**Résumé**

## Réponse à la RQ1 : Quels sont les critères d’une utilisabilité efficace d’API ?

1. **Critères liés à la facilité d’utilisation (Article 22)**

La facilité d'utilisation comprend des attributs tels que la **facilité d'apprentissage** d'une API, la **productivité** des programmeurs, l'**utilisation fréquente** des API, les bogues et parfois les problèmes de sécurité importants, ainsi que la **flexibilité** et **l’efficacité d'utilisation**ou les utilisateurs doivent être en mesure d'accomplir leurs tâches avec l'API efficacement.

1. **Critères liés à la documentation**

La documentation doit être suffisante et complète **(Article 22).**

1. **Critères liés au développement du code**
   1. **Présence des constructeurs par défaut (Article 21)**

Une étude a examiné les effets des paramètres constructeurs requis: classes sans constructeur par défaut (sans paramètre), de sorte que tous les constructeurs disponibles nécessitent certains paramètres à spécifier. Cette étude a révélé que, si les concepteurs de l'API s'attendaient à ce que les constructeurs requis soient plus faciles à utiliser et guideraient les utilisateurs dans l'utilisation correcte d'un objet, les programmeurs étaient plus rapides et faisaient moins d'erreurs lorsqu'il y avait des constructeurs par défaut.

* 1. **Nombre de classes dans un paquet (Article 24)**

Un nombre élevé de classes a un impact négatif sur la performance lors de la recherche d'une classe et donne une impression négative aux programmeurs. Lorsque le nombre de classes qui correspond à la fenêtre de complétion du code n’est pas trop grand, les programmeurs peuvent facilement obtenir un aperçu rapide des classes disponibles.

* 1. **Nombre de paramètres de la méthode (Article 24)**

Il est clair qu'un nombre plus élevé de paramètres de méthode signifie également qu'il faut plus de temps pour utiliser une méthode. Pour les comptages de paramètres entre 1 et 4 paramètres, certaine études montrent que chaque paramètre prend environ 5-8 secondes de temps du programmeur. Les paramètres qui n'étaient pas attendus par le programmeur ont tendance à prendre plus de temps que prévu, car ils nécessitent une réflexion supplémentaire, et tendent à faire le programmeur de chercher une meilleure méthode appropriée ou une surcharge.

1. **Critères liés à l’effet d’apprentissage (Article 24)**

L'effet d'apprentissage a un impact important sur la performance globale d'un programmeur. Quand un programmeur a utilisé une classe / méthode une ou deux fois, il n'a plus besoin de la rechercher, ce qui signifie que le temps de recherche est réduit à zéro. En outre, le temps d'utiliser la classe / méthode est réduit d'un certain degré.

La durée de ce temps est réduite en fonction de la complexité de la classe / de la méthode. Bien qu'il soit probable que le programmeur sera plus rapide lorsqu’il utilise une méthode avec 4 paramètres pour la deuxième fois, il n'y aura pas un grand changement pour une méthode avec 0 paramètres. Puisque la recherche d'une classe / méthode prend environ la moitié du temps, on peut s'attendre à ce que l'effet d'apprentissage entraîne toujours au moins une réduction de 50% du temps. On peut s'attendre à ce que l'effet d'apprentissage maximal soit atteint avec le troisième usage, et qu'il n'y ait plus d'amélioration de performance ultérieure.

## Réponse à la RQ2 : Comment améliorer l’utilisabilité d’une API ?

1. **Améliorations de la documentation**
2. **Création de documents élaborés (Article 20)**

De nombreuses fonctions API sont destinées à une utilisation généralisée, leurs auteurs sont motivés à investir beaucoup d'efforts dans la création de documents élaborés qui spécifie tout ce qu'un client peut avoir besoin de savoir sur une fonction.

1. **Éviter la déconnexion entre les auteurs d'une documentation et les développeurs** **(Article 20)**

Cette déconnexion est due non seulement au contenu du texte, mais aussi au moment où, comment et où il est présenté.

1. **Aidez les utilisateurs à reconnaître, à diagnostiquer et à récupérer des erreurs (Article 22)**

Un nombre surprenant d'API fournissent des informations d'erreur inutiles ou même pas du tout quand quelque chose tourne mal, ce qui diminue la convivialité et peut-être aussi affecter la correction et la sécurité.

1. **Proposer des exemples d’utilisation (Article 31)**

Fournir suffisamment des exemples d’utilisation dans la documentation officielle. Il y’a deux sources pour l’extraction des exemples d’utilisation d’API :

* Le code source des projets clients qui contiennent les usages réels des API.
* Les pages Web qui contiennent la discussion de certains problèmes d'utilisation des API.

Les approches existantes basées sur ces deux sources peuvent difficilement fonctionner pour les API qui sont nouvelles ou qui ne sont pas encore largement utilisées.

* Rassembler les exemples d'utilisations similaires de l'API et supprimer la redondance afin d’améliorer la représentativité des exemples extraits.

1. **Améliorations des mécanismes du complétion de code**
2. **Surcharges basées sur la simplicité des types de paramètres**

La plus part des IDEs commandent des surcharges d'abord par nombre de paramètres puis par ordre alphabétique (à commencer par le nom du type du premier paramètre), cela ne semble pas intuitif pour la plupart des programmeurs. Au lieu d'un ordre alphabétique, les articles peuvent par exemple être ordonnés par la simplicité des types de paramètres, par ex. Un paramètre de chaîne serait toujours commandé au-dessus d'un type complexe.

1. **Les surcharges importantes doivent être empêchées d'être bien en bas dans la liste (Article 23)**

Les programmeurs avaient tendance à ne pas défiler plus loin lorsqu'ils s'attendaient à trouver une surcharge facile à utiliser, par exemple, l'un avec un seul paramètre, alors que la meilleure surcharge appropriée avait trois paramètres. Les programmeurs ont tendance à ne pas regarder à travers la liste entière, donc il faut enlever de plus haut les surcharges rarement utilisés.

1. **Améliorations liées au développement du code**

1. **Nombre de classes dans un paquet (Article 24)**

Une restructuration simple en plusieurs (sous-) packages peut aider à améliorer les performances de recherche et l'expérience du programmeur.

  Lorsque seules les classes principales sont laissées dans le paquet principal, les programmeurs peuvent facilement identifier ce qui est le plus important dans l'API.

1. **Facilité d’instantiation (Article 26)**

Il est particulièrement important que les classes qui servent de point de départ à l'utilisation aient une faible barrière à l'entrée en exigeant un petit nombre de paramètres de constructeur.

Le nombre de paramètres du constructeur est une propriété importante concernant l'utilisabilité de ce concept. De plus, lors du choix du constructeur à utiliser, bien sûr également le nombre de surcharges du constructeur est important.

En ce qui concerne les paramètres du constructeur, non seulement le nombre de paramètres est important, mais aussi la complexité de chaque paramètre, car tous les paramètres ne sont pas aussi faciles ou difficiles à comprendre.

1. **Nombre de paramètres de la méthode (Article 24)**

Le nombre de paramètres de la méthode doit être maintenu aussi bas que possible. Les surcharges définissant des valeurs par défaut doivent être fournies pour des paramètres qui n'ont souvent pas besoin d'être définis explicitement. Un nombre élevé de surcharges se sont révélées significativement meilleures que le besoin de fournir un nombre élevé de paramètres.

1. **Changer les noms des méthodes avec le même préfixe (Article 23)**

Les programmeurs avaient la plupart des problèmes lorsque la méthode souhaitée était si loin dans la liste qu'il n'était pas directement visible sans le premier défilement vers le bas. Dans les IDE évalués, la fenêtre de complétion du code contenait environ 10 à 15 éléments, de sorte que lorsqu'une méthode importante apparaît plus basse que la 10ème position, il convient de supprimer ou de renommer la méthode ou d'autres méthodes placées ci-dessus pour améliorer l'utilisabilité. Les noms de méthode distinctifs aident à présenter au programmeur le moins de choix possibles.

1. **Plus de conscience pour les directives associées aux fonctions (Article 20)**

Les développeurs qui examinent un fragment de code doivent avoir plus de conscience pour les directives importantes qui sont associés aux fonctions invoquées car ces directives peuvent aider également ceux qui apprennent à utiliser l'API à partir d'échantillons de code.

## Améliorations liées à la conception de l’API

## Proposer une meilleure conception de l’API (Article 25)

Une bonne conception d'API peut considérablement réduire le temps et la difficulté qu'il faut à un développeur pour utiliser une API et peut aider à éviter les bugs coûteux dans le code qui utilise l'API.

1. **Utilisation des dimensions cognitives (Article 25)**

Les programmeurs préféraient et étaient plus efficaces avec les API qui ne nécessitaient aucun argument constructeur. Les raisons de cela sont devenues plus claires lorsqu’ils ont utilisé les dimensions cognitives pour analyser les résultats de l’étude. Une des raisons était que les paramètres du constructeur forcèrent les programmeurs à instancier chacun des objets de paramètres avant qu'ils ne sentaient qu'ils pouvaient explorer l'objet construit en utilisant des outils de conception comme la complétion du code.

**Consigne pour une meilleure conception d’un API (on peut mettre ça dans la présentation finale) (Article 29)**

. Une API ne doit pas être trop complexe. Complexité et flexibilité Devrait être équilibrée.

. Les noms des méthodes, des attributs et des classes doivent être auto-documentés et utilisés de façon cohérente.

. Fournir de la documentation et des exemples.

. Concevoir des API cohérentes (ordre des paramètres, sémantique des appels) et respecter les conventions (méthodes get / set).

. Aider les programmeurs à utiliser correctement une API en utilisant des éléments corrects.

. Faut pas utilisez de nombreux paramètres. Les valeurs de retour doivent indiquer le résultat de la méthode. Utilisez des exceptions lorsqu'un traitement exceptionnel est exigé.

. Toujours fournir un constructeur par défaut plutôt qu’un constructeur avec plusieurs paramètres.

. Définir les membres de la classe comme publics uniquement lorsque cela est nécessaire. Les exceptions doivent être traitées à proximité de l'endroit où elles se sont produites. Le message d'erreur doit transmettre des informations suffisantes.