

UNIVERSITÉ NICE - SOPHIA ANTIPOLIS

*IUT DE NICE - COTE D'AZUR*

*DÉPARTEMENT INFORMATIQUE*

*41 bd. Napoléon III - 06200 NICE*

RAPPORT DE STAGE POUR L'OBTENTION DU DIPLOME  
UNIVERSITAIRE DE TECHNOLOGIE

Session 2019-2020

Certification de la qualité d'un projet par des  
diagrammes de justification lors de l'intégration  
continue.

Présenté par Nicolas Corbière



Sous la direction de : Madame Mireille Blay-Fornarino

## Remerciements

Je tiens à remercier Madame Blay de m'avoir choisie pour ce sujet. Elle m'a conseillé et aidé lors de mon travail. J'ai aussi pris beaucoup de plaisir à faire ce projet et j'y ai appris énormément de choses intéressantes.

Je remercie aussi l'IUT d'avoir mis en place l'alternative des projets individuels, cela m'a énormément soulagé, et ce fut très enrichissant.

## Résumé

Afin de pouvoir valider mon DUT, j'ai effectué un Projet Individuel (PI). Effectivement, en conséquence de la pandémie, les entreprises ont des difficultés à accepter des stagiaires et le département informatique de l'IUT a mis en place l'alternative des PI.

Ce document présente le travail que j'ai effectué pendant 2 mois sur un projet exploratoire à propos des diagrammes de justification.

Les diagrammes de justification sont des représentations graphiques permettant de justifier voire certifier du processus de construction d'un système ou d'un produit en traçant les preuves produites pour en attester. Dans le cadre de ce projet, nous étudions son utilisation dans un contexte d'Intégration Continue pour justifier de la qualité d'une version du logiciel.

Ces diagrammes correspondent à des arbres dont les feuilles sont les preuves produites, et la racine est la conclusion. Les nœuds intermédiaires sont soit des conclusions partielles soit des stratégies.

Il existait un projet Java permettant de générer ces diagrammes à partir d'une description textuelle et des fichiers « todo » correspondant aux preuves à fournir. Le projet qui m'a été confié avait deux objectifs. Le premier était de pouvoir utiliser ce projet dans un contexte d'intégration continue, et le second était d'ajouter une notion d'état à chaque nœud afin de pouvoir visualiser l'avancement du projet.

Le projet lui-même est en intégration continue et chaque étape est justifiée en utilisant les diagrammes de justification. Pour le second objectif, j'ai permis (1) de tracer l'évolution du projet (2) d'analyser ces traces pour valider la production des preuves (3) de visualiser différemment un diagramme de justification en fonction de son état (4) d'enrichir l'étape de validation par des actions telles que vérifier l'existence de certains fichiers, s'assurer de la qualité de la couverture de tests. La mise en œuvre des actions permet de facilement en ajouter de nouvelles.

Enfin, j'ai beaucoup travaillé sur la documentation, notamment le README du projet, afin de permettre à d'autres de justifier de leur intégration continue.

## Summary

In order to be able to validate my OTC, I completed an Individual Project (IP). Indeed, because of the pandemic, companies have difficulties to accept trainees and the IT department of the IUT has set up the alternative of the IP.

This document presents the work I did during 2 months on an exploratory project about justification diagrams.

Justification diagrams are graphical representations allowing to justify or even certify the construction process of a system or a product by tracing the evidence produced to prove it. In the context of this project, we are studying its use in a Continuous Integration context to justify the quality of a software version.

These diagrams correspond to trees whose leaves are the evidence produced, and the root is the conclusion. The intermediate nodes are either partial conclusions or strategies.

There was a Java project to generate these diagrams from a textual description and "todo" files corresponding to the proofs to be provided. The project I was assigned had two objectives. The first was to be able to use this project in a context of continuous integration,

and the second was to add a notion of state at each node in order to be able to visualize the progress of the project.

The project itself is in continuous integration and each step is justified using the justification diagrams. For the second objective, I allowed (1) to trace the evolution of the project, (2) to analyze these traces to validate the production of evidence, (3) to visualize differently a justification diagram according to its state, (4) to enrich the validation step by actions such as checking the existence of certain files, ensuring the quality of the test coverage. The implementation of the actions makes it easy to add new ones.

Finally, I worked a lot on the documentation, especially the project's README, in order to allow others to be able to justify their continuous integration.

## Table des matières

Remerciements .....	2
Résumé .....	2
Summary .....	3
Introduction .....	5
Développement.....	5
Présentation de diagramme.....	6
Gestion d'état.....	8
Fichier de réalisation.....	8
Fichier d'action .....	10
Intégration continue .....	18
Build, test et archivage.....	18
Création de mon diagramme .....	20
SonarCloud .....	21
Résultats .....	22
Conclusion.....	25
Glossaire.....	26
Bibliographie .....	27

## Introduction

Ce projet est issu de la thèse de Clément DUFFAU. Dans celle-ci, il explique que pour tout projet ayant des risques, il est nécessaire de fournir une documentation qui certifie le bon fonctionnement d'un système ou d'un produit.

Cependant, dans la plupart des cas, on se retrouve avec énormément de documents.

Il a donc voulu créer une nouvelle forme de documentation graphique permettant de vérifier tout le cycle de vie du projet, tout en étant lisible et digne de confiance.

Cette documentation prend la forme d'un diagramme de justification qui est composé de plusieurs nœuds représentant chacun une étape du projet.

Un framework pour la construction de ces diagrammes a été créé et reposait sur une architecture complexe à bases de services.

Plus tard, Madame Corinne Pulgar, qui était une étudiante, à créer un nouveau projet java plus simple qui génère des diagrammes et des « todo list » à partir d'un fichier « .jd ». Elle était sous la supervision de Sébastien Mosser de l'UQAC qui est au CANADA.

Madame Blay a récupéré ce projet pour les projets individuels et m'a confié deux objectifs. Le premier était d'ajouter ce projet à un contexte d'intégration continue, et le second était d'ajouter une notion « d'État » pour chaque nœud, permettant de voir l'avancement d'un projet.

Ce projet est un projet exploratoire, il demandait des compétences dans l'intégration continue et avoir la capacité à fournir du code de qualité, pouvant facilement évoluer.

## Développement

Dans cette partie, je présente dans un premier temps, les diagrammes, puis j'explique comment j'ai géré la notion d'état pour chaque nœud et je termine par l'intégration continue.

## Présentation de diagramme

Un diagramme de justification est un graphe orienté acyclique avec une unique racine. Les feuilles correspondent aux preuves fournies (*e.g.* fichiers de tests, document), la racine est la conclusion (*e.g.* le logiciel est bien construit), les nœuds intermédiaires sont des stratégies (*e.g.* vérifier la couverture de test) ou des conclusions intermédiaires (*e.g.* La couverture de tests correspond aux exigences).

Pour pouvoir créer un diagramme, Madame Corinne Pulgar a créé son propre langage. Dans un fichier « jd » l'utilisateur définit la structure du diagramme en utilisant ce langage. À partir de celui-ci il est possible de générer une « todo liste ».

Dans un fichier « jd », vous déclarez les différents nœuds en leur donnant un type, un alias et un label. L'alias est ensuite utilisé pour pouvoir lier les différents nœuds entre eux et le label est le texte qui sera affiché dans votre diagramme et votre liste.

Ces diagrammes de justification sont composés de plusieurs nœuds qui représentent des étapes, chaque nœud a un label et il existe plusieurs types de nœuds qui reflètent la nature de l'étape. Les nœuds sont liés les uns aux autres, et les nœuds du bas sont les preuves qui justifient les conclusions du haut.

Vous pouvez voir dans la figure 1, ci-dessous un exemple très simple d'un fichier « jd », de la liste de « TODO » et de son diagramme qui sont générés. Nous pouvons voir dans le diagramme à gauche qu'il y a 4 nœuds avec des formes et couleurs différentes. « JUnit test logs » et « Jenkins test logs » sont tous deux des « supports », ils représentent des preuves. Ensuite, nous avons une étape « Software is fonctionnal », celle-ci est une « stratégie », et elle représente une action. Tout en haut, nous avons « Software is ready for launch » qui est la conclusion du diagramme.

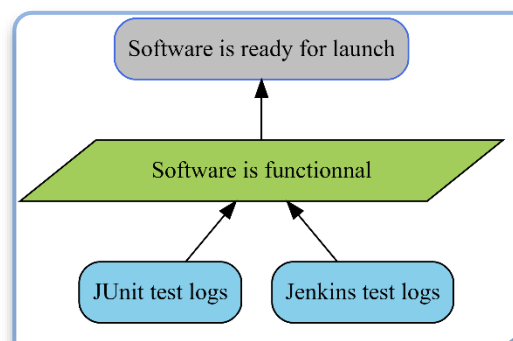
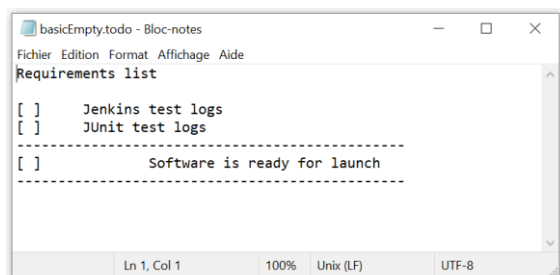
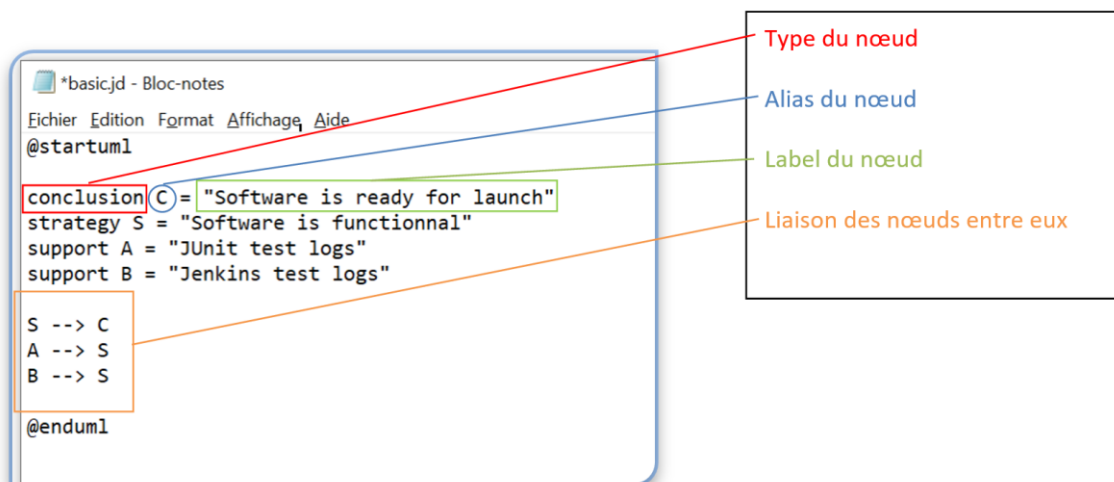


Figure 1: Du langage à la représentation graphique et à la todo liste.

Nous pouvons lire ce diagramme de la manière suivante, « pour que mon logiciel soit prêt à être lancé, il doit fonctionner. Son fonctionnement dépend alors des logs de tests JUnit et Jenkins ».

Lors de mon projet, j'ai créé un diagramme (cf. Figure 2 à la page suivante) qui représente mon projet. Vu que je vais ensuite créer un diagramme en fonction des états, je vais appeler ce diagramme 'mon patron'.

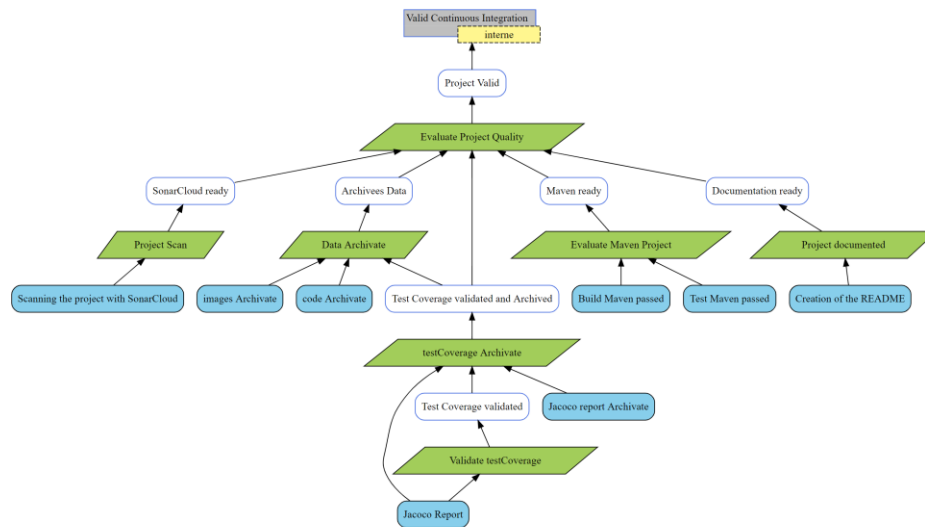


Figure 2 : Patron de justification pour le projet lui-même

Nous pouvons voir que mon projet comporte la création d'un rapport jacoco et son archivage, le build de mon projet et le teste de celui-ci, un scan de mon projet avec SonarCloud, de l'archivage de données et enfin, de la documentation.

## Gestion d'état

### Fichier de réalisation

À partir du patron, il s'agit à présent de tracer une intégration continue et relativement au patron de visualiser ce qui a été fait et ce qu'il reste à faire.

La gestion des états était l'un des objectifs du projet.

Soit un nœud a des enfants, et donc son état dépend de ces derniers, soit un nœud n'a pas d'enfants, et c'est donc une feuille. Les feuilles correspondent aux preuves à fournir

Tracer la réalisation d'un système consiste alors à écrire dans un fichier tous les labels des feuilles qui sont « réalisées ». Il est alors très simple de vérifier ce qui a été fait et ce qui reste à faire ; si le label est contenu dans mon fichier de réalisation, alors il est « fait », sinon il est « à faire ».

Pour construire le diagramme de justification du réalisé, j'utilise le fichier de réalisation comme suit.

Jattribue un état à chacune des feuilles qui sont soit « DONE » soit « TODO ».



Par la suite, j'attribue l'état aux nœuds père : si tous ses nœuds fils sont « faits » alors il est aussi dans l'état « DONE », sinon il est dans l'état « TODO ».

Ce fichier de réalisation me permet donc de créer une « todo liste » et un diagramme, qui est l'instance de mon patron visualisant l'avancée de mon projet.

Pour l'instance de mon patron, chaque nœud qui est « fait » aura un contour et une couleur de texte en bleu, sinon, ce sera en rouge. Quant à ma todo list, les nœuds qui seront faits seront cochés, sinon ils ne le seront pas.

Pour mon projet, mon fichier de réalisation ressemble à cela (cf. Figure 3).

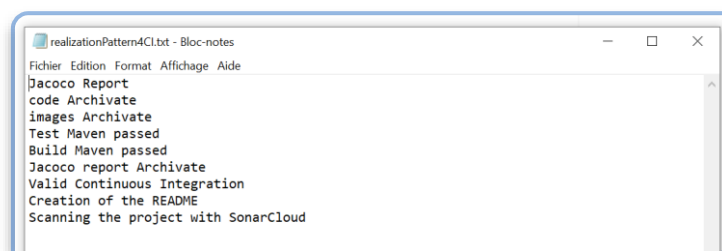


Figure 3 : Fichier de réalisation correspondant au projet

Dans ce cas, je vais me retrouver avec ce diagramme et cette liste (cf. Figure 4) :

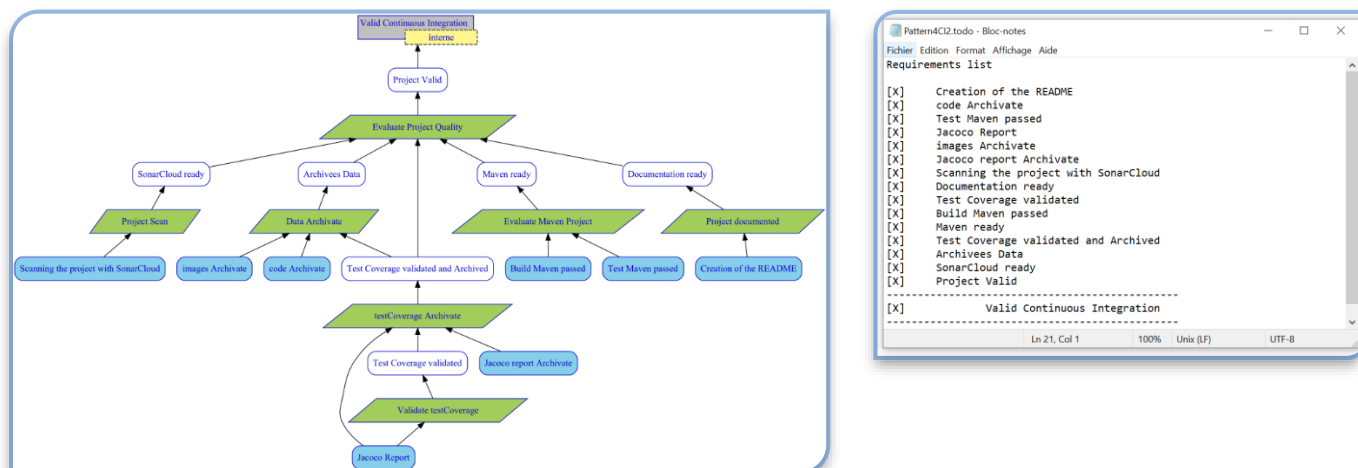


Figure 4 : Diagramme de justification du réalisé et todo list : Tout a été réalisé.

Cependant, si dans mon fichier de réalisation, il n'y a pas le label « Test Maven passed », je vais obtenir ces résultats (cf. Figure 5) :

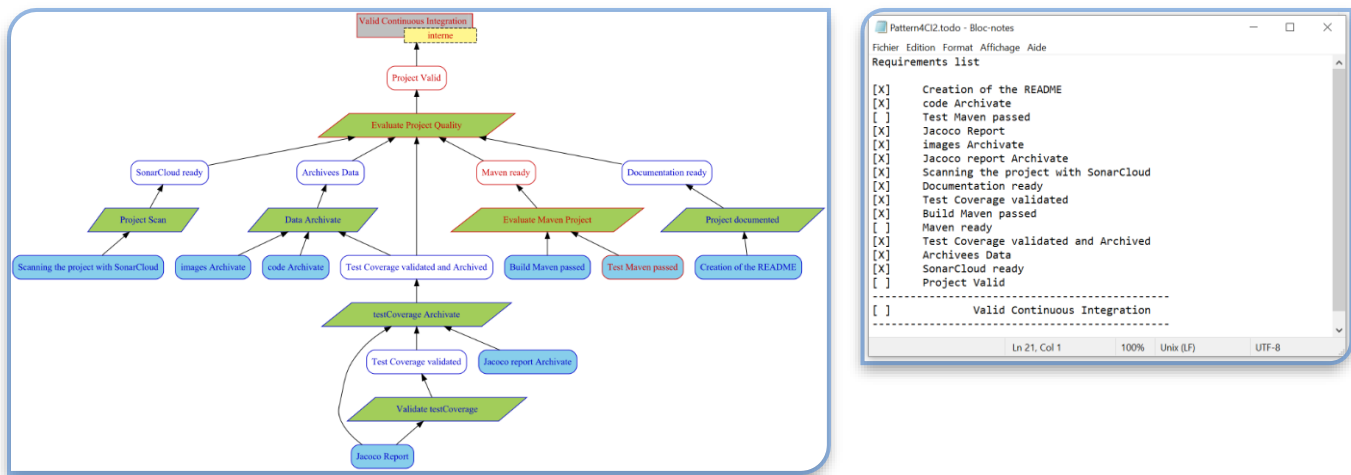


Figure 5 : Diagramme de justification et todo list après échec de justification

Nous pouvons voir que « Test Maven passed » est en rouge, ça veut dire qu'il n'a pas été fait et son état se propage à ses nœuds père (Evaluate Maven Project, Maven ready, etc....).

De plus, dans ma « todo list » ces mêmes étapes ne sont pas cochées.

## Fichier d'action

Nous avons fait le choix d'enrichir ce processus en permettant à l'utilisateur de préciser quelles sont les actions à réaliser pour considérer qu'une « justification » est valide. Pour cela, il peut associer à chaque nœud du patron les actions à réaliser sous la forme d'un fichier « json » qui contiendra les différentes actions et compléments d'information.

Vous pouvez voir un exemple ci-dessous que je détaille au fur et à mesure (cf. Figure 6 à la page suivante) :

```

1 ▼ [
2 ▼   {
3 ▼     "Node":{
4       "Label":"Jacoco Report",
5       "Files": [
6         "target/site/jacoco/index.html"
7       ]
8     },
9   },
10 ▼  {
11 ▼    "Node": {
12      "Label":"images Archivate",
13      "FilesNumber": [
14        {
15          "Path":"justification/output/images/",
16          "Number":"13"
17        },
18        {
19          "Path":"justification/",
20          "Number":"2"
21        },
22      ],
23    },
24  },
25 },
26 ],
27 },
28 ▼ {
29 ▼   "Node":{
30     "Label":"Documentation ready",
31     "Optional":"true",
32   },
33 },
34 ▼ {
35 ▼   "Node":{
36     "Label":"code Archivate",
37     "Reference":"generatedCode",
38   },
39 },
40 ]

```

Figure 6 : Aperçu d'un fichier d'action

### Vérification de fichier

Nous avons choisi d'ajouter la vérification de l'existence d'un fichier pour valider un nœud. Par exemple, on peut dire que le nœud « Jacoco Report » est valide uniquement si le rapport est généré, c'est-à-dire si le répertoire « target/site/jacoco » existe.

Pour ce faire, dans mon fichier « json », je vais indiquer le label « Jacoco Report » et dans une liste de « Files » je vais indiquer tous les fichiers que je veux vérifier pour pouvoir valider cette réalisation, et dans cet exemple, il s'agira uniquement de « target/site/jacoco ».

Vous pouvez voir la syntaxe à la page suivante (cf. Figure 7) :

```

1 ▼ [
2 ▼   {
3 ▼     "Node":{
4       "Label":"Jacoco Report",
5       "Files": [
6         "target/site/jacoco/"
7       ]
8     },
9   },

```

Figure 7 : Fichier d'action, vérification de l'existence d'un fichier

En fonction du résultat de ma vérification, je vais avoir une incidence sur mon diagramme et ma « todo list ».

Si je trouve ce fichier, je vais avoir le résultat suivant (cf. Figure 8) :

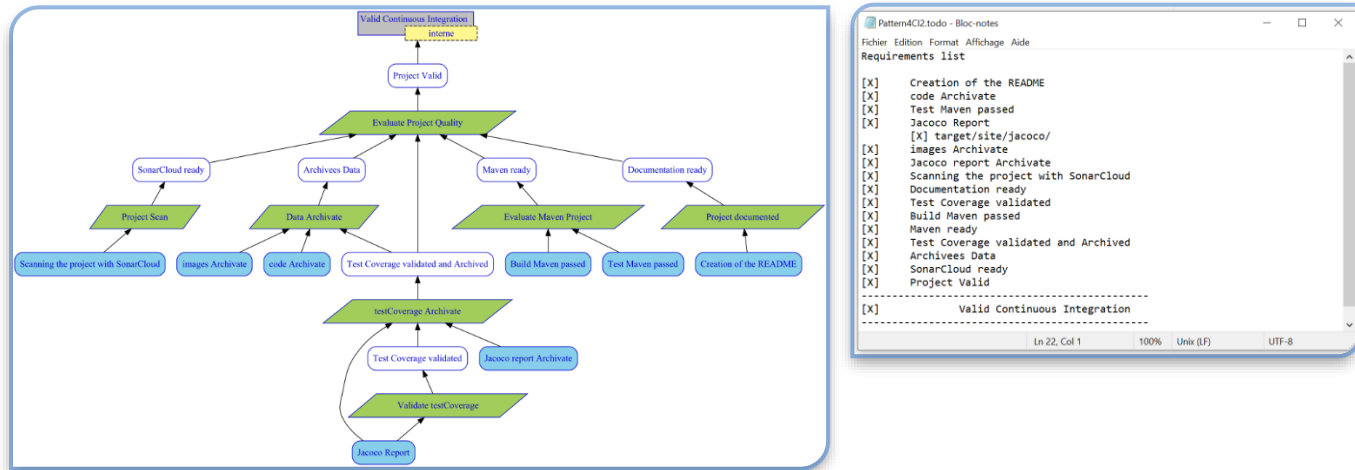


Figure 8 : Diagramme de justification et todo list après une vérification réussie de l'existence d'un fichier

Vous pouvez noter que dans ma liste à droite, en dessous de « Jacoco Report », j'ai une case « target/site/jacoco » qui s'est ajoutée. Celle-ci sera cochée si le fichier existe, sinon elle ne le sera pas et aura un message d'erreur tout à droite « (not found) ».

Si je ne trouve pas ce fichier, je vais me retrouver avec ce résultat (cf. Figure 9) :

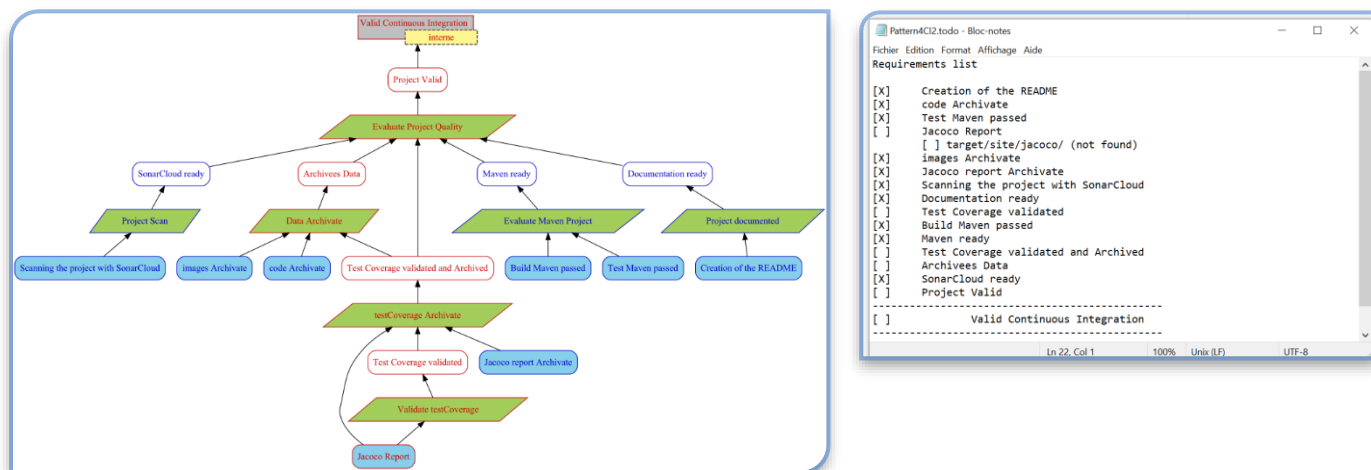


Figure 9 : Diagramme de justification et todo list après l'échec d'une vérification de l'existence d'un fichier

Cependant, pour pouvoir valider mes tests unitaires, je devais vérifier l'existence de 13 fichiers. Je devais donc indiquer tous les noms des fichiers. C'est à ce moment où j'ai eu l'idée du point suivant.

#### *Vérification que le fichier contient un certain nombre de fichiers*

Pour mes tests unitaires, j'ai ajouté une nouvelle option qui est de vérifier qu'un certain répertoire contient un certain nombre de fichiers. Pour ce faire, j'utilise un champ « FilesNumber » qui va contenir une liste d'objets. Ces derniers vont avoir pour champ un répertoire et le nombre qui lui est associé.

De cette manière, je peux vérifier le nombre de fichiers de plusieurs répertoires pour pouvoir valider un nœud. Reprenons mon exemple, mes tests unitaires créés 13 fichiers dans mon répertoire « justification/output/image » et je veux vérifier que c'est bien le cas.

Je vais donc écrire ceci (cf. Figure 10) :

```
10 {
11   "Node": {
12     "Label": "Test Maven Passed",
13     "FilesNumber": [
14       {
15         "Path": "justification/output/images/",
16         "Number": "13"
17       },
18       {
19         "Path": "justification/",
20         "Number": "2"
21       },
22     ],
23   },
24 },
25
26
27 }
```

*Figure 10 : fichier d'action, vérification que des répertoires contient un certain nombre de fichiers*

Vous pouvez voir que je vérifie aussi que le répertoire « justification » contient 2 fichiers.  
Si la vérification est juste, je vais me retrouver avec ce résultat (cf. Figure 11) :

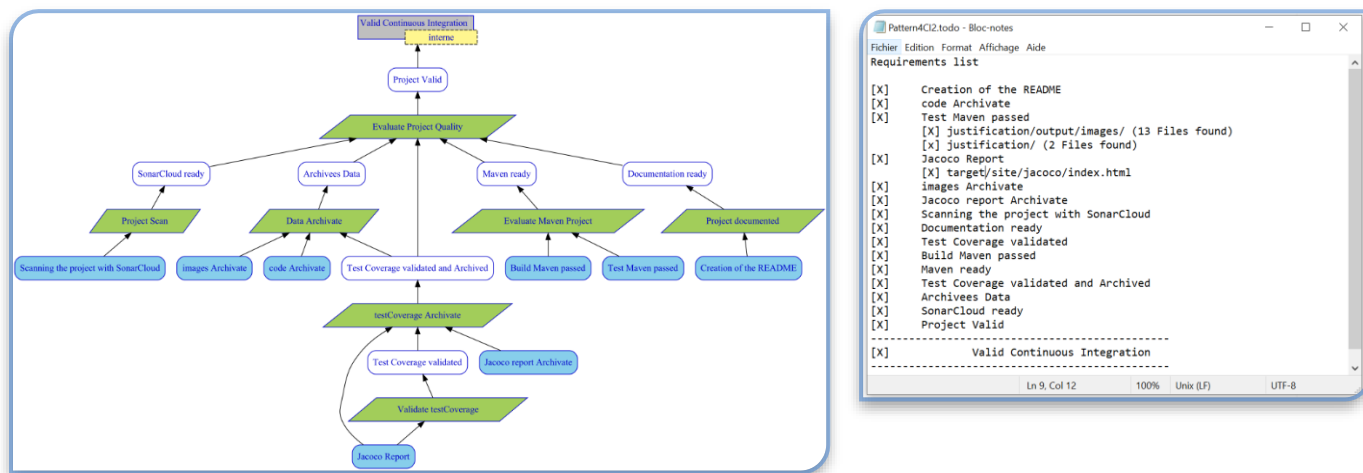


Figure 11 : Diagramme de justification du réalisé et todo list les répertoires sont valides

Nous pouvons voir dans mon fichier « TODO » que j'ai le répertoire  
« justification/output/images » ainsi que le nombre de fichiers que j'ai trouvé.

Cependant, si j'indique que je veux trouver 12 fichiers et que j'en trouvais 13, je vais me retrouver avec ce résultat (cf. Figure 12)

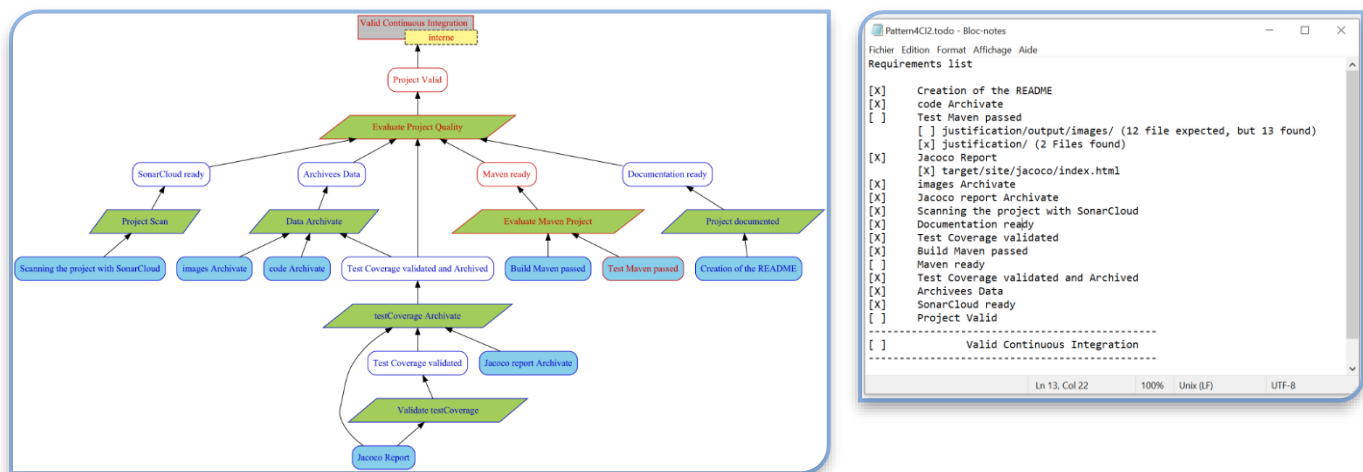


Figure 12 : Diagramme de justification du réalisé et todo list les répertoires ne sont pas valides

Dans mon diagramme, « Test Maven Passed » ne sera pas valide, et sera donc en « TODO ». Son état va se propager à ses pères et dans mon fichier « TODO », je vais avoir un message comme quoi 12 fichiers étaient attendus, mais que j'en ai trouvé 13.

### *Ajout de référence et passage d'un nœud en facultatif*

Par la suite, j'ai voulu inclure des références pour chaque nœud en plus de pouvoir indiquer qu'un nœud est facultatif.

Si un nœud est facultatif, son état ne se propagera pas à ses nœuds père. En effet, ça serait dommage que tout mon projet soit considéré comme étant non fonctionnel à cause de ma documentation, qui elle, ne serait pas faite par exemple.

Pour ce faire, dans mon fichier « json », j'ai ajouté un champ « Optional », et s'il a pour valeur « true », le nœud sera optionnel. J'ai aussi ajouté le champ « Référence » pour pouvoir ajouter des références à mes nœuds qui seront affichés dans la « todo list ».

Voici un exemple où je mets « Documentation ready » en optionnel et j'ajoute la référence « generated code » au nœud « code Archive » (cf. Figure 13) :

```
28 ▼ {
29 ▼   "Node":{
30     "Label":"Documentation ready",
31     "Optional":"true",
32   }
33 },
34 ▼ {
35 ▼   "Node":{
36     "Label":"code Archive",
37     "Reference":"generatedCode",
38   }
39 }
40 ]
```

*Figure 13 : fichier d'action, ajout de référence et d'optionnalité*

Je vais obtenir le résultat suivant si mon nœud « Documentation ready » est « à faire » (cf. Figure 14) :

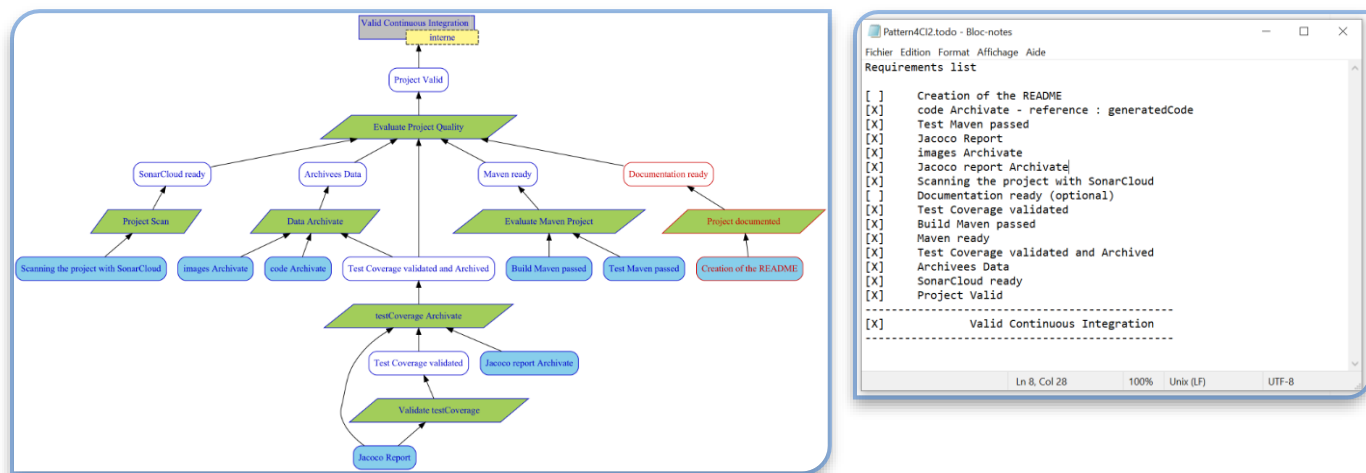


Figure 14 : Diagramme de justification du réalisé et todo list, avec une documentation optionnelle.

Nous pouvons voir que « Evaluate Project Quality » n'est pas touché par l'état de la documentation, et dans ma « todo list », je vais avoir une indication que le nœud « Documentation ready » est optionnel.

De plus, nous pouvons voir la référence de « code Archive ».

### Action de vérification

Par la suite, nous avons eu pour idée d'ajouter un champ qui contiendra une liste de command. Ces Command sont ensuite traités par le design pattern « Command » et permettront de faire une suite d'actions. Nous nous en servons pour ajouter de nouvelles vérifications et actuellement il n'y en a qu'une seule qui me permet de vérifier que mon test de couverture est bien égal, supérieur ou inférieur à un certain seuil à partir de mon rapport jacoco.

Dans mon projet, pour pouvoir valider mon rapport jacoco, je veux vérifier qu'il a été généré et que le test de couverture soit supérieur soit égal à 70. De plus, je dois indiquer soit le fichier « jacoco.csv » soit l'index.html » de mon rapport jacoco.



Toutes ces informations doivent être indiquées comme arguments de ma ligne de command « CheckCover ». Voici un exemple (cf. Figure 15) :

```

1 [
2 {
3   "Node":{
4     "Label":"Jacoco Report",
5     "Files": [
6       "target/site/jacoco/index.html"
7     ],
8     "Actions": [
9       "CheckCoverage target/site/jacoco/index.html >= 70",
10    ],
11  },
12 },

```

Figure 15 : Fichier d'action, action de vérification

Si tout se passe bien, je vais me retrouver avec le résultat ci-dessous (cf. Figure 16) :

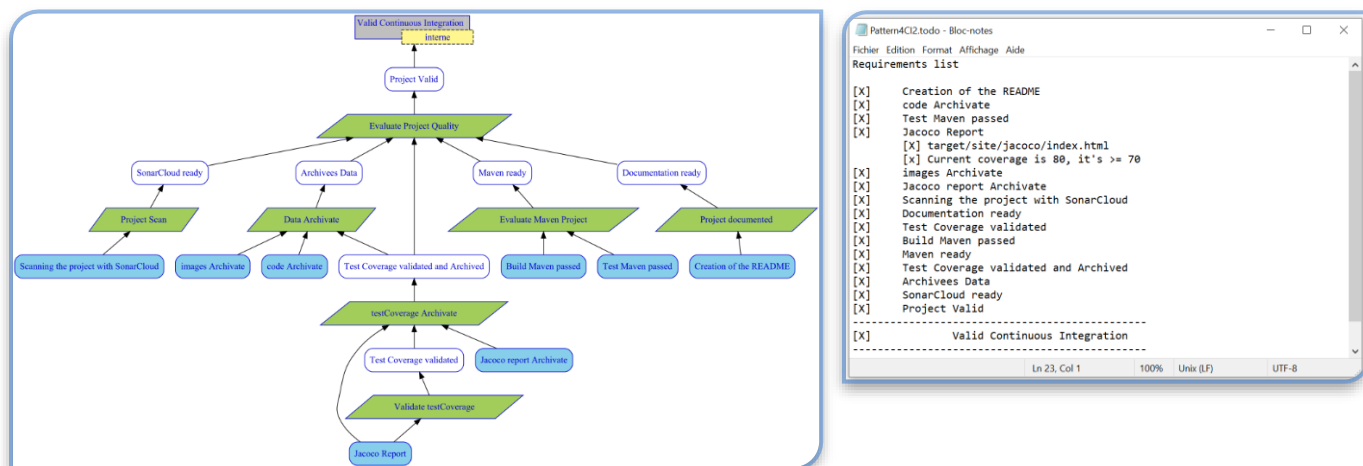


Figure 16 : Diagramme de justification du réalisé et todo list, lorsque le test de couverture est valide.

Dans ma « todo list », je vais voir le pourcentage de test actuel ainsi que le seuil qui était exigé. Cependant, si j'avais demandé à avoir plus de 90% de couverture, je n'aurais pas validé mon nœud « Jacoco Report » et j'aurais eu pour message « Current coverage is 80, it's not >= 90 ».

## Intégration continue

Mon deuxième objectif était d'intégrer ce projet à un contexte d'intégration continue.

Pour ce faire, j'ai travaillé avec un fichier « .yaml » et j'ai pu visualiser les résultats dans l'onglet « action » de github.

J'ai décomposé ce fichier en plusieurs parties pour pouvoir mieux l'expliquer.

### Build, test et archivage

Dans un premier temps, j'indique la branche dans laquelle je travaille, ici, il s'agira de master. Je demande ensuite à créer une machine virtuelle « ubuntu » pour pouvoir exécuter les intégrations continues.

Je termine en indiquant que je travaille avec le JDK 1.8 (cf. Figure 17).

```
4  name: Java CI with Maven
5
6  on:
7    push:
8      branches: [ master ]
9    pull_request:
10     branches: [ master ]
11
12  jobs:
13    build:
14
15     runs-on: ubuntu-latest
16
17     steps:
18     - uses: actions/checkout@v2
19     - name: Set up JDK 1.8
20       uses: actions/setup-java@v1
21       with:
22         java-version: 1.8
```

Figure 17 : Un début de test d'intégration

Après avoir installé l'environnement, je build et je teste mon projet. De plus, pour pouvoir faire créer l'instance de mon patron, je vais incrémenter petit à petit mon fichier de réalisation. Je vais indiquer les labels qui correspondent aux étapes effectuer. Ici, étant donné que je viens de faire le build, je vais écrire « Build Maven passed » qui correspond à l'un des nœuds de mon diagramme.

De même pour ma documentation et pour les tests (cf. Figure 18 à la page suivante) :

```

23 #Build of the project, 'Creation of the README' is done
24 - name: Creation of the realization file
25   run:
26     echo -e "\nCreation of the README" >> realization.txt;
27 #Build of the project, 'Build Maven passed' is done
28 - name:
29   Build with Maven;
30   run:
31     mvn -B package --file pom.xml;
32     echo -e "Build Maven passed" >> realization.txt;
33 #Test of the project, 'Test Maven passed' is done
34 - name: Test with Maven
35   run:
36     mvn test;
37     echo -e "Test Maven passed" >> realization.txt;

```

Figure 18 : Build et Test de mon projet lors de l'intégration continue

Grâce au build et au test, j'ai généré mon rapport jacoco et les images de mes tests unitaires.

Je vais archiver ces derniers, en plus de mon code source, dans différents artefacts grâce aux lignes de la page suivante (cf. Figure 19).

```

43 #I archive the diagrams generated during the test
44 - name: Archive images
45   uses: actions/upload-artifact@v2
46   with:
47     name: images
48     path: justification/output/images
49
50 #I archive the Jacoco report
51 - name: Archive Jacoco report
52   uses: actions/upload-artifact@v2
53   with:
54     name: jacoco
55     path: target/site/jacoco
56
57 #I archive the generated codes
58 - name: Archive generated codes
59   uses: actions/upload-artifact@v2
60   with:
61     name: generatedCode
62     path: src/main/java/models
63
64

```

Figure 19 : Archivage des données issues du Build et Test

L'artefact « images » contiendra tous les éléments contenus dans le répertoire « justification/output/images », l'artefact jacoco contiendra le rapport jacoco et « generatedCode » archivera les codes source.

Dans mon onglet « Action » de mon projet github, je vais alors me retrouver avec ces artefacts (cf. Figure 20) :







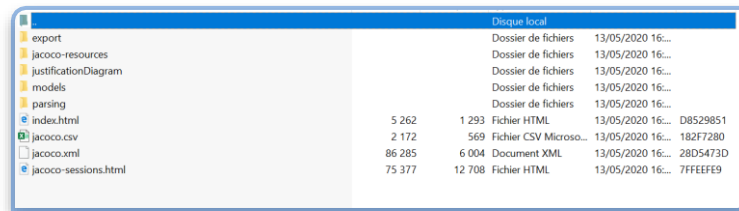
Artifacts		
 jacoco	511 KB	
 images	216 KB	
 generatedCode	4.23 KB	

Figure 20 : Artéfact des données issues du Build et Test

Prenons par exemple l'artéfact « jacoco », qui a archivé le rapport suivant (cf. Figure 21) :



nom	taille	type	date	hash
Dossier de fichiers			13/05/2020 16:00	
Dossier de fichiers			13/05/2020 16:00	
Dossier de fichiers			13/05/2020 16:00	
Dossier de fichiers			13/05/2020 16:00	
Dossier de fichiers			13/05/2020 16:00	
Fichier HTML	5 262		13/05/2020 16:00	D8529851
Fichier CSV Microsoft	2 172		13/05/2020 16:00	182F7280
Document XML	86 285		13/05/2020 16:00	28D5473D
Fichier HTML	75 377		13/05/2020 16:00	7FFEEFE9

Figure 21 : Archive des données issues du Build et Test

Pour mieux appréhender le résultat, vous pouvez aller voir sur ce lien le résultat :

<https://github.com/MireilleBF/JustificationDiagram/actions/runs/103791524>

### Création de mon diagramme

Ensuite, pour créer mon diagramme, je dois continuer à incrémenter mon fichier de réalisation, puis j'exécute mon projet. Au lieu d'exécuter celui-ci, je pourrais aussi utiliser la jar de mon projet, mais le principe reste le même.

Je dois alors indiquer comme paramètre les points suivants :

- Mon fichier « .jd »
- Mon répertoire de sortie ainsi que le nom que je veux lui donner (-o)
- Mon fichier de réalisation (-rea) et mon fichier d'action (-info)
- J'indique que je veux générer mon patron (-svg), l'instance de mon patron (-svgR) et ma « todo list » (-td).

Voici ce que j'ai écrit (cf. Figure 22) :

```

69
70 #-----JustificationDiagram-----
71
72 #!Jacoco Report' and 'Jacoco report Archive' are done
73 - name: Realization part1
74   run: echo -e "Jacoco Report\nJacoco report Archive" >> realization.txt
75
76 #!images Archive' and 'code Archive' are done
77 - name: Realization part2
78   run:
79     echo -e "images Archive" >> realization.txt;
80     echo -e "code Archive" >> realization.txt;
81
82 #I generate the two diagrams and the TODO list
83 - name: JD&TODO generation
84   run: mvn exec:java -Dexec.mainClass="JDCompiler" -Dexec.args="justification/examples/exampleCI/Pattern4CI.jd -o
      justification/output/GeneratedJD/Pattern4CI -rea realization.txt -act justification/examples/exampleCI/infoValid.json -svg -svgR
      -td "

```

Figure 22 : Ajout de label dans le fichier de réalisation et génération de mes diagrammes

Par la suite, je vais archiver le fichier de réalisation, d'action, ma « todo list » et mes diagrammes générés dans l'artefact « GeneratedJD » (cf. Figure 23).

```
81 #I archive my diagrams create during the CI
82 - name: Archive JD&TODO
83   uses: actions/upload-artifact@v2
84   with:
85     name: GeneratedJD
86     path: justification/output/GeneratedJD/
87
88 #I archive my realization file in the same artifacts as my diagrams
89 - name: Archive realization
90   uses: actions/upload-artifact@v2
91   with:
92     name: GeneratedJD
93     path: realization.txt
94
95 #I archive my information file in the same artifacts as my diagrams
96 - name: Archive information
97   uses: actions/upload-artifact@v2
98   with:
99     name: GeneratedJD
100    path: justification/examples/exampleCI/infoValid.json
101
```

Figure 23 : Archivage de mes données généraies

## SonarCloud

Enfin, on m'a demandé d'ajouter un scan du projet grâce à SonarCloud, ce qui permet d'obtenir un rapport sur la qualité du code du projet.

Pour ce faire, j'ai ajouté la ligne ci-dessous dans le « yml », avant la génération du diagramme (cf. Figure 24).

```
35 #Scanning the project with SonarCloud
36 - name: SonarCloud Scan
37   run:
38     mvn -B verify sonar:sonar;
39     echo -e "Scanning the project with SonarCloud" >> realization.txt;
40   env:
41     GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
42     SONAR_TOKEN: ${ secrets.SONAR_TOKEN }
43
```

Figure 24 : Lancement du scan de mon projet

La ligne « mvn -B verify sonar:sonar » me permet de lancer un scan du projet et j'obtiens alors le rapport suivant (cf. Figure 25 à la page suivante) :

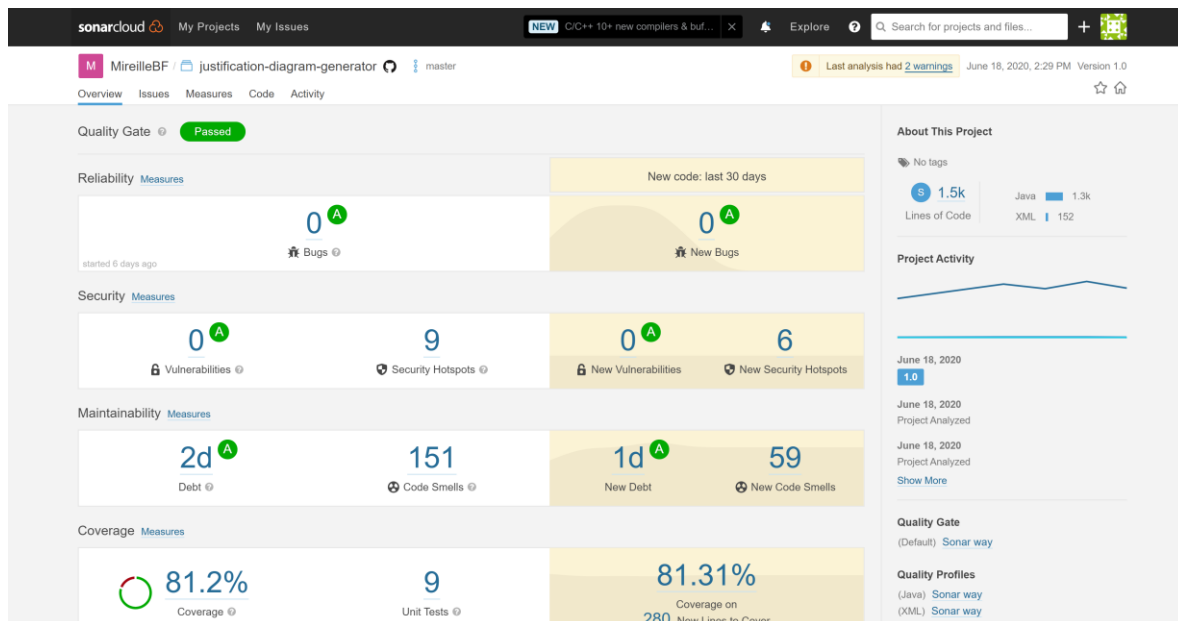


Figure 25 : aperçu du rapport de qualité de mon projet

Vous pouvez voir la totalité du rapport grâce aux liens suivants :

[https://sonarcloud.io/dashboard?id=MireilleBF\\_JustificationDiagram](https://sonarcloud.io/dashboard?id=MireilleBF_JustificationDiagram)

## Résultats

Pour résumer les différents éléments de mon projet, j'ai un fichier « .jd » qui est la structure de mon diagramme et de ma « todo list ». J'ai ensuite un fichier de réalisation qui est la trace de ce qui a été fait.

Pour ajouter des informations complémentaires et des informations à vérifier, j'ai un fichier « .json » qui est mon fichier d'action.

Pour finir, pour gérer mon intégration continue et le workflow, j'ai mon fichier « .yaml ».

Ces éléments me permettent de créer un patron, une instance de mon patron et une « todo list ». Voici les documents de justification de mon projet :

Patron (cf. Figure 26) :

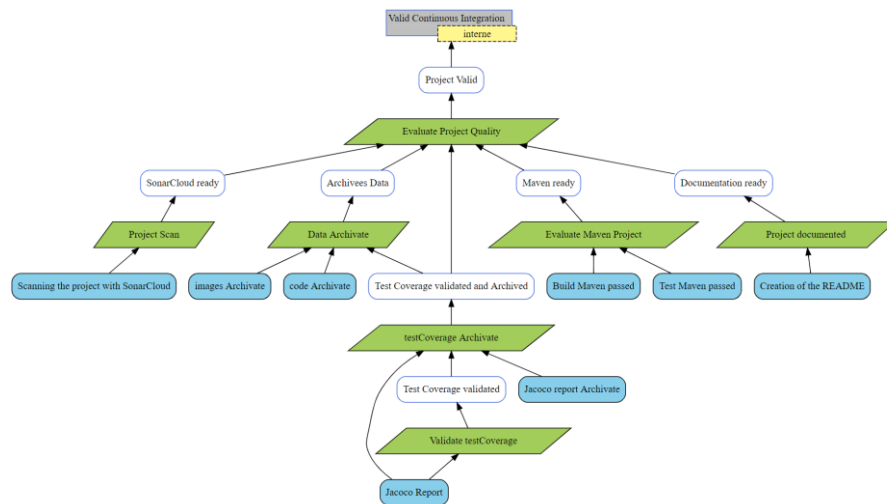


Figure 26 : Patron de justification pour le projet lui-même

Instance de mon patron (cf. Figure 27) :

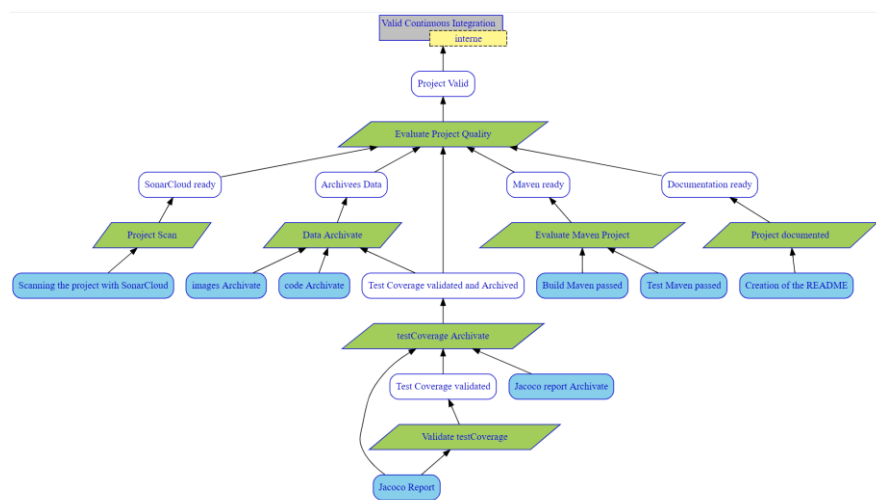


Figure 27 : Instance de mon Patron, Diagramme de justification du réalisé

« Todo list » (cf. Figure 28) :

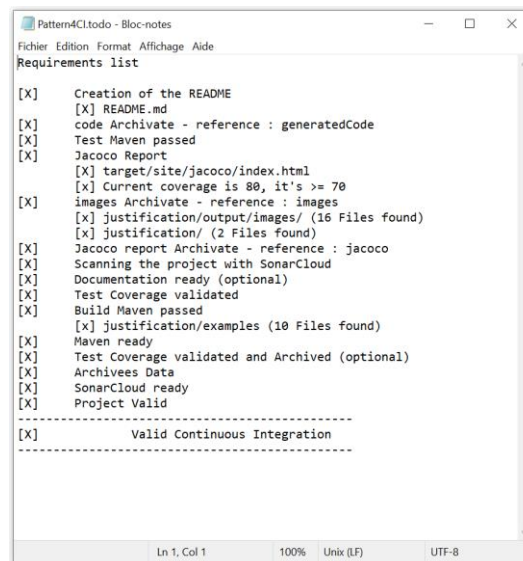


Figure 28 : todo list des réalisations de mon projet

Vous pouvez retrouver ces documents dans l'artéfact « GeneratedJD » à ce lien :

<https://github.com/MireilleBF/JustificationDiagram/actions/runs/143319901>

Pour quantifier mon projet, en comptant le fichier « yml », les tests et le code source et sans compter les lignes vides, le projet était à 1550 lignes, et aujourd'hui, il en compte 2650.

Si vous désirez voir la qualité de mon code, vous pouvez voir le rapport SonarCloud ci-dessous :

[https://sonarcloud.io/dashboard?id=MireilleBF\\_JustificationDiagram](https://sonarcloud.io/dashboard?id=MireilleBF_JustificationDiagram)



## Conclusion

J'ai travaillé sur le projet de Corinne Pulgar.

Ce dernier générait des diagrammes de justification qui sont des documents qui certifient le bon fonctionnement d'un projet. Ils sont décomposés en plusieurs nœuds représentant des étapes.

Madame Blay m'a donné pour objectif d'utiliser ce projet dans un contexte d'intégration continue en plus d'ajouter une notion d'État pour chaque nœud du diagramme. Ces états devaient me permettre de créer un diagramme qui montre l'avancement du projet en plus de créer une « todo list ». En plus de cela, j'ai ajouté une vérification d'information pour pouvoir valider l'état de certains nœuds.

J'ai aussi travaillé sur la documentation et j'ai créé un projet pour pouvoir tester mon programme grâce à une jar. Enfin, j'ai fait une démonstration à Sébastien Mosser, le premier superviseur du projet.

Finalement, j'ai réussi à générer un patron, qui est un diagramme représentant la démarche à prendre pour faire un projet et l'instance du patron, qui est un diagramme montrant l'avancement du projet. J'ai aussi généré une « todo list ».

Lors de ce projet individuel, j'ai appris l'intégration continue, la lecture d'un fichier « .json », l'ajout d'un scan d'un projet avec SonarCloud et l'application du design pattern « Command ».

J'ai aussi beaucoup appris en créant la documentation.

Lors de ce projet, les principales difficultés rencontrées étaient d'adapter le projet à l'usage et de le rendre « pratique ». Cependant, j'ai trouvé des solutions. Il s'agissait aussi d'un projet exploratoire, je n'en ai pas l'habitude, mais cela ne m'a pas dérangé. Pour finir, ce projet a énormément de potentiel. On pourrait facilement rajouter d'autres informations à vérifier dans le champ « action » de mon fichier « json ». Cela grâce au design pattern « Command ».

On pourrait facilement appliquer ce projet à d'autres domaines que l'informatique.

Ce projet ouvre différentes perspectives du court au long terme.

À court terme, des tests utilisateurs doivent être réalisés pour évaluer et faire évoluer mon projet.

À moyen terme, la possibilité de définir des actions doit permettre d'ajouter de nouvelles actions comme vérifier le format de fichier, approfondir l'analyse d'un rapport jacoco etc.

À plus long terme, l'approche doit pouvoir être appliquée à des champs d'applications non encore couverts et nécessitant des mécanismes automatiques de vérification.

## Glossaire

Patron :

Un patron est un diagramme de justification qui ne tient pas compte de l'état du projet. C'est une planification de comment va se passer le projet

Instance du patron :

une instance du patron est un diagramme qui nous permet de visualiser l'état du projet.

Workflow :

flux de travail permettant de modéliser et d'automatiser des flux d'information. C'est le processus qui est utilisé pour gérer l'intégration continue.

## Bibliographie

Design pattern « Command » :

[https://fr.wikipedia.org/wiki/Commande\\_\(patron\\_de\\_conception\)](https://fr.wikipedia.org/wiki/Commande_(patron_de_conception))

Aide sur le Workflow et l'intégration continue :

<https://help.github.com/en/actions/reference/workflow-syntax-for-github-actions>

Tuto SonarCloud :

<https://dev.to/remast/using-sonarcloud-with-github-actions-and-maven-31kg>

SonarCloud :

<https://sonarcloud.io/>

Github du projet individuel :

<https://github.com/MireilleBF/JustificationDiagram>

Rapport SonarCloud du projet individuel :

[https://sonarcloud.io/dashboard?id=MireilleBF\\_JustificationDiagram](https://sonarcloud.io/dashboard?id=MireilleBF_JustificationDiagram)

Github du projet de test :

<https://github.com/Nicolas-Corbiere/TestProjet>

Rapport SonarCloud du projet de test :

[https://sonarcloud.io/dashboard?id=Nicolas-Corbiere\\_TestProjet](https://sonarcloud.io/dashboard?id=Nicolas-Corbiere_TestProjet)

Thèse de Clément Duffau sur les diagrammes de justification :

<https://tel.archives-ouvertes.fr/tel-01978192/document>