

# Cours d'Informatique de Base

## Travaux Pratiques

### Applications des Techniques d'Inspection

### Document de Règles pour l'élaboration de

### Spécification

Année Académique 2000-2001

Ce document décrit un certain nombre de règles à suivre lors de l'écriture de spécifications conceptuelles de programmes. Les règles sont réparties en deux groupes :

- des règles de complétude et cohérence du produit par rapport aux sources ;
- des règles consuisant à la qualité du produit lui-même.

Il faut noter qu'en aucun cas, ces règles ne peuvent être considérées comme une ensemble de règles complet, suffisant ou prescriptif. Des questions autres que celles soulevées par l'examen de ces règles pourraient survenir lors d'une inspection.

## 1 Complétude et Cohérence par Rapport aux Sources

Les règles énoncées ci-dessous permettent de déterminer si la spécification reprend tous les éléments des documents sources et si elle n'est pas en contradiction avec ces sources.

Les documents source pour écrire une spécification conceptuelle sont l'énoncé informel du problème et la conceptualisation et structuration de ce problème.

**Règle 1.1** *Tout module identifié dans la conceptualisation/structuration est décrit par une section du produit (un rapport de spécification d'un groupe).*

**Règle 1.2** *Toute fonction définie dans la conceptualisation/structuration est spécifiée dans le produit. Toutes les données entrant et sortant de la fonction correspondent à des paramètres de la fonction correspondante dans le produit.*

**Règle 1.3** *Toute connaissance explicite de l'énoncé pertinente pour la spécification de niveau conceptuel doit être présente dans la spécification (produit). Il ne faut pas laisser de connaissance implicite. Dans la mesure du possible, la spécification seule devrait pouvoir servir de base pour l'implémentation.*

**Règle 1.4** *La spécification ne doit pas être en contradiction avec les sources.*

## 2 Qualité du Produit

Les règles suivantes permettent de déterminer si le produit lui-même est de bonne qualité en vérifiant :

- que tous les éléments qui doivent être définis le sont ;
- que tous les éléments utilisés sont suffisamment (complètement) définis ;
- que les éléments définis le sont correctement.

Les règles sont groupées en fonction du concept principal sur lequel elles portent (fonction, type de données, assertions). Une dernière règle concerne l'absence de surspécification. Elle est suivie de quelques suggestions générales qui permettent d'améliorer la qualité des spécifications. Celles-ci peuvent être mentionnées lors de la réunion de révision pour des points spécifiques.

### 2.1 Fonctions

**Règle 2.1** *Toute fonction doit définir les “objets utilisés” (types de données logiques “globaux”).*

**Règle 2.2** *Toute fonction doit définir les paramètres qu'elle reçoit en entrée.*

**Règle 2.3** *Toute fonction doit définir les paramètres qu'elle produit en sortie.*

**Règle 2.4** *Toute fonction doit définir les paramètres qu'elle modifie.*

**Règle 2.5** *Toute fonction doit définir une précondition.*

**Règle 2.6** *Toute fonction doit définir une postcondition.*

**Règle 2.7** *Tout paramètre d'une fonction doit être défini par un nom (de préférence court mais significatif) et un type.*

**Règle 2.8** *Tout objet utilisé par une fonction doit être défini par un nom, un type et éventuellement des invariants et des opérations (suivant l'approche objet).*

**Règle 2.9** *Les noms de tous les objets utilisés et paramètres d'une fonction doivent être distincts.*

**Règle 2.10** *Tout paramètre produit ou modifié par une fonction apparaît dans la postcondition de cette fonction. Une assertion exprime comment sa valeur dérive des paramètres en entrée et des objets utilisés. Lorsque le type du paramètre est complexe (se décompose en éléments de types plus simples), on veillera à définir la façon dont chacun de ses éléments sont “calculés”.*

**Règle 2.11** *Tout paramètre reçu ou objet utilisé par une fonction apparaît soit dans la précondition, soit dans la postcondition.*

**Règle 2.12** *Toute variable utilisée dans la précondition ou la postcondition d'une fonction est soit un objet utilisé par cette fonction, un paramètre reçu, produit ou modifié par cette fonction ou une variable intermédiaire.*

**Règle 2.13** *La précondition d'une fonction ne peut porter que sur des paramètres reçus ou des objets utilisés.*

## 2.2 Types de Données

**Règle 2.14** *Tout type de donnée utilisé doit être soit simple (entier, caractère, string, booléen,...), soit défini en termes de types plus simples (record, séquence, ensemble, ...). Un dessin ou une formulation textuelle peuvent être utilisés, mais on évitera les formulations ambiguës.*

**Règle 2.15** *La spécification conceptuelle ne doit pas contenir de concepts propres au langage de programmation. On veille à y utiliser des types de données logiques (ensembles, séquences, files, piles, ...) et non des structures de données "physiques" (listes chaînées, tableaux, fichiers, ...). On parlera par exemple d'objets utilisés (globaux) plutôt que de fichiers.*

## 2.3 Assertions

**Règle 2.16** *Toute variable utilisée dans une assertion est définie (a un nom et un type. Elle est soit un objet utilisé, un paramètre de la fonction ou une variable intermédiaire.*

**Règle 2.17** *Une assertion ne doit pas être ambiguë. Elle doit pouvoir être lue et comprise par quelqu'un qui n'a pas connaissance du ou des documents source(s). Par exemple, on ne pourra pas simplement énoncer "ref1 est similaire à ref2 en fonction des paramètres de similitude". La notion de similitude est suffisamment complexe que pour justifier une définition précise et non ambiguë.*

**Règle 2.18** *Les assertions doivent être syntaxiquement cohérentes avec les structures de données (types) utilisées. Par exemple, si l'on utilise un ensemble, on pourra parler d'appartenance. Si on utilise une séquence (liste), on parlera d'un élément présent à une position donnée (en utilisant un indice). Dans un ensemble, par exemple, il n'y a pas d'ordre ni de doubles.*

## 2.4 Absence de Sur-spécification

**Règle 2.19** *La spécification conceptuelle ne doit pas contenir d'information de séquençement ni d'algorithme (tests, boucles, ...). Elle doit exprimer quel est le problème et non pas comment on va le résoudre. Elle ne doit pas contenir d'informations d'optimisation. Des choix de représentation seront fait plus tard et tiendront compte des besoins de performance.*

## 2.5 Conseils Généraux

**Règle 2.20** *On peut utiliser des prédicats qui correspondent à des sous-fonctions pour spécifier une fonction qui a été décomposée. Cependant, ceux-ci doivent alors être clairement définis.*

**Règle 2.21** *Il est conseillé d'associer une description en français aux assertions compliquées ainsi qu'à certaines données complexes (paramètres, objets utilisés, types), en plus de la description précise.*