AT INTERNET

Stratégie de test

Date	Version	Commentaire	Auteur
29/09/2015	0.1	Première version de travail	Alexandre AUBERT
16/10/2015	1.0	Relecture / validation	Vincent LATASTE Claire ALLALI
06/04/2017	1.1	Nouvelle section : Flux des bugfixes Nouvelle section : Flux de développement Update : Communication et visibilité	Alexandre AUBERT Vincent LATASTE Stéphane PERRIN
19/06/2017	1.2	Nouvelle section : Suivi de la production	Equipe Qualité Vincent LATASTE
12/09/2019	1.3	Nouvelle section : Tests liés à la sécurité Suppression de la section : Suivi de la production Update : Flux de développement (nouveau visuel) Update : Communications et visibilité (nouveaux indicateurs liés aux tests / retrait des informations liées aux Changelogs)	Alexandre AUBERT Vincent LATASTE Nicolas BOUDILLON
02/06/2021	1.4	Update : III. Niveaux de test (précisions sur les tests « boite noire » : système, intégration, end2end) Update : V. Validation/intégration continue (préprod : env de release, recette/e2e sur l'integ, tests liés à la sécurité) Nouvelle section : VIII. Visibilité Update : IX. Communication	Alexandre AUBERT Vincent LATASTE Erwan LOAEC
25/01/2022	1.5	Update : précision sur les niveaux de test, responsabilités et stratégies d'investissement Ajout : annexes et glossaire	Alexandre AUBERT Vincent LATASTE

I. Objet de ce document

La stratégie de test décrit la méthodologie générale des tests chez AT Internet. Cela inclut la façon d'utiliser le test pour gérer les risques produit et les risques projet, la séparation en niveaux de test et les activités de haut niveau associées au test. Les responsabilités ainsi que les ressources disponibles pour les tests sont également décrites dans ce document.

La stratégie sert de document de référence et donne un cadre stable pour les prises de décisions concernant les activités de test. Elle est en accord avec la politique de test AT Internet et peut être revue et modifiée aussi souvent que nécessaire afin de refléter au mieux la réalité des tests dans l'entreprise.

II. La stratégie de test AT Internet

La stratégie de test AT Internet s'articule en deux grandes approches :

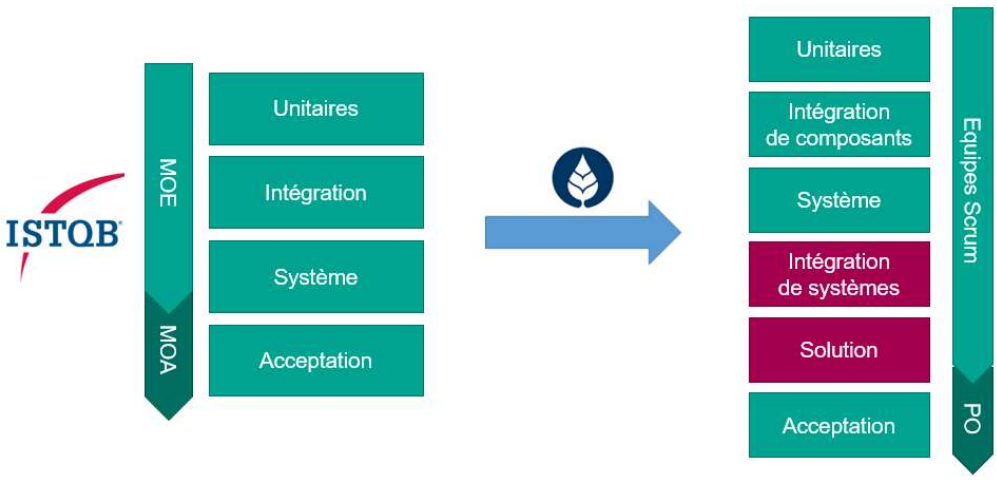
- Une stratégie basée sur le processus de développement Agile : à chaque itération les équipes analysent les user stories décrivant des fonctionnalités particulières, estiment l'effort de test pour chaque fonctionnalité lors du processus de planification de l'itération, identifient pour chaque user story les critères d'acceptation, exécutent les tests couvrant ces conditions.

- Une stratégie réactive et basée sur l'expérience visant à améliorer la qualité des solutions en production. Cette stratégie s'appuie sur les retours d'exploitation et l'expérience des équipes pour prioriser, définir, concevoir et implémenter de nouveaux tests.

Le test manuel est tout aussi valorisé que le test automatisé, suivant le contexte considéré. L'automatisation des tests est effectuée après analyse du rapport entre le gain généré par l'exécution automatique et continue de certains tests et l'effort à fournir pour la mise en place et la maintenance de ces automatismes.

III. Les niveaux de tests

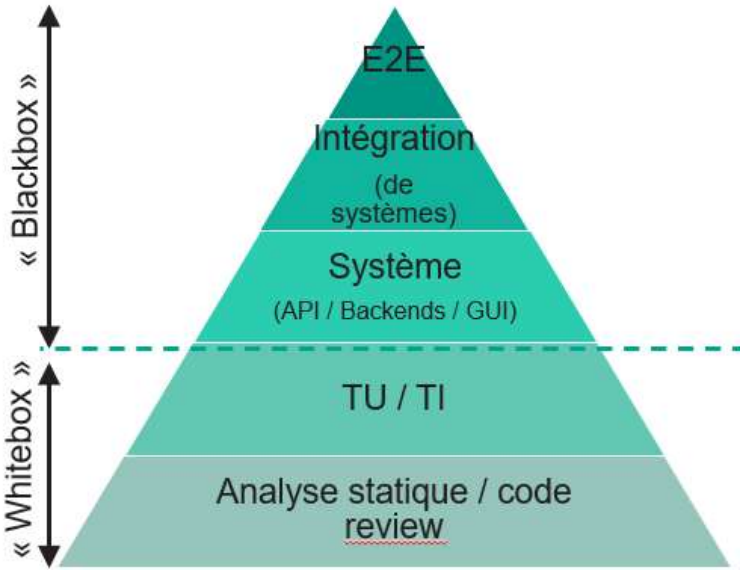
Compte tenu de l'ampleur de notre solution logicielle et du nombre de systèmes qui la composent, nous avons fait le choix, chez AT Internet, d'avoir deux niveaux de tests supplémentaires (comparé aux niveaux standards ISTQB) : les tests d'intégration de système et les tests de solution. Tous les niveaux de test sont détaillés dans cette section.



A. Boite noire / boite blanche

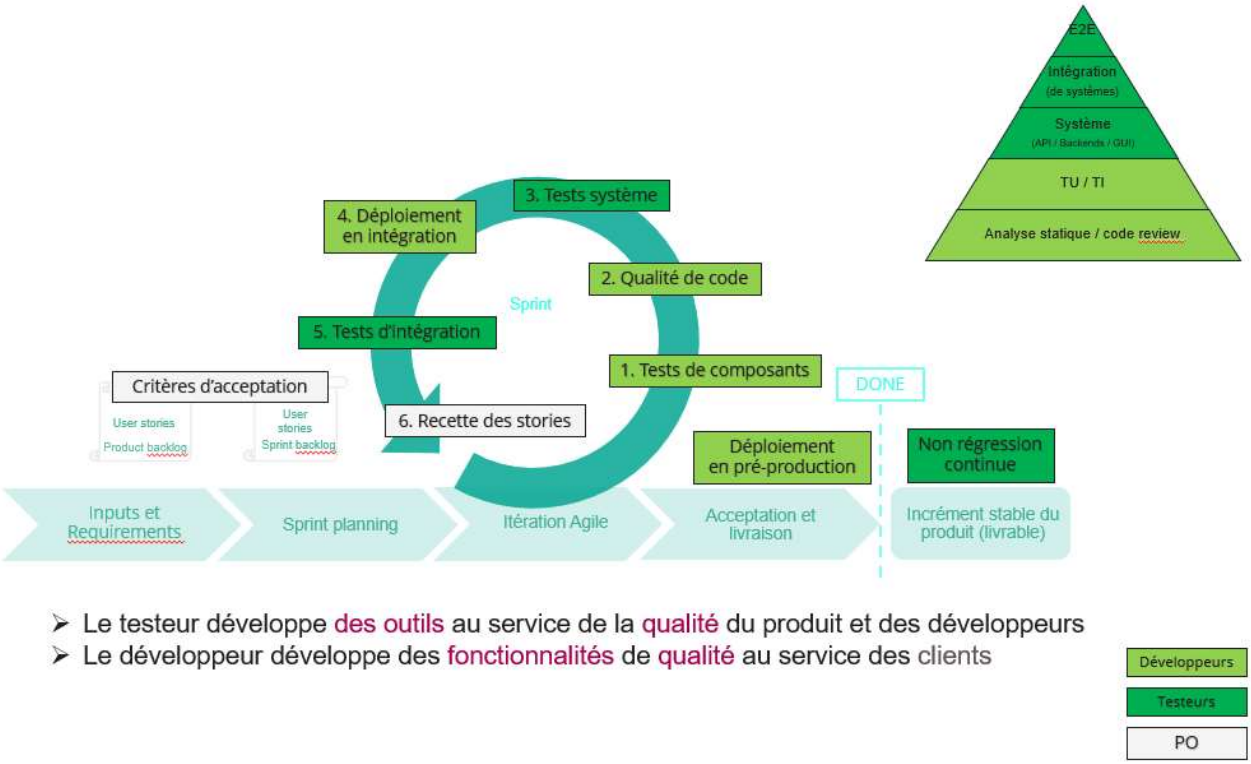
Les tests ayant accès au code sont dits « boite blanche » : ils ne nécessitent pas de déploiement du produit pour être exécutés.

A l'inverse, les tests des niveaux supérieurs, dits « boite noire », sont complètement indépendants du code et sollicitent le produit sous forme d'un livrable déployé sur un environnement. Ces tests garantissent les aspects métier fourni à l'utilisateur en cas de refacto importante, changement de technologie etc...



B. Rôles et responsabilités

Chaque membre de l'équipe est responsable de l'ensemble des tests à mener sur le produit, pour assurer un niveau de qualité acceptable, en accord avec la stratégie de test définie sur chaque projet. Cependant, chaque membre de l'équipe, en fonction de son métier, est naturellement plus proche de certaines activités de test :



C. Stratégie d'investissement

Tests « boite blanche » obligatoires

Les tests « boite blanche » : tests unitaires et sonar (=analyse statique de code) sont obligatoires partout et sont de la responsabilité principale des développeurs.

Nous ne voulons aucune impasse sur ces deux niveaux de test, quel que soit le coût que cela peut représenter. C'est ce que nous attendons de nos développeurs.

Des manques sur ces investissements seraient en effet très lourds à assumer sur la suite du projet : dette technique, régressions multiples (TDB par ex.), besoin de refactoring profond/refonte totale…

Approche stratégique des investissements en tests « boite noire »

Les tests « boite noire » sont constitués de 3 niveaux (chez AT) :

- Les tests système (tests d'API, Interface (cypress/noctua) par ex) : l'objectif est de couvrir au maximum le métier des stories à ce niveau de test. Ils sont de la responsabilité principale des testeurs mais certaines équipes n'ont pas de testeur et mettent tout de même en place ce genre de tests (ex : ESP, exports…). L'absence de ressources de test suffisantes dans les équipes (avec ou sans testeur d'ailleurs) impose parfois tout de même de faire des choix et de ne pas couvrir certains systèmes avec ce genre de test.

- Les tests d'intégration de systèmes : ces tests sollicitent les produits de plusieurs équipes. Ils sont mis en place suivant les projets et sont de la responsabilité conjointe des équipes concernées (ex : tests d'imports entre DM/Storage). Ils peuvent être coûteux à mettre en place, à surveiller et à maintenir.

- Les tests de solution (end 2 end) : ils simulent l'activité du client, de bout en bout. Ils sont mis en place suivant les projets et sont de la responsabilité conjointe des équipes concernées (ex : tests Sales

Insights : des events reçus aux interfaces). Ils peuvent être coûteux à mettre en place, à surveiller et à maintenir.

- ⇨ L'approche stratégique de chaque projet par le PO vise à définir où et comment on souhaite investir sur ces 3 niveaux de test « boite noire », compte tenu des risques, de l'architecture du projet et des ressources impliquées sur le projet.

Les sections suivantes décrivent plus en détail chaque niveau de test.

D. Test unitaire

Objectifs

Le test unitaire (ou test de composant) est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel, d'une portion d'un programme, objet ou classe testable séparément.

Les tests unitaires sont destinés à vérifier tant les fonctionnalités que les caractéristiques non fonctionnelles. Ils sont utilisés pour s'assurer que les composants individuels fonctionnent correctement.

Rejoués de façon continue, ils fournissent rapidement l'information d'apparition d'une éventuelle régression en cas de changement effectué dans le code.

Etendue

Les tests unitaires sont mis en place sur tous les nouveaux développements et sur les développements de maintenance autant que possible. Ils sont également étendus dans le cadre des corrections de défauts.

Critères d'entrée

- ✓ Les stories définissent clairement ou permettent de définir clairement le comportement attendu pour chacun des composants à implémenter

Critères de sortie

- ✓ Les tests unitaires sont exécutés de façon continue sans erreur
- ✓ Aucune régression n'est constatée en se basant sur les tests en place

Technique de test

Les tests unitaires peuvent être basés sur la structure du code (tests boîte blanche) ou sur les exigences (tests boîte noire). Des bouchons, pilotes ou simulateurs sont souvent utilisés. Les tests doivent être faciles à implémenter et avoir une structure simple : un test unitaire complexe à mettre en place met souvent en évidence une complexité trop importante du code.

Les tests unitaires doivent être clairement nommés : leur nom doit refléter ce que fait le test afin de permettre une compréhension rapide du rôle des tests présents et faciliter l'analyse et l'estimation de risque en cas d'erreur.

Les défauts trouvés sont corrigés dès qu'ils sont détectés.

Le TDD (Test Driven Development) est une pratique privilégiée dans le cadre des tests unitaires.

Ressources

On bénéficie de l'environnement de développement ou du poste de travail de chaque développeur pour la conception et l'exécution des tests unitaires (framework de test, harnais, bouchons...).

Jenkins assure l'exécution continue des tests unitaires, déclenchée à chaque changement dans le code source.

Responsabilités

Ces tests sont conçus, implémentés et exécutés par le développeur qui écrit le code ou qui possède un profil similaire (dans le cas du pair programming, par exemple).

E. Test d'intégration des composants

Objectifs

Les tests d'intégration de composants visent à valider les interfaces et les interactions entre les différents composants développés par une même équipe. Ils sont exécutés après les tests unitaires et s'intéressent à la communication entre les composants. Ces tests peuvent être fonctionnels ou non.

Etendue

Les tests d'intégration sont mis en place dès que plusieurs composants développés par une même équipe sont réunis et amenés à communiquer entre eux. Ils sont également alimentés lors de la détection de défauts liés à l'intégration des composants.

Critères d'entrée

- ✓ Les tests unitaires sont passés avec succès sur chacun des composants et sont exécutés de façon continue.
- ✓ Au moins deux composants devant échanger entre eux sont disponibles
- ✓ Les interactions entre composants et les flux de données sont définies et stables

Critères de sortie

- ✓ Les composants sont intégrés les uns aux autres et tous les messages sont échangés
- ✓ Les tests d'intégration de composants sont exécutés de façon continue
- ✓ Aucune régression n'est constatée en se basant sur les tests en place

Technique de test

Les tests d'intégration de composants peuvent être basés sur la structure du code, les documents d'architecture (tests boîte blanche) ou sur les exigences (tests boîte noire). La manière d'intégrer les composants peut varier suivant la granularité et la possibilité d'isolation de ces composants et suivant le périmètre de chaque équipe (par fonctionnalité, par voisinage, big-bang, backbone…)

Les tests d'intégration doivent être clairement nommés : leur nom doit refléter ce que fait le test afin de permettre une compréhension rapide du rôle des tests présents et faciliter l'analyse et l'estimation de risque en cas d'erreur.

Le TDD (Test Driven Development) pourra être utilisé dans le cadre des tests d'intégration des composants.

On pourra noter que certains tests unitaires utilisant des bouchons (ou mocks) peuvent devenir directement des tests d'intégration en remplaçant simplement le bouchon par le composant réel qu'il remplace.

Ressources

Les tests d'intégration de composants sont exécutés sur les environnements de développement et leur exécution est automatisée dans Jenkins afin de s'assurer de la non régression des intégrations de composants.

Responsabilités

Les tests d'intégration sont effectués dans les équipes, par les développeurs ou par les testeurs en collaboration avec les développeurs.

F. Test système

Les tests au niveau système constituent le premier niveau de tests dits « boite noire ». Ils sont exécutés sur le système déployé, observé et sollicité depuis l'extérieur.

Objectifs

Le test système vise à valider le comportement du système complet, d'un point de vue fonctionnel ou non fonctionnel. Les documentations, procédures d'installation, de déploiement ainsi que la configuration font également partie du périmètre. Ce niveau de test se limite au périmètre de livraison d'une équipe, les tests de l'ensemble de la solution AT Internet étant effectués au niveau dit 'solution'.

Etendue

Les tests système sont mis en place systématiquement afin de couvrir les caractéristiques telles que définies dans les user stories. Ils peuvent également être alimentés lors des phases de débogage par les tests de confirmation.

Critères d'entrée

- ✓ L'intégration de l'ensemble des composants livrés par l'équipe a été effectuée avec succès
- ✓ Les exigences fonctionnelles et/ou non fonctionnelles du système doivent être clairement exprimées et disponibles

Critères de sortie

- ✓ La couverture des exigences (critères d'acceptation) est complète
- ✓ La traçabilité entre tests système et les user stories du backlog de l'équipe est disponible
- ✓ Les défauts à corriger ont été corrigés et validés par du test de confirmation
- ✓ Les impacts des défauts à ne pas corriger sont évalués
- ✓ Aucune régression n'a été constatée par les systèmes en place

Technique de test

Lors des tests système, on s'intéresse à la fois aux aspects fonctionnels et non fonctionnels. On utilise principalement les techniques de test 'boite noire', sans avoir connaissance de la structure interne du système. Parmi les techniques les plus courantes, nous pouvons citer :

- Partitions d'équivalence
- Analyse des valeurs limites
- Tables de décisions
- Test des transitions d'état
- Cas d'utilisation

L'ATDD (Acceptance Test Driven Development) et/ou le BDD (Business Driven Development) pourront être des pratiques utilisées dans le cadre des tests système.

Ressources

Le test système est effectué sur l'environnement de développement stable de l'équipe.

Responsabilités

Les tests système sont effectués par l'équipe qui produit le système, par les développeurs et/ou les testeurs.

G. Test d'intégration de systèmes (intégration inter-équipes)

Objectifs

Les tests d'intégration de systèmes visent à valider les interfaces et interactions entre les différents systèmes produits par les équipes. Ils sont exécutés une fois les systèmes individuels validés par les niveaux de test inférieurs.

Etendue

Les tests d'intégration de systèmes sont mis en place dès que deux systèmes (le plus souvent impliquant deux équipes différentes) ont des interactions, s'échangent des messages ou des données. Ils sont également alimentés lors de la détection de défauts liés à l'intégration des systèmes.

Critères d'entrée

- ✓ Au moins deux systèmes devant échanger entre eux sont disponibles
- ✓ Les tests système sont passés avec succès sur chacun des systèmes
- ✓ Les interactions entre systèmes et les flux de données sont définis et stables

Critères de sortie

- ✓ Les systèmes sont intégrés les uns aux autres et tous les messages sont échangés
- ✓ Les tests d'intégration de systèmes sont exécutés de façon continue
- ✓ Aucune régression n'est constatée en se basant sur les tests en place

Technique de test

Les tests d'intégration de systèmes peuvent être basés sur la structure du code, les documents d'architecture (tests boîte blanche) ou sur les exigences (tests boîte noire). La manière d'intégrer les systèmes peut varier suivant la granularité et la possibilité d'isolation de ces systèmes. On privilégiera la méthode dite 'par voisinage' qui permet d'intégrer les systèmes deux à deux en tenant compte de leur proximité technique.

On pourra noter que certains tests systèmes utilisant des bouchons (ou mocks) peuvent devenir directement des tests d'intégration de systèmes en remplaçant simplement le bouchon par le système réel qu'il remplace.

Ressources

L'environnement d'intégration sera utilisé par les équipes pour déployer leurs systèmes validés et procéder aux tests d'intégration de systèmes. Ces tests sont automatisés dans Jenkins, dans une démarche d'intégration continue.

Responsabilités

Les tests d'intégration de systèmes sont de la responsabilité conjointe des équipes concernées. Les différentes équipes sont amenées à travailler ensemble sur les tâches qui y sont associées.

H. Test de solution (end to end)

Les tests de solutions sont couramment appelés 'tests end to end' car l'utilisateur final est directement présent aux deux extrémités du test. Ces tests sont menés pendant la phase dite 'de recette', qu'elle soit automatisée ou non.

Objectifs

Le test au niveau solution vise à détecter les éventuelles défaillances dans la solution AT Internet. Il s'assure également de la conformité aux exigences et spécifications du produit dans son ensemble d'un point de vue fonctionnel et non fonctionnel. Les documentations, procédures d'installation, de déploiement ainsi que la configuration font également partie du périmètre.

Etendue

Les tests de solution sont mis en place dès qu'une nouveauté ou un changement est apporté à la solution. Ils peuvent également être alimentés lors des phases de débogage par les tests de confirmation.

Critères d'entrée

- ✓ La solution complète est disponible
- ✓ L'intégration des systèmes est terminée
- ✓ Les exigences fonctionnelles et/ou non fonctionnelles de la solution doivent être clairement exprimées et disponibles

Critères de sortie

- ✓ Les exigences définies dans les stories sont couvertes
- ✓ Les défauts à corriger ont été corrigés et validés par du test de confirmation
- ✓ Les informations de densité de défauts par système sont disponibles
- ✓ Les impacts des défauts à ne pas corriger sont évalués

Technique de test

Lors des tests solution, on s'intéresse à la fois aux aspects fonctionnels et non fonctionnels. On utilise principalement les techniques de test 'boite noire', sans avoir connaissance de la structure interne du système. Parmi les techniques les plus courantes, nous pouvons citer :

- Partitions d'équivalence
- Analyse des valeurs limites
- Tables de décisions
- Test des transitions d'état
- Cas d'utilisation

L’aspect métier est également considéré lors de sessions de test exploratoires. Ces sessions peuvent être orientées par fonctionnalité, cas d’utilisation métier ou caractéristique (performance, sécurité, stabilité…).

L’ATDD (Acceptance Test Driven Development) et/ou le BDD (Business Driven Development) pourront être des pratiques utilisées dans le cadre des tests solution.

Ressources

Les tests de solution sont effectués sur l’environnement d’intégration (ou de pré-production dans certains cas).

Responsabilités

Les tests de solution sont assurés par les équipes de développement. L’objectif est d’automatiser leur exécution pour s’assurer de l’absence de régression au cours du temps des cas d’utilisation client de la solution. Cependant, tout le monde peut ponctuellement prendre part à des sessions de test exploratoires durant cette phase de test, lorsqu’un complément par du test manuel s’avère nécessaire pour assurer la qualité de la livraison. (ex : « testa fiesta », recette manuelle)

I. Acceptation

Aujourd’hui, ces tests sont menés en même temps que les tests de solution. Ce niveau de test supplémentaire pourra être envisagé plus en détail dans l’avenir, en fonction de notre niveau de maturité sur les phases précédentes et du besoin, par exemple, de mise en place d’une plateforme beta à disposition des clients.

Objectifs

Le test d’acceptation vise à obtenir l’acceptation du système dans son ensemble par le client, l’utilisateur ou son représentant. Les éléments fonctionnels et non fonctionnels, les documentations, procédures d'installation, de déploiement ainsi que la configuration font également partie du périmètre. On ne recherche pas de défauts à ce niveau de test mais simplement à assurer un niveau de confiance élevé.

Etendue

Les tests d’acceptation ont lieu à chaque fois qu’une nouveauté ou un changement métier dans la solution est mis en place. Ils peuvent également être alimentés lors des phases de débogage afin d’étendre leur couverture du métier.

Critères d'entrée

- ✓ La solution a été testée et validée
- ✓ Les dernières corrections ont été implémentées et testées
- ✓ Le niveau de maturité de la solution est considéré comme suffisant pour la livraison

Critères de sortie

- ✓ Les représentants des utilisateurs ayant participé aux tests acceptent la mise en production de la solution

Technique de test

Les tests d'acceptation sont principalement de type 'boîte noire', basés sur les exigences métier telles que définies dans les user stories. Suivant le contenu des exigences, ils peuvent également être menés en collaboration avec les administrateurs du système pour vérifier certains points d'ordre opérationnel.

Ressources

Les tests d'acceptation sont menés sur l'environnement de pré-production ou sur une plateforme 'beta', à disposition des utilisateurs.

Responsabilités

Les tests d'acceptation sont menés par représentants des utilisateurs ou les utilisateurs eux-mêmes, pouvant être accompagnés de testeurs.

IV. Tests liés à la sécurité

A. Analyse de code statique (SAST)

Objectifs

L'analyse statique vise à détecter les défauts et les vulnérabilités potentiels dans nos produits. Elle est déclenchée après chaque ajout ou modification du code et se base sur les règles de codage en vigueur (par langage).

Etendue

L'analyse de code est effectuée sur l'ensemble de la base de code AT Internet.

Critères d'entrée

- ✓ Le nouveau code est disponible

Critères de sortie

- ✓ Le code a passé avec succès l'analyse de code statique
- ✓ Toutes les vulnérabilités détectées ont été prises en compte

Technique de test

- « SonarQube » (outil tiers) est utilisé pour automatiser les analyses de code statiques
- Les (CWE, SANS Top 25, OWASP top 10) sont prises en compte
- La barrière qualité « AT Internet default QG » doit être appliquée à tous les nouveaux projets (ou une barrière plus restrictive, au choix de l'équipe). Les conditions minimales à respecter sont :
    - Tout le projet
        - Sécurité = A
    - Nouveau code uniquement
        - Pas de bugs
        - Score de maintenabilité = A
        - Pas de défauts bloquants ou critiques (mauvaises pratiques)

Ces conditions devront s'appliquer sur la totalité du code, suivant un calendrier établi par l'équipe en fonction des ressources qu'elle pourra allouer.

- Les vulnérabilités sont revues et communiquées régulièrement par les équipes de développement elles-mêmes.
- Les équipes de développement doivent solliciter l'équipe Sécurité ou le Responsable de tests pour statuer sur l'acceptation d'un risque lié à une vulnérabilité détectée.
- Un linteur local (SonarLint, ESLint…) peut aussi être utilisé pour détecter ces vulnérabilités pendant l'écriture du code

Ressources

Le serveur SonarQube est disponible en interne pour recevoir le code à analyser.

Responsabilités

L'analyse statique de code est déclenchée et les résultats pris en compte par le développeur qui produit le code ou par quelqu'un ayant un profil similaire (en cas de pair programming par exemple).

B. Contrôle des dépendances (process en évaluation)

Ce type de test fait partie de nos objectifs liés à la sécurité. Les équipes de développement sont invitées, dans un premier temps, à se familiariser avec ce type d'outils.

Objectifs

Le contrôle des dépendances vise à détecter, dans nos produits, des vulnérabilités potentielles introduites par une ou plusieurs dépendances externes.

Etendue

Le contrôle des dépendances est effectué sur tous les produits AT Internet, dans la limite des possibilités d'analyse des outils utilisés :

- « Dependency check » : https://jeremylong.github.io/DependencyCheck/analyzers/index.html
- Yarn audit (projets NodeJS)

Critères d'entrée

- ✓ Des dépendances externes sont identifiées dans un produit AT Internet

Critères de sortie

- ✓ Le produit a passé avec succès le contrôle de dépendances
- ✓ Toutes les vulnérabilités liées aux dépendances ont été prises en compte

Technique de test

- OWASP dependency check (outil tiers) est utilisé pour automatiser le contrôle des dépendances
- « Yarn audit » est utilisé pour automatiser le contrôle des dépendances des projets éligibles (plus précis que OWASP dependency check)
- Ces outils répondent partiellement au problème OWASP Top 10 2017 A9-Using Components with Known Vulnerabilities en détectant la présence de vulnérabilités connues, publiquement divulguées dans nos produits.

- Les vulnérabilités sont revues et communiquées régulièrement par les équipes.

Ressources

Le contrôle de dépendances peut être exécuté directement sur les slaves Jenkins, pendant la phase de build ou par une mécanique indépendante de la phase de build, exécutée régulièrement.

Responsabilités

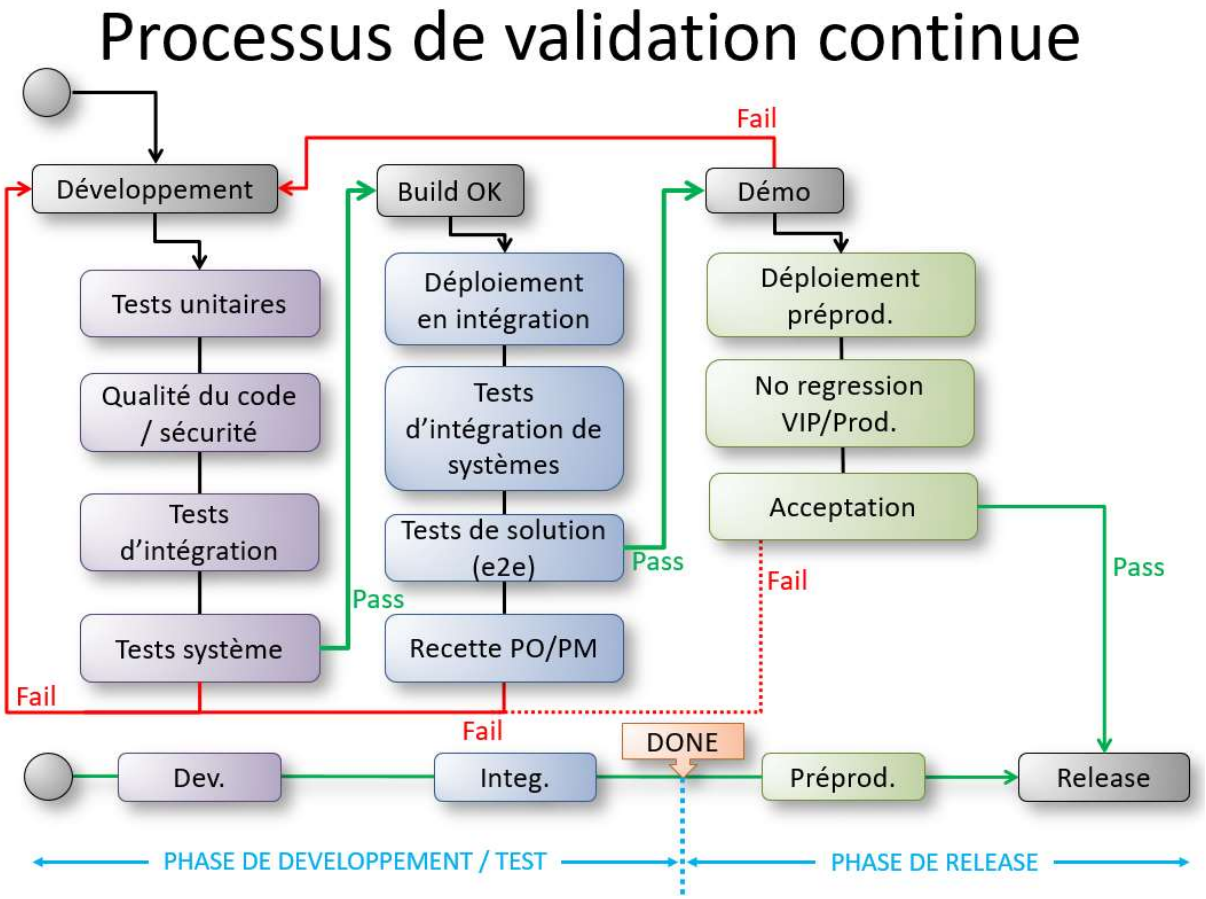
Le contrôle des dépendances est déclenché et les résultats pris en compte par les équipes de développement elles-mêmes. Les équipes de développement doivent solliciter l'équipe Sécurité ou le Responsable de tests pour statuer sur l'acceptation d'un risque lié à une vulnérabilité détectée.

V. Validation/intégration continue

La validation continue est menée de façon à détecter au plus tôt des défauts introduits dans la solution.

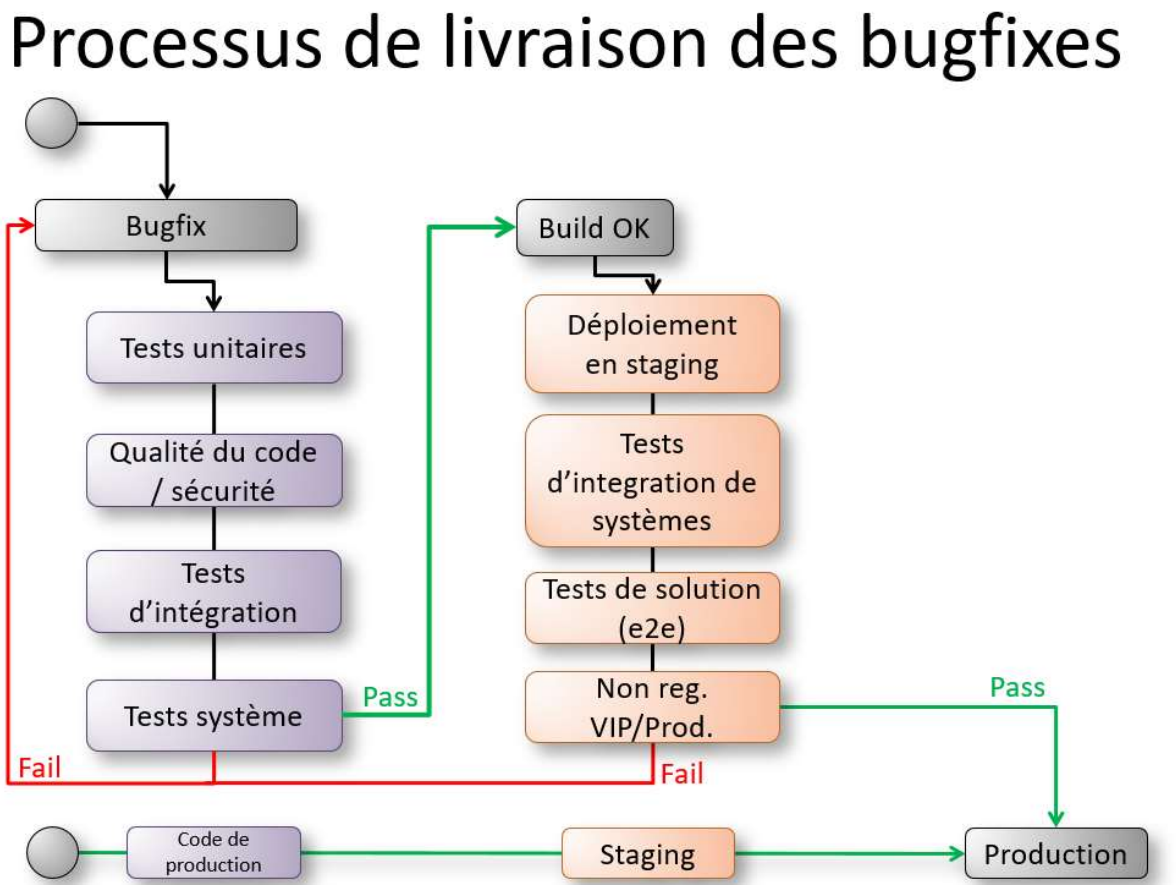
Les différentes étapes de cette validation sont orchestrées par Jenkins. Les environnements disponibles sont : développement, intégration et préproduction.

Le processus de test est le suivant :



VI. Flux des bugfixes

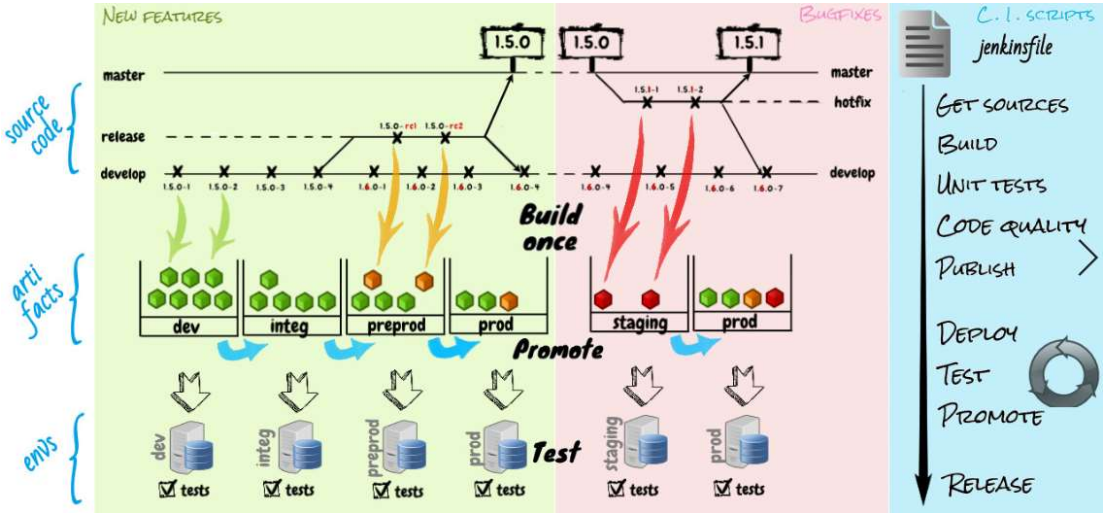
Les bugfixes sont appliqués sur le code de production et transitent par l'environnement de staging pour validation, avant de passer effectivement en production :



VII. Flux de développement

Le flux de développement AT INTERNET, visant à sécuriser les livraisons et à optimiser la détection de bugs au plus tôt se décline suivant différents axes :

- Les versions de livrables
- La gestion du code source (branches, tags, builds)
- L'utilisation des environnements internes
- La promotion des livrables



A. Les versions de livrables

Les versions de livrables sont définies conformément au SEMVER : MAJOR.minor.patch_build

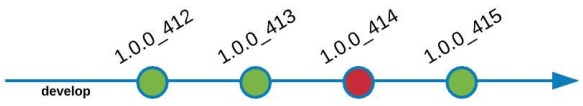
- MAJOR : incompatible changes
- minor : new backwards-compatible functionalities
- patch : backwards-compatible bug fixes
- build : unique build identifier

B. La gestion du code source

Le flux de code source AT INTERNET est basé sur le GitFlow, voici les différentes bonnes pratiques à respecter :

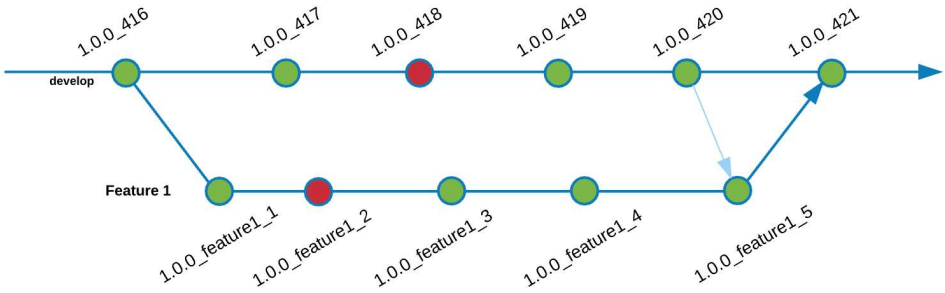
- **Chaque livrable doit être identifié de manière unique**, dans un souci de traçabilité. Une information unique de build est ajoutée par Jenkins, lors de la création de tout livrable (zip, dll, exe, binaire…)

Par ex :



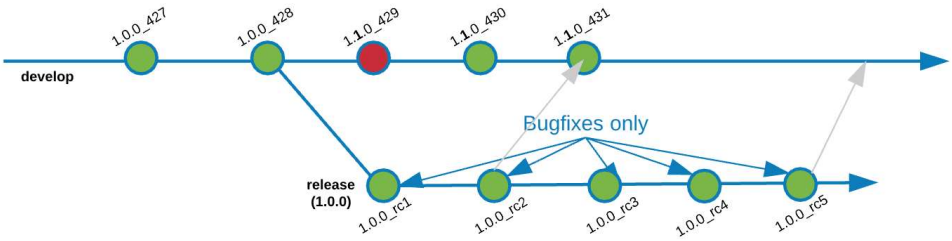
- Des branches sont créées pour chaque fonctionnalité importante et buildées séparément avant d'être réintégrées à la branche de développement. Les branches de fonctionnalités sont bien mises à jour (merge de develop dans la branche) et **validées avant réintégration**.

Par ex :



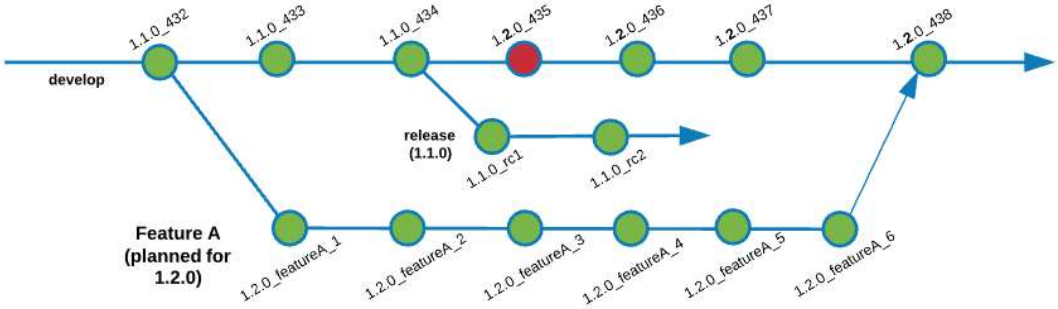
- Une branche de release est créée à chaque préparation de mise en production. **Seules des corrections de bugs y sont appliquées**, en vue de la livraison en production. Ces corrections de bugs sont reportées sur la branche de développement.

Par ex :



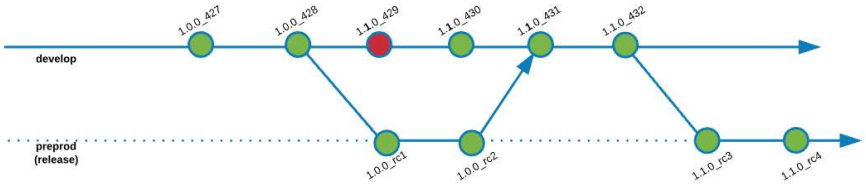
- Les branches de fonctionnalités sont réintégrées à la branche de développement si leur livraison en production est prévue **à la prochaine release**

Par ex :



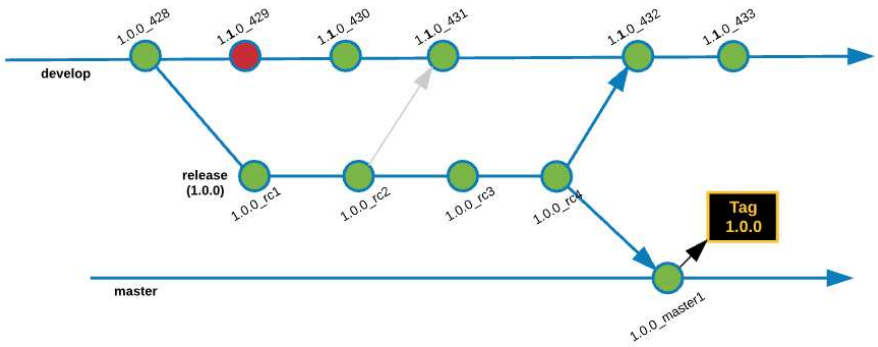
- Une branche ‘préprod’ pourra être utilisée comme branche de release pour les releases successives, afin de conserver l’historique sur une même branche.

Par ex :



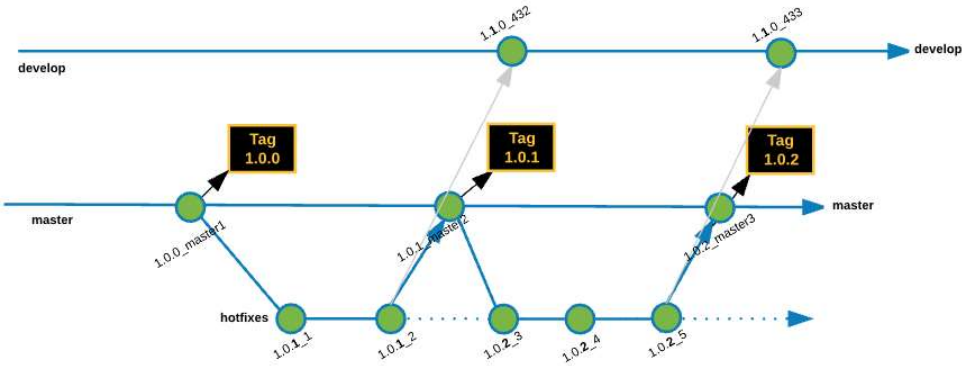
- Lors de la livraison en production, le code est mergé de la branche ‘release’ vers ‘master’ et **un tag de la version livrée en production est effectué**.

Par ex :



- Pour les hotfixes de production, **une branche 'hotfixes' est tirée depuis le tag concerné** (version en production). Les hotfixes sont appliqués sur cette branche qui est ensuite réintégrée au master et donne lieu à un nouveau tag (version patch). Les hotfixes sont également reportées dans la branche de développement.

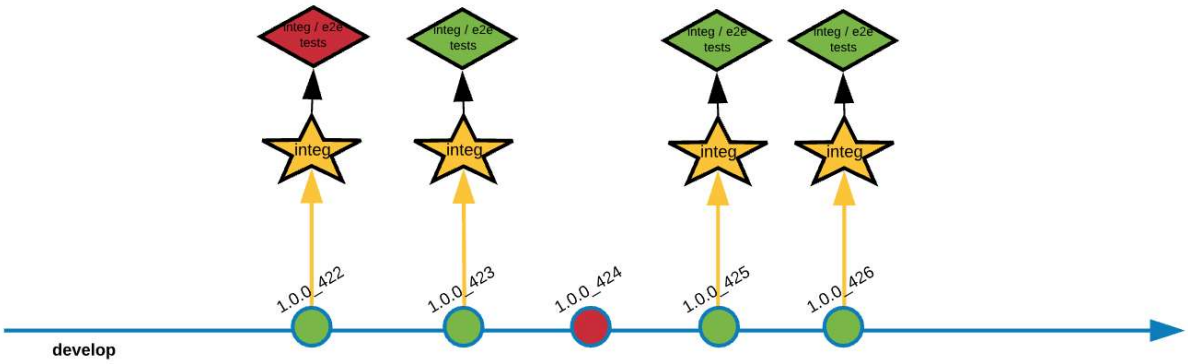
Par ex :



C. L'utilisation des environnements internes

- **L'environnement d'intégration** sert à valider la bonne intégration du produit dans son environnement logiciel. Chaque build valide en développement est déployé sur l'environnement d'intégration, **en vue d'y jouer les tests appropriés** : intégration de systèmes/solution (end2end), voir section III. Les niveaux de test)

Par ex :

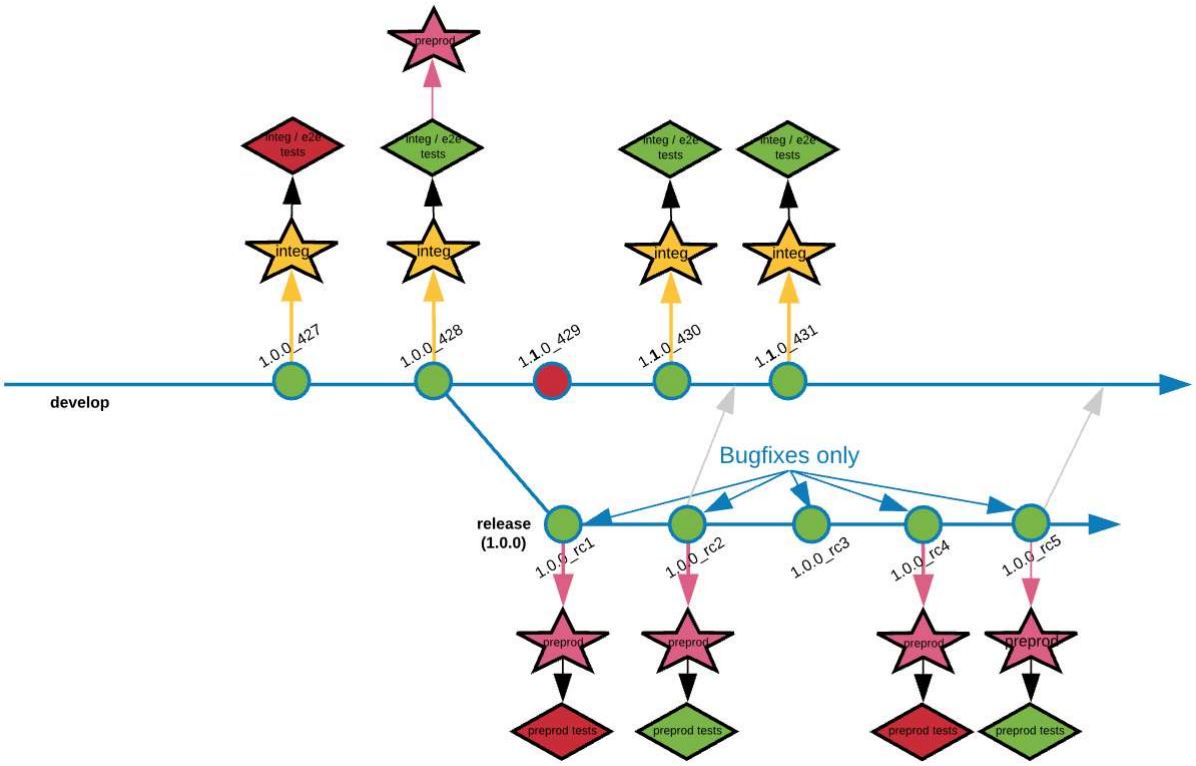


- **L'environnement de préproduction** accueille les fonctionnalités prévues pour la prochaine release, une fois validées avec succès en dev et en intégration. Lors de la création de la branche release en vue de la prochaine livraison, le fonctionnel est déployé sur

l'environnement de préproduction. Tous les fixes appliqués sur la branche release donnent également lieu à un redéploiement sur l'environnement de préproduction.

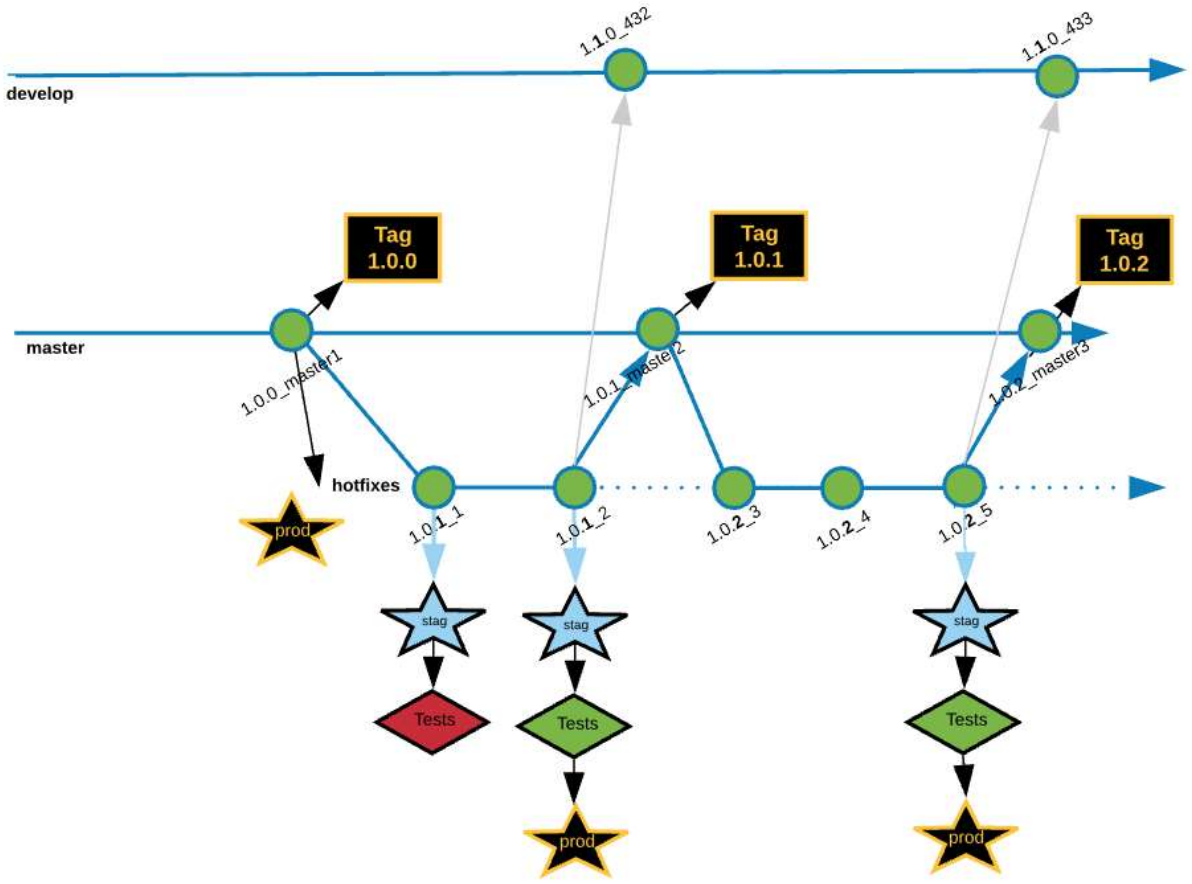
L'environnement de préproduction doit être assimilé à un environnement de production interne. Il peut ainsi accueillir des données client. Les exigences de sécurité en matière d'intégrité, confidentialité et traçabilité doivent être identiques à celles de l'environnement de production.

Par ex :



- **L'environnement de staging** est utilisé pour **valider les hotfixes de production** avant leur mise en production afin d'éviter toute régression en production. (voir section V. Flux des bugfixes)

Par ex :



Automatismes d'incrément des versions :

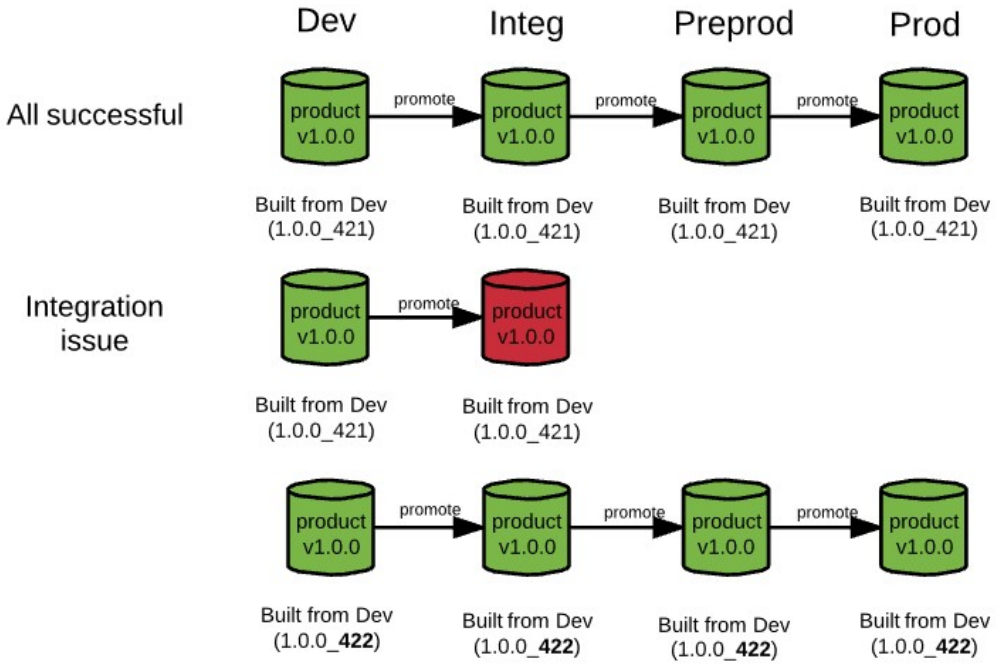
- Nouveau build de dev
  - Increment du build number sur develop (passage à v M.m.0_(build+1)
- Déploiement en integration
  - Pas de changement
- Création de la branche release/preprod
  - Increment de la mineure sur develop (passage à v M.m+1.0)
- Déploiement en préproduction
  - Pas de changement
- Mise en production
  - Tag de la version M.m.p sur master
- Préparation d'un hotfix prod
  - Increment de la version patch sur hotfix (passage à v M.m.p+1)
- Nouveau build sur hotfix
  - Increment du build number sur hotfix (passage à v M.m.p_(build+1)

D. La promotion des livrables

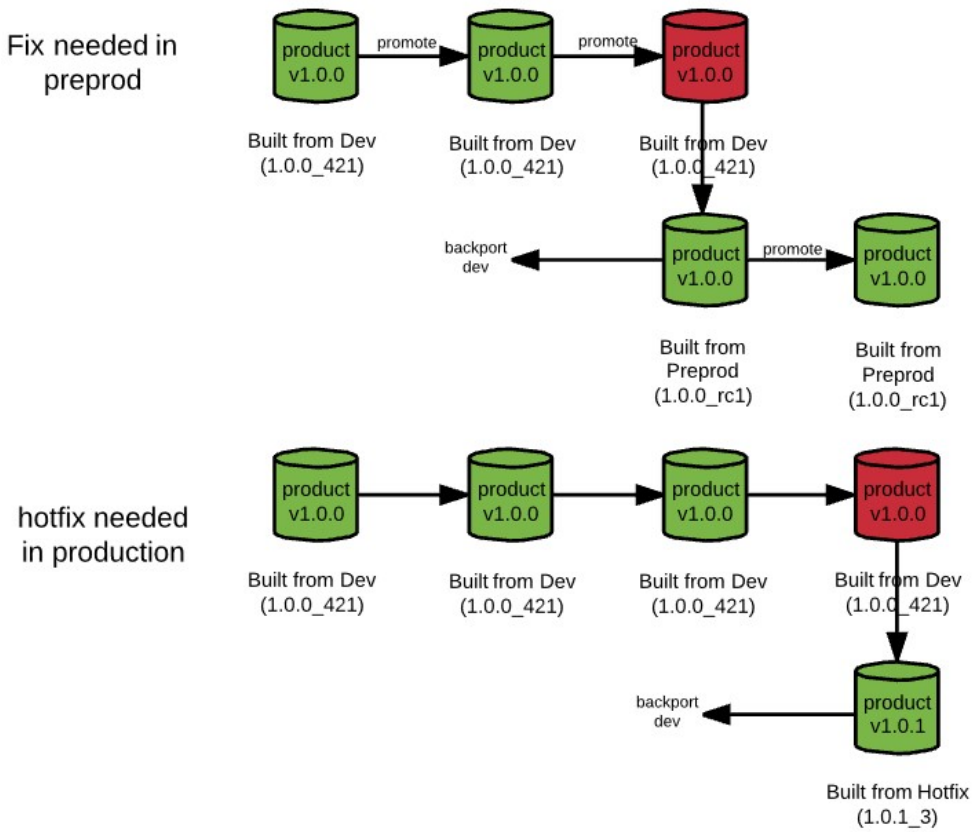
Afin d'éviter les soucis potentiels liés à la compilation, **on ne builde qu'une seule fois un livrable**, pour un même code. **Le livrable est promu d'environnement en environnement** jusqu'à la production tant que les tests sont passés avec succès.

Voici les différents cas que l'on peut rencontrer en effectuant des promotions de livrables, en fonction de l'environnement sur lequel on découvre un bug :

- ✓ Cas de découverte d'un bug pendant la phase de développement (en dev ou en intégration) : on rebuilde un livrable depuis la dev



- ✓ Cas de découverte d'un bug en préproduction ou en production : on rebuilde un livrable depuis la branche release ou la branche hotfix :



E. La gestion des changelogs

Nous ne disposons pas aujourd'hui de changelogs qui nous permettent de suivre les modifications apportées à nos produits sur les différents environnements.

VIII. Visibilité sur les résultats des tests

Le statut des tests implémentés doit rester disponible, pour permettre à chacun d'en prendre connaissance à tout moment. La mise en visibilité dépend du niveau de test concerné, et donc du public visé.

A. Tests unitaires

Destinataires

- ✓ Les développeurs

Mise en visibilité du scope des tests

- ✓ Les tests eux-mêmes (par de bonnes pratiques de nommage)
- ✓ Rapports de tests (JUnit par ex.) publiés dans Jenkins

Mise en visibilité du statut

- ✓ Statut du build dans Jenkins
- ✓ Statut des tests unitaires directement dans l'IDE

B. Analyse statique de code

Destinataires

- ✓ Les développeurs

Mise en visibilité du scope des tests

- ✓ Profil qualité disponible dans Sonarqube (règles d'analyse)
- ✓ Barrière qualité disponible dans Sonarqube (critères de succès)

Mise en visibilité du statut

- ✓ Statut du build dans Jenkins (Quality gate)
- ✓ Analyse statique directement dans l'IDE
- ✓ Analyses Sonarqube (sur le serveur Sonar AT Internet)

C. Tests système

Destinataires

- ✓ L'équipe produisant le système (devs, tests, PO, PM)

Mise en visibilité du scope des tests

- ✓ Les fonctionnalités couvertes par les tests au niveau système sont listées dans des fichiers Excel, par projet. Les fichiers sont disponibles dans ce dossier partagé.
- ✓ Une vue globale des usecases est également disponible dans ce fichier, qui regroupe toutes les données, par projet client, composant technique et équipe.
- ✓ Des indicateurs issus de ces données sont également exposés à deux endroits :
  - TechCenter : https://techcenter.atinternet.tools/couvertures-fonctionnelles/
  - Project sheets app : http://quality.dev.aws.intraxiti.com/

Mise en visibilité du statut

- ✓ Statut du job de test dans Jenkins
- ✓ Les statuts de certains jobs jenkins exécutant les tests au niveau système sont disponibles sur cette page : http://quality.dev.aws.intraxiti.com/ (**A FAIRE EVOLUER**)

D. Tests d'intégration de systèmes (inter-équipes)

Destinataires

- ✓ Les équipes impliquées dans la réalisation du projet client concerné (dev, test, PO, PM)

Mise en visibilité du scope des tests

- ✓ **TO BE DEFINED**

Mise en visibilité du statut

- ✓ Statut du job de test dans Jenkins
- ✓ Les statuts de certains jobs jenkins exécutant les tests au niveau système sont disponibles sur cette page : http://quality.dev.aws.intraxiti.com/ (**A FAIRE EVOLUER**)

E. Tests de solution (end 2 end)

Destinataires

- ✓ Les équipes impliquées dans la réalisation du projet client concerné (dev, test, PO, PM)

Mise en visibilité du scope des tests

- ✓ **TO BE DEFINED**

Mise en visibilité du statut

- ✓ Statut du job de test dans Jenkins
- ✓ Les statuts de certains jobs jenkins exécutant les tests au niveau système sont disponibles sur cette page : http://quality.dev.aws.intraxiti.com/ (**A FAIRE EVOLUER**)

IX. Communications

Différents niveaux de communications sont nécessaires pour offrir une bonne visibilité sur le test dans l'entreprise. Le contenu et le format des communications varient suivant les contextes et les interlocuteurs.

A. Stratégie de test par projet

Contenu

- ✓ Couverture fonctionnelle des tests
- ✓ Nombre de cas de tests identifiés/automatisés (par projet/équipe/criticité)
- ✓ Décisions de ne pas couvrir certaines fonctionnalités par des tests automatisés
- ✓ Etat d'avancement, risques identifiés
- ✓ Indicateurs liés au code : note de sécurité, nb lines of code

Format

- ✓ Indicateurs globaux et par projet : https://techcenter.atinternet.tools/couvertures-fonctionnelles/
- ✓ Une synthèse de tous les projets est disponible : http://quality.dev.aws.intraxiti.com/
- ✓ Des rapports de situation, par projet, sont également envoyés par mail

Fréquence

- ✓ Les indicateurs liés aux tests sont mis à jour mensuellement pour chaque projet
- ✓ Un rapport « indicateurs qualité » du projet est alors envoyé par mail

B. L'effort de test dans l'entreprise

Certains indicateurs sont suivis afin de s'assurer du bon investissement en test dans l'entreprise. Certains de ces indicateurs nous permettent aussi de réorienter l'effort de test ou de prioriser certaines actions.

Contenu

- ✓ L'ensemble des indicateurs définis dans notre politique de test, permettant de mettre en relation :
  - L'effort de détection de défauts (interne/prod)
  - Le coût de correction des défauts (interne/prod)
  - La répartition des défauts par composant / équipe
- ✓ Cette évaluation nous permet de nous assurer de la cohérence de notre stratégie d'investissement en test et de réagir en prenant des décisions avisées.

Format

- ✓ Synthèse des différents indicateurs, sous forme d'une présentation (ppt)

Fréquence

- ✓ Publication trimestrielle / bilan annuel

X. Annexes

A. Les caractéristiques Qualité selon ISO 25010 (SQuaRE)

La qualité d'un logiciel peut être exprimée suivant 8 caractéristiques, elles-mêmes décomposées en sous caractéristiques :

- Fonctionnalité
  Aptitude, exactitude, complétude
- Fiabilité
  Maturité, disponibilité, tolérance aux fautes, capacité de récupération
- Rendement
  Efficacité des ressources employées, efficacité des temps de réalisation, capacité
- Compatibilité
  Interopérabilité, conformité
- Utilisabilité
  Facilité de reconnaissance de la pertinence, Facilité d'apprentissage, Opérabilité, Résistance aux erreurs des utilisateurs, Esthétique, Accessibilité
- Sécurité
  Confidentialité, Intégrité, Non répudiation, Authenticité, Responsabilité
- Maintenabilité
  Modularité, Réutilisabilité, Facilité de modification, Facilité d'analyse, Testabilité
- Portabilité
  Facilité d'installation, facilité de migration, adaptabilité

B. Les 7 principes du test (ISTQB)

- Le test montre la présence de défauts
  Il ne peut garantir leur absence
- Découvrir les défauts le plus tôt possible
  Importance du test dès les premières phases
- Le test exhaustif est impossible
  Nécessité de prioriser / adapter l'effort de test

- Agrégation de défauts
  Nécessité de considérer la répartition des défauts constatés pour cibler l'effort de test
- Paradoxe du pesticide
  Nécessité de mettre à jour / faire vivre les jeux de tests
- Le test dépend du contexte
  Nécessité d'adapter les pratiques et objectifs suivant les contextes
- Illusion de l'absence de défauts
  Trouver/corriger des bugs n'est pas gage de satisfaction du client, le produit doit aussi répondre à son besoin

C. Glossaire

Afin d'assurer une compréhension commune des termes utilisés dans ce document et de faciliter les échanges dans l'entreprise, voici certaines définitions adoptées par l'ISTQB :

Coût de la qualité : Le coût total imputé aux activités et problèmes liés à la qualité, souvent divisé en coûts de prévention, coûts d'estimation, coûts des défaillances internes et coûts des défaillances externes.

Défaillance : Déviation constatée du composant ou système de la fourniture, du service ou du résultat attendu [d'après Fenton]; Fin de la capacité du système ou du composant à effectuer la fonction requise, ou à l'effectuer dans les limites spécifiées. [D'après IEEE 729] Incapacité d'un système ou d'un composant d'exécuter une fonction requise dans les limites spécifiées. Une défaillance peut être produite quand un défaut est rencontré [DO-178B].

Défaut : imperfection dans un composant ou un système qui peut conduire à ce qu'un composant ou un système n'exécute pas les fonctions requises, par exemple une instruction ou une définition de données incorrecte. Un défaut, si rencontré lors de l'exécution, peut causer la défaillance d'un composant ou d'un système.

Politique de tests : un document de haut niveau décrivant les principes, approches et objectifs majeurs de l'organisation ayant trait aux tests.

Qualité : degré par lequel un composant, système ou processus atteint des exigences spécifiées et/ou des besoins ou attentes des clients ou utilisateurs [d'après IEEE 610].

Stratégie de test : un document de haut niveau définissant, pour un programme, les niveaux de tests à exécuter et les tests dans chacun de ces niveaux (pour un ou plusieurs projets).

Test : processus consistant en toutes les activités du cycle de vie, statiques et dynamiques, concernant la planification et l'évaluation de produits logiciels et produits liés pour déterminer s'ils satisfont aux exigences, pour démontrer qu'ils sont aptes aux objectifs et détecter des anomalies.

Test de confirmation : type de test qui exécute des cas de test qui ont été en échec la dernière fois qu'ils furent exécutés, de façon à vérifier le succès des actions de correction.

Test de non régression : type de test d'un programme préalablement testé, après une modification, pour s'assurer que des défauts n'ont pas été introduits ou découverts dans des parties non modifiées du logiciel, comme suites des modifications effectuées. Ces tests sont effectués quand le logiciel ou son environnement est modifié. Ils peuvent être pratiqués à tous les niveaux de test.

Test fonctionnel : type de test basé sur une analyse des spécifications d'une fonctionnalité d'un composant ou d'un système. Des tests fonctionnels peuvent être implémentés à tous les niveaux de test.

Test non-fonctionnel : type de test des attributs d'un composant ou d'un système qui ne sont pas liés aux fonctionnalités (p.ex. fiabilité, rendement, utilisabilité, maintenabilité et portabilité). Des tests non-fonctionnels peuvent être implémentés à tous les niveaux de test.

Test structurel : type de test basé sur une analyse de la structure interne du composant ou système. (également appelé 'test boîte blanche').