Comment s’attaquer aux problèmes (Libres conseils 10/42)

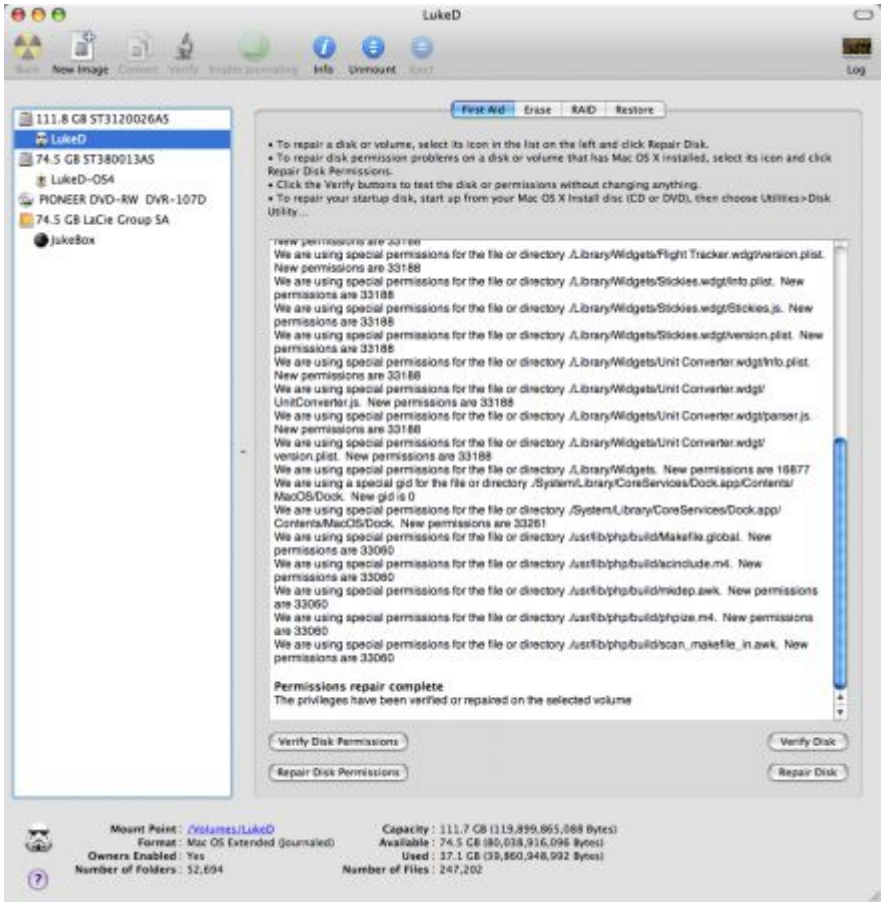
Chaque jeudi à 21h, rendez-vous sur le framapad de traduction, le travail collaboratif sera ensuite publié ici même.

Traduction Framalang : Sky, LIAR, lerouge, Goofy, peupleLa, lamessen LAuX, KoS, Nys, Julius22, okram, kalupa, 4nti7rust, CoudCoud, zn01wr + Sinma

L’art de résoudre les problèmes

Thiago Madeira

Thiago Macieira est doublement diplômé. Il a une maîtrise en administration des affaires (MBA) et un diplôme d’ingénieur. Mais son implication dans le mouvement open source, depuis près de 15 ans maintenant, est antérieur à ses diplômes. Participant actif des communautés KDE, Qt et MeeGo, il a été ingénieur logiciel et responsable produit pour Qt, il a fait des conférences et il a écouté les gens. À présent, Thiago vit à Oslo en Norvège et quand il ne travaille pas sur Qt, il essaye — sans grande réussite — d’améliorer son skill à StarCraft 2.



Les problèmes forment une routine à laquelle nous sommes confrontés presque tous les jours ; nous les résolvons et c'est tellement habituel que bien souvent nous n'en avons même pas conscience. Cela peut être des situations aussi simples que chercher le meilleur chemin pour arriver à destination ou trouver la meilleure façon de tout faire tenir dans le réfrigérateur. Ce n'est que lorsque nous ne parvenons pas à les résoudre immédiatement que nous remarquons les problèmes car nous devons alors nous arrêter et y réfléchir. Notre vie professionnelle n'échappe pas à cette règle et la résolution de problèmes commence à faire partie de la description du poste à pourvoir.

La résolution de problèmes était le sujet de mon premier cours quand j'ai commencé ma formation d'ingénieur. Dans cet amphithéâtre bondé du siècle dernier, notre professeur expliquait à environ 700 étudiants de première année en quoi les ingénieurs étaient des *solutionneurs* de problèmes et comment nos vies professionnelles consisteraient à enchaîner les problèmes à résoudre. Certains seraient des problèmes faciles résolus en deux temps trois mouvements ; d'autres seraient tellement difficiles que nous aurions besoin d'une structure de projet et d'une équipe pour les résoudre — mais la plupart se situeraient entre ces deux extrêmes. Puis il commença à donner des exemples sur la façon dont sa propre mentalité de « solutionneur de problèmes » l'avait aidé dans sa vie

professionnelle et personnelle, et nous offrit même un exemple en direct quand tout à coup le projecteur nous tomba dessus.

La faculté de résoudre des problèmes est un talent que nous pouvons affiner par la pratique et un travail de fond. La pratique est quelque chose que l'on ne peut acquérir que par l'expérience, par succession d'essais et d'erreurs (1) ; ce n'est donc pas quelque chose qu'on peut apprendre dans un livre. Se mettre en situation de résoudre des problèmes, en revanche, est quelque chose que l'on peut apprendre. Face au problème, l'expérience est comme notre boîte à outils, et les techniques de résolution le mode d'emploi des outils.

Formuler correctement la question

La question à laquelle nous essayons de répondre fournit la direction que nous allons prendre en essayant de résoudre le problème. Posez la mauvaise question et les réponses seront peu pertinentes, invalides ou juste complètement fausses. Par conséquent, poser la bonne question est essentiel. De plus, poser correctement la bonne question est important, car cela apporte des indices quant à ce que nous recherchons. La manière la plus inutile d'énoncer un problème qu'on puisse rencontrer est : « ça marche pas », c'est pourtant un grand classique. Certes, l'énoncé est juste, puisque manifestement quelque chose a planté. Néanmoins, cette façon de présenter le problème n'apporte aucun indice sur le point de départ pour rechercher des solutions.

Les systèmes de gestion de bogues imposent souvent au rapporteur du bogue de préciser les actions effectuées qui ont conduit à ce problème, la description de ce qui s'est passé (c'est-à-dire le symptôme) et une description du comportement attendu. La comparaison entre le symptôme et le comportement attendu est un bon point de départ pour poser la question fondamentale : « pourquoi cela s'est-il produit, pourquoi cet autre comportement ne s'est-il pas produit ? ». Même si ce n'est pas la seule manière d'y arriver, appliquer cette technique à des problèmes peut certainement aider à formuler la question.

Formuler correctement le problème et la question, dans ses moindres détails, est aussi une manière de décrire davantage le problème tel qu'il s'est manifesté. En premier lieu, nous devons avoir conscience que le problème ne se trouve probablement pas où nous nous attendons à le trouver — si c'était le cas, nous l'aurions probablement déjà résolu. Présenter tous les détails du problème permet

à l'assistance technique d'avoir plus d'informations pour travailler. De plus, même si c'est contre-intuitif, le fait de décrire le problème dans sa totalité conduit souvent à trouver la solution, si bien que de nombreux groupes de développement ont besoin que des développeurs se concentrent sur cette tâche, soit en discutant avec un collègue soit en s'adressant à un être innocent, tel qu'un canard en caoutchouc ou M. Patate.

De plus, il faut revenir régulièrement à la question afin de garder l'objectif dans le viseur. Lors de la résolution du problème, il convient de faire attention à ne pas se concentrer exclusivement sur l'une de ses parties en perdant de vue l'objectif global. Pour la même raison, il est nécessaire de reprendre la question de départ lorsqu'on a trouvé une solution éventuelle, pour pouvoir s'assurer qu'elle couvre bien l'intégralité du problème. Là encore, cela prouve bien la nécessité de poser la bonne question, qui décrira le problème dans son intégralité : sans la question complète, la solution pourrait être également incomplète.

Diviser pour mieux régner (2)

L'expérience que j'ai acquise en aidant des utilisateurs en ligne à résoudre leurs problèmes m'a appris que la plupart des personnes considérent leurs difficultés comme des blocs d'achoppement, monolithiques et indivisibles, qu'il faut traiter comme un tout. Vu sous cet angle, un vaste problème pose une question à laquelle il est très difficile de répondre entièrement.

À vrai dire, la grande majorité de ces problèmes peut se décomposer en plusieurs petits problèmes qu'il est donc plus facile de traiter séparément afin de déterminer s'ils sont la cause originelle du problème, sans parler de la possibilité qu'il y ait plusieurs origines au symptôme rapporté. Répéter cette opération, ne serait-ce qu'un petit nombre de fois, revient à s'attaquer à des problèmes mieux circonscrits, et amène par conséquent à des solutions plus rapides.

Cependant, plus nous sommes obligés de décomposer, plus nous devons connaître le fonctionnement interne du système que nous avons sous la main. De fait, celui qui doit résoudre un problème ne pourra le décomposer qu'aussi loin que sa connaissance du sujet le lui permettra, et c'est depuis ce point qu'il pourra ensuite traiter la question.

Pour ce qui concerne le développement logiciel, les sous-systèmes utilisés sont

souvent de bons indices pour trouver comment décomposer le problème. Par exemple, si le problème implique une transmission de données par TCP/IP, deux subdivisions possibles sont l'expéditeur et le destinataire : il ne sert à rien de chercher le problème du côté du destinataire si l'expéditeur ne transmet pas les données correctement. De même, une application graphique qui n'affiche pas les données appelées dans une base de données a une division claire : ce serait une bonne idée de vérifier d'abord que l'accès à la base de données fonctionne avant d'enquêter sur la cause du mauvais affichage. Par ailleurs, on pourrait également envoyer des informations quelconques aux fonctions d'affichage et vérifier que ces données s'affichent correctement.

Même quand les regroupements ne sont pas faciles à faire, diviser le problème peut tout de même contribuer à éclairer la question. En fait, les divisions sont presque toujours utiles, car elles réduisent la quantité de code à inspecter et le niveau de complexité à gérer. Au pire, le simple fait de diviser le code en deux parties et de chercher le problème dans l'une des deux peut être utile. Cette technique, nommée bissection, est recommandée si les divisions créées à partir des sous-systèmes et des interfaces n'ont pas encore apporté de solution.

Une succession de divisions appropriées aura pour résultat final un petit exemple autonome qui expose le problème. À ce stade, l'une des trois options suivantes est habituellement la bonne : le problème peut être identifié et localisé ; le code est en fait correct et c'était ce que l'on en attendait qui était faux ; ou bien un bogue a été trouvé dans la couche de code de plus bas niveau. Un des avantages de ce procédé, c'est qu'il génère un scénario de test à joindre à un rapport de bogue, pour peu qu'un bogue soit en cause.

Conditions aux limites (3)

Une question similaire à la division du problème est celle des conditions aux limites. En mathématiques et en physique, les conditions aux limites sont l'ensemble des valeurs qui déterminent la région de validité des équations résolues. Pour le logiciel, les conditions aux limites sont l'ensemble des conditions qui doivent être satisfaites pour que le code s'exécute correctement. Habituellement, les conditions aux limites sont loin d'être simples : à la différence des mathématiques ou de la physique, les variables des systèmes logiciels sont beaucoup trop nombreuses, ce qui signifie que leurs conditions aux limites sont également légion.

Dans les systèmes logiciels, les conditions aux limites sont souvent nommées « conditions préalables », c'est-à-dire des conditions qui doivent être satisfaites *avant* qu'une certaine action ne soit autorisée. Vérifier que ces conditions prélalables ont été satisfaites est un bon exercice dans la recherche d'une réponse, car leur violation est clairement un problème qui doit être résolu — quand bien même ce n'est pas la cause première du problème initial. Des conditions préalables peuvent tout simplement prendre la forme d'un pointeur qui doit être valide avant qu'on puisse le déréférencer, ou d'un objet qui ne doit pas être éliminé avant de pouvoir être utilisé. Les conditions préalables complexes seront très probablement documentées en vue de l'utilisation du logiciel.

Un autre groupe intéressant de conditions aux limites se caractérise, curieusement, par ce qui n'est pas autorisé : le comportement indéfini. Ce type de conditions aux limites est très commun lorsque l'on traite des spécifications qui essaient d'être très explicites sur la manière dont le logiciel est censé se comporter. Les compilateurs et les définitions de langage en sont un bon exemple. À strictement parler, déréférencer un pointeur *null* est un comportement indéfini : la conséquence la plus commune en est l'enregistrement d'une exception du processeur et l'arrêt du programme, mais d'autres comportements sont aussi autorisés, y compris le fonctionnement sans faille.

Le bon outil pour le bon usage

Si les ingénieurs sont des solutionneurs de problèmes, la devise de l'ingénieur est « Utilise le bon outil pour le bon usage ». Cela peut sembler évident, étant donné qu'on ne s'attend pas à ce que quelqu'un utilise un marteau pour résoudre un problème électronique. Cependant, les cas d'utilisation du mauvais outil sont plutôt fréquents, souvent parce qu'on ignore qu'il existe un meilleur outil.

Certains de ces outils sont la base du développement logiciel, comme le compilateur et le débogueur. L'incapacité à se servir de ces outils est impardonnable : le professionnel qui se retrouve dans un environnement d'outils nouveaux ou inconnus, si par exemple il change de poste ou d'emploi, doit consacrer du temps à apprendre à les utiliser, à se familiariser avec leurs fonctionnalités et limitations. Par exemple, si un programme plante, être capable de déterminer l'endroit du plantage ainsi que les variables appelées dans cette portion du code peut aider à trouver la cause du problème et donc la solution.

D’autres outils peu connus sont plus évolués, prévus pour des emplois spécifiques, ou encore ne sont disponibles qu’à un prix ou sous des conditions que l’ingénieur ne peut réunir. Ils peuvent toutefois être incroyablement utiles pour contribuer à la résolution de problèmes. Il peut s’agir de vérificateurs de code statique ou de processus, de débogueurs de mémoire, d’enregistreurs d’événements matériels, etc. Par exemple, le matériel de développement inclut souvent un système permettant de le contrôler à l’aide d’une interface spéficique comme JTAG ou de lister toutes les instructions exécutées et l’état des processeurs. Mais cela nécessite d’avoir du matériel et des outils spécifiques, qui ne sont pas facilement accessibles et coûtent plus cher que les machines et périphériques grand public. Un autre exemple est la suite d’outils Valgrind (4), qui comprend un vérificateur de processus et des débogueurs de mémoire. L’ensemble est gratuit, facilement disponible, mais fait partie de ces outils spécifiques de haut niveau dont l’usage n’est pas enseigné à l’école.

Connaître le contenu de sa boîte à outils est un savoir précieux. L’utilisation d’un outil spécialisé pour chercher un problème va probablement donner un résultat plus rapide, qu’il soit positif — et confirme le problème — ou négatif, et oriente la recherche dans une autre direction. Par ailleurs, il est important de savoir comment utiliser ces outils, ce qui justifie le temps passé à lire la documentation, à s’entraîner ou simplement à expérimenter ces outils avec des problèmes connus pour comprendre comment ils fonctionnent.

Conclusion

Résoudre les problèmes est un art accessible à tous. Comme pour tous les arts, certaines personnes semblent avoir une telle facilité qu’ils semblent être nés avec cette compétence. Mais en réalité, avec assez d’expérience et de pratique, la résolution des problèmes devient une activité insconsciente.

Quand on est confronté à un problème qui n’est pas évident à résoudre, il faut s’asseoir et le considérer dans son intégralité. Quel problème avons-nous ? Pouvons-nous formuler la question à laquelle nous devons répondre ? Une fois que nous savons ce que nous cherchons, nous pouvons commencer à examiner où peut être situé le problème. Peut-on le décomposer en parties plus petites et plus maniables ? Quels sont les meilleurs outils à utiliser pour chaque partie ? Avons-nous vérifié que nous utilisions correctement les fonctionnalités et services disponibles ?

Après avoir résolu de nombreux problèmes, on commence à repérer des schémas. Il devient plus facile de détecter des indices subtils à partir des symptômes et de diriger les recherches vers le problème réel. Un correcteur de problèmes expérimenté peut même ne pas se rendre compte que cette action a lieu. C'est un signe que l'expérience et les automatismes se sont si bien mis en place qu'il n'y a plus besoin d'effort conscient pour accéder à ces compétences.

Bien sûr il, y aura toujours des problèmes qui seront difficiles à résoudre dans la vie : problèmes professionnels, existentiels, philosophiques ou même ceux qui sont causés par la pure curiosité. Là encore, c'est le défi qui nous stimule, le besoin de comprendre toujours plus et mieux. Sans cela, la vie serait trop triste.

(1)
http://fr.wikipedia.org/wiki/Apprentissage#Apprentissage_par_essais_et_erreurs

(2) http://fr.wikipedia.org/wiki/Diviser_pour_régner_(informatique)

(3) http://fr.wikipedia.org/wiki/Condition_aux_limites

(4) http://fr.wikipedia.org/wiki/Valgrind

Crédit photo Luxuryluke (CC BY-NC-ND 2.0)