de votre application. Voici quelques métriques pouvant être utilisées :

- **Le nombre de fichiers, de lignes de code, de commentaires, le taux de commentaires** par rapport au nombre de lignes de code.

- La **complexité cyclomatique** définit le nombre de **chemins uniques** qu'il est possible d'emprunter. La complexité cyclomatique n'est autre que la somme de tous les chemins uniques possibles. De fait, plus il y a de chemins uniques (comme des conditions) dans le code source de votre application, plus la complexité cyclomatique est élevée. Elle permet de **se rendre compte visuellement de la complexité de la structure** d'un programme. L'objectif est d'éviter les pertes de temps liées à la compréhension du code.

Par exemple, observons deux fonctions visibles dans 2 colonnes différentes. 2 codes sources sont présentés avec la même complexité cyclomatique de 4. Mais avec une complexité cognitive différente. Le **code de droite** est beaucoup **plus lisible** grâce à l'utilisation de structures de contrôles différentes et révèle moins de complexité cognitive.

Exemple de complexité cyclomatique

```
int sumOfPrimes(int max) {                    // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) {       // +1
    for (int j = 2; j < i; ++j) {             // +1
      if (i % j == 0) {                       // +1
        continue OUT;
      }
    }
    total += i;
  }
  return total;
}                    // Cyclomatic Complexity 4
```

```
String getWords(int number) { // +1
    switch (number) {
      case 1:                 // +1
        return "one";
      case 2:                 // +1
        return "a couple";
      case 3:                 // +1
        return "a few";
      default:
        return "lots";
    }
  }      // Cyclomatic Complexity 4
```

- La **complexité cognitive** est le fait de quantifier l'effort nécessaire et la **facilité avec laquelle une personne** (développeur ou développeuse) **est capable de le comprendre**. Elle mesure la charge mentale induite par le code source. Elle encourage les pratiques de codage "propres" en incrémentant les constructions de code qui demandent un effort supplémentaire à le comprendre.

La lecture du code source peut être **complexe**. Déterminer des métriques permet de mieux observer le code. Ces indicateurs permettent de définir la rapidité avec laquelle le code source va être compris, que ce soit par une machine ou un humain. Par exemple, selon la **complexité cognitive**, ce sera plus ou moins simple d'étudier le code source et de le reprendre.

Il faut donc déterminer si le code source est **qualitatif, lisible, propre et compréhensible** par tous. Attention, en cas de reprise de code source, plus il est complexe, plus il sera long à étudier.

Analyser ces relevés permet d'évaluer la complexité du code, de mesurer les portions du code pour les comparer et pointer les endroits les plus pertinents en termes d'amélioration. **Ces métriques se concentrent sur les zones à risque, plus susceptibles de contenir des vulnérabilités.**

DÉTERMINER LA QUALITÉ DU CODE

Trois aspects caractérisent la qualité du code source.

- La **stabilité** se définit par le **nombre de bugs**, d'anomalies potentielles détectées dans le code,

- La **sécurité** correspond au **nombre de vulnérabilités potentielles** qui pourraient être exploitées par des hackers. Il est important que les bonnes pratiques en matière de sécurité soient respectées dans le code source. En effet, il est plus fréquent que ce que l'on croit de trouver des mots de passe en clair dans du code source, rendant votre application vulnérable.

- La **maintenabilité** est matérialisée par ce que l'on appelle les "code smells". Un code smell, n'est rien d'autre que du "mauvais code" soit **un code de mauvaise qualité**. Il est le résultat de mauvaises pratiques de conception.

Il est intéressant de mesurer la qualité du code pour définir si le code est maintenable sur le long terme.

SÉCURITÉ ET VULNÉRABILITÉS

Du point de vue de la sécurité et des vulnérabilités, il existe **Open Web Application Security Project (OWASP)**.

C'est une communauté (et une fondation) qui travaille à l'élaboration des **bonnes pratiques en matière de sécurité** des applications web. OWASP est aussi responsable du projet Top Ten. Ce projet est aujourd'hui une référence internationale dans le domaine de la sécurité. Beaucoup d'outils d'analyse utilisent cette **codification** pour classer les vulnérabilités détectées.

Le projet Top Ten classe et détaille **les 10 risques les plus critiques** (codifiés de A1 à A10) avec des indicateurs vous donnant des informations sur la gravité de certaines des failles. Voici le classement qui identifie les **vulnérabilités** de sécurité les plus critiques.

Top Ten - Les 10 risques les plus critiques

A1 : Défaut d'injection	**A6 : Mauvaise configuration de sécurité**
A2 : Authentification de mauvaise qualité	**A7 : Cross-Site Scripting (XSS)**
A3 : Exposition de données sensibles	**A8 : Désérialisation non sécurisée**
A4 : XML External Entities (XXE)	**A9 : Utilisation de composants avec des vulnérabilités connues**
A5 : Violation de contrôle d'accès	**A10 : Insuffisance de journalisation et de surveillance**

Source : OWASP Top Ten Web Application Security Risks | OWASP

Chaque vulnérabilité est automatiquement supposée comme **potentiellement critique** et doit être analysée précisément. Certaines de ces failles vont être exploitables plus ou moins facilement. Des **failles de sécurité** peuvent subsister, car elles sont la plupart du temps dues à des erreurs de programmation de l'application.

Le manque de compétences en sécurité de certains développeurs peut expliquer les vulnérabilités récurrentes qu'elles présentent. Par exemple, une application peut comporter des erreurs perturbant son bon fonctionnement, en causant une lecture du code difficile (complexité cognitive).

On distingue **4 comportements possibles** en cas de détection d'une vulnérabilité :

- Un **vrai négatif** est une activité normale correctement considérée
- Un **faux positif** appelé aussi fausse alerte, est une activité normale considérée à tort comme une attaque
- Un **faux négatif** est une attaque non détectée, considérée donc à tort comme une activité normale
- Un **vrai positif** est une attaque correctement détectée et considérée comme telle.

Les **vrais négatifs** et **vrais positifs** correspondent aux **comportements souhaités**, l'efficacité d'une protection se mesure par le taux de faux positifs et de faux négatifs.

LES COÛTS LIÉS À L'ANALYSE DU CODE

Lors d'une analyse de code, nous cherchons aussi à obtenir les coûts liés pour **remettre la qualité du code à un niveau satisfaisant**.

Il y a **2 types de coûts** à prendre en considération :

- Le **coût de remédiation**, qui est une estimation du temps utile pour **corriger les bugs et les vulnérabilités**.

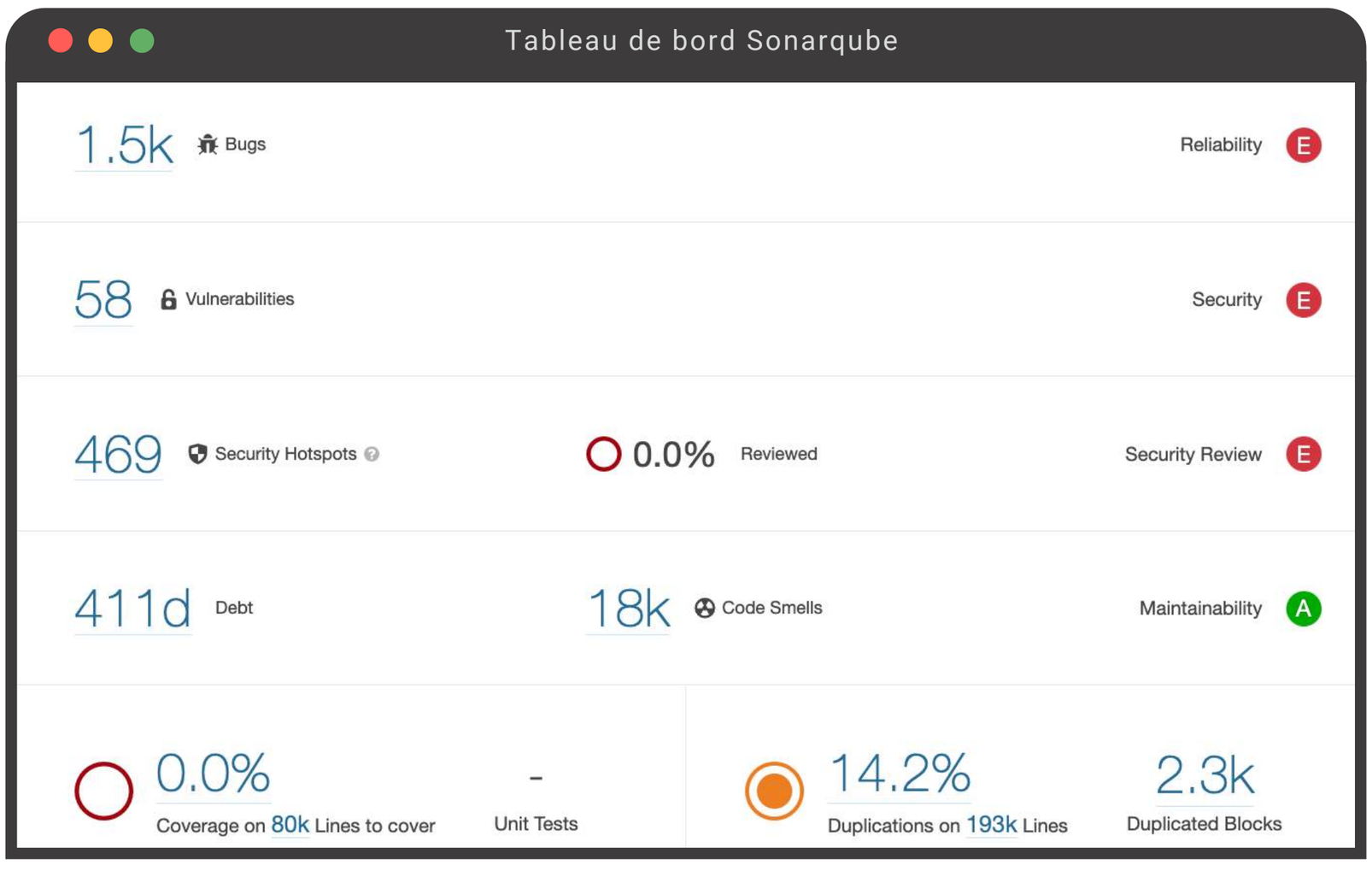
- Le **coût de la dette technique**, qui estime le temps nécessaire pour **corriger les problèmes liés à la maintenabilité et aux mauvaises pratiques** de conception.

ETUDE DE CAS : ANALYSE DU CODE SOURCE

Nous prenons comme exemple l'analyse d'une partie d'un code source avec l'outil *Sonarqube*. L'outil a détecté un **bug** (problème) potentiel jugé **critique**. Il indique qu'il faut **10 minutes** d'effort pour corriger ce problème précis dans le code.

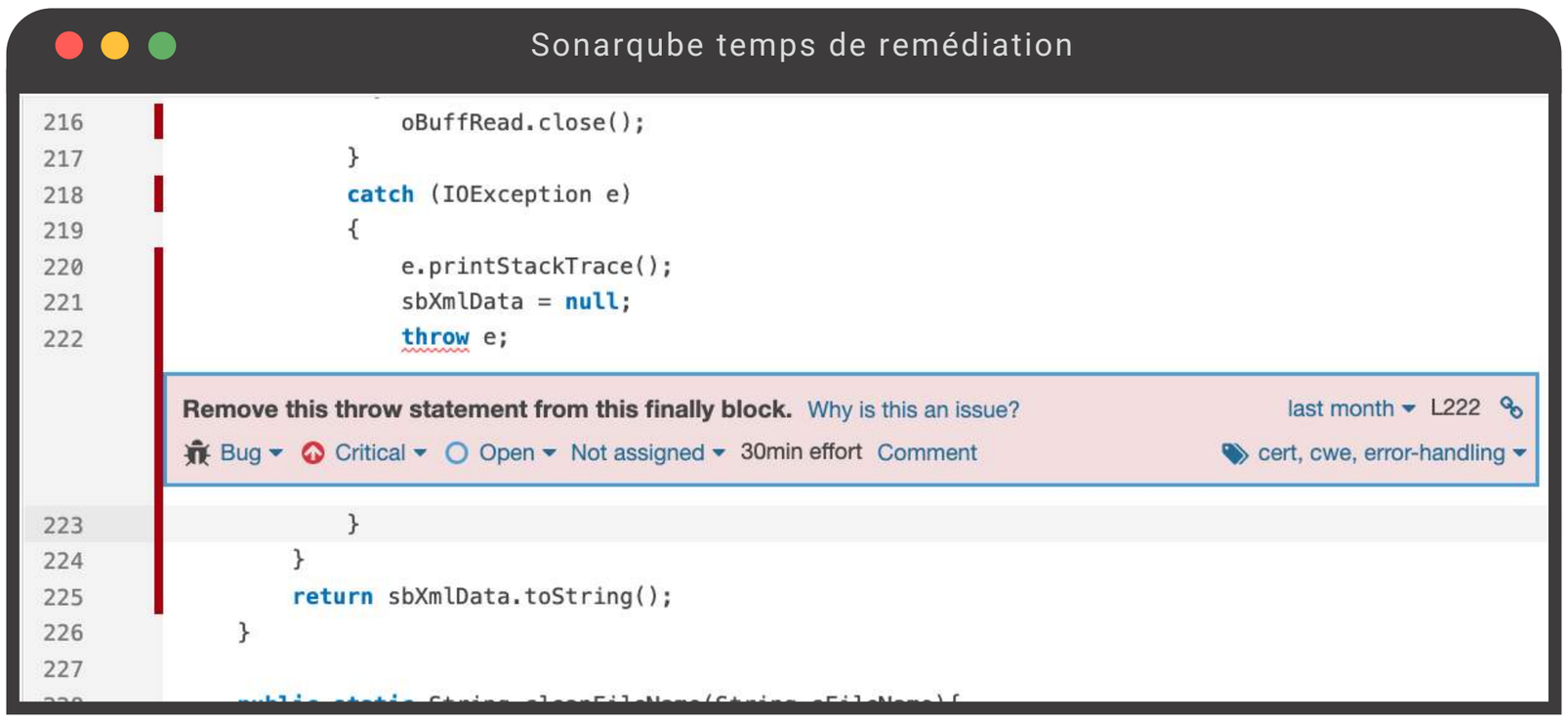
Voici un autre exemple présentant un tableau de bord avec une vue complète d'analyses et des mesures.

L'outil a détecté 1 500 bugs dans le code source, cela se traduit par une mauvaise note (E) et une mauvaise note en termes de fiabilité du code, mais il n'a pas soulevé de vulnérabilité particulière.

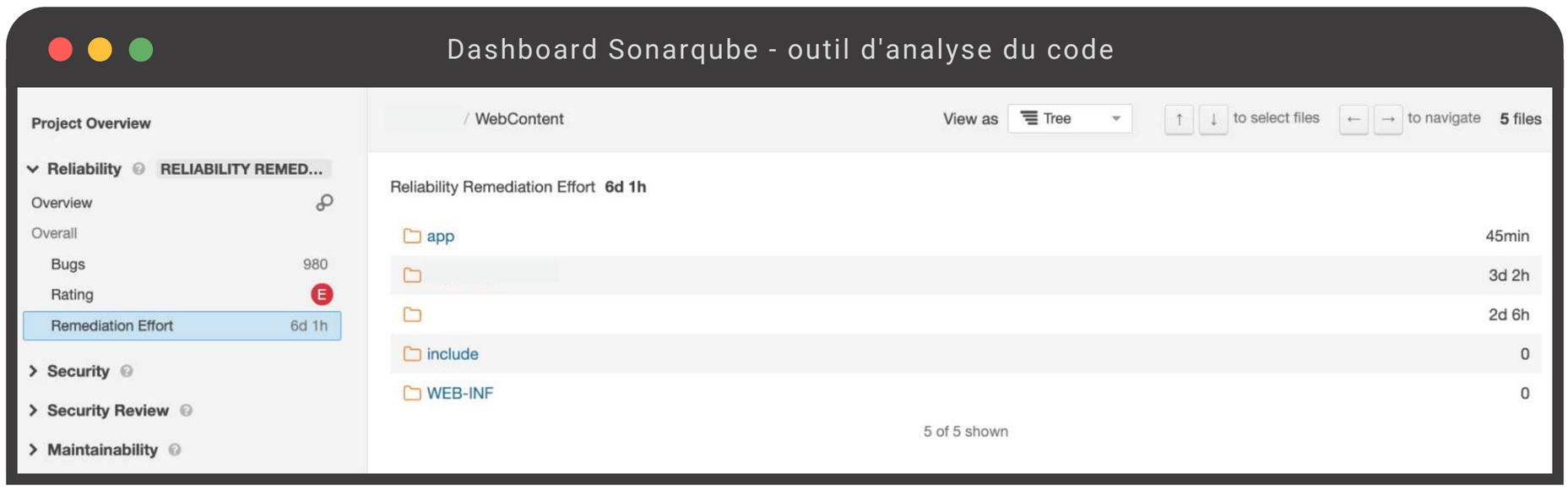


L'outil évoque une amélioration possible **en 411 jours** de dette technique (ce qui est très élevé en matière de temps) et indique qu'il y a 18 000 codes smells à vérifier, ce qui est un très bon indice de maintenabilité (ce chiffre est relativisé avec le nombre de lignes de codes ici 193 000). Soyez également attentif au concept de Hotspot. Les hotspots des endroits sensibles dans le code source de votre application ; il faut donc aller vérifier qu'il ne contient pas d'erreurs.

Dans la vue ci-dessous, l'outil Sonarqube montre qu'il existe des **efforts de remédiation** à réaliser. Il détermine un nombre de minutes d'efforts pour corriger et améliorer le code source pour chacun des dossiers utilisé dans l'application.



Comme **un audit du code source** est chronophage, des solutions intégrées à ces outils vous feront gagner du temps et vous permettront de mettre en place un plan d'action pour prioriser vos actions. Un planning prévisionnel vous aidera à gérer aussi bien **les actions correctrices** que **les actions préventives**, pour **empêcher l'augmentation de la dette technique**.



Nous vous conseillons d'auditer uniquement ce que vous maîtrisez à l'instant de l'audit applicatif. Pour ce qui est de la gestion du temps de travail que cela nécessite, **vérifiez les problèmes un à un**, les bugs et les codes smells. **Un audit peut durer de quelques jours à plusieurs semaines**. Cela varie en fonction de l'ampleur du code à auditer. En effet, plus l'application est complexe, plus le volume de code source est important, plus le temps à y accorder est important.

Une mauvaise gestion des erreurs peut **exposer des données sensibles**, ou se traduire par un **code non maintenable** (et donc coûteux à l'entretien). Les développeurs testent alors la sécurité de l'application lors du processus de développement, et vérifient que la nouvelle version ou la version mise à jour ne présente aucune vulnérabilité. Un audit de sécurité peut garantir **la conformité de l'application** avec un ensemble spécifique de critères de sécurité. Une fois que l'application a passé cet audit, les développeurs doivent s'assurer que seuls les utilisateurs autorisés peuvent y accéder.

Pour réduire les coûts liés à la dette technique, **un travail régulier sur la qualité du code** est indispensable. Il arrive que des vulnérabilités s'insèrent dans le code. Pour éviter cette situation, il est nécessaire de former les développeurs à l'identification des risques et d'utiliser des outils adéquats pour repérer plus facilement les vulnérabilités.

Si les applications ne sont pas maintenues à jour, elles deviendront vulnérables et s'exposeront à des attaques malveillantes pouvant porter atteinte à la confidentialité, à l'intégrité ou à la disponibilité des informations. Pour faire face à ces menaces, il est recommandé de procéder à des audits de code, de **former et d'outiller les développeurs** à l'identification des vulnérabilités et surtout à la qualité du code.

Avoir de la dette technique ne fait pas de votre application un échec. Le plus important est de réussir à la **traiter régulièrement** afin d'en améliorer la qualité et réduire les risques. L'ajout de nouvelles fonctionnalités peut amener de la dette technique, il faut donc **procéder régulièrement à des vérifications**. Il est nécessaire d'être conscient des failles à corriger afin de les réduire, même si cela engage du temps.

EVALUER LA COUVERTURE DES TESTS, DE LA DOCUMENTATION, DU PROCESSUS DE DÉVELOPPEMENT ET DE DEPLOIEMENT

A cette étape de l'audit, il est temps d'**évaluer la couverture des tests**, de la **documentation**, du **processus de développement et de déploiement**. Sans entrer dans un détail excessivement technique, nous allons voir quels en sont les principes généraux.

Il existe deux grandes familles de tests pour les applications web et mobiles :

- Les **tests unitaires** sont écrits **du point de vue du développeur**. Ils sont faits pour s'assurer qu'une **fonctionnalité** particulière effectue l'ensemble des tâches spécifiques qui lui sont affectées correctement. Pour les besoins de l'audit, l'évaluation des tests unitaires sera réalisée par l'analyse du code source de l'application.

- Les **tests fonctionnels**, aussi appelés tests d'acceptance, sont écrits **du point de vue de l'utilisateur**. Ils assurent que le système fonctionne comme les utilisateurs s'y attendent. Dans le cadre d'un audit applicatif, il faudra vérifier le **plan de tests**.

TESTS UNITAIRES, TESTS FONCTIONNELS ET PLAN DE TESTS

Les tests unitaires sont des **tests techniques** : ils permettent au développeur de tester une fonction, une procédure ou un module par exemple, indépendamment du reste de l'application. L'objectif est de s'assurer que l'unité testée fonctionne correctement en toutes circonstances. Cette vérification est **essentielle**, en particulier dans les **applications métiers** soutenant directement l'activité d'une entreprise.

L'évaluation des tests unitaires passe par **l'analyse du code source de l'application**. Les questions à se poser sont les suivantes :

- *Quel est le taux de couvertures des tests unitaires ?*
  Les outils d'analyse du code source (Sonarqube, Code Climate, Codacy, Coverity, Checkmarx,... etc.) vous permettent d'obtenir cette information qui sera exprimée en pourcentage.

- *Quel est le framework de tests unitaires utilisé ?*
  Est-ce un outil du marché ou open source répandu, est-il à jour ?

- *Comme pour les tests fonctionnels, est-ce que des jeux de données de tests sont prévus et cohérents ?*

- *A quel moment les tests unitaires sont-ils exécutés ? Manuellement par les développeurs ?*
  Dans un processus d'intégration continue, automatique ?

Il faut également tester votre application web ou mobile d'un point de vue utilisateur, c'est le rôle des **tests fonctionnels**. Une enquête d'AxioCode sur les pratiques pour la création rapide d'applications web et mobiles pérennes révèle que les tests fonctionnels sont loin d'être **systématiques** et encore moins **automatisés**.

Pourtant, ces tests sont **déterminants** dans l'objectif de la réalisation d'applications pérennes. Chaque livraison d'une mise à jour ou d'une nouvelle version de l'application doit être précédée d'une phase de tests complets. Il est en effet fréquent qu'une modification ait des **répercussions inattendues**, des effets de bord, qui se traduisent par des bugs. Il n'y a rien de pire que de livrer une nouvelle version de l'application qui ne fonctionne pas correctement.

Pour éviter cela, il est nécessaire de réaliser des tests fonctionnels **exhaustifs**. Réalisés “manuellement”, ils sont chronophages. D’où l’intérêt de mettre en place des tests fonctionnels **automatisés** qui vont simuler le comportement des différents profils d’utilisateurs et signaler d’éventuelles anomalies.

Découvrez dans cet article, les bonne pratiques pour créer une application web et mobile pérenne.

Pour les besoins de l’audit, il convient de vérifier qu’un plan de tests existe et d’évaluer sa qualité en se posant les questions suivantes :

- *Est-ce que chaque fonctionnalité / exigence de l’application est couverte par au moins un cas de test ?*
  Cela témoigne du degré de **complétude** du plan de tests, facteur essentiel de la qualité des tests.
  Un plan de tests partiel est contre productif.

- *Est-ce que les cas de tests sont correctement décrits ?*
  Ils doivent indiquer clairement quelle **action** est réalisée, par quel **profil d’utilisateur**, et quel **résultat concret** est attendu.

- *Est-ce que des jeux de données de tests sont prévus et cohérents ?*
  Les cas de tests doivent prévoir quelles **informations** seront manipulées ; autant que possible, d’une **campagne de tests** à une autre, les cas de tests reprendront les mêmes données qui doivent donc être **inventoriées**.

- *Quand les campagnes de tests sont-elles effectuées ?*
  A la demande, à chaque mise à jour de l’application, uniquement lors des mises à jour importantes ?
  Idéalement, **chaque livraison d’une nouvelle de l’application doit faire l’objet d’une campagne de tests** ; c’est d’autant plus important si l’application est vitale pour l’activité de l’entreprise ou l’organisation.

- *Par qui sont effectués les tests ? par les développeurs uniquement, des utilisateurs identifiés ?*
  Il est souvent délicat pour les développeurs de “changer de casquette”, de passer de la fonction de développeur qui réalise des tests techniques dits “unitaires” à un rôle d’utilisateur de l’application. D’où l’intérêt d’**impliquer de “vrais” utilisateurs** dans les phases de tests. Les équipes de développement les mieux organisées disposent de compétences spécifiques pour l’organisation et la réalisation des campagnes de tests de leurs applications.

- *Est-ce que certains tests, notamment les plus récurrents ou fastidieux, sont automatisés ?*
  Tester une application dans son ensemble demande beaucoup de temps. C’est la raison pour laquelle les tests sont rarement exhaustifs, presque toujours incomplets. La solution est de **mettre en place des tests fonctionnels automatisés** : ils sont codés et mis en œuvre via un système spécifique (comme Sentry par exemple) et ont tout leur intérêt pour les fonctionnalités les plus utilisées et les plus critiques.

ANALYSER LA QUALITÉ DE LA DOCUMENTATION

Le **cahier des charges** initial de l’application comporte le plus souvent des spécifications techniques. Mais au fil du temps et des changements apportés à l’application, cette documentation n’est pas toujours mise à jour.

L’enquête d’AxioCode sur les pratiques pour la **création rapide d’applications web et mobiles pérennes** indique que les **spécifications** (fonctionnelles et techniques) ne sont “le plus souvent mises à jour” que pour un peu plus de 50% des participants. La principale raison pour laquelle les spécifications ne sont pas mises à jour est le **manque de temps ou de budget**.

Pourtant, des **spécifications non mises à jour** sont problématiques. En particulier lorsqu’il faut se replonger dans l’application après quelque temps ou lorsque de nouveaux développeurs doivent intervenir. Lorsque l’on se base sur des spécifications qui ne sont pas à jour, le risque est d’apporter des modifications non pertinentes ou incohérentes, **sources d’erreurs** ou d’oublis. Il faudra parfois refaire ce qui avait été fait, ce qui se traduit finalement par de la perte de temps.

Le maintien à jour de la **documentation technique** est un investissement fondamental pour la bonne santé de l’application et sa pérennité.

L'audit de la documentation technique consiste à vérifier que celle-ci comporte des informations essentielles :

- l'**architecture logicielle détaillée** de l'application
- les **diagrammes pertinents**, par exemple en UML avec les classes utilisées
- le **schéma de la base de données**
- le cas échéant, la **description des API et des web services** utilisés
- la **liste des composants techniques** utilisés, documentés autant que possible

Il convient par ailleurs de se pencher sur le **circuit de rédaction, de relecture et de validation** de la documentation technique. Et bien sûr de vérifier que la documentation technique est bien maintenue dans le temps.

ANALYSER LA QUALITÉ DU PROCESSUS DE DÉVELOPPEMENT ET DE DÉPLOIEMENT

Les méthodes et les processus de développement et de déploiement sont déterminants dans la qualité et la pérennité des applications web et mobiles.

Les questions à se poser sont les sont les suivantes :

Quelle méthodologie l'équipe de développement suit-elle ?
Est-ce une méthode classique ou une méthode agile ?
La méthode retenue est-elle adaptée ?

En pratique, les méthodes agiles mettent le "**propriétaire**" de l'application (product owner) au **centre de l'organisation** des travaux, au quotidien. Ces méthodes ne peuvent pas garantir à la fois le périmètre fonctionnel (liste exhaustive des fonctionnalités de l'application) et les **délais de réalisation** : ce qui sera livré, c'est ce que l'équipe de développement aura eu le temps de coder en fonction des décisions prises de sprint en sprint. Les méthodes agiles sont donc difficilement compatibles avec un développement en sous-traitance, au forfait. A l'inverse, c'est une approche à retenir avec une équipe de développement interne qui travaille main dans la main avec les experts métiers.

Est-ce que la pratique de revue de code est en place ? Est-elle systématique ? Est-elle efficace ?

Comme l'indique Wikipédia, la revue de code est *"un examen systématique du code source de l'application. Il peut être comparé au processus ayant lieu dans un comité de lecture, l'objectif étant de trouver des bugs ou des vulnérabilités potentielles ou de corriger des erreurs de conception afin d'améliorer la qualité, la maintenabilité et la sécurité du logiciel (...). La revue de code est depuis longtemps reconnue comme un moyen performant d'améliorer la qualité des applications"*.

L'efficacité des **revues de code** peut être analysée au travers des bugs de l'application en se demandant pourquoi ils n'ont pas été détectés dans la phase de revue de code. Il convient également de consulter **les historiques de revue de code**, dont les commentaires seront instructifs.

Est-ce que des environnements techniques différents sont disponibles ?

Comme par exemple un environnement d'intégration pour les développeurs et un environnement de validation pour la recette (les tests), tous deux différents de l'environnement final de production ?

Existe-t-il un processus d'intégration continue permettant de s'assurer que les nouveaux développements ne génèrent pas de régression ? Quelle stratégie de déploiement en production est en place ? Le déploiement est-il automatique ou manuel ?

Vous avez en main les clés d'**analyse de la couverture des tests, de la documentation, du processus de développement et de déploiement** d'une application web ou mobile. Pour mener à bien ces analyses, contrairement à d'autres phases d'audit comme la vérification de la complétude fonctionnelle par exemple, vous aurez besoin de compétences techniques. Cette mission peut être par exemple effectuée par le chef de projet technique ou un développeur de l'application.

ESTIMER LA VALEUR FINANCIERE

Une fois toutes ces étapes passées et toutes les informations en votre possession vient le **processus de décision**. Faut-il remplacer l'application existante ? La moderniser ? Pour vous aider dans ce choix stratégique, l'une des clés est l'estimation de **la valeur financière de l'application**, des délais et des coûts de remplacement, de mise à niveau ou d'isolement.

Il est donc important pour vous de savoir comment :

- **Estimer la valeur financière de votre application**
- **Etablir un ratio entre le coût de maintenance et la valeur de votre application**
- **Réaliser un chiffrage du coût de réalisation de la version souhaitable de l'application**
- **Effectuer le choix stratégique de maintenance ou de remplacement**

ÉTABLIR LE COÛT

Si vous avez fait appel à un ou plusieurs prestataires, **basez vous sur les factures** qu'ils vous ont adressées. Pour les travaux réalisés en interne, vous pouvez **valoriser les temps passés** selon les coûts moyens journaliers (salaires, charges sociales, quote part de frais généraux) ou selon les tarifs du marché.
Il vous faut répartir ces coûts en deux catégories :

1. Les **coûts de développement**, de changements et d'améliorations fonctionnelles,
2. Les **coûts de maintenance** technique et de résolution d'anomalies.

Indiquez une date de mise en production qui permettra de calculer une dépréciation en fonction du temps écoulé.

Tableau d'amortissement linéaire sur 3 ans

Amortissement linéaire 3 ans. Valeur résiduelle = 1/3 de la valeur initiale											
Valorisation à la date du	15/5/2021										
Mise en prod	**Période**	**Objet**	**Type**	**Nb jours**	**Coût / jour**	**Coût**	**Ancienneté**	**Valeur**	**Valeur nette**	**Plancher**	**Retenu**
10/2/2017	2017	Application v1	Développement	65.00	550.00 €	35,750.00 €	51 mois	11,916.67 €	0.00 €	11,916.67 €	11,916.67 €
15/6/2017	2017	Application v2 finale	Développement	30.00	550.00 €	16,500.00 €	47 mois	5,500.00 €	0.00 €	5,500.00 €	5,500.00 €
31/12/2017	2017	Maintenance 2017	Maintenance			2,480.00 €	40 mois	826.67 €	0.00 €	826.67 €	826.67 €
31/12/2018	2018	Maintenance 2018	Maintenance			3,265.00 €	28 mois	1,088.33 €	0.00 €	1,088.33 €	1,088.33 €
31/12/2019	2019	Maintenance 2019 + techno	Maintenance			18,540.00 €	16 mois	10,300.00 €	10,300.00 €	6,180.00 €	10,300.00 €
20/6/2020	2020	Evolutions & connexion ERP	Développement	35.00	600.00 €	21,000.00 €	10 mois	15,166.67 €	15,166.67 €	7,000.00 €	15,166.67 €
31/12/2020	2020	Maintenance 2020	Maintenance			27,865.00 €	4 mois	24,768.89 €	24,768.89 €	9,288.33 €	24,768.89 €
						125,400.00 €		69,567.22 €			69,567.22 €

L'exemple ci-dessus illustre une application mise en production en 2017.

- La première version a représenté un budget de **65 jours** valorisés au prix du marché de **550 € par jour**
- La seconde version a représenté un budget de **30 jours**

Par la suite, cette application a fait l'objet de travaux de maintenance jusqu'à une nouvelle version consistant en diverses évolutions, notamment la connexion de l'application à un ERP (système de gestion de l'activité de l'entreprise). Nous constatons qu'à la suite de la livraison de cette nouvelle version, un **budget de maintenance important** a été engagé. Nous pouvons imaginer que la connexion à l'ERP a posé problème, peut-être a-t-elle été réalisée par l'éditeur de l'ERP et non par l'équipe de développement initiale. **Les travaux ont pu être sous-estimés**, mal réalisés. Toujours est-il que la connexion de l'application à l'ERP a nécessité des développements complémentaires significatifs.

Finalement, le développement et la maintenance de l'application ont coûté **125 400 € au total**.

VALORISER LES COÛTS

La valeur de votre application **diminue au fil du temps**.

Pour constater cette dépréciation, nous retenons un **amortissement linéaire** sur trois ans avec u**ne valeur résiduelle de 33%**. Cela revient à considérer que l'application perd un tiers de sa valeur au bout d'un an et un deuxième tiers au bout de 2 ans. **La valeur minimum de l'application est du tiers de son coût**.

Vous pouvez effectuer ce calcul en comparant les dates de livraison en production avec la date à laquelle vous souhaitez calculer la valorisation.

Tableau d'amortissement linéaire sur 3 ans

Amortissement linéaire 3 ans. Valeur résiduelle = 1/3 de la valeur initiale										
Valorisation à la date du	15/5/2021									
Mise en prod	**Période**	**Objet**	**Type**	**Nb jours**	**Coût / jour**	**Coût**	**Ancienneté**	**Valeur**	**Valeur nette**	**Plancher**	**Retenu**
10/2/2017	2017	Application v1	Développement	65.00	550.00 €	35,750.00 €	51 mois	11,916.67 €	0.00 €	11,916.67 €	11,916.67 €
15/6/2017	2017	Application v2 finale	Développement	30.00	550.00 €	16,500.00 €	47 mois	5,500.00 €	0.00 €	5,500.00 €	5,500.00 €
31/12/2017	2017	Maintenance 2017	Maintenance			2,480.00 €	40 mois	826.67 €	0.00 €	826.67 €	826.67 €
31/12/2018	2018	Maintenance 2018	Maintenance			3,265.00 €	28 mois	1,088.33 €	0.00 €	1,088.33 €	1,088.33 €
31/12/2019	2019	Maintenance 2019 + techno	Maintenance			18,540.00 €	16 mois	10,300.00 €	10,300.00 €	6,180.00 €	10,300.00 €
20/6/2020	2020	Evolutions & connexion ERP	Développement	35.00	600.00 €	21,000.00 €	10 mois	15,166.67 €	15,166.67 €	7,000.00 €	15,166.67 €
31/12/2020	2020	Maintenance 2020	Maintenance			27,865.00 €	4 mois	24,768.89 €	24,768.89 €	9,288.33 €	24,768.89 €
						125,400.00 €		69,567.22 €			69,567.22 €

TELECHARGER NOTRE MODELE DE TABLEAU D'EVALUATION

Dans l'exemple ci-dessus, identique au point précédent :

- La première version de l'application a été mise en production en **février 2017**, soit **51 mois avant la date de valorisation**. Au-delà de 24 mois, nous prenons en compte sa valeur résiduelle, soit 1/3 de son coût (35 750 € / 3 = **11 916,67 €**).
- Les évolutions & connexion ERP ont été livrées en juin 2020, soit **10 mois avant la date de valorisation**. Sur cette période, un amortissement linéaire sur 3 ans représente 21 000 € (coût des évolutions) x 1/3 x 10/12 mois = 5 833,33 €. La valeur est en conséquence de 21 000 € - 5 833,33 € = **15 166,67 €**.

Au total, nous obtenons une valorisation de notre application pour **un montant proche de 70 000 €**. Bien évidemment, vous pouvez retenir une méthode de calcul différente qui prendra en compte les spécificités de votre application et son contexte.

Le recensement des coûts de l'application fait **la distinction entre les coûts de développement et les coûts de maintenance**. Ce sont ces derniers qui vont nous intéresser pour établir un ratio entre le coût de maintenance et la valeur de l'application.

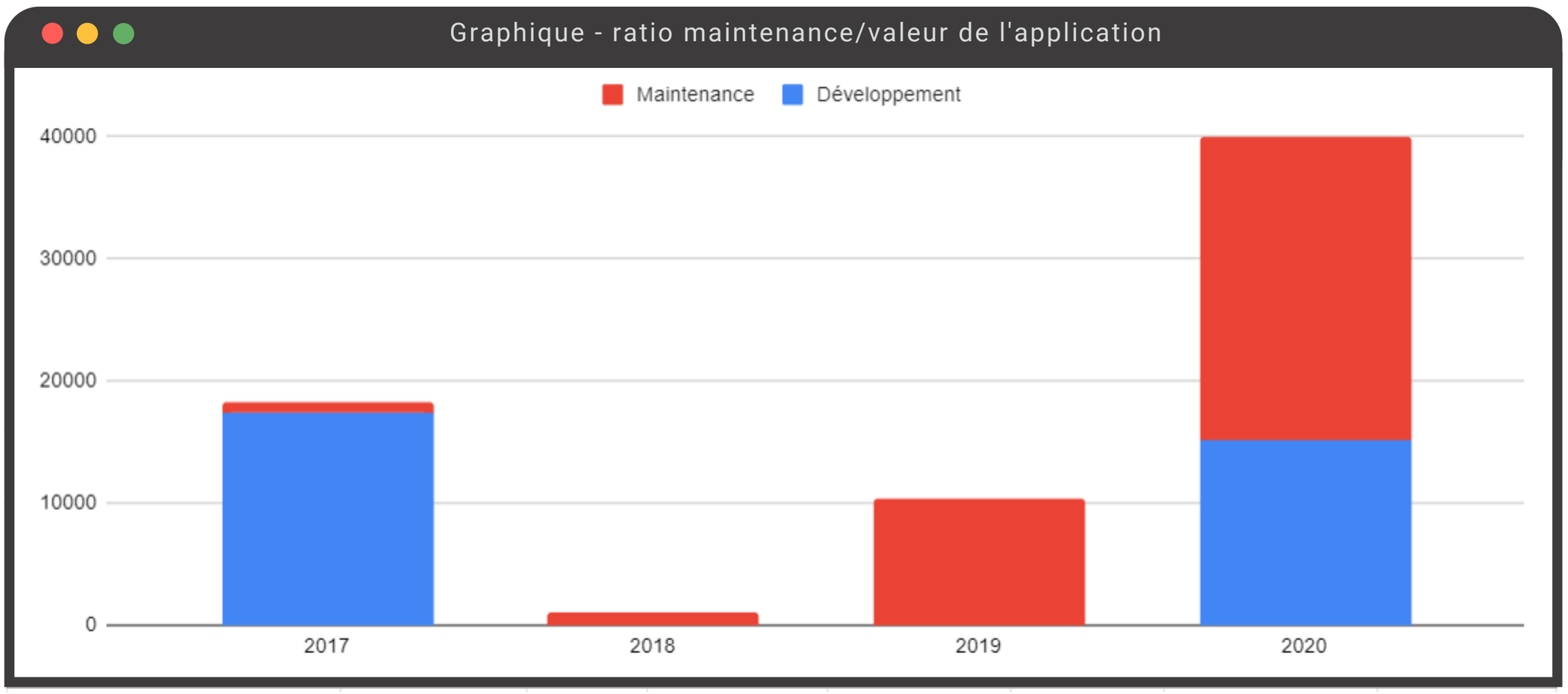
Ce ratio est un **indicateur très révélateur d'obsolescence**. Plus les coûts de maintenance sont élevés par rapport à la valeur de l'application, plus celle-ci est obsolète. Il n'est pas rare que certaines organisations dont les systèmes d'information sont anciens dépensent **80% de leur budget informatique en maintenance**, ce qui diminue d'autant les possibilités de développement de nouvelles applications ou de nouveaux services. Nous pouvons considérer qu'**une application dont les coûts de maintenance dépassent sa valeur est à coup sûr obsolète**.

Dans notre exemple, le ratio maintenance/valeur de l'application est de 53,16%, ce qui n'est plus négligeable. En y regardant de plus près, nous pouvons constater que les coûts de maintenance, non seulement sont élevés, mais de surcroît progressent de façon importante au fil du temps.

Calcul du ratio maintenance/valeur de l'application

Valeur totale	**69,567.22 €**	
Développement	32,583.33 €	46.84%
Maintenance	36,983.89 €	53.16%

SUM of Retenu	*Type*		
Période	Développemer	Maintenance	Grand Total
2017	17,416.67 €	826.67 €	18,243.33 €
2018		1,088.33 €	1,088.33 €
2019		10,300.00 €	10,300.00 €
2020	15,166.67 €	24,768.89 €	39,935.56 €
Grand Total	**32,583.33 €**	**36,983.89 €**	**69,567.22 €**



Il est essentiel de **comprendre pourquoi**. Parmi les raisons les plus fréquentes : des **spécifications de mauvaise qualité** et/ou qui ne sont **pas mises à jour**, **pas de tests** fonctionnels automatisés, des **technologies obsolètes** et difficiles à mettre à jour, une **dette technique** importante, des problèmes de **qualité du code** etc.

Dans notre exemple, nous pouvons imaginer que c'est la compatibilité technique entre l'application originelle et l'ERP auquel il a été connecté qui pose problème. Il a fallu reprendre une bonne partie du code, et donc perdre du temps et de l'argent, pour que cela fonctionne. Il ne faut pas courir le risque de voir les coûts de maintenance continuer à augmenter, ce qui traduirait **une augmentation de l'obsolescence de l'application**. Auquel cas il sera trop tard pour intervenir, **l'application devra être abandonnée et remplacée** par une autre solution.

Nous n'en sommes pas encore à ce stade. En tout cas, il n'est pas trop tard pour chiffrer les travaux nécessaires à la pérennisation de l'application.

CHIFFRER LE COÛT DES ÉVOLUTIONS

Si nous souhaitons une application pérenne, qui soit beaucoup moins ou pas du tout obsolète, combien cela coûterait-il ? S'agissant de développement sur-mesure, **l'estimation à réaliser porte principalement sur la charge de travail**. Il se peut aussi qu'il faille faire l'acquisition de composants logiciels.

Schématiquement, deux options peuvent être envisagées : **la mise à niveau** de l'application existante ou son **remplacement**. Une troisième solution consiste à **isoler l'application**, ce qui est une solution provisoire qui permet le plus souvent de parer à un problème de compatibilité de composants techniques.

Dans l'exemple que nous suivons au long de ce Livre Blanc, nous chiffrons **le coût de mise à jour** de l'application d'une part, et **le coût de remplacement** de l'application d'autre part.

Le coût de mise à jour est envisagé à iso-fonctionnalité : dans notre exemple, les fonctionnalités existantes sont les bonnes, il n'est pas nécessaire d'en ajouter ni d'en retirer. L'évaluation du coût de mise à jour dépend du résultat d'autres phases d'audit de l'application :

- il est d'abord nécessaire de **mettre à jour les spécifications fonctionnelles et techniques** pour repartir sur de bonnes bases
- si nous avons constaté que des **tests fonctionnels automatisés** ne sont pas en place, il faudra prévoir de les implémenter
- une analyse de code statique réalisée par un outil du marché nous a fourni une évaluation de **dette technique** (principalement de qualité du code) à hauteur de 15 jours de travail

- **l’inventaire des composants techniques** de l’application nous a montré que certains sont obsolètes ou en voie de l’être, avec des dates de fin de support à long terme qui sont proches voir dépassées ; il faut prévoir de mettre à jour ces composants techniques

- le **développement d'une nouvelle API** qui permettra de régler durablement le problème de connexion avec l’ERP. Pour cela nous prévoyons de développer une nouvelle API, c'est-à-dire un système qui permettra une communication robuste entre l’application et l’ERP.

Le total représente un budget estimé à près de **61 000 euros**. Ceci sans compter les coûts spécifiques de l’ERP (licences, hébergement… etc.)

Calcul du coup de mise à niveau de l'application

Coût de mise à niveau iso fonctionnalité	Nb jours	Coût / jour	Coût
Rétro conception fonctionnelle et technique	6	600,00 €	3 600,00 €
Implémentation de tests fonctionnels automatisés	20	600,00 €	12 000,00 €
Résolution de la dette technique	15	600,00 €	9 000,00 €
Mise à niveau des composants techniques	35	600,00 €	21 000,00 €
Développement d'une nouvelle API	25	600,00 €	15 000,00 €
Total			**60 600,00 €**

Le coût de remplacement est estimé pour la réalisation d’une nouvelle application. Dans notre exemple, il faudra y **intégrer les fonctionnalités de l’ERP** dont nous avons besoin. Nous pouvons imaginer que le choix de cet ERP a été fait pour réduire les coûts de développement sur-mesure et que nous n’avons pas besoin de l’intégralité des fonctionnalités que procure cet outil.

Une première estimation donne le chiffre de **150 jours de développement**, ce qui est légèrement supérieur aux 130 jours passés sur les développements de l’application existante.

Au préalable, il faudra prévoir le temps nécessaire à la réalisation de spécifications fonctionnelles. Cette étape est indispensable pour obtenir un budget de développement réaliste. Il est très fréquent que **la charge de développement des applications soit optimiste et sous-estimée**. Les études réalisées en la matière (notamment par Standish Group) montrent qu’une infime minorité des projets informatiques se termine dans les budgets et les délais prévus.

La principale cause est un **défaut de spécifications** : elles ne sont pas suffisamment précises, parfois incomplètes. Pour pallier ce problème, la bonne solution est de réaliser des spécifications fonctionnelles en bonne et due forme. Elles permettront d’**obtenir une évaluation précise du temps de développement nécessaire** et du budget correspondant.

Au total dans notre exemple, le coût de remplacement fait l’objet d’une première évaluation à **96 000 euros**.

Calcul du coup de remplacement de l'application

Coût de remplacement avec fonctionnalités ERP			
Ateliers et rédaction des spécifications fonctionnelles	10	600,00 €	6 000,00 €
Budget de développement (première estimation)	150	600,00 €	90 000,00 €
Total			**96 000,00 €**

Alors que faire : mettre à jour ou remplacer notre application ?

LE CHOIX STRATEGIQUE : MAINTENANCE OU REMPLACEMENT ?

Sur quels critères, aussi objectifs que possible, effectuer ce choix ? Sur la base des indications financières que nous avons vues plus haut, mais pas seulement.

En principe, la maintenance est la solution à privilégier par rapport au remplacement. En effet, le remplacement est une méthode “révolutionnaire” qui peut être brutale autant que **risquée**. Brutale vis-à-vis des utilisateurs s’ils ne sont pas correctement impliqués dans le projet alors qu’ils vont devoir changer d’outil. **Il faut anticiper les efforts et les coûts de formation et de conduite du changement**.

Risqué, en particulier s’il est nécessaire de maintenir l’historique avec les données de la précédente application : **la migration des données peut tourner au casse-tête** et parfois à la catastrophe lorsqu’il s’agit de faire entrer des choses carrées dans des trous ronds, nombre de projets de migration en ont fait les frais. Pour y pallier, une étude technique spécifique est indispensable.

Dans notre exemple, cela tombe bien car nous avons estimé un coût de maintenance inférieur au coût de remplacement. La solution semble donc toute trouvée : **une modernisation par étapes**, en commençant par le plus urgent, en l’occurrence le travail sur l’API qui permettra de réduire la dépendance vis-à-vis de l’ERP et les coûts de maintenance qui vont avec.

D’autres facteurs de choix doivent être pris en compte, qui sont analysés dans les différentes phases d’audit applicatif que nous mettons en œuvre :

- **l’analyse de la complétude fonctionnelle** de l'application : répond-t-elle correctement aux besoins ?
- **l’état des ressources humaines** : les compétences techniques et métiers sont-elles disponibles et suffisantes ?
- **l’évaluation de l’obsolescence** des technologies de l’application,
- **la mesure de la dette technique** et **l’identification des failles de sécurité**,
- **le degré de couverture des tests**, de la **documentation**, du **processus de développement et de déploiement**.

Tous ces éléments sont à prendre en considération pour choisir la meilleure solution.

A cela s’ajoute un critère sans doute plus déterminant encore, qui porte sur la **stratégie digitale** de l’entreprise à moyen et long terme.

Dans quelle politique digitale se place notre application ? Dans quelle direction doit-elle évoluer pour contribuer avec efficacité à la stratégie informatique de l’entreprise ? La dépendance avec une solution tierce (ici le fameux ERP) est-elle admise, voire nécessaire s’il s’agit par exemple d’une solution déjà choisie pour toute l’entreprise ?

Au final, toutes ces questions peuvent faire pencher la balance vers la solution qui n’est pas forcément la plus économique en termes financiers.

LES SERVICES AXIOCODE

RÉALISEZ UN DIAGNOSTIC RAPIDE EN AUTONOMIE

AxioCode a créé un outil qui vous permet d'obtenir un premier diagnostic de votre application. L'objectif est de mettre en lumière les problématiques qui peuvent lui être liées. Nous vous proposons de mesurer les besoins de modernisation de votre application métier au travers quatre indicateurs :

- **le degré de dette fonctionnelle de votre application**
- **le degré de dette technologique**
- **le degré de dette des tests**
- **le degré de dette technique**

Faites en autonomie un **diagnostic rapide** de votre application. Utilisez notre service **gratuit** de diagnostic dédié aux applications métiers web et mobiles. Vous obtiendrez un rapport de diagnostic de votre application. Le questionnaire est simple et ne requiert pas de compétences techniques. Il ne vous demande que quelques minutes.

Vous souhaitez savoir si votre application est obsolète ?

EVALUER L'OBSOLESCENCE DE MON APPLICATION >

Ce diagnostic et les recommandations qui y sont associées vous donneront **un état de la situation** et les pistes à creuser pour mener l'audit de votre application.

RÉALISER L'AUDIT DE VOTRE APPLICATION

Vous souhaitez réaliser l'**audit de votre application web ou mobile** ?

N'hésitez pas à nous contacter : **le diagnostic et les premiers échanges avec nos experts sont gratuits**. Cette première étape va permettre de définir les premier travaux d'audit à réaliser répondant spécifiquement aux contraintes de votre système d'information et à vos besoins d'évolution.

Suite au résultat de votre diagnostic *(voir page précédente : réalisez votre diagnostic rapide en autonomie)*, les experts d'AxioCode seront en mesure de **définir les travaux d'audit** permettant de répondre à vos problématiques. Nos spécialistes sont en mesure de vous fournir un audit complet et précis de votre application, comprenant tous les points évoqués précédemment, à savoir :

- **un audit fonctionnel**, afin de déterminer si les fonctionnalités métiers clés sont complètes et que les bonnes pratiques sont respectées pour garantir la pérennité fonctionnelle de l'application,

- **un audit technique et RH** comprenant un inventaire technologique, un état des ressources humaines affectées à l'application, une évaluation d'obsolescence des technologies utilisées, une analyse du code source et une liste des failles de sécurité, une évaluation de la couverture des tests, de la documentation, du processus de développement et de déploiement,

- **un audit financier** soit une estimation de la valeur financière de l'application, des délais et des coûts de remplacement, de mise à niveau ou d'isolement.

En fin d'audit, Axiocode vous remettra **un rapport d'audit**. Nous échangeons sur les résultats de l'audit et prenons en compte les contraintes spécifiques à votre système d'information pour vous **présenter nos recommandations** d'implémentation et notre proposition de feuille de route.

PRENDRE RENDEZ-VOUS AVEC UN EXPERT >

DÉCOUVREZ
TOUS NOS OUTILS POUR AUDITER UNE APPLICATION

En cliquant sur le bouton ci dessous vous accéderez aux outils évoqués dans ce Livre Blanc.
Ils ont pour but de vous aider à estimer l'obsolescence de vos applications et de vous guider lors de l'audit de vos applications professionnelles.

ACCEDER AUX OUTILS >

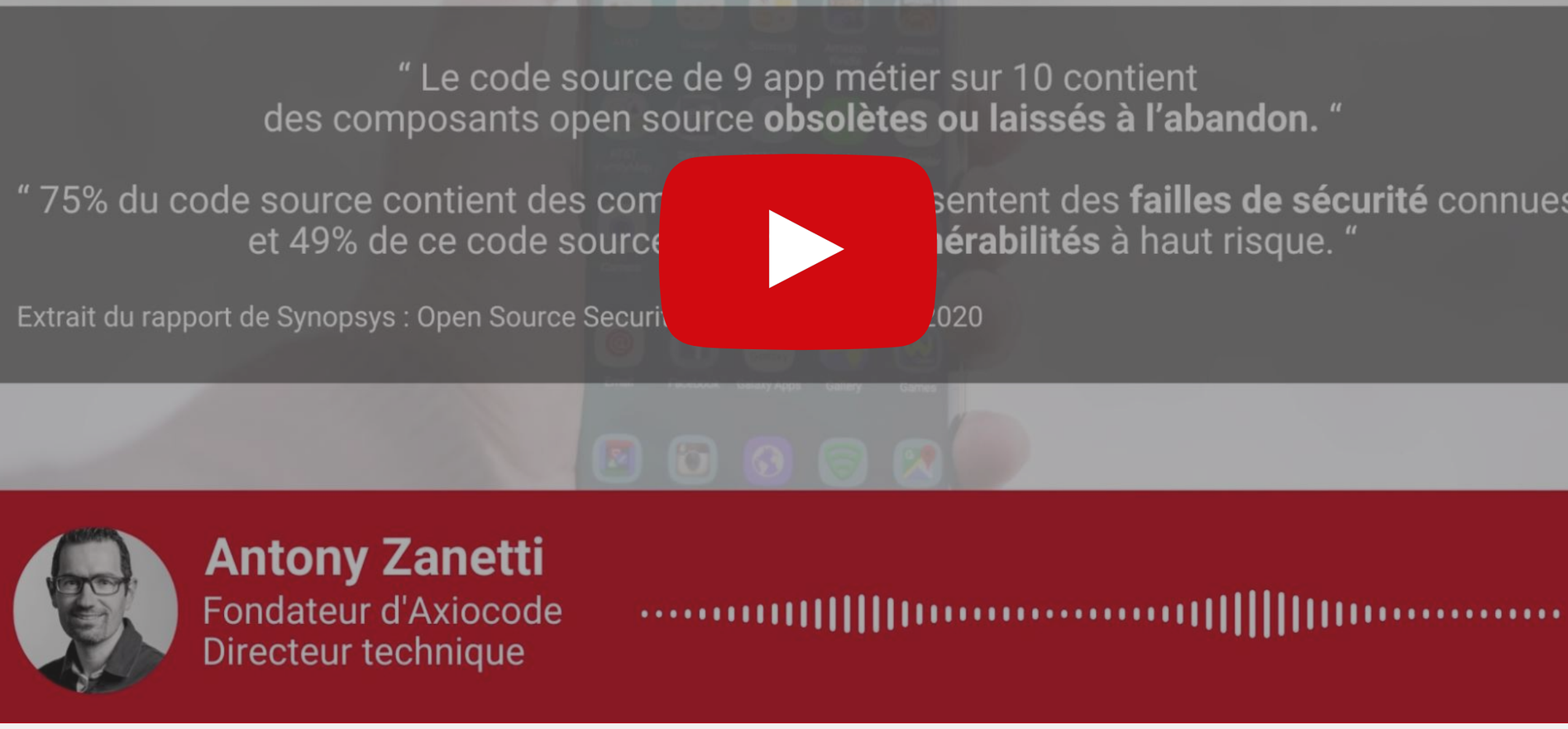
BONUS VIDEO

Besoin d'en savoir plus ?

Découvrez nos vidéos sur l'audit applicatif !

Playlist - Auditer une application web ou mobile (7 étapes)

axiocode
DÉVELOPPEMENT WEB & MOBILE



Antony Zanetti
Fondateur d'Axiocode
Directeur technique

REGARDER LES VIDEOS >

axiocode
DÉVELOPPEMENT WEB & MOBILE

DES APPLICATIONS WEB ET MOBILES SUR-MESURE

ANALYSE

Une analyse précise de vos besoins : nos experts vous proposent la solution optimale en fonction du besoin exprimé.

EXPERTISE

Notre expertise technique : votre solution est 100% sur-mesure et évolutive, avec des fonctionnalités modulables et adaptables.

TESTS

Des tests approfondis : testez votre solution mobile avant-même la livraison définitive.

ILS NOUS FONT CONFIANCE

ALAIN AFFLELOU · ENEDIS L'ELECTRICITE EN RESEAU · FEI FÉDÉRATION ÉQUESTRE INTERNATIONALE · syngenta · CRISTAL UNION

axiocode
DÉVELOPPEMENT WEB & MOBILE

DES APPLICATIONS DE QUALITÉ, DÉLIVRÉES À TEMPS, AU MEILLEUR COÛT

DEVELOPPEMENT WEB ET MOBILE SUR MESURE

Quelle que soit la nature de votre projet : logiciel de gestion sur mesure, plateforme de mise en relation, plateforme de réservation, site web, intranet, application mobile, etc, nous vous accompagnons tout au long de vos projets et vous fournissons des solutions personnalisées, adaptées à vos besoins.

DOMAINES D'EXCELLENCE

- Développement d'applications métiers
- Géolocalisation, cartographie web
- Gestion des flux monétiques
- Plateformes de mise en relation B2B, B2C
- Développement natif Android et Apple
- Systèmes de planification et de réservation
- Application mobiles disponibles en ligne et hors-ligne

PRENDRE RENDEZ-VOUS AVEC UN EXPERT >

axiocode
DÉVELOPPEMENT WEB & MOBILE

NOUS CONTACTER

03 67 67 46 12

contact@axiocode.com

www.axiocode.com

57 rue Lothaire, 57000 Metz

PRENDRE RENDEZ-VOUS AVEC UN EXPERT

NOUS SUIVRE SUR LES RESEAUX SOCIAUX

@axiocode