REENGINEERING

1. **Idée Générale**

Le Reengineering est l'examen, l'analyse, et la restructuration d'un logiciel existant
pour le reconstituer sous une nouvelle forme et implémenter ainsi cette nouvelle forme. Le but de la réingénierie est de :

* Comprendre les artefacts du logiciel existant, à savoir, la spécification, la conception, l’implémentation, et la documentation ; et
* D’améliorer la fonctionnalité et la qualité des attributs du système. Quelques exemples
des attributs de qualité sont l'évolutivité, les performances et la réutilisabilité

Fondamentalement, un nouveau système est généré à partir d'un système opérationnel, tel que le nouveau système possède de meilleurs facteurs de qualité. Les facteurs de qualité de logiciels souhaités comprennent la fiabilité, l'exactitude, l'intégrité, l'efficacité, la maintenabilité, la convivialité, la flexibilité, la testabilité, l'interopérabilité, la réutilisation et la portabilité.

1. **Les concepts de la réingénierie**

L’abstraction et le raffinement sont les concepts clés utilisés dans le développement de logiciels, et les deux concepts sont également utiles dans la réingénierie.

**Principe de l’abstraction**. Le niveau d'abstraction de la représentation d'un système peut être progressivement développé en remplaçant successivement les détails par une information abstraite.

**Principe de raffinement**. Le niveau d'abstraction de la représentation du système est progressivement diminué en remplaçant successivement certains aspects du système avec plus de détails

Un nouveau logiciel est créé en allant vers le bas à partir du haut, du plus haut niveau d’abstraction vers le bas niveau d’abstraction. D'autre part, le mouvement vers le haut à travers les couches d'abstraction est appelée rétro-ingénierie (reverse engineering). Le Reverse engineering des systèmes logiciels est un processus comprenant les étapes suivantes : **(i)** Analyser le logiciel pour déterminer ses composantes et les relations entre ses composants et **(ii)** Représenter le système à un niveau supérieur d'abstraction ou sous une autre forme.

Les concepts d'abstraction et de raffinement sont utilisés pour créer des modèles de développement de logiciels sous forme de séquences de phases, tel que les phases correspondent à des niveaux spécifiques d’abstraction ou de raffinement, comme le montre la figure 1. Les quatre niveaux, à savoir, le niveau conceptuel, les besoins, la conception et l’implémentation, sont décrits ci-dessous :



**Figure 1**. Levels of abstraction and refinement. © 1992 IEEE

**Niveau conceptuel**. Au plus haut niveau d'abstraction, le logiciel est décrit en termes de concepts de très haut niveau et sa raison d'être (pourquoi ? pourquoi le système existe-t-il ?)

**Le niveau des besoins**. A ce niveau, les caractéristiques fonctionnelles (quoi ?) du système
sont décrites à un niveau élevé, (que permet de faire le système ?)

**Niveau de la conception**. Au niveau de la conception-raffinement, les caractéristiques du système (quoi ? et comment ?), à savoir, les principaux composants, le style architectural des composants, les interfaces entre les composants, les algorithmes, structures de données internes majeures, et les bases de données sont décrites en détail.

**Niveau d’implémentation**. Ceci est le plus bas niveau d'abstraction dans la hiérarchie. À ce niveau, le système est décrit à un niveau très faible d’abstraction en termes de détails d’implémentation dans un langage de haut niveau.

En plus des deux principes de l'abstraction et de raffinement, un principe optionnel appelé altération sous-tend de nombreuses méthodes de réingénierie.

**Principe d'altération**. Tout changement effectué à la représentation d’un système est connu comme l'altération. L’altération n’implique aucun changement au degré de l'abstraction, et n’implique aucune modification, ni suppression ni ajout d’information.

La Figure 2 montre l'utilisation des trois principes fondamentaux pour expliquer les caractéristiques de réingénierie.



FIGURE 2. Conceptual basis for the reengineering process. © 1992 IEEE

Un autre terme étroitement liée à «l’Altération» est la restructuration. Dans le contexte de la réingénierie, le terme «restructuration» est défini comme la transformation d'une forme de représentation à une autre au même niveau d’abstraction tout en préservant le comportement extérieur du système.

1. **Un modèle général pour la RÉINGÉNIERIE LOGICIELLE**

Le processus de réingénierie accepte en entrée le code existant d'un système et produit le code du système rénové. Fondée sur les différents niveaux d'abstraction utilisés dans le développement de logiciels, la figure 3, initialement proposé par Eric J. Byrne[[1]](#footnote-1), représente les processus pour tous les niveaux d'abstraction de la réingénierie. Ce modèle présente la réingénierie comme une séquence de trois activités : « reverse engineering », « Alteration », et « forward engineering »



FIGURE 3. General model of software reengineering. © 1992 IEEE

Une métaphore visuelle appelée fer à cheval, comme le montre la figure 4, a été développé par Kazman et al.[[2]](#footnote-2) pour décrire un processus de réingénierie architectural en trois étapes. Trois segments distincts du fer à cheval sont le côté gauche, la partie supérieure, et le côté droit. La première étape, représentée sur le côté gauche, vise à extraire l'architecture à partir du code source en utilisant le principe de l'abstraction. La deuxième étape, représentée sur la partie supérieur, implique la transformation de l’architecture vers l'architecture cible en utilisant le principe de l’altération. Enfin, la troisième étape, représentée sur le côté droit, implique la production de la nouvelle architecture au moyen de raffinement.



FIGURE 4. Horseshoe model of reengineering. © 1998 IEEE

En résumé, il est évident que la réingénierie consiste à : **(i)** la création d’une vue plus abstraite du système par le biais de certaines activités de rétro-ingénierie (reverse engineering) ; **(ii)** la restructuration de la vue abstraite ; et **(iii)** Implémentation du système dans une nouvelle forme au moyen d'activités du forward engineering. Ce processus est formellement présenté par Jacobson et Lindstorm[[3]](#footnote-3) par l'expression suivante :

Reengineering = Reverse engineering + Δ + Forward engineering.

"Reverse engineering", est une activité pour créer une forme facile à comprendre et plus abstraite du système. Le troisième élément, à savoir, " Forward engineering " est le processus traditionnel de passage d'une abstraction de haut niveau et logique, de conception indépendante de l'implémentation à l’implémentation physique du système. Le deuxième élément "Δ" représente les modifications apportées au système d'origine. Deux grandes dimensions de l’altération sont le changement dans la fonctionnalité et le changement dans la technique de l’implémentation. Un changement dans la fonctionnalité est issu d'une modification des règles de gestion, et pour ce qui est du changement dans la technique d’implémentation, la réingénierie est souvent associée à l’introduction de nouvelles technologies de développement.

* 1. **Les types de modification**

Le modèle de la figure 3 suggère qu'un système existant peut être restructuré par l'un des quatre chemins suivants : (recode, redesign, respecify, rethink). Les modifications au sein d'un groupe particulier se traduisent par des modifications dans les niveaux d'abstraction inférieurs. Par exemple, un besoin se traduit par de nombreux éléments de conception, et un composant de conception est réalisée par un bloc de code source. Ainsi, un petit changement dans un composant de conception peut nécessiter plusieurs modifications au code.

**Recoder**. Les modifications de niveau du code source sont effectuées au moyen de reformulation (rephrasing) et de traduction du programme. Dans cette dernière approche, un programme est transformé en un programme dans un langage différent. D'autre part, la reformulation maintient le programme dans le même langage. Des exemples de scénarios de reformulation sont la normalisation, l'optimisation, la refactorisation (refactoring), et la rénovation.

**Reconcevoir**. Les caractéristiques de conception du logiciel sont modifiées par la reconception du
système. Les Changements généraux à la conception du logiciel incluent : **(i)** la restructuration de l'architecture ; **(ii)** la modification du modèle de données du système ; et **(iii)** le remplacement d'une procédure ou d'un algorithme par un autre plus efficace.

**Respécifier**. Cela implique de changer les caractéristiques des besoins du système de deux façons : **(i)** changement de la forme des besoins et **(ii)** changement de la portée des besoins. La première façon se réfère seulement à la modification de la forme des besoins existants, le dernier type de changements inclut les modifications comme l'ajout de nouveaux besoins, la suppression de certains besoins et la modification de certains besoins existants.

**Repenser**. Ceci implique de modifier les caractéristiques conceptuelles du système, ceci peut conduire le système à un changement fondamental. La transition du développement pour les téléphones cellulaires ordinaires vers le développement des smartphones est un exemple de « Rethink ».

* 1. **les stratégies de la réingénierie logicielle**

Les Trois stratégies qui spécifient les étapes de base de la réingénierie sont la réécriture (rewrite), retravailler (rework), et remplacer (replace).



FIGURE 5. Conceptual basis for reengineering strategies. © 1992 IEEE

**La stratégie Rewrite**. Au moyen de l’altération, un système opérationnel est transformé en un nouveau système, tout en préservant le niveau d'abstraction du système initial.

**La stratégie Rework**. La stratégie « Retravailler » applique les trois principes. Tout d'abord, par le biais du principe d'abstraction, permet d'obtenir une représentation du système avec moins de détails que ce qui est disponible à un niveau donné. Ensuite, le modèle du système reconstitué est transformé dans la représentation du système cible, au moyen d'une altération, sans changer le niveau d'abstraction. Enfin, par le biais de raffinement, une nouvelle représentation appropriée du système est créée à un niveau inférieur de l'abstraction.

**La stratégie Replace**. Dans le but de changer une certaine caractéristique d'un système : **(i)** le système est reconstruit à un niveau d'abstraction plus élevé en cachant les détails de la caractéristique et **(ii)** une représentation appropriée pour le système cible est générée à un niveau d'abstraction inférieur par application de raffinement.

1. **Processus de réingénierie**

Un ensemble ordonné d'activités conçues pour effectuer une tâche spécifique est appelé **Processus**.
Pour faciliter la compréhension et la communication, les processus sont décrits par des modèles de processus. Dans cette section, nous discuterons de deux modèles de processus pour la réingénierie du logiciel. Nous allons d’abord expliquer cinq grandes approches dans les paragraphes suivants.

* 1. **Les approches de la réingénierie**

Plusieurs considérations sont faites lors du choix d’une approche de réingénierie, notamment :

* objectifs du projet ;
* la disponibilité des ressources ;
* l'état actuel du système en cours de réingénierie ; et
* les risques dans le projet de réingénierie.

**L’approche Big Bang**. Une fois l’effort de réingénierie est lancée, il est poursuivi jusqu'à ce que tous les objectifs du projet soient atteints et le système cible soit construit. Cette approche est généralement utilisée si la réingénierie ne peut se faire en plusieurs étapes. L'avantage conséquent de cette approche est que le système est mis dans son nouvel environnement en une seule étape. L’approche Big Bang consomme trop de ressources à la fois pour les grands systèmes et prend beaucoup de temps avant que le nouveau système ne voit le jour.

**L’approche incrémentale**. Par cette approche, la réingénierie du système est effectuée graduellement. Les versions provisoires successives répondent de plus en plus aux objectifs du projet que les versions précédentes. L'approche incrémentale encourt un risque plus faible que l'approche «Big Bang» en raison du fait que, lorsqu’un composant est restructuré (reengineered), les risques associés au code correspondant peuvent être identifiés et surveillés.

**L’approche partielle**. Dans cette approche, une partie seulement du système est restructuré (reengineered) et elle est intégrée à la partie du système non restructurée (non engineered). Les trois étapes suivantes sont suivies dans l'approche partielle :

* Dans la première étape, le système existant est divisé en deux parties : une partie est
identifiée pour être restructurée (reengineered) et la partie restante à ne pas restructurer.
* Dans la deuxième étape, la réingénierie est effectuée en utilisant soit le "Big Bang"
soit l'approche "incrémentale".
* Dans la troisième étape, les deux parties, à savoir, la partie restructurée et la partie non restructurée sont intégrées pour constituer le nouveau système.

**L’approche itérative.** Le processus de réingénierie est appliqué sur quelques procédures à la fois du code source, avec un court laps de temps pour chaque opération de réingénierie. Ce processus est exécuté à plusieurs reprises sur différents composants à différentes étapes.

**Approche évolutive**. Semblable à l'approche «incrémentale», dans cette approche les composants du système d'origine sont substitués par les composants restructurés (reengineered). Cependant, dans cette approche, les composants existants sont regroupés par fonctions et restructurés en de nouveaux composants.

* 1. **Modèle de référence de réingénierie de code source (SCORE/RM)**

Le modèle a été proposé par Colbrook, Smythe, et Darlison[[4]](#footnote-4). Le cadre, illustré à la figure 6, se compose de quatre types d'éléments : la fonction, la documentation, la base de données et le système métrique. L'élément « fonction » est divisé en huit couches, à savoir, l'encapsulation, la transformation, la normalisation, l'interprétation, l'abstraction, la causalité, la régénération et la certification.

Les améliorations effectuées dans le logiciel comme résultat de la réingénierie sont quantifiés au moyen de l'élément du système métrique. L'élément de métrique est décrit en termes de métriques logicielles pertinentes avant d'exécuter une couche et après l'exécution de la même couche. Les spécifications, les contraintes, et les détails d’implémentation de l'ancienne et la nouvelle version du logiciel sont décrits dans l'élément de documentation. La base de données est la banque d'informations pour le processus de réingénierie en entier, contenant les types d'informations suivants : le système métrique, la documentation, ainsi que l'ancien et le nouveau code.



FIGURE 6. Source code reengineering reference model. © 1990 IEEE

**Encapsulation**. Dans cette première couche un identifiant est affecté à chaque version du logiciel source pour faciliter sa réingénierie. Les fonctions suivantes sont réalisées dans cette couche :

* **Gestion de la configuration**. Les modifications apportées au logiciel en cours de maintenance sont enregistrées en suivant une procédure bien documentée et définie pour une utilisation ultérieure dans le nouveau code source.
* **Analyse**. Les parties du logiciel nécessitant une réingénierie sont évaluées.
* **Parsing**. Le code source du système à restructurer est traduit en un langage intermédiaire (IL). Tous les algorithmes de la réingénierie peuvent agir sur la représentation IL du code source.
* **Génération de test**. la conception de tests de certification pour évaluer le nouveau logiciel.

**Transformation**. Le flux de contrôle du code est modifié pour obtenir un code structuré.

**Normalisation**. A ce stade, les données et leurs structures sont examinées par les fonctions suivantes

* Réduction des données : les données dupliquées sont éliminés.
* Représentation des données. Une nouvelle représentation des données est générée.

**Interprétation**. La couche d'interprétation réalise les fonctions suivantes :

* Fonctionnalisation. Les données et les structures du code sont vérifiées et améliorées
* La lecture du programme. Cela signifie annoter le code source avec des commentaires.

**Abstraction**. Le code source annoté est examiné par le moyen d’abstraction pour identifier les hiérarchies d'objets sous-jacents. La couche d'abstraction assure les fonctions suivantes :

* Identification des objets.
* Interprétation des objets.

**Lien de causalité**. Cette couche réalise les fonctions suivantes :

* Spécification des actions. Concerne les services fournis à l'utilisateur.
* Spécification de contraintes. Concerne les limitations dans lesquelles le logiciel fonctionne correctement.
* Modification de la spécification. La spécification est étendue et / ou réduite pour correspondre aux besoins de l'utilisateur.

**Régénération**. Signifie la ré-implémentation du code source en utilisant les besoins et les spécifications fonctionnelles. La couche effectue les fonctions suivantes :

* Génération de la conception. Concerne la production et la documentation de la conception détaillée.
* Génération de code. Concerne la génération du nouveau code par la réutilisation des portions de
code d'origine et l'utilisation de bibliothèques standards.
* Génération de test. De nouveaux tests sont générés pour effectuer des tests unitaires et d'intégration sur le code source développé et réutilisé.

**Certification** : Le logiciel nouvellement produit est analysée pour établir qu'il :

(i) fonctionne correctement ;

(ii) accompli les besoins spécifiées ; et

(iii) Est compatible avec le code original. La couche assure les fonctions suivantes :

* Validation et vérification. Le nouveau système est testé pour s’assurer de sa performance.
* Conformité. Les tests sont effectués pour montrer que le code source rénové effectue au minimum toutes les fonctionnalités effectuées par le code source d'origine.

**4.3 Modèle de réingénierie en phase**

Le modèle comprend cinq phases : l'analyse et la planification, la rénovation, le test du système cible, re-documentation, et les tests d'acceptation et la transition du système, tel que représenté dans la figure 7.



FIGURE 7. Software reengineering process phases. © 1992 IEEE

**Analyse et planification**. L’Analyse traite trois problèmes techniques et un enjeu économique. Le premier problème technique concerne l'état actuel du système à restructurer et la compréhension de ses propriétés. Le deuxième problème technique concerne l'identification de la nécessité pour la restructuration du système. Le troisième problème technique concerne la spécification des caractéristiques du nouveau système à produire. L'enjeu économique concerne une analyse des coûts et bénéfices du projet de réingénierie.

La planification comprend (i) la compréhension de la portée du travail ; (ii) l'identification
des ressources nécessaires ; (iii) identifier les tâches et les étapes importantes ; (iv) l'estimation de l’effort nécessaire ; et (v) la préparation d'un calendrier.

**Rénovation**. Un système opérationnel est modifié vers le système cible dans la phase de rénovation. Deux principaux aspects du système sont examinées dans cette phase : (i) la représentation du système et (ii) la représentation des données externes. En général, le premier aspect concerne le code source, mais il peut inclure le modèle de conception et la spécification des besoins du système existant. D'autre part, le deuxième concerne la base de données et/ou les fichiers de données utilisés par le système.

**Test du système cible**. Dans cette phase de test du système, les défauts sont détectés dans le
système cible. Ces défauts pourraient avoir été introduits au cours de la réingénierie. La détection des défauts est effectuée en appliquant le plan de test du système cible sur le système cible.

**Redocumentation**. Dans la phase de redocumentation, les documentations sont réécrites,
mise à jour, et / ou remplacées pour refléter le système cible. Les deux principales tâches au sein de cette phase sont : (i) d'analyser le nouveau code source et (ii) créer la documentation.

**Acceptation et transition du système**. Dans cette phase finale, le système restructuré (reegineered) est évalué en effectuant les tests d'acceptation. Les critères d'acceptation devraient déjà avoir été mis en place au début du projet. À l'issue des tests d'acceptation, le système restructuré est rendue opérationnel, et l'ancien système est mis hors service.

1. **Rétro-ingénierie de code (code reverse engineering)**

Dans le contexte de l'ingénierie logicielle, Chikofsky et Cross II [[5]](#footnote-5) ont défini la rétro-ingénierie comme un processus à **(i)** identifier les composants d’un logiciel opérationnel ; **(ii)** identifier les relations entre ces composants ; et **(iii)** représenter le système à un niveau supérieur d'abstraction ou sous une autre forme.

Les facteurs responsables de la nécessité d’un reverse engineering sont les suivants :

* Le langage de l'application est devenu obsolète, et le système doit être migré vers un nouveau langage
* Il y a peu de documentation sur le système.
* Le système nécessite des adaptations et / ou des améliorations.
* Le logiciel ne fonctionne pas comme prévu.

Un haut niveau de reverse engineering signifie la création d’une abstraction du code source sous forme de conception, d’architecture et / ou de documentation. Un Bas niveau reverse engineering, signifie la création de code source à partir du code objet ou code assembleur.

Le reverse engineering est effectuée pour atteindre deux objectifs clés : la redocumentation
de artéfacts et le recouvrement (récupération) de la conception.

On identifie six étapes clés dans le reverse engineering, tel que documenté dans la norme IEEE pour
la Maintenance logicielle :

* Partitionner le code source en unités ;
* décrire les significations de ces unités et identifier les unités fonctionnelles ;
* créer les schémas d'entrée et de sortie des unités identifiées ;
* décrire les unités connectées ;
* décrire l'application du système ; et
* créer une structure interne du système.

Les trois premières des six étapes impliquent une analyse locale, parce qu’elles sont effectuées
au niveau de l'unité. D'autre part, les trois étapes restantes impliquent une analyse globale, parce que ces étapes sont effectuées au niveau du système.

1. **Le techniques utilisées dans le reverse engineering**

Les techniques d'analyse qui facilitent le reverse engineering sont l'analyse lexicale, analyse syntaxique, l'analyse des flux de contrôle, l'analyse des flux de données, le découpage en tranches du programme (program slicing), la visualisation, et les mesures du programme. Dans les sections suivantes, nous détaillons ces techniques, une par une.

* 1. **L’analyse lexicale :** Analyse lexicale est le processus de décomposition d’une séquence de caractères dans le code source en ses unités lexicales constitutives. Le programme effectuant une analyse lexicale est appelé un analyseur lexical, et il fait partie du compilateur de langage de programmation.
	2. **L’analyse syntaxique :** L’analyse syntaxique est effectuée par un « parseur ». Deux types de représentations sont utilisés pour faire valoir les résultats de l'analyse syntaxique : l’arbre d’analyse (parse tree) et l’arbre de syntaxe abstraite (abstract syntax tree). Le premier est similaire aux schémas d'analyse utilisés pour montrer comment une phrase en langage naturel est décomposée en ses constituants. Cependant, un arbre d'analyse contient des détails étrangers au sens du programme, tels que la ponctuation. L’élimination de ces détails superflus produit une structure appelée arbre de syntaxe abstraite (AST). Un AST contient seulement des détails qui se rapportent à la signification réelle du programme.
	3. **Analyse du flux de contrôle :** Les deux types d’analyse du flux de contrôle (CFA) sont l'analyse intra-procédurale et l'analyse inter-procédurale. La première montre l'ordre dans lequel les instructions sont exécutées dans un sous-programme, tandis que la seconde montre la relation d'appel entre les unités de programme. L’analyse intraprocédurale est effectuée en générant un graphe de contrôle de flux (CFG) de sous-programmes. Un CFG montre une vue abstraite de la manière dont un sous-programme peut s’exécuter.

L’analyse Inter-procedurale est effectuée par la construction d'un graphe d'appel.
Les relations d’appel entre les sous-programmes d'un programme sont représentées comme un graphe d'appel qui est essentiellement un graphe orienté. En particulier, une procédure dans le code source est représentée par un nœud dans le graphe, et l’arc entre les nœuds "f" à "g" indique que la procédure "f" appelle procédure "g".

* 1. **Analyse du flux de données** : L’analyse du flux de données (DFA) concerne la façon dont les valeurs des variables définies affluent et sont utilisées dans un programme. Le CFA peut détecter la possibilité de boucles, alors que DFA peut déterminer des anomalies de flux de données. Un exemple d’anomalie de flux de données est qu’une variable non définie soit référencée. Un autre exemple d’anomalie de flux de données est qu’une variable soit définie successivement sans être référencée entre les deux définitions.
	2. **Découpage de programme** : La tranche d'un programme pour une variable donnée à une ligne donnée de code est la partie du programme qui donne une valeur à la variable en ce point. Par conséquent, si on détermine pendant le débogage que la valeur d'une variable à une ligne spécifique est incorrecte, on peut regarder la tranche de programme correspondant pour trouver le code défectueux. Dans la définition de Weiser, un critère de découpage d'un programme P est S <p;v> où p
	est un point de programme et v est un sous-ensemble de variables dans P.

Un « backward slice » par rapport à un variable v et un point donné p contient toutes les instructions et les prédicats qui affectent la valeur de v au point p. Alors qu’un « forward slice » pour une variable v et un point p dans un programme, comprend toutes les instructions et les prédicats qui peuvent dépendre de la valeur de v au point p.

A titre d'exemple, considérons le programme illustré à la figure 8 qui est un bloc de code écrit en C. Le « backward slice » compte tenu du critère de tranchage S <[7]; somme> est montré dans la figure 9. Pour le critère de tranchage S <[3]; produit>, le « forward slice » est représenté sur la figure 10. Outre la détection de défauts, le découpage du programme est également utilisé pour extraire des règles d'affaires (business rules) et dans le refactoring.



FIGURE 8. A block of code to compute the sum and product of all the even integers in

the range [0,N] for N≥3



FIGURE 9. The backward slice of code obtained from Figure 8 by using the criterion

S<[7]; sum>



FIGURE 10. The forward slice of code obtained from Figure 8 by using the criterion

S<[3]; product>

* 1. **La visualisation** : La visualisation de logiciels est une stratégie utile pour permettre à un utilisateur de mieux comprendre le logiciel. Dans cette stratégie, un système logiciel est représenté au moyen d'un objet visuel pour avoir une idée sur la façon dont le système a été structuré. Deux notions importantes pour la conception de visualisations de logiciels utilisant les technologies 3D et la réalité virtuelle sont les visualisations et les représentations.
* La Représentation. est la description d'un seul composant au moyen de graphiques et autres médias.
* La Visualisation. est une configuration d'un ensemble d’informations liées sur les représentations constituant un composant de haut niveau.

Essentiellement, les symboles graphiques sont utilisés pour représenter les composants. Dans ce graphe, par exemple, les nœuds et les arcs sont les représentations, alors que le graphe lui-même est la visualisation.

* 1. **Les métriques de programme** : Pour comprendre et contrôler le processus global d'ingénierie logicielle, des mesures du programme sont appliqués. Le Tableau 2 résume les mesures du programme couramment utilisés.

L'un des indicateurs de complexité les plus largement utilisés est la complexité cyclomatique. Le concept de point de fonction (FP) a été introduit en 1970 par Albrecht [[6]](#footnote-6) comme une autre mesure basée sur le simple comptage de ligne de code. Le but de la FP est de mesurer la quantité de fonctionnalité fournie par un programme. Intuitivement, plus la fonctionnalité d'un programme est élevée, plus son FP est grand. Basé sur les caractéristiques du flux information du « fan-in » et « fan-out » d’un module, Henry et Kafura [[7]](#footnote-7) ont défini une mesure de complexité, Cp = (fan-in × fan-out)2. Fan-in et fan-out sont expliqués dans le tableau 2. Un grand fan-in et un grand fan-out peuvent être des symptômes d'une mauvaise conception.

Table 2. Métriques logicielles couramment utilisées



1. E. J. Byrne. 1992. A Conceptual Foundation for Software Reengineering. Proceedings

of the International Conference on Software Maintenance, November 1992, Orlando,

Florida. IEEE Computer Society Press, Los Alamitos, CA. pp. 226–235. [↑](#footnote-ref-1)
2. R. Kazman, S. Woods, and J. Carrière. 1998. Requirements for Integrating Software Architecture and Reengineering Models: Corum ii. Proceedings of Working Conference on Reverse Engineering (WCRE), Washington, DC. IEEE Computer Society Press, Los Alamitos, CA. pp. 154–163. [↑](#footnote-ref-2)
3. I. Jacobson and F. Lindstrom. 1991. Re-engineering of Old Systems to an Object-oriented Architecture. Proceedings of the ACM Conference on Object Oriented Programming Systems Languages and Applications, October 1991. ACM Press, New York, NY. pp. 340–350. [↑](#footnote-ref-3)
4. A. Colbrook, C. Smythe, and A. Darlison. 1990. Data Abstraction in a Software Reengineering Reference Model. Proceedings of the International Conference on Software Maintenance, November 1990, San Diego, CA. IEEE Computer Society Press, Los Alamitos, CA. pp. 2–11. [↑](#footnote-ref-4)
5. E. J. Chikofsky and J. H. Cross II. 1990. Reverse engineering and design recovery. IEEE Software, January, 13–17. [↑](#footnote-ref-5)
6. A. J. Albrecht. 1979. Measuring Application Development Productivity. Process Joint SHARE/GUIDE/IBM Application Development Symposium, October 1979. Reprinted in 1981 in Programming Productivity: Issues for the Eighties (Ed. Capers Jones), No. EHO 186-7, pp. 34–43. Computer Society Press. [↑](#footnote-ref-6)
7. S. M. Henry and D. G. Kafura. 1981. Software structure metrics based on information flow. IEEE Transactions on Software Engineering, 7(5), 510–518. [↑](#footnote-ref-7)