IEC/PAS 62814

Edition 1.0 2012-12

PUBLICLY AVAILABLE SPECIFICATION

PRE-STANDARD

Dependability of software products containing reusable components – Guidance for functionality and tests

IEC/PAS 62814:2012(E)

IEC/PAS 62814

Edition 1.0 2012-12

PUBLICLY AVAILABLE SPECIFICATION

PRE-STANDARD

Dependability of software products containing reusable components – Guidance for functionality and tests

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE XA

ICS 03.120.01

ISBN 978-2-83220-501-3

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

DEPENDABILITY OF SOFTWARE PRODUCTS CONTAINING REUSABLE COMPONENTS – GUIDANCE FOR FUNCTIONALITY AND TESTS

FOREWORD

A PAS is a technical specification not fulfilling the requirements for a standard, but made available to the public.

IEC/PAS 62814 has been processed by IEC technical committee 56: Dependability.

The text of this PAS is based on the following document: | This PAS was approved for publication by the P-members of the committee concerned as indicated in the following document

Draft PAS	Report on voting
56/1479/PAS	56/1490/RVD

Following publication of this PAS, which is a pre-standard publication, the technical committee or subcommittee concerned may transform it into an International Standard.

This PAS shall remain valid for an initial maximum period of 3 years starting from the publication date. The validity may be extended for a single period up to a maximum of 3 years, at the end of which it shall be published as another type of normative document, or shall be withdrawn.

INTRODUCTION

Technological growth is accelerating; development cycles for products are becoming shorter and shorter. At the same time software is taking an increasingly important part in the control and functionality of products and in integrating the functions of hardware components. The disciplined development of software has been going on for more than 40 years and software is now available in many forms and formats. Apparently, the cost of software development can easily be amortized if it is embedded as often, and in as many different products, as possible. This potential benefit of software reuse should at no time be at the expense of dependability. Dependability is the ability of a system to perform as and when required to meet specific objectives under given conditions of use.

Any innovative product that has matured enough to hit the shelves needs a new and progressive approach. Dependability of the products is an attribute that is mandatory for newly developed or reused software (and the complete product into which the software is embedded) to be accepted and sold. Therefore, the dependability of software and its components should be assured in just the same way that the dependability of hardware and its components have been assured for many decades. This requires the standardization of software and software components to keep up with the ever higher levels which hardware components continue to achieve.

The dependability of a system infers that the system is trustworthy and capable of performing the desired service upon demand to satisfy user needs. Whereas a software component may be perfectly suited to one application, it may prove to cause severe faults in other applications. To allow the innovators to concentrate on their main task – to create new and better products with an extended functionality – it is fundamental to provide the certainty that reused software is dependable in its new application and does not need to be re-designed from scratch. Safety and security aspects might be combined if required. Therefore an adequate test process considering the changed purpose and the different application configuration in combination with new, reused, or further used components is needed. Altogether, testing of software products containing reused components is an important target to be reached.

An additional, important aspect to be considered is the energy efficiency and eco-friendliness of hardware products controlled by software. Reuse of a component with a bad energy consumption behaviour will multiply this bad behaviour, and thus negatively impact the entire energy consumption of the new system that is composed of such components; the same way as an undependable component impacts the dependability of the system into which it will be built. A rule of thumb is that reused software should not result in a product consuming more energy than a comparable energy-efficient product on the market.

This publicly available specification (PAS) addresses the functionality, testing and dependability of software components to be reused and products that contain software to be used in more than one application; that is, reused by the same or by another development organization, regardless of whether it belongs to the same or another legal entity than the one that has developed this software.

DEPENDABILITY OF SOFTWARE PRODUCTS CONTAINING REUSABLE COMPONENTS – GUIDANCE FOR FUNCTIONALITY AND TESTS

1 Scope

This publicly available specification introduces the concept of assuring reused components and their usage within new products. It provides information and criteria about the tests and analysis required for products containing such reused parts. The objective is to support the engineering requirements for functionality and tests of reusable software components and composite systems containing such components in evaluating and assuring reuse dependability.

Focus is on the dependability of software reuse and, thus, this PAS complements IEC 62309 which exclusively considers hardware reuse. In addition to this previous hardware-related IEC standard, the present PAS also crosses further, appropriate software-related standards to be applied in the development and qualification of software components that are intended to be reused and products that reuse existing components. In other words, this PAS encompasses the features of software components for reuse, their integration into the receiving system, and related tests. Their performance and qualification and the qualification of the receiving system is subject to existing standards, for example ISO/IEC 25000 [01][1], IEC 61508-3 [01] and IEC 61508-4 [03]. The process framework of ISO/IEC 12207 [04] on systems and software engineering and ISO/IEC 25000 [01] on system aspects of dependability on software engineering apply to this PAS.

The purpose of this PAS is to ensure through analysis and tests that the functionality, dependability and eco-friendliness of a new product containing reused software components is comparable to a product with only new components. This would justify the manufacturer providing the next customer with a warranty for the functionality and dependability of a product with reused components. As each set of hardware/software has a unique relationship and is governed by its operational scenario, the dependability determination has to consider the underlying operational background. Dependability also influences safety. Therefore, wherever it seems necessary, safety aspects have to be considered the way IEC 60300-1 addresses safety issues.

This PAS can also be applied in producing product-specific standards by technical committees responsible for an application sector.

This PAS is not intended for certification purposes.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60300-1, *Dependability management – Part 1: Dependability management systems*

IEC 62628, *Guidance on software aspects of dependability*

1 Numbers in square brackets refer to the Bibliography.

IEC 62309, *Dependability of products containing reused parts – Requirements for functionality and tests*

3 Terms, definitions and abbreviations

For the purposes of this document, the following terms and definitions apply.

3.1 Terms related to software engineering

3.1.1
software
programs, procedures, rules, documentation, and data of an information processing system

Note 1 to entry: Software is an intellectual creation that is independent of the medium upon which it is recorded.

Note 2 to entry: Software requires hardware devices to run, to store, and transmit data.

Note 3 to entry: Documentation includes: requirements specifications, design specifications, source code listings, comments in source code, "Help" text and messages for display at the computer/human interface, installation instructions, operating instructions, user manuals, and support guides used in software maintenance.

3.1.2
embedded software
software within a system whose primary purpose is realizing an application

EXAMPLE Software used in the brake control systems of motor vehicles, or to control an x-ray system in medical health-care.

3.1.3
software unit/software module
software element in programming codes that can be separately specified, compiled, documented and tested to perform a task or activity to achieve a desired outcome of a software function

Note 1 to entry: The terms "unit" and "module" are often used interchangeably or defined to be sub-elements of one another in different ways depending upon the context. The relationship of these terms is not yet standardized.

3.1.4
software (configuration) item
software item that has been configured and treated as a single item in the configuration management process

Note 1 to entry: A software configuration item can consist of one or more software units to perform a software function.

3.1.5
software function/(software) function block
elementary operation performed by the software module or unit as specified or defined as per stated requirements to fulfil a well-defined user or system function or a part of it

EXAMPLE Calculation of sinus of a given angle is a function block of a unit to calculate trigonometric functions; giving the address to buy a ticket is a function block of a web portal.

Note 1 to entry: Software units consist of function blocks.

Note 2 to entry: A function block contains input variables, output variables, through variables, internal variables, and an internal behaviour description of the function block.

3.1.6
software system
defined set of software items that, when integrated, behave collectively to satisfy a requirement

EXAMPLES Application software for accounting and information management, application-oriented system software for text processing / performance analysis / programming tools, system software for linking library functions.

3.1.7
(computer) program
set of coded instructions executed to perform specified logical and mathematical operations on data

Note 1 to entry: Programming is the general activity of software development in which the programmer or computer user states a specific set of instructions that the computer has to perform.

Note 2 to entry: A program consists of a combination of coded instructions and data definitions that enable computer hardware to perform computational or control functions.

3.1.8
program code
character or bit pattern that is assigned a particular meaning to express a computer program in a programming language

Note 1 to entry: "Source codes" are coded instructions and data definitions expressed in a form suitable for input to a transducer, that is, assembler, compiler, or other translator.

Note 2 to entry: "Object code" or "binary code" or "executable code" is the bit pattern obtained from a translator and is ready to run.

Note 3 to entry: "Coding" is the process of transforming of logic and data from design specifications or descriptions into a programming language.

Note 4 to entry: A programming language is a language used to express computer programs.

3.1.9
product line
collection of systems potentially derivable from a single architecture

3.2 Terms related to software dependability

3.2.1
software dependability
ability of the software to perform as and when required when integrated in system operation

3.2.2
reuse dependability
ability of a composite system containing reusable components to perform as and when required to meet users' service needs

3.2.3
software fault
bug
error or flaw in a software item that may prevent it from performing as required

Note 1 to entry: Software faults are either specification faults, or design faults, or programming faults, or compiler-inserted faults, or faults introduced during software maintenance.

Note 2 to entry: A software fault is dormant until activated by a specific trigger and usually reverts to being dormant when the trigger is removed.

Note 3 to entry: In the context of this standard, a bug is a special case of software fault, also known as latent software fault.

3.2.4
software failure
failure that is a manifestation of a software fault

Note 1 to entry: A single software fault will continue to manifest itself as a failure until it is removed.

3.2.5
validation
confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled

Note 1 to entry: Validation answers the question whether or not the right software has been developed.

3.2.6
verification
confirmation by examination and through the provision of objective evidence that specified requirements have been fulfilled

Note 1 to entry: Verification answers the question whether or not the developed software is correct.

3.2.7
qualification
process of validation and verification (V&V), used to demonstrate that the product is capable of meeting its specification for all the required conditions and environments

3.2.8
quality target
specified level of quality as a goal; wherever possible, quantified using a software metric

EXAMPLE The overall reliability of the composite system in terms of the required MTBF, or the requirement that cyclomatic complexity of a software unit be kept below 7.

3.3 Terms related to software reuse

3.3.1
software reuse
using a software asset, i.e., software or software knowledge, in the solution of a different problem in order to construct new software [05]

Note 1 to entry: This notion covers both "heritage" and "legacy" software, and it is refined into categories: "black-box", "white-box", "adaptive", "systematic", and "accidental" reuse (see 3.3.12 to 3.3.16).

Note 2 to entry: Opposite of "software reuse" is "software one-use" that requires being developed from scratch.

EXAMPLE Some dedicated software routines, such as security codes, are not designed for reuse; they are one-use components.

3.3.2
software (reuse) asset
software configuration item that has been designed for use in multiple contexts and domains

EXAMPLE 1 Design, specification, source code, documentation, test suites, manual procedures, etc.

EXAMPLE 2 Availability of the information that a specific navigation function uses an algorithm based on Kalman filter.

Note 1 to entry: Also a software-based or software-oriented knowledge is an asset.

3.3.3
context
software environment tied to mission and software requirements

3.3.4
domain
problem space or application area

3.3.5
software reusability
degree to which a (reuse) asset can be used in more than one software system or in building other assets

Note 1 to entry: In a (reuse) repository software reusability represents the characteristics of an asset that make it easy to reuse vertically or horizontally.

Note 2 to entry: Usability is a measure of software unit's or system's functionality, ease of use, and efficiency.

3.3.6
(software) component
constituent of a software system with specified interfaces and explicit context and domain dependencies

Note 1 to entry: A software component can consist of one or more software units to perform a software function.

EXAMPLE 1 An individual component is a software package, a web service, or a module that encapsulates a set of related functions (or data).

3.3.7
(software) component off the shelf
COTS
commercially available components

Note 1 to entry: A COTS software can consist of one or more software units to perform a software function.

Note 2 to entry: Components in governmental use are called "government off the shelf (GOTS)".

Note 3 to entry: COTS and GOTS software usually represents components; they can be also stand-alone applications.

Note 4 to entry: COTS and GOTS typically realize reuse incorporation or integration.

Note 5 to entry: COTS and GOTS are designed to be implemented easily into existing systems without the need for customization ("glue code", "wrappers", 3.3.8, 3.3.10).

EXAMPLE 1 Microsoft Office is a stand-alone COTS application that is a packaged software solution for businesses. An operating system, a word processor, a compiler, etc. are further examples of stand-alone COTS.

EXAMPLE 2 Also libraries that need linkage to an application code, e.g., graphic engines, Windows DLLs, etc., are COTS components.

EXAMPLE 3 Software that is used to create software, but is not part of a composite software system, is not COTS software; it is a development tool. However, development environments with runtime modules are COTS (e.g., Visual Basic™, Sysbase™), or information retrieval applications (e.g., hypertext and data mining tools), or operating system utilities (e.g., for file operations and memory management).

3.3.8
glue code
software that intermediates between the reusable component and the receiving system

3.3.9
connector
interface elements of composite system to receive reusable components

3.3.10
wrappers
additional software to complete the functional and interface requirements if they are not priorly fulfilled

3.3.11
(reuse) repository
storage of a collection of reusable components

Note 1 to entry: In a narrower sense, “software libraries” have the same function as software repositories, e.g., building sets of reusable software units such as trigonometric functions.

3.3.12
accidental reuse
reuse without strategy, typically reusing software components not designed for reuse

Note 1 to entry: Also known as “ad hoc” or “opportunistic” reuse.

3.3.13
systematic reuse
developing software components intended for reuse and/or building new applications from those reusable components, following a formal plan of product line, also known as “planned reuse”

3.3.14
adaptive reuse
using previously developed software that is modified only for portability, e.g., a new application on a different operating system

3.3.15
black-box reuse
reuse of unmodified software components, incorporating existing software components into a new application without modification

3.3.16
white-box reuse
modifying and integrating software (function) blocks into new applications

3.3.17
vertical reuse
reuse in the same domain

3.3.18
horizontal reuse
reuse in different domains

3.3.19
internal reuse
in-house reuse
reuse of a software unit developed within the company, or government unit

3.3.20
external reuse
reuse of a software unit of another company, or government unit

Note 1 to entry: A “third party software” is usually written by another company as a legal entity. It incorporates external reuse if it will be used in a context or domain other than that for which it has been designed and developed. It can be, however, also a dedicated, one-use software component.

EXAMPLE 1 Open-source software (OSS), mostly for external reuse.

EXAMPLE 2 A service-oriented architecture (SOA) can operate on components in internal or external reuse.

3.3.21
heritage software
inherited software reused from a previous mission that has currently been in usage