Comprendre les programmes des autres

L’effet combiné à grande échelle des logiciels d’aujourd’hui, de l’utilisation répandue de composants open source et des fréquents changements de projets et de tâches font que l’ingénieur logiciel doit gérer un code inconnu. Bien que ses études et son expérience puissent l'aider à acquérir des connaissances sur le framework, il doit encore cartographier de grands ensembles de code source en modèles avec lesquels il peut travailler.

De toute évidence, toute la documentation existante et le personnel compétent pourraient être une excellente introduction dans ce vaste nouveau monde. Mais souvent, l'ingénieur doit travailler seul avec le code source et trouver son chemin.

Le code source a une structure naturelle avec tous ses répertoires et ses fichiers. En descendant encore, les fonctions et les variables globales - et dans les langages plus récents, les classes et les espaces de noms/paquets - montrent le niveau suivant de la structure de programme. Les relations d'utilisation telles que les appels et l'utilisation des données relient ces éléments structurels. Toutes ces structures peuvent être appelées l'architecture physique du logiciel.

Les diagrammes architecturaux exprimant l'architecture physique sont essentiellement des cartes qui aident l'ingénieur logiciel à naviguer et à comprendre les éléments et leurs interdépendances. Pour que ces diagrammes architecturaux deviennent utiles, un élément clé est qu'ils soient interactifs et permettent à l'utilisateur de se déplacer librement à travers les niveaux d'abstraction. Mais contrairement au zoom sur des cartes géographiques où le reste disparaît à l'exception de la zone d'intérêt, les diagrammes architecturaux peuvent toujours montrer les zones environnantes à des niveaux d'abstraction plus élevés, tout en explorant une zone. De cette façon, ils s'adaptent au comportement des programmeurs qui travaillent à tous les niveaux d'abstraction.

Le graphe d'appel traditionnel en tant qu'abstraction de l'ordre d'exécution instruction par instruction a bien sûr toujours sa place dans le processus de compréhension. Un point de départ pourrait bien être le point d'entrée du programme, regroupant des parties non intéressantes en sous-systèmes au fur et à mesure de sa croissance. Encore une fois, suite à l'observation selon laquelle les programmeurs travaillent à tous les niveaux d'abstraction, les outils qu'ils utilisent doivent fournir ces différents niveaux d'affichage de la structure et du flux de contrôle, ainsi que des moyens faciles de combiner et de basculer entre eux. Idéalement, ces niveaux vont jusqu'au niveau du bloc d'instructions, affiché sous forme d'organigrammes.

Selon le but de l'exploration du programme, une approche de flux de données pourrait conduire à une compréhension plus rapide des parties pertinentes. Le flux de données dans sa forme la plus simple commence par la variable de données et examine toutes les fonctions qui le manipulent directement ou indirectement. Une approche plus évoluée consiste à suivre la façon dont les valeurs des données sont calculées à partir d'autres valeurs de données. Cette

tranche basée sur les données du programme peut en théorie montrer comment chaque variable de données est calculée à partir de l'ensemble des valeurs d'entrée.

La troisième dimension de la compréhension du programme est la concurrence. Bien sûr, certains programmes ont un flux de contrôle purement linéaire et les notions de contrôle décrites ci-dessus sont donc suffisantes. Mais l'utilisation de tâches dans des systèmes intégrés ou de threads dans des applications hébergées est plus courante maintenant et nécessite des modèles supplémentaires pour la compréhension.

La première étape consiste à identifier les tâches du programme. En règle générale, les tâches commencent par un appel à une routine système et peuvent donc être déterminées à l'aide de recherches de base sur le graphique des appels. L'étape suivante consiste à déterminer comment les tâches interagissent entre elles via des variables de données partagées ou via des événements ou des signaux. Ceux-ci nécessitent des modèles graphiques supplémentaires sous la forme de flux de tâches ou de diagrammes de collaboration.

Dans l'ensemble, le processus de compréhension du code inconnu consiste à construire des modèles dans l'esprit du programmeur. Les modèles consistent à résumer et à voir les concepts utilisés dans le programme. Pour aider le programmeur avec cela, différents niveaux d'abstraction doivent être combinés et il doit être facile de basculer entre eux.

Des outils tels que **Imagix 4D** https://www.verifysoft.com/fr_imagix4d.html sont à la disposition du programmeur pour une analyse efficace du code source.

Source: https://www.imagix.com/blog/understanding-other-peoples-programs/