

Calculer les métriques de vos projets avec Metrics

par Baptiste Wicht ([home](#))

Date de publication :

Qui ne s'est jamais demandé combien de lignes de code comptait son projet ? Le plugin Metrics pour Eclipse, répondra à cette question ainsi qu'à bien d'autres en calculant les métriques de vos projets Java.

| | |
|----------------------------------|----|
| I - Introduction..... | 3 |
| II - Installation..... | 4 |
| III - Utilisation..... | 5 |
| IV - Les métriques..... | 7 |
| V - Graphique de dépendance..... | 8 |
| VI - Configuration..... | 10 |
| VI - Conclusion..... | 12 |

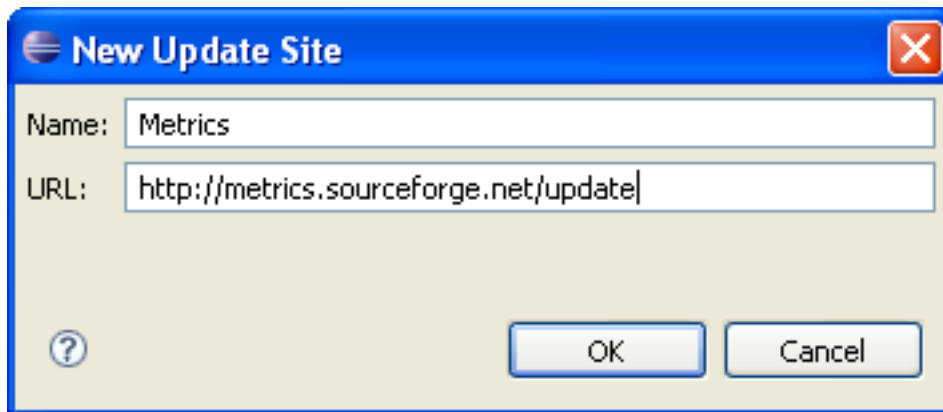
I - Introduction

Vous souhaitez connaître le nombre de lignes de code de votre projet, le nombre de classes et d'interfaces qu'il contient ? Ou encore la complexité cyclomatique de McCabe de votre projet ? C'est-à-dire en quelque sorte le nombre de chemins possibles à l'intérieur d'une méthode ; plus il y a de conditions et de boucles plus la complexité est grande et donc la méthode est plus compliquée à tester. Alors, Metrics est fait pour vous. Ce petit plugin pour Eclipse va vous permettre de calculer les métriques de vos projets. En plus des éléments cités ci-dessus, Metrics vous indiquera de nombreuses autres informations et vous indiquera quand est-ce qu'elles sont critiques.

II - Installation

Nous allons commencer par installer le plugin dans Eclipse. Aller dans le menu Help-> Software Updates -> Find and install. Choisissez l'option 2 "Search for new features to install". Ensuite, ajoutez un nouveau "remote site" et saisissez les informations suivantes :

- **Name** : Metrics
- **URL** : `http://metrics.sourceforge.net/update`



New Update Site

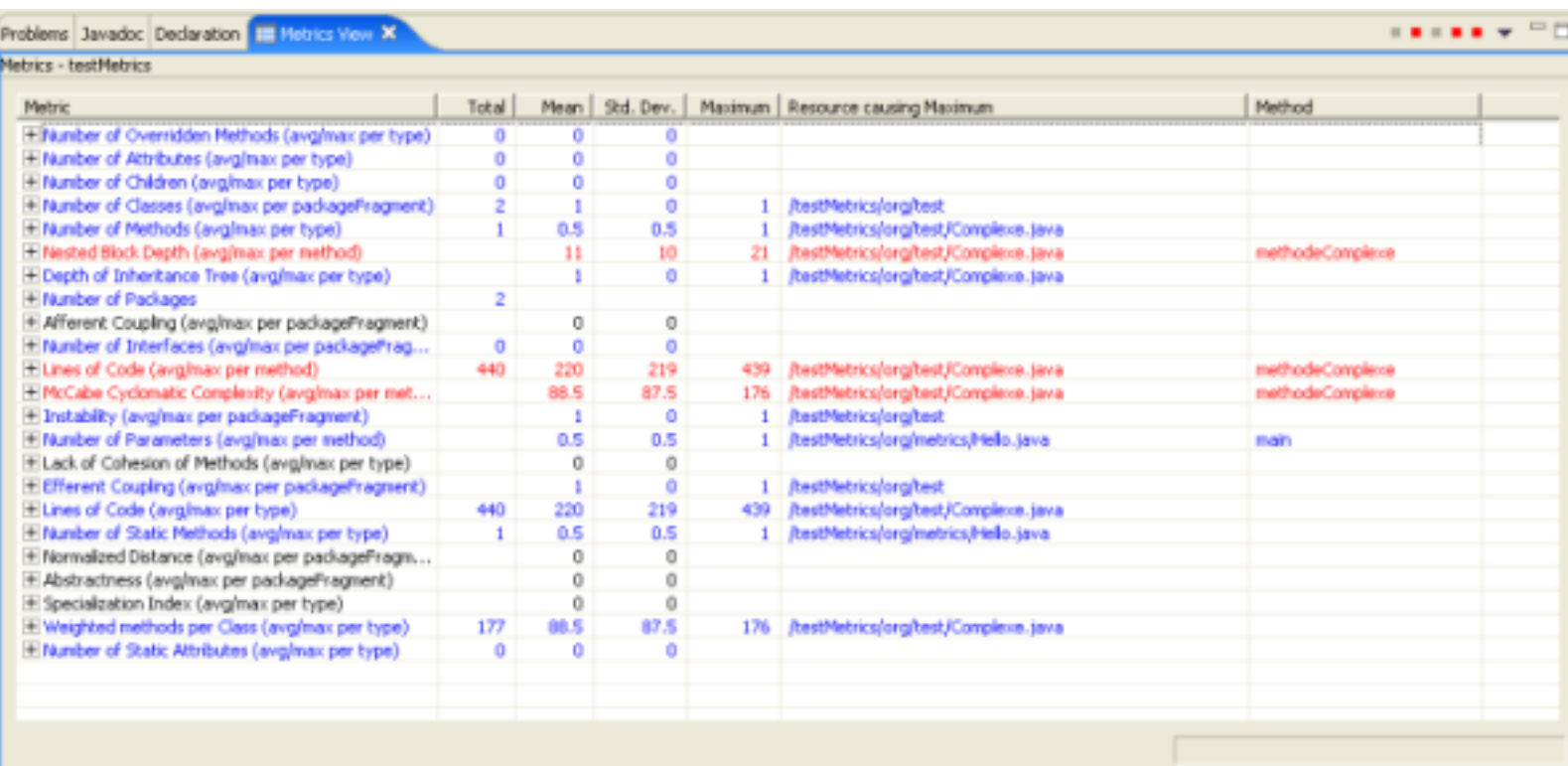
Cliquez ensuite sur OK, puis sélectionnez le site Metrics dans la liste et cliquez sur Finish. Si vous n'avez pas sélectionné l'option Automatically select mirror, choisissez le bon miroir de téléchargement et attendez. Ensuite, sélectionnez Metrics dans la liste des ressources trouvés, téléchargez-le et attendez le téléchargement. Ensuite, quand la fenêtre de confirmation de l'installation va arriver, cliquez sur Install pour installer le plugin. Il vous faudra ensuite redémarrer Eclipse pour que le plugin soit correctement installé.

Pour vérifier si Metrics est bien installé, cliquez droit sur un projet, puis sélectionnez Properties et vous devriez voir une option Metrics dans les préférences du projet. Si cette option est bien là, Metrics est correctement installé et nous pouvons maintenant passer à la suite du tutoriel. Si l'option n'est pas ici, c'est qu'il y a eu un problème pendant l'installation. Si vous êtes sûr d'avoir redémarré Eclipse, reprenez les étapes dès le début et recommencez l'opération.

III - Utilisation

Nous allons maintenant configurer Metrics pour un de nos projets. Choisissez donc un projet existant ou créez en un nouveau. Allez dans les propriétés du projet et dans les options Metrics, sélectionnez la case Enable Metrics puis appliquez les changements. Cela devrait normalement lancer un build du projet.

Ensuite, vous pouvez ouvrir la première vue de Metrics. Allez dans Window -> Show View -> Other et choisissez Metrics View dans le dossier Metrics. Vous devriez normalement voir apparaître une nouvelle vue dans Eclipse et normalement vous devriez voir le calcul de votre projet en cours. Dès que le calcul est terminé pour tous les fichiers du projet, vous devrez voir apparaître une fenêtre avec les résultats. Il peut arriver que la fenêtre n'arrive pas. Dans ce cas, exécutez un build complet de votre projet et vous devriez voir cette fois les résultats apparaître.



| Metric | Total | Mean | Std. Dev. | Maximum | Resource causing Maximum | Method |
|--|-------|------|-----------|---------|-------------------------------------|-----------------|
| Number of Overridden Methods (avg/max per type) | 0 | 0 | 0 | | | |
| Number of Attributes (avg/max per type) | 0 | 0 | 0 | | | |
| Number of Children (avg/max per type) | 0 | 0 | 0 | | | |
| Number of Classes (avg/max per packageFragment) | 2 | 1 | 0 | 1 | /testMetrics/org/test | |
| Number of Methods (avg/max per type) | 1 | 0.5 | 0.5 | 1 | /testMetrics/org/test/Complexe.java | |
| Nested Block Depth (avg/max per method) | | 11 | 10 | 21 | /testMetrics/org/test/Complexe.java | methodeComplexe |
| Depth of Inheritance Tree (avg/max per type) | | 1 | 0 | 1 | /testMetrics/org/test/Complexe.java | |
| Number of Packages | 2 | | | | | |
| Afferent Coupling (avg/max per packageFragment) | | 0 | 0 | | | |
| Number of Interfaces (avg/max per packageFrag... | 0 | 0 | 0 | | | |
| Lines of Code (avg/max per method) | 440 | 220 | 219 | 439 | /testMetrics/org/test/Complexe.java | methodeComplexe |
| McCabe Cyclomatic Complexity (avg/max per met... | | 88.5 | 87.5 | 176 | /testMetrics/org/test/Complexe.java | methodeComplexe |
| Instability (avg/max per packageFragment) | | 1 | 0 | 1 | /testMetrics/org/test | |
| Number of Parameters (avg/max per method) | | 0.5 | 0.5 | 1 | /testMetrics/org/metrics/Hello.java | main |
| Lack of Cohesion of Methods (avg/max per type) | | 0 | 0 | | | |
| Efferent Coupling (avg/max per packageFragment) | | 1 | 0 | 1 | /testMetrics/org/test | |
| Lines of Code (avg/max per type) | 440 | 220 | 219 | 439 | /testMetrics/org/test/Complexe.java | |
| Number of Static Methods (avg/max per type) | 1 | 0.5 | 0.5 | 1 | /testMetrics/org/metrics/Hello.java | |
| Normalized Distance (avg/max per packageFragm... | | 0 | 0 | | | |
| Abstractness (avg/max per packageFragment) | | 0 | 0 | | | |
| Specialization Index (avg/max per type) | | 0 | 0 | | | |
| Weighted methods per Class (avg/max per type) | 177 | 88.5 | 87.5 | 176 | /testMetrics/org/test/Complexe.java | |
| Number of Static Attributes (avg/max per type) | 0 | 0 | 0 | | | |

Métriques d'un projet

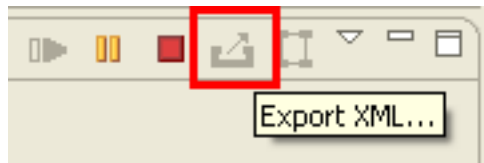
Vous trouverez l'explication de chacun des différentes métriques dans le chapitre suivant.

Vous pouvez soit sélectionner directement tout le projet, soit faire une sélection plus fine, en descendant au niveau du package ou même de la classe. Les résultats sont affichés d'abord au niveau global, mais vous pouvez étendre le résultat, pour descendre au niveau des packages et des classes. Vous aurez aussi chaque fois l'indication de la classe ou la méthode qui obtient le maximum. Pour les valeurs dépassant les rangs admis, la ligne sera colorée en rouge.

Voilà ce à quoi correspond les différentes colonnes des résultats :

- **Metric** : La métrique concernée
- **Total** : Indique le résultat total pour cette métrique, par exemple le nombre total de lignes de code
- **Mean** : Indique la moyenne. Vous pouvez voir dans le nom de la métrique sur quels éléments est calculée la moyenne, par exemple pour "Methods Lines of Code" : "avg/max per method" veut dire que la moyenne sera la moyenne de lignes de code par méthode.
- **Std. Dev** : Indique l'écart type entre la moyenne et le maximum
- **Maximum** : Indique la valeur maximale atteinte par notre projet pour cette métrique
- **Ressource causing Maximum** : Indique la classe ou le package qui cause le maximum pour cette métrique
- **Method** : Indique la méthode qui cause la maximum pour cette métrique. Elle est souvent pas employée

Vous pouvez exporter les données sous forme de rapport XML en cliquant sur le bouton suivant :



Exporter vers XML

IV - Les métriques

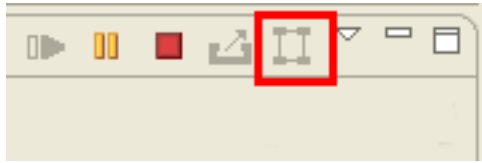
En tout, Metrics vous donne 23 données différentes sur votre projet. Voici les différentes informations et leur signification :

- **Lines of Code (LOC):** Le nombre total de lignes de code. Les lignes blanches et les commentaires ne sont pas comptabilisés
- **Number of Static Methods (NSM):** Le nombre de méthodes statiques dans l'élément sélectionné.
- **Afferent Coupling (CA):** Le nombre de classes hors d'une package qui dépendent d'une classe dans le package
- **Normalized Distance (RMD):** $RMA + RMI - 1$: Ce nombre devrait être petit, proche de zéro pour indiquer une bonne conception des parquets.
- **Number of Classes (NOC):** Le nombre de classes dans l'élément sélectionné.
- **Specialization Index (SIX):** $NORM * DIT / NOM$: Moyenne de l'index de spécialisation.
- **Instability (RMI):** $CE / (CA + CE)$: Ce nombre vous donnera l'instabilité de votre projet. C'est-à-dire les dépendances entre les paquets.
- **Number of Attributes (NOF):** Le nombre de variables dans l'élément sélectionné.
- **Number of Packages (NOP):** Le nombre de packages dans l'élément sélectionné.
- **Method Lines of Code (MLOC):** Le nombre total de lignes de codes dans les méthodes. Les lignes blanches et les commentaires ne sont pas comptabilisés
- **Weighted Methods per Class (WMC):** La somme de la complexité cyclomatique de McCabe pour toutes les méthodes de la classe.
- **Number of Overridden Methods (NORM):** Le nombre de méthodes redéfinies dans l'élément sélectionné.
- **Number of Static Attributes (NSF):** Le nombre de variables statiques dans l'élément sélectionné.
- **Nested Block Depth (NBD):** La profondeur du code
- **Number of Methods (NOM):** Le nombre de méthodes dans l'élément sélectionné.
- **Lack of Cohesion of Methods (LCOM):** Une mesure de la cohésion d'une classe. Plus le nombre est petit est plus la classe est cohérente, un nombre proche de un indique que la classe pourrait être découpée en sous-classe. Néanmoins, dans le cas de Javabeau, cette métrique n'est pas très correcte, car les getteurs et les setteurs sont utilisés comme seules méthodes d'accès aux attributs. Le résultat est calculé avec la méthode d' Henderson-Sellers : on prend $m(A)$, le nombre de méthodes accédant à un attribut A, on calcule la moyenne de $m(A)$ pour tous les attributs, on soustrait le nombre de méthodes m et on divise par $(1-m)$.
- **McCabe Cyclomatic Complexity (VG):** La complexité cyclomatique d'une méthode. C'est-à-dire le nombre de chemins possibles à l'intérieur d'une méthode, le nombre de chemin est incrémenté par chaque boucle, condition, opérateur ternaire, Il ne faut pas que ce nombre soit trop grand pour ne pas compliquer les tests et la compréhensibilité de la méthode.
- **Number of Parameters (PAR):** Le nombre de paramètres dans l'élément sélectionné.
- **Abstractness (RMA):** Le nombre de classes abstraites et d'interfaces divisés par le nombre total de classes dans un package. Cela vous donne donc le pourcentage de classes abstraites par package
- **Number of Interfaces (NOI):** Le nombre d'interfaces dans l'élément sélectionné.
- **Efferent Coupling (CE):** Le nombre de classes dans un packages qui dépendent d'une classe d'un autre package.
- **Number of Children (NSC):** Le nombre total de sous-classes directes d'une classe
- **Depth of Inheritance Tree (DIT):** Distance jusqu'à la classe Object dans la hiérarchie d'héritage.

Vous trouverez chaque fois entre parenthèses l'abréviation utilisée dans les préférences et les calculs de Metrics.

V - Graphique de dépendance

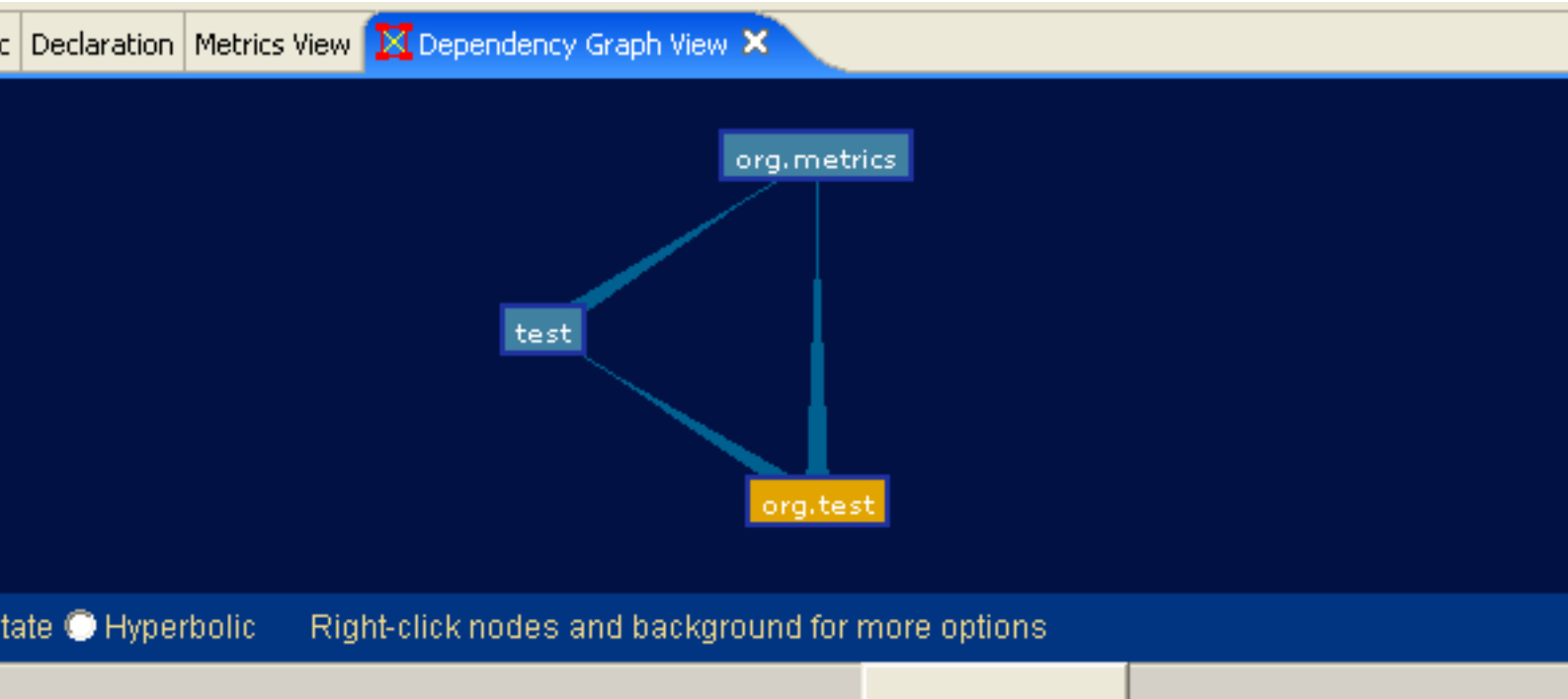
Vous trouverez chaque fois entre parenthèses l'abréviation utilisée dans les préférences et les calculs de Metrics.



Ouvrir le graphique de dépendance

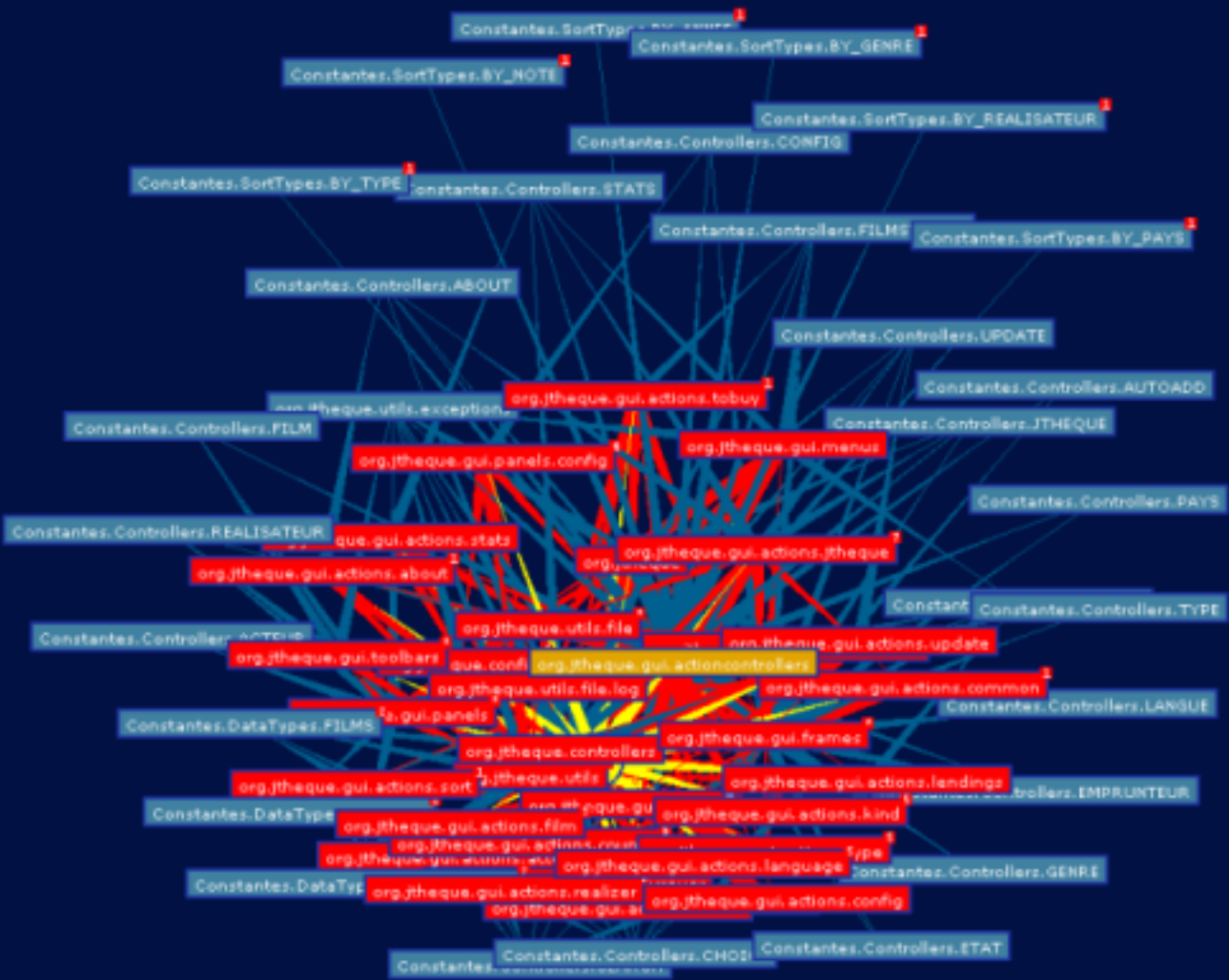
Soit aller dans Window -> Show view -> Other et sélectionner "Dependency Graph View". Il faut que le projet entier soit sélectionné pour utiliser le graphique de dépendance.

Cela va vous afficher un graphique de dépendance entre packages. Vous pouvez aussi étendre des packages pour afficher les dépendances entre classes.



Graphique de dépendance simple

Je ne vais pas m'étendre plus longtemps sur ce graphique. Les informations de base étant toutes contenues dans les métriques. Ce graphique est vite un peu fouilli pour les gros projets, pour vous donner une idée de ce que ça donne avec un projet assez gros (plusieurs centaines de classes), voici le diagramme de dépendances d'un de mes projets :

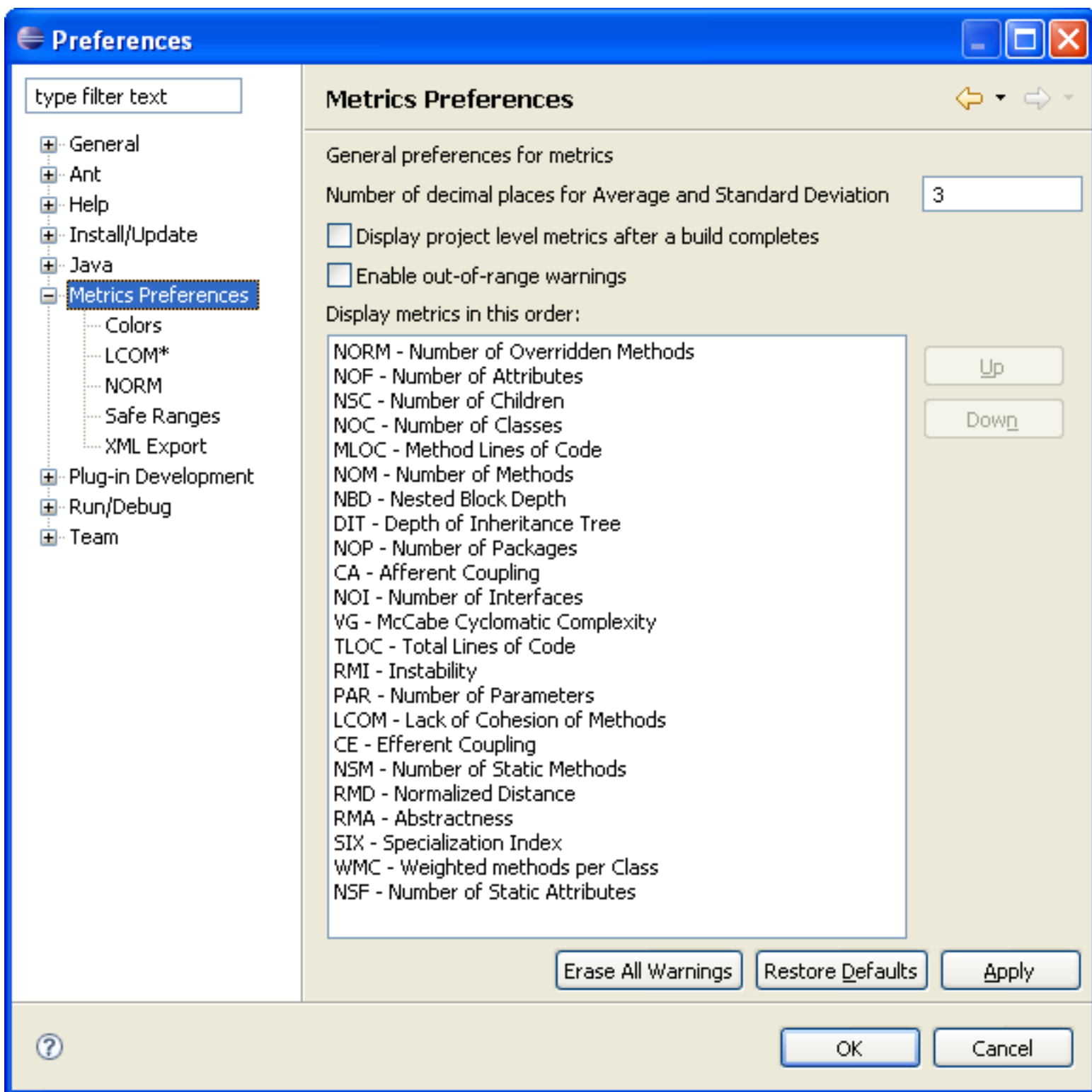


Graphique de dépendance complexe

Il n'est donc pas facile de s'y retrouver, c'est donc là qu'il faut jouer avec les différents nœuds pour les étendre et les refermer. Vous avez aussi accès à un petit outil qui permet de rechercher le chemin le plus court entre 2 points. Il vous suffit de cliquer droit dans le diagramme pour y avoir accès.

VI - Configuration

Vous pouvez configurer quelques petites propriétés de Metrics. Pour cela, allez dans Windows -> Préférences puis dans Metrics dans le panneau de sélection à gauche.



Préférences de Metrics

Dans la fenêtre principale, vous pouvez configurer l'ordre d'apparition des différentes données. Pour changer cet ordre, il vous suffit de sélectionner une donnée et d'utiliser les boutons Up et Down pour changer sa position.

Vous pouvez aussi indiquer si vous voulez activer les warnings pour les valeurs dépassant les rangs configurés.

Dans Colors, vous pouvez configurer les différentes couleurs d'affichage de Metrics. Dans LCOM*, vous pouvez configurer la métrique " Lack of Cohesion of Methods ". Vous pouvez choisir de compter les attributs statiques et les méthodes statiques ou non. Dans NORM, vous pouvez choisir de compter les méthodes héritées d'une classe abstraite et les méthodes invoquant super. Vous pouvez choisir d'exclure certaines méthodes du nombre total de méthodes redéfinies. Dans Safe Ranges, vous pouvez indiquer des minimas et des maximas pour chacun des métriques. Vous pouvez aussi définir un message d'aide pour indiquer comment résoudre le problème. Et enfin, dans XML Export, vous pouvez choisir la méthode d'exportation des métriques.

VI - Conclusion

Pour conclure, j'espère que ce tutoriel vous a été utile et que Metrics vous aidera à produire un code moins complexe et plus facile à maintenir ou tout simplement à fournir des statistiques de votre projet. Si vraiment vous voulez aller plus loin avec le graphique de dépendance, je vous invite à utiliser le **site officiel**.

Pour tous ceux qui s'intéressent aux métriques et au refactoring, sachez que le livre "Refactoring des applications Java/J2EE" (dont vous pouvez lire la critique [ici](#)) a un chapitre consacré aux métriques. De plus, je ne peux que vous conseiller ce livre qui se révèle très riche en conseils de refactoring pour des applications tant Java SE que Java EE.