PROFESSIONNELS DE SANTÉ

OUTILS POUR LA PRATIQUE

CONCEPTION DE LOGICIELS POUR LE DIAGNOSTIC CLINIQUE PAR SÉQUENÇAGE HAUT-DÉBIT

e-cancer.fr

INSTITUT NATIONAL DU CANCER

L'Institut national du cancer (INCa) est l'agence d'expertise sanitaire et scientifique en cancérologie chargée de coordonner la lutte contre les cancers en France.

Ce document doit être cité comme suit : © *Conception de logiciels pour le diagnostic clinique par séquençage haut-débit,* collection Outils pour la pratique, INCa, février 2018.

Ce document est téléchargeable sur **e-cancer.fr**

TABLE DES MATIÈRES

1. CONTEXTE

L'Institut national du cancer soutient depuis 2013 sept équipes dites « référentes en bioinformatique » dans le cadre de l'appel à projet « Structuration du séquençage de nouvelle génération à visée de diagnostic des cancers ». Un des objectifs de ces équipes est de mettre à disposition des laboratoires du réseau d'oncogénétique et des plateformes de génétique moléculaires des cancers des chaînes de traitement d'analyses (appelées « pipeline ») validées en routine clinique. La mise à disposition d'un pipeline d'analyse repose sur le développement d'un logiciel qui a pour tâche d'automatiser la chaîne de traitement, de mettre à disposition les résultats, de détecter les erreurs, d'assurer une traçabilité des analyses conduites et de garantir leur reproductibilité. Ce document propose un ensemble de recommandations jugées importantes, à prendre en compte lors du développement d'un outil utilisé pour le diagnostic. Sans aller vers une certification de logiciel selon une norme (ISO 25 000, SQuaRE par exemple) ou de parler de démarche relative à la norme CE-IVD (ISO 13 485, NF EN 6 236), le fait de suivre ces recommandations permettra de fournir aux laboratoires utilisateurs des éléments justifiant de la maîtrise du logiciel mis à disposition par ses développeurs. L'adoption de ces recommandations durant les développements devrait permettre de faciliter la qualification du logiciel au sein d'un laboratoire accrédité ou en voie d'accréditation selon la norme ISO 15 189. En effet, l'adoption d'un logiciel d'analyse par un laboratoire de biologie moléculaire doit passer par une étape de qualification de ce logiciel par le laboratoire. Cette étape de qualification peut s'appuyer sur des recommandations de société savantes, comme par exemple le guide de qualification des pipelines rédigé et diffusé par l'Association nationale de génétique moléculaire (ANPGM).

De manière générale, lors d'un développement informatique, l'évaluation de la performance d'un logiciel est une mesure difficile. Néanmoins, il est généralement rapporté que le développement d'un logiciel dans un environnement bénéficiant d'un management de la qualité défini selon la norme ISO 9 001 favorise la maîtrise des procédures de développement. Les laboratoires de biologie médicale français ont une obligation légale d'entrée dans une démarche qualité selon la norme ISO 15 189 pour une accréditation totale de leur activité en octobre 2020. Le management de la qualité de cette norme étant hérité de la norme ISO 9 001, nous pouvons admettre que des logiciels développés dans un laboratoire accrédité selon l'ISO 15 189 répondraient aux exigences de la norme ISO 9 001 dans la mesure où les développeurs seraient en capacité de montrer l'intégration de leur travail dans le système de management de la qualité du laboratoire auquel ils sont affiliés. Nous pouvons considérer ici qu'il s'agit alors d'un premier élément de maîtrise des développements informatiques.

En 2017, les analyses dérivées du séquençage à haut-débit sont des actes définis comme innovants par la Direction générale de l'offre de soins. Ils ne sont donc pas soumis à une obligation d'accréditation selon la norme ISO 15 189. Néanmoins, au terme de l'évaluation médico-économique de ces actes prévue dans ce cadre légal, la décision sera prise ou non de les intégrer à la nomenclature des actes de biologie médicale. En cas d'intégration, l'obligation d'accréditation sera effective. Dans ce contexte, ces recommandations relatives au développement logiciel visent à simplifier la qualification des outils développés pour les applications liés au séquençage à haut-débit, par des laboratoires accrédités selon la norme ISO 15 189.

L'objet de ce guide de bonnes pratiques est donc :

- d'améliorer la robustesse, d'un point de vue informatique et analytique, des pipelines bioinformatiques développés dans une structure académique (laboratoires de biologie moléculaires eux-mêmes ou structures partenaires) ;
- de proposer des indicateurs qui assurent que les développements suivent de bonnes pratiques adaptées à l'environnement de développement (recueil des éléments de preuve).

REMARQUES

Les tests de biologie moléculaire sont toujours réalisés au sein d'un laboratoire de biologie moléculaire. En fonction du mode d'organisation et des choix réalisés par les établissements, le traitement des données peut être réalisé :

- en interne, au sein du laboratoire de biologie moléculaire ;
- en partenariat avec un laboratoire de bioinformatique local ;
- par un prestataire extérieur à l'établissement.

Lorsque le traitement bioinformatique d'un examen est réalisé par une structure extérieure au laboratoire de biologie moléculaire, le partenariat doit être formalisé par un accord tripartite entre le laboratoire de biologie moléculaire, la structure assurant les traitements bioinformatiques et la direction des systèmes d'information de l'établissement. Cet accord doit notamment rappeler les exigences en matière de sécurité, de performance des solutions d'analyse, et doit rappeler les prérequis nécessaires à toute analyse. De même, des tests de non régression validés par le laboratoire doivent être mis en place avec le développeur. Le développement et l'utilisation des solutions proposées par un laboratoire extérieur doivent répondre à des exigences de qualité. En cela, l'adoption de la norme ISO 9 001 par ces structures prestataires est souhaitable, notamment pour assurer la mise à disposition des processus qualité.

1.1. Méthodologie

Un groupe de travail a été constitué en 2017 pour l'élaboration de ce document. Ce groupe est composé de bioinformaticiens et de biologistes représentant plusieurs laboratoires de bioinformatique et de biologie moléculaire qui ont développé des pipelines d'analyse à des fins de diagnostic en génétique constitutionnelle et somatique. Le document a été soumis à une relecture externe.

GROUPE DE TRAVAIL

Nom	Établissement	Fonction
Baudet Christian	Centre Léon Bérard, Lyon	Bioinformaticien
Brault Baptiste	Centre François Baclesse, Caen	Bioinformaticien
Castera Laurent	Centre François Baclesse, Caen	Biologiste
Choumouss Kamoun	Institut Curie, Paris	Bioinformaticienne
Coutant Sophie	CHU de Rouen	Bioinformatienne
De Tayrac Marie	CHU de Rennes	Biologiste
Denoual Florent	CHU de Rennes	Bioinformaticien
Gautheret Daniel	Gustave Roussy, Villejuif	Bioinformaticien
Hupé Philippe	Institut Curie, Paris	Bioinformaticien
Lanos Raphaël	CHU de Rouen	Bioinformaticien
Le Béchec Antony	IRC de Strasbourg	Bioinformaticien
Lermine Alban	AP-HP, Paris	Bioinformaticien
Paimparay Germain	Centre François Baclesse, Caen	Bioinformaticien
Rousselin Antoine	Centre François Baclesse, Caen	Bioinformaticien
Sohier Émilie	Centre Léon Bérard, Lyon	Bioinformaticienne
Truntzer Caroline	Centre Georges François Leclerc, Dijon	Bioinformaticienne

RELECTEURS

Nom	Établissement	Fonction
Gérard Bénédicte	CHRU de Strasbourg	Biologiste
Muller Jean	CHRU de Strasbourg	Biologiste / bioinformaticien
Rouleau Etienne	Gustave Roussy, Villejuif	Biologiste
Stalter Fabrice	CHRU de Strasbourg	RSSI/Qualiticien

1.2. Cibles

Ce document s'adresse aux équipes bioinformatiques développant des pipelines à visée diagnostique ainsi qu'aux laboratoires de biologie moléculaire recherchant des solutions pour l'analyse des données de séquençage à haut-débit.

2. MAÎTRISE DU DÉVELOPPEMENT INFORMATIQUE

De manière générale, chaque développement informatique d'un logiciel ayant pour objectif l'analyse de données à visée diagnostique doit savoir répondre à cinq questions :

- le logiciel répond-il aux exigences des analyses auxquelles il est dédié ?
- le logiciel est-il suffisamment optimisé pour minimiser le délai de rendu du diagnostic ?
- le logiciel se comporte-t-il comme attendu et donne-t-il des résultats de sortie corrects pour chaque entrée possible ?
- à quel point le logiciel pourrait-il être maintenu et s'adapter à l'expression de nouveaux besoins ?
- le logiciel renvoie-t-il un résultat identique pour une même analyse ?

Le développement informatique devra donc avoir pour objectif de répondre à des exigences compatibles avec une utilisation dans le cadre d'une analyse dédiée à un diagnostic moléculaire :

- maintien d'une sensibilité et d'une spécificité compatible avec l'application ;
- robustesse ;
- efficience ;
- maintenabilité ;
- ergonomie.

Pour répondre à ces exigences de qualité, il est important de définir des règles dans la gestion du développement, l'écriture du code, la validation et la diffusion du logiciel. Ces règles, héritées des bonnes pratiques de développement en informatique, sont contextualisées ici à l'aune des logiciels d'analyse bioinformatique dédiés à l'analyse des données de génomique dans un contexte de diagnostic.

REMARQUES

Les notions de sensibilité et de spécificité sont liées aux particularités du logiciel exécuté (choix de l'algorithmique et du paramétrage des logiciels associés) et des données qui sont manipulées. Une évaluation de ces paramètres doit être réalisée avant la diffusion du logiciel développé. Notons que le logiciel peut être entraîné et testé par les développeurs sur des jeux de données si possible réels et mis à disposition pour cette qualification par le laboratoire, mais également sur un jeu de données simulées. Cette qualification par les développeurs ne se substitue ni à la qualification qui doit être réalisée par le laboratoire utilisateur, ni à la détermination de la sensibilité et de la spécificité de l'analyse biomédicale dans laquelle le logiciel est intégré (i.e. validation de méthode propre à chaque laboratoire utilisateur).

2.1. Gestion du développement informatique

2.1.1. Cycle et plan de développement

L'utilisation d'un outil de gestion de projet est recommandée lors du développement. Cela est d'autant plus nécessaire lorsqu'il s'agit de développements collaboratifs. Il permet de :

- tracer les actions ;
- définir, prioriser et attribuer les tâches ;
- tracer le résultat, les anomalies et les actions correctives ;
- consolider l'historique des développements ;
- préfigurer la documentation technique.

Les demandes de nouvelles fonctionnalités et d'améliorations sont enregistrées et intégrées à la liste des tâches à faire lors du prochain cycle de développement. Il est possible de ventiler ces modifications selon leur caractère d'urgence sur plusieurs versions du logiciel, afin de ne pas bloquer la livraison des futures versions stables. Il est préférable que l'outil de gestion de projet soit couplé à un outil de gestion des versions du code source qui facilite le développement collaboratif du logiciel. Il est recommandé de travailler sous forme de cycle de développement.

EXEMPLES

Outils pouvant être utilisés pour la gestion de projet :
Jira (https://fr.atlassian.com/software/jira)
Redmine (http://www.redmine.org/)

Outils pouvant être utilisés pour la gestion des versions du code :
git (https://git-scm.com/)
subversion (https://subversion.apache.org/)

Outils couvrant des fonctionnalités de gestion de projet et de gestion de versions :
Gitlab (https://about.gitlab.com)
Gogs (https://gogs.io/)

Il est conseillé d'associer aux cycles de développement un membre « futur utilisateur » (généticien moléculaire) qui réalise une veille de l'évolution des pratiques afin d'assurer en continu la bonne adéquation des développements avec l'expression des besoins, et d'alerter rapidement en cas de modification des besoins. En effet, les évolutions techniques en génétique moléculaire sont aujourd'hui très rapides, et le risque d'obsolescence du logiciel existe entre le moment où le besoin est formulé et la solution délivrée.

Avant d'entamer un cycle de développement, il convient de :

1. **Recueillir les besoins**, de préférence sous forme écrite (cahier des charges), afin de définir clairement les objectifs et les différents jalons du projet. Ce recueil doit être défini par les utilisateurs (ou représentants) en collaboration avec les informaticiens et bio-informaticiens (ou représentants)[1].
2. **Formaliser les besoins,** sous forme de diagrammes ou d'algorithmes décrivant les étapes clés du processus d'analyse.
3. **Réaliser un prototype**, afin d'estimer le coût de développement et définir le choix de la solution à adopter pour atteindre les objectifs. Ce prototype doit rester un outil pour les bio-informaticiens. Il doit être minimaliste en termes de fonctionnalités et dans son interface. Sa diffusion auprès des utilisateurs doit être limitée et accompagnée d'une communication claire sur ses objectifs.
4. **Définir l'architecture logicielle et la sélection des solutions technologiques** permettant le passage du prototype à la version industrielle du pipeline. Les choix doivent être guidés par la robustesse et la pérennité des solutions, leur portabilité et leur facilité de mise en œuvre.

Le cycle de développement comporte différentes étapes:

1. **Étape 1 - le développement informatique :** cette étape va en réalité comporter différentes phases :
    a. **L'écriture du code** à proprement parler, sur la base du cahier des charges préalablement formalisé. L'informaticien implémente les fonctionnalités attendues.
    b. **La réalisation d'un ensemble de tests** dont on distinguera plusieurs niveaux :
        i. **les tests unitaires** confirment qu'une portion de code fournit bien une sortie correcte en fonction de paramètres d'entrées donnés. C'est le développeur qui se charge de la réalisation de ces tests.
        ii. **les tests d'intégration** vérifient que les interfaces des composants sont cohérentes entre elles et que le résultat de leur intégration permet de réaliser les fonctionnalités prévues.
        iii. **les tests système (ou tests fonctionnels)** valident que le logiciel complet fonctionne et correspond bien à la définition des besoins tels qu'ils avaient été exprimés. Idéalement, si l'effectif de l'équipe le permet, c'est une personne autre que le développeur qui se charge de la réalisation de ces tests.

[1] Il faut noter que la gamme des profils de compétences des bioinformaticiens est large. Certains montrent des compétences poussées en développement informatique alors que d'autres se spécialisent dans la bio-analyse. En conséquence, certains bio-informaticiens seront « consommateurs » des développements alors que d'autres les auront conçus (éventuellement en collaboration avec des informaticiens). Il s'agit donc dans le cadre d'un projet de développement de ce type d'outils de définir les rôles précis de chacun des intervenants.

iv. **les tests de non-régression** vérifient que la correction des erreurs ou l'ajout de nouvelles fonctionnalités n'ont pas affecté les parties déjà testées (repasser en systématique tous les tests déjà exécutés). Ces tests sont indispensables aux changements de versions et le résultat de ceux-ci devraient être rendus disponible au laboratoire utilisateur afin de faciliter la qualification sur site de la nouvelle version (exigences ISO 15189 sur la maitrise des procédures informatiques).

2. **Étape 2 - les tests d'acceptation (validation ou recette) :** une fois les tests précédents concluants, l'utilisateur doit pouvoir valider l'ensemble des fonctionnalités développées dans un environnement similaire à la production en terme de paramétrage et d'utilisation. Il est conseillé que l'utilisateur puisse tester des scénarii réalistes avec des données réelles. Dans le cas d'un logiciel à plusieurs composantes, il s'agit de vérifier qu'il n'y a pas de perte de données entre les différentes étapes. Par exemple, si le logiciel annote une liste de variants, le nombre de variants en entrée et en sortie doit être le même sans transformation de leur position ou de leur nature. De même, si le logiciel est une interface de visualisation des résultats, l'intégrité des données mises à disposition doit être vérifiée. Des tests de résilience du logiciel peuvent également être mis en œuvre, par exemple pour vérifier si l'application se relance correctement d'elle-même après une interruption ou une panne. Cette phase permet également de relever d'éventuelles erreurs à corriger avant la mise en production. Cette phase doit rester limitée dans le temps et aboutir sur la phase de production ou un retour en développement.
3. **Étape 3 - le déploiement en production :** après validation, la solution logicielle avec ces nouvelles fonctionnalités est installée dans l'environnement de production. Cette étape est donc réalisée dans le laboratoire ou par les structures informatiques sous-traitantes du laboratoire (serveur de calcul hébergé au laboratoire ou serveur hébergé et administré par la DSI hospitalière locale).

CAS DE LA BIOINFORMATIQUE EXTERNALISÉE

Dans le cas particulier où les développements bioinformatiques sont effectués en dehors du laboratoire de biologie moléculaire, ces étapes doivent figurer clairement dans les procédures qualité du prestataire, en fournissant tout particulièrement les preuves des fonctionnalités et de non-régression (Étape 1.b.iii et 1.b.iv), ainsi que la liste des modifications (sous forme de Release Notes). Concernant l'étape de validation (Étape 2), le prestataire et le laboratoire doivent définir contractuellement la procédure.

2.1.2. Gestion des versions du code

À toutes les étapes de développement logiciel (développement, validation et production), l'utilisation d'un dépôt distant avec gestion des versions pour le code est indispensable car cela permet de :

- conserver le code source relatif aux différentes versions du logiciel. Les différentes phases de développement et versions associées sont clairement identifiées. Il est à noter que les différentes versions du code doivent être conservées aussi longtemps que les données à visée diagnostique qui ont été générées avec celui-ci ;
- protéger le projet contre la perte de données dans l'environnement de travail et les défaillances techniques ;
- faciliter le travail collaboratif en équipe et le partage du code ;
- pouvoir revenir à une version antérieure du logiciel.

Le dépôt distant peut se trouver sur un serveur dont la sauvegarde des données est assurée en cas de défaillance. Il est également possible de déposer le code sur les dépôts en ligne tel que Sourceforge ou Github, en étant pleinement conscient que toutes données déposées sur ces sites tombent sous la législation du pays auquel le site est rattaché. Un dépôt maîtrisé localement est donc souhaitable. Préalablement au dépôt du code sur des dépôts en ligne, le développeur devra en informer son service juridique et apposer une licence logicielle à son code qui définit les conditions d'utilisation, de modification et de diffusion de celui-ci.

LICENCES LOGICIELLES

Il existe différents types de licences mais nous pouvons par exemple mentionner les licences CeCILL pour leur compatibilité avec le droit français :

CeCILL-B, qui est totalement compatible avec les licences de type BSD (BSD, X11…) qui ont une forte obligation de citation (qui va plus loin qu'une simple note de copyright). La licence GNU GPL ne supporte pas cette exigence ; ce qui la rend incompatible avec CeCILL-B.

CeCILL-C, pour « composant », qui est totalement compatible avec la licence GNU LGPL de la FSF.

Il est conseillé de fixer les versions utilisées à l'aide d'un tag ou numéro de version (eg v1.2.3) et de les tracer pour chaque étape de développement, validation et production. Annexé au code, le fichier CHANGELOG devra lister les différents numéros de version par ordre décroissant et décrire les principales modifications, corrections, etc. ajoutées à chacune de ces versions. Quand bien même un seul informaticien serait impliqué dans le développement logiciel, il est recommandé d'utiliser systématiquement un outil de « versionning » pour tracer toutes les modifications du code.

REMARQUE

Pour la numérotation des versions, il est recommandé d'utiliser la gestion sémantique des versions (SemVer) décrite sur le site http://semver.org/lang/fr/.

2.1.3. Utilisation d'environnements de travail distincts

Il est indispensable que les différentes étapes décrites précédemment soient réalisées dans des environnements distincts pour :

- garantir que le logiciel utilisé en production soit toujours dans une version stable ;
- permettre aux utilisateurs de valider une nouvelle version sans impacter ni la version de production, ni la version en développement ;
- permettre aux développeurs de d'implémenter de nouvelles fonctionnalités sans porter préjudice aux utilisateurs qui valident une nouvelle version ou utilisent la version de production.

Ces différents environnements doivent être étanches afin de garantir une séparation complète des droits d'accès et des usages. Ces environnements doivent être maîtrisés et si possible identiques. S'ils diffèrent par leur infrastructure (ex: serveurs physiques différents, montages disques différents, dimensionnement des ressources), l'impact de ces différences doit être testé.

ORGANISATION POSSIBLE AVEC TROIS ENVIRONNEMENTS DISTINCTS

Environnement de développement

Environnement de validation

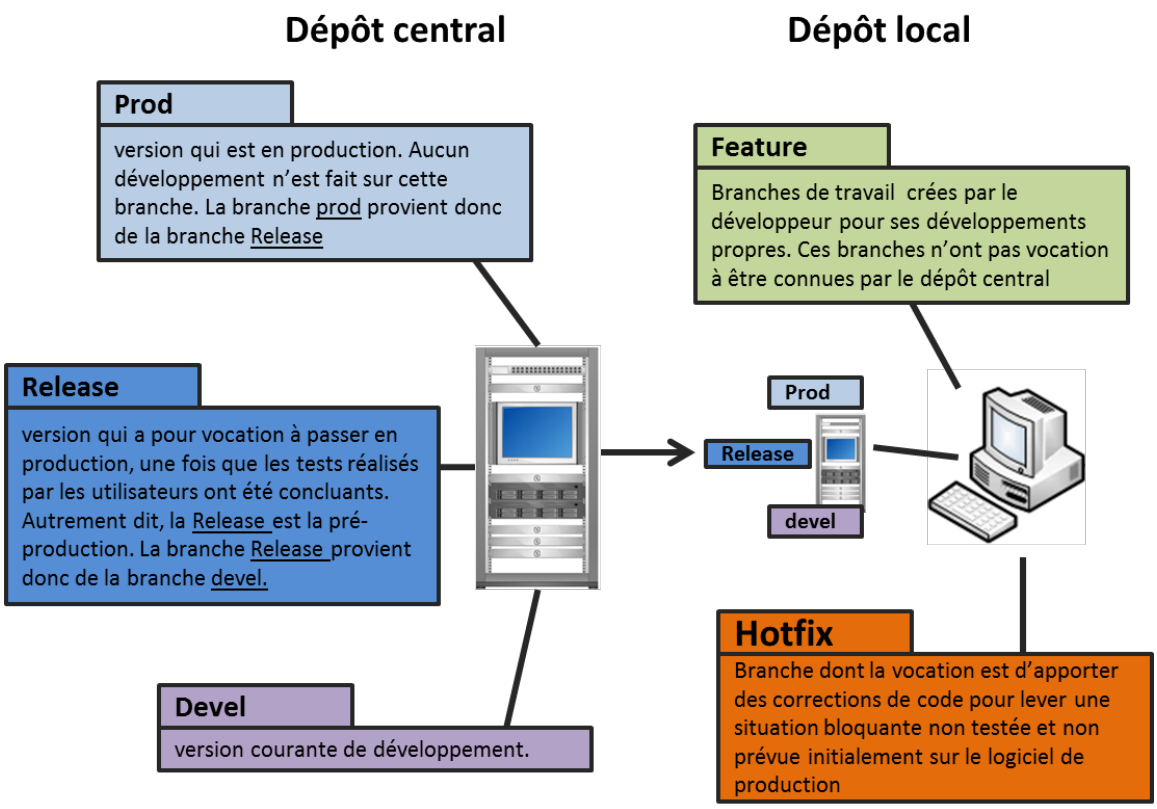
Environnement de production

2.1.4. Cycle de vie du logiciel

Le principe de séparation des environnements de travails évoqué ci-dessus doit idéalement s'appliquer aussi à la gestion des versions du code. En effet, lors de l'installation du logiciel en validation et en production, il est conseillé d'isoler les nouvelles versions sur de nouvelles branches de développement. L'objectif est de disposer d'une branche stable et indépendante où des corrections de bugs bloquants restent possibles.

Pour cela, différentes branches sont créées sur le dépôt central, chaque branche correspondant à une finalité bien précise. Le répertoire de travail local contient une copie des branches présentes sur le réseau central ainsi que les branches propres au développeur.

Figure 1 **Modèle d'organisation des branches de développement recommandé**



REMARQUE

Si les différentes branches concernées du projet ne sont pas maîtrisées et que celui-ci se situe sur un dépôt GitHub, GitLab ou consorts, il est possible de créer un « fork » du projet dans son espace personnel pour agir sur sa propre version de production tout en restant en lien avec le projet originel. Il est ainsi possible de basculer, sur le projet originel, les correctifs qui sont déployés sur sa propre version.

EXEMPLE (FIGURE 2)

Dans un **mode nominal**, le cycle de développement se déroule de la manière suivante :

1. Le développeur travaille sur sa branche **« feature »** pour implémenter les nouvelles fonctionnalités demandées, puis les propage via le dépôt central sur la branche **« devel ».**

2. Une fois les tests unitaires et fonctionnels concluants, la version de la branche **« devel »** est propagée sur la branche **« release ».**

3. L'utilisateur final peut désormais procéder à la qualification de la nouvelle solution :

- soit l'utilisateur valide la nouvelle version, dans ce cas le code de la branche **« release »** est propagé sur la branche **« prod »** et le code est déployé en production.
- soit l'utilisateur ne valide pas la nouvelle version. Le développeur apporte les correctifs nécessaires sur la branche **« devel »** jusqu'à la validation par l'utilisateur.

Dans une situation où une **anomalie critique** survient dans l'environnement de production :

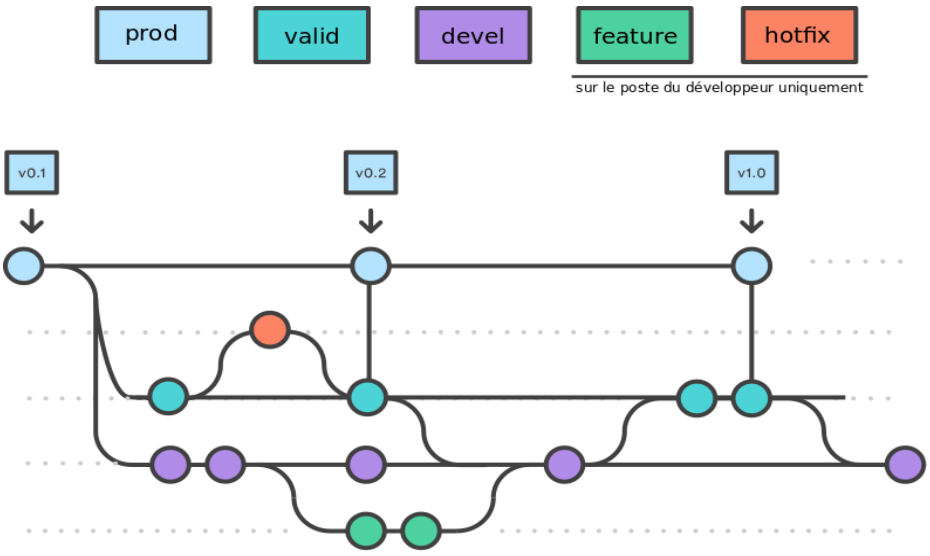
1. A partir de la branche **« release »,** le développeur crée une nouvelle branche **« hotfix »**.

2. il apporte les correctifs nécessaires et les fait valider par l'utilisateur final. Une fois validé, le code de la branche **« release »** est alors propagé sur la branche « **prod »** et le code est déployé en production. Les nouveaux correctifs sont alors propagés aussi sur la branche **« devel »** pour que ceux-ci ne soient pas perdus lors du développement de la future version.

Notons également que, dans une logique d'amélioration continue des processus, la survenue d'une anomalie critique doit permettre l'ajout de tests supplémentaires pour éviter qu'une situation identique ne se produise à nouveau.

Dans tous les cas, toutes les modifications sont tracées et « versionnées ».

Figure 2 **Exemple de cycle de développement du logiciel**



2.2. Écriture du code

2.2.1. Convention d'écriture

Pour que le code soit maintenable, c'est à dire lisible et compréhensible par les membres de l'équipe et extérieurs, il est important de suivre les conventions d'écriture en vigueur pour le(s) langage(s) de programmation utilisé(s). Par ailleurs, le code source doit être suffisamment commenté pour permettre sa compréhension par un tiers.

EXEMPLES

Python : https://www.python.org/dev/peps/pep-0008/

Shell : https://google.github.io/styleguide/shell.xml

2.2.2. Documenter l'exécution du code

Tout logiciel développé doit avoir un système d'historique des événements appelé « log » ou de messages pour informer les utilisateurs de l'avancement des analyses à chaque lancement, mais aussi pour identifier les sources probables des erreurs rencontrées. Le niveau de log doit pouvoir être défini par l'utilisateur. Il est recommandé d'utiliser un format de fichiers log de référence.

EXEMPLES

norme POSIX : https://standards.ieee.org/develop/project/1003.1.html

https://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format

2.2.3. Tests informatiques

Plusieurs types de tests informatiques existent et sont normalisés par le Comité Français des Test logiciels (http://www.cftl.fr). Afin de s'assurer de la fiabilité du code et d'éviter sa dérive, il est nécessaire de mettre en place une stratégie de tests informatiques.

REMARQUES

Certains logiciels utilisent des algorithmes non déterministes. Il en résulte un risque de trouver des résultats différents lorsque l'analyse est reproduite plusieurs fois. Généralement, l'utilisation de ce type d'algorithmes implique l'utilisation d'un générateur de nombres pseudo-aléatoires. Contrôler l'initialisation de ce générateur en choisissant la graine permet de garantir la reproductibilité. Des mesures doivent être prises pour maîtriser ce risque et il est nécessaire de prouver la bonne reproductibilité de ce type de logiciels.

2.2.4. Relecture croisée du code

Les tests informatiques peuvent également être appuyés par des stratégies de relecture du code (« code reviewing », « pair programming ») intra ou inter-équipe afin de mettre en évidence d'éventuelles erreurs, défauts de conception, tests manquants ou manquements à la convention d'écriture.

2.3. Containérisation et déploiement

Le principe de containérisation consiste à isoler le logiciel informatique du système hôte sur lequel il est installé. Selon cette méthode, le logiciel embarque tout ou partie d'un système d'exploitation nécessaire à son fonctionnement.

La containérisation d'un logiciel présente plusieurs avantages, principalement liés à la maîtrise des dépendances associées au logiciel :

Facilité de déploiement :

- permet d'assurer que les différents environnements de travail (ex : « devel », « release », « prod ») sont bien strictement identiques et garantit un fonctionnement équivalent du logiciel lors du déploiement des nouvelles « releases » dans des environnements matériels distincts ;
- généralement, l'équipe de développement n'est pas l'équipe assurant le déploiement des logiciels, ce qui nécessite un transfert d'informations plus ou moins lourd entre elles pour la réussite d'un déploiement en production. Avec l'utilisation d'un système de containers, l'information se résume à une commande unique permettant d'augmenter le nombre de « releases » sur un temps équivalent ;
- facilite la distribution du logiciel car la première installation ne nécessite pratiquement aucun prérequis particulier (agnostique de l'OS utilisé, des versions de langage et librairies déjà présentes), excepté l'outil permettant l'exécution du système de containers ;
- Robustesse du logiciel en réduisant les dépendances au système sous-jacent et ainsi les effets de bords d'une mise à jour de dépendances partagées entre différents logiciels.

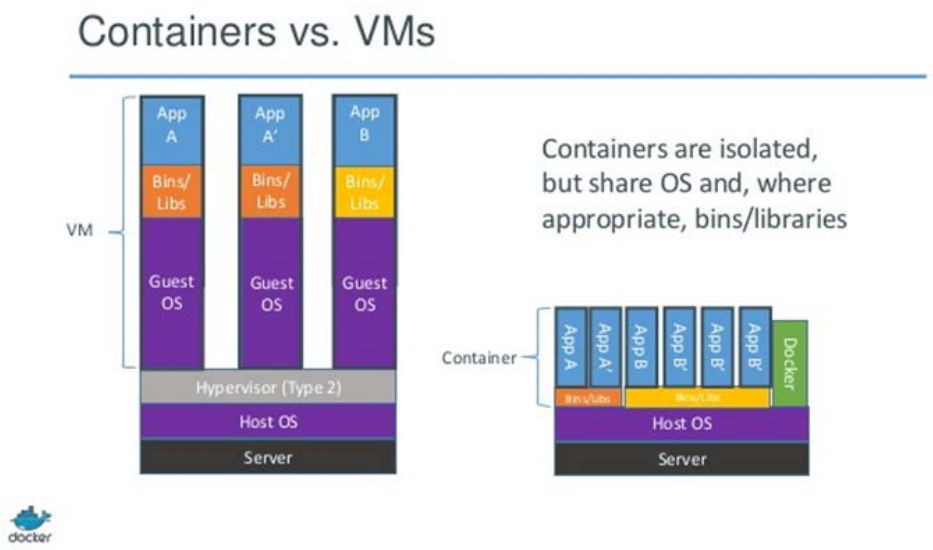
Avant de démarrer un projet de développement, il est important de définir en amont si une méthode de containérisation va être employée, car cela a un fort impact d'une part sur les compétences requises par les développeurs, et d'autre part sur la stratégie de développement. En effet, avec l'utilisation d'un système de containérisation, il convient de découper intelligemment son logiciel (par exemple : base de données/Front-end/Back-end) afin de pouvoir faire évoluer indépendamment les différents éléments en fonction du besoin.

On peut distinguer deux stratégies distinctes de containérisation :

- la containérisation stricte (ex : Docker) ;
- les machines virtuelles (ex : VirtualBox).

La différence principale entre ces deux stratégies réside dans le fait qu'une machine virtuelle embarque l'intégralité des composants du système d'exploitation et est strictement indépendante du système hôte alors que la containérisation stricte embarque les librairies systèmes spécifiques du logiciel mais reste dépendante du système hôte dont elle utilise les composants principaux.

Figure 3 Représentation schématique des différentes stratégies de containérisation



Ainsi, une machine virtuelle se doit de contenir les données d'entrée du logiciel et génère les données de sortie en son sein. Pour la containérisation stricte, aucune donnée d'entrée ni de sortie n'est introduite ou produite au sein du container. Le lien entre données et logiciel est réalisé par un « mapping » de répertoires entre le système hôte et le container. Des données temporaires peuvent toutefois être produites dans un container puisqu'elles seront supprimées avec l'instance.

Par nature, la containérisation stricte propose de meilleures performances, car elle est moins gourmande en ressources et totalement extensible. Pour augmenter les ressources allouées à un logiciel en container, il suffit de démarrer autant de containers que nécessaire pour répondre au besoin (micro-services).

D'une manière générale, réaliser un projet de développement de logiciels avec une stratégie de containérisation demande plus de temps de développement au démarrage mais représente un réel avantage sur le long terme en optimisant les temps de déploiement et en assurant la robustesse de son application.

REMARQUES

Un container pèse généralement moins de 1 Go tandis qu'une machine virtuelle peut peser jusqu'à plusieurs dizaines de Go. Il est à noter que pour des raisons de sécurité, il est parfaitement possible de combiner les deux stratégies en utilisant des containers dans une machine virtuelle.

3. GESTION DOCUMENTAIRE

3.1. Documentation technique

En plus de la documentation interne au code, une documentation technique devra être consultable en dehors du code source. Cette documentation doit reprendre les différentes fonctions du code et leurs utilités et fournir une aide à toute personne voulant reprendre le développement du projet. Ce type de documentation peut-être écrit ou généré automatiquement si la documentation interne au code suit un format compatible avec un système de génération de documentation (ex : Sphinx pour Python).

Cette documentation technique doit décrire les prérequis à l'installation et à l'utilisation du logiciel. Elle doit décrire les ensembles de tests mis en place au cours du développement. Elle doit être rendue disponible et peut être utile aux laboratoires cherchant à qualifier le logiciel utilisé pour une analyse biomédicale

3.2. Documentation pour l'utilisateur final

La documentation « utilisateur » doit permettre à une personne extérieure au projet de prendre en main les différents outils. Cette documentation peut-être disponible au format PDF ou sous forme de wiki ou de page web. En plus de la documentation technique du logiciel disponible décrite précédemment, doivent s'ajouter une description des éléments du pipeline et une description précise des entrées et sorties des logiciels.

3.2.1. Description des outils

Une représentation schématique du pipeline doit être mise à disposition. Il convient de décrire chaque outil du pipeline traitant les données de séquençage et d'en préciser la fonction attendue. Il est conseillé de documenter la commande native utilisée en cas d'intégration du code source ou du pilotage d'un logiciel tiers. Il est également souhaitable de documenter le paramétrage fin de chaque outil si le pipeline est déjà qualifié et intégré à une méthode de laboratoire validée au sens de la norme ISO 15 189. Avec un pipeline ainsi décrit, une reproductibilité de l'analyse entre différents sites est donc envisageable pour une même application. Il est important de documenter la version (ou la date de téléchargement si inexistante) et les références (ex : site web du téléchargement, article scientifique, etc.) de chaque dépendance, logiciel ou paquet informatique utilisé.

3.2.2. Descriptif des résultats attendus

La documentation utilisateur doit intégrer un descriptif complet des options de chaque pipeline. Il est recommandé de proposer une configuration par défaut qui correspond à celle validée par un laboratoire qui utilise le logiciel en routine diagnostique. L'impact prévu de la modification d'un paramètre devrait être mentionné.

Cette documentation doit présenter les produits finaux du pipeline. Un lexique et un dictionnaire peuvent être proposés afin d'éviter des erreurs d'interprétation des résultats. Si les résultats sont présentés de manière dynamique dans une interface (contrairement à des rapports PDF ou de type tableur), une notice dédiée d'utilisation doit être fournie.

3.3. Documentation pour l'administrateur et le service informatique

3.3.1. Prérequis informatiques

La documentation doit contenir la liste des prérequis informatiques, c'est à dire la description de l'architecture et des performances minimales à mettre en place pour le bon fonctionnement du logiciel. Elle doit également décrire les prérequis à l'installation du système (système d'exploitation, configuration machine et dépendances)

3.3.2. Descriptif des dépendances

La documentation doit également contenir la liste des librairies et packages utilisés pour le développement ainsi que leurs licences de diffusion s'ils sont distribués conjointement avec le projet. De cette manière l'utilisateur sera averti s'il doit acquérir une quelconque licence. Un référencement rigoureux des versions est indispensable. Il est nécessaire que soient indiqués un système d'exploitation ainsi que sa version, ou au minimum le système utilisé au cours de la phase de recette.

3.3.3. Procédure d'installation système

La documentation doit contenir une procédure d'installation détaillée sur un ou plusieurs systèmes d'exploitation. Les différentes commandes doivent être listées et accompagnées d'explications précises pour permettre à l'utilisateur, si son système d'exploitation n'est pas disponible dans la procédure, d'être autonome dans son installation.

3.3.4. Configuration des outils

Si un ou plusieurs outils du projet nécessitent une configuration, un paragraphe détaillant pour chacun d'entre eux la procédure à suivre doit également être présent dans la documentation. Dans ces cas, des exemples ou cas d'utilisation concrets sont recommandés.

4. IDENTIFICATION DES RISQUES

L'analyse des risques peut être effectuée selon une méthode de référence existante (5M, AMDEC…). Dans le cas du développement d'outils bioinformatiques, tout élément risquant de compromettre la mise à disposition du produit dans un délai adéquat peut être considéré comme un risque.

L'identification des risques relatifs à l'utilisation du pipeline par les laboratoires relève plutôt des étapes de qualification du pipeline et peut notamment s'appuyer sur le guide de qualification des pipelines rédigé et diffusé par l'Association nationale de praticien de génétique moléculaire (ANPGM).

EXEMPLES

Il existe des risques propres aux activités de bioinformatique. On peut notamment citer :

- le départ d'un développeur : le nombre de bio-informaticiens présents dans les laboratoires étant souvent limité, des départs peuvent avoir de grandes conséquences sur la mise en œuvre des projets ;
- la disparition, ou l'arrêt de maintenance des dépendances ;
- l'obsolescence des outils : les technologies de séquençage sont en développement rapide. Dans ce contexte, il est impératif d'avoir un cycle de développement court, adapté au besoin de réactivité des utilisateurs.

5. FORMATION

Le succès d'une démarche de développement bioinformatique suppose que la livraison de l'outil à l'utilisateur soit accompagnée des formations adéquates qui garantiront qu'il sera utilisé selon les règles de l'art. Cet aspect est particulièrement important dans le cadre de la bioinformatique clinique. Cette étape de formation peut être articulée par niveau, depuis l'utilisateur simple à l'expert, et donner lieu à un certificat qui pourra faire partie d'une démarche d'habilitation.

Édité par l'Institut national du cancer (INCa)

Conception : INCa
Réalisation : INCa
Illustrations : INCa

ISBN : 978-2-37219-362-7
ISBN net : 978-2-37219-363-4

DEPÔT LÉGAL FÉVRIER 2018

RÉF : OPLOGDIAGN18

e-cancer.fr

INSTITUT NATIONAL DU CANCER