PECOS 4SMEs

PERSONALIZED TRAINING ON CROSS BORDER e-COMMERCE

PECOS4SMEs

D1.5 Standards and Guidelines

Version 1.0 – 27/02/2013

Project	PECOS4SMEs		
Author(s)	FC		
Reviewer(s)	CCS	DMAR	

This project has been funded with support from the European Commission.

Circulation List

Person Name	Organization Name
Gianluca Coppola	Eurocrea Merchant
Christos Anthis	CrystalClearSoft
Onno Hansen	OHENNENNOH BV
Kenny Payne	OAKE Associates
Robert Sanders	European Business & Innovation Centre Network (EBN)
Małgorzata Mikłosz	DANMAR COMPUTERS
Dimitris Diamantis	FAVINOM Consultancies

Revision History

Version	Date	Author	Description	Action	Pages
0.1	27/02/2013	FC,DMAR,CCS	Creation of the document	C	15

(*) Action: C = Creation, I = Insert, U = Update, R = Replace, D = Delete

Referenced Documents

ID	Reference	Title
1	507562-LLP-2012-GR-Leonardo-LMP	PECOS4SMEs Proposal
2	507562-LLP-2012-GR-Leonardo-LMP	Evaluation Comments

Applicable Documents

ID	Reference	Title
1	FAVINOM Consultancies QMS	Quality Management Procedures

Executive Summary

This document forms the D1.5 Standards and Guidelines for the Implementation of the PECOS4SMEs Project (henceforth, "Project").

Southern Europe SMEs lack the needed knowledge to take advantage of the e-Commerce potential resulting in Southern Europe lagging within the e-Business area. PECOS4SMEs will develop a training programme concentrated on the provision of suitable knowledge and tools, which the SMEs can use to increase cross border eCommerce revenue. This includes changing sales and marketing strategies and new types of organisation and knowledge about e-Commerce technologies in a future world dominated by pervasive Internet.

The promotion of useful e-Commerce strategies for SMEs includes transfer of knowledge and practices from top performing countries in e-Commerce to lagers thus bridging the cross border sales gap between Northern and Southern Europe and helping the economic recovery of the hard hit South. As a consequence of the introduction of new technologies and concepts facilitating cross border e-Commerce, (e.g. effective link building, search engine user attitudes etc.), SMEs will make their businesses more profitable and improve their ICT and methodological competence for interactive and collaborative learning.

The project is co-funded by the Education and Culture DG under the Lifelong Learning Programme, Leonardo Multilateral projects.

The current deliverable falls under the technical management activity and comprises the documentation of all standards and guidelines to be followed by the partners with regards to development of the project tools and the population and use of the repositories, as well as the use of project management and collaboration tools.

Especially for the web application and Genie and the repositories which are going to be main products of the project it is important that standards are followed to ensure sustainability.

TABLE OF CONTENTS

LIST OF TABLES

Δεν βρέθηκαν καταχωρήσεις πίνακα εικόνων.

LIST OF FIGURES

Δεν βρέθηκαν καταχωρήσεις πίνακα εικόνων.

1. Introduction

1.1. Purpose of the D1.5 Standards and Guidelines

The purposes of the current D1.5 Standards and Guidelines document is to capture the development standards and set the development guidelines.

1.2. Scope of the project

PECOS4SMEs will develop an innovative training system bringing new types of organisation and knowledge about e-Commerce technologies and trends and concentrated on transforming Internet trends into applicable tools for SMEs to change sales and marketing strategies. The approach to training SMEs is based on their situation (sector, financial capability, educational background, geographical location, existing infrastructure, etc.). After completion of the training, SMEs are expected to be able to engage foreign consumers by successfully implementing e-Commerce strategies designed specifically for them.

1.3. Project Objectives

The specific purpose of the proposed project is the development of e-Commerce oriented material delivered through a training system targeting solely SMEs and the deriving needs (e.g. websites friendlier to the foreign consumer, order tracking process, clear dispatch and return policies, etc.) with respect to effective use of the Internet as a sales channel. This includes changing sales and marketing strategies and new types of organisation and knowledge about e-Commerce technologies in a future world dominated by pervasive Internet.

PECOS4SMEs, in particular, aims to support European SMEs to:

- Raise awareness and interest about cross border e-Commerce.
- Learn more about the risks associated with online payments.
- Learn more about the uses of the Internet as a revenue channel.
- Analyze the knowledge development in the SME and see the strategic perspectives of this knowledge in the context of the business strategy.
- Integrate e-Commerce in business planning and innovation strategies.
- Get the strategies written down, so they are not just ideas, but an active choice, which can be communicated and understood by SMEs.
- Get the strategies incorporated in practical procedures that can be constantly updated.
- Be aware of the European cross-border trade legislation and provisions of consumer rights.

An additional goal is to make it easier and less costly for businesses, particularly small and medium-sized enterprises (SMEs), to do business abroad and to enable consumers to reap the full benefit of the Single Market. This goal is aligned with the Europe 2020 strategy – launched on 3 March 2010 (IP/10/225) and with which the Commission is currently tackling bottlenecks in the Single Market to drive economic recovery.

1.4. This document is according to

- IEEE std. 830 19981 IEEE Recommended Practice for Software Requirements Specifications
- IEEE std. 610.12 1990 IEEE Standard Glossary of Software Engineering Terminology

1.5. Textual References

[1] NASA Software Assurance Technology Center, "Principal Components of Orthogonal Object-Oriented Metrics (323-08-14)", NASA Software Assurance Technology Center, 2001

[2] Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design",

IEEE Transactions on Software Engineering, June 1994.

[3] McCabe T. J., "A Complexity Measure", IEEE Transactions on Software Engineering,

December 1976

[4] Borland Together: http://www.borland.com/us/products/together/index.html

[5] Similarity Analyser: http://www.redhillconsulting.com.au/products/simian/

[6] Houari A. Sahraoui, Robert Godin, Thierry Miceli, "Can Metrics Help Bridging the Gap Between the Improvement of OO Design Quality and Its Automation?", Université de Montréal.

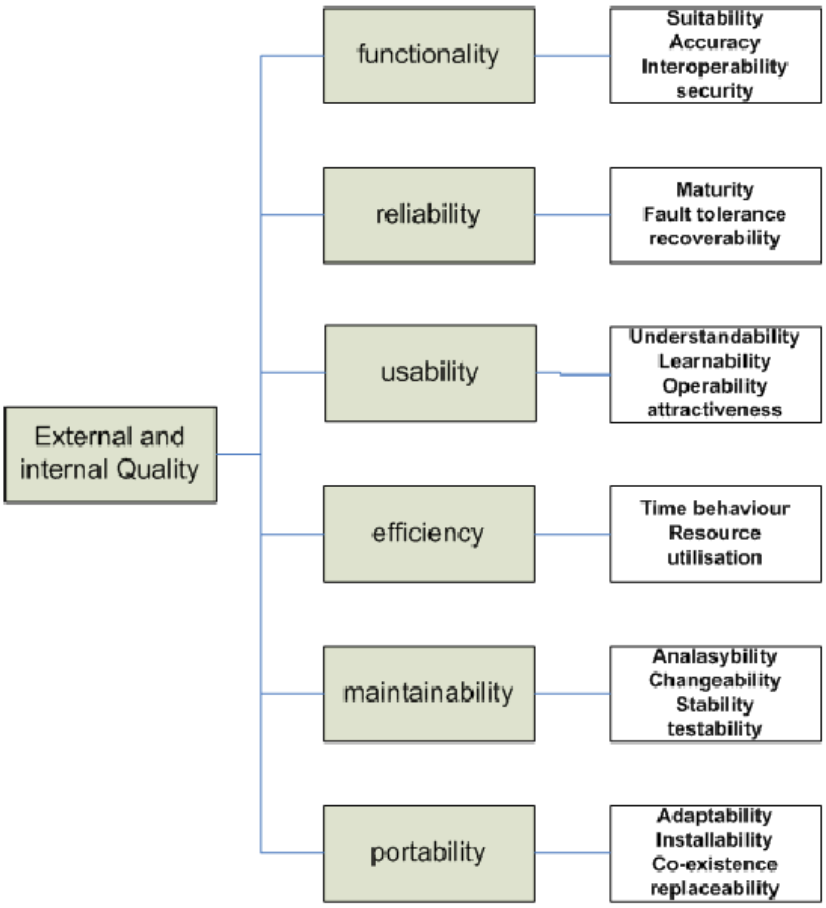
[7] Edmond VanDoren, "Cyclomatic Complexity", Software Engineering Institute.

http://www.sei.cmu.edu/str/descriptions/Cyclomatic_body.html

[8] ISO 9126 Software engineering — product quality. Geneva, Switzerland: ISO/IE, 2001.

2. Background

A Software Quality Check consists of gathering of metrics over the source code, inspection of requirements and design documentation and the use of profilers and other performance tools on the binary code compiled with debugger options. The aim of such a check is, supposing that the software performs the required functionality, to obtain a clear idea of the quality of the chosen solution. In other words two implementations of the same functionality can be complete but still one might be better than the other in terms of certain characteristics that are universally accepted as desirable. What characteristics are universally accepted as desirable in a software product? The ISO 9126 standard has given a list of these characteristics and their sub-characteristics, as shown in Fig.1.



ISO 9126 explains that the quality characteristics are twofold, by one hand there is the effect perceived by the user, which is the external side of the characteristic and by the other hand the same characteristic has an internal aspect, one perceived by the developers and maintainers of the software.

3. Code Quality Analysis

3.1. Background on Metrics and code standards used

3.1.1. Metrics: Chidamber and Kemerer metric set

The Chidamber & Kemerer metrics suite originally consists of 6 object oriented metrics calculated for each class: weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between objects (CBO), response for a class (RCF) and lack of cohesion in methods (LOCOM1). Although the metric set was proposed in 1994, it is an industry standard and is widely supported by commercial applications.

Weighted methods per class (WMC)

The weighted methods per class just count how many methods are defined in a given class. The WMC is a nice predictor of the time needed to develop and maintain a class. The greater the WMC the harder and complex is to develop and maintain the class, so WMC should be kept as low as possible. The WMC is also an indicator of the impact that a superclass has over its subclasses. A subclass inherits all the superclass methods, so a change on any of it could be a potential problem on the subclass. Finally, a class with a lot of methods is very likely to be tied to a specific application, limiting the possibility of reuse.

Depth of inheritance tree (DIT)

Depth of inheritance tree is the maximum distance from the root of the hierarchy tree to a given node class. DIT metric is an indicator of how difficult is to predict class behaviour and how complex the application design is. Since all superclasses inherit all the superclass method, the deeper the hierarchy is the harder is to predict the subclass behaviour and more complex the design is. In the positive side, a deep inheritance tree increases the potential reuse of functionality, so a balance must be kept between hierarchies with few levels and with lots of levels.

Number of children (NOC)

The number of children counts the number of immediate direct subclasses of a given superclass. The greater the number of children is the greater the functionality is reused, but an abnormally high number of children may also predict a misuse of inheritance, because of an improper abstraction of the parent class.

Coupling between objects (CBO)

The coupling between object is the number of classes a class is coupled with, for example by using one of its methods. CBO is a key indicator of the application design. A high CBO denotes a non modular design and prevents reusing. High coupling makes a class non reusable and ties it to a determined application. To promote modularity and encapsulation CBO should be kept as low as possible. CBO is also an indicator of how complex the testing of a class is, because the higher the CBO is the harder the tests are.

Response for a class (RFC)

The response set of a class is the number of methods that can be potentially executed in response of a message sent to an object of a class. With high values of the RFC, the level of understanding of how the class works and collaborate with its coupled classes needed to test and debug rises exponentially. This affects to the time needed to properly test a class.

Lack of cohesion (LOCOM1)

Cohesion measures how similar the methods of a class are. A method is similar to another method if both methods access the same class attributes. This metric is the object oriented counterpart of the classic cohesion metric that measure the relations between program portions. Cohesion within a class is desirable because it promotes encapsulation. Lack of cohesion usually implies that class should be split in two or more classes and exposes flaws in the design. Also, low cohesion increases complexity since the class is probably taking too many responsibilities.

3.1.2. Other metrics

Aside from Chidamber and Kemerer set, three other metrics are obtained: Cyclomatic complexity (CC), lines of code per class (LOC) and duplicated code (DC)

McCabe Cyclomatic complexity (CC)

The Cyclomatic complexity is *a measure of a module control flow complexity based on graph theory* [5]. This means that Cyclomatic complexity count all the possible execution branches for a class. It is a great indicator of the class complexity. Also, the CC metric can be taken as a lowest limit for the number of tests needed to achieve a 100% of testing coverage. CC offers a great insight of the effort needed to test an application.

Lines of code per class (LOC)

Lines of code per class are just the measure of the sentences (any no comment line) of a class. LOC is the simplest approach to measure the complexity of a class, but also the most criticized one. It is a nice indicator of a high coupling, lack of cohesion and excessive responsibility of a class.

Duplicated code (DC)

Duplicated code is the amount of code replicated between classes. Duplicated code raises the possibility of duplicate errors through the code and duplicates the effort needed to resolve a bug (it has to be fixed all the times it is replicated).

3.2. Code audits

3.2.1. Coding standards deviation

This set of audits searches for deviations of the Java Coding Conventions. Typically, these audits do not denote problems at execution time, but code standardization promotes the

comprehensibility of the code of other team members. This affects to the maintainability of the project, because *hardly any software is maintained for its whole life by the original author* [6]. A code convention allows developers to understand the code of other people easily and thoroughly. Every audit is categorized attending to its relevance using the *Severity* variable. These are the audits used for OHIM.

Accessing Static Members through Objects (ASMO)

Static members should be referenced through class names rather than through objects.

Severity: **high**

Non-Final Static Attributes (NFSA)

This rule helps you to avoid non-final static attributes.

Severity: normal

Provide Incremental in 'for' Statement or Use 'while' Statement (PIFS)

This rule checks whether the third argument of a for statement is missing.

Severity: normal

String Literals (SL)

String literals in code affect the readability of the code, making it difficult to understand it.

Severity: normal

Constant Private Attributes Should Be Final (CPASBF)

Private attributes that never get their values changed should be declared final. When you explicitly declare them like this, the source code provides some information to the reader about how the attribute is supposed to be used.

Severity: **high**

List Public and Package Members First (LPPMF)

Enforces standard to improve readability. Methods/data in your class should be ordered properly.

Severity: **high**

Order of Declaration of Class Members (ODCM)

According to Sun Code Conventions for Java, the parts of a class or interface declaration should appear in the following order:

- ⇒ Class (static) variables. First the public class variables, then the protected, then package level (no access modifier), and then the private.
- ⇒ Instance variables. First the public class variables, then the protected, then package level (no access modifier), and then the private.

⇒ Constructors
⇒ Methods

Severity: Low

Order of Modifiers (OM)

Checks for correct ordering of modifiers.

⇒ For classes: visibility (public, protected or private), abstract, static, final.
⇒ For attributes: visibility (public, protected or private), static, final, transient, volatile.
⇒ For operations: visibility (public, protected or private), abstract, static, final, synchronized, native.

Severity: Normal

Naming Conventions (NC)

Takes a regular expression and item name and reports all occurrences where the pattern does not match the declaration.

Severity: Normal

Use Conventional Variable Names (UCVN)

One-character local variable or parameter names should be avoided, except for temporary and looping variables, or where a variable holds an undistinguished value of a type.

To avoid potential conflicts, change the names of local variables or parameters that consist of only two or three uppercase letters and coincide with initial country codes and domain names, which could be used as first components of unique package names.

Severity: Low

3.3. Documentation related findings

Documentation audits look for *Javadoc* defects.

Bad Tag in Javadoc Comments (BTJC)

This rule verifies code against accidental use of improper Javadoc tags.

Severity: normal

Provide File Comments (PFC)

According to Sun Code Conventions for Java, all source files should begin with a C-style comment that lists the class name, version information, date, and copyright notice.

Severity: low

Provide Javadoc Comments (PJDC)

Checks whether Javadoc comments are provided for classes, interfaces, methods and attributes. In the options boxes, you can specify whether to check Javadoc comments for public, package, protected, or all classes and members. Sun Code Conventions for Java also recommend that the order of @param tags should correspond to the order of operation parameters, and @throws (or @exception) tags should be sorted alphabetically. When the Ordered options are checked, the audit also verifies whether the corresponding tags are ordered correctly.

Severity: low

3.3.1. Potential problems

Potential problems audits look for possible source of errors in the code. These errors can be performance issues, hiding information problems and encapsulation problem.

Hiding Inherited Attributes (HIA)

This rule detects when attributes declared in child classes hide inherited attributes.

Severity: **critical**

Hiding Inherited Static Methods (HISM)

This rule detects when inherited static operations are hidden by child classes.

Severity: **critical**

Hiding Names (HN)

Declarations of names should not hide other declarations of the same name.

Severity: **critical**

Overriding a Private Method (OPM)

A subclass should not contain a method with the same name and signature as in a super-class if these methods are declared to be private.

Severity: **critical**

Appending to String Within a Loop (ASWL)

Performance enhancements can be obtained by replacing String operations with StringBuffer operations if a String object is appended to within a loop.

Severity: **high**

Performance enhancements can be obtained by replacing String operations with StringBuffer operations if a String object is appended to within a loop.

Severity: high

Declaring Variables Inside Loops (DVIL)

This rule recommends declaring local variables outside the loops. The reason: as a rule, declaring variables inside the loop is less efficient.

Severity: low

Multiple String Concatenations (MSC)

String classes are designed to be immutable, and every method in the class that appears to modify a String actually creates and returns a brand new String object containing the modification. The original String is left untouched, but the compiler can optimize your code. Using StringBuffer instead of String for multiple mutable operations is better for performance. This audit helps you to find suspect code.

Severity: normal

'Synchronized' Modifier (SM)

The synchronized modifier on methods can sometimes cause confusion during maintenance as well as during debugging. This rule therefore recommends using synchronized statements as replacements instead.

Priority: normal

Assignments in Conditional Expressions (ACE)

Use of assignments within conditions makes the source code hard to understand.

Priority: normal

Unused Local Variables and Formal Parameters (ULVFP)

Local variables and formal parameters declarations should be used.

Priority: low

Public and Package Attributes (PPA)

Declare the attributes either private or protected and provide operations to access or change them.

Priority: normal

4. Checking the quality of the code

Metrics and audits are obtained using Borland Together 6.1 [7] with the default configuration.

The name of the following metrics is changed in Borland Together:

- ⇒ WMC: the name of this metric in Borland Together is WMPC2
- ⇒ DIT: the name of this metric in Borland Together is DOIT
- ⇒ NOC: the name of this metric in Borland Together is NOCC

Duplicated code is calculated with Similarity Analyser (Simian) v 2.2.17 [8], a free open source application that is one of the de-facto standards to calculate this metric. Two duplication thresholds are defined:

- ⇒ Blocks of 6 lines (Simian's default, a very aggressive value) and
- ⇒ Blocks of 15 lines (more relaxed value).

4.1. Acceptable quality of code

These are the limit values that OHIM is prepared to accept for applications:

	WMC	**DIT**	**NOC**	**CBO**	**RFC**	**LOCOM1**	**CC**	**LOC**
Mean	**6**	**1,7**	**0,33**	**2,40**	**8**	**9**	**5**	**40**
Standard Deviation	5	1	0,66	2,40	7	15	4	30
Maximum	60	6	25	20	45	500	45	400

	DC % duplicated code (6-line blocks)	**DC % duplicated code (15-line blocks)**
Maximum	12 %	3 %

	Coding standard deviations	**Documentation problems**	**Potential problems**
	(ASMO + NFSA + PIFS + SL + CPASBF + LPPMF + ODCM + OM + NC + UCVN)	(BTJC + PFC + PJDC)	(HIA + HISM + HN + OPM)
Maximum average defects per class	**5**	**7,5**	**0,25**