**Résumé**

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

1. **Critères liés à la facilité d’utilisation**

Les critères liés à la facilité d’utilisationd’une API répondent à certaines questions [23] :

* À quel point l'API est facile à apprendre,
* Est-ce que l'API est efficace dans le cas où elle est utilisée pour des tâches spécifiques,
* À quel point les appels API sont faciles à retenir,
* Quelles sont les erreurs les plus fréquents que les programmeurs font en utilisant l'API.
* Comment les programmeurs perçoivent l'API.

Pour évaluer l’utilisabilité d’une API, on peut se baser sur Cognitive Dimensions of Notations (CDs) qui est un cadre léger créé pour décrire et analyser l'utilisabilité des systèmes de notation, tels que les interfaces utilisateur, les langages de programmation et les API. Les dimensions (CDs) de la facilité d'utilisation des API [27] :

* **Compréhensibilité**: évalue de l'effort nécessaire pour comprendre comment utiliser l'API pour atteindre une fonctionnalité souhaitée.
* **Abstraction**: Se rapporte à la capacité de l'API à garantir que les programmeurs peuvent utiliser la compétence de l'API sans exiger des connaissances ou des hypothèses spécifiques en relation avec ses détails d'implémentation. L'abstraction est typiquement corrélée avec le degré de confort que les développeurs ressentent lorsqu'ils utilisent l'API.
* **L'expressivité** peut être considérée comme la capacité d'inférer facilement le but d'une entité. L'expressivité est aussi liée à la facilité avec laquelle le programmeur construit son code sans avoir à assumer de modèle cognitif spécifique sur l'utilisation de l'API.
* **La réutilisabilité** détermine si le code client est maintenable et extensible. En particulier, cette dimension aborde le problème typique de la difficulté de modifier un code préexistant et de l'adapter à des exigences légèrement étendues ou plus générales.
* **Apprentissage :** Adresse la capacité du processus de l'apprentissage de l'API à être incrémenter. Les API apprenables permettent aux développeurs de comprendre les API de façon graduelle sans exiger des efforts disproportionnés initiaux.

1. **Critères liés à la documentation**
2. **Critères liés au développement du code**

* Une API doit être minimale, n’impose pas de désagréments injustifiés sur l'appelant.
* Le développeur ne devrait pas être tenu de faire quoi que ce soit le module pourrait le faire lui-même.
* Le niveau d'abstraction d'une API doit répondre aux attentes de l'utilisateur.
* L'API doit signaler les erreurs d'utilisation le plutôt possible, le mieux lors de la compilation.

1. **Critères liés à l’effet d’apprentissage**

* **Apprentissage :** Adresse la capacité du processus de l'apprentissage de l'API à être incrémenter. Les API apprenables permettent aux développeurs de comprendre les API de façon graduelle sans exiger des efforts disproportionnés initiaux.

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

1. **Améliorations de la documentation**

* Les solutions alternatives à un problème d'utilisabilité de l'API pourraient consister à modifier la documentation et à fournir plus d'exemples de code.
* Fournir différentes variantes de documentations d’API pour les débutants et les experts peut améliorer la courbe d'apprentissage globale.
* Les chercheurs ont vu clairement que les exemples de code commentés, et qui sont correctement construits, pourraient aider les programmeurs à apprendre très rapidement les entrées et les sorties des nouveaux appels API et leurs relations dans le contexte.
* **Fournir des exemples de code** : Pour répondre à la façon dont une certaine fonctionnalité peut être mise en œuvre à l'aide de l'API, des exemples de code sont parfois donnés ; Ils décrivent quelles méthodes et dans lequel l'ordre doit être appelé pour la mise en œuvre d'une certaine fonctionnalité [24].
* Les concepteurs d’API doivent améliorer la documentation et soutenir les utilisations inattendues. De plus ils doivent soutenir la rétrocompatibilité et accommoder les besoins changeants [51].
* Pour améliorer l’utilisabilité il faut améliorer la documentation des API. Puisque, les informations utiles aux programmeurs peuvent être enterrées dans un texte non pertinent ou associées à un élément API non intuitif, ce qui rend difficile la découverte. Les auteurs ont proposé de détecter et de recommander des fragments de documentation API potentiellement importants pour un programmeur qui a déjà décidé d'utiliser un certain élément API. Ils ont classé les fragments de texte dans la documentation de l'API selon qu'ils contiennent ou non des informations indispensables, utiles ou non. A partir des fragments qui contiennent des connaissances dignes de recommandation, ils ont extrait les modèles de mots, et d'utilisent ces modèles pour trouver automatiquement de nouveaux fragments qui contiennent des connaissances similaires dans la documentation invisible [52].

1. **Améliorations des mécanismes du complétion de code**

* Une approche de la complétion de code qui fournit rapidement aux développeurs des suggestions de programmation, telles que les méthodes d'appel et les expressions à utiliser comme paramètres.
* En raison du grand nombre d'API existantes, il est difficile d'apprendre, de se souvenir et de les utiliser lors du développement d'un logiciel pour atténuer le problème, les environnements de développement intègre des installations de complétion de code qui permettent aux développeurs de se rappeler chaque détail.
* Le système de complétion de code doit fournir des recommandations plus informatives des paramètres API et les classer de manière plus sophistiquée.
* Une autre approche améliore le système de complétion du code en introduisant de nouvelles fonctionnalités telles que le regroupement, le tri et le filtrage [21].
* Complétion de code basé sur l'exemple, qui exploite les informations contextuelles pour fournir une meilleure completion d'appel de méthode [25].
* Technique de complétion de code, appelée BMN. Cette technique est basée sur l’algorithme k-voisin le plus proche pour recommander des appels de méthode pour une variable.
* Classer les appels de méthode en fonction de leurs probabilités d'apparence dans le contexte de code existant pour aider le développeur à faire des choix plus efficaces.
* Détecter les usages suspects de l'API dans le code existant (c'est-à-dire les uns rarement / peu susceptibles d'être utilisés) et les réparer avec des usages plus probables [25].

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

* Les classes pour différentes tâches doivent être placées dans des packages/namespaces différents, et le paquet / espace de noms principal ne doit contenir que des classes pour des scénarios courants. Cela rend l'API plus facile à explorer pour les développeurs qui apprennent en expérimentant.
* Minimiser les désagréments injustifiés sur l'appelant car l'exposition inutile de la logique API interne, comme les classes internes et les méthodes, peut confondre l'utilisateur et donc conduire à une mauvaise utilisabilité.
* Une approche qui vise à améliorer la qualité du code source en remplaçant le code imitant par la méthode de bibliothèque qu'il imite.

1. **Améliorations liées à la conception de l’API**

* La désignation des éléments API est très importante. Les noms devraient être faciles à comprendre, cohérent, distinctif et proche de la langue du domaine.
* La meilleure façon pour un concepteur d'améliorer l'utilisabilité basée sur les résultats de mesure est en supprimant les concepts les plus complexes ou en réduisant leur complexité, en augmentant la réutilisation des concepts et en essayant généralement de réduire le nombre global de ces derniers.
* Rendre une API plus simple en réduisant sa fonctionnalité entraînera moins de complexité.
* Réduire le nombre de paramètres de méthode et de valeurs de retour parce qu’ils ont une forte influence sur l’utilisabilité de l’API.