

J. Racaud; A. Simon; J. Harrault; J. Blondeel; S. Daguenet; F. Corradin

Music Sheet Writer

Music Sheet Writer

Stratégie de Tests

Objectifs du document

Résumé

Glossaire

***– R –***

Release : Mise à disposition d’une version de l’application. Elle peut être privée, semi publique ou publique. En général une release fait état de la version commerciale (ou plus détaillée pour des releases fermées).

Description du document

|  |  |
| --- | --- |
| Titre | Music Sheet Writer : Stratégie de Tests |
| Date de création | 22/10/2015 |
| Date de publication | 15/11/2015 |
| Auteur | J. Racaud; A. Simon; J. Harrault; J. Blondeel; S. Daguenet; F. Corradin |
| Responsable | Jonathan Racaud |
| E-mail | musicsheetwriter\_2017@labeip.epitech.eu |
| Sujet | Stratégie de Tests |
| Version du modèle | 1.0 |

Tableau des révisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Auteur** | **Section(s)** | **Commentaire** |
| 22/10/2015 | Jeremy HARRAULT | Toutes | Création du document |
| 08/11/2015 | Julien BLONDEEL | Toutes | Rédaction |
| 11/10/2015 | Julien BLONDEEL | Toutes | Modification suite à la réunion |

Sommaire

[1. Introduction 1](#_Toc433295550)

[1.1. Objectif de l’EIP 1](#_Toc433295551)

[1.2. Principe de base du système 1](#_Toc433295552)

[1.3. Documents de référence 1](#_Toc433295553)

[2. Version/release 2](#_Toc433295554)

[3. Types de tests et périmètre 3](#_Toc433295555)

[4. Livrables 4](#_Toc433295556)

[4.1. Cas de tests 4](#_Toc433295557)

[4.2. Rapport d’exécution de tests 4](#_Toc433295558)

[4.3. Rapport sur le suivi global de la qualité 4](#_Toc433295559)

[5. Types de tests 5](#_Toc433295560)

[5.1. Tests XXX-1 5](#_Toc433295561)

[5.2. Tests XXX-2 5](#_Toc433295562)

[6. Estimation Globales de la charge de travail 6](#_Toc433295563)

[7. Outils 7](#_Toc433295564)

[7.1. Dépôt de tests 7](#_Toc433295565)

[7.2. Gestion de rejets/erreurs 7](#_Toc433295566)

[7.3. Autres outils 7](#_Toc433295567)

# Introduction

## Objectif de l’EIP

## Principe de base du système

## Documents de référence

# Version/release

Voici la liste des releases majeurs pour le logiciel. Leurs numéros de version seront notés sous la forme X.0 où X est le numéro de la release majeur. Ce nombre sera incrémenté de 1 à chaque release.

D’autres releases mineures pourront être faites si :

* Une fonctionnalité rapide à implémenter est fortement demandée par la communauté et/ou devient urgent à implémenter pour satisfaire l’expérience MusicSheetWriter.
* Des « bugs » sont trouvés sur le logiciel, soit par l’équipe de développement, soit pas la communauté.

Une release mineure sera noté sous la forme X.Y où X est le numéro de la release majeure en cours et Y est le numéro de la release mineure.

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro de version | Date de sortie prévue | Description | Fonctionnalités ajoutées |
| 1.0 | 18/03/16 | Cette version sera la première version du logiciel. Elle implémentera les fonctionnalités de bases du logiciel, à savoir la gestion de projet, la lecture et l’édition de partition. | **Gérer un projet :**   * créer un projet * ouvrir / fermer un projet * sauvegarder un projet * importer un projet * exporter un projet   **Lire une partition :**   * démarrer la lecture * mettre en pause la lecture * placer et déplacer le curseur de lecture   **Editer une partition :**   * Modifier le type de mesure * Placer / supprimer un élément musical * Modifier le tempo * Placer les curseurs d’édition * Ajouter les notes à la partition |
| 2.0 | 13/05/16 | Cette version sera tournée vers une des deux principales fonctionnalités du logiciel. Dans cette version, l’utilisateur pour générer une partition directement en jouant de son instrument branché par port MIDI. | **Générer une partition par port MIDI :**   * Enregistrer le morceau * Identifier la note * Définir la durée de la note * Sauvegarder la note |
| 3.0 | 30/09/17 | Cette version sera la version finale du logiciel. . Dans cette version, l’utilisateur pour générer une partition directement en jouant de son instrument branché par port JACK | **Générer une partition par port JACK :**   * Enregistrer le morceau * Décomposer le son en fréquence * Filtrer les pics de fréquence * Déterminer les harmoniques de l’instrument joué * Supprimer ces harmoniques du spectre fréquentielle * Identifier la note grâce à la fréquence fondamentale |

# Types de tests et périmètre

## Test composant UI

Ces tests ont pour but de simuler les interactions entre l’utilisateur et le logiciel. Pour chaque élément ajouté dans l’interface, tel qu’un bouton ou un menu, le développeur doit tester ce dernier, afin de s’assurer que l’action entrepris à la bonne conséquence (ouverture d’une nouvelle fenêtre, lancement de la lecture, etc.).

## Tests Unitaires

Ces tests ont pour but de confirmer qu'une portion de code fournit bien une sortie correcte en fonction de paramètres d'entrées donnés, ceci afin de s'assurer qu'elle répond aux [spécifications fonctionnelles](https://fr.wikipedia.org/wiki/Sp%C3%A9cification_(informatique)) et qu'elle fonctionne correctement en toutes circonstances. Ces tests seront effectués sur l’ensemble des fonctions qui sont utilisées dans le logiciel.

## Tests fonctionnels

Ces tests ont pour but de valider un dispositif complet et non pas une simple conversion d'entrées-sorties. C’est un ensemble de tests unitaires ayant pour but de valider le bon déroulement d’une fonctionnalité du logiciel, comme par exemple la création d’un projet ou l’édition d’une partition.

## Tests de non-régression

Ces tests ont pour but de vérifier que l’intégration ou la modification d’un bout de code n’a pas modifié le comportement du logiciel antérieurement validé. Ces tests portent sur l’exécution de tests unitaires déjà joués afin de s’assurer que le système répond toujours aux exigences spécifiées.

# Livrables

## Cas de tests

L’ensemble des tests pour le logiciel se trouveront dans un répertoire Test/Logiciel. Dans ce répertoire, les tests seront classé par fonctionnalités, par exemple nous aurons un répertoire, Test/Logiciel/Gérer un projet, qui regroupe l’ensemble des tests unitaires qui sont utilisés pour Gérer un projet.

#### Tests Fonctionnels

Les tests fonctionnels visent à tester le bon déroulement d’une fonctionnalité. Cela implique un ensemble de tests unitaires qui, une fois tous réalisés avec succès, permet au test d’être validé.

|  |  |
| --- | --- |
| *Chaque test comportera…* | *Afin de…* |
| Une courte description présentant le test | Connaitre la fonctionnalité testée |
| La liste des tests unitaires utilisés | Savoir quelle valeur est testé, les conditions à succès. |

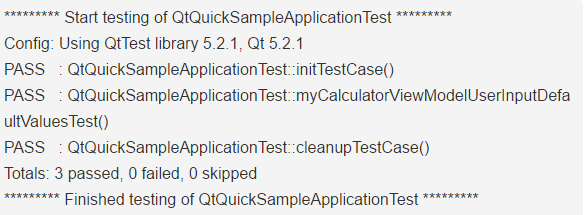
#### Tests Unitaire

Un test unitaire est une simple fonction C++, qui fait appel à la librairie QTestLib, pour valider la valeur de retour de la fonction testé.

|  |  |
| --- | --- |
| *Chaque test comportera…* | *Afin de…* |
| Une courte description présentant le test | Connaitre l’élément testé |
| Les données utilisées en temps qu’entré pour le test | Savoir quelle valeur sont testés et par substitution, lesquels ne le sont pas |
| Les conditions à succès | Connaitre quelle et comment est la sortie du test |
| La durée approximative du test | Connaitre le temps nécessaire à son exécution normale. |

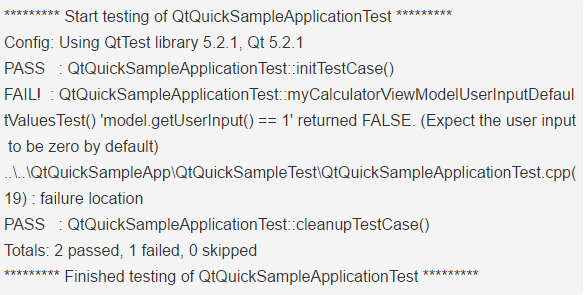
## Rapport d’exécution de tests

Les rapports d’exécution de tests apparaissent dans une console lorsqu’on lance le test. Néanmoins, la librairie QTestLib fournit la possibilité d’exporter ces résultats au format XML.



Sortie console d’un test avec succès

En cas d’erreur, la fonction ayant échoué est mentionné par un FAIL ! et un message correspondant à l’erreur est indiqué ainsi que le fichier et la ligne à laquelle l’erreur s’est produit.



## 

Sortie console d’un test avec échec

## Rapport sur le suivi global de la qualité

# Types de tests

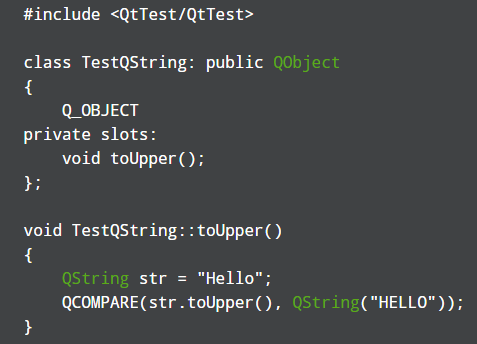
## Tests unitaires

### Objectif

Il s'agit pour le programmeur de tester une fonction, indépendamment du reste du programme, ceci afin de s'assurer qu'elle répond aux [spécifications fonctionnelles](https://fr.wikipedia.org/wiki/Sp%C3%A9cification_(informatique)) et qu'elle fonctionne correctement en toutes circonstances.

### Environnement et conditions de réalisation

Les tests unitaires peuvent être joué à n’importe quel moment de développement et cela sur toute nouvelle fonction implémentée au sein du projet. Ces tests seront réalisé à l’aide de QT (sous version Windows et MAC) qui possède une librairie spécialisée pour les tests, QTestLib.

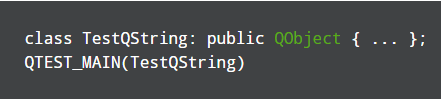


Exemple de la classe de test

Lorsqu’on souhaite réaliser un test, il faut écrire une nouvelle classe, qui hérite de QObject. Chaque private slot représente une fonction qu’on souhaite tester. Il suffit ensuite d’implémenter cette fonction et de réaliser les tests à l’aide des macros de la librairie QTestLib.

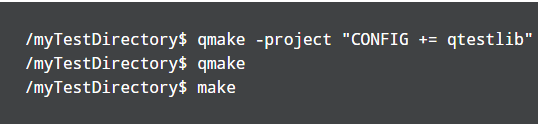
### Configurations particulières

Comme vu précédemment, lorsqu’on souhaite réaliser un test, il faut écrire une classe qui hérite de QObject. Une fois notre classe créée et les fonctions qu’on souhaite tester sont implémentées, il suffit d’ajouter à la fin du fichier QTEST\_MAIN(NomDeLaClasseTest). Cela va créer un exécutable lors de la compilation, qui une fois lancé va exécuter les tests dans l’ordre dans lequel sont définis les privates slots.



Utilisation de QTEST\_MAIN

Pour la compilation, if faut indiquer à qmake qu’on utilise la librairie QTestLib.

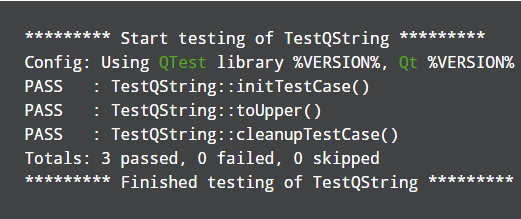


Compilation d’un test unitaire

### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacrée au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

### Critères de passage/échec



Sortie d’exécution du programme de test

Les critères de passage/échec seront définis via les tests effectués. Si une erreur est détectée, celle-ci est retournée et le test s’arrête.

initTestCase() et cleanupTestCase() sont automatiquement appelé, le premier permet d’initialiser la fonction qui va être testé et la seconde permet de libérer la place pour la prochaine fonction à test.

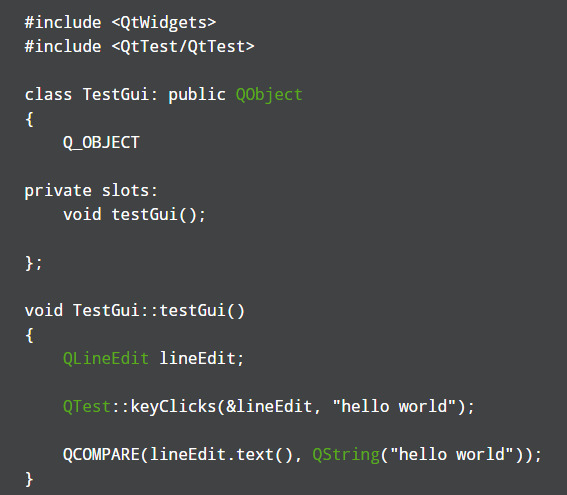
## Tests logiciel UI

### Objectif

Ces tests ont pour but de vérifier le bon déroulement des interactions entre l’utilisateur et l’interface utilisateur du logiciel.

### Environnement et conditions de réalisation

Ces tests interviennent dès qu’on ajoute un élément, tel qu’un bouton ou un champ texte, à l’interface utilisateur. Comme pour les tests unitaires, on se sert la librairie QTestLib qui permet notamment de simuler les actions utilisateurs.



Exemple de test sur un champ texte

Dans la fonction testGui, on simule le clic sur un champ texte dans lequel on écrit ‘’hello world’’ et on vérifie ensuite que le contenu de ce champ texte est bien ‘’hello world’’.

### Configurations particulières

Voir partie 5.1.3

### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacrée au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

### Critère de démarrage des tests

???????????????????????????????????????????????????

### Critères de passage/échec

Voir partie 5.1.6

## Tests Fonctionnels

### Objectif

C’est un ensemble de tests unitaires ayant pour but de valider le bon déroulement d’une fonctionnalité du logiciel, comme par exemple la création d’un projet ou l’édition d’une partition.

### Environnement et conditions de réalisation

????????????????????????????????????????

### Configurations particulières

????????????????????????????????????????

### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacrée au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

### Critère de démarrage des tests

????????????????????????????????????????

### Critères de passage/échec

Pour que le test soit considéré comme un succès, l’ensemble des tests unitaires qui compose le test de fonctionnalité doivent réussir.

## Tests de régression

### Objectif

Ces tests ont pour objectif de vérifier que les modifications apportées au projet, n’ont pas entraîné d’effets de bord non prévus qui pourraient dégrader le comportement du logiciel antérieurement validé.

### Environnement et conditions de réalisation

???????????????????????????????????????????

### Configurations particulières

Il faut écrire ces tests en réutilisant les tests déjà joués et qui ont réussi afin de s’assurer que le système répond toujours aux exigences spécifiées.

### Planning et charge

Ces tests sont pris en compte, sur le Gantt, dans la période de temps consacrée au développement d’une fonctionnalité du logiciel. La charge de travail dépend de ce qui est testé.

### Critère de démarrage des tests

??????????????????????????????????????????????

### Critères de passage/échec

Pour que le test soit considéré comme un succès, if faut que malgré l’ajout ou la modification de code, ce dernier répond toujours aux exigences spécifiées.

# Estimation Globales de la charge de travail

La charge de travail a été évaluée au travers du document du Gantt. Des périodes plus ou moins longues ont été attribué aux divers tâches, ces périodes prennent en compte les tests qui doivent être réalisés.

# Outils

## Dépôt de tests

## Gestion de rejets/erreurs

## Autres outils

### Outils d’automatisation des tests

### Outils pour les tests de performance