Programmation Orientée Projet

Méthodes de travail pour le développement du projet

6. Exercices (solution)

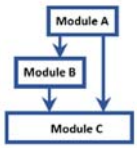
Les questions de a) à f) portent sur le cas général (*dans le monde réel*) tandis que les questions g) portent sur la mise en œuvre de notre projet avec ses adaptations à un cours de niveau introductif.

a) Spécifications

- les spécifications contiennent-elles une description des éléments suivants ?
  - o les actions que doit réaliser le programme [OUI]
  - o la manière de tester si ces actions sont correctement effectuées [OUI]
  - o les structures de données à utiliser [NON]
  - o les algorithmes à employer [NON]
  - o la décomposition du projet en modules [NON]
- Dans l'approche de développement par *prototypage* est-il encore nécessaire de distinguer les six étapes du développement ?
  - o *cf intro section 3* [OUI]

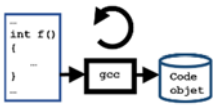
b) Analyse

- Quelle est la différence entre *l'interface* et *l'implémentation* d'un module ?
  - o *L'interface décrit le BUT du module en exportant essentiellement les prototypes d'un ensemble de fonctions dans le fichier en-tête (.h). L'interface est destinée aux personnes qui vont utiliser le module.*
  - o *L'implémentation décrit le COMMENT avec la définition détaillée de ces fonctions et de tout autre fonction et éléments nécessaires au bon fonctionnement du module. L'implémentation est réservée à la personne qui conçoit le module. Elle est responsable de son bon fonctionnement conformément au « contrat » (c'est-à-dire l'interface) qui la lie aux utilisateurs.*
- Lequel des deux éléments précédents doit être précisé dans cette phase ?
  - o *L'interface car on se concentre sur les buts des modules, leurs responsabilités, sans entrer dans les détails de l'écriture des fonctions : définir leur prototype est suffisant à ce stade.*
- Qu'est-ce que *l'architecture* du projet ?
  - o *C'est une représentation de haut niveau qui montre seulement les relations de dépendances entre les modules. Une dépendance entre un module A et un module B veut dire en général qu'il existe une fonction de A qui appelle une fonction de B.*
- A-t-on besoin du compilateur pour cette étape ?
  - o *NON, c'est une étape papier-crayon-discussions.*
- Pourquoi dit-on que l'interface d'un module est un *contrat* ?
  - o *Car elle contient un ensemble de fonctions dont le but est documenté (avec d'éventuelles conditions d'utilisations, domaine de validité des paramètres, etc…) et la personne qui offre cette interface s'engage à ce que ces fonctions soient conformes à cette description de bon fonctionnement.*



c) Codage

- Dans un module A, peut-on écrire des appels à une fonction définie dans un module B et compiler pour obtenir le code objet du module A ?
  - *OUI du moment qu'on a inclus le fichier en-tête du module B le compilateur dispose de la syntaxe des prototypes de fonctions et peut vérifier qu'elles sont correctement appelées.*



- Quand est-il recommandé de compiler :
  - Quand le module est totalement écrit ? *[NON]*
  - Chaque fois qu'une fonction est totalement écrite ? *[NON]*
  - Pour chaque groupe de quelques lignes de codes ? *[OUI]*
  - A chaque nouvelle ligne de code ?
    - *risque de prendre plus de temps que l'option précédente*
- L'étape du codage détecte-t-elle les erreurs syntaxiques ou sémantiques ?
  - *Les erreurs syntaxiques puisqu'on travaille seulement à l'étape de compilation pour produire le code objet ; on ne peut pas produire d'exécutable à ce niveau (étape suivante).*

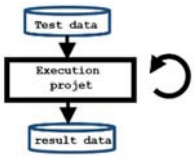
d) Tests Unitaires

- Puis-je tester l'exécution du Module A s'il dépend de fonctions du module B et que ce module B n'est pas encore finalisé et testé ?
  - *OUI pour autant qu'une version minimale du module B a été implémentée à l'aide de stubs (section 4.3). En gros les fonctions du module B sont vides ou presque mais ça permet quand même de faire des tests utiles sur le module A.*
- Comment définiriez-vous un test utile ?
  - *C'est un test qui caractérise une classe de scénario d'utilisation du programme qui n'est pas déjà couvert par un autre test ; cette classe de comportement du programme suit un chemin d'exécution différent des autres classes déjà testées.*

- Y a-t-il un ordre particulier à respecter pour effectuer les test unitaires ?
  - *Pas particulièrement, on peut commencer par tester les modules de bas niveau ou de haut-niveau en premier ou le faire en parallèle par les membres d'un groupe.*
- Ce travail peut-il être effectué indépendamment et en parallèle par différentes personnes pour tous les modules ?
  - *OUI grâce aux stubs. On est déjà plus ou moins dans la phase suivante de l'intégration quand on utilise les modules déjà testés.*

e) Intégration et Vérification

- Y a-t-il un ordre particulier à respecter pour intégrer les modules ?
  - *Pas particulièrement, on peut commencer par intégrer les modules de bas niveau ou de haut-niveau en premier.*
- Pourquoi dois-je refaire tous les tests à chaque étape de l'intégration ?
  - *Parce qu'on ne peut pas exclure de refaire le même bug ultérieurement dans une autre partie du code. Il faut donc continuellement s'assurer qu'un bug précis n'est pas présent en faisant exécuter le programme sur un test bien défini pour lequel on connait le résultat attendu.*



f) Validation

- Comment se pourrait-il que la validation donne un résultat différent de la vérification ?
  - *Simplement si le client fait des tests qui couvrent des classes de comportements non couverts par les tests de l'étape de vérification.*

g) Travail en groupe de deux personnes

- Quelles sont les étapes du développement où il faut travailler ensemble et celles où on peut travailler indépendamment de l'autre membre du groupe ?
  - *Ensemble : Analyse et intégration*
  - *Indépendamment : Codage et Tests unitaires*

- Identifier les faiblesses possibles de l'étape d'analyse pouvant induire des pertes de temps à l'étape du codage.
  - *Si les responsabilités des modules sont insuffisamment claires, deux modules pourraient faire plus ou moins la même chose (= insuffisante séparation des fonctionnalités)*

  - *si les échanges de données entre modules ne sont pas assez bien identifiés (input/output des fonctions) alors l'interface est instable car les prototypes de fonctions doivent être redéfinis fréquemment. Cela nuit au travail indépendant pour le codage et le test unitaire.*

  - *Défaut fréquent : faire faire à un module de niveau supérieur ce que peut faire un module de niveau inférieur. Le module de niveau inférieur dispose d'un accès facile aux données dont il a la responsabilité ; il peut donc faire des calculs de manière efficace avec du code léger. Cependant on observe souvent le défaut de mettre à disposition beaucoup de fonctions get() pour exporter les données et faire ces même calculs à un niveau supérieur, ce qui est plus lent et bien moins clair à cause des fonctions get().*

- En supposant que les deux membres d'un groupe se sont répartis le codage de deux modules A et B ; dans quels cas une personne est-elle cliente de l'autre personne ? Quel contrat les lie ?
  - *La personne du module A est cliente du module B si elle inclut l'interface du module B pour appeler une de ses fonctions. Le contrat qui les lie est l'interface du module B qui décrit le but et les conditions d'utilisation des fonctions qui sont exportées par le module B.*

- Dans quel contexte est-il recommandé d'effectuer l'étape du codage ensemble ? Pourquoi ?
  - *En cas de recherche de bug on est plus efficace à deux personnes (section 5). Une seconde paire d'yeux est un atout précieux, de même que la verbalisation de la description du bug.*

(les réponses aux questions avancées sont disponibles dans un autre document)