Stabilisation de la plateforme Geomedia

Auteur : Damien Dosimont

Date : 30/10/2015

# Introduction

L’objet du contrat concerne la stabilisation de la plateforme de collecte de flux RSS du projet Geomedia. Nous détaillons ici les différentes tâches constituant ce travail.

## Rappel des points d’intervention

Plusieurs tâches ont été définies lors d’un entretien avec Claude Grasland, le responsable du projet, et Hugues Pecout, l’ingénieur géomaticien impliqué dans le projet.

* Résolution d’un problème d’interruption de la collecte. La plateforme de collecte de flux RSS possède un comportement instable qui aboutit à des ruptures de collecte pendant des périodes allant de plusieurs heures à plusieurs jours, pour des raisons à présent indéterminées. L’objectif de la tâche est d’identifier la source du problème est de la corriger.
* Résolution de problèmes liés à l’extraction des données depuis la base de données contenant les flux RSS. On distingue deux comportements indésirables de l’application :
  + Lors de la tentative d’une extraction à partir de l’interface graphique accessible à partir du site internet de l’application (<http://geomedia.huma-num.fr/RSSAgregate>), censée permettre le téléchargement de fichiers csv contenant les flux sélectionnés par l’utilisateur pour un export depuis la base de données, le site internet génère une page d’erreur de type 404 (page internet non disponible)
  + Certaines extractions produisent des fichiers csv incomplets dont il manque une partie des flux.
* Amélioration de l’ergonomie générale de l’application. Certaines tâches apparaissent fastidieuses à l’utilisateur à cause de la nécessité d’effectuer manuellement un grand nombre d’opérations (ou clics). Les principes généraux de conception d’une interface graphique préconisent au contraire de minimiser le nombre d’interactions requis pour effectuer une tâche.

## Tâches décelées à postériori

Suite à une période de prise en main de l’application, mais aussi au cours de mes interactions avec les informaticiens d’Huma-num en charge de l’administration du serveur hébergeant l’application, j’ai défini de nouvelles tâches indispensables à la stabilisation de la plateforme Geomedia :

* Réutilisabilité de l’application. Il est nécessaire de faire en sorte que l’application puisse facilement être prise en main, c’est-à-dire compilable, installable et configurable facilement, et ce, dans le but de minimiser le délai d’opérabilité de tout personnel amené à la modifier ou à l’améliorer.
* Co-administration du serveur. Certaines tâches d’administration du serveur hébergeant l’application nécessitent une connaissance de la structure de cette dernière et de ses processus. Elles ne peuvent donc être réalisées par les administrateurs d’Huma-num sans une expertise préalable.

# « Réutisabilité » de l’application

## Motivations

L’état initial (c’est-à-dire au début de ce contrat, à t0) de l’application ne permet pas de recompiler ses sources, de l’installer et de la configurer dans un temps raisonnable. Plusieurs raisons expliquent cela :

* Aucun « packaging » de l’application n’a été réalisé. Cela signifie que les sources, fournies sous la forme d’un projet NetBeans, ne peuvent pas être compilées telles quelles sans un traitement préalable.
* il n’existe pas de version opérationnelle parfaitement définie, ni d’historique de développement clairement établi. Il n’y a aucune certitude que les sources qui m’ont été fournies correspondent à la version de production qui est actuellement utilisée par les membres du projet Geomedia.
* La structure de l’application et sa complexité ne permettent pas une prise en main rapide. En particulier, nous distinguons la présence de nombreux fichiers de configuration, et de fichiers générés par l’application à de multiples emplacements.
* L’organisation des fichiers et répertoires sur le serveur la stockant et l’exécutant, mal définie, et asymétrique entre la version de production et de développement.
* La documentation existante est sommaire et insuffisante. En particulier, elle ne détaille pas les étapes de prise en main (compilation, configuration et installation), ni même la structure de l’application, l’organigramme de ses tâches, l’ensemble de ses fonctionnalités, ou les modèles et structure de données employés. Il n’existe pas de cahier des charges définissant ses spécifications.
* L’absence, à t0, d’un environnement de développement (serveur tomcat correctement configuré pour exécuter l’application, accès à une base de données de développement).

L’objectif de cette première tâche est de palier à ces différents problèmes afin de permettre de fournir aux futurs développeurs éventuels de l’application le matériel capable de diminuer drastiquement le temps de prise en main de l’application.

## Actions

Plusieurs actions ont été mises en place pour répondre à cet objectif.

### Gestion de version

La première est a mise en place d’une gestion de version sur le site internet GitHub (adresse du projet : <https://github.com/geomedia> , qui repose sur le protocole de gestion de version git. L’intérêt de ce genre de technologies est d’assurer un hébergement des sources de l’application, mais également la mise en place d’un historique de développement permettant de revenir à des états de développement précédents de l’application. Les avantages de GitHub, par rapport à d’autres solutions fondées sur le même principe, sont multiples :

* La possibilité d’associer plusieurs dépôts à un projet, permettant une organisation plus claire des différents éléments qui le constituent.
* La présence d’un « bugtracking » sous la forme d’ « issues », qui permettent de centraliser les informations sur les bugs actuels de l’application, et d’archiver les discussions autour de ces problèmes entre les différents intervenants.
* La possibilité d’associer à chaque dépôt des pages wiki, idéales pour héberger des tutoriels ou autres informations utiles aux développeurs et utilisateurs.
* La présence d’outils de gestion de projet et d’attributions des tâches, liées au concept des issues.

Certains points négatifs sont toutefois à noter :

* Le caractère privé de l’hébergeur peut poser problème quant à la pérennité du stockage. Néanmoins, le protocole git étant décentralisé, il est possible de sauvegarder les dépôts sur d’autres serveurs de stockage, comme la forge Inria par exemple, et même de mettre au point facilement des scripts assurant la synchronisation des différents dépots.
* Les sources sont publiques, il est donc important de ne jamais mettre d’information confidentielles dans le dépôt, en particulier des mots de passes. Il existe des solutions pour supprimer de l’historique toute information critique qui aurait été mise en ligne malencontreusement.
* Il y a redondance et concurrence entre l’emploi de la forge Inria, qui contient certains matériaux du projet (projet geomediamapper), et l’emploi de GitHub. Il serait préférable de tout centraliser.

### Compilation

L’état des sources de l’application à t0 ne permet pas leur compilation immédiate. Cela est dû à l’absence des bibliothèques dont est dépendante l’application et à des erreurs dans le code. Les bibliothèques ont été incorporées au dépôt, tandis que j’ai supprimé les tests, apparemment obsolètes, contenant des erreurs. J’ai renommé l’application, originellement nommée « RSSAgregate », en « rssaggregate », ce qui permet d’estampiller cette nouvelle version, désormais fonctionnelle.

Je propose plusieurs recommandations :

* Afin de mieux gérer les dépendances avec des bibliothèques externes, je préconise l’emploi d’un gestionnaire de dépendances tel que maven. Ceci n’a pas été réalisé car ce n’est pas une tâche de haute priorité.
* Tirer parti du mécanisme de branches lié à git pour séparer les versions de développement et les versions stables.
* Ne jamais « commiter » et envoyer sur le dépôt, dans la branche principale master, correspondant à la version stable, une version contenant des erreurs empêchant la compilation. De manière générale, il convient d’effectuer une procédure de tests avant de valider ces versions.

### Installation et configuration

J’ai mis en place des scripts permettant d’installer l’application facilement, à partir d’une simple commande Linux. J’ai identifié et centralisé l’ensemble des fichiers de configuration. Ces derniers seront installés dans les emplacements adéquats par le script d’installation. Ceci permet de simplifier le déploiement de l’application sur d’autres serveurs.

### Documentation

J’ai rédigé des pages wiki expliquant comment prendre en main le framework. Elles décrivent la procédure d’importation du projet dans l’IDE NetBeans, sa compilation, sa configuration et son installation, de même que l’organisation globale des fichiers et structure sur le serveur exécutant l’application. Ces pages sont disponibles à l’adresse suivante : <https://github.com/geomedia/rssaggregate/wiki>

Mes recommandations :

* Mettre à jour la documentation le plus souvent possible
* Indiquer les bugs ou les fonctionnalités à implémenter en utilisant le mécanisme d’issues fournies par GitHub, même si aucun développeur ne maintient l’application, afin de laisser une trace écrite et accessible.

# Co-administration du serveur

## Motivations

Les tâches d’administration du serveur imputent en théorie aux administrateurs de la plateforme Huma-num. Néanmoins certaines opérations nécessitent une connaissance de la structure et des processus de l’application, et l’interaction avec le développeur en charge de l’application, voire la réalisation de tâches d’administration par ce dernier, est nécessaire.

Plusieurs problèmes ont été soulevés par les administrateurs d’Huma-num.

* D’une part, l’organisation « anarchique » des répertoires. En particulier, l’asymétrie entre l’organisation des répertoires de la version de développement et de la version de production. Ils déplorent aussi la présence d’une grande quantité de répertoires ou fichiers reliquats, c’est-à-dire de versions obsolètes de sauvegarde encombrant le serveur. Le nom des répertoires et leur rôle est parfois obscur. Le principal problème lié à la réorganisation de ces répertoires est la dépendance avec les fichiers de configuration employés par l’application pour déterminer où chercher d’autres fichiers de configuration, ou où générer certains fichiers (logs, fichiers temporaires, fichiers de contrôle, etc.).
* D’autre part, le volume des fichiers générés par l’application et le serveur peuvent atteindre des dizaines de Go. Certains de ces fichiers étant archivés régulièrement par Huma-num, il est nécessaire de supprimer ceux dont la présence est dispensable. Parmi ces fichiers, on distingue principalement les logs, qui sont des fichiers de trace contenant des informations sur le déroulement de l’application, l’exécution du serveur, des évènements particuliers survenus, ainsi que l’occurrence d’erreurs, les extractions, qui sont des fichiers générés lorsqu’un utilisateur souhaite exporter une partie ou une sous-partie de la base de données dans un format csv lisible par un humain ou par un programme tiers, et enfin les dumps de la base de données, qui sont des sauvegardes de son état à un instant donné.

## Actions

Voici l’ensemble des actions effectuées pour résoudre ces différents problèmes.

### Réorganisation des répertoires

L’organisation des répertoires a été symétrisée. Les répertoires obsolètes ont été supprimés, mais sauvegardés dans des dépôts du projet sur le site GitHub. J’ai identifié et mis à jour les fichiers contenant les dépendances, renseignés sur le wiki. Les logs générés par l’application sont désormais centralisés dans un répertoire plutôt que d’être éparpillés. Enfin, la redirection vers la page d’accueil <http://geomedia.huma-num.fr/> , qui générait une erreur, a été corrigée, et pointe désormais vers <http://geomedia.huma-num.fr/RSSAgregate> .

Mes recommandations :

* Rassembler tous les types de logs (ceux générés par le serveur et par l’application, pour les versions de développement et de production) à un seul endroit.
* Séparer complètement :
  + Les fichiers de configuration
  + Les fichiers générés par l’application (fichiers temporaires, fichiers de contrôle)
  + Les logs
* Les extractions devraient être dans un répertoire externe au répertoire contenant les sources du site web. De manière générale, il convient de ne pas générer de fichiers là où se trouvent des sources.

### Nettoyage

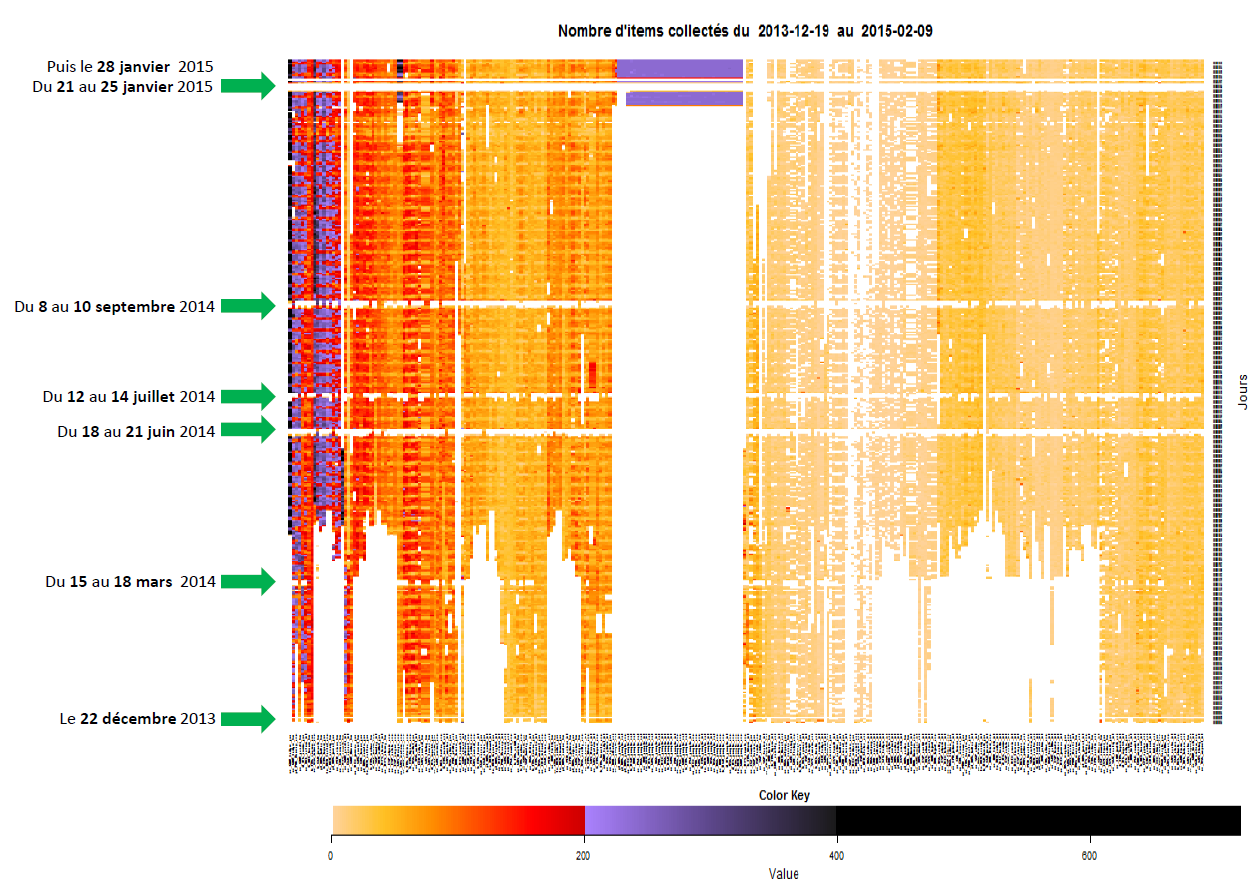
Les logs de la version de développement ont été supprimés, mais pas ceux de la version de production, qui sont nécessaire pour les investigations relatives au problème de rupture de la collecte. Les dumps de la base de données effectués par le développeur originel de l’application ont été supprimés. J’ai corrigé un bug de l’application qui rendait la suppression des extractions obsolètes inopérantes. Néanmoins, j’ai désactivé cette tâche, qui devrait être effectuée par un script externe.

Mes recommandations :

* Définir une politique claire de sauvegarde/suppression des fichiers
* Cela doit être concomitant avec l’établissement d’une cartographie précise des répertoires qui contiennent ces fichiers.
* Il est essentiel de supprimer les extractions, qui prennent un volume conséquent d’espace disque. Une durée de vie de deux semaines semble raisonnable en regard du nombre d’utilisateurs actuels.

# Interruption de la collecte

## Rappel du problème



Graphique représentant la collecte des flux RSS (abscisse) au cours du temps (ordonnée)

La collecte des flux RSS effectuée par l’application est interrompue de manière récurrente. Ci-dessous, une liste des symptômes relatifs à ce problème :

* Les ruptures de collecte ont une durée variable de plusieurs minutes à plusieurs jours
* Certains flux sont parfois épargnés
* L’observation a pu être faite à partir des dates de récupération associée à chaque objet collecté
* Les perturbations sont visibles après extraction et également grâce aux graphiques générés par l’application web
* Un script Huma-num permet de détecter les coupures en évaluant la date du dernier objet inscrit dans la base de données
* Si deux applications sont actives en même temps (exécutées sur deux serveurs différents, par exemple), on constate que les ruptures de collecte ne surviennent pas de manière concomitante.

## Hypothèses

Plusieurs hypothèses sont envisageables pour expliquer ce comportement. Nous démontrons l’invalidité d’une première série d’entre elles :

* Coupure du serveur ou de l’application. Non plausible car certains flux sont toujours récupérés, et des logs sont toujours générés pendant les périodes de rupture.
* Coupure du réseau. Peu plausible car l’application reste accessible pendant les périodes de rupture.
* Erreur d’assignation de la date. Les logs d’erreur corroborent l’état anormal de l’application par la quantité importante d’exceptions, puis l’interruption progressive des tâches de collecte qui ne donnent plus de signes de vie.

D’autres hypothèses, au contraire, semblent plus crédibles :

* Un problème de synchronisation, tel qu’un interblocage des tâches de collecte, est suspecté. La complexité de l’application, sa structure sous forme de tâches exécutées en parallèle et la nécessité de synchroniser l’accès à certaines ressources partagées (telle que la base de données, mais également des variables) peut favoriser ce genre de comportement s’il existe des défauts de conception.
* La configuration des timeouts : la durée du timeout de chaque type de tâche est définie dans un fichier de configuration. Une valeur trop faible peut empêcher les tâches de s’exécuter correctement s’il y a un peu de latence sur le réseau.
* L’abandon des mécanismes de rétrocontrôle : les vestiges d’un mécanisme de détection d’interblocage sont visibles dans le code, mais ne sont plus actifs.

## Actions

Plusieurs actions ont été entreprises afin de résoudre ce problème.

### Analyse du code et des logs

L’investigation du problème s’est principalement fondée sur l’analyse des logs, qui contiennent une trace des évènements survenant pendant les ruptures de collecte, et en particulier les erreurs, et l’analyse du code de l’application elle-même. Les erreurs contenues dans les logs sont exprimées sous la forme d’un message désignant l’ensemble des fonctions appelées par l’application qui déclenchent une exception, c’est-à-dire un évènement problématique qu’il est nécessaire de traiter par une procédure particulière.

Voici un exemple de message généré par une exception :

java.net.SocketTimeoutException: connect timed out

Log Erreur : java.net.SocketTimeoutException: connect timed out at java.net.PlainSocketImpl.socketConnect(Native Method) at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:339) at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:200) at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:182) at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392) at java.net.Socket.connect(Socket.java:579) at sun.net.NetworkClient.doConnect(NetworkClient.java:175) at sun.net.www.http.HttpClient.openServer(HttpClient.java:378) at sun.net.www.http.HttpClient.openServer(HttpClient.java:473) at sun.net.www.http.HttpClient.(HttpClient.java:203) at sun.net.www.http.HttpClient.New(HttpClient.java:290) at sun.net.www.http.HttpClient.New(HttpClient.java:306) at sun.net.www.protocol.http.HttpURLConnection.getNewHttpClient(HttpURLConnection.java:995) at sun.net.www.protocol.http.HttpURLConnection.plainConnect(HttpURLConnection.java:931) at sun.net.www.protocol.http.HttpURLConnection.connect(HttpURLConnection.java:849) at org.jsoup.helper.HttpConnection$Response.execute(HttpConnection.java:425) at org.jsoup.helper.HttpConnection$Response.execute(HttpConnection.java:410) at org.jsoup.helper.HttpConnection.execute(HttpConnection.java:164) at org.jsoup.helper.HttpConnection.get(HttpConnection.java:153) at rssagregator.services.tache.TacheDecouverteAjoutFlux.callCorps(TacheDecouverteAjoutFlux.java:117) at rssagregator.services.tache.TacheImpl.executeProcessus(TacheImpl.java:192) at rssagregator.services.tache.TacheImpl.call(TacheImpl.java:221) at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334) at java.util.concurrent.FutureTask.run(FutureTask.java:166) at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(ScheduledThreadPoolExecutor.java:178) at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:292) at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615) at java.lang.Thread.run(Thread.java:724)

Dans les phases de rupture de collecte, un grand nombre d’exceptions est de type timeout, c’est-dire que la tâche exécutée prend trop de temps et est interrompue par le système. Un interblocage pourrait causer ce genre de phénomène, la tâche restant bloquée à cause d’une mauvaise synchronisation.

Néanmoins, remonter au code à partir des logs ne permet pas d’estimer l’ensemble des processus pouvant aboutir à un état problématique de l’application, la structure de cette dernière étant trop complexe pour être appréhendée à partir du code source uniquement. Plus de documentation est indispensable pour estimer les relations entre les tâches, la structure globale de l’application, et la signification d’une erreur (ses causes possibles, par exemple).

### Conception d’un « chien de garde »

En informatique, un chien de garde est un logiciel utilisé pour s’assurer qu’un programme ne reste pas bloqué à une étape particulière d’un traitement. Le chien de garde redémarre le système si une action définie n’est pas réalisée dans un temps imparti.

Ici, j’ai pu tirer parti d’un script conçu par les administrateurs d’Huma-num, qui avertit les membres du projet lorsque la collecte est perturbée, en se fondant sur la date du dernier objet écrit dans la base de données. J’ai modifié ce script, qui, dorénavant, redémarre le serveur tomcat exécutant l’application, lorsque ce délai est dépassé. Ce script est exécuté à l’aide de cron, un service linux qui permet de programmer l’exécution périodique de scripts ou de programmes, toutes les heures, et le seuil d’inactivité de la base de données au-delà duquel le serveur est redémarré est deux heures. Ces paramètre pourront être ajustés si jamais ils s’avéraient ne pas être optimaux.

Listons les avantages de cette technique :

* Le chien de garde réinitialise l’état de l’application, ce qui permet de sortir d’un interblocage, et d’éviter des ruptures de collecte longues.
* Il assure que le serveur est bien actif
* Il est simple à utiliser, à configurer et à modifier (faire appel à Huma-num le cas échéant)
* Il est robuste
* Il est externe à l’application, ce qui permet un meilleur rétrocontrôle de celle-ci. En effet, si l’application vient à dysfonctionner, cela n’aura aucun impact sur l’exécution du chien de garde.

Cependant, il existe plusieurs points négatifs :

* Tout utilisateur actif sur le site sera éjecté lors d’un redémarrage
* Le temps de redémarrage du serveur est à prendre en compte, même s’il semble assez court (de l’ordre de la minute)
* Le chien de garde s’attaque au symptôme, mais ne résout pas le problème initial
* Il ne s’active pas si certains flux sont encore actifs
* L’application prend un certain temps avant de se stabiliser après son démarrage (de l’ordre de l’heure). Il faudra veiller à configurer correctement le script de manière à prendre ce délai en compte, afin d’éviter de redémarrer le serveur en boucle avant la fin du délai d’initialisation et d’empêcher l’application de se remettre à collecter les flux.
* Il est probable que cela n’empêche pas la perte de certains flux. Cela limitera cependant la perte d’une grande quantité de flux à cause d’une longue rupture.
* Le redémarrage de l’application uniquement, qui semble une solution plus satisfaisante, ne fonctionne pas correctement, d’où la nécessité de redémarrer le serveur.

# Tâches non traitées

L’ajout à postériori des tâches liées à la réutisabilité et à la co-administration du serveur, dont la priorité était élevée, a entrainé le report des tâches relatives à l’extraction et à l’amélioration de l’interface, qui n’ont donc pas été réalisées.

# Recommandations générales

Après analyse de l’application, de sa structure, de ses fonctionnalités et de son code source, au cours de la réalisation des différentes tâches de ce contrat, j’ai pu également dresser un bilan général et en dégager un ensemble de recommandations.

Si l’application de collecte représente une preuve de concept très satisfaisante, sa qualité est insuffisante pour en faire un produit fini et pérenne. Globalement, on peut dénoter un manque de stabilité et de fiabilité, à cause d’un certain nombre de bugs et de dysfonctionnements, qui ont un impact sur la qualité de la collecte, mais aussi de l’expérience utilisateur. Sa structure complexe et le manque de documentation rendent toute intervention difficile et coûteuse en termes de temps.

Dans l’optique de produire un logiciel de qualité professionnelle utilisable en production, je recommande, plutôt que la réparation et l’amélioration de ce premier prototype, la conception d’une nouvelle application, dont les fonctionnalités seraient similaires, mais avec des contraintes de qualité, de stabilité et de fiabilité accrues. Je rappelle ici un certain nombre de règles et principes de développement logiciels qui pourraient aider à réaliser ce produit dans les meilleures conditions possibles.

## Elaboration d’un cahier des charges

La première étape de la conception d’une application est l’élaboration d’un cahier des charges précis et rigoureux. Tout d’abord, les fonctionnalités de l’application doivent être spécifiées par le client (les membres du projet Geomedia) et le développeur. Ce dernier proposera, en conséquence, un environnement de développement, constitué des langages, technologies, bibliothèques, frameworks dont dépendra le code de l’application. Il est également important de spécifier ce qui sera configurable, et comment. Les modèles de données et la structure de l’application devront également être spécifiés, de même que son interface. Il est nécessaire d’établir une cartographie de l’ensemble des fichiers générés par l’application, et de définir dès le début une politique claire de sauvegarde des fichiers.

## Quelques méthodologies de développement

Le cahier des charges, bien qu’il doive être pensé en amont de la conception de l’application, ne doit pas être trop rigide et peut être remodelé tout au long du développement. Certaines méthodologies de développement intègrent l’idée d’une conception dynamique et évolutive. Je conseille la lecture d’articles sur les méthodes agiles, telles que scrum ou extrem programing. De manière générale, je ne saurais que trop conseiller la conception d’une application simple dont la complexité répond de manière pragmatique aux besoins en termes de fonctionnalités et de performance.

## Définir les besoins en termes de performance

Selon moi, un des principaux facteurs d’instabilité de l’application de collecte actuelle est sa complexité. En effet, sa structure fait appel à un ensemble de tâches hétérogènes exécutées en parallèle, ce qui nécessite l’emploi de mécanismes de synchronisation pour accéder à des ressources partagées. De manière générale, la programmation parallèle est réputée difficile à maîtriser et requiert de définir rigoureusement la structure de l’application et l’interaction entre les tâches ou processus exécutés en parallèle. Il est également utile d’évaluer l’accélération induite par la parallélisation du traitement. Dans le cas de la collecte de flux RSS, la principale contrainte en termes de performances est la bande passante du réseau et des serveurs hébergeant ces flux. Les traitements de mise en forme ne nécessitent pas une forte puissance de calcul. Cet aspect est à prendre en compte afin de déterminer quelles tâches doivent être parallélisées (accès aux flux rss), et quelles tâches peuvent être réalisées de manière séquentielle (dédoublonnage, mise en forme, stockage dans la base de données).

## Définir la structure de l’application avant de commencer le développement

Il est important de formaliser la chaîne de traitement, et de séparer le traitement en étapes, à la façon d’un pipeline, dans le but d’améliorer la lisibilité. Il est également indispensable de définir un modèle de données clair, et ne pas hésiter à employer des bases de données locales pour stocker des traitements intermédiaires. Il est également envisageable de déporter certaines tâches annexes (suppression de reliquats, envoi de mails), qui peuvent être exécutés par des scripts externes, voire de découper le framework en plusieurs applications indépendantes exécutées sur le serveur.

## Utiliser des outils de gestion de projet

Ces outils permettent une meilleure organisation et visibilité du travail du développeur, mais aussi une meilleure interaction avec les partenaires du projet. On les utilisera pour

* Décrire les tâches à réaliser
* Prévoir un suivi et définir des objectifs hebdomadaires
* Permettre aux utilisateurs/clients de formaliser leurs demandes, renseigner des bugs

L’emploi d’un gestionnaire de version est indispensable.

## Conception de l’interface

Concernant l’interface, il est intéressant de se pencher sur la réalisation d’une interaction non graphique avec l’application via un terminal en plus d’une interface graphique, utile pour réaliser des tâches complexes ou répétitives grâce à l’élaboration de scripts.

Au niveau de l’ergonomie, il est nécessaire d’identifier avec les utilisateurs les étapes coûteuses en termes de clics. Enfin, il est indispensable de réfléchir à une solution viable pour le téléchargement des fichiers produits lors de l’extraction, qui s’avèrent généralement volumineux (par exemple, envoi d’un mail avec le lien du téléchargement, utilisation du protocole ftp ou ssh, solution externe de transfert de fichiers, etc.).

## Gestion des erreurs

Une bonne pratique concernant la gestion des erreurs est tout d’abord de documenter celles dont la susceptibilité de se produire est probable. Il est nécessaire d’intercepter les exceptions, de retourner des messages d’erreur clairs et interprétable y compris par des utilisateurs dont les connaissances en informatique ne sont pas avancées. Il est également utile de prévoir des mécanismes de rétro-contrôle, tout en veillant à ce qu’ils ne se chevauchent pas.

## Documentation

De manière générale, l’application doit être documentée. Cette documentation peut prendre la forme de wikis ou tutoriels à destination d’autres développeurs ou des utilisateurs. Il est indispensable de fournir, avec l’application, sa structure, la chaine de traitements, et les modèles de données employés.

## Divers

J’aimerais également soulever d’autres points plus généraux. Il est impératif de respecter certaines contraintes relatives au stockage des mots de passe : ces derniers sont actuellement stockés dans certains fichiers de configuration, et cela peut représenter un risque en termes de sécurité à cause de leur intégration dans les dépôts git.

J’aimerais également souligner la nécessité de choisir une licence pour les sources du programmes, et ce, afin d’en clarifier les conditions d’utilisation et de modification. Il est à noter qu’en France, le droit d’auteur s’applique même si aucune licence n’a été définie (la valeur juridique de ces dernières est d’ailleurs sujette à controverse).