ScribeQuality Documentation

Version 0.4.2

megaplanet

01 December 2015

Table des matières

Ce site définit différentes règles de qualité pouvant être utilisées tout au long du cycle de vie du logiciel. Les règles sont rangées par paquetage, chaque paquetage. Chaque paquetage correspond soit à une étape du cycle de vie, spoit à un aspect transversal. Certains paquetages correspondent à des règles vérifiées par des outils existants tel que Modelio ou StarUML.

Attention : Ce site ne contient que la définition de règles de qualité. Certains outils sont présentés dans le projet ScribeTools.

Astuce : Utiliser la fonction “Search” pour utiliser ce site.

Les paquetages de règles ci-dessous correspondent à différentes étapes dans le cycle de vie du logiciel.

CHAPITRE 1

Collecte des besoins

1.1 Glossaire

1.1.1 NomenclatureGlossaire

Le nom des glossaires doit être en style MajMin (voir *MajMin*).

meta Glossary.name

type OK

paquetage *Glossaire*

1.1.2 NomTerme

Le nom d'un terme doit être au singulier s'il s'agit d'un nom et doit correspondre si possible au terme le plus au terme utilisé dans le contexte correspondant au glossaire.

exemple PointDAcces, Piece, Vehicule, VehiculeAccidente

meta Term.name

type OK

paquetage *Glossaire*

1.1.3 NomenclatureTerme

Le nom d'un terme doit être en style MajMin (voir *MajMin*).

commentaire Cette convention peut être fort utile pour faire ressortir dans un texte l'utilisation des termes définis dans un glossaire et donc pour renforcer le fait que ce terme à été utilisé de manière consciente et raisonnée.

exemple VehiculeAccidente

meta Term.name

type OK

paquetage *Glossaire*

1.1.4 TermeTropCompose

Le nom du terme est composé de plusieurs mots ou sous-termes mais certains de ceux-ci semblent ne pas être pertinents ou nécessaires dans la composition totale. Il est préférable de les enlever pour rester à des termes essentiels.

exemple Dans "AjouterDansPanier" le terme essentiel est clairement "Panier", mais le composant "AjouterDans" semble superflu. Elle l'est en tout cas si la notion d'ajout à laquelle tout un chacun peut penser est différente du concept référencé par "AjouterDansPanier". Dans le contexte d'un système de contrôle d'accès "BatimentAAccesControle" pourrait certainement être simplifié en "Batiment" car dans ce contexte si les batiments auxquels on fait référence sont toujours ce type de batiment. C'est évidemment le cas dans une définition comme celle-ci : "BatimentAAccesControle : Bâtiment appartenant à une [Zone] nécessitant des [DroitDAcces]s pour y pénétrer."

commentaire Dans l'exemple "AjouterDansPanier", il est probable qu'une confusion existe entre d'une part le nom du terme et d'autre par le nom d'une exigence, ou d'un cas d'utilisation. Ces derniers résultent naturellement de la composition de verbes (plus ou moins généraux, et pouvant donc être dans certains cas définis dans un glossaire) et de formes nominales définies dans des glossaires.

meta Term.name

type KO

paquetage *Glossaire*

1.1.5 TermeFlou

Le terme correspond à une notion floue ou subjective dans le domaine considéré ou la définition associée au terme est trop floue ou subjective pour pouvoir être exploitable. S'il s'agit d'un terme général définir ce terme n'est peut être pas nécessaire, ou au contraire il s'agit peut être d'une notion importante pour lequel un terme plus précis devra être trouvé.

exemple Dans la définition suivante le terme "Mecanisme" est très flou, le terme "Adéquat" est subjectif, et la définition ne permet pas de clarifier ces aspects : "MecanismeAdequat : Un mécanisme adéquat permet de vérifier qu'une seule personne passe à la fois.". Dans ce cas il est sans doute important de trouver un terme plus précis permettant de caractériser cet élément qui semble important pour le fonctionnement du système. Par contre dans la définition suivante le terme est non seulement flou mais sans doute inutilement défini car trop général : "Information : Ensemble des messages circulant dans le [Systeme]". Ce terme peut certainement être supprimé.

meta Term.name

type KO

paquetage *Glossaire*

1.1.6 DefinitionTerme

La définition d'un terme doit être relativement courte et concise et écrite dans un style similaire à celui que l'on pourrait trouver dans un dictionnaire. Généralement une telle définition commence par une forme nominale définissant la nature du terme. Ce n'est pas une phrase avec un verbe.

exemple Si un verbe est défini une définition pourrait commencer par "action de ...". S'il s'agit d'un participe passé, la définition pourrait commercer par "état ...". S'il s'agit d'un concept ou d'un objet, celui-ci est catégorisé par rapport à une taxonomie supérieur. Par exemple une "fourchette" pourrait être défini comme "ustensile permettant ...".

meta Term.definition

type OK

paquetage *Glossaire*

1.1.7 DefinitionTermeAQuestions

De part les zone d'ombres qu'elle comporte la définition d'un terme pose un certain nombre de questions alors qu'une définition devrait uniquement apporter des réponses.

exemple Considérons la définition suivante : "Identifiant : Clé qui permet d'identifier de manière unique une [information]". Dans cette définition la notion de 'cle' est sans doute beaucoup plus obscure pour des non-informatitions que la notion d'identifiant et il est donc préférable soit d'éliminer cette définition (voir *TermeFlou*), soit de la reformuler.

commentaire L'objectif même des glossaires et de répondre à toutes les questions terminologiques. Il est donc indispensable de ne pas utiliser ni paraphrases inutiles (voir *Paraphrase*) ni termes qui posent plus de questions qu'ils n'apportent de réponses. En cas de difficulté pour définir un terme, le recours à des exemples est tout à fait conseillé.

meta Term.definition

type KO

paquetage *Glossaire*

1.1.8 DefinitionAmbigueTerme

La définition associée au terme semble ambigue ou fait référence à différents sens. Une signification unique et précise doit être donnée.

commentaire Dans un dictionnaire plusieurs significations sont traditionnellement associées à un terme, car la pluspart des termes sont polysémiques. Dans un glossaire, on cherche au contraire à éviter les ambiguités et à indiquer de manière explicite quelle est la signification retenue dans le contexte associé à l'utilisation du glossaire. Un glossaire est un vocabulaire contrôlé.

meta Term.definition

type KO

paquetage *Glossaire*

1.1.9 DefinitionTermeTropGenerale

La définition proposée pour un terme est trop générale par rapport au contexte associé au glossaire dans lequel le terme est défini.

meta Term.definition

type KO

paquetage *Glossaire*

1.1.10 TermeAGlossaire

Un ou des termes devraient être ajoutés dans l'un des glossaires dans la mesure où s'agit d'un terme spécifique ou d'un concept important.

meta Term.definition ; ...

type KO

paquetage *Glossaire*

1.1.11 ClassificationTerme

Le terme dans lequel le glossaire apparait n'est pas le plus approprié.

exemple Le trigramme associé à un membre de l'équipe projet devra figurer dans le glossaire du projet et non pas dans le glossaire du logiciel.

meta Term-Glossaire

type KO

paquetage *Glossaire*

1.1.12 TermesAlternatifs

Différents termes alternatifs peuvent être associés si néccessaire à un terme. Ces differentes formes alternatives peuvent soit correspondre à des déclinaisons linguistiques (par exemple le passage d'un substantif à un verbe, etc), soit à des termes perçus comme synonymes dans le contexte du glossaire considéré.

commentaire Il n'est pas nécessaire d'introduire des alternatives que si celles-ci sont effectivement utilisées dans le contexte considéré. Par ailleurs il ne faut pas confondre (1) d'une part les termes alternatifs à qui ont associe la même signfication que le terme principal et (2) les exemples qui eux sont des termes, des expressions, des artefacts ou des concepts plus spécifiques.

exemple Dans le contexte d'un système de contrôle d'accès, un terme principal pourrait être "PorteurDeBadge" avec comme termes alternatifs "PossesseurDeBadge", "PersonneABadge", "Badgeur". Si le système permet de définir des types arbitraires de "PorteurDeBadge" les termes suivants sont alors naturellement simplement des exemples "Etudiant", "PersonnelAdministratif", "Technicien", etc.

meta Term.alternatives

type OK

paquetage *Glossaire*

1.1.13 ReferenceVersTerme

Une ou plusieurs expressions correspondent à des termes dans le glossaire (ou à des synonymes de ces termes) et devraient donc être remplacée(s) par une référence vers ce terme (principal) (voir *FormatReferenceTerme*).

exemple Dans la phrase "Le [ChefDAtelier] renseigne dans CyberGarage le temps de réparation pris par un mécanicien pour le véhicule", les termes "[CyberGarage]", "[TempsDeReparation]", "[Mecanicien]", "[Vehicule]" devraient être référencés si ceux-ci sont dans un glossaire, ou sinon, ils devraient être sans doute introduits dans le glossaire (cf $)

meta

type KO

paquetage *Glossaire*

1.1.14 FormatReferenceTerme

Lorsqu'un terme défini dans un glossaire est utilisé dans un texte une référence vers ce terme doit être créé sous la forme du terme tel que défini dans le glossaire et entre crochets ([]). Dans le cas de termes au pluriel la marque du pluriel suivra immédiatement la référence. Les cas particuliers pourront être traités grace aux "alternatives" associés à un terme dans un glossaire.

exemple "Les [Terme]s sont dans des [GlossairePredefini]s mais ce n'est qu'un [Exemple]."

meta Term.definition ; ...

type OK

paquetage *Glossaire*

1.1.15 ReferenceTermePrincipal

Les références à des termes du glossaire doivent référencer le terme principal plutôt que ses alternatives.

type OK

paquetage *Glossaire*

1.1.16 ReferenceTermeInconnu

Un terme est référencé mais n'est défini dans aucun glossaire.

type KO

paquetage *Glossaire*

1.1.17 DefinitionMultipleTerme

Un terme semble être défini plusieurs fois dans le même glossaire, (1) soit parcequ'il s'agit du même nom ou d'une déclinaison du même nom, (2) soit parceque les définitions associées aux deux temes sont si proches qu'il semble que les deux termes sont en fait des synonymes. Dans les deux cas, la solution semble être soit de fusionner les termes et leur définitions, doit de clarifier explicitement la définition de chacun des termes.

commentaire L'objectif d'un glossaire est de définir les termes de manière non ambigüe, en tout cas dans le cadre d'un glossaire et il est donc nécessaire de n'avoir qu'une seule définition, par terme. Evidemment si deux termes sont "fusionnés", l'un prendre certainement le rôle de termes alternatifs.

type KO

paquetage *Glossaire*

1.1.18 TermesCroises

Les definitions des termes dans un glossaire doivent faire référence aux autres termes de ce glossaire ou d'autres glossaires.

type OK

paquetage *Glossaire*

1.2 Exigence

1.2.1 NomExigence

Le nom de l'exigence doit faire clairement référence à une exigence ; le type de cette exigence doit si possible transparaître dans le nom ; le nom doit autant que possible faire référence à des termes définis dans les glossaires.

commentaire Il est généralement préférable de donner aux exigences un nom plutôt qu'un numéro car le nom est significatif. Par ailleurs utiliser un numéro implique de garder un "compteur" pour s'assurer qu'un numéro ne sera pas réutilisé.

type OK

paquetage *Exigence*

1.2.2 NomExigenceFonctionnelle

Le nom d'une exigence fonctionnelle doit débuter par un verbe à l'infinitif. Cette règle est cohérente avec la règle correspondante pour les cas d'utilisation (voir *NomCU*).

commentaire Cette règle permet de reflêter clairement qu'une exigence fonctionnelle correspond à une fonction devant pouvoir être exécutée par un acteur en utilisant le système.

exemple "InscrireUneEquipe"

type OK

paquetage *Exigence*

1.2.3 NomenclatureExigence

Le nom d'un exigence doit être en style MajMin (voir *MajMin*).

type OK

paquetage *Exigence*

1.2.4 DefinitionExigence

Le définition d'une exigence doit énoncer de manière claire et concise une contrainte imposée sur le système à développer ou sur le processus de développement de ce système. La définition doit se limiter à l'expression de cette contrainte. Une exigences ne doit pas entre autre décrire un scénario, une suite d'actions, une caractéristique liée à l'exigence, des restrictions ou détails techniques non pertinents, des actions internes réalisées par le système et sans rapport avec les objectifs des parties prenantes, etc. Certaines de ces informations peuvent être utiles dans certains cas, mais dans ce cas il faut les consigner dans une ou des notes associées à l'exigence.

exemple La phrase suivante "L'[EquipeTechniqueGaragis]" ayant une expérience de [Struts], il serait préférable d'utiliser [Struts] dans ce projet.". Cette phrase donne lieu à la définition d'exigence "DeveloppementStruts : [CyberGarage] doit être développé avec le framework [Struts]" avec la note indiquant la motivation suivante "Contexte : L'[EquipeTechniqueGaragis]" possède une expérience de [Struts]". Noter par ailleurs que la priorité associé à la forme modale "il serait préférable" a été extraite de la définition (cf ! ! !PrioritéExigence).

type OK

paquetage *Exigence*

1.2.5 DefinitionExigenceFonctionnelle

Sachant qu'une exigence fonctionnelle correspond à une fonctionnalité du système destinées à un ou plusieurs acteurs, la définition d'une telle exigence peut être rédigée sous la forme "[SSS] doit permettre à [AAA] de ..." où [AAA] est le nom du système, [AAA] le nom de l'acteur ou des acteurs et ... définit la fonctionnalité proposée. La partie "[SSS] doit permettre à" peut être éliminée si il est absolument clair que [AAA] est un acteur et que [SSS] est le système dont on parle.

exemple "[CyberGarage] doit permettre au [ChefDeMagazin] d'enregistrer les [Piece]s qu'il fourni aux [Mecanicien]s lorsque ceux-ci lui demande".

commentaire La première partie faisant intervenir le nom du système explicitement n'est pas obligatoire mais elle permet de rendre explicite le fait que le système réalise la fonction. Cela permet d'éliminer les phrases ambigues où le rôle du système n'est pas explicité. Par exemple la phrase suivante ne permet pas de savoir quel est le rôle exacte du système dans le processus décrit, et ainsi on ne peut pas vérifier qu'il s'agit d'un exigence fonctionnelle : "Le [ChefDeMagazin] fourni les [Piece]s aux [Mecanicien]s lorsque ceux-ci lui demande".

type OK

paquetage *Exigence*

1.2.6 ExigencesMultiples

Le texte fait référence à plusieurs exigences simultanément et/ou les descriptions de ces exigences devraient être séparées. Cette séparation peut être nécessaire par exemple pour clairement identifier le type de chaque sous-exigence, pour attribuer à chacune de ces sous-exigences des propriétés différentes, par exemple des priorités différentes, etc.

commentaire La définition d'une exigence doit être généralement courte et concise. De muliples lignes dans une exigences ou l'utilisation de connecteurs (et, ou, " ;") peuvent facilement mener à des problèmes d'exigences multiples. Une seule phrase peut également correspondre à des exigences multiples. C'est le cas par exemple si l'on fait à la fois référence à ce que doit faire le système et que c'est l'objectif d'une partie de la phrase, et qu'une autre partie consiste à donner des indications de performances par exemple.

exemple

type KO

paquetage *Exigence*

1.2.7 ExigenceIncoherente

L'exigence est incohérente avec une autre exigence décrite avant ou après.

type KO

paquetage *Exigence*

1.2.8 ExigenceInvalide

L'exigence n'est pas ou ne semble pas être valide par rapport aux besoins exprimés par le client.

type KO

paquetage *Exigence*

1.2.9 SurExgigence

La description de l'exigence comporte un ou des éléments plus restrictifs que ceux exprimés par le client ou certaines contraintes exprimées ne semblent pas strictement nécessaires.

type KO

paquetage *Exigence*

1.2.10 SousExigence

L'exigence décrite n'est ne semble pas suffisemment restrictive par rapport à l'expression des besoins exprimées par le client ou par rapport à une situation jugée réaliste.

type KO

paquetage *Exigence*

1.2.11 TypeDExigence

Le type de l'exigence n'est pas correct ou la phrase contient différentes exigences de types différents (voir *Exigences-Multiples*).

type KO

paquetage *Exigence*

1.2.12 PrioriteExigence

La priorite associée à une exigence doit être clairement exprimée et ce séparemment de la définition de l'exigence qui elle doit être rédigée de manière neutre par rapport à cet aspect.

commentaire Une des difficultés concernant les priorités est que celles-ci doivent toujours être considérées les unes par rappot aux autres, et de plus les priorités doivent pouvoir être ajustées au cours d'un projet. La définition d'une exigence ne doit pas comporter des formes modales tels que "devrait", "Il serait souhaitable que", "On souhaite que", etc. La définition doit au contraire exprimer la contrainte sur le système de manière impérative, la priorité faisant office de modulation. Cette séparation des préoccupations est importante en pratique car cela permet (1) d'avoir en un endroit clairement localisé et dumment codifié la liste des priorités et (2) de pouvoir changer si nécessaire ces priorités sans avoir à reformuler le texte des exigences.

exemple La définition "DeveloppementJDBC : Il est serait utile que l'interface [JDBC] soit utilisée pour l'accès à la base de données" devra être réécrit "L'interface [JDBC] doit être utilisée pour l'éccès à la base de données" en indiquant dans l'attribut priorité la priorité correspondante après concertation éventuelle avec le client.

type KO

paquetage *Exigence*

1.2.13 ProprieteExigenceInadaptee

La valeur de la propriété associée à l'exigence semble inadaptée.

type KO

paquetage *Exigence*

CHAPITRE 2

Modelisation

2.1 Systeme

2.1.1 NomSysteme

Les noms des systèmes et des sous-systèmes doivent clairement reflêter leur rôle et/ou la décomposition réalisée, ne doivent pas être générique, et doivent montrer leur status de systèmes.

exemple "Systeme" est à éviter car ce nom est trop générique. "Batiment" n'est pas adapté comme nom de sous-système car ce terme fait référence à un système physique. "GestionDesBatiments" ou "SystemeDeGestionDesBatiments" sont mieux adaptés.

paquetage *Systeme*

2.1.2 NomenclatureSysteme

Les noms de système et sous-systèmes doivent être en style MajMin (voir *MajMin*).

exemple "GestionDesIncidents"

paquetage *Systeme*

2.1.3 DecompositionSousSysteme

La décomposition en termes de sous -ystèmes ne semble pas adéquate, pas équilibrée et/ou pas justifiée.

paquetage *Systeme*

2.1.4 SurDecomposition

La décomposition en sous-systèmes fait apparaître trop de sous-systèmes sans pour autant que ceux-ci semblent justifiés et/ou il serait peut être pertinent de les regrouper en sous-systèmes plus "gros", quitte éventuellement à réaliser une décomposition hiérarchique.

paquetage *Systeme*

2.1.5 LimiteDuSysteme

Les limites du systeme ne sont pas clairement identifiées et/ou il n'est pas clairement établi quel est le rôle exact du système dans la situation décrite.

paquetage *Systeme*

2.2 CasDUtilisation

2.2.1 NomActeur

Le nom d'un acteur doit être (1) une forme nominale, (2) un terme métier défini dans le glossaire (voir *NomCUGlossaire*), et (3) ne pas être trop générique (par exemple "Utilisateur" et "Acteur" sont à éviter).

commentaire La notion d'acteur est définie par le *rôle* joué par l'acteur par rapport au système et non pas par la *position* de l'acteur dans l'organisation.

exemple "SpectateurDistant", "Superviseur" sont des noms potentiels d'acteurs. "Utilisateur" ou "Acteur" sont trop génériques dans la mesure ou toutes les personnes potentiellement intéragissant avec le système sont des "utilisateurs" de ce système. "Directeur" pourrait correspondre à une position dans une entreprise ; ce n'est pas forcemment un bon nom de role selon les cas.

meta Actor.name

paquetage *CasDUtilisation*

2.2.2 NomenclatureActeur

Le nom d'un acteur doit être en style MajMin (voir *MajMin*).

exemple "SpectateurDistant"

meta Actor.name

paquetage *CasDUtilisation*

2.2.3 NomActeurGlossaire

Les termes importants utilisés dans le nom d'un acteur doivent être définis dans le glossaire.

commentaire Généralement il est utile de faire figurer le terme entier correspondant à l'acteur dans le glossaire. En effet il est souhaitable de définir au plus tôt les termes associés à ce type de rôle.

exemple L'acteur "SpectateurDistant" donnera lieu sans doute au terme "SpectateurDistant" dans un glossaire, mais aussi peut être à "Spectateur" et éventuellement "Distant" si ces termes font du sens dans d'autres contextes et ne correspondent pas à des notions triviales.

meta Actor.name

paquetage *CasDUtilisation*

2.2.4 NomActeurInstancie

Les noms des personnes jouant le role d'acteur doivent dans des scénarios instanciés doivent être à la fois particuliers pour être mémotechniques mais aussi représenter la diversité culturelle associée au contexte du système et du projet associé.

exemple "ahmed", "marie", "bob" sont des noms d'acteurs instanciés valides. "mrPropre" ou "babar" sont à proscrire car il donne une image enfantine et peu professionnelle du projet.

meta Instance.name

paquetage *CasDUtilisation*

2.2.5 NomenclatureActeurInstancie

Le nom d'un acteur instancié doit être en style minMaj (voir *MinMaj*)

commentaire Cette convention est liées au fait qu'il s'agit d'instances alors que les éléments du niveau "classes" commencent par une majuscule.

exemple "ahmed"

meta Instance.name

paquetage *CasDUtilisation*

2.2.6 DescriptionActeur

Chaque acteur doit être décrit en précisant des informations telles que (1) sa position dans l'organisation ou les organisations dans lequel le système est déployé, (2) l'importance éventuelle de cet acteur par rapport au projet, ou à l'utilisation du système, (3) des éléments de volumétrie indiquant des ordres de grandeurs concernant le nombre de personnes pouvant jouer ce rôle dans le contexte de différentes installation du système, (4) des caractéristiques éventuelles supplémentaires sur les tranches d'ages, d'handicap éventuels, etc.

commentaire En pratique ces informations sont fondamentales car c'est de tels éléments entre autre qui servent à définir des priorités, des caractéristiques non fonctionnelles concernant les interfaces, etc.

meta Instance.Description.content

paquetage *CasDUtilisation*

2.2.7 NomCU

Le nom des cas d'utilisation doivent correspondre à des formes verbales simples, représentant explicitement la fonctionalité que l'acteur principal désire réaliser au moyen du système, sachant que l'acteur principal jouera le role de sujet dans cette forme verbale. Le nom du cas d'utilisation doit clairement faire référence à un but ($ActeurSujet).

exemple "DeclarerLEntreeDUnVehicule" est valide. "EntreeVehicule" n'est pas valide car ce n'est pas une phrase verbale.

meta UseCase.name

paquetage *CasDUtilisation*

2.2.8 NomCUGlossaire

Les termes utilisés dans le nom d'un cas d'utilisation doivent être définis dans le glossaire, en tout cas pour les termes principaux et ceux dont l'interprétation pourrait poser un problème. Si une abbréviation est utilisée celle-ci devra être impérativement définie dans le glossaire.

exemple Si l'on considère le cas d'utilisation "DeclarerLEntreeDUnVehicule" il faudra s'assurer que "Vehicule" et peut être "EntreeDUnVehicule" ou "Entree" soient définis dans le glossaire. Si nécessaire on pourrait également définir "Declaration" mais le nom complet "DeclarerLEntreeDUnVehicule" sera défini de toute façon via la description de ce cas d'utilisation.

meta UseCase.name

paquetage *CasDUtilisation*

2.2.9 NomenclatureCU

Le nom des cas d'utilisation doivent être en MajMin (voir *MajMin*).

commentaire les cas d'utilisation correspondent à des classes de scenarii et il est donc logique d'utiliser la même convention que pour les classes à savoir l'utilisation d'une majuscule en début de nom.

exemple "DeclarerLEntreeDUnVehicule"

meta UseCase.name

paquetage *CasDUtilisation*

2.2.10 ActeurSujetCU

Le nom de l'acteur principal associé à un cas d'utilisation est le sujet de la forme verbale correspondant au nom du cas d'utilisation.

exemple "AcheterUnBillet" peut avoir comme acteur "Client" car la phrase "un client achète un billet" correspond à une des fontionalités que doit délivrer le système. Par contre "ControlerAccesUtilisateur" et "Utilisateur" ne forment pas une combinaison valide car l'utilisateur n'est pas le sujet de cette forme verbale. Il y a ici une confusion entre ce que faire le système et l'objectif de l'acteur. Un cas d'utilisation doit correspondre à un but de l'acteur (voir *ButCU*), par exemple "EntrerDansUneZone".

meta Actor-UseCase

paquetage *CasDUtilisation*

2.2.11 AuMoinsUnActeur

Chaque cas d'utilisation doit être associé à au moins un acteur.

meta Actor-UseCase

paquetage *CasDUtilisation*

2.2.12 AuMoinsUnCU

Au moins un cas d'utilisation doit être associé à chaque acteur.

commentaire Si un acteur n'utilise aucun cas d'utilisation, alors il ne s'agit pas d'un acteur. Un acteur doit nécessairement être impliqué dans une interaction directe au moins avec un système et ces interactions sont modélisées par les cas d'utilisations. Dans le cadre d'UML uniquement les interactions directes sont modélisées et prises en compte.

exemple "Vigile" n'est pas un acteur d'un système de contrôle d'accès à un batiment si cet celui-ci se limite à surveiller le batiment mais n'interagit jamais avec le système.

meta Actor-UseCase

paquetage *CasDUtilisation*

2.2.13 ImplicationSystemeCU

Le système doit être impliqué dans tous cas d'utilisation, sachant qu'un cas d'utilisation représente par définition une suite d'interactions entre le système et le (ou les) acteur(s).

exemple "AppelerPompiers" n'est pas un cas d'utilisation si cette action se fait via un téléphone ou tout être élément externe au système.

paquetage *CasDUtilisation*

2.2.14 ImplicationActeurCU

L'acteur doit être impliqué dans chaque cas d'utilsation avec lequel il est relié car un cas d'utilisation représente par définition une suite d'interactions entre le système et un acteur (au moins). Si aucune interaction n'a lieu entre le système et un acteur, alors il ne peut y avoir de cas d'utilisation.

exemple Un cas d'utilisation nommé "GarderHistorique" implique qu'un acteur demande par exemple que la sauvegarde se fasse ou que l'acteur soit notifié de cette sauvegarde. Si ce n'est pas le cas, il ne s'agit sans doute pas d'un cas d'utilisation.

paquetage *CasDUtilisation*

2.2.15 ButCU

Un ou plusieurs cas d'utilisation ne correspondent pas à un but de l'acteur principal ou ne sont pas nommés pour refléter cet aspect. Un cas d'utilisation doit correspondre à un objectif "métier" de l'acteur principal et les différentes interactions que ce dernier entreprent avec le système dans ce contexte doivent lui premettre de réaliser un but ultime. Si le métier le veux le cas d'utilisation peut correspondre à la réalisation d'un but intermediaire, et ce afin d'accomoder la règle d'unité de temps (voir *UniteDeTempsCU*) et d'espace , mais la notion de but reste néanmoins valide.

commentaire Cette règle s'applique dans le cas standard où les cas d'utilisation ne sont pas utilisé comme élément de modélisation dans des modèles détaillés de cas d'utilisation. C'est la règle recommandée. Notons que le but ultime associé au cas d'utilisation n'est pas forcément réalisé dans les cas de scenarii d'erreurs, mais il doit l'étre dans les différents scenarii positifs. Le nom du cas d'utilisation correspond normallement au but visé et non pas à la méthode employée.

exemple "EnregistrerEntrer", "SIdentifier", "EntrerPendantLesHeuresDOuvertures", "TaperSonCode" ne sont pas des noms valides de cas d'utilisation. Par contre "RetirerDeLArgent" ou "Entrer" sont valides car ils décrivent clairement le but visé par l'utilisateur.

meta UseCase

paquetage *CasDUtilisation*

2.2.16 UniteDeTempsCU

Les cas d'utilisations doivent correspondre à des "unités de temps" en ce qui concerne les interactions entre un acteur et le système.

commentaire Généralement un cas d'utilisation dans un sytème interactif n'excéde pas la notion de "session" qui correspond à une unité de temps maximale. Plusieurs cas d'utilisation peuvent avoir lieu dans la même "session" par exemple si l'acteur désire réaliser plusieurs buts avec le système.

exemple Dans le cas d'un système d'achat de billets sportifs, s'il est possible d'annuler son billet après l'achat et la transaction terminée (par exemple en se reconnectant au système) quelques jours après alors "AcheterUnBillet" et "AnnulerUnBillet" seront deux cas d'utilisation séparés.

meta UseCase

paquetage *CasDUtilisation*

2.2.17 RelationCU

Pas de relation entre acteurs sauf éventuellement une spécialisation.

paquetage *CasDUtilisation*

2.2.18 HeritageActeur

Un acteur spécifique peut réaliser tous les CU de l'acteur qu'il spécialise.

paquetage *CasDUtilisation*

2.2.19 SousTypageActeur

Un acteur spécifique est un cas particulier de l'acteur qu'il spécialise.

paquetage *CasDUtilisation*

2.2.20 RepresentationActeurNonHumain

Dans un diagramme de cas d'utilisations les acteurs non humains doivent être représentés graphiquement sous forme de rectangle avec le stéréotype <<Actor>> plutot que sous forme de bonhomme.

commentaire Cette règle est utile dans le cas de réunions par exemple avec des personnes non expertes en UML. L'utilisation de "bonhomme" pour des systèmes peut perturber inutilement l'auditoire et le lecteur. De plus cela permet visuellement de bien faire la différence entre les acteurs humains et les acteurs non humains.

commentaire Avec le logiciel Modelio cela peut se faire en sélectionnant l'acteur et en utilisant l'onglet `Symbol` puis `Representation mode > Structured`.

paquetage *CasDUtilisation*

2.2.21 PasDeRelationEntreCU

L'utilisation de relations entre cas d'utilisation n'est recommandée.

paquetage *CasDUtilisation*

2.2.22 RelationsCUIncoherentes

Les relations de dépendences <<includes>> et <<extends>> existant entre cas d'utilisations ne sont pas cohérentes avec les descriptions détaillées de ceux-ci

paquetage *CasDUtilisation*

2.2.23 IncludeMultiple

Un cas d'utilisation inclu via une relation dépendence <<includes>> doit l'être dans au moins deux cas d'utilisation.

paquetage *CasDUtilisation*

2.2.24 CUAuxiliaireDecore

Dans le cadre du **StyleCUDecore**, le stéréotype <<auxiliary>> doit être associé aux acteurs auxillaires.

style StyleCUDecore

paquetage *CasDUtilisation*

2.2.25 StyleCUEssentiel

Dans le cadre du **StyleCUEssentiel** la description du scenario ne doit pas faire de références inutiles à la manière dont les acteurs et le système intéragissent dans le détail, sachant que l'objectif d'un *cas d'utilisation essentiel* n'est pas de décrire des exigences sur une ou des interfaces personnes systèmes.

style StyleCUEssentiel

paquetage *CasDUtilisation*

2.2.26 CUPrimaireAGauche

Dans le cadre du **StyleCUGaucheDroite** les acteurs primaires doivent être représentés à gauche du système, les acteurs secondaires à droite.

style StyleCUGaucheDroite

paquetage *CasDUtilisation*

2.2.27 CUSeulementPrimaire

Dans le cadre du **StyleCUGaucheDroite** seuls les acteurs primaires doivent être representés dans les diagrammes de cas d'utilisation.

paquetage *CasDUtilisation*

2.3 Scenario

2.3.1 NomScenario

Chaque scenario doit être nommé et le nom d'un scénario doit faire référence (1) au cas d'utilisation qu'il réalise et (2) à la (ou aux) caractéristique(s) principale(s) de ce scénario qui le différentie des autres scénarios. Si ce n'est pas possible un numéro pourra être associé au nom de scénario. Dans tous les cas un résumé décrira le contenu ou l'intention du scénario (voir *IntentionScenario*).

exemple "cloreDossier_Normal" et "cloreDossier_AnnulationClient" sont deux scénarii correspondants clairement au même cas d'utilisation "CloreDossier". Le premier scénario correspond au scénario dit "nominal". Si de nombreux scénarii devaient être associés au même cas d'utilisation et s'il est difficile de leur donner un nom court on alors choisir des noms du style "cloreDossier_S1", "cloreDossier_S2", ... "cloreDossier_S12" par exemple.

paquetage *Scenario*

2.3.2 NomenclatureScenario

Le nom d'un scenario doit être en style MinMajSouligne (voir *MinMajSouligne*).

commentaire Les scénarii devant être référencés par plusieurs autres éléments de modèles il est utile de nommer de manière précise les scénarii. Comme un scénarii est au niveau "instance", le style minMajSougligne est recommandé et ce par opposition au style MajMin (voir *MajMin*) recommandé pour les Cas d'Utilisation (voir *NomenclatureCU*). L'utilisation du souligné peut être utile pour séparer le nom du cas d'utilisation du qualificatif correspondant au scénario.

exemple "cloreDossier_DechargeClient"

paquetage *Scenario*

2.3.3 NomScenarioGlossaire

Les termes importants utilisés dans le nom des scénarii doivent être définis dans le glossaire.

exemple Dans “cloreDossier_DechargeClient” il est peut être nécessaire de définir les termes suivants ou certains de ces termes : “Decharge” ou “DechargeClient” et/ou “Client” selon les cas.

paquetage *Scenario*

2.3.4 NomScenarioInstancie

Le nom d’un scénario instancié doit faire autant que possible référence aux instances considérées dans le scénarios notamment à l’acteur instancié ou aux jeux de données considérées. Si trop d’information sont à décrire, il peut être préférable de numéroter les scénarii (voir *NomScenario*) et de définir leur contenu via le résumé du scénario (voir *IntentionScenario*).

paquetage *Scenario*

2.3.5 IntentionScenario

La description détaillée d’un scénario sous forme d’une séquence d’actions doit être précédée d’un résumé décrivant l’intention associée à ce scénario. Ce “résumé” doit principalement (1) décrire l’intention du scénario, (2) positionner celui-ci par rapport aux autres scénarii correspondant au même cas d’utilisation, (3) introduire éventuellement les données et instances essentielles référencées dans le scénario.

paquetage *Scenario*

2.3.6 SequenceDActions

La description du scénario doit se faire strictement sous forme d’une séquence d’actions avec une seule action par ligne (voir *FormatAction*).

exemple Les actions suivantes peuvent former une séquence valide (une action par ligne) “* [Mamadou] introduit sa [carteBancaire13] dans le [distributeur637] de [cyberBanqueDeLorient]”, “* [cyberBanqueDeLorient] affiche l’[ecran17] sur le [distributeur637]”, “* [Mamadou] introduit son code 7878”, etc.

paquetage *Scenario*

2.3.7 FormatAction

Une action doit être décrite sous forme d’une ligne de texte commençant par un asterisque (“*”)

commentaire “* [paul] s’identifie auprès de [cyberCompetition].” doit être suivie d’un saut de ligne.

paquetage *Scenario*

2.3.8 SujetAction

Le sujet d’une action apparaissant dans un scénario doit (1) soit être le système (2) soit un des acteurs intervenants dans le cas d’utilisation. Dans tous les cas il doit être explicitement identifié.

commentaire S’il s’agit d’une action intervenant dans une description de scénario instancié il s’agira impérativement d’une instance (voir *SujetActionInstancie*).

commentaire Cela veut dire qu’une action prend soit la forme “* [unSystemeInstancie] ... ” soit la forme “* [unActeurInstancie] ... “

exemple "* [cyberKebabLondres] affiche l'[ecran112] demandant à [bree] de valider sa [commande1223]"

exemple "* [bree] valide la [commande1223] gràce à [cyberKebabLondres]"

paquetage *Scenario*

2.3.9 SujetActionInstancie

Le sujet d'une action doit correspondre (1) soit à un acteur instancié, (2) soit à un système instancié. De plus il doit faire référence à un élément de modèles défini par ailleurs.

commentaire Dans un scenario instancié il est important d'instancier les acteurs et le système dans la mesure ou ces scénarii doivent être aussi concrets que possible pour pouvoir être validés par les différents intervenants. Par ailleurs, donner référencer des acteurs ou systèmes instanciés permet de décrire les caractéristiques de ces derniers plus en détails et par exemple de définir leur profil utilisateur lorsqu'il s'agit d'acteurs humains. Faire référence à un système instancié permet également de situer le scénario dans un contexte plus précis, en prenant en compte par exemple l'état du système instancié (qui pourrait en effet correspondre à un état particulier). Un tel degré de précision peu se réveler fort utile dans le cadre de l'élaboration de tests à partir

exemple "Le système" devrait être remplacé par "cyberBatimentIMAG" si le système que l'on considère dans le scénario instancié correspond à l'instanciation du système CyberBatiment. Pour être plus précis, CyberBatiment est vu comme une classe de système pouvant être instancié (installé, configuré, etc.) dans différents contextes. Chaque instance de ce même système va maintenir un état, une configuration, etc, qui va être différente et les mêmes actions sur ces différentes instances de systèmes vont donc potentiellement donner des résultats différents.

paquetage *Scenario*

2.3.10 DebutSujetAction

Le sujet d'une action doit débuter la phrase décrivant cette action.

exemple "* [bree] valide la [commande1223] gràce à [cyberKebabLondres]"

paquetage *Scenario*

2.3.11 IntermediaireAction

La ou les actions doivent être reformulées de manière à ce que le sujet de l'action soit clairement identifié (cf $SujetAction et $SujetActionInstancie) même si des intermediaires peuvent figurer dans l'action à titre d'illustration et/ou pour donner des détails quand aux interactions concretes entres les acteurs et le systeme.

exemple Dans la phrase d'action "[paul] passe son [badge210] dans le [lecteurDeBadge214]" le système de controle d'acces n'est pas représenté de manière explicite, alors que il est le destinataire du message dans un scenario externe. Le lecteurDeBadge214 joue simplement le rôle d'intermediaire, ou plus précisemment d'interface entre l'acteur et les éléments internes du systèmes. Si la description de ces éléments d'interfaces sont utiles, la phrase d'action devrait être reformulée de la manière suivante par exemple "[paul] s'identifie auprès du [systemeDeControleIMAG] via son [badge210] qu'il passe devant le [lecteurDeBadge214]". Ici badge210 et lecteurDeBadge214 sont des intermediaires dans l'interaction entre paul et systemeDeControleIMAG. De manière plus abstraite, et si l'on veut faire abstraction de ces interfaces, on pourrait dire "[paul] s'identifie auprès du [systemeDeControleIMAG]".

paquetage *Scenario*

2.3.12 ActionAtomique

Certaines descriptions d'actions font références implicitement ou explicitement (via des connecteurs "et" par exemple) à plusieurs actions atomiques qui devraient décomposées.

commentaire Séparer ces actions permet une meilleure traçabilité entre les différents modèles, par exemple entre les scénarii décrits textuellement et les diagrammes de séquences ou de communication.

exemple

paquetage *Scenario*

2.3.13 ActionConcrete

L'action ou les actions ne sont pas décrites de manières suffisemment concrètes, soit en terme des moyens utilisés pour les interactions, soit en termes d'informations échangées.

commentaire Les scénarii, notemment les scénarii instanciés doivent contenir suffisemment d'information pour pouvoir par la suite générer automatiquement ou manuellement des tests. Si des données précises ne sont pas fournies lors de l'écriture des scénarii, un travail supplémentaire devra être fait par la suite. Par ailleurs il est toujours utile de donner des exemples concrètes de valeurs ou d'objets pour pouvoir valider les scénarii.

exemple

paquetage *Scenario*

2.3.14 ParametreConcret

Les paramètres des actions doivent avoir des valeurs concrétes (voir *ValeurConcrete*).

commentaire Cet aspect est particulièrement à plusieurs titres (voir *ValeurConcrete*).

paquetage *Scenario*

2.3.15 ActionMetier

La description de l'action doit faire référence à des termes métiers et ne doit pas comporter par exemple de détails techniques inutiles ou ne correspondant pas au niveau d'abstraction du scénario.

exemple "Paul demande la création d'un formulaire" n'est pas une action métier. Non seulement le métier de l'acteur ne consiste pas à "demander des formulaires", mais de plus ce genre de détails techniques contraint inutilement les choix futurs de conception ou de réalisation.

paquetage *Scenario*

2.3.16 ExempleScenario

Les scénarios instanciés doivent être basés sur des exemples réalistes et utiliser en priorité les exemples déjà existants dans le cadre du projet s'il en existe.

commentaire Il est assez classique dans des réunions, des discussions ou des documents d'utiliser des exemples précis et ces exemples doivent être utiliser autant que possible dans la mesure où ces derniers sont déjà connus par les intervenants du projet et peuvent être associés à des données ou documents existants.

exemple Dans le cadre de système de gestion de conférences, la conférence WAT2012 et le comité de programme présidé par de ralf gasevic, peut être utilisé dans les scénarios. Si des documents comme le site de la confférnce, la liste des articles acceptés, le comité d'organisation, sont disponibles il est bien évidemment très utile de se baser sur ces données comme objets dans les scénarios.

paquetage *Scenario*

2.3.17 MessageInexplique

La raison menant au déclenchement du message n'est pas facilement compréhensible ou devrait être explicitée.

paquetage *Scenario*

2.3.18 TypeDeMessage

Il n'est pas clair si le message correspond à l'invocation d'une opération ou à une valeur de retour.

commentaire Cette règle peut être appliquée dans le cas où les valeurs de retours des opérations sont modélisées par des messages.

paquetage *Scenario*

2.3.19 ValeurDeRetour

Le message devrait correspondre à une valeur de retour et non pas à l'invocation d'une opération.

commentaire Cette règle peut être appliquée dans le cas où les valeurs de retours des opérations sont modélisées par des messages.

paquetage *Scenario*

2.3.20 RetourInexplique

Il n'est pas facile de comprendre à quelle invocation d'opération ce message, qui semble correspondre à une valeur de retour, doit être associé.

commentaire Cette règle peut être appliquée dans le cas où les valeurs de retours des opérations sont modélisées par des messages.

paquetage *Scenario*

2.3.21 RetourManquant

Il n'est pas facile de comprendre quel et le retour associé à l'invocation d'une opération soit parcequ'il ne semble pas être fait mention d'un tel retour, soit parceque plusieurs messages pouvant correspondre à des retours sont des candidats potentiels.

paquetage *Scenario*

2.3.22 Responsabilites

La répartition des responsabilités entre objets n'est pas claire ou ne semble pas être logique.

commentaire Ce peut être le case par exemple lorsqu'une opération est appelée sur un objet d'une classe alors que cet objet n'a pas la responsabilité de réaliser cette fonctionalité ou d'offrir le service correspondant. Ce peut être également le cas lorsqu'un paramètre n'est pas indiqué car l'objet appelant suppose que l'objet appelé maintient la valeur de ce paramètre ou un état correspondant.

paquetage *Scenario*

2.3.23 ReferenceScenario

Le diagramme de séquence ou de communication n'est pas clairement identifié, ou si cet identificateur existe, celui-ci n'est pas en lien direct et systèmatique avec l'identificateur du scenario qu'il représente. La tracabilité entre representation graphique et textuelle des scenarios n'est pas assurée.

commentaire les diagrammes de séquences ou de communication et les représentations textuelles sont formés de suites d'actions ne sont qu'une représentation graphique alternative d'un scenario et il devrait donc y avoir le même identificateur ou la même racine d'identificateur.

paquetage *Scenario*

2.3.24 PresenceObjet

La raison de la présence de l'objet dans le diagramme n'est pas clairement explicitée, ou ne semble pas logique. Pour qu'un objet soit dans un diagramme correspondant à un scénario il doit soit être (1) préxister au scénario, (2) soit être créé dans le cadre du scénario, (3) soit correspondre à un objet retourné par une opération, (3) soit figurer comme paramêtre d'une opération. Dans le cas (3) et (4) au moins un résultat ou paramètre doit faire référence au nom de l'objet.

paquetage *Scenario*

2.4 Sequence

2.4.1 ObjetClassifie

Un ou plusieurs objets n'indiquent pas de manière satisfaisante la classe dont ils sont à l'origine.

commentaire Dans Modelio ce problème peut correspondre au fait que le champ "base" de certains objets n'a pas été renseigné correctement.

paquetage *Sequence*

2.5 Valeur

2.5.1 ResultatConcret

Il est nécessaire de donner des valeurs concrètes aux résultats (voir *ValeurConcrete*).

commentaire Cet aspect est particulièrement à plusieurs titres (voir *ValeurConcrete*).

paquetage *Valeur*

2.5.2 ValeurConcrete

Il est nécessaire d'utiliser des valeurs concrète, correspondant à des valeurs scalaires précises, à des identificateurs d'objets ou à des valeurs structurées. Les valeurs scalaires ou identificateurs d'objets peuvent être définis de manière globale et il est utile de les utiliser de manière cohérente et identique à la fois dans les descriptions textuelles et dans les diagrammes.

commentaire Plus les élements intervenants dans les scénarii sont concrets, plus les différents intervenants sont en mesure d'apprehender et de valider les éléments de modélisation. Le fait d'utiliser des formats et des identificateurs précis permet de faire référence à des éléments définis par ailleurs de manière encore plus détaillée. Ces objets et valeurs peuvent également être utilisés dans le cadre des

tests et par exemple pourront figurer par la suite dans le code source des tests. Si les conventions pour les noms d'objets sont utilisées (voir *NomObjet*) les scénarii ne perdent pas en lisibilité.

exemple Par exemple badge231 identifie certainement un objet de type Badge (voir *NomObjet*) ; la constante 2.5 est une valeur concrète de type réel ; "1728EGT" est une chaîne de caractère ; "une caillou bloquait la porte" est une valeur contrète pouvant faire sens dans un scénario.

paquetage *Valeur*

2.5.3 ParametreObjet

Un ou des paramétres prennent des valeurs scalaires alors qu'ils devrait plutôt correspondre à des objets et que des noms d'objets doivent donc être fourni (voir *NomObjet*).

exemple Badge=145 devrait être remplacé par badge145 qui correspond au nom d'un objet de type Badge qui pourrait/devrait être déclaré par ailleurs.

paquetage *Valeur*

2.5.4 AbusDeString

Une utilisation abusive du type string est faite dans la modélisation.

commentaire le typage est l'une des plus avancées les plus importantes dans l'histoire de l'informatique et l'utilisation de type string lorsqu'un type plus précis, voir un type d'objets ou de collections aurait pu être utilisés est souvent le reflet d'une modélisation de médiocre qualité ou même souvent l'absence de modélisation ou de reflexion. L'encodage d'information sous forme de chaînes de caractères doit être faite uniquement lorsque cela est strictement justifié.

paquetage *Valeur*

2.5.5 FormatValeur

Le format de la valeur semble incorrect, imprécis, incohérent ou non défini.

paquetage *Valeur*

2.5.6 TypeValeur

Il n'est pas facile d'inférer quel est le type de la valeur ou le type de valeur inféré ne semble pas être correct ou suffisemment précis.

commentaire L'utilisation de guillemets permet d'indiquer les constantes de type chaîne de caractères ; un format systématique doit être utilisé pour les constantes de type date et/ou heure (par exemple 12/02/2012 :12 :03 :00) ; les objets peuvent être nommés précisément et de manière à ce que leur identificateur soit conforme à la nomenclature (voir *NomenclatureObjet*).

exemple Il n'est pas facile de déterminer si 012 est une valeur de type entier ou s'il s'agit d'une chaîne de caractères. Par contre il est naturel de penser que bob est un objet de type personne si ce type existe dans le modèle mais que "bob" est une chaîne de caractères.

paquetage *Valeur*

2.5.7 TypeValeurIncorrect

Le type de la valeur fournie semble incorrect par rapport au type attendu par exemple par une variable, un parametre formel ou un type de résultat. Le problème peut provenir du fait que la valeur correspond par exemple au resultat d'une opération et que le nom de l'opération ne semble par cohérent avec ce type de retour.

paquetage *Valeur*

2.5.8 ValeurInexpliquee

Il n'est pas facile de comprendre ce que la valeur signifie, d'où elle provient et/ou comment elle est calculée/produite.

paquetage *Valeur*

2.5.9 ValeurConstante

TODO

commentaire Utiliser des noms symboliques pour des constantes peut être utile par exemple dans le cas de longues chaines de caractères, de messages, etc. On pourra alors utiliser le nom symbolique en lieu en place du literal dans les ses differents contexte d'usages (position de parametre, de retour, de valeur d'attribut, etc), et définir le literal à un autre endroit (sous la forme d'une note, d'un élement de modèle, d'un élément de document, etc).

paquetage *Valeur*

2.5.10 ValeurReflechie

Une ou plusieurs valeurs semblent totalement factices et ne pas résulter d'une reflexion approfondie. Des valeurs comme 123456 ou 001 reflêtent généralement l'absence de reflexion de la part d'un auteur et parfois de telles valeurs ne sont pas réalistes.

paquetage *Valeur*

2.5.11 Surcodification

L'utilisation de "codes" ne semble pas correspondre à la réalité du métier ou peut impliquer une charge cognitive inutilement élevée dans le cas d'interfaces personne systeme.

exemple Par exemple un code est demandé à un acteur dans une interaction personne système sans que cet utilisateur aie, de part ses caractéristique et celle de son rôle, l'ensemble des codes "en tête".

paquetage *Valeur*

2.5.12 UniteValeur

TODO

paquetage *Valeur*

2.5.13 CardinalVsOrdinal

TODO

paquetage *Valeur*

2.5.14 ValeurPlausible

TODO

paquetage *Valeur*

2.5.15 ValeurComposite

TODO

paquetage *Valeur*

2.5.16 ValeurCollection

TODO

paquetage *Valeur*

2.5.17 LiteralEnumeration

TODO

commentaire TODO

paquetage *Valeur*

2.6 Tache

2.6.1 NomTache

Dans un modèle de tâches, le nom des tâches doit correspondre à une forme verbale à l'infinitif et les tâches correspondant à des cas d'utilisation doivent suivre les règles correspondantes (voir *NomCU*). De plus le nom des tâches doit faire référence autant que possible aux termes définis dans le glossaire.

exemple La tâche "ReserverUnePlace" correspond bien à une forme verbable. "Place" devrait probablement être dans le glossaire. Selon les cas "Reserver" ou "ReserverUnePlace" pourrait aussi y figurer si la signification associée n'est pas claire.

paquetage *Tache*

2.6.2 NomenclatureTache

Le nom des tâches doit être en style MajMin (voir *MajMin*).

commentaire Certaines tâches correspondent à des cas d'utilisation et il est donc important d'utiliser la même règle (voir *NomenclatureCU*).

paquetage *Tache*

2.6.3 TacheComposite2

Une tâche composite doit comporter au moins deux sous-tâches.

commentaire La décomposition de tâches en sous-tâches n'a d'intérêt que si plusieurs sous tâches existent.

paquetage *Tache*

2.6.4 TacheElementaire

Une tâche élementaire ne peut pas être une tâche abstraite.

paquetage *Tache*

2.6.5 TypeTacheComposite

Une tâche composite est (1) soit abstraite, (2) soit du même type que toutes ses sous-tâches.

paquetage *Tache*

2.7 Classe

2.7.1 NomClasse

Le nom d'une classe doit correspondre à une forme nominale au singulier.

commentaire Une classe représente généralement un concept et les concepts sont généralement identifiés par des noms communs au singulier. Le nom de la classe est au singulier car il fait référence au concept et non pas à l'extension de la classe. Il s'agit là d'une différence importante avec les noms de tables des bases de données car dans ce cas il est fait références à l'extension, c'est à dire à l'ensemble des instances contenues dans la table.

paquetage *Classe*

2.7.2 NomenclatureClasse

Le nom des classes doivent être dans le style MajMin (voir *MajMin*).

paquetage *Classe*

2.7.3 NomAttribut

Le nom d'un attribut doit normallement correspondre à une forme nominale ou éventuellement à un forme verbale lorsque le type de l'attribut correspond à un booleen.

commentaire Lorsque l'attribut est de type booleen, la notion représentée correspond en générale à un prédicat et la forme grammaticale correspond généralement au fait que l'objet vérifie ou pas une propriété.

exemple "estEteinte" est un attribut de type booléen sur la classe "Lampe", "puissance" est de type entier, "interrupteurs".

paquetage *Classe*

2.7.4 NomenclatureAttribut

Le nom de ou des attributs doivent être en style minMaj (voir *MinMaj*).

paquetage *Classe*

2.7.5 NomObjet

Le nom d'un objet doit correspondre à une forme nominale et doit permettre autant que possible de déterminer le nom de la classe auquel il appartient. Il peut prendre par exemple (1) soit la forme d'un nom propre, (2) soit d'un identifiant naturel, (3) soit d'un rôle qu'il joue au sein du système ou dans le cadre d'une interaction donnée, (4) soit d'une forme derivée à partir de la classe à laquelle appartient l'objet.

exemple Par exemple (1) "ahmed" ou "paysBas" sont des noms propres faisant des objets de type "Personne" ou "Pays" par exemple. (2) "batimentIMAGC" utilise l'identifiant naturel du batiment C de l'institut IMAG. (3) "pereDeSophie" ou "gardien" ou fait référence à des personnes via leur rôles joué dans le système ou dans le cadre de collaborations particulières (4) Finalement "personne232" fait clairement référence à une personne et l'on peut supposer que le nom "p" fait référence à un objet de même type si dans le contexte direct seule la classe Personne débute par la lettre p.

paquetage *Classe*

2.7.6 NomenclatureObjet

Un nom de ou des objets doivent être en style minMaj (voir *MinMaj*).

paquetage *Classe*

2.7.7 NomOperation

Le nom d'une opération doit normallement correspondre à une forme verbale dont le "sujet" est l'objet auquel l'opération s'applique.

commentaire L'invocation d'une opération sur un objet représente une action que doit réaliser l'objet

paquetage *Classe*

2.7.8 NomenclatureOperation

Le nom de ou des operations doivent être en style minMaj (voir *MinMaj*).

paquetage *Classe*

2.7.9 NomenclatureMethode

Le nom de ou des methodes doivent être en style minMaj (voir *MinMaj*).

paquetage *Classe*

2.7.10 NomParametre

Le nom du ou des paramètres formels doivent correspondre à des formes nominales et désigner les rôles que les valeurs des paramètres vont jouer dans le cadre de l'opération ou de la méthode concernée.

commentaire les règles sont mêmes que pour nommer les objets (voir *NomObjet*) si ce n'est que les noms propres et identifiant naturels doivent être proscrits car un paramêtre formel ne correspond pas à un objet concret particulier.

paquetage *Classe*

2.7.11 NomenclatureParametre

Le nom de ou des methodes doivent être en style minMaj (voir *MinMaj*).

paquetage *Classe*

2.7.12 NomRole

Le nom d'un rôle doit normallement correspondre à une forme nominale et en tout état de cause à un rôle que peuvent jouer le ou les objets destination du rôle.

commentaire les règles et commentaires associées au nom d'attribut s'appliquent au nom des rôles (voir *NomAttribut*) si ce n'est qu'un rôle ne peut pas correspondre à un prédicat, car ne peut pas être de type booléen, et que le nom d'un rôle ne doit donc pas correspondre à une forme verbale.

paquetage *Classe*

2.7.13 NomenclatureRole

Le nom de ou des roles doivent être en style minMaj (voir *MinMaj*).

paquetage *Classe*

2.7.14 NomAssociation

Le nom de l'association doit a priori correspondre à une forme verbale ; les objets jouant le rôle de sources pour cette association jouant le rôle de "sujets" de cette forme verbale.

paquetage *Classe*

2.7.15 NomenclatureAssociation

Le nom de ou des associations devrait être en style MajMin (voir *MajMin*).

paquetage *Classe*

2.7.16 RoleClasse

Le nom d'une classe semble correspondre à un rôle ou inversement ; la modélisation pourrait être revue.

paquetage *Classe*

2.7.17 RoleAssociation

Le nom du rôle semble être interverti par rapport à un nom d'association ou vice versa.

paquetage *Classe*

2.7.18 Navigabilite

Un ou plusieurs roles portent des indications de navigabilite sans que cela semble justifié ou cohérent.

paquetage *Classe*

2.7.19 Cardinalite

Une ou plusieurs cardinalites sont manquantes, non justifiées ou erronées.

commentaire Toutes les cardinalites devraient être décrites dans un diagramme de classes. Souvent le manque de cardinalité correspond à l'absence de reflexion et ainsi à l'absence de validation du modèle.

paquetage *Classe*

2.7.20 CardinaliteInversee

Une ou plusieurs cardinalites semblent être inversées ou sinon il s'agit peut être d'erreurs de cardinalités.

commentaire Cette erreur est rencontrée de manière relativement fréquente lorsque l'auteur du modèle confond les conventions UML avec les conventions utilisées dans d'autres langages de modélisation. Généralement ce défaut est associé également à l'utilisation de constante "n", ce qui n'est pas non plus correct en UML (cf $CardinaliteNM :).

paquetage *Classe*

2.7.21 CardinaliteNM

En UML les cardinalités minimales ou maximales doivent être formées des constantes entières positives ou * comme cardinalité maximale. Alors que 0..n n'est pas correct en UML par contre 0,4-6,9-* est correct.

paquetage *Classe*

2.7.22 Composition1

Le cardinalité maximale associée à une association de composition est au maximum.

commentaire Un composant est au maximum dans un composite et la cardinalité maximale est de 1. Par contre la cardinalité minimale peut être 0 dans le cas ou plusieurs association de composition sont issues de la même classe "de composant".

paquetage *Classe*

2.7.23 CompositionUnique

Il existe à partir d'une classe "de composants" plusieurs associations de composition avec une cardinalité minimale de 1 alors que cela n'est pas possible car un objet "composant" ne peut être dans plusieurs composites à la fois. Les cardinalités minimales doivent être 0 sur toute les associations de compositions.

paquetage *Classe*

2.7.24 AggregationNonJustifiee

L'utilisation d'une ou plusieurs associations d'aggregation ne semble pas adaptée ou l'intérêt d'utiliser de telles modélisations ne semble pas pertinent sans justification explicite.

commentaire La notion d'aggrégation peut être interpretée de multiple manières et dans la pluspart des contextes il est fort probable que differents lecteurs feront des interpretations de la modélisation. Par ailleurs la différence entre une association d'aggrégation et une association normalle est parfois si tenue que cette notion n'est pas forcemment très utile ; Il est donc préférable de s'abstenir d'utiliser les symboles d'aggrégation. D'ailleurs sachant qu'aucun consensus n'a jamais pu être obtenu autour de ce concept, il a finalement été éliminé à partir de la version 2.0 d'UML. Seule la notion de composition, plus précise, consensuelle, et moins sujette à interprétation, est restée dans le standard.

paquetage *Classe*

2.8 Etat

2.8.1 NomEtat

Le nom d'un état doit faire référence explicitement à la période de temps dans lequel l'objet se trouve dans l'état.

commentaire Contrairement aux cas des noms de classes ou d'opérations qui correspondent à des catégories linguistiques caractéristiques (respectivement forme nominale et forme verbale), il n'y a pas de correspoIl n'y a pas de correspondance siml D'un point de vue linguistique cela correspond généralement à un participe passé, à une forme basée sur la réalisation future, passée ou présente d'une action (avec des préfixes tels que "EnCoursDe", "EnAttenteDe", etc.), ou a des formes en "-ing" en anglais.

exemple Par exemple un document sera dans l'état "Modifié" (participe passé), "EnCoursDeModification", ou encore "EnAttenteDeValidation".

paquetage *Etat*

2.8.2 NomenclatureEtat

TODO

paquetage *Etat*

2.8.3 NomTransition

TODO

paquetage *Etat*

2.8.4 NomenclatureTransition

TODO

paquetage *Etat*

2.8.5 NomTransitionInutile

Les noms de certaines transitions semblent inutiles, trop génériques, ou inappropriés.

commentaire Il n'est généralement pas nécessaire de nommer les transitions dans la mesure où celles-ci sont décrites intégralement par les gardes, les événements, les actions et résultats qui leur sont associés. Leur donner un nom peut éventuellement être pratique si l'on utilise des outils de transformations, ou que l'on veut référencer de manière directe une transition, mais généralement les transitions se passent de noms.

paquetage *Etat*

2.8.6 EtatInitial

L'état initial est manquant.

paquetage *Etat*

2.8.7 JustificationEtat

La presence ou l'absence d'un ou plusieurs états n'est pas justifiées ou pourrait être remise en cause.

commentaire Un état correspond normallement à une durée de temps "significative" pour l'objet ou le système et pendant laquelle le système va avoir un comportement différent par rapport à son environement exterieur durant cet état. Ce n'est donc pas la notion absolue de temps qui défini la notion d'état mais le fait que pendant la période considérée l'objet ou le système à un comportement différent.

paquetage *Etat*

2.8.8 UtiliteEtat

L'utilité d'un ou plusieurs états n'est pas claire et certains devraient peut être être supprimés (voir *JustificationEtat*).

commentaire Chaque état doit pouvoir être justifié par rapport au comportement du système ou de l'objet (voir *JustificationEtat*). Si un état n'est pas "perceptible" depuis il est peut être préférable de supprimer celui-ci de reporter les informations correspondantes sur une ou des transitions.

exemple Dans le cas d'un système d'ouverture de porte automatique l'état "EnCoursDOuverture" n'est peut être pas pertinent si on ne prend pas en compte l'ensemble des anomalies ou cas particuliers qui peuvent se passer pendant cet "instant". Si ces éléments ne sont pas pertinents, une action "ouvrir" sur une transition sera suffisante (voir *EtatManquant*). De la même manière l'état "EnregistrerLAccesDUnePersonne" est sans doute une action sur une transition plutot qu'un état.

paquetage *Etat*

2.8.9 EtatManquant

Un ou des états semblent manquants pour modéliser le comportement de l'objet ou du système (voir *JustificationEtat*).

commentaire Le comportement du système n'est peut être pas décrits de manière suffisemment fine et il n'est peut être pas possible avec la machine à état décrite de différentier des comportements pourtant différents de l'objet ou du système à des instants différents (voir *JustificationEtat*). Parfois, le problème peut provenir d'une situation modélisée par une transition alors qu'il devrait s'agir d'un état. Une transition est réputée être immédiate, mais si des évenements peuvent survenir pendant cette transition et avoir un effet sur le système alors un état est clairement manquant.

exemple Dans le cas d'un système d'ouverture de porte automatique, si l'on s'intéresse aux différents cas d'exceptions, il sera sans doute nécessaire de créer un état "EnCoursDOuverture" car pendant que la porte s'ouvre un objet ou une personne peut la bloquer par exemple et changer donc l'état du système. On pourra ainsi modéliser que la porte est considérée dans l'état "PorteBloquée" au bout d'un certain temps, qu'elle essaie au contraire de se refermer, etc. L'utilité de tels états dépend entièrement de l'intention de la modélisation (voir *JustificationEtat*)(voir *UtiliteEtat*).

paquetage *Etat*

2.8.10 EtatCree

Il n'est a a priori pas nécessaire d'introduire un état nommé "Créé" dans un diagramme d'état car c'est à cela que correspond l'état initial de l'automate.

paquetage *Etat*

2.8.11 DuplicationEtat

Deux états semblent correspondre au même état.

paquetage *Etat*

2.8.12 SpecificationTransition

La specification d'une ou plusieurs transitions est manquante ou n'est pas appropriée.

commentaire Sauf si le diagramme d'état est dans un état très préliminaire, il est nécessaire de spécifier en détails l'intégralité des transitions (sauf éventuellement celle qui part de l'état initial (voir *TransitionInitialeAutomatique*) et celles qui vont vers l'état final. La specification de chaque transition doit se faire en respectant la syntaxe des expressions de transitions (voir *SyntaxeTransition*). Notons qu'il est très utile de décrire les transitions, mais généralement pas de les nommer (voir *NomTransitionInutile*).

paquetage *Etat*

2.8.13 SyntaxeTransition

La syntaxe des expressions de transitions n'est pas respectée et/ou il existe une ou plusieurs confusions possibles entre les gardes, les événements déclencheurs our déclenchés ou les actions executées.

commentaire Les transitions entre deux états doivent être décorées par des expressions de la forme <evenement1> "[" <garde> "]" / <action> ^ <evenement2> où <evenement1> exprime l'évenement provoquant la transition, <garde> exprime la condition éventuelle devant être vérifiée pour que la transition ait lieu, <action> indique l'action a executer lors de la tranisition et <evenement2> l'évenement déclenché.

paquetage *Etat*

2.8.14 ConfusionEvenementAction

Il semble qu'une confusion soit faite sur une ou plusieurs transitions entre les évenements provoquant les transitions et les actions réalisées lorsque ces transitions sont opérées. Ce problème peut être lié à une mauvaise compréhension du fonctionnement des machines à état ou à une méconnaissance de la syntaxe des expressions de transitions (cf $SyntaxeTransition :).

paquetage *Etat*

2.8.15 ConfusionNomEtatEvenement

Il semble qu'une confusion soit faite entre le nom d'une ou plusieurs transitions et les évenements provoquant ces transitions.

paquetage *Etat*

2.8.16 TransitionInitialeAutomatique

Il n'est pas nécessaire de décorer la transition qui va de l'état initial à un état nommé et en tout état de cause l'évenement correspondant à cette transition ne peut pas correspondre à l'évenement de création de l'objet.

paquetage *Etat*

2.8.17 TransitionManquante

Une ou des transitions semble être manquantes.

commentaire Ce peut être pour modéliser des conditions alternatives, des transitions s'opérant au bout d'un certain temps si aucun événement ne survient, des transitions correspondant à des cas d'exception.

paquetage *Etat*

2.8.18 Puit

Il existe un ou plusieurs états sans transitions sortantes et il ne semble pas que cette situation corresponde à une modélisation réaliste. Des transitions vers l'état final ou des transitions iteratives sont sans doute manquantes (voir *IterationEtats*)(voir *TransitionManquante*).

commentaire Tant que l'objet ou le système est dans un état, cet objet est en vie et il a donc un comportement. Généralement l'objet ou le système peut revenir dans un état précédent.

paquetage *Etat*

2.8.19 AmbiguiteTransition

Parmis les transitions sortantes d'un ou plusieurs états, il n'est pas nécessairement évident de savoir par quelles transitions l'objet sortira d'un état, soit parceque les événements ou gardes sont exprimées de manière trop ambigues, soit parcequ'il existe un chevauchement entre les conditions exprimées par les gardes, soit parces que spécifications des transistions sont inexistantes ou trop pauvrement documentées (voir *SpecificationTransition*).

paquetage *Etat*

2.8.20 IterationEtats

Les transitions ne permettent pas d'itérations entre les différents états alors que c'est le comportement de l'objet ou du système présente cette caractéristique (voir *TransitionManquante*).

exemple Une automate d'une porte d'acces a un batiment doit modeliser de multiple entrées successives et certaines transitions de la machine a état forme nécessairement un cycle.

paquetage *Etat*

2.8.21 CouvertureAutomate

L'automate décrit ne couvre qu'une partie du comportement de l'objet ou du système modélisé. Il manque différents états et transitions (voir *EtatManquant*)(voir *TransitionManquante*).

commentaire Plusieurs explications peuvent être à la source de ce défaut. (1) Le modèle n'est peut être tout simplement pas suffisemment détaillé. (2) Les cas d'exceptions ne sont peut être pas suffisemment pris en compte. (3) Il n'est peut être pas compris qu'un automate ne représente pas un scénario particulier parmis n, mais au contraire couvre l'intégralité du comportement de l'objet tout cas confondu (contrairement aux diagrammes de communication ou aux diagrammes de sequence les automates et diagrammes d'états qui se focalisent sur 1 scenario mais n objets).

paquetage *Etat*

2.9 Deploiement

2.9.1 NomenclatureNoeud

Le nom d'un noeud de deploiement doit être en style MajMin ((voir *MajMin*) s'il s'agit d'une architecture de déploiement général ou en style MinMaj (voir *MinMaj*) s'il s'agit d'un exemple d'architecture déployée sur un site particuler.

commentaire La distinction MajMin et MinMaj permet de souligner le fait qu'il s'agit d'instances ou de classes.

paquetage *Deploiement*

2.9.2 NoeudExecution

Un noeud est un lieu d'exécution, ce n'est pas un acteur humain, un logiciel ou un composant logiciel.

Commentaire ; Un noeud de deploiement est typiquement une machine physique ou virtuelle.

exemple Un noeud peut être par exemple un server d'application, un PC, une machine virtuelle Java, un périphérique, un système embarqué, etc. Par contre une BaseDeDonnees, un fichier, une application peuvent être déployées sur des noeuds mais ne peuvent pas sont pas des noeuds.

paquetage *Deploiement*

2.9.3 Protocole

Le nom de l'association ou du lien devrait faire référence à un protocole de communication.

paquetage *Deploiement*

Les paquetages ci-dessous sont liés à des outils de modèlisation.

2.10 UMLModelio

2.10.1 ModelioR1000

A Model Element cannot abstract itself.

paquetage *UMLModelio*

2.10.2 ModelioR1010

The top Partitions of an Activity must not have a Super-Partition.

paquetage *UMLModelio*

2.10.3 ModelioR1020

The source and the target of a Flow must be contained in the same Structured Activity Node.

paquetage *UMLModelio*

2.10.4 ModelioR1030

The source and the target of a Flow must be owned by the same Activity.

paquetage *UMLModelio*

2.10.5 ModelioR1040

An Activity Parameter Node must be represeneted by a Behavior Parameter owned by the same Activity.

paquetage *UMLModelio*

2.10.6 ModelioR1050

An Activity Parameter Node cannot have both incoming and outgoing edges.

paquetage *UMLModelio*

2.10.7 ModelioR1060

Activity Parameter Nodes with no incoming flow and one or more outgoing flow must have a Behavior Parameter with “In” or “In/Out” parameter passing mode.

paquetage *UMLModelio*

2.10.8 ModelioR1070

Activity Parameter Nodes with no outgoing flow and one or more incoming flow must have a Behavior Parameter with “Out” or “In/Out” parameter passing mode.

paquetage *UMLModelio*

2.10.9 ModelioR1080

All Partitions of the same nesting level must represent Parts of the same Classifier.

paquetage *UMLModelio*

2.10.10 ModelioR1090

If a Sub-Partition is non-external and represents a Classifier then the represented Classifier must be nested in the Classifier represented by its Super-Partition or be the extremity of a Composition with the latter.

paquetage *UMLModelio*

2.10.11 ModelioR1100

If a Sub-Partition represents a Part nested in a Classifier then its Super-Partition must represent the Classifier or an instance of the latter.

paquetage *UMLModelio*

2.10.12 ModelioR1110

There must be one to one correspondence between : (A) the Pins of a Call Behavior Action, and (B) the In, Out, InOut or Return Behavior Parameters of the called Behaviour.

paquetage *UMLModelio*

2.10.13 ModelioR1130

The type and the maximum cardinality of a Call Action”s Pin must match the type and max multiplicity of the represented Parameter.

paquetage *UMLModelio*

2.10.14 ModelioR1140

There must be one to one correspondence between : (A) the Pins of a Call Operation Action, and (B) the In, Inout, Out and Return parameters of the called Operation.

paquetage *UMLModelio*

2.10.15 ModelioR1150

The Call Operation Action or Send Signal Action has more than one target Pin.

paquetage *UMLModelio*

2.10.16 ModelioR1160

A target Pin can only be owned by Call Operation Actions and Send Signal Actions

paquetage *UMLModelio*

2.10.17 ModelioR1170

The type of the target Pin must be the same as the type that owns the Operation.

paquetage *UMLModelio*

2.10.18 ModelioR1180

Control Flows may not have Object Nodes at either end, except for Object Nodes with control type.

paquetage *UMLModelio*

2.10.19 ModelioR1190

The Decision-Merge Node is used both as a Decision node and as a Merge node at the same time.

paquetage *UMLModelio*

2.10.20 ModelioR1200

The edges coming into and out of a Decision Merge Node must be either all Object Flows or all Control Flows.

paquetage *UMLModelio*

2.10.21 ModelioR1230

Only Control Flows can have Initial Nodes as their source.

paquetage *UMLModelio*

2.10.22 ModelioR1250

If a Fork/Join Node has an Object Flow in its incoming edges, it must have an Object Flow in its outgoing edges and vice versa. The same applies for Control Flows.

paquetage *UMLModelio*

2.10.23 ModelioR1280

Object Flows may not have Actions at either end.

paquetage *UMLModelio*

2.10.24 ModelioR1290

Object Nodes connected by an Object Flow, with optionally intervening control nodes, must have compatible types. In particular, the downstream Object Node type must be the same or a super type of the upstream Object Node type.

paquetage *UMLModelio*

2.10.25 ModelioR1300

Object Nodes connected by an Object Flow, with optionally intervening control nodes, must have the same upper bounds.

paquetage *UMLModelio*

2.10.26 ModelioR1310

An edge with constant weight may not target an Object Node, or lead to an Object Node downstream with no intervening actions and with an upper bound less than the weight.

paquetage *UMLModelio*

2.10.27 ModelioR1320

An Object Flow must not be simultaneusly multi-cast and multi-receive.

paquetage *UMLModelio*

2.10.28 ModelioR1350

If an Object Node has a ‘’Selection behavior’‘, then the ’‘Ordering’‘ of the Object Node is ordered and vice versa.

paquetage *UMLModelio*

2.10.29 ModelioR1360

Input Pins may have outgoing edges only when both the following conditions are met : (1) they are on Actions that are Structured Nodes, and (2) these edges must target a Node contained by the Structured Node.

paquetage *UMLModelio*

2.10.30 ModelioR1370

Output Pins may have incoming edges only when both the following conditions are met : (1) they are on Actions that are Structured Nodes, and (2) these edges must come from a node contained by the Structured Node.

paquetage *UMLModelio*

2.10.31 ModelioR1380

There must be one to one correspondence between : (A) the Pins of a Send Signal Action, and (B) the attributes of the sent Signal.

paquetage *UMLModelio*

2.10.32 ModelioR1390

The max cardinality of an argument Pin must be the same as for the represented Attribute.

paquetage *UMLModelio*

2.10.33 ModelioR1400

An Activity Parameter Node can only belong to an Activity.

paquetage *UMLModelio*

2.10.34 ModelioR1410

Only one Association End of an Association can be aggregate or composite.

paquetage *UMLModelio*

2.10.35 ModelioR1420

Actors and UseCases can only have binary Associations.

paquetage *UMLModelio*

2.10.36 ModelioR1430

Multiplicities of an AssociationEnd must be consistent : MultiplicityMin cannot be '*' and MultiplicityMin must be inferior to MultiplicityMax.

paquetage *UMLModelio*

2.10.37 ModelioR1440

AssociationEnds cannot be composite on n-ary Associations.

paquetage *UMLModelio*

2.10.38 ModelioR1450

If an association is a composition, then the opposite maximum multiplicity must be 1.

paquetage *UMLModelio*

2.10.39 ModelioR1460

A public association oriented from a public Classifier cannot be linked to a private or protected Classifier.

paquetage *UMLModelio*

2.10.40 ModelioR1470

The name of an AssociationEnd's qualifiers must be unique.

paquetage *UMLModelio*

2.10.41 ModelioR1480

An Attribute must be typed by a primitive type.

paquetage *UMLModelio*

2.10.42 ModelioR1490

In an instance, the type of an instantiated attribute must be in the instantiated class or in its parent classes.

paquetage *UMLModelio*

2.10.43 ModelioR1500

In an instance, the name of an instantiated attribute must be the same as the corresponding attribute.

paquetage *UMLModelio*

2.10.44 ModelioR1520

The name of a BindableInstance must be unique in it Classifier.

paquetage *UMLModelio*

2.10.45 ModelioR1530

An association or a port should have a name.

paquetage *UMLModelio*

2.10.46 ModelioR1540

A BindableInstance's RepresentedFeature must not refer itself, directly or indirectly.

paquetage *UMLModelio*

2.10.47 ModelioR1550

If a BinbdableInstance has a type and has a represented feature, the type of the instance must be compatible with the type of this feature.

paquetage *UMLModelio*

2.10.48 ModelioR1560

Sub classes of an active class must be active.

paquetage *UMLModelio*

2.10.49 ModelioR1570

A class cannot represent more than one ClassAssociation.

paquetage *UMLModelio*

2.10.50 ModelioR1580

Attributes, Associations and Operations cannot simultaneously be abstract and class.

paquetage *UMLModelio*

2.10.51 ModelioR1590

Primitive GeneralClass cannot have associations.

paquetage *UMLModelio*

2.10.52 ModelioR1600

A primitive class cannot have collaborations.

paquetage *UMLModelio*

2.10.53 ModelioR1610

A primitive class cannot have state machines.

paquetage *UMLModelio*

2.10.54 ModelioR1620

A non-abstract Classifier cannot have abstract methods.

paquetage *UMLModelio*

2.10.55 ModelioR1640

A maximum of one ElementImport must exist between a NameSpace and another NameSpace or between an Operation and a NameSpace.

paquetage *UMLModelio*

2.10.56 ModelioR1650

An Enumeration cannot be abstract.

paquetage *UMLModelio*

2.10.57 ModelioR1660

An enumeration is always prilmitive.

paquetage *UMLModelio*

2.10.58 ModelioR1670

EnumlerationLitteral defined in an Enumeration must have an unique name.

paquetage *UMLModelio*

2.10.59 ModelioR1680

For a Call-type Event, the 'Called operation' field must be defined, whereas the 'Instanciated signal' must be empty.

paquetage *UMLModelio*

2.10.60 ModelioR1690

The 'Expression' field for a Change-type Event must be defined, whereas the 'Called operation' and 'Instanciated signal' fields must be empty.

paquetage *UMLModelio*

2.10.61 ModelioR1700

The 'Instantiated signal' field for a signal-type Event must be defined, whereas the 'Called operation' and 'Expression' fields must be empty.

paquetage *UMLModelio*

2.10.62 ModelioR1710

The 'Expression' field for a Time-type Event must be defined, whereas the 'Called operation' and 'Instanciated signal' fields must be empty.

paquetage *UMLModelio*

2.10.63 ModelioR1720

An abstract NameSpace should only inherit from an abstract NameSpace.

paquetage *UMLModelio*

2.10.64 ModelioR1730

A generalisation must be created between two model elements of the same type, except in the case of a signal, which can specialize a Signal or a Class.

paquetage *UMLModelio*

2.10.65 ModelioR1740

An InformationFlow should convey information.

paquetage *UMLModelio*

2.10.66 ModelioR1750

Repetition of names is forbidden for all AtrributeLinks.

paquetage *UMLModelio*

2.10.67 ModelioR1760

There cannot be inconsistency in the multiplicities of an Instance

paquetage *UMLModelio*

2.10.68 ModelioR1780

The name of an Instance must be unique in its NameSpace.

paquetage *UMLModelio*

2.10.69 ModelioR1790

An instance must have a name, or the instantiation association must be defined.

paquetage *UMLModelio*

2.10.70 ModelioR1800

If an Operator is of type opt, loop, break or neg, there cannot be more than one Operand.

paquetage *UMLModelio*

2.10.71 ModelioR1810

An actual Gate on an InteractionUse must reference a formal Gate contained by the referenced Interaction.

paquetage *UMLModelio*

2.10.72 ModelioR1820

A gate cannot cover a lifeline.

paquetage *UMLModelio*

2.10.73 ModelioR1830

A PartDecomposition cannot receive ‘create’ or ‘destroy’ messages.

paquetage *UMLModelio*

2.10.74 ModelioR1860

In an interface, the visibility of all Features must be public.

paquetage *UMLModelio*

2.10.75 ModelioR1870

An interface cannot be implemented twice by the same class or the same component.

paquetage *UMLModelio*

2.10.76 ModelioR1910

A Link that instantiates an association must be coherent with this association.

paquetage *UMLModelio*

2.10.77 ModelioR1950

Messages of type ‘reply’ cannot invoke an Operation.

paquetage *UMLModelio*

2.10.78 ModelioR1960

A message must have the same name as the invoked Operation.

paquetage *UMLModelio*

2.10.79 ModelioR1970

A TemplateParameterSubstitution must reference a TemplateParameter.

paquetage *UMLModelio*

2.10.80 ModelioR1980

The names of a Classifier’s Attributes and AssociationEnds must be unique.

paquetage *UMLModelio*

2.10.81 ModelioR1990

The name of a Classifier's inherited Attributes and Roles must be unique.

paquetage *UMLModelio*

2.10.82 ModelioR2010

In a Dictionary, the name of each element must be unique.

paquetage *UMLModelio*

2.10.83 ModelioR2030

In a PropertyContainer, the name of each EnumerationPropertyType must be unique.

paquetage *UMLModelio*

2.10.84 ModelioR2050

Some elements must have a name.

paquetage *UMLModelio*

2.10.85 ModelioR2060

The name of a NameSpace must be unique in its NameSpace.

paquetage *UMLModelio*

2.10.86 ModelioR2080

In a PropertySet, the name of each Property must be unique.

paquetage *UMLModelio*

2.10.87 ModelioR2100

In a EnumerationPropertyType, the name of each PropertyEnumerationLiteral must be unique.

paquetage *UMLModelio*

2.10.88 ModelioR2120

In a PropertyContainer, the name of each PropertySet must be unique.

paquetage *UMLModelio*

2.10.89 ModelioR2140

In a PropertyContainer, the name of each PropertyType must be unique.

paquetage *UMLModelio*

2.10.90 ModelioR2160

In an Analyst Container, the name of each element must be unique.

paquetage *UMLModelio*

2.10.91 ModelioR2170

The name of a Behavior must be unique in its NameSpace.

paquetage *UMLModelio*

2.10.92 ModelioR2180

No cycles can exist in a NameSpace inheritance graph.

paquetage *UMLModelio*

2.10.93 ModelioR2190

A maximum of one generalization may exist between two namespaces.

paquetage *UMLModelio*

2.10.94 ModelioR2200

A NameSpace cannot both derive and import another NameSpace.

paquetage *UMLModelio*

2.10.95 ModelioR2210

A leaf NameSpace cannot be derived.

paquetage *UMLModelio*

2.10.96 ModelioR2220

A leaf NameSpace cannot be abstract.

paquetage *UMLModelio*

2.10.97 ModelioR2230

A root NameSpace cannot inherit from any other NameSpace.

paquetage *UMLModelio*

2.10.98 ModelioR2240

There can be no inter-package/inter-component dependency cycle.

paquetage *UMLModelio*

2.10.99 ModelioR2250

All operations in a Classifier must have a different signature from inherited public and protected operations. Except for constructor, destructor and redefined operations.

paquetage *UMLModelio*

2.10.100 ModelioR2260

Each Operation in a Classifer must have a different signature.

paquetage *UMLModelio*

2.10.101 ModelioR2270

All an Operation's Collaborations must have a different name.

paquetage *UMLModelio*

2.10.102 ModelioR2330

All an Operation's Parameters must have a different name.

paquetage *UMLModelio*

2.10.103 ModelioR2340

A redefined Operation must belong to a parent or an implemented Interface of the owner of the Operation.

paquetage *UMLModelio*

2.10.104 ModelioR2350

A private Operation cannot be redefined.

paquetage *UMLModelio*

2.10.105 ModelioR2360

The visibility of an Operation cannot be greater than that of the Operations it redefines.

paquetage *UMLModelio*

2.10.106 ModelioR2370

A class (static) Operation cannot be redefined.

paquetage *UMLModelio*

2.10.107 ModelioR2380

An abstract Operation must not redefine a concrete Operation.

paquetage *UMLModelio*

2.10.108 ModelioR2390

A constructor cannot have return parameters.

paquetage *UMLModelio*

2.10.109 ModelioR2400

A destructor cannot have any kind of parameters.

paquetage *UMLModelio*

2.10.110 ModelioR2410

An operation cannot own both ‘create’ and ‘destroy’ stereotypes.

paquetage *UMLModelio*

2.10.111 ModelioR2420

An Operation must have the same signature as the Operation it redefines.

paquetage *UMLModelio*

2.10.112 ModelioR2430

All an Operation’s StateMachines must have a different name.

paquetage *UMLModelio*

2.10.113 ModelioR2440

An Operation cannot belong to an Enumeration.

paquetage *UMLModelio*

2.10.114 ModelioR2450

A package cannot have inheritance links.

paquetage *UMLModelio*

2.10.115 ModelioR2470

A maximum of one PackageImport link may exist between a NameSpace and a Package.

paquetage *UMLModelio*

2.10.116 ModelioR2500

An ‘out’ Parameter cannot have a default value.

paquetage *UMLModelio*

2.10.117 ModelioR2510

There cannot be any direct link between two Class Ports.

paquetage *UMLModelio*

2.10.118 ModelioR2520

If a Port runs a delegation towards an internal part, it must provide at least one interface.

paquetage *UMLModelio*

2.10.119 ModelioR2530

If a Port receives a delegation from an internal part, it must provide at least one interface.

paquetage *UMLModelio*

2.10.120 ModelioR2540

The interfaces provided by a port must be implemented by the Class that types the Port.

paquetage *UMLModelio*

2.10.121 ModelioR2550

If a Port is a behavior port, its provided interfaces must be implemented by the Class it belongs to.

paquetage *UMLModelio*

2.10.122 ModelioR2560

A behavior Port must provide at least one interface.

paquetage *UMLModelio*

2.10.123 ModelioR2570

If a Port is a behavior port, the type of the port must be either the Class it belongs to or undefined.

paquetage *UMLModelio*

2.10.124 ModelioR2580

A region cannot contain more than one deep history state.

paquetage *UMLModelio*

2.10.125 ModelioR2590

A region cannot contains more than one initial state.

paquetage *UMLModelio*

2.10.126 ModelioR2600

A state machine or a state cannot have two states with the same name.

paquetage *UMLModelio*

2.10.127 ModelioR2610

Only submachine states can have connection point references.

paquetage *UMLModelio*

2.10.128 ModelioR2620

Submachine states should not have entry or exit pseudo states defined.

paquetage *UMLModelio*

2.10.129 ModelioR2630

A region cannot contain more than one shallow history state.

paquetage *UMLModelio*

2.10.130 ModelioR2640

The context of a state machine cannot be an interface.

paquetage *UMLModelio*

2.10.131 ModelioR2650

The context of a protocol state machine must be a Classifier.

paquetage *UMLModelio*

2.10.132 ModelioR2660

A state in a protocol state machine cannot have entry, exit, or do activity actions.

paquetage *UMLModelio*

2.10.133 ModelioR2670

A protocol state machine cannot have history vertexes.

paquetage *UMLModelio*

2.10.134 ModelioR2680

The number of parameter of a TaggedValue must be the same as the number of parameter defined in the TaggedValue declaration.

paquetage *UMLModelio*

2.10.135 ModelioR2690

An element cannot have a TemplateBinding towards itself.

paquetage *UMLModelio*

2.10.136 ModelioR2700

A TemplateBinding can only substitute each TemplateParameter of the instantiated element once.

paquetage *UMLModelio*

2.10.137 ModelioR2720

A TemplateBinding must be created between two elements of the same type or between a Class and a DataType.

paquetage *UMLModelio*

2.10.138 ModelioR2730

A TemplateBinding must substitute all the TemplateParameters of the instanciated template element, and the TemplateParameterSubstitution must be defines in the same order as the TemplateParameters.

paquetage *UMLModelio*

2.10.139 ModelioR2740

In a TemplateBinding, the TemplateParameterSubstitution must belong to the instantiated template element.

paquetage *UMLModelio*

2.10.140 ModelioR2750

A transition from a fork or join pseudo state should not have guards or triggers.

paquetage *UMLModelio*

2.10.141 ModelioR2760

A fork segment must always target a state.

paquetage *UMLModelio*

2.10.142 ModelioR2770

A join segment must always originate from a state.

paquetage *UMLModelio*

2.10.143 ModelioR2780

Transitions outgoing pseudostates may not have a trigger (except for those coming out of the initial pseudostate).

paquetage *UMLModelio*

2.10.144 ModelioR2790

A transition from one region to another in the same immediate enclosing composite state is not allowed.

paquetage *UMLModelio*

2.10.145 ModelioR2800

An initial vertex can have at most one outgoing transition.

paquetage *UMLModelio*

2.10.146 ModelioR2810

History vertices can have at most one outgoing transition.

paquetage *UMLModelio*

2.10.147 ModelioR2820

The target of a transition cannot be an initial vertex.

paquetage *UMLModelio*

2.10.148 ModelioR2830

The source of a transition cannot be a final vertex.

paquetage *UMLModelio*

2.10.149 ModelioR2840

A transition should have only one of Processed, Effects, or BehaviorEffet defined.

paquetage *UMLModelio*

2.10.150 ModelioR2850

An element cannot have a usage dependency towards itself.

paquetage *UMLModelio*

2.10.151 ModelioR2860

A maximum of one dependency may exist between two use cases.

paquetage *UMLModelio*

2.10.152 ModelioR2870

There must be no cycle in use cases << extend >> dependency graph.

paquetage *UMLModelio*

2.10.153 ModelioR2880

There must be no cycle in use cases << include >> dependency graph.

paquetage *UMLModelio*

2.10.154 ModelioR2890

A communication link cannot have the same actor or use case as its source and target.

paquetage *UMLModelio*

2.10.155 ModelioR2900

An << extend >> use case dependency must reference at least one extension point.

paquetage *UMLModelio*

2.10.156 ModelioR2910

An << extend >> use case dependency can only reference the target's extension points.

paquetage *UMLModelio*

2.10.157 ModelioR2920

Extension points can only be referenced by an << extend >> use case dependency.

paquetage *UMLModelio*

2.10.158 ModelioR2930

Message and CommunicationMessage cannot have both Signal and Operation properties defined.

paquetage *UMLModelio*

2.10.159 ModelioR2940

All transitions incoming a join vertex must originate in different regions of an orthogonal state.

paquetage *UMLModelio*

2.10.160 ModelioR2950

All transitions outgoing a fork vertex must target states in different regions of an orthogonal state.

paquetage *UMLModelio*

2.10.161 ModelioR2960

Synonym, antonym, homonym, context, and kind-of dependencies can only link two terms.

paquetage *UMLModelio*

2.10.162 ModelioR2970

An Assigned dependency must be from an Actor, an Interface, a Package, or a Process, toward a Goal.

paquetage *UMLModelio*

2.10.163 ModelioR2980

A Measure dependency must be from a ModelElement toward a Goal.

paquetage *UMLModelio*

2.10.164 ModelioR2990

A Guarantee dependency must be from a Requirement toward a Goal.

paquetage *UMLModelio*

2.10.165 ModelioR3000

Positive influence and Negative influence dependencies must be between two Goals.

paquetage *UMLModelio*

2.10.166 ModelioR3010

A refers dependency must be between a Business Rule and a Term.

paquetage *UMLModelio*

2.10.167 ModelioR3020

A related dependency must be must be between two Business Rules or two Terms.

paquetage *UMLModelio*

2.10.168 ModelioR3030

A refine dependency must be between either : 1) from a Model Element or a Requirement towards a Requirement 2) from a Business Rule, an Activity or an Operation towards a Business Rule.

paquetage *UMLModelio*

2.10.169 ModelioR3040

An implement dependency must be from a Process or a Class towards a Business Rule.

paquetage *UMLModelio*

2.10.170 ModelioR3050

A part dependency must be between two Requirements or between two Goals.

paquetage *UMLModelio*

2.10.171 ModelioR3060

A satisfy or verify dependency must be from a ModelElement towards a Requirement.

paquetage *UMLModelio*

2.10.172 ModelioR3070

A derive dependency must be from a UseCase or a Requirement towards a Requirement.

paquetage *UMLModelio*

2.10.173 ModelioR3080

All FlowNodes should be part of a sequence starting with a StartEvent and finishing with an EndEvent.

paquetage *UMLModelio*

2.10.174 ModelioR3090

A SequenceFlow cannot have its source or target in different Pools.

paquetage *UMLModelio*

2.10.175 ModelioR3100

A SequcneFlow in a SubProcess must have its origin and target in the same SubProcess.

paquetage *UMLModelio*

2.10.176 ModelioR3110

A SequenceFlow cannot target a StartEvent nor have an EndEvent as its source.

paquetage *UMLModelio*

2.10.177 ModelioR3130

A MessageFlow cannot target a StartEvent or an IntermediateThrowEvent, nor have an EndEvent or an IntermediateCatchEvent as its source.

paquetage *UMLModelio*

2.10.178 ModelioR3140

All outgoing SequenceFlow from an EventBasedGateway or a ParallelGateway must have its guard properties empty.

paquetage *UMLModelio*

2.10.179 ModelioR3150

A MessageFlow cannot link two elements in the same lane.

paquetage *UMLModelio*

2.10.180 ModelioR3160

A MessageFlow cannot have a Gateway as its source or target.

paquetage *UMLModelio*

2.10.181 ModelioR3170

Inclusive Gateway,Complex Gateway and Parallel Gateway must have at least two outgoing Sequence Flows.

paquetage *UMLModelio*

2.10.182 ModelioR3180

A FlowElement (and respectively a BaseElement) cannot have a SequenceFlow (respectively a MessageFlow) towards itself.

paquetage *UMLModelio*

2.10.183 ModelioR3190

A DataAssociation cannot target a DataInput nor have a DataOutput as its source.

paquetage *UMLModelio*

2.10.184 ModelioR3220

A SequenceFlow outgoing from an EventBasedGateway must target an IntermediaryCatchEvent.

paquetage *UMLModelio*

2.10.185 ModelioR3230

All SequenceFlows outgoing from an ExclusiveGateway must have a guard, except for the default SequenceFlow.

paquetage *UMLModelio*

2.10.186 ModelioR3240

There can only be one sequence in a Process, a SubProcess or a Pool.

paquetage *UMLModelio*

2.10.187 ModelioR3250

A Process, a SubProcess, or a Pool should have at least one StartEvent and one EndEvent.

paquetage *UMLModelio*

2.11 UMLStarUML

2.11.1 StarUML1

AssociationEnds within an Association must have unique names. — Association

paquetage *UMLStarUML*

2.11.2 StarUML2

Two or more Aggregations or Composite AssociationEnds cannot exist within an Association. — Association

paquetage *UMLStarUML*

2.11.3 StarUML3

Parameters must have unique names. — BehavioralFeature

paquetage *UMLStarUML*

2.11.4 StarUML4

Attributes of the same name cannot exist within a Classifier. — Classifier

paquetage *UMLStarUML*

2.11.5 StarUML5

AssociationEnds on the other side must have unique names. — Classifier

paquetage *UMLStarUML*

2.11.6 StarUML6

An Attribute cannot have the same name as the Association on the other side, or as elements included in Classifier. — Classifier

paquetage *UMLStarUML*

2.11.7 StarUML7

AssociationEnd on the other side cannot have the same name as elements included in Classifier or its Attribute name. — Classifier

paquetage *UMLStarUML*

2.11.8 StarUML8

Root element cannot have elements that are more generalized than itself. — GeneralizableElement

paquetage *UMLStarUML*

2.11.9 StarUML9

Leaf element cannot have elements that are more specialized than itself. — GeneralizableElement

paquetage *UMLStarUML*

2.11.10 StarUML10

Looped inheritance structure is not allowed. — GeneralizableElement

paquetage *UMLStarUML*

2.11.11 StarUML11

All features of interfaces must be public. — Interface

paquetage *UMLStarUML*

2.11.12 StarUML12

ComponentInstance must accurately assign a component as its origin. — ComponentInstance

paquetage *UMLStarUML*

2.11.13 StarUML13

NodeInstance must accurately assign a node as its origin. — NodeInstance

paquetage *UMLStarUML*

2.11.14 StarUML14

AssociationEndRole must be connected with ClassifierRole. — AssociationEndRole

paquetage *UMLStarUML*

2.11.15 StarUML15

ClassifierRole cannot have its own features. — ClassifierRole

paquetage *UMLStarUML*

2.11.16 StarUML16

ClassifierRole cannot become the ClassifierRole for another ClassifierRole. — ClassifierRole

paquetage *UMLStarUML*

2.11.17 StarUML17

Sender and receiver of a message must participate in the collaboration that constitutes the interaction context. — Message

paquetage *UMLStarUML*

2.11.18 StarUML18

Actor can only have associations that are connected to UseCase, Class or Subsystem. — Actor

paquetage *UMLStarUML*

2.11.19 StarUML19

CompositeState can have a maximum of one initial state only. — CompositeState

paquetage *UMLStarUML*

2.11.20 StarUML20

CompositeState can have a maximum of one deep history only. — CompositeState

paquetage *UMLStarUML*

2.11.21 StarUML21

CompositeState can have a maximum of one shallow history only. — CompositeState

paquetage *UMLStarUML*

2.11.22 StarUML22

Concurrent composite state must contain a minimum of two composite states. — CompositeState

paquetage *UMLStarUML*

2.11.23 StarUML23

Concurrent state can only have composite state as its sub state. — CompositeState

paquetage *UMLStarUML*

2.11.24 StarUML24

Final state cannot have outgoing transition. — FinalState

paquetage *UMLStarUML*

2.11.25 StarUML25

Initial state can have a maximum of one outgoing transition and cannot have incoming transition. — Pseudostate

paquetage *UMLStarUML*

2.11.26 StarUML26

History state can have a maximum of one outgoing transition. — Pseudostate

paquetage *UMLStarUML*

2.11.27 StarUML27

Junction vertex must have a minimum of one incoming transition and one outgoing transition each. — Pseudostate

paquetage *UMLStarUML*

2.11.28 StarUML28

Choice vertex must have a minimum of one incoming transition and one outgoing transition each. — Pseudostate

paquetage *UMLStarUML*

2.11.29 StarUML29

StateMachine can be integrated either with Classifier or with BehavioralFeature. — StateMachine

paquetage *UMLStarUML*

2.11.30 StarUML30

Top state must always be composite state. — StateMachine

paquetage *UMLStarUML*

2.11.31 StarUML31

No state can contain top state. — StateMachine

paquetage *UMLStarUML*

2.11.32 StarUML32

Top state cannot have outgoing transition. — StateMachine

paquetage *UMLStarUML*

2.11.33 StarUML33

SubmachineState cannot have concurrency. — SubmachineState

paquetage *UMLStarUML*

2.11.34 StarUML34

Transition that points to Pseudostate cannot have Trigger. — Transition

paquetage *UMLStarUML*

2.11.35 StarUML35

ActivityGraph can express dynamic behavior of Package, Classifier or BehavioralFeature. — ActivityGraph

paquetage *UMLStarUML*

2.11.36 StarUML36

ActionState cannot have internal transition, exit action or do activity. — ActionState

paquetage *UMLStarUML*

2.11.37 StarUML37

Outgoing transition of ActionState cannot have trigger event. — ActionState

paquetage *UMLStarUML*

2.11.38 StarUML38

SubactivityState must have connection to ActivityGraph. — SubactivityState

paquetage *UMLStarUML*

CHAPITRE 3

Implementation

3.1 BaseDeDonnees

3.1.1 NomRelation

Le nom d'une relation doit correspondre à une forme nominale plurielle. Par ailleurs les termes utilisés dans le nom doivent généralement être définis dans le glossaire. Si une abbréviation est utilisée celle-ci devra être impérativement définie dans le glossaire.

exemple "LesPersonnes" ou "TheLoanedBooks"

commentaire Contrairement au nom d'une classe (voir *NomClasse*) qui est une forme nominale au singuler, les relations correspondent à un ensemble d'entités.

exemple Les objets de classe "Personne" seront donc naturellement stockées dans la relation "LesPersonnes".

paquetage *BaseDeDonnees*

3.1.2 NomenclatureRelation

Le nom d'une relation doit être en style MajMin (voir *MajMin*).

paquetage *BaseDeDonnees*

3.1.3 NomRelationGlossaire

Les termes utilisés dans le nom des relations doivent être définis dans le glossaire. Si une abbréviation est utilisée celle-ci devra être impérativement définie dans le glossaire.

paquetage *BaseDeDonnees*

3.1.4 NomColonne

Dans une relation, le nom de chaque colonne doit correspondre à une forme nominale correspondant à l'attribut ou au concept représenté, sauf eventuellement pour les colonnes représentant une valeur booléenne auxquel cas une forme verbale peut être acceptable. Par ailleurs les termes utilisés dans le nom doivent être définis dans le glossaire. Si une abbréviation est utilisée celle-ci devra être impérativement définie dans le glossaire.

exemple "adresse", "estArrive"

paquetage *BaseDeDonnees*

3.1.5 NomenclatureColonne

Le nom d'une relation doit être en style minMaj (voir *MinMaj*).

paquetage *BaseDeDonnees*

3.1.6 NomColonneGlossaire

Les termes utilisés dans le nom des colonnes des relations doivent être définis dans le glossaire, en tout cas pour les termes principaux et ceux dont l'interprétation ne pose pas problème. Si une abbréviation est utilisée celle-ci devra être impérativement définie dans le glossaire.

paquetage *BaseDeDonnees*

3.1.7 NomCleEtrangere

Le nom des colonnes correspondant à des clés étrangères doit permettre d'identifier clairement le type d'entités référencés ainsi que la clé utilisé pour ce référencement.

paquetage *BaseDeDonnees*

3.1.8 Schema1FN

Le schéma de la base de données doit être en 1ère forme normale.

paquetage *BaseDeDonnees*

3.1.9 Schema2FN

Le schéma de la base de données doit être en Zème forme normale.

paquetage *BaseDeDonnees*

3.1.10 Schema3FN

Le schéma de la base de données doit être en 3ème forme normale.

paquetage *BaseDeDonnees*

3.2 ProgrammationWeb

3.2.1 NomPageJSP

TODO

paquetage *ProgrammationWeb*

3.2.2 NomFichierCSS

TODO

paquetage *ProgrammationWeb*

3.3 JavaCheckStyle

3.3.1 JavaAbbreviationAsWordInName

The Check validate abbreviations(consecutive capital letters) length in identifier name, it also allow in enforce camel case naming.

paquetage *JavaCheckStyle*

3.3.2 JavaAbstractClassName

Ensures that the names of abstract classes conforming to some regular expression.

paquetage *JavaCheckStyle*

3.3.3 JavaAnnotationLocation

Check location of annotation on language elements.

paquetage *JavaCheckStyle*

3.3.4 JavaAnnotationUseStyle

This check controls the style with the usage of annotations.

paquetage *JavaCheckStyle*

3.3.5 JavaAnonInnerLength

Checks for long anonymous inner classes.

paquetage *JavaCheckStyle*

3.3.6 JavaArrayTrailingComma

Checks if array initialization contains optional trailing comma.

paquetage *JavaCheckStyle*

3.3.7 JavaArrayTypeStyle

Checks the style of array type definitions.

paquetage *JavaCheckStyle*

3.3.8 JavaAtclauseOrder

Checks the order of at-clauses.

paquetage *JavaCheckStyle*

3.3.9 JavaAvoidEscapedUnicodeCharacters

Restrict using Unicode escapes.

paquetage *JavaCheckStyle*

3.3.10 JavaAvoidInlineConditionals

Detects inline conditionals.

paquetage *JavaCheckStyle*

3.3.11 JavaAvoidNestedBlocks

Finds nested blocks.

paquetage *JavaCheckStyle*

3.3.12 JavaAvoidStarImport

Check that finds import statements that use the * notation.

paquetage *JavaCheckStyle*

3.3.13 JavaAvoidStaticImport

Check that finds static imports.

paquetage *JavaCheckStyle*

3.3.14 JavaBooleanExpressionComplexity

Restricts nested boolean operators (&&, ||, &, | and ^) to a specified depth (default = 3).

paquetage *JavaCheckStyle*

3.3.15 JavaClassDataAbstractionCoupling

This metric measures the number of instantiations of other classes within the given class.

paquetage *JavaCheckStyle*

3.3.16 JavaClassFanOutComplexity

The number of other classes a given class relies on.

paquetage *JavaCheckStyle*

3.3.17 JavaClassTypeParameterName

Checks that class type parameter names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.18 JavaConstantName

Checks that constant names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.19 JavaCovariantEquals

Checks that if a class defines a covariant method equals, then it defines method equals(java.lang.Object).

paquetage *JavaCheckStyle*

3.3.20 JavaCustomImportOrder

Checks that the groups of import declarations appear in the order specified by the user.

paquetage *JavaCheckStyle*

3.3.21 JavaCyclomaticComplexity

Checks cyclomatic complexity against a specified limit.

paquetage *JavaCheckStyle*

3.3.22 JavaDeclarationOrder

Checks that the parts of a class or interface declaration appear in the order suggested by the Code Conventions for the Java Programming Language

paquetage *JavaCheckStyle*

3.3.23 JavaDefaultComesLast

Check that the default is after all the cases in a switch statement.

paquetage *JavaCheckStyle*

3.3.24 JavaDescendantToken

Checks for restricted tokens beneath other tokens.

paquetage *JavaCheckStyle*

3.3.25 JavaDesignForExtension

Checks that classes are designed for inheritance.

paquetage *JavaCheckStyle*

3.3.26 JavaEmptyBlock

Checks for empty blocks.

paquetage *JavaCheckStyle*

3.3.27 JavaEmptyForInitializerPad

Checks the padding of an empty for initializer ; that is whether a space is required at an empty for initializer, or such spaces are forbidden.

paquetage *JavaCheckStyle*

3.3.28 JavaEmptyForIteratorPad

Checks the padding of an empty for iterator ; that is whether a space is required at an empty for iterator, or such spaces are forbidden.

paquetage *JavaCheckStyle*

3.3.29 JavaEmptyLineSeparator

Checks for blank line separators.

paquetage *JavaCheckStyle*

3.3.30 JavaEmptyStatement

Detects empty statements (standalone ‘ ;’).

paquetage *JavaCheckStyle*

3.3.31 JavaEqualsAvoidNull

Checks that any combination of String literals with optional assignment is on the left side of an equals() comparison.

paquetage *JavaCheckStyle*

3.3.32 JavaEqualsHashCode

Checks that classes that override equals() also override hashCode().

paquetage *JavaCheckStyle*

3.3.33 JavaExecutableStatementCount

Restricts the number of executable statements to a specified limit (default = 30).

paquetage *JavaCheckStyle*

3.3.34 JavaExplicitInitialization

Checks if any class or object member explicitly initialized to default for its type value (null for object references, zero for numeric types and char and false for boolean.

paquetage *JavaCheckStyle*

3.3.35 JavaFallThrough

Checks for fall through in switch statements Finds locations where a case contains Java code - but lacks a break, return, throw or continue statement.

paquetage *JavaCheckStyle*

3.3.36 JavaFileLength

Checks for long source files.

paquetage *JavaCheckStyle*

3.3.37 JavaFileTabCharacter

Checks to see if a file contains a tab character.

paquetage *JavaCheckStyle*

3.3.38 JavaFinalClass

Checks that class which has only private ctors is declared as final.

paquetage *JavaCheckStyle*

3.3.39 JavaFinalLocalVariable

Ensures that local variables that never get their values changed, must be declared final.

paquetage *JavaCheckStyle*

3.3.40 JavaFinalParameters

Check that method/constructor/catch/foreach parameters are final.

paquetage *JavaCheckStyle*

3.3.41 JavaGenericWhitespace

Checks that the whitespace around the Generic tokens < and > are correct to the typical convention.

paquetage *JavaCheckStyle*

3.3.42 JavaHeader

Checks the header of the source against a fixed header file.

paquetage *JavaCheckStyle*

3.3.43 JavaHiddenField

Checks that a local variable or a parameter does not shadow a field that is defined in the same class.

paquetage *JavaCheckStyle*

3.3.44 JavaHideUtilityClassConstructor

Make sure that utility classes (classes that contain only static methods) do not have a public constructor.

paquetage *JavaCheckStyle*

3.3.45 JavaIllegalCatch

Catching java.lang.Exception, java.lang.Error or java.lang.RuntimeException is almost never acceptable.

paquetage *JavaCheckStyle*

3.3.46 JavaIllegalImport

Checks for imports from a set of illegal packages.

paquetage *JavaCheckStyle*

3.3.47 JavaIllegalInstantiation

Checks for illegal instantiations where a factory method is preferred.

paquetage *JavaCheckStyle*

3.3.48 JavaIllegalThrows

Throwing java.lang.Error or java.lang.RuntimeException is almost never acceptable.

paquetage *JavaCheckStyle*

3.3.49 JavaIllegalToken

Checks for illegal tokens.

paquetage *JavaCheckStyle*

3.3.50 JavaIllegalTokenText

Checks for illegal token text.

paquetage *JavaCheckStyle*

3.3.51 JavaIllegalType

Checks that particular class are never used as types in variable declarations, return values or parameters.

paquetage *JavaCheckStyle*

3.3.52 JavaImportControl

Check that controls what packages can be imported in each package.

paquetage *JavaCheckStyle*

3.3.53 JavaImportOrder

Ensures that groups of imports come in a specific order.

paquetage *JavaCheckStyle*

3.3.54 JavaIndentation

Checks correct indentation of Java Code.

paquetage *JavaCheckStyle*

3.3.55 JavaInnerAssignment

Checks for assignments in subexpressions, such as in String s = Integer.toString(i = 2) ;.

paquetage *JavaCheckStyle*

3.3.56 JavaInnerTypeLast

Check nested (internal) classes/interfaces are declared at the bottom of the class after all method and field declarations.

paquetage *JavaCheckStyle*

3.3.57 JavaInterfaceIsType

Implements Bloch, Effective Java, Item 17 - Use Interfaces only to define types.

paquetage *JavaCheckStyle*

3.3.58 JavaInterfaceTypeParameterName

Checks that interface type parameter names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.59 JavaJavaNCSS

This check calculates the Non Commenting Source Statements (NCSS) metric for Java source files and methods.

paquetage *JavaCheckStyle*

3.3.60 JavaJavadocMethod

Checks the Javadoc of a method or constructor.

paquetage *JavaCheckStyle*

3.3.61 JavaJavadocPackage

Checks that all packages have a package documentation.

paquetage *JavaCheckStyle*

3.3.62 JavaJavadocTagContinuationIndentation

Checks the indentation of the continuation lines in at-clauses.

paquetage *JavaCheckStyle*

3.3.63 JavaJavadocParagraph

Checks Javadoc paragraphs.

paquetage *JavaCheckStyle*

3.3.64 JavaJavadocStyle

Custom Checkstyle Check to validate Javadoc.

paquetage *JavaCheckStyle*

3.3.65 JavaJavadocType

Checks the Javadoc of a type.

paquetage *JavaCheckStyle*

3.3.66 JavaJavadocVariable

Checks that a variable has Javadoc comment.

paquetage *JavaCheckStyle*

3.3.67 JavaLeftCurly

Checks the placement of left curly braces on types, methods and other blocks :

paquetage *JavaCheckStyle*

3.3.68 JavaLineLength

Checks for long lines.

paquetage *JavaCheckStyle*

3.3.69 JavaLocalFinalVariableName

Checks that local final variable names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.70 JavaLocalVariableName

Checks that local, non-final variable names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.71 JavaMagicNumber

Checks for magic numbers.

paquetage *JavaCheckStyle*

3.3.72 JavaMemberName

Checks that instance variable names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.73 JavaMethodCount

Checks the number of methods declared in each type.

paquetage *JavaCheckStyle*

3.3.74 JavaMethodLength

Checks for long methods.

paquetage *JavaCheckStyle*

3.3.75 JavaMethodName

Checks that method names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.76 JavaMethodParamPad

Checks the padding between the identifier of a method definition, constructor definition, method call, or constructor invocation ; and the left parenthesis of the parameter list.

paquetage *JavaCheckStyle*

3.3.77 JavaMethodTypeParameterName

Checks that class type parameter names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.78 JavaMissingCtor

Checks that classes (except abstract one) define a ctor and don't rely on the default one.

paquetage *JavaCheckStyle*

3.3.79 JavaMissingDeprecated

This class is used to verify that both the

paquetage *JavaCheckStyle*

3.3.80 JavaMissingOverride

This class is used to verify that the

paquetage *JavaCheckStyle*

3.3.81 JavaMissingSwitchDefault

Checks that switch statement has “default” clause.

paquetage *JavaCheckStyle*

3.3.82 JavaModifiedControlVariable

Check for ensuring that for loop control variables are not modified inside the for block.

paquetage *JavaCheckStyle*

3.3.83 JavaModifierOrder

Checks that the order of modifiers conforms to the suggestions in the Java Language specification, sections 8.1.1, 8.3.1 and 8.4.3.

paquetage *JavaCheckStyle*

3.3.84 JavaMultipleStringLiterals

Checks for multiple occurrences of the same string literal within a single file.

paquetage *JavaCheckStyle*

3.3.85 JavaMultipleVariableDeclarations

Checks that each variable declaration is in its own statement and on its own line.

paquetage *JavaCheckStyle*

3.3.86 JavaMutableException

Ensures that exceptions (defined as any class name conforming to some regular expression) are immutable.

paquetage *JavaCheckStyle*

3.3.87 JavaNPathComplexity

Checks the npath complexity against a specified limit (default = 200).

paquetage *JavaCheckStyle*

3.3.88 JavaNeedBraces

Checks for braces around code blocks.

paquetage *JavaCheckStyle*

3.3.89 JavaNestedForDepth

Restricts nested for blocks to a specified depth (default = 1).

paquetage *JavaCheckStyle*

3.3.90 JavaNestedIfDepth

Restricts nested if-else blocks to a specified depth (default = 1).

paquetage *JavaCheckStyle*

3.3.91 JavaNestedTryDepth

Restricts nested try-catch-finally blocks to a specified depth (default = 1).

paquetage *JavaCheckStyle*

3.3.92 JavaNewlineAtEndOfFile

Checks that there is a newline at the end of each file.

paquetage *JavaCheckStyle*

3.3.93 JavaNoClone

Checks that the clone method is not overridden from the Object class.

paquetage *JavaCheckStyle*

3.3.94 JavaNoFinalizer

Checks that no method having zero parameters is defined using the name finalize.

paquetage *JavaCheckStyle*

3.3.95 JavaNonEmptyAtclauseDescription

Checks that the at-clause tag is followed by description .

paquetage *JavaCheckStyle*

3.3.96 JavaNoLineWrap

Checks that chosen statements are not line-wrapped.

paquetage *JavaCheckStyle*

3.3.97 JavaNoWhitespaceAfter

Checks that there is no whitespace after a token.

paquetage *JavaCheckStyle*

3.3.98 JavaNoWhitespaceBefore

Checks that there is no whitespace before a token.

paquetage *JavaCheckStyle*

3.3.99 JavaOneStatementPerLine

Checks there is only one statement per line.

paquetage *JavaCheckStyle*

3.3.100 JavaOneTopLevelClass

Checks that each top-level class, interfaces or enum resides in a source file of its own.

paquetage *JavaCheckStyle*

3.3.101 JavaOperatorWrap

Checks line wrapping for operators.

paquetage *JavaCheckStyle*

3.3.102 JavaOuterTypeFilename

Checks that the outer type name and the file name match.

paquetage *JavaCheckStyle*

3.3.103 JavaOuterTypeNumber

Checks for the number of defined types at the “outer” level.

paquetage *JavaCheckStyle*

3.3.104 JavaPackageAnnotation

This check makes sure that all package annotations are in the package-info.java file.

paquetage *JavaCheckStyle*

3.3.105 JavaPackageDeclaration

Ensures there is a package declaration and (optionally) in the correct directory.

paquetage *JavaCheckStyle*

3.3.106 JavaPackageName

Checks that package names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.107 JavaParameterAssignment

Disallow assignment of parameters.

paquetage *JavaCheckStyle*

3.3.108 JavaParameterName

Checks that parameter names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.109 JavaParameterNumber

Checks the number of parameters that a method or constructor has.

paquetage *JavaCheckStyle*

3.3.110 JavaParenPad

Checks the padding of parentheses ; that is whether a space is required after a left parenthesis and before a right parenthesis, or such spaces are forbidden, with the exception that it does not check for padding of the right parenthesis at an empty for iterator.

paquetage *JavaCheckStyle*

3.3.111 JavaRedundantImport

Checks for imports that are redundant.

paquetage *JavaCheckStyle*

3.3.112 JavaRedundantModifier

Checks for redundant modifiers in interface and annotation definitions.

paquetage *JavaCheckStyle*

3.3.113 JavaRegexp

A check that makes sure that a specified pattern exists (or not) in the file.

paquetage *JavaCheckStyle*

3.3.114 JavaRegexpHeader

Checks the header of the source against a header file that contains a

paquetage *JavaCheckStyle*

3.3.115 JavaRegexpMultiline

Implementation of a check that looks that matches across multiple lines in any file type.

paquetage *JavaCheckStyle*

3.3.116 JavaRegexpSingleline

Implementation of a check that looks for a single line in any file type.

paquetage *JavaCheckStyle*

3.3.117 JavaRegexpSinglelineJava

Implementation of a check that looks for a single line in Java files.

paquetage *JavaCheckStyle*

3.3.118 JavaRequireThis

Checks that code doesn't rely on the "this" default.

paquetage *JavaCheckStyle*

3.3.119 JavaReturnCount

Restricts return statements to a specified count (default = 2).

paquetage *JavaCheckStyle*

3.3.120 JavaRightCurly

Checks the placement of right curly braces.

paquetage *JavaCheckStyle*

3.3.121 JavaSeparatorWrap

Checks line wrapping with separators.

paquetage *JavaCheckStyle*

3.3.122 JavaSingleLineJavadoc

Checks that a JavaDoc block which can fit on a single line and doesn't contain at-clauses

paquetage *JavaCheckStyle*

3.3.123 JavaSimplifyBooleanExpression

Checks for overly complicated boolean expressions.

paquetage *JavaCheckStyle*

3.3.124 JavaSimplifyBooleanReturn

Checks for overly complicated boolean return statements.

paquetage *JavaCheckStyle*

3.3.125 JavaStaticVariableName

Checks that static, non-final variable names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.126 JavaStringLiteralEquality

Checks that string literals are not used with == or !=.

paquetage *JavaCheckStyle*

3.3.127 JavaSummaryJavadoc

Checks that Javadoc summary sentence does not contain phrases that are not recommended to use.

paquetage *JavaCheckStyle*

3.3.128 JavaSuperClone

Checks that an overriding clone() method invokes super.clone().

paquetage *JavaCheckStyle*

3.3.129 JavaSuperFinalize

Checks that an overriding finalize() method invokes super.finalize().

paquetage *JavaCheckStyle*

3.3.130 JavaSuppressWarnings

This check allows you to specify what warnings that

paquetage *JavaCheckStyle*

3.3.131 JavaSuppressWarningsHolder

This check allows for finding code that should not be reported by Checkstyle

paquetage *JavaCheckStyle*

3.3.132 JavaThrowsCount

Restricts throws statements to a specified count (default = 1).

paquetage *JavaCheckStyle*

3.3.133 JavaTodoComment

A check for TODO comments.

paquetage *JavaCheckStyle*

3.3.134 JavaTrailingComment

The check to ensure that requires that comments be the only thing on a line.

paquetage *JavaCheckStyle*

3.3.135 JavaTranslation

The TranslationCheck class helps to ensure the correct translation of code by checking property files for consistency regarding their keys.

paquetage *JavaCheckStyle*

3.3.136 JavaTypeName

Checks that type names conform to a format specified by the format property.

paquetage *JavaCheckStyle*

3.3.137 JavaTypecastParenPad

Checks the padding of parentheses for typecasts.

paquetage *JavaCheckStyle*

3.3.138 JavaUncommentedMain

Detects uncommented main methods.

paquetage *JavaCheckStyle*

3.3.139 JavaUniqueProperties

Detects duplicated keys in properties files.

paquetage *JavaCheckStyle*

3.3.140 JavaUnnecessaryParentheses

Checks if unnecessary parentheses are used in a statement or expression.

paquetage *JavaCheckStyle*

3.3.141 JavaUnusedImports

Checks for unused import statements.

paquetage *JavaCheckStyle*

3.3.142 JavaUpperEll

Checks that long constants are defined with an upper ell.

paquetage *JavaCheckStyle*

3.3.143 JavaVariableDeclarationUsageDistance

Checks the distance between declaration of variable and its first usage.

paquetage *JavaCheckStyle*

3.3.144 JavaVisibilityModifier

Checks visibility of class members.

paquetage *JavaCheckStyle*

3.3.145 JavaWhitespaceAfter

Checks that a token is followed by whitespace, with the exception that it does not check for whitespace after the semicolon of an empty for iterator.

paquetage *JavaCheckStyle*

3.3.146 JavaWhitespaceAround

Checks that a token is surrounded by whitespace.

paquetage *JavaCheckStyle*

3.3.147 JavaWriteTag

Outputs a JavaDoc tag as information.

paquetage *JavaCheckStyle*

3.4 PythonPyLint

3.4.1 Python0102

Black listed name “”%s”“

%s already defined line %s

Dangerous default value %s as argument

paquetage *PythonPyLint*

3.4.2 Python0103

Invalid %s name “”%s”“

%r not properly in loop

paquetage *PythonPyLint*

3.4.3 Python0111

Missing %s docstring

paquetage *PythonPyLint*

3.4.4 Python0112

Empty %s docstring

paquetage *PythonPyLint*

3.4.5 Python0121

Missing required attribute “”%s”“

Use raise ErrorClass(args) instead of raise ErrorClass, args.

paquetage *PythonPyLint*

3.4.6 Python0202

Class method %s should have cls as first argument

An attribute affected in %s line %s hide this method

Unable to check methods signature (%s / %s)

paquetage *PythonPyLint*

3.4.7 Python0203

Metaclass method %s should have mcs as first argument

Access to member %r before its definition line %s

paquetage *PythonPyLint*

3.4.8 Python0204

Metaclass class method %s should have %s as first argument

paquetage *PythonPyLint*

3.4.9 Python0301

Line too long (%s/%s)

Unnecessary semicolon

paquetage *PythonPyLint*

3.4.10 Python0302

Too many lines in module (%s)

paquetage *PythonPyLint*

3.4.11 Python0303

Trailing whitespace

paquetage *PythonPyLint*

3.4.12 Python0304

Final newline missing

paquetage *PythonPyLint*

3.4.13 Python0321

More than one statement on a single line

Old : Format detection error in %r

paquetage *PythonPyLint*

3.4.14 Python0322

Old : Operator not preceded by a space

paquetage *PythonPyLint*

3.4.15 Python0323

Old : Operator not followed by a space

paquetage *PythonPyLint*

3.4.16 Python0324

Old : Comma not followed by a space

paquetage *PythonPyLint*

3.4.17 Python0325

Unnecessary parens after %r keyword

paquetage *PythonPyLint*

3.4.18 Python0326

%s space %s %s %sn%s

paquetage *PythonPyLint*

3.4.19 Python1001

Old-style class defined.

Use of __slots__ on an old style class

Use of “”property”” on an old style class

paquetage *PythonPyLint*

3.4.20 Python0001

(syntax error raised for a module ; message varies)

(error prevented analysis ; message varies)

Unable to run raw checkers on built-in module %s

paquetage *PythonPyLint*

3.4.21 Python0011

Unrecognized file option %r

Locally disabling %s

paquetage *PythonPyLint*

3.4.22 Python0012

Bad option value %r

Locally enabling %s

paquetage *PythonPyLint*

3.4.23 Python0100

__init__ method is a generator

paquetage *PythonPyLint*

3.4.24 Python0101

Explicit return in __init__

Unreachable code

paquetage *PythonPyLint*

3.4.25 Python0102

Black listed name “”%s””“

%s already defined line %s

Dangerous default value %s as argument

paquetage *PythonPyLint*

3.4.26 Python0103

Invalid %s name “”%s””“

%r not properly in loop

paquetage *PythonPyLint*

3.4.27 Python0104

Return outside function

Statement seems to have no effect

paquetage *PythonPyLint*

3.4.28 Python0105

Yield outside function

String statement has no effect

paquetage *PythonPyLint*

3.4.29 Python0106

Return with argument inside generator

Expression “”%s”” is assigned to nothing

paquetage *PythonPyLint*

3.4.30 Python0107

Use of the non-existent %s operator

Unnecessary pass statement

paquetage *PythonPyLint*

3.4.31 Python0108

Duplicate argument name %s in function definition

Lambda may not be necessary

paquetage *PythonPyLint*

3.4.32 Python0202

Class method %s should have cls as first argument

An attribute affected in %s line %s hide this method

Unable to check methods signature (%s / %s)

paquetage *PythonPyLint*

3.4.33 Python0203

Metaclass method %s should have mcs as first argument

Access to member %r before its definition line %s

paquetage *PythonPyLint*

3.4.34 Python0211

Method has no argument

Static method with %r as first argument

paquetage *PythonPyLint*

3.4.35 Python0213

Method should have “”self”” as first argument

paquetage *PythonPyLint*

3.4.36 Python0221

Interface resolved to %s is not a class

Arguments number differs from %s method

paquetage *PythonPyLint*

3.4.37 Python0222

Missing method %r from %s interface

Signature differs from %s method

paquetage *PythonPyLint*

3.4.38 Python0235

__exit__ must accept 3 arguments : type, value, traceback

paquetage *PythonPyLint*

3.4.39 Python0501

Old : Non ascii characters found but no encoding specified (PEP 263)

paquetage *PythonPyLint*

3.4.40 Python0502

Old : Wrong encoding specified (%s)

paquetage *PythonPyLint*

3.4.41 Python0503

Old : Unknown encoding specified (%s)

paquetage *PythonPyLint*

3.4.42 Python0601

Using variable %r before assignment

Global variable %r undefined at the module level

paquetage *PythonPyLint*

3.4.43 Python0602

Undefined variable %r

Using global for %r but no assigment is done

paquetage *PythonPyLint*

3.4.44 Python0603

Undefined variable name %r in __all__

Using the global statement

paquetage *PythonPyLint*

3.4.45 Python0604

Invalid object %r in __all__, must contain only strings

Using the global statement at the module level

paquetage *PythonPyLint*

3.4.46 Python0611

No name %r in module %r

Unused import %s

paquetage *PythonPyLint*

3.4.47 Python0701

Bad except clauses order (%s)

Raising a string exception

paquetage *PythonPyLint*

3.4.48 Python0702

Raising %s while only classes, instances or string are allowed

No exception type(s) specified

paquetage *PythonPyLint*

3.4.49 Python0710

Raising a new style class which doesn't inherit from BaseException

Exception doesn't inherit from standard ""Exception"" class

paquetage *PythonPyLint*

3.4.50 Python0711

NotImplemented raised - should raise NotImplementedError

Exception to catch is the result of a binary ""%s"" operation

paquetage *PythonPyLint*

3.4.51 Python0712

Catching an exception which doesn't inherit from BaseException : %s

Implicit unpacking of exceptions is not supported in Python 3

paquetage *PythonPyLint*

3.4.52 Python1001

Old-style class defined.

Use of __slots__ on an old style class

Use of ""property"" on an old style class

paquetage *PythonPyLint*

3.4.53 Python1002

Use of super on an old style class

paquetage *PythonPyLint*

3.4.54 Python1003

Bad first argument %r given to super()

paquetage *PythonPyLint*

3.4.55 Python1004

Missing argument to super()

paquetage *PythonPyLint*

3.4.56 Python1101

%s %r has no %r member

paquetage *PythonPyLint*

3.4.57 Python1102

%s is not callable

paquetage *PythonPyLint*

3.4.58 Python1103

%s %r has no %r member (but some types could not be inferred)

paquetage *PythonPyLint*

3.4.59 Python1111

Assigning to function call which doesn't return

Assigning to function call which only returns None

paquetage *PythonPyLint*

3.4.60 Python1120

No value passed for parameter %s in function call

paquetage *PythonPyLint*

3.4.61 Python1121

Too many positional arguments for function call

paquetage *PythonPyLint*

3.4.62 Python1122

Old : Duplicate keyword argument %r in function call

paquetage *PythonPyLint*

3.4.63 Python1123

Passing unexpected keyword argument %r in function call

paquetage *PythonPyLint*

3.4.64 Python1124

Parameter %r passed as both positional and keyword argument

paquetage *PythonPyLint*

3.4.65 Python1125

Old : Missing mandatory keyword argument %r

paquetage *PythonPyLint*

3.4.66 Python1200

Unsupported logging format character %r (%#02x) at index %d

paquetage *PythonPyLint*

3.4.67 Python1201

Logging format string ends in middle of conversion specifier

Specify string format arguments as logging function parameters

paquetage *PythonPyLint*

3.4.68 Python1205

Too many arguments for logging format string

paquetage *PythonPyLint*

3.4.69 Python1206

Not enough arguments for logging format string

paquetage *PythonPyLint*

3.4.70 Python1300

Unsupported format character %r (%#02x) at index %d

Format string dictionary key should be a string, not %s

paquetage *PythonPyLint*

3.4.71 Python1301

Format string ends in middle of conversion specifier

Unused key %r in format string dictionary

paquetage *PythonPyLint*

3.4.72 Python1302

Mixing named and unnamed conversion specifiers in format string

paquetage *PythonPyLint*

3.4.73 Python1303

Expected mapping for format string, not %s

paquetage *PythonPyLint*

3.4.74 Python1304

Missing key %r in format string dictionary

paquetage *PythonPyLint*

3.4.75 Python1305

Too many arguments for format string

paquetage *PythonPyLint*

3.4.76 Python1306

Not enough arguments for format string

paquetage *PythonPyLint*

3.4.77 Python1310

Suspicious argument in %s.%s call

paquetage *PythonPyLint*

3.4.78 Python0001

(syntax error raised for a module ; message varies)

(error prevented analysis ; message varies)

Unable to run raw checkers on built-in module %s

paquetage *PythonPyLint*

3.4.79 Python0002

%s : %s (message varies)

paquetage *PythonPyLint*

3.4.80 Python0003

ignored builtin module %s

paquetage *PythonPyLint*

3.4.81 Python0004

unexpected inferred value %s

paquetage *PythonPyLint*

3.4.82 Python0010

error while code parsing : %s

Unable to consider inline option %r

paquetage *PythonPyLint*

3.4.83 Python0202

Class method %s should have cls as first argument

An attribute affected in %s line %s hide this method

Unable to check methods signature (%s / %s)

paquetage *PythonPyLint*

3.4.84 Python0220

failed to resolve interfaces implemented by %s (%s)

paquetage *PythonPyLint*

3.4.85 Python0321

More than one statement on a single line

Old : Format detection error in %r

paquetage *PythonPyLint*

3.4.86 Python0401

Unable to import %s

Cyclic import (%s)

Wildcard import %s

paquetage *PythonPyLint*

3.4.87 Python0001

(syntax error raised for a module ; message varies)

(error prevented analysis ; message varies)

Unable to run raw checkers on built-in module %s

paquetage *PythonPyLint*

3.4.88 Python0010

error while code parsing : %s

Unable to consider inline option %r

paquetage *PythonPyLint*

3.4.89 Python0011

Unrecognized file option %r

Locally disabling %s

paquetage *PythonPyLint*

3.4.90 Python0012

Bad option value %r

Locally enabling %s

paquetage *PythonPyLint*

3.4.91 Python0013

Ignoring entire file

paquetage *PythonPyLint*

3.4.92 Python0014

Used deprecated directive “”pylint :disable-all”” or “”pylint :disable=all””“

paquetage *PythonPyLint*

3.4.93 Python0020

Suppressed %s (from line %d)

paquetage *PythonPyLint*

3.4.94 Python0021

Useless suppression of %s

paquetage *PythonPyLint*

3.4.95 Python0022

Deprecated pragma “”pylint :disable-msg”” or “”pylint :enable-msg””“

paquetage *PythonPyLint*

3.4.96 Python0201

Method could be a function

Attribute %r defined outside __init__

paquetage *PythonPyLint*

3.4.97 Python0401

Unable to import %s

Cyclic import (%s)

Wildcard import %s

paquetage *PythonPyLint*

3.4.98 Python0801

Similar lines in %s files

paquetage *PythonPyLint*

3.4.99 Python0901

Too many ancestors (%s/%s)

paquetage *PythonPyLint*

3.4.100 Python0902

Too many instance attributes (%s/%s)

paquetage *PythonPyLint*

3.4.101 Python0903

Too few public methods (%s/%s)

paquetage *PythonPyLint*

3.4.102 Python0904

Too many public methods (%s/%s)

paquetage *PythonPyLint*

3.4.103 Python0911

Too many return statements (%s/%s)

paquetage *PythonPyLint*

3.4.104 Python0912

Too many branches (%s/%s)

paquetage *PythonPyLint*

3.4.105 Python0913

Too many arguments (%s/%s)

paquetage *PythonPyLint*

3.4.106 Python0914

Too many local variables (%s/%s)

paquetage *PythonPyLint*

3.4.107 Python0915

Too many statements (%s/%s)

paquetage *PythonPyLint*

3.4.108 Python0921

Abstract class not referenced

paquetage *PythonPyLint*

3.4.109 Python0922

Abstract class is only referenced %s times

paquetage *PythonPyLint*

3.4.110 Python0923

Interface not implemented

paquetage *PythonPyLint*

3.4.111 Python0101

Explicit return in __init__

Unreachable code

paquetage *PythonPyLint*

3.4.112 Python0102

Black listed name “”%s”“

%s already defined line %s

Dangerous default value %s as argument

paquetage *PythonPyLint*

3.4.113 Python0104

Return outside function

Statement seems to have no effect

paquetage *PythonPyLint*

3.4.114 Python0105

Yield outside function

String statement has no effect

paquetage *PythonPyLint*

3.4.115 Python0106

Return with argument inside generator

Expression “”%s”” is assigned to nothing

paquetage *PythonPyLint*

3.4.116 Python0107

Use of the non-existent %s operator

Unnecessary pass statement

paquetage *PythonPyLint*

3.4.117 Python0108

Duplicate argument name %s in function definition

Lambda may not be necessary

paquetage *PythonPyLint*

3.4.118 Python0109

Duplicate key %r in dictionary

paquetage *PythonPyLint*

3.4.119 Python0110

map/filter on lambda could be replaced by comprehension

paquetage *PythonPyLint*

3.4.120 Python0120

Else clause on loop without a break statement

paquetage *PythonPyLint*

3.4.121 Python0121

Missing required attribute “”%s”“

Use raise ErrorClass(args) instead of raise ErrorClass, args.

paquetage *PythonPyLint*

3.4.122 Python0122

Use of exec

paquetage *PythonPyLint*

3.4.123 Python0141

Used builtin function %r

paquetage *PythonPyLint*

3.4.124 Python0142

Used * or ** magic

paquetage *PythonPyLint*

3.4.125 Python0150

%s statement in finally block may swallow exception

paquetage *PythonPyLint*

3.4.126 Python0199

Assert called on a 2-uple. Did you mean 'assert x,y' ?

paquetage *PythonPyLint*

3.4.127 Python0201

Method could be a function

Attribute %r defined outside __init__

paquetage *PythonPyLint*

3.4.128 Python0211

Method has no argument

Static method with %r as first argument

paquetage *PythonPyLint*

3.4.129 Python0212

Access to a protected member %s of a client class

paquetage *PythonPyLint*

3.4.130 Python0221

Interface resolved to %s is not a class

Arguments number differs from %s method

paquetage *PythonPyLint*

3.4.131 Python0222

Missing method %r from %s interface

Signature differs from %s method

paquetage *PythonPyLint*

3.4.132 Python0223

Method %r is abstract in class %r but is not overridden

paquetage *PythonPyLint*

3.4.133 Python0231

__init__ method from base class %r is not called

paquetage *PythonPyLint*

3.4.134 Python0232

Class has no __init__ method

paquetage *PythonPyLint*

3.4.135 Python0233

__init__ method from a non direct base class %r is called

paquetage *PythonPyLint*

3.4.136 Python0234

iter returns non-iterator

paquetage *PythonPyLint*

3.4.137 Python0301

Line too long (%s/%s)

Unnecessary semicolon

paquetage *PythonPyLint*

3.4.138 Python0311

Bad indentation. Found %s %s, expected %s

paquetage *PythonPyLint*

3.4.139 Python0312

Found indentation with %ss instead of %ss

paquetage *PythonPyLint*

3.4.140 Python0331

Use of the <> operator

paquetage *PythonPyLint*

3.4.141 Python0332

Use of “”l”” as long integer identifier

paquetage *PythonPyLint*

3.4.142 Python0333

Use of the ‘‘ operator

paquetage *PythonPyLint*

3.4.143 Python0401

Unable to import %s

Cyclic import (%s)

Wildcard import %s

paquetage *PythonPyLint*

3.4.144 Python0402

Uses of a deprecated module %r

paquetage *PythonPyLint*

3.4.145 Python0403

Relative import %r, should be %r

paquetage *PythonPyLint*

3.4.146 Python0404

Reimport %r (imported line %s)

paquetage *PythonPyLint*

3.4.147 Python0406

Module import itself

paquetage *PythonPyLint*

3.4.148 Python0410

__future__ import is not the first non docstring statement

paquetage *PythonPyLint*

3.4.149 Python0511

(warning notes in code comments ; message varies)

paquetage *PythonPyLint*

3.4.150 Python0512

Cannot decode using encoding “”%s””, unexpected byte at position %d

paquetage *PythonPyLint*

3.4.151 Python0601

Using variable %r before assignment

Global variable %r undefined at the module level

paquetage *PythonPyLint*

3.4.152 Python0602

Undefined variable %r

Using global for %r but no assigment is done

paquetage *PythonPyLint*

3.4.153 Python0603

Undefined variable name %r in __all__

Using the global statement

paquetage *PythonPyLint*

3.4.154 Python0604

Invalid object %r in __all__, must contain only strings

Using the global statement at the module level

paquetage *PythonPyLint*

3.4.155 Python0611

No name %r in module %r

Unused import %s

paquetage *PythonPyLint*

3.4.156 Python0612

Unused variable %r

paquetage *PythonPyLint*

3.4.157 Python0613

Unused argument %r

paquetage *PythonPyLint*

3.4.158 Python0614

Unused import %s from wildcard import

paquetage *PythonPyLint*

3.4.159 Python0621

Redefining name %r from outer scope (line %s)

paquetage *PythonPyLint*

3.4.160 Python0622

Redefining built-in %r

paquetage *PythonPyLint*

3.4.161 Python0623

Redefining name %r from %s in exception handler

paquetage *PythonPyLint*

3.4.162 Python0631

Using possibly undefined loop variable %r

paquetage *PythonPyLint*

3.4.163 Python0632

Possible unbalanced tuple unpacking with sequence%s : ...

paquetage *PythonPyLint*

3.4.164 Python0633

Attempting to unpack a non-sequence%s

paquetage *PythonPyLint*

3.4.165 Python0701

Bad except clauses order (%s)

Raising a string exception

paquetage *PythonPyLint*

3.4.166 Python0702

Raising %s while only classes, instances or string are allowed

No exception type(s) specified

paquetage *PythonPyLint*

3.4.167 Python0703

Catching too general exception %s

paquetage *PythonPyLint*

3.4.168 Python0704

Except doesn't do anything

paquetage *PythonPyLint*

3.4.169 Python0710

Raising a new style class which doesn't inherit from BaseException

Exception doesn't inherit from standard ""Exception"" class

paquetage *PythonPyLint*

3.4.170 Python0711

NotImplemented raised - should raise NotImplementedError

Exception to catch is the result of a binary ""%s"" operation

paquetage *PythonPyLint*

3.4.171 Python0712

Catching an exception which doesn't inherit from BaseException : %s

Implicit unpacking of exceptions is not supported in Python 3

paquetage *PythonPyLint*

3.4.172 Python1001

Old-style class defined.

Use of __slots__ on an old style class

Use of ""property"" on an old style class

paquetage *PythonPyLint*

3.4.173 Python1111

Assigning to function call which doesn't return

Assigning to function call which only returns None

paquetage *PythonPyLint*

3.4.174 Python1201

Logging format string ends in middle of conversion specifier

Specify string format arguments as logging function parameters

paquetage *PythonPyLint*

3.4.175 Python1300

Unsupported format character %r (%#02x) at index %d

Format string dictionary key should be a string, not %s

paquetage *PythonPyLint*

3.4.176 Python1301

Format string ends in middle of conversion specifier

Unused key %r in format string dictionary

paquetage *PythonPyLint*

3.4.177 Python1401

Anomalous backslash in string : '%s'. String constant might be missing an r prefix.

paquetage *PythonPyLint*

3.4.178 Python1402

Anomalous Unicode escape in byte string : '%s'. String constant might be missing an r or u prefix.

paquetage *PythonPyLint*

3.4.179 Python1501

""%s"" is not a valid mode for open.

paquetage *PythonPyLint*

CHAPITRE 4

Livraisons

4.1 Livrable

4.1.1 NomenclatureLivrable

Le nom d'un ou de plusieurs ressources livrées n'est pas conforme aux règles spécifiées (voir *PackagingLivrable*).

commentaire Les livraisons sont des points clés de la vie d'un produit logiciel et l'attention qui doit y être portée est extrême. Ne pas respecter des règles de nomenclature spécifiées auparavant est un problème important. D'une part cela montre que l'organisation productrice n'est pas capable de suivre des règles élémentaires, d'autre par cela rend impossible le traitement automatique des éléments livrés par l'organization cliente.

exemple S'il a été demandé de livrer un seul fichier sous la forme CyberKebab-GXXX-Y.pdf ou XXX est le numéro d'un groupe et Y le numéro de livraison, alors CyberKebab-G203-2.pdf est valide mais Cyberkebab_210.pdf ne l'est pas. Si l'organisation client doit gérer de multiples livraisons il est fort à parier que des scripts automatisent certaines parties. Ne pas respecter les conventions peut lier à des problèmes plus où moins importants.

paquetage *Livrable*

4.1.2 DelaiLivrable

Le ou les délais de livraison n'ont pas été respectés.

paquetage *Livrable*

4.1.3 FormatLivrable

Le format des ressources livrées n'est pas conforme aux attentes (voir *PackagingLivrable*).

paquetage *Livrable*

4.1.4 VerificationLivrable

Chaque élément constitutif du livrable doit absolument être vérifie avant la livraison et toutes les règles qualités applicables doivent être impérativement contrôlées.

commentaire La qualité des livrables reflête généralement la qualité de ce qui se fait dans une organisation. Le client recevant un livrable de mauvaise qualité doit non seulement faire face aux problèmes posés par cette absence de qualité mais aura de plus une mauvaise image de l'organisation.

exemple Il est inadmissible de fournir un document sans le relire.

paquetage *Livrable*

4.1.5 DescriptifLivrable

Le descriptif d'un ou plusieurs livrable est manquant, incomplet ou incohérent.

commentaire Dans le cas où un livrable composite est livré, c'est à dire que le livrable est formé de différents artefacts, par exemple rassemblés dans une archive, il est indispensable d'ajoindre un descriptif du contenu du livrable en mentionnant quels sont les artefacts livrés mais également les relations qui les lient. Ce descriptif peut prendre la forme d'un fichier "README", d'un manifeste, ou de tout autre artefact clairement identifiable.

paquetage *Livrable*

4.1.6 PackagingLivrable

Le packaging du livrable, c'est à dire la manière dont les différents artéfacts ou éléments on été assemblés et conditionnés ne correspond pas aux attentes, ne sont pas conforme aux éventuels critères spécifiés ou requière de la part du client un effort supplémentaire de conditionnement ou déconditionnement qui doit lui être épargné.

commentaire L'activité de packaging est à la charge du producteur et non pas à celle du client. Ce dernier est en droit est droit d'attendre un produit livré, assemblé, conditionné, et généralement directement utilisable. C'est le client qui connait mieux le produit qu'il livre, sa structure et ses composants, et c'est à lui que revient l'effort du packaging car cela fait partie intégrante de la production.

exemple Si un fichier .pdf est demandé ou une structure précise en terme de fichiers dans une archive .zip est demandé, il est absolument indispensable de respecter ces consignes et de livrer ce qui est demandé sous la forme demandée.

paquetage *Livrable*

4.1.7 NonLivre

Un ou des artefacts, ou des parties d'artefacts non pas été livrés et la livraison n'est donc pas conforme aux résultats attendus.

paquetage *Livrable*

4.1.8 Auteur

Le ou les auteurs du document, qu'il s'agisse de personnes physiques ou morale, ne sont pas indiquées clairement ou de manière appropriées.

paquetage *Livrable*

4.1.9 Copyright

Les indications de copyrights associées livrées sont inappropriées, trop imprécises ou manquantes, ou ne peuvent pas être clairement associées à une ou plusieurs des ressources livrées.

paquetage *Livrable*

4.1.10 DefautDejaMentionne

Un ou des défauts ont déjà été mentionnés dans un audit précédent et n'ont pas été corrigés ou amendés dans le livrable courant.

commentaire Cette situation est inacceptable car elle remet en cause le processus d'évolution et le principe même d'audit. Si les défauts détectés au cours des audits successives ne sont pas commentés, pris en compte ou corrigés, ils risquent d'être impossible de converger vers un produit final de qualité. Par ailleurs, les audits ayant un coût non négligeable, devoir redétecter des défauts déjà mentionnés constitue à la fois une perte de temps pour l'équipe qualité, mais marque également une dégradation de la confiance par rapport à la capacité de l'équipe de production de délivrer un produit final.

paquetage *Livrable*

4.1.11 Date

Une des dates mentionnées semble être incorrectes, non mise à jours, ou une date semble manquante.

paquetage *Livrable*

4.1.12 GestionDeVersions

La gestion des versions semble inexistante, instatisfaisante ou présente des défaults.

commentaire La gestion de versions est un des aspects essentiels pour la réussite des projets. La gestion de version est l'un des éléments essentiels pour passer du niveau initial et "chaotique" au niveau répétable du modèle CMM. Il existe de nombreux cas documentés de projets de grande envergure dont l'échec à été directement pu être directement et explicitement relié à l'absence d'une gestion de versions cohérente et systèmatique.

paquetage *Livrable*

4.1.13 VersionLivrable

L'identification de la version du livrable semble être manquant, incorrect ou inadapté au status de livrable.

commentaire Il est important de distinguer le système de versionnement pour les artefacts internes au projet (par exemple le code source, les modèles, etc), du système de versionnement utilisé pour les livraisons. Ce dernier système etant exposé à l'exterieur et visible par des tierces parties, un soin particulier doit être apportés aux interprétations pouvant être associés à ce système et aux identifiants correspondants. (voir *GestionDeVersions*)

paquetage *Livrable*

4.1.14 MiseAJourVersion

Un numéro de version est incorrect ou ne semble pas avoir mis à jour, ce qui est un problème essentiel du point de vue de la gestion de versions (voir *GestionDeVersions*).

paquetage *Livrable*

4.1.15 ResumeModifications

Le ou les artefacts devraient contenir un résumé des modifications apportées. Si c'est déjà le cas, le résumé pas assez structuré, trop ou pas assez précis, ou plus généralement inadapté au contexte courant.

commentaire Le ou les artefacts peuvent utilement comporter différents deltas in situ (cf. $Deltas), mais leur dissemination dans les artefacts et leur nombre rend généralement nécessaire l'ajout d'un résumé des modifications. Ce résumé peut de plus comporter des éléments décrivant l'intention des modifications, alors que les deltas sont généralement seulement des éléments factuels concernant les différences entre versions successives.

paquetage *Livrable*

4.1.16 Deltas

Les "deltas" entre versions ne sont pas indiqués de manière appropriée.

commentaire Dans le cadre de l'évolution d'un document et de relectures successives par exemple, il est nécessaire de mentionner quelles modifications ont été apportées. Contrairement au résumé des modifications (voir *ResumeModifications*) qui est localisé à un endroit pré-défini et qui peut mentionner l'intention des modifications, les deltas montrent ces modifications in situ dans le corps d'un ou de plusieurs artefacts (cf ResumeModifications). Concrétemment il s'agit de signaler les éléments ajoutés, modifiés ou supprimés. Différentes techniques peuvent être utilisées selon le granularité des éléments considérés et le type des d'artefacts considérés (voir *DeltasTextuels*)(voir *DeltasGraphiques*).

paquetage *Livrable*

4.1.17 DeltasTextuels

Les parties du texte ayant été ajoutées/supprimées/modifiées devraient être rendus explicites dans le corps du document ou du texte considéré.

commentaire Ceci se fait traditionnellement via du surlignage, des textes barrés, des "barres de marges", etc. Dans le cas de modifications plus importantes il peut être utile d'utiliser des balises de début et de fin d'ajout par exemple. Les editeurs de documents classiques tel qu'OpenOffice ou Word permettent propose généralement des options de "suivi des modifications".

paquetage *Livrable*

4.1.18 DeltasGraphiques

Les éléments d'un graphique ayant été ajoutés/supprimés/modifiés devraient être rendus explicites.

commentaire Utiliser par exemples des couleurs ou des notes associés aux éléments graphiques. Il peut être nécessaire de fournir une légende (par exemple en début de document ou dans un contexte global) pour que les conventions utilisées soient comprises de tous.

paquetage *Livrable*

CHAPITRE 5

Divers

Les paquetages de règles ci-dessous sont généralement orthogonaux au cycle de vie et peuvent être utilisés tout au long du projet.

5.1 TexteTechnique

5.1.1 Langage

Le texte comporte un ou plusieurs éléments de langages incorrects et/ou peu appropriés au contexte du document.

commentaire Le niveau d'exigence en terme de qualité des textes dépend des artéfacts et de leur status. Les textes figurant dans des livrables sont des éléments dépassant le contexte de la sphére proche de l'auteur et une attention plus importante doit être apportées aux différents éléments de langages.

paquetage *TexteTechnique*

5.1.2 Langues

Des éléments en différentes langues cohabitent au sein d'une même phrase, d'un même texte ou d'un même identificateur, sans pour autant que cela soit justifié.

paquetage *TexteTechnique*

5.1.3 Orthographe

Le texte comporte une ou plusieurs fautes d'orthographe.

commentaire (voir *Langage*).

paquetage *TexteTechnique*

5.1.4 Ponctuation

Les règles de ponctuation associées au langage utilisé ne sont pas respectées.

commentaire Pour la langue française voir par exemple l'url suivante http://www.la-ponctuation.com/

paquetage *TexteTechnique*

5.1.5 Grammaire

La grammaire du langage n'est pas respectée.

paquetage *TexteTechnique*

5.1.6 Style

Le style du texte est inappoprié.

exemple Par exemple le style peut être trop "télégraphique", trop verbeux, trop technique, etc.

paquetage *TexteTechnique*

5.1.7 Formatage

Le formatage du texte n'est pas adéquat.

paquetage *TexteTechnique*

5.1.8 Abbreviation

Le texte ou les identificateurs comportent une ou plusieurs abbréviations et/ou acronymes n'étant pas définis/nécessaires/compréhensibles et/ou indispensables.

commentaire La plus grosse difficulté consiste non pas à "taper" des textes ou des identificateurs dans des artefacts logiciels, mais plutot à comprendre ces artefacts et ces textes. Alors qu'une la "frappe" des caractères se fait une fois et ne pose aucun problème notament avec les environements modernes d'édition fournissant des mécanisms de "complétion", la lecture des textes ou identificateurs par de multiples parties prenantes est toujours associés à des problèmes de compréhension bien supérieur, sauf si les la liste exacte des abbréviations se trouvent dans le glossaire. En fait le seul cas où les abbrévations sont utiles est lorsque les noms deviennent beaucoup trop longs pour être identifiés visuellement ou apparaissent par exemple dans des formules donc de manière locale, contrainte et répétée. Dans tous les cas de figure, sauf cas trivial, les abbréviations doivent être définies dans le glossaire.

paquetage *TexteTechnique*

5.1.9 ArticleInDefini

Un article défini est utilisé (par exemple "le", "au", ...) sans que le ou les objets referencés soit clairement identifiés ou un article indéfini est utilisé (par exemple "un" , "des" , ...) alors que ce devrait être un article défini.

paquetage *TexteTechnique*

5.1.10 RerefenceAmbigue

Une référence ambigüe est faite à un objet. Ce peut être une référence via un article défini (e.g. "le"), une référence temporelle (p.e. "le jour", "auparavant"), etc.

paquetage *TexteTechnique*

5.1.11 Vocabulaire

L'utilisation des termes utilisés ne semble pas systèmatique ou il est difficile de déterminer quelles combinaisons de termes sont spécifiques au domaine considéré ou l'utilisation de synonymes et/ou de paraphrases est inadapté.

paquetage *TexteTechnique*

5.1.12 Glossaire

Un ou des termes potentiellement spécifiques à un domaine particulier sont utilisés sans que ceux-ci soient présentant dans un glossaire ou l'utilisation de ces termes dans le texte ne semble pas compatible avec la définition donnée dans le glossaire.

paquetage *TexteTechnique*

5.1.13 TermeMetier

Un ou des termes utilisés ne semble(nt) pas être conformes au vocabulaire utilisé par les experts membres du métier considéré ou un terme plus précis semble être disponible dans ce domaine.

paquetage *TexteTechnique*

5.1.14 Temps

Le temps (passé, présent, futur ...) ou la modalité (devoir, pouvoir, savoir, ...) associé à un ou plusieurs éléments de la phrase est inadapté, incorrect, flou, et/ou à préciser.

paquetage *TexteTechnique*

5.1.15 Contexte

Certains éléments du texte ne sont pas facilement interprétables en l'absence d'un contexte clairement défini ou la dépendance par rapport à ce contexte devrait être limitée.

paquetage *TexteTechnique*

5.1.16 PhraseMalConstruite

Une ou plusieurs phrases sont mal construites, trop longues, non achevées, contiennent trop d'imbrications, d'enchainements et/ou de références ambigües.

paquetage *TexteTechnique*

5.1.17 ImbricationInutile

Les éléments d'imbrications telles que les parenthèses, les guillemets, les tirets, les deux points, et autres types de ponctuations devraient être remplacés par des structures de phrases plus explicitant.

paquetage *TexteTechnique*

5.1.18 TexteSection

Les titres des sections et de sous-sections ne doivent s'enchainer sans qu'un texte d'introduction ou de liaison ne les séparent.

paquetage *TexteTechnique*

5.1.19 ReferenceTrigramme

L'utilisation systématique des trigrammes (voir *Trigramme*) est nécessaire à chaque fois qu'une référence est faite à une partie prenante.

paquetage *TexteTechnique*

5.1.20 Justification

L'information fournie n'est pas claire ou n'est pas justifiée.

paquetage *TexteTechnique*

5.1.21 Subjectif

Le texte fait référence à un ou des éléments pouvant être interpretée de manière subjective, imprécise et/ou non quantifiable.

paquetage *TexteTechnique*

5.1.22 Precision

Le texte comporte des termes flous ou trop imprécis par rapport au contexte du document. Par exemple le connecteur "ou" peut être interpretée comme étant exclusif ou non.

paquetage *TexteTechnique*

5.1.23 Redondance

Le texte comporte des éléments redondants entre eux ou par rapport à d'autres descriptions.

paquetage *TexteTechnique*

5.1.24 Paraphrase

Le texte comporte des paraphrases qui n'apportent rien, donne une impression de redite, ou le sentiment d'un certain malaise lié à l'envie de re-phraser des concepts non définis ou mal exprimés auparavant. Il peut aussi s'agir d'une figure ou d'un titre de figure qui ne fait que "rephraser" la même information sans plus value réelle.

paquetage *TexteTechnique*

5.1.25 Incoherence

Le texte comporte des éléments pouvant se révéler incohérents entre eux ou par rapport à d'autres descriptions.

paquetage *TexteTechnique*

5.1.26 Completude

Le texte est trop incomplet ou ne fait pas référence à tous les objets pertinents dans l'univers du discours.

paquetage *TexteTechnique*

5.1.27 Invalide

La texte fait référence à une propriété, un objet, ou un fait pouvant être jugé invalide ou irréaliste.

paquetage *TexteTechnique*

5.1.28 Homogeneite

Le texte fourni n'est pas homogène avec les descriptions suivantes et précédentes faisant références à des objets similaires.

paquetage *TexteTechnique*

5.1.29 Exemple

Le status d'exemple, d'illustration ou de cas général n'est pas explicite et/ou il est souhaitable de séparer de manière explicite les éléments qui releve de l'illustration ou du cas général.

paquetage *TexteTechnique*

5.1.30 Sujet

Le sujet de la phrase n'est pas clair, peu explicite ou erroné.

paquetage *TexteTechnique*

5.1.31 Complexite

La formulation de la phrase est inutilement complexe et peut être simplifée.

paquetage *TexteTechnique*

5.1.32 Interpretation

La phrase est difficile a interpretée et/ou peut être interpretée de manière inadéquate, erronée et/ou ambigüe.

paquetage *TexteTechnique*

5.1.33 NonAbstraction

Le texte ou le modèle comporte un ou plusieurs éléments faisant référence à des concepts ou objets correspondant à des niveaux d'abstractions différents et/ou trop détaillés. Le niveau d'abstraction devrait être homogène globalement et le fait que certaines parties soient très détaillées et d'autres plus abstraites pose problème si cela n'est pas justifié par ailleurs.

paquetage *TexteTechnique*

5.1.34 HypotheseNonValidee

Une hypothèse est faite implicitement ou explicitement sans pour autant que cette hypothèse ai été validée.

paquetage *TexteTechnique*

5.1.35 Pipe

Une confusion est faite entre la description/representation/identification d'un objet et cet objet lui même.

paquetage *TexteTechnique*

5.2 Nomenclature

5.2.1 Orthographeldentificateur

Une ou plusieurs fautes d'orthographes sont présents dans un ou plusieurs identificateurs.

commentaire La présence de fautes d'orthographes dans les identificateurs sont beaucoup plus importants que dans du texte. Dans du texte, seul la lecture est génée et l'auteur potentiellement décribilisé dans ca capacité de relire ou faire relire le texte qu'il a produit (et donc dans sa capacité à livrer des artefacts de qualité). La situation dans un identificateur est de toute autre nature, et le problème de plusieurs ordre de magnitude plus important. En effet les identificateurs sont fait pour être référencés, recherchés, dérivés, etc. et toute erreur qui s'introduit dans un identificateur risque d'avoir des impacts très lourds en l'absence par exemple de technique de "renommage" car toutes les occurrences de l'identificateur erronées devront être renommées avec tous les risques que cela présente. Il est possible que l'erreur ne soit pas corrigée lorsqu'elle est découverte pour éviter d'éventuels impacts. Rechercher et référencer des identificateurs avec des erreurs d'orthographes risquent de générer des erreurs en cascades, des problèmes de gestion des impacts, etc. La liaison entre les différents artefacts comme le code et le glossaire du domaine risque de ne pas pouvoir non plus être fait.

paquetage *Nomenclature*

5.2.2 Identificateur

Les identificateurs doivent être clairs, compréhensibles en dehors de leur contexte immédiat, doivent reflêter les objets auquels ils font références et ne pas constituer de paraphrases complexes de l'objet auquel ils font référence.

paquetage *Nomenclature*

5.2.3 FormeNominale

Une forme nominale doit être utilisée pour référencer l'objet considéré.

paquetage *Nomenclature*

5.2.4 FormeVerbale

Une forme verbale doit être utilisée pour référencer l'objet considéré.

paquetage *Nomenclature*

5.2.5 Generique

Le ou les termes utilisés sont trop génériques et ne fournissent pas d'information ou des termes plus spécifiques sont peut être disponibles dans le vocabulaire du domaine.

paquetage *Nomenclature*

5.2.6 Connecteur

Les connecteurs tel que "et", "ou", "/", "+", signes de ponctuations ou d'imbrications ne devraient pas être utilisés dans un identificateur dans la mesure ou l'objet identifié n'est pas clairement conceptualisé ou nommé.

commentaire un identificateur correspond normallement à un concept ou à une entitée particulière définie et il existe généralement un terme décrivant ce concept, en tout cas dans un vocabulaire métier ou dans un jargon particulier. Si ce n'est pas le cas on peut se poser la question de la réalité ou de l'utilité ou de la réalité de ce concept. Si le concept est effectivement utile, dans ce cas il est généralement préférable de l'associer à un mot existant ou à l'une de ses dérivations (et à ajouter ces termes dans le glossaire), plutot que d'introduire des connecteurs. Très souvent l'utilisation de connecteurs correspond à une justaposition non réflechie d'éléments. Un identificateur composé par des connecteurs peut correspondre également à des pratiques de programmation ou de modélisation problématiques qu'il s'agira soit d'éliminer, soit de documenter avec soin.

paquetage *Nomenclature*

5.2.7 HomogeneiteIdentificateurs

Les identificateurs utilisés ne sont globalement pas homogènes et soit il existe une absence totale de style, soit trop de styles sont utilisés sans que cela soit justifié.

commentaire Tous les identificateurs d'un même genre (e.g. identificateurs de classes, de scénarios, de cas d'utilisation) devraient être homogènes et respecter des règles de nomenclature portant à la fois sur le plan de la typographie (utilisation de minuscules, majuscules, soulignés ou tirets, etc.), de l'ensemble des caractères utilisés (il est généralement recommandé de ne pas utilisés d'accents ou d'autres caractères diacritiques), des formes grammaticales utilisées (par exemple des formes verbales au passif et au participe présent ne sont pas homogénes), des connecteurs (e.g. des articles) et abbréviations utilisés ou non. Le manque d'homogénéité peut avoir des impacts néfastes sur la lecture, la possibilité de référencer de manière systèmatique des éléménts, la possibilité de faire des recherches textuelles d'identificateurs. Elle met également en péril toute possibilité d'automatisation, d'extraction d'information, de référencement, etc.

exemple "supprimer employé" et "CreationDUnePers" ne sont pas homogènes car ils diffèrent par raport à (1) la casse, (2) l'ensemble des caractères utilisés, (3) le fait d'utiliser des articles ou non, (4) la forme grammaticale mise en oeuvre (infinitif vs. nom), (5) l'utilisation ou la suppression des articles, (6) l'utilisation d'abbréviation.

paquetage *Nomenclature*

5.2.8 Trigramme

Un trigramme est une séquence de trois lettres majsucules faisant référence de manière unique à une personne dans un *contexte* donné. La règle appliquée est de concaténer

1. la première lettre du premier prénom,
2. la première lettre du premier nom de famille, et

3. la dernière lettre du premier nom de famille. Les particules nobiliaires et aux prépositions nominales (cf wikipedia) ne sont pas considérées dans cette règle (e.g. selon les langues 'van der', 'von', 'de', 'dit', 'le', 'von', 'el', etc.)

Si le trigramme formé avec les règles ci-dessus est déjà utilisé dans le contexte considéré, l'avant dernière lettre du nom est utilisée en place de la dernière et ainsi de suite.

observation Dans les projets informatiques les parties prenantes (stakeholders en anglais) sont souvent identifiées de manière unique par un trigramme identifiant la personne de manière unique. Il existe plusieurs règles selon les enterprises, mais l'objectif est toujours de minimiser la probabilité d'avoir deux personnes ayant par défault le même trigramme (auquels cas une autre règle est appliquée pour la sectond personne). Les trigrammes sont utilisés de manière ubiquitaire dans les projets et autre autre dans les comptes rendus de réunions, les documents, les méls, le code source, les fichiers de suivis de temps, de gestion de projets, etc.

exemple Le trigramme de Djiamila Maria WONG CONNOR est DWG.

Le trigramme de Jean Baptiste VON DER WECK PILOW est JWK car *von der* est une particule nobiliaire en allemand.

paquetage *Nomenclature*

5.2.9 UtilisationTrigramme

Un trigramme (voir *Trigramme*) doit être utilisé pour referencer une partie prenante.

paquetage *Nomenclature*

5.2.10 Portrait

Chaque partie prenante doit être identifiée visuellement pas un portrait unique la représentant de face ou de profil mais permettant son identification sans ambiguité. Sont donc à proscrire tout icones, graphiques, ou représentation de personnages fictifs ne correspondant pas à la partie prenante.

commentaire Dans un monde professionnel, les entreprises maintiennent traditionnelement un "trombinoscope" plus ou moins formels selon son usage et l'entité qui le gère (équipe, niveau global de la corporation, direction des ressources humaines, etc). Dans le cadre d'organisations complexes, d'organisations virtuelles ou de projets globaux géographiquement répartis, pouvoir identifier les différentes parties prenantes et les différents interlocuteurs prenant part à des activités collaboratives est particulièrement important. De la même manière qu'aller travailler avec un masque de tortue ninja n'est généralement pas considéré comme faisant partie des pratiques professionnelles, se cacher derrière un tel avatar ou la représentation d'un nounours ne répond ni besoin de communication de l'organisation, ni à une image de professionalisme que devrait afficher toutes les parties prenantes. Par ailleurs, cette image pourrait être réutilisée contre

la personne ayant déposé ce portrait dans le cadre d'une réunion importante, par exemple

pour décribiliser l'entreprise entière.

commentaire La manière de mettre à jour son portrait dépend de chaque système utilisé Dans moodle cette information se trouve dans la section `règlage de mon profil`.

paquetage *Nomenclature*

5.2.11 NomPersonne

Chaque personne est identifiée par son (ou ses) prénom(s) d'usage suivi et de son (ou ses) nom(s) d'usage orthographiés systématiquement de la même manière et séparés systématiquement par la même ponctuation. Pour distinguer le (ou les) nom(s) ceux-ci sont écrits en majuscules.

commentaire Lorsque nécessaire, et si un champ n'est pas prévu spécifiquement à cet effet, l'utilisation de trigramme se fera après chaque partie prenante entre parenthèses.

exemple "Djiamila Maria WONG CONNOR (DWG)".

exemple "Amelia Perdita DA SILVA PEREZ (ASA)".

exemple "Jean-Marie FAVRE DIT CROTIN" (JFE)".

commentaire Si l'outil utilisé ne comporte pas de champs spécifique pour le trigramme celui-ci peut etre mis dans le champs nom entre parenthèse. Par exemple "Amelia Perdita" sera dans le champ "prénom" (s'il existe) et "DA SILVA PEREZ (ASA)" dans le champ "nom".

commentaire La manière de mettre à jour son identité dépend de chaque système utilisé Dans moodle cette information se trouve dans la section `règlage de mon profil`.

paquetage *Nomenclature*

5.2.12 FormatDate

paquetage *Nomenclature*

5.2.13 IdASCII

Les identificateurs ne doivent comporter que des caractères ASCII et les accents sont donc à proscrire.

commentaire Dans le cadre de transformation de modèles ou de génération de code cette règle est essentielle car de nombreux outils et langages gère de manière indaptée les accentes par exemple.

paquetage *Nomenclature*

5.2.14 MajMin

L'identificateur doit correspondre à une suite de caractères ASCII (voir *IdASCII*) formés de majuscules, minuscules ou chiffres, débutant par une majuscule.

commentaire L'identificateur ne doit comporter ni espaces, ni accents, ni délimiteurs.

exemple "ConnecteurDInterface2", "SMSRenvoye"

paquetage *Nomenclature*

5.2.15 MinMaj

L'identificateur doit correspondre à une suite de caractères ASCII (voir *IdASCII*) formés de majuscules, minuscules ou chiffres, débutant par une minuscule.

commentaire L'identificateur ne doit comporter ni espaces, ni accents, ni délimiteurs.

exemple "lesConnecteurs", "smsRenvoye2", "les4SMSRecus"

paquetage *Nomenclature*

5.2.16 MinMajSouligne

L'identificateur doit correspondre à une suite de majuscules, minuscules, chiffres ou souligné ("_"), débutant par une minuscule.

commentaire L'identificateur ne doit comporter ni espaces, ni accents, ni délimiteurs autre que le souligné "_".

exemple "acheterUnTicket_normal"

paquetage *Nomenclature*

5.2.17 MAJSouligneMAJ

L'identificateur doit correspondre à une suite séquences de majuscules, chiffres et soulignés ("_").

commentaire L'identificateur ne doit comporter ni espaces, ni accents, ni délimiteurs autre que le souligné "_".

exemple "CONST_WINDOW_CLOSED"

paquetage *Nomenclature*

5.2.18 StyleSIdentificateur

Différents styles d'intentificateurs sont utilisés sans pour autant que l'on puisse déterminer dans quelles conditions ces styles varient, s'ils sont utilisés de manière consistentes ou non. C'est le cas par exemple lorsque certains indentificateurs sont issues à la fois de styles MajMin, MinMaj, MAJSouligneMAJ etc, ou dans toutes autres combinaisons ad-hoc.

paquetage *Nomenclature*

5.2.19 RoleDansPatron

Le role joué par un objet ou une classe dans le patron n'est pas facilement identifiable.

paquetage *Nomenclature*

5.2.20 InteractionProscrite

Une ou des interactions entre couches ne sont pas conformes aux règles établies par le patron.

commentaire Dans certaines versions du patron MVC les controleurs jouent les intermediaires entre les modeles et les vues et les interactions directes entre ces couches sont interdites. Les modèles doivent être complétement indépendants des autres couches et donc ne connaître ni les controleurs, ni les vues mais peuvent intégagir entre eux. Les vues ou interfaces, qu'elles correspondent à des dispositifs d'entrée, de sorties, à des actuateurs ou à des capteurs, peuvent intéragir entre elles ou avec des controleurs. Les controleurs peuvent intéragir avec les controleurs, les vues et les modèles et jouent donc un rôle central.

paquetage *Nomenclature*

5.3 Diagramme

5.3.1 NomDiagramme

Le nom des diagrammes doit reflêter ce qu'ils modélisent et peuvent donc utilement faire référence à un modèle, à un artéfact, etc. Le type de diagramme (voir *TypeDeDiagramme*) peut également être utilement inséré dans ce titre.

commentaire Le type de diagramme est peut généralement être aisement inféré en regardant le diagramme, mais si le nom du diagramme est utilisé comme titre de figure et que ce dernier est dans une liste de figure, il est intéressant de disposer de cette information. Le modèle ou artéfact auquel fait référence le diagramme est parfois facile a inférer via le contexte dans lequel le diagramme est disposé, mais hors de ce contexte cette information est perdue et il est donc essentiel d'indiquer "à quoi" correspond le diagramme.

paquetage *Diagramme*

5.3.2 TitreDiagramme

(voir *NomDiagramme*) TODO : to be removed

paquetage *Diagramme*

5.3.3 FigureLibrePourModele

Ne jamais utiliser de "figures libres" avec l'intention de créer des éléments de modèles.

commentaire Certains éditeurs de modèles propose dans leur palette la création libre de figure. Il ne faut jamais utiliser ces artifices graphiques car de tels artifices ne font pas parties du modèle, mais juste du diagramme et sont donc complètement ignorés dans le modèle et par tous les outils qui exploitent ce modèle (génération de documentation, génération de code, etc.)

exemple Dans le logiciel modelio, la palette nommée "Free Drawing" permet de dessiner des formes de bases mais cela n'a strictement aucun effet sur les modèles produits avec modelio.

paquetage *Diagramme*

5.3.4 DensiteDiagramme

La densite des éléments dans le diagramme est soit trop importante soit insuffisante et le diagramme pourrait utilement être compacté, étendu ou décomposé en différents diagrammes.

commentaire Une erreur classique pour les novices en modélisation est de vouloir "tout représenter" sur un même diagramme. Cette approche ne fonctionne pas du tout pour des systèmes de tailles normales et différents diagrammes doivent alors être fait. Les outils de modélisations actuels permettent de créer facilement différents diagrammes et de les maintenir synchronisés.

paquetage *Diagramme*

5.3.5 Disposition

La disposition spatiale des différents éléments graphiques dans le diagramme n'est pas conventionelle, nuit à la lisibilité du diagramme et/ou devrait être améliorée ou optimisée.

paquetage *Diagramme*

5.3.6 Couleurs

L'utilisation des couleurs n'est pas optimale et pourrait être améliorée soit en diminuant, soit en augmentant le nombre des couleurs, soit en rendant explicites leur signification dans le diagramme par exemple via une note.

paquetage *Diagramme*

5.3.7 Chevauchements

De nombreux chevauchements d'éléments graphiques rendent la lecture du diagramme difficile.

paquetage *Diagramme*

5.3.8 Surcharge

Le diagramme comporte trop d'éléments graphiques et/ou textuels.

paquetage *Diagramme*

5.3.9 ContenuPauvre

Le contenu du diagramme est trop pauvre pour que ce dernier soit réellement pertinent. Soit le diagramme manque de détails soit l'existence du diagramme ou plus simplement son indroduction dans un document pourrait être mise en cause ; c'est le cas si l'information contenue dans le diagramme peut être dérivée à partir d'autres éléments déjà présents dans le document et d'une certaine manière "n'apporte rien".

paquetage *Diagramme*

5.3.10 ContenuHeterogene

Le contenu du diagramme est hétérogène et il n'est pas facile de comprendre quelle est la cohérence entre les différents éléments présentés.

commentaire Dans le cas de modèle non triviaux, un même modèle peut comporter trop d'élément pour être représenté graphiquement en un seul diagramme est il est donc souhaitable de fournir plusieurs vues sur le modèles sous la forme de différents diagrammes. Chaque vue doit être consistente et correspondre à une intention particulière. La répartition des éléments dans les différents diagrammes doivent pouvoir être justifié.

exemple Si un modèle de cas d'utilisation est complexe, différents diagrammes de cas d'utilisation doivent certainement être créés. La manière de regrouper les différents cas d'utilisation en diagrammes doit pouvoir être justifié.

paquetage *Diagramme*

5.3.11 TypeDeDiagramme

Le type de diagramme n'est pas explicite.

paquetage *Diagramme*

5.4 Tracabilite

5.4.1 FormatReferenceLignes

La référence à une ligne <L> d'une ressource <R> se fait de la manière suivante : [<R>#<L>]. Plusieurs lignes d'une même ressources peuvent être séparées par des virgules, et un interval de lignes peut être constitué en utilisant un '-'. Plusieurs ressources différentes peuvent être séparées par un point virgule.

commentaire Les espaces ne sont pas autorisés.

exemple [CR001#2,4-5 ;CR002#34] est équivalent à [CR001#2][CR001#4][CR001#5][CB002#34]

paquetage *Tracabilite*

5.4.2 CUExigenceFonctionnelle

Un cas d'exigence doit être justifié par au moins un exigence fonctionnelle.

paquetage *Tracabilite*

5.4.3 ExigenceFonctionnelleCU

Un exigence fonctionnelle peut donner lieu généralement à un seul cas d'utilisation. Si ce n'est pas le cas (si plusieurs cas d'utilisation sont associés à l'exigence fonctionnelle), il est important de vérifier s'il ne s'agit pas d'une erreur ou d'un manque de précision dans la définition de l'exigence fonctionnelle.

paquetage *Tracabilite*

5.4.4 CURoleExigences

Le role joué par les exigences reliées au cas d'utilisation n'est pas clair, et il n'est par exemple pas facile de déterminer quelles sont les types des exigences via leur nom, quelles sont l'exigence fonctionnelle principale réalisée par le cas d'utilisation, etc.

paquetage *Tracabilite*

5.5 Document

5.5.1 EnteteDocument

Le titre, sous titre, ou plus généralement l'identification du document est manquant, inapproprié ou non conforme.

commentaire Dans certains contextes l'entête du document doit suivre un certain format et standard imposé par la structure dans laquelle ce document est produit et/ou évalué.

exemple Pour un rapport de stage, on s'attend à trouver le nom du stagiaire, l'entreprise d'accueil, la période du stage, le contexte dans lequel il s'est déroulé, le titre ou l'identificateur du projet, etc.

exemple Pour une thèse de doctorat, le format est généralement imposé par l'université d'accueil et l'entête du document doit être conforme aux règles établies.

paquetage *Document*

5.5.2 TableDesMatieres

Le plan du document doit être explicité par une table des matières.

paquetage *Document*

5.5.3 PlanDesequilibre

Le plan est deséquilibré soit en nombre de pages (voir *PlanDesequilibreEnPages*), soit en termes de profondeur (voir *PlanDesequilibreEnProfondeur*).

paquetage *Document*

5.5.4 PlanDesequilibreEnPages

Le plan du document doit être mieux équilibré en terme de longueur relative des sections en termes de pages.

commentaire Dans la pluspart des documents les sections rédigées qui constituent le corps du document doivent être de taille relativement similaire en nombre de pages. Sont exclues de cette règle les sections particulières comme les annexes, les introductions, les conclusions, les sections techniques telles que les abbréviations, les sections automatiquement générées par un outil, etc.

commentaire Lors de l'évaluation d'un plan (et plus généralement d'un document), vérifier que le plan est équilibré est une opération aisé. Ce défaut sera donc souvent détecté si l'on n'y prend garde.

exemple Sur un document de 70 pages on évitera par exemple d'avoir une section 3 rédigée de 50 pages (section 3) suivie d'une section 4 de 6 pages car cela reflète souvent une mauvaise organisation du contenu. Ici la section 3 représente plus des 2/3 du documents et elle devrait sans doute être scindée. Les sous sections 3.1, 3.2, 3.3 pourrait être "remontées" d'un niveau (e.g. 3, 4, 5), quitte à ajouter auparavant une section expliquant le contenu de chacune de ces sections. Une telle opération peut régler les problèmes associés à un plan trop profond (voir *PlanTropProfond*) ou à un plan déséqulibré en profondeur (voir *PlanDesequilibreEnProfondeur*).

paquetage *Document*

5.5.5 PlanDesequilibreEnProfondeur

La hierarchie des sections et sous sections n'est pas suffisemment "balancée" et certaines sous sections sont par exemple profondes alors que d'autres sont très plates.

exemple La section 2 comporte 2.1 et 2.2 alors que la section 3 comporte des sous sections telles que 3.1.2.1.a

commentaire ce défaut survient souvent comme une conséquence d'un plan déséquilibré en nombre de pages (voir *PlanDesequilibreEnPages*).

paquetage *Document*

5.5.6 PlanTropProfond

Le plan du document tel qu'il est présenté révèle le document dans une trop grande profondeur.

exemple Le plan montre des sections telles que 2.4.2.3.2.a. Même si toute les sections atteignent ce niveau de profondeur, celle-ci est trop importante.

commentaire Les traitements de textes permettent généralement de limiter le nombre de niveaux affichés dans le plan du document. Via ce mécanisme de filtrage, le document peut comporter des sous sections profondes (voir *SectionTropProfonde*) sans que le plan soit lui même trop profond.

commentaire Pour une lecture du plan aisée (voir *LecturePlan*) on ne devrait pas afficher plus de 2 ou 3 niveaux de profondeurs dans les sections.

commentaire Si le document est un document de référence, cette règle ne s'applique peut être pas car le plan peut faire office d'index et peut être utilisé pour montrer l'intégralité des sous sections du document et des concepts associés.

paquetage *Document*

5.5.7 SectionTropProfonde

Le document comporte une ou des sections trop profondes.

exemple S'il ne s'agit pas d'un document de référence, une section 2.4.2.3.2.a reflête certainement une structuration trop profonde.

commentaire Le plan du document peut masquer des sections profondes (voir *PlanTropProfond*).

paquetage *Document*

5.5.8 SectionOrpheline

Une sous section ne peut pas être seule à l'intérieure d'une section.

exemple Dans la section 2.3 la section 2.3.1, si elle existe, ne peut être seule. On devrait avoir une sous section 2.3.2 et eventuellement d'autres sous-sections au même niveau (e.g. 2.3.3, 2.3.4, etc.).

paquetage *Document*

5.5.9 LecturePlan

Un ou plusieurs defauts rendent le plan difficilement "lisible".

commentaire Le plan décrit l'architecture du document et doit rendre très explicite à la fois sa structure, mais aussi via les différents termes utilisés dans les titres des sous sections, les concepts intervenants dans le document.

paquetage *Document*

5.5.10 HomogeneiteTitreSection

Les titres des sections ne sont pas homogénes.

exemple La présence ou non d'articles doit être uniforme entre sections similaires. Ce n'est pas le cas ici pour les titres suivants : "3.1 Conception", "3.2 La réalisation", "3.3 Test de l'application".

paquetage *Document*

5.5.11 TitreSectionNeutre

Le titre d'une ou plusieurs sections n'est pas neutre et comporte par exemple une forme interrogative ou affirmative.

exemple "3.2 Comment le logiciel a été deployé ?"

commentaire Les formes interrogatives réthoriques sont généralement à proscrire dans les documents techniques.

paquetage *Document*

5.5.12 TitreHorsContexte

Le titre d'une section ou plusieurs sections sont difficiles à comprendre hors contexte ou dans le seul contexte du plan.

commentaire il est généralement préférable d'éliminer l'utilisation de sigles dans le titre d'une section si ce sigle n'a pas été défini dans le résumé du document ou à un niveau global. La lecture du plan est en effet rendue plus difficile (voir *LecturePlan*) alors que l'on devrait pouvoir à partir du plan comprendre l'architecture et le contenu global du document.

exemple "3.2 Intégration à UOP" pourrait être remplacée par "Intégration dans l'Unité Opérationelle de Planification (UOP)".

paquetage *Document*

5.5.13 NumerotationSection

La numérotation des sections comporte un ou plusieurs défauts.

exemple 2.3.a suivi de 2.3.2

paquetage *Document*

5.5.14 IndexDesFigures

Un index des figures doit être inclu dans le document.

paquetage *Document*

5.5.15 TitreFigure

Une ou des figures n'ont pas de titres ou leurs titres ne sont pas appropriés, ou explicite par exemple parceque le titre de la figure sera difficile à interpréter dans l'index des figures par exemple.

paquetage *Document*

5.5.16 DescriptionFigure

Une ou des figures ne sont pas documentée(s) ou décrite(s) ; il semble utile de décrire pourquoi telle ou telle figure est présentée, quels sont les éléments qui y sont représentés, pourquoi ceux-ci ont été séléctionnés, etc.

paquetage *Document*

5.5.17 ReferenceFigure

Une ou plusieurs figures ne sont pas référencées dans le texte.

paquetage *Document*

5.5.18 LegendeFigure

Les symboles ou conventions utilisées dans la où les figures ne sont pas explicités et une légende pourrait remédier à ce problème, ou si une légende est présente celle-ci n'est pas adéquate ou complète.

paquetage *Document*

5.5.19 TailleFigure

Certains éléments de la figure sont inadaptés et sont soit trop gros, soit trop petits, nuisant ainsi à la lisibilité de la figure.

paquetage *Document*

5.5.20 ResolutionFigure

La résolution de l'image ou de la figure n'est pas satisfaisante.

paquetage *Document*

5.5.21 IndexDesTables

Un index des tables doit être inclu dans le document.

paquetage *Document*

5.6 Presentation

5.6.1 TransparentsNumerotes

Tout les transparents doivent être numérotés.

commentaire Cela est indispensable entre autre pour que l'auditoire puisse référencer les transparents de manière unique en cas de questions à la fin. Nécessaire également pour prendre des notes lorsqu'un élément sur un trasparent est discuté.

paquetage *Presentation*

5.6.2 TransparentsLisibles

Sauf cas particulier tous les transparents doivent se baser sur une fonte suffisemment grande pour être lisible. Cette règle s'applique aussi bien aux textes, qu'aux diagrammes ou images.

commentaire Des transparents avec des fontes très petites peuvent très rapidement frustrer l'auditoire.

commentaire Si nécessaire prévoir des 'zooms' sur telle ou telle partie.

paquetage *Presentation*

5.6.3 TransparentsUniformes

Les différents transparents d'une présentation doivent présenter une certaine uniformité, entre autre dans le style utilisé.

commentaire Cette règle s'applique dans tous les cas. Même si plusieurs personnes interviennent et produisent des transparents. S'il s'agit d'une présentation de groupe il doit virtuellement être impossible, en terme de style, de savoir quel personne à fait quel transparent.

paquetage *Presentation*

5.6.4 TempsPresentation

Dans la pluspart des contextes il est important de réaliser la présentation en un temps donné, possiblement à une ou deux minutes près dans certains contextes. Cela peut inclure une éventuelle séance de question.

commentaire En pratique il est indispensable de faire un ou plusieurs répétitions en chronométrant le temps potentiel passé sur chaque transparent.

paquetage *Presentation*

5.6.5 ControleDuTempsPresentation

Le temps de présentation doit être géré par le ou les présentateurs. Le rôle de "gardien du temps' incombe soit à l'auditoire, soit au groupe qui présente, soit à une autre personne, mais ce rôle existe plus où moins toujours et il est indispensable de savoir qui va jouer se role avec quelles contraintes.

commentaire Les présentateurs doivent toujours avoir 'un oeil' sur le temps qui passe (via une montre ou autre) pour ne pas risquer de se faire couper brutalement voir ensuite violamment.

commentaire Ce contrôle du temps s'applique non seulement à la presentation mais aussi aux discussions et questions.

paquetage *Presentation*

5.6.6 InitialisationPresentation

Quel que soit le contexte il est indispensable d'être prêt avant l'heure de présentation avec soit un ordinateur portable allumé, présentation prête, soit un support contenant la présentation dans différents formats standard. Il est nécessaire de pouvoir débuter la présentation en moins de une ou deux minutes montre en main.

commentaire si apporter un portable allumé n'est pas possible, il est préférable de se synchroniser avec les présentateurs précédents le cas échéant, de manière à installer la présentation sur leur machine auparavant.

commentaire s'il les autres techniques ne sont pas possible il est nécessaire d'apporter une ou deux clés USB, avec la présentation en différents formats, dont .pdf car il s'agit sans doute du seul format qui sera sur la machine de présentation avec une probabilité assez importante.

paquetage *Presentation*

5.6.7 MaterielPresentation

Il peut selon les cas être nécessaire de fournir à l'auditoire, en début de présentation, ou auparavant, du matériel pour que les personnes concernées puissent suivre la présentation et éventuellement consulter le matériel fourni.

paquetage *Presentation*

5.6.8 NomenclaturePresentation

Tout comme dans les autres artefacts produits dans le cadre d'un projet, il est nécessaire pendant la présentation d'utiliser des références précises en utilisant la nomenclature établie dans le projet.

commentaire Il s'agit par exemple d'utiliser des trigrammes, des références vers des LOTs, vers des objets de modélisation. La rigueur est la précision est indispensable. Sans cela l'auditoire ne pourra pas par exemple chercher dans tel ou tel matériel l'artéfacts auquel il est faire référence.

commentaire Si les références ne sont pas 'lisibles' ou facilement interprétable par l'audience, il est alors nécéssaire d'utiliser à la fois un label (pour les humains) et la référence (pour un traitement plus automatique). Par exemple Arbia PANNANOTIS (APS) combine les deux.

commentaire Une présentation étant un artéfact du projet à part entière, celle-ci peut être indexée, etc. L'absence de référence précise rend inopérante tout processus de recherche et d'automatisation.

commentaire Dans certains contexte il peut être utile de founrir à l'auditoire un index des références utilisées dans la présentation. Cela permet entre autre des discussions précises levant toute ambiguité quand aux entités référencées. Dans le cas contraire il sera toujours possible à quelqu'un d'argumenter que ce n'est pas de tel ou tel LOT par exemple qui était là surce des discussions.

paquetage *Presentation*

CHAPITRE 6

778 REGLES DANS 29 PAQUETAGES

Avertissement : Ces informations ne sont actuellement pas collectées automatiquement. Elle peuivent ne pas être à jour.

29 paquetages triés par ordre alphabétique.

- *BaseDeDonnees* (10 rules)
- *CasDUtilisation* (17 rules)
- *Classe* (24 rules)
- *Deploiement* (1 rules)
- *Diagramme* (10 rules)
- *Document* (21 rules)
- *Etat* (21 rules)
- *Exigence* (13 rules)
- *Glossaire* (18 rules)
- *JavaCheckStyle* (147 rules)
- *Livrable* (17 rules)
- *Nomenclature* (19 rules)
- *ProgrammationWeb* (2 rules)
- *PythonPyLint* (179 rules)
- *Scenario* (23 rules)
- *Sequence* (1 rules)
- *Systeme* (5 rules)
- *Tache* (5 rules)
- *TexteTechnique* (35 rules)
- *Tracabilite* (3 rules)
- *UMLModelio* (187 rules)
- *UMLStarUML* (38 rules)
- *Valeur* (17 rules)