Débogage

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Lecture, écriture
- Instructions conditionnelles et itératives
- Fonctions
- Variables, tableaux, collections

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Lecture, écriture
- Instructions conditionnelles et itératives
- Fonctions
- Variables, tableaux, collections

Pourquoi aller plus loin ?

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Lecture, écriture
- Instructions conditionnelles et itératives
- Fonctions
- Variables, tableaux, collections

Pourquoi aller plus loin ?

Passage à l'échelle !

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Lecture, écriture
- Instructions conditionnelles et itératives
- Fonctions
- Variables, tableaux, collections

Pourquoi aller plus loin ?

Passage à l'échelle !

Écrire des programmes corrects

A. Corriger les erreurs : le débogage

Exemple

debogage.cpp

```
/** Teste si mot est un palindrome
 * @param mot une chaîne de charactères
 * @result un booléen
 **/
bool estPalindrome(string mot) {
    int n = mot.size()
    bool result = true;
    for ( int i = 0; i < n/2; i++ ) {
        if ( mot[i]  != mot[n-i] ) {
            result = false;
        } else {
    result = true;}
    }
    return result;
}
```

A. 1. Débogage, selon le type d'erreur

Erreurs de syntaxe

Détectées par le compilateur

A. 1. Débogage, selon le type d'erreur

Erreurs de syntaxe

Détectées par le compilateur

Débogage

1. Bien lire les messages d'erreurs
2. Le compilateur pointe vers là où il détecte l'erreur
   Pas forcément là où est l'erreur

Erreurs à l'exécution

- Segmentation fault !
- Exceptions

Erreurs à l'exécution

- Segmentation fault !
- Exceptions

Débogage

- Analyser l'état du programme juste avant l'erreur
- En Python, Java, ... : regarder la pile d'appel
- Utilisation du débogeur !

Erreurs sémantiques

- Le programme s'exécute « normalement »
  mais le résultat est incorrect
- Le programme ne fait pas ce que le programmeur souhaitait

Erreurs sémantiques

- Le programme s'exécute « normalement »
  mais le résultat est incorrect
- Le programme ne fait pas ce que le programmeur souhaitait
- Le programme fait ce que le programmeur lui a demandé !

Erreurs sémantiques

– Le programme s'exécute « normalement »
  mais le résultat est incorrect
– Le programme ne fait pas ce que le programmeur souhaitait
– Le programme fait ce que le programmeur lui a demandé !

Difficulté

Isoler une erreur glissée dans :

– Un programme de millions de lignes
– De grosses données
– Des milliards d'instructions exécutées

Erreurs sémantiques

– Le programme s'exécute « normalement »
  mais le résultat est incorrect
– Le programme ne fait pas ce que le programmeur souhaitait
– Le programme fait ce que le programmeur lui a demandé !

Difficulté

Isoler une erreur glissée dans :

– Un programme de millions de lignes
– De grosses données
– Des milliards d'instructions exécutées

Un travail de **détective !**

– Peut être très frustrant, surtout sous stress
– Peut être une très belle source de satisfaction

Erreurs sémantiques

– Le programme s'exécute « normalement »
  mais le résultat est incorrect
– Le programme ne fait pas ce que le programmeur souhaitait
– Le programme fait ce que le programmeur lui a demandé !

Difficulté

Isoler une erreur glissée dans :

– Un programme de millions de lignes
– De grosses données
– Des milliards d'instructions exécutées

Un travail de **détective !**

– Peut être très frustrant, surtout sous stress
– Peut être une très belle source de satisfaction
– Gérer ses émotions, et celle de son équipe ...

Stratégie : le débogage pas à pas

Un outil essentiel : le débogueur

- Observation du programme en cours de fonctionnement
- Exécution pas à pas :
  - En passant à la ligne suivante (next)
  - En rentrant dans les sous fonctions (step)

Stratégie : le débogage pas à pas

Un outil essentiel : le débogueur

- Observation du programme en cours de fonctionnement
- Exécution pas à pas :
  - En passant à la ligne suivante (next)
  - En rentrant dans les sous fonctions (step)
- Points d'arrêts (conditionnels)
- Analyse de la pile d'exécution
- Analyse de l'état de la mémoire

Stratégie : le débogage pas à pas

Un outil essentiel : le débogueur

- Observation du programme en cours de fonctionnement
- Exécution pas à pas :
  - En passant à la ligne suivante (next)
  - En rentrant dans les sous fonctions (step)
- Points d'arrêts (conditionnels)
- Analyse de la pile d'exécution
- Analyse de l'état de la mémoire

Débogueur `gdb` et `Code::Blocks`

- En arrière plan `gdb` : GNU DeBugger
- `Code::Blocks` rend l'utilisation du débogueur facile
- Il n'est disponible que si l'on a créé un projet !

Utilisation du débogueur pas à pas : résumé

– Compiler avec l'option `-g` :

```
g++ -g monprogramme.cpp -o monprogramme
```

Utilisation du débogueur pas à pas : résumé

- Compiler avec l'option `-g` :

```
g++ -g monprogramme.cpp -o monprogramme
```

- Lancer le débogueur avec :

```
gdb --tui monprogramme
```

Utilisation du débogueur pas à pas : résumé

– Compiler avec l'option `-g` :

```
g++ -g monprogramme.cpp -o monprogramme
```

– Lancer le débogueur avec :

```
gdb --tui monprogramme
```

– Commandes du débogueur (raccourcis : `s`, `n`, `p expr`, `q`, ...)

```
start          // lance le programme en pas à pas
step           // instruction suivante; rentre dans les fonctions
next           // instruction suivante
print expr     // affiche la valeur de l'expression
quit           // quitte
```

– Commandes plus avancées ♣

```
display expr // affiche la valeur de l'expression (persistant)
continue     // continue l'exécution du programme
show locals  // affiche les variables locales
where full   // affiche la pile d'appel
help         // aide en ligne
```

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !
2. Faire une expérience pour déterminer dans quelle « moitié » du programme est l'erreur.

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !
2. Faire une expérience pour déterminer dans quelle « moitié » du programme est l'erreur.
3. Trouver le plus petit exemple incorrect
   En faire un test !

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !
2. Faire une expérience pour déterminer dans quelle « moitié » du programme est l'erreur.
3. Trouver le plus petit exemple incorrect
   En faire un test !
4. Exécuter pas à pas la fonction sur cet exemple

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !
2. Faire une expérience pour déterminer dans quelle « moitié » du programme est l'erreur.
3. Trouver le plus petit exemple incorrect
   En faire un test !
4. Exécuter pas à pas la fonction sur cet exemple
5. Trouver l'erreur

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !
2. Faire une expérience pour déterminer dans quelle « moitié » du programme est l'erreur.
3. Trouver le plus petit exemple incorrect
   En faire un test !
4. Exécuter pas à pas la fonction sur cet exemple
5. Trouver l'erreur
6. Corriger l'erreur

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !
2. Faire une expérience pour déterminer dans quelle « moitié » du programme est l'erreur.
3. Trouver le plus petit exemple incorrect
   En faire un test !
4. Exécuter pas à pas la fonction sur cet exemple
5. Trouver l'erreur
6. Corriger l'erreur
7. Vérifier les tests (non régression)

Stratégie : réduire le problème par dichotomie

1. Caractériser le bogue : « lorsque j'appelle telle fonction avec tels paramètres, la réponse est incorrecte »
   En faire un test !
2. Faire une expérience pour déterminer dans quelle « moitié » du programme est l'erreur.
3. Trouver le plus petit exemple incorrect
   En faire un test !
4. Exécuter pas à pas la fonction sur cet exemple
5. Trouver l'erreur
6. Corriger l'erreur
7. Vérifier les tests (non régression)
8. Rajouter des tests ?

Être efficace dans la boucle essai-erreur ! ! !

Gagner du temps : développement piloté par les tests

`http://fr.wikipedia.org/wiki/Test_Driven_Development`

Durant le développement

Pour ajouter une nouvelle fonctionnalité :

- Écrire les spécifications (typiquement sous forme de javadoc !)
- Écrire le test correspondant
- Attention aux cas particuliers !
- Le développement est terminé lorsque les tests passent

Gagner du temps : développement piloté par les tests

`http://fr.wikipedia.org/wiki/Test_Driven_Development`

Durant le développement

Pour ajouter une nouvelle fonctionnalité :

- Écrire les spécifications (typiquement sous forme de javadoc !)
- Écrire le test correspondant
- Attention aux cas particuliers !
- Le développement est terminé lorsque les tests passent

Durant le débogage

Pour corriger un bogue signalé :

- Écrire un test qui met en évidence le bogue
- Le débogage est terminé quand les tests passent

B. Tests : pour aller plus loin ♣

Approche qualité traditionnelle

- Équipes très hiérarchisées :
  - Analystes programmeurs
  - Équipe de test
  - Développeur

B. Tests : pour aller plus loin ♣

Approche qualité traditionnelle

- Équipes très hiérarchisées :
  - Analystes programmeurs
  - Équipe de test
  - Développeur

- Procédures strictes (voire lourdes)

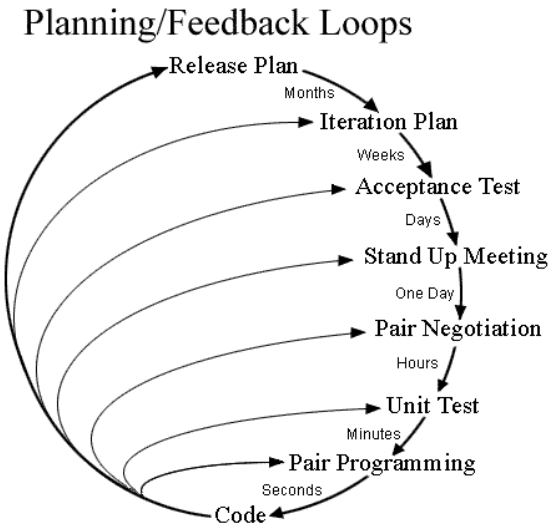
B. Tests : pour aller plus loin ♣

Approche qualité traditionnelle

- Équipes très hiérarchisées :
  - Analystes programmeurs
  - Équipe de test
  - Développeur
- Procédures strictes (voire lourdes)
- Évolution lente, planifiée longuement à l'avance

Méthodes agiles ♣

Exemple : http://fr.wikipedia.org/wiki/Extreme_programming

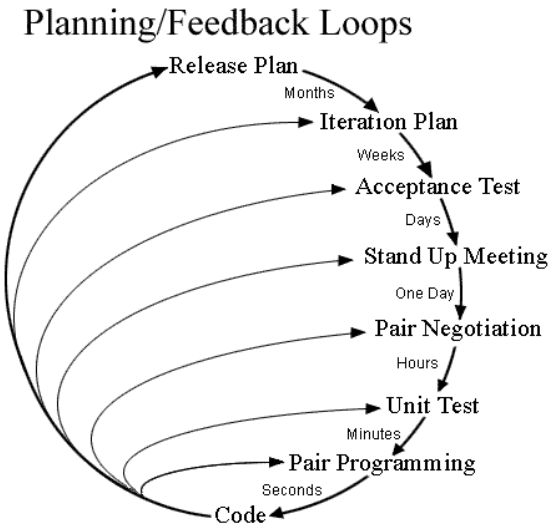


Objectif : remettre le développeur au coeur du processus

- Créativité
- Responsabilité
- Auto assurance : Tests !

Méthodes agiles ♣

Exemple : `http://fr.wikipedia.org/wiki/Extreme_programming`



Objectif : remettre le développeur au coeur du processus

- Créativité
- Responsabilité
- Auto assurance : Tests !

Les tests sont votre outil de libération !

B. 1. Les objectifs et types de tests ♣

– Mesurer la qualité du code

B. 1. Les objectifs et types de tests ♣

- Mesurer la qualité du code
- Mesurer les progrès

B. 1. Les objectifs et types de tests ♣

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`

B. 1. Les objectifs et types de tests ♣

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`
- Garantir la robustesse d'une brique avant d'empiler dessus
  Tests unitaires

B. 1. Les objectifs et types de tests ♣

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`
- Garantir la robustesse d'une brique avant d'empiler dessus
  Tests unitaires
- Anticiper la mise en commun de plusieurs briques
  Tests d'intégration

B. 1. Les objectifs et types de tests ♣

- Mesurer la qualité du code
- Mesurer les progrès
- Formaliser (et anticiper !) les demandes du client
  Tests fonctionnels
  Cahier de recette
  `http://fr.wikipedia.org/wiki/Recette_%28informatique%29`
- Garantir la robustesse d'une brique avant d'empiler dessus
  Tests unitaires
- Anticiper la mise en commun de plusieurs briques
  Tests d'intégration
- Anticiper la mise en production
  Tests de charge
- Alerte immédiate si l'on introduit un bogue
  Tests de non régression

B. 2. Il faut aller plus loin ! ♣

- Article « infecté par les tests »
  http://junit.sourceforge.net/doc/testinfected/testing.htm
- http://fr.wikibooks.org/wiki/Introduction_au_test_logiciel

- Tests unitaires en Java :
  http://junit.org/
- Tests unitaires en C++ :
  http://cppunit.sourceforge.net/doc/cvs/cppunit_cookbook.html

Résumé

Stratégies de déboggage

- Réduire le problème
- Dévelopement incrémental :
  ne jamais trop s'éloigner d'une version qui marche
- Débogueur

Résumé

Stratégies de déboggage

- Réduire le problème
- Dévelopement incrémental :
  ne jamais trop s'éloigner d'une version qui marche
- Débogueur

Tests

- Pour du code robuste
- Pour éviter ou simplifier le débogage
- Pour être serein

Résumé

Stratégies de déboggage

- Réduire le problème
- Dévelopement incrémental :
  ne jamais trop s'éloigner d'une version qui marche
- Débogueur

Tests

- Pour du code robuste
- Pour éviter ou simplifier le débogage
- Pour être serein

Tests automatique

- `ASSERT( f( ... ) == ... );`
- Pour être efficace