Cours #1 - Débogage

Contrôle de la qualité I (420-PA4-AG)
Automne 2018

1

Objectifs

- Définitions
- Cycle de vie d'un bogue
- Classification des bogues
- Processus général
- Techniques de débogage

2

3

Définitions

4

Qu'est-ce qu'un bogue?

- Quelque chose que le logiciel fait et qu'il n'est pas censé faire

ou

- Quelque chose que le logiciel ne fait pas et qu'il est censé faire

Qu'est-ce que le débogage?

- Processus consistant à comprendre le comportement d'un système pour faciliter la suppression des bogues.

5

Pourquoi se préoccuper des bogues ?

6

Coûts des bogues

- Faible productivité
- Taux de roulement du personnel élevé
- Nouveaux employés = Hausse $ formation
- Mauvaise image = Hausse $ marketing
- Délai de livraison = Diminution $ profit

7

Cycle de vie d'un bogue

8

Pourquoi les bogues sont-ils créés?

- Complexité
- Manque de temps
- Manque d'argent
- Changements constants
- Manque d'expérience
- Mauvaise compréhension
- Mauvaises spécifications

9

Classification des bogues

10

Fuite de mémoire (1/2)

- Bogue dans lequel la mémoire est allouée à partir du système d'exploitation ou d'un «pool» de mémoire interne, mais jamais libérée.
- Les fuites de mémoire peuvent également constituer des défaillances pour récupérer des ressources.

11

Fuite de mémoire (2/2)

```
// Can you spot the "memory leak"?
public class Stack {
    private Object[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(Object e) {
        ensureCapacity();
        elements[size++] = e;
    }

    public Object pop() {
        if (size == 0)
            throw new EmptyStackException();
        return elements[--size];
    }

    /**
     * Ensure space for at least one more element, roughly
     * doubling the capacity each time the array needs to grow.
     */
    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```


12

Erreur de logique (1/2)

- Une erreur de logique se produit lorsque le code est syntaxiquement correct mais ne fait pas ce qu'on s'attend de lui.
- Des erreurs de logique peuvent se produire en raison d'erreurs typographiques de programmation.
- Une erreur de logique ne peut pas être interceptée par un compilateur, un débogueur ou analyseur de code car le code est correct et fonctionnera comme écrit.

13

Erreur de logique (2/2)

```
// Make sure that the input is valid. For this value, valid ranges
// are 1-10 and 15-20
if ( ( input >= 1 && input <= 10) && ( input >= 15 && input <= 20 ) )
{
     // Do something for valid case
}
else
{
    // Do something for invalid case
}
```


14

Erreur de boucle (1/4)

15

Les erreurs de boucle se divisent en plusieurs sous-types différents:

- boucles infinies
- boucles hors-de-un
- boucles mal sorties

Erreur de boucle (2/4)

16

- boucle infinie

```
public class While02
{
   public static void main(String[] args)
   {
      int a;

      a = 1;

      while ( a <= 10 )            // While-statement
      {
         System.out.println(a);    // Print a
                                   // NO a++ statement !!!
      }

      System.out.println("Done");
      System.out.println("Exit: a = " + a);
   }
}
```

Erreur de boucle (3/4)

17

- boucle hors-de-un

```
anArray : Array[1..50] of Integer;
Index : Integer;
begin
Index := 50;
While ( Index >= 0 ) do
    Begin
       anArray[Index] := 0;
       Index := Index - 1;
   End;
```


Erreur de boucle (4/4)

18

- boucle mal sortie

```
int nIndex = 0;
for ( int I=0; I<kMaxIterations; ++I )
{
       while ( nIndex < 20 )
       {
            ComputeSomething( I*20 + nIndex );
            nIndex ++;
      }
}
```

Erreur conditionnelle (1/2)

- Une erreur conditionnelle est une erreur due à une instruction conditionnelle mal écrite ou mal pensée.
- Les erreurs conditionnelles peuvent être de simples incompréhensions de l'algèbre booléenne, ou des erreurs dans les conditions imbriquées.

19

Erreur conditionnelle (2/2)

```
int nOper = GetUserInput();
switch ( nOper )
{
        case kOpenFile:
            DoOpenFile();
            break;
      case kCloseFile:
           DoCloseFile();
      case kDeleteFile:
           DoDeleteFile();
           break;
}
```


20

Erreur de conversion

Une donnée dans un format est convertie de manière incorrecte dans un format différent.

Causé par:

- un processus de conversion incorrect
- les données contiennent des informations qui ne peuvent pas être converties correctement dans un autre format.
- les conversions implicites d'un type de données à un autre.

21

Processus général

22

Identifiez le problème (1/4)

Est-ce vraiment un bogue ?

Il y a deux cas de base où un problème n'est pas un bogue:

1. Les spécifications indiquent que le programme doit fonctionner de cette manière.
2. Le rapport de bogue est précédé de la phrase "Ce serait vraiment bien si…". Les demandes de modification, les améliorations, les nouvelles fonctionnalités et autres ne sont pas des bogues.

Identifiez le problème (2/4)

Pourquoi le bogue est un bogue ?

1. La spécification dit que l'application devrait fonctionner d'une façon, et le code fonctionne d'une autre.
2. Le programme fait quelque chose qui est indésirable pour toutes les parties.

Identifiez le problème (3/4)

Que devrait faire le programme au moment où le problème survient ?

Tant que vous ne savez pas quelle est la bonne réponse, vous ne pouvez vraiment pas corriger le bogue.

25

Identifiez le problème (4/4)

Qu'est-ce que le programme fait vraiment ?

Même si ce que le code est en train de faire n’est pas incorrect, cela peut conduire à un bogue.

26

Collectez des informations (1/4)

- Descriptions du problème par les utilisateurs
  - Les comptes-rendus de première main sont toujours utiles.
  - Assurez-vous de noter exactement ce qu'on vous dit. De cette façon, vous pouvez comparer plusieurs rapports du même problème et rechercher des similitudes.
  - Parce que le problème réside souvent dans une région complètement différente du code, assurez-vous que les personnes décrivent tout ce qu'elles ont fait d'aussi loin que possible.

27

Collectez des informations (2/4)

- Fichiers journaux

Au minimum, une entrée de fichier journal doit contenir les informations suivantes pour être utile:

- Date et heure de l'événement
- Fonction, méthode et nom du processus générant l'événement
- Niveau de gravité de l'événement
- Une description du problème
- Toute valeur variable pertinente présente au moment de l'événement

Une règle de base simple pour les entrées de journal: Soyez détaillé pour les erreurs et concis pour les événements normaux.

28

Collectez des informations (3/4)

29

- Changements récents

  Les changements les plus récents sont par définition ceux les moins bien testés.
  - Les changements de dernière minute sont souvent ceux qui sont insérés pour ajouter des fonctionnalités supplémentaires qui ne figureraient pas initialement dans cette version ou pour résoudre un problème détecté juste avant le blocage de la version.

Collectez des informations (4/4)

30

- Informations sur l'environnement d'exécution

  Gardez une trace de l'environnement et des autres éléments externes de la machine sur laquelle votre programme s'exécute.
  - Les modifications de la mémoire, de l'espace disque, de la version du système d'exploitation et d'autres programmes en cours d'exécution peuvent facilement entraîner de nouveaux problèmes dans votre application.

Formulez une hypothèse (1/2)

- Utilisez des techniques de traçage pour surveiller la sortie du programme.
- Éliminer du code pour formuler une hypothèse.
  - Commentez ou supprimez des sections de code jusqu'à ce que vous trouviez le plus petit ensemble de code possible qui reproduit le problème.

Formulez une hypothèse (2/2)

- En général, les bogues appartiennent à l'une des catégories suivantes:
  - Erreurs de mémoire
  - Erreurs de codage
  - Erreurs de logique
  - Erreurs de configuration
  - Echec de la vérification des conditions d'erreur
  - Erreurs de chronométrage / multithread
  - Erreurs générées en externe.

Testez votre hypothèse (1/2)

- Pour tester correctement le problème, vous devez faire trois choses:
  1. Vérifier que ce que vous pensez est en train de se passer.
  2. Prouver que la base de votre hypothèse est la cause première du problème.
  3. Prédire ce qui devrait se passer compte tenu de votre hypothèse et d'un ensemble différent de données.

33

Testez votre hypothèse (2/2)

- Ne supposez pas simplement que les choses vont marcher.
- Ne supposez pas que le symptôme d'un problème est la cause première.
- Ne jamais réparer quelque chose en traitant les symptômes ou en ne comprenant pas la véritable cause du problème.

34

Itérer jusqu'à ce qu'une hypothèse prouvée soit trouvée

- Itérez jusqu'à ce que vous arriviez à une solution qui réussisse les tests et explique tous les problèmes observés

  À un moment donné, vous constaterez que vous avez une confiance extrême dans la source du problème et que toutes les observations appuient votre conclusion.

35

Proposez une solution

- La solution proposée devrait couvrir trois domaines:
  1. Elle doit expliquer pourquoi le changement résoudra le problème.
  2. Elle doit afficher les zones concernées du code source de l'application si elles se trouvent en dehors de la source d'origine du problème.
  3. Elle doit donner l'assurance que le traitement ne sera pas pire que la maladie, ce qui signifie que cela ne causera pas de problèmes supplémentaires.

36

Testez la solution

- Créez un scénario de test reproductible et vérifiez que votre solution résout le problème.
- Ce processus comporte deux étapes:
  1. Vérifier que les étapes que vous avez découvertes pour reproduire le problème sont suivies pour voir si le bogue existe toujours.
  2. Essayer d'autres méthodes pour provoquer le même problème.

37

Test de régression

- Le système complet doit être testé pour vérifier qu'il n'y a pas de nouveaux problèmes causés par la modification apportée au système pour le correctif.
- Un script automatisé devrait tester les problèmes les plus courants.

38

39

Techniques de débogage

40

Débogage intrusif vs non intrusif (1/2)

- Les techniques de débogage intrusives sont généralement celles qui impliquent des modifications du système ou de l'environnement pour l'application au moment de l'exécution.
- Une technique de débogage non intrusive ne nécessite aucune modification du code source ou de l'environnement dans lequel le système s'exécute.

Débogage intrusif vs non intrusif (2/2)

- Pour déterminer si une technique est intrusive ou non intrusive, posez-vous les trois questions suivantes:

1. La technique nécessite-t-elle des modifications du code source qui ne seraient pas autrement présentes ?
2. La technique modifie-t-elle l'environnement sous lequel le programme s'exécute normalement ?
3. Est-ce que la technique introduit l'enregistrement, des écrans de diagnostic spéciaux ou des terminaisons anormales, ce qui ne se produirait pas pendant le fonctionnement normal du système?

41

Court terme vs long terme (1/2)

- Les techniques à long terme ont pour but de trouver des problèmes alors que le système fonctionne normalement sur une période assez longue.
- Les techniques à court terme sont des approches utilisées sur une courte période.
- En général, les techniques à court terme sont plus intrusives que les techniques à long terme.

42

Court terme vs long terme (2/2)

- Les techniques à court terme ne sont normalement utilisées que lorsque vous pouvez déboguer l'application dans un environnement de développement plutôt que dans un environnement de production.
- Pour déterminer si quelque chose est une technique à long terme ou à court terme, posez-vous la question suivante:

  Le système peut-il fonctionner "normalement" avec la technique en place?

Compromis pour la production (1/2)

- Vous ne pourrez probablement pas installer votre logiciel directement; au lieu de cela, vous devrez compter sur les opérateurs pour faire le travail à votre place.
- Vous devrez vous fier aux responsables des opérations pour surveiller et livrer les fichiers de journalisation pour vous.

Compromis pour la production (2/2)

- Si vous travaillez dans un environnement client/serveur, vous ne pourrez pas vous séparer du reste de la population. Vous serez obligé d'effectuer le débogage avec plusieurs utilisateurs en ligne.
- Si vous expédiez un produit autonome, un compromis possible consiste à expédier une version de débogage spécialisée uniquement à un petit ensemble d'utilisateurs "bêta". Cela peut entraîner un meilleur retour d'informations et ne pas déranger les départements marketing et support client.