



# IEEE Trial-Use Standard for Software Non-Functional Sizing Measurements

IEEE Computer Society

Developed by the  
Software & Systems Engineering Standards Committee (C/S2EC)

IEEE Std 2430™-2019

# IEEE Trial-Use Standard for Software Non-Functional Sizing Measurements

Developed by the

**Software & Systems Engineering Standards Committee (C/S2ESC)**  
of the  
**IEEE Computer Society**

Approved 13 June 2019

**IEEE-SA Standards Board**

**Abstract:** A method for the sizing of nonfunctional software requirements is defined in this standard. It complements ISO/IEC 20926:2009, which defines a method for the sizing of functional user requirements. Non-functional categories for data operations, interface design, technical environment, and architecture software are included in this standard. Steps to determine and calculate the non-functional size are also included. Handling requirements involving both functional and non-functional requirements are explained in this standard, which also covers how to apply non-functional sizing estimates in terms of cost, project duration and quality, and considerations of software performance in terms of productivity and quality. The combination of functional and non-functional size should correspond to the total size necessary to produce the software. The functional size and non-functional size are orthogonal, and both are needed when sizing the software. The complementarity of the functional and the non-functional sizes, to avoid overlaps or gaps between the two size methods, are described in this standard. Calculating the implementation work effort and duration of the non-functional requirements is outside the scope of this standard.

**Keywords:** IEEE 2430™, IFPUG, non-functional size measurements, non-functional requirements, SNAP

---

The Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2019 by The Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved. Published 16 October 2019. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

W3C is trademarks or registered trademarks of the W3C®, (registered in numerous countries) World Wide Web Consortium. Marks of W3C are registered and held by its host institutions: Massachusetts Institute of Technology (MIT), European Research Consortium for Information and Mathematics (ERCIM), and Keio University, Japan.

PDF: ISBN 978-1-5044-5987-7 STD23763  
Print: ISBN 978-1-5044-5988-4 STDPD23763

*IEEE prohibits discrimination, harassment, and bullying.*

*For more information, visit <https://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.*

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

## Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.” They can also be obtained on request from IEEE or viewed at <https://standards.ieee.org/ipr/disclaimers.html>.

### Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers in IEEE working groups are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

## Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

## Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

## Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board  
445 Hoes Lane  
Piscataway, NJ 08854 USA

## Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

## Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

## Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

## Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. A current IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit IEEE Xplore at <https://ieeexplore.ieee.org/> or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at <http://standards.ieee.org>.

## Errata

Errata, if any, for IEEE standards can be accessed via <https://standards.ieee.org/standard/index.html>. Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in IEEE Xplore: <https://ieeexplore.ieee.org/browse/standards/collection/ieee/>. Users are encouraged to periodically check for errata.

## Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at <https://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

## **Trial-use standards**

Publication of this trial-use standard for comment and criticism has been approved by the Institute of Electrical and Electronics Engineers, Inc. Trial-use standards are effective for 36 months from the date of publication. Comments for revision will be accepted for 24 months after publication. Suggestions for revision should be directed to the Secretary, IEEE-SA Standards Board, 445 Hoes Lane, Piscataway, NJ 08855, and should be received no later than 16 October 2021.

## Participants

At the time this IEEE trial-use standard was completed, the P2430 Working Group had the following membership:

### **Talmon Ben-Cnaan, *Chair***

Bill Curtis  
Victoria (Vicky) Hailey  
Annette Reilly

Robert Schaaf  
Roopali Thapar

Charley Tichenor  
Altaz Valani  
Kiran Yeole

The following members of the individual balloting committee voted on this trial-use standard. Balloters may have voted for approval, disapproval, or abstention.

Robert Aiello  
Talmon Ben-Cnaan  
Juris Borzovs  
Pieter Botman  
Demetrio Bucaneg Jr.  
Paul Cardinal  
Lawrence Catchpole  
Jan de Liefde  
Yaacov Fenster  
Andrew Fieldsend  
David Fuschi  
Julian Gomez  
Randall Groves  
Victoria (Vicky) Hailey  
Mark Henley

Werner Hoelzl  
Noriyuki Ikeuchi  
Atsushi Ito  
Srinivasa Rao  
Kanneganti  
Piotr Karocki  
David Leciston  
Johnny Marques  
LaMont McAliley  
Rajesh Murthy  
Nick S.A. Nikjoo  
Mark Paulk  
Annette Reilly  
Saurabh Saxena  
Robert Schaaf  
Stephen Schwarm

Carl Singer  
Kendall Southwick  
Friedrich Stallinger  
Thomas Starai  
Walter Struppler  
Marcy Stutzman  
Sachin Thakur  
Roopali Thapar  
Charley Tichenor  
John Vergis  
Scott Willy  
Steven Woodward  
Jian Yu  
Oren Yuen  
Janusz Zalewski

When the IEEE-SA Standards Board approved this trial-use standard on 13 June 2019, it had the following membership:

### **Gary Hoffman, *Chair***

### **Ted Burse, *Vice Chair***

### **Jean-Philippe Faure, *Past Chair***

### **Konstantinos Karachalios, *Secretary***

Masayuki Ariyoshi  
Stephen D. Dukes  
J.Travis Griffith  
Guido Hiertz  
Christel Hunter  
Thomas Koshy  
Joseph L. Koepfinger\*  
Thomas Koshy

John D. Kulick  
David J. Law  
Joseph Levy  
Howard Li  
Xiaohui Liu  
Kevin Lu  
Daleep Mohla  
Andrew Myles

Annette D. Reilly  
Dorothy Stanley  
Sha Wei  
Phil Wennblom  
Philip Winston  
Howard Wolfman  
Feng Wu  
Jingyi Zhou

\*Member Emeritus



## Introduction

This introduction is not part of IEEE Std 2430-2019, IEEE Trial-Use Standard for Software Non-Functional Sizing Measurements.

Having both software functional size and non-functional size provides significant information for the management of software product development. The functional size is quantifiable and represents a good measure of the functional project/application size. Providing a quantifiable measure for the non-functional requirements (NFR) allows organizations to build historical data repositories that can be referenced to assist in decision making for the technical and/or quality aspects of applications.

Non-functional sizing assists organization in multiple ways (see [Annex A](#)). It provides insight into projects and applications to assist in effort and cost estimating and in the analysis of quality and productivity. Used in conjunction with function-point analysis, non-functional sizing provides information that can identify additional software size, which may impact quality and productivity in a positive or negative way. Having this information enables software professionals to:

- Better plan, schedule, and estimate projects.
- Identify areas of process improvement.
- Assist in determining future technical strategies.
- Quantify the impacts of the current technical strategies.
- Improve quality: Analyzing non-functional size may assist in identifying implicit requirement and may assist in analyzing the components of the solution to meet the NFR by looking at the sizing attributes.

By learning the methodology as described in this standard and by performing the non-functional sizing together with functional sizing, the added time and effort to size the NFR is small.

## Acknowledgments

The author thanks the International Electrotechnical Commission (IEC) for permission to reproduce information from its international standards. All such extracts are copyright of IEC Geneva, Switzerland. All rights reserved. Further information on the IEC is available from [www.iec.ch](http://www.iec.ch). IEC has no responsibility for the placement and context in which the extracts and contents are reproduced by the author, nor is IEC in any way responsible for the other content or accuracy therein.

Portions of the Software Non-functional Assessment Process (SNAP), Assessment Practices Manual, Release 2.4, reprinted with permission from IFPUG, ©2017.

## Contents

1. Overview .....	10
1.1 Scope .....	10
1.2 Purpose .....	10
1.3 Word usage .....	11
2. Normative references .....	11
3. Definitions, acronyms, and abbreviations .....	12
3.1 Definitions .....	12
3.2 Abbreviations .....	16
4. Introductory Information .....	17
4.1 Non-Functional Software Size Measurement (NFSSM) introduction .....	17
4.2 Software-intensive system and software product .....	18
4.3 Software domains .....	18
4.4 The relations between non-functional requirements (NFR) definition and functional user requirements (FUR) .....	18
4.5 Key features of the NFSSM .....	20
4.6 Future evolution of NFR .....	21
4.7 Objectives and benefits .....	22
5. Non-functional size: Categories and sub-categories .....	23
5.1 Category 1: Data operations .....	23
5.2 Category 2: Interface design .....	32
5.3 Category 3: Technical environment .....	41
5.4 Category 4: Architecture .....	47
5.5 Sizing code data .....	52
6. The sizing process .....	55
6.1 The timing of the non-functional sizing .....	56
6.2 Non-functional sizing and FSM .....	56
6.3 Steps to determine the non-functional size .....	57
6.4 Calculating the Non-functional size .....	63
7. Complementarity of the functional and the non-functional sizes .....	66
7.1 Requirements involving functional and non-functional requirements .....	67
8. Use of non-functional software sizing .....	77
8.1 Functional size and non-functional size .....	77
8.2 Project management .....	77
8.3 Performance management .....	81
Annex A (informative) NFSSM strengths .....	84
Annex B (informative) Bibliography .....	85

# IEEE Trial-Use Standard for Software Non-Functional Sizing Measurements

## 1. Overview

### 1.1 Scope

This standard defines a method for the sizing of non-functional software requirements. It complements ISO/IEC 20926:2009, which defines a method for the sizing of functional user requirements (FUR).<sup>1</sup>

This standard also describes the complementarity of functional and non-functional sizes, so that sizing both functional and non-functional requirements (NFR) do not overlap. It also describes how non-functional size, together with functional size, should be used for measuring the performance of software projects, setting benchmarks, and estimating the cost and duration of software projects.

In general, there are many types of non-functional software requirements. Moreover, non-functional aspects evolve over time and may include additional aspects in the as technology advances. This standard does not intend to define the type of NFR for a given context. Users may choose ISO 25010:2011 or any other standard for the definition of NFR. It is assumed that users will size the NFR based on the definitions they use.

This standard covers a subset of non-functional types. It is expected that, with time, the state of the art can improve and that potential future versions of this standard can define an extended coverage. The ultimate goal is a version that, together with ISO/IEC 20926:2009, covers every aspect that may be required of any prospective piece of software, including aspects such as process and project directives that are hard or impossible to trace to the software's algorithm or data. The combination of functional and non-functional size would then correspond to the total size necessary to bring the software into existence.

Calculating the effort and duration of the implementation of the NFR is outside the scope of this standard.

### 1.2 Purpose

The purpose of this standard is to define the method of sizing of NFR and describe how to use this size, alongside with the functional size.

---

<sup>1</sup>Information on references can be found in [Clause 2](#).

### 1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (shall equals is required to).<sup>2,3</sup>

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (should equals is recommended that).

The word *may* is used to indicate a course of action permissible within the limits of the standard (may equals is permitted to).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (can equals is able to).

## 2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

ISO/IEC 14143-1:2007, Information Technology—Software measurement—Functional size measurement.<sup>4,5</sup>

ISO/IEC 20926:2009, Software and systems engineering—Software measurement—IFPUG functional size measurement method 2009.

ISO/IEC 25010:2011, Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models.

ISO/IEC TR 14143-5:2004 Information technology—Software measurement—Functional size measurement—Part 5: Determination of functional domains for use with functional size measurement.

ISO/IEC/IEEE 24765:2017, Systems and software engineering—Vocabulary.<sup>6,7</sup>

---

<sup>2</sup>The use of the word *must* is deprecated and shall not be used when stating mandatory requirements, *must* is used only to describe unavoidable situations.

<sup>3</sup>The use of *will* is deprecated and shall not be used when stating mandatory requirements, *will* is only used in statements of fact.

<sup>4</sup>ISO publications are available from the International Organization for Standardization (<https://www.iso.org/>) and the American National Standards Institute (<https://www.ansi.org/>).

<sup>5</sup>IEC publications are available from the International Electrotechnical Commission (<https://www.iec.ch/>). IEC publications are also available in the United States from the American National Standards Institute (<http://www.ansi.org>).

<sup>6</sup>The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

<sup>7</sup>IEEE publications are available from The Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (<https://standards.ieee.org/>).

### 3. Definitions, acronyms, and abbreviations

#### 3.1 Definitions

For the purposes of this document, the following terms and definitions apply. For terms not defined in this clause, consult ISO/IEC/IEEE 24765:2017 or the *IEEE Standards Dictionary Online*.<sup>8</sup>

**adaptive maintenance:** Modification of a software product, performed after delivery, to keep a software product usable in a changed or changing environment.

NOTE 1 —Adaptive maintenance provides enhancements necessary to accommodate changes in the environment in which a software product must operate. These changes are those that must be made to keep pace with the changing environment. For example, the operating system might be upgraded, and some changes may be made to accommodate the new operating system.<sup>9</sup>

NOTE 2 —From ISO/IEC 14764:2006 ed 2.0, Copyright © 2006 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch). [B8].<sup>10</sup>

**application:** A cohesive collection of automated procedures and data supporting a business objective; it consists of one or more components, modules, or subsystems. Examples: accounts payable, accounts receivable, payroll, procurement, shop production, assembly line control, air search radar, target tracking, weapons firing, flight line scheduling, and passenger reservations.

NOTE —From ISO/IEC 20926:2009 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**boundary (application boundary):** A conceptual interface between the software under study and its users. The boundary (also referred to as application boundary) defines what is external to the application; it indicates the border between the software being measured and the user; it acts as a “membrane” through which data processed by transactions pass into and out of the application; it is dependent on the user’s external business view of the application; it is independent of non-functional and/or implementation considerations. The boundary is identical when assessing the functional size and the non-functional size

NOTE —From ISO/IEC 14143-1:2007 ed 2.0, Copyright © 2007 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**code data:** A type of data entities, used for software sizing (in addition to Business Data and Reference Data). According to ISO/IEC 20926:2009 [IFPUG function point analysis (FPA)], “code data” usually exists to satisfy non-functional requirements (NFR) from the user (for quality requirements, physical implementation and/or a technical reason). Code data provides a list of valid values that a descriptive attribute may have. Typically, the attributes of the code data are Code, Description and/or other “standard” attributes describing the code; e.g., standard abbreviation, effective date, termination date, audit trail data, etc. The different categories of data are outlined below to assist in identification.

**corrective maintenance:** Reactive modification of a software product performed after delivery to correct discovered problems [B8].

NOTE 1 —The modification repairs the software product to satisfy requirements.

NOTE 2 —From ISO/IEC 14764:2006 ed 2.0, Copyright © 2006 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

<sup>8</sup>*IEEE Standards Dictionary Online* is available at: <http://dictionary.ieee.org>.

<sup>9</sup>Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

<sup>10</sup>The numbers in brackets correspond to those of the bibliography in [Annex B](#).

**data element type (DET):** A unique, non-repeated attribute, which can be in Business Data, Reference Data, or code data. The number of different types of Data Elements of all tables is the number of DETs. Instances of control information are also counted as a DET, such as the “Enter” key to facilitate an external input (EI).

**database view:** The result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object. This pre-established query command is kept in the database dictionary. Unlike ordinary base tables in a relational database, it is a virtual table computed or collated dynamically from data in the database, when access to that view is requested. Changes applied to the data in a relevant underlying table are reflected in the data shown in subsequent invocations of the view.

**elementary process (EP):** The smallest unit of activity that is meaningful to the user(s).

NOTE —From ISO/IEC 20926:2009 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**external input (EI):** An elementary process (EP) that processes data or control information sent from outside the boundary.

NOTE —From ISO/IEC 20926:2009 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**external inquiry (EQ):** An elementary process (EP) that sends data or control information outside the boundary.

NOTE —From ISO/IEC 20926:2009 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**external interface file (EIF):** user recognizable group of logically related data or control information, which is referenced by the application being measured, but which is maintained within the boundary of another application.

NOTE —From ISO/IEC 20926:2009 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**external output (EO):** An elementary process (EP) that sends data or control information outside the boundary and includes additional processing logic beyond that of an External Inquiry.

NOTE —From ISO/IEC 20926:2009 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**family of platforms:** Software Platforms that are serving the same purpose. for example, operating systems; browsers. *See also:* **platforms**.

**file type referenced (FTR):** A data function read and/or maintained by a transactional function. A file type referenced includes: An internal logical file read or maintained by a transactional function, or an external interface file read by a transactional function. Code data is grouped into one additional FTR.

**function points (FPs):** A unit of measurement to express the amount of functionality an information system (as a product) provides to a user. In this standard, function points (FPs).

NOTE —From ISO/IEC 20926:2009 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**functional sizing measurements (FSM):** The process of measuring functional size.

NOTE —From ISO/IEC 14143-1:2007 ed 2.0, Copyright © 2007 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**functional user requirements (FUR):** A sub-set of the user requirements. Requirements that describe what the software shall do, in terms of tasks and services.

NOTE 1—FUR relate to, but are not limited to:

- Data transfer (for example: input customer data; send control signal)
- Data transformation (for example: calculate bank interest; derive average temperature)
- Data storage (for example: store customer order; record ambient temperature over time)
- Data retrieval (for example: list current employees; retrieve latest aircraft position)

NOTE 2—Examples of user requirements that are not FUR include, but are not limited to:

- Quality constraints (for example: usability, reliability, efficiency and portability)
- Organizational constraints (for example: locations for operation, target hardware and compliance to standards)
- Environmental constraints (for example: interoperability, security, privacy and safety)
- Implementation constraints (for example: development language, delivery schedule)

NOTE 3—From ISO/IEC 14143-1:2007 ed 2.0, Copyright © 2007 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**internal logical file (ILF):** User recognizable group of logically related data or control information maintained within the boundary of the application being measured.

**logical file:** A logical group of data as seen by the user. A logical file is made up of one or more data entities. A data function represents functionality provided to the user to meet internal and external data storage requirements. A data function is either an internal logical file or an external interface file. Grouping of data into logical files is the result of the combined effect of two grouping methods: Method a) is process driven, based on the user transactions in the application. Method b) is data-driven, based on the business rules.

**multiple instance approach:** The case where each method of delivery of same functionality is counted separately *See also: single instance approach.*

**non-functional size:** Size of the software derived by quantifying the non-functional requirements (NFR), defined by a set of rules.

**non-functional requirements (NFR):** Any requirement for a software-intensive system or for a software product, including how it should be developed and maintained, and how it should perform in operation, except any functional user requirement for the software. Non-functional requirements (NFR) concern: the software system or software product quality, the environment in which the software system or software product must be implemented and which it must serve, and the processes and technology to be used to develop and maintain the software system or software product and the technology to be used for their execution. *See also: functional user requirements.*

**number of DETs:** The sum of all DETs which are part of the input/output/query of the elementary process (EP), plus the data elements which are read or updated internally to the boundary.

**partition:** A set of software functions within an application boundary that share homogeneous criteria and values. A partition requires development effort, which may not be reflected when sizing the functional aspect of the project/product, using FPA. Partition is designed to improve non-functional perspective of the users,



such as maintainability, portability, or installability, (and is not based on technical or physical considerations). Partitions do not overlap.

**perfective maintenance:** modification of a software product after delivery to detect and correct latent faults in the software product before they are manifested as failures.

NOTE 1—Perfective maintenance provides enhancements for users, improvement of program documentation, and recoding to improve software performance, maintainability, or other software attributes (from ISO/IEC 14764:2006 ed 2.0, Copyright © 2006 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch)).

NOTE 2—Contrast with adaptive maintenance, corrective maintenance.

**platform:** (A) Type of computer or hardware device and/or associated operating system, or a virtual environment, on which software can be installed or run. (B) A collection of hardware and software components that are needed for a CASE tool to operate. (C) Combination of an operating system and hardware that makes up the operating environment in which a program runs.

NOTE 1—A platform is distinct from the unique instances of that platform, which are typically referred to as devices or instances.

NOTE 2—(A) is from ISO/IEC 19770-1:2017 ed 3.0, 3.9, Copyright © 2017 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch). (B) is from IEEE Std 1175.2-2006 [B3]. (C) is from ISO/IEC 26513:2017 ed 2.0, 3.26 and ISO/IEC/IEEE 24765:2017, Copyright © 2017 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch)

**precautionary principle:** The principle that the introduction of a new product or process whose ultimate effects are unknown or disputed should be resisted until such risks are appropriately mitigated. The precautionary principle denotes a duty to prevent harm, when it is within our power to do so, even when all the evidence is not in. This principle has been codified in several international treaties and in international law.

**primary intent:** intent that is first in importance.

**record element type (RET):** User recognizable sub-group of data element types (DETs) within a data function, and code data group as defined in the “code data” paragraph.

**single instance approach:** The case where each method of delivery of same functionality is counted only once. *See also:* **multiple instance approach.**

**Software Non-functional Assessment Process (SNAP) category:** A group of components, processes or activities that are used in order to meet the non-functional requirement. Categories classify the non-functional requirements (NFR), and are generic enough to allow for future technologies. Categories are divided into sub-categories, so that the SNAP category groups the sub-categories based on the same level of operations and/or similar type of sizing activities.

**Software Non-functional Assessment Process (SNAP) counting unit (SCU):** A component or activity, in which non-functional complexity and size are measured. The SCU is identified according to the nature of each sub-category. It may contain both functional and non-functional characteristics; therefore, sizing an SCU should be performed for its functional sizing, using function point analysis (FPA), and for its non-functional sizing, using SNAP.

**Software Non-functional Assessment Process (SNAP) points (SP):** A unit of measurement to express the size of a non-functional requirement. The non-functional size of the software non-functional assessment process (SNAP) counting units (SCUs) is identified in a sub-category. After identifying all the SCUs that are provided to meet the non-functional requirements (NFR), the SNAP points (SPs) of all SCUs are added together to obtain the calculated SPs.



**Software Non-functional Assessment Process (SNAP) Sub-category:** A component, a process or an activity executed within the project, to meet a non-functional requirement. (A non-functional requirement may consume more than one sub-category).

**Software product:** A set of computer programs, procedures, and possibly associated documentation and data. [B7].

NOTE—From ISO/IEC/IEEE 12207:2017 ed 2.0, Copyright © 2009 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch) [B7].

**software boundary (also referred to as “application boundary”):** A conceptual interface between the software under study and its users. The boundary defines what is external to the application; it indicates the border between the software being measured and the user; it acts as a “membrane” through which data processed by transactions pass into and out of the application. Software boundary is dependent on the user’s external business view of the application.

**software project:** The set of work activities, both technical and managerial, required to satisfy the terms and conditions of a project agreement. A software project has specific starting and ending dates, well-defined objectives and constraints, established responsibilities, a budget and a schedule. A software project may be self-contained or may be part of a larger project. In some cases, a software project may span only a portion of the software development cycle. In other cases, a software project may span many years and consist of numerous subprojects, each being a well-defined and self-contained software project.

**software-intensive system:** A system for which software is a major technical challenge and is perhaps the major factor that affects system schedule, cost, and risk. In the most general case, a software-intensive system is comprised of hardware, software, people, and manual procedures [B2].

NOTE—From IEEE Std 1062-2015 [B2].

**user:** any person or thing that communicates or interacts with the software at any time.

NOTE 1—Examples of “thing” include, but are not limited to, software applications, animals, sensors, or other hardware.

NOTE 2—From ISO/IEC 14143-1:2007 ed 2.0, Copyright © 2007 IEC Geneva, Switzerland. [www.iec.ch](http://www.iec.ch).

**Web Content Accessibility Guidelines (WCAG):** part of a series of web accessibility guidelines published by the Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C®). They are a set of guidelines that specify how to make content accessible, primarily for people with disabilities—but also for all user agents, including highly limited devices, such as mobile phones.

NOTE—WCAG 2.0 was published in December 2008 by the the World Wide Web Consortium (W3C®<sup>11</sup>) and became an ISO standard, ISO/IEC 40500:2012 in October 2012. WCAG 2.1 [B15] was approved on 05 June 2018.

## 3.2 Abbreviations

DET	data element type
EDI	electronic data interchange
EI	external input
EO	external output

<sup>11</sup>W3C is trademarks or registered trademarks of the W3C®, (registered in numerous countries) World Wide Web Consortium. Marks of W3C are registered and held by its host institutions: Massachusetts Institute of Technology (MIT), European Research Consortium for Information and Mathematics (ERCIM), and Keio University, Japan.

EP	elementary process
EQ	external inquiry
FSM	functional size measurement
FUR	functional user requirements
FP	function point
FPA	function point analysis
GUI	graphical user interface
NFR	non-functional requirements
NFSSM	non-functional software size measurement
RET	record element type
SCU	SNAP counting unit
SOA	service-oriented architecture
SNAP	Software Non-functional Assessment Process
SP	SNAP Point
WCAG	Web Content Accessibility Guidelines

## 4. Introductory Information

### 4.1 Non-Functional Software Size Measurement (NFSSM) introduction

In software engineering, size is an important attribute of software. For instance, the size of a piece of software-under-planning is used to calculate the resources (including time) necessary to develop the software, provided historical or benchmark productivity data is available. The size of a piece of software once finished is used for the normalization of certain software performance indicators (e.g., the number of defects found in the software) or for the calculation of the development organization's performance, including its productivity. This information is used to estimate the effort of future software projects.

The size of software can be measured in several ways, with different units of measure. For software that already exists, size can be measured by counting the source statements that form the software's algorithm and data. However, one major disadvantage of “source statement” as a unit of measure is that early in the development of software there are no statements yet to count. The substitute for estimating the number of source statements needed to satisfy the software requirements is quite imprecise due to a dependency on implementation specifics typically decided later in the project. This imprecision can be avoided by basing the sizing activity on present software requirements instead of future source statements.

Several sizing methods have been put forward that are based on software requirements. One of these methods is defined in ISO/IEC 20926:2009, which focuses on a subtype of the software requirements, the FUR, and assigns “function points” (FPs) (the unit of measure) to elements of FUR.

In addition to functional and non-functional size, other project-constraint attributes can be considered to provide perspective to an appropriate level of detail. This includes, for example, development methodology, maturity level of the organization, tools, experience, quality, and collaboration effectiveness between team members.

This standard complements ISO/IEC 20926:2009 by focusing on non-functional (software) requirements. Instead of FPs, this standard assigns SNAP points (SPs) to NFR.

## 4.2 Software-intensive system and software product

Along with software, a software-intensive system may include hardware, data, and other components, such as documents and training. The NFR of the software-intensive system may be met by the software product or by the other components of the software-intensive system. For example, requirements for improving the response time of a system may be met by advanced hardware, by improving the efficiency of the software, or by both. NFR may be also met by the changes to the data maintained or used by the software product, and not by the software product.

Non-functional software size measures the software product, it does not measure the NFR that are met by non-software components of the software-intensive system.

## 4.3 Software domains

ISO 14143-1:2007 defines software functional domain as a “class of software based on the characteristics of Functional User Requirements.” The term “software domains” replaces the phrase “software types” (which have been loosely defined) and is used to differentiate between categories of software products, so that performance indicators can be compared within similar types of software products.

ISO/IEC 14143-1:2007 requires that a Functional Size Measurement (FSM) Method shall describe the functional domain(s) to which it can be applied. ISO/IEC TR 14143-5:2004 defines the method to determine the functional domain.

A non-functional software size measurement (NFSSM) should use the same method of functional domains, as described in ISO/IEC 14143-1:2007 when comparing the performance of the non-functional aspects of software products or software projects.

## 4.4 The relations between non-functional requirements (NFR) definition and functional user requirements (FUR)

### 4.4.1 NFR

The boundary between functional requirements and NFR does not have a universally-agreed definition. ISO/IEC/IEEE 24765:2017 [B10] defines NFR as “a software requirement that describes not what the software will do, but how the software will do it.” ISO/IEC 25010:2011 defines a product quality model and categorizes product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability). ISO/IEC 25010:2011 supersedes ISO/IEC 9126:1991, which identified six quality characteristics, which were intended to be exhaustive. As the software capabilities evolve, the definition of NFR and their boundary with functional requirements may change, and a new standard may supersede ISO/IEC 25010:2011 in the future.

This standard defines how to size the NFR of the software product irrespective of the way NFR are defined. Instead, a non-functional requirement is mapped to the category and sub-category of the software non-functional sizing method. SNAP sub-categories do not define or describe NFR; they classify how these requirements are met within the software product.

### 4.4.2 The relations between NFR and SNAP sub-categories

The implementation of a non-functional requirement, which can be defined using ISO/IEC 25010:2011 or any other standard, should be sized using SNAP sub-categories. SNAP measures the size of the implementation, and not the non-functional requirement.

The categories and sub-categories and the method to size the sub-categories are defined in [Clause 5](#). They are as follows::

- a) Category 1—Data operations
  - 1) Sub-Category 1.1—Data entry validation
  - 2) Sub-Category 1.2—Logical and mathematical operations
  - 3) Sub-Category 1.3—Data formatting
  - 4) Sub-Category 1.4—Internal data movements
  - 5) Sub-Category 1.5—Delivering added value to users by data configuration
- b) Category 2: Interface design
  - 1) Sub-Category 2.1—User interfaces
  - 2) Sub-Category 2.2—Help methods
  - 3) Sub-Category 2.3—Multiple input methods
  - 4) Sub-Category 2.4—Multiple output methods
- c) Category 3: Technical environment
  - 1) Sub-Category 3.1—Multiple platforms
  - 2) Sub-Category 3.2—Database technology
  - 3) Sub-Category 3.3—Batch processes
- d) Category 4: Architecture
  - 1) Sub-Category 4.1—Component-based software
  - 2) Sub-Category 4.2—Multiple input/output interfaces

A non-functional requirement may be implemented in the software product by using one or more SNAP sub-categories. Accordingly, a sub-category may cater for several non-functional characteristics, as defined by these standards, for example, a requirement to improve the way system recovers from a crash. Using ISO/IEC 25010:2011, this requirement falls under the area of reliability, and the sub characteristic is recoverability.

The software product is designed as follows:

- An algorithm is added to identify corrupted data in specific fields
- Time stamps are added to database records
- An algorithm is written to reconstruct corrupted data using uncorrupted record

The design involves the following SNAP sub-categories:

- Database technology (adding time stamp)
- Logical and mathematical operations

In this example, the “recoverability” type of non-functional requirement is mapped to two SNAP sub-categories, “Database Technology” and “Logical and Mathematical Operations.”

More examples are illustrated in [Figure 1](#).

		Snap Categories and sub-categories													
		Data Operation					Interface Design				Technical Environment			Architecture	
		Data Entry Validations	Logical and Mathematical Operations	Data Formatting	Internal Data Movements	Delivering Added Value to Users by Data	User Interfaces	Help Methods	Multiple Input Methods	Multiple Output Methods	Multiple Platforms	Database Technology	Batch Processes	Component-based Software	Multiple Input/Output Interfaces
ISO/IEC 25010:2011 Sub-characteristics (Partial list)		1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	2.4	3.1	3.2	3.3	4.1	4.2
Portability	Replaceability														
	Installability														
	Adaptability														
Security	Accountability														
	Authenticity	Ex. 2													
Reliability	Fault Tolerance														
	Recoverability														
Usability	Accessibility														
	User Error Protection														
	Operability								Ex. 4						
	User Interface Aesthetics						Ex. 1								
	Learnability														
	Appropriateness Recognizability														
Performance Efficiency	Capacity														
	Resource Utilization														
	Time Behavior											Ex. 3			
Functional Suitability	Functional Completeness														
	Functional Appropriateness														
	Functional Correctness														

**Figure 1—Mapping ISO/IEC 25010:2011 characteristics to SNAP sub-categories**

**Example 1:** Improving the understandability and learnability by adding pop-up help menus (Sub-category 2.1) and rearranging the screens (Sub-category 2.2).

**Example 2:** Improving security by adding more validations to the authentication process, using Sub-category 1.1 “Data Entry Validations” and Sub-category 1.2 “Logical and Mathematical Operations.”

**Example 3:** Improving performance by adding indices to the database and improving queries (Sub-category 3.2).

**Example 4:** Adding barcode reading as additional input method (Sub-category 2.3).

## 4.5 Key features of the NFSSM

A non-functional size assists organization in multiple ways. It provides insight into the delivery of software projects and maintenance of software applications to assist in effort estimating and in the analysis of key performance indicators, such as quality and productivity. Used in conjunction with FSM, the non-functional size provides information that can identify items impacting quality and productivity in a positive or negative way.

Having this information enables software professionals to do the following:

- a) Better plan and estimate projects
- b) Communicate the impact of NFR on the project with users and customers
- c) Compare project and analyze trends of improvement, and compare it to benchmarks
- d) Identify areas of improvement
- e) Assist in determining future non-functional strategies
- f) Quantify the impacts of the current non-functional strategies
- g) Provide specific data when communicating non-functional issues to various audiences
- h) A method to help users determine the benefit of an application package to their organization by assessing portions or categories that specifically match their requirements
- i) Determine the non-functional size of a purchased application package

Other key features include the following:

- j) NFSSM is independent of the way NFR are defined.
- k) This standard contains rules on how to use ISO/IEC 20926:2009 (IFPUG FSM) and NFSSM together, so that there should be no gaps and no overlaps between functional size and non-functional size. A requirement that contains both functional and non-functional aspects should be sized using ISO/IEC 20926:2009 for its functional aspects and SNAP for its non-functional aspects.
- l) FSM and NFSSM together should provide a complete view of the size of the software product.<sup>12</sup>
- m) This standard contains rules on how to use both functional size and non-functional size to calculate project performance (such as quality and productivity).

## 4.6 Future evolution of NFR

### 4.6.1 The challenge

As systems evolve, new variations on the types of requirements emerge from the enhanced capabilities of software. Machine learning, artificial intelligence, and the Internet of Things generate the need for new types of requirements, such as ethics, social responsibility, privacy, and sustainability. Similar to other NFR, such as security and accountability, these attributes shape the functional behaviors of the system and its components.

These requirements may fail to gain sufficient notice by software development teams until after a user finds a fault or a logic error or the system in use has had unintended consequences on user communities. Then everyone takes notice, with software development teams working to find corrective and preventive mitigations to address what may not fit the current definitions of functional requirements and the classification of NFR.

Similar to other requirements that are intended to shape system behaviors through the assurance of adherence to defined values, emerging requirements should reflect stakeholder values, some of which are often not explicitly defined or beyond the immediate access of the development environment (e.g., robotics as prosthetics). Software development teams need to understand the system's context of use, the risks of harm, then the application of the "precautionary principle" as the risk mitigation technique necessary to anticipate and mitigate against undesirable consequences such as harm to life of any kind and quality of life, the environment, a species, or a community.

---

<sup>12</sup>Future non-functional requirements may evolve in time. See also 1.1 for possible future enhancements of SNAP.

#### 4.6.2 Current classification of these NFR

In ISO/IEC 25010:2011 terms, these emerging social requirements falls within the category of Freedom from risk. It defines Freedom from risk as:

The quality in use of a system characterizes the impact that the product (system or software product) has on stakeholders. It is determined by the quality of the software, hardware and operating environment, and the characteristics of the users, tasks and social environment. All these factors contribute to the quality in use of the system.

Systems should be evaluated by the following criteria:

- How risk free is the use of the system?
- How risk free does updating the content of the system need to be?
- How risk free does making maintenance changes to the system or porting the system need to be?
- How risk free does using the output from the system need to be?

#### 4.6.3 Sizing quality-in-use requirements

Meeting these user requirements consumes time, effort, and other resources that should be evaluated and considered when planning and managing a software project. Therefore, like other user requirements, they should be sized.

SNAP can be used to size quality-in-use requirements. For example, algorithms to analyze if a certain situation may impose a risk and assessing the probability and the impact of such a risk can be sized using Sub-category 1.2—Logical and Mathematical Operations. Storing data for this purpose may be sized using Sub-category 3.2—Database Technology. However, the current set of sub-categories may be enhanced in the future to cover more aspects of these new requirements.

### 4.7 Objectives and benefits

#### 4.7.1 Objectives

SNAP measures software by quantifying the size of NFR. With this in mind, the objectives of SNAP are to:

- a) Measure the non-functional size of the software that the user requests and receives
- b) Demonstrate the full economic value of the application, including its functional aspects as well as the non-functional aspects (have the non-functional application size in addition to the functional application size, to demonstrate the full economic value).
- c) Measure software development and maintenance projects based on the NFR
- d) Size technical projects, to which FPA is not applicable

In addition to meeting the above objectives, the process of sizing NFR should be:

- e) Simple enough to minimize the overhead of the measurement process.
- f) A consistent measure among various projects and organizations. SNAP allows determining (by counting each of the four categories, from each one of the sub-characteristics) the possibility of sizing and, therefore, better estimate a project with/without FPs, according to the set of user requirements received for a project.



### 4.7.2 Benefits

Non-functional sizing assists organizations in multiple ways. It provides insight into the delivery of projects and maintenance of applications to assist in estimating and in the analysis of quality and productivity. Used in conjunction with FSM, the non-functional sizing provides information that can identify items impacting quality and productivity in a positive or negative way.

Having this information enables software professionals to:

- a) Better plan and estimate projects
- b) Identify areas of process improvement
- c) Assist in determining future non-functional strategies
- d) Quantify the impacts of the current non-functional strategies
- e) Provide specific data when communicating non-functional issues to various audiences

Organizations can apply SNAP as:

- A method to measure the non-functional size of a software product to support quality and productivity analysis
- A method to estimate cost and resources required for software development and maintenance
- A method to measure cost reduction for software development and maintenance, in addition to FPA
- A normalization factor for software comparison
- A method to determine the non-functional size of a purchased application package by sizing all the portions and categories included in the package
- A method to help users determine the benefit of an application package to their organization by sizing portions or categories that specifically match their requirements

NOTE 1—"Function Points + SPs" are not equal to the overall product size:

As of the date of this publication, the size of a software application is considered to have two distinct parts: the size of the FURs and the size of the NFRs. For example, if an application's functional size is 700 FPs and non-functional size is 200 SPs, then the entire size could be stated as "700 FPs, and 200 SPs." The two sizes do not sum up to "900 points."<sup>13</sup>

The ISO/IEC 20926:2009 FSM (IFPUG functional sizing methodology) does not change when measuring the NFR using SNAP.

NOTE 2—A project may have zero Function Points and a nonzero number of SPs, or 0 SPs and a nonzero number of Function Points, or any combination of Function Points and SPs

## 5. Non-functional size: Categories and sub-categories

### 5.1 Category 1: Data operations

The data operations category relates to how data is processed within the SNAP counting unit (SCU) to meet the NFR in the application.

<sup>13</sup>See 8.2.4 for information on how "equivalent size" is calculated.



### 5.1.1 Sub-category 1.1: Data entry validation

Operations that are performed either to allow only certified (predefined) data or to prevent the acceptance of uncertified data.

#### 5.1.1.1 The SNAP Counting Unit

The SCU is the elementary process (EP).

#### 5.1.1.2 Terms

##### 5.1.1.2.1 Nesting level

The number of conditional validations (If-Else combo/“While” loop/“For” loop or any other validation blocks) in the longest chain of validation.

##### 5.1.1.3 Complexity parameters

The nesting level complexity is shown in [Table 1](#)

**Table 1—Nesting level complexity**

Complexity level	Condition
Low complexity	2 nesting levels or fewer
Average complexity	3 to 5 nesting levels
High complexity	6 nesting levels or more

The number of unique DETs used across all validations. For example, there are two fields validated in the SCU, one uses three DETs for nested validation and another uses one DET, which is not one of the three DETs above, count four DETs.

##### 5.1.1.4 Calculation method

Identify the complexity based on nesting level (see [Table 2](#). Calculate SPs based on the constant factor and the number of DETs (#DETs).

**Table 2—SNAP sizing for data entry validations**

	Nesting level complexity		
	Low	Average	High
	1-2	3-5	6+
SP =	2 × the number of DETs	3 × the number of DETs	4 × the number of DETs

NOTE 1—Data entry may result from any source (e.g., UI, transactions).

NOTE 2—Number of nesting level is derived from the business requirements and high-level solution, and not from how the code is written.

NOTE 3—Validations are nested when there is a dependency between validations.

Example: A number must be between 0 and 10; if it is less than 1, it must have two digits after the decimal point. If it is 1 or more, it may have one or no digits after the decimal point.

NOTE 4—Several validations on a field are not nested when they are independent.

Example: A value must be numerical, greater than 0 and smaller than 1 000 000.

NOTE 5—This sub-category may include requirements for error handling or exceptions handling.

NOTE 6—DETs refer to all types of data.

NOTE 7—If code data is used for data entry validations, then any changes to code data values (add/change/delete) is counted using this category.

### Examples:

- a) A date field must have a certain size; a value entered must be in a certain range of values; a code must be present in a certain table; a field is bound to the value of the previous field (e.g., state, county, city).
- b) Data entry validation enabled using code data for validation of airport names.<sup>14</sup> A traveling order application has a screen with details of the departure airport, destination airport and the option to add multiple destinations. The current system validates the airport abbreviations (such as LHR, NYC) but cannot identify airport name. In this example, the user requirement is that the user shall be able to key in either the abbreviation or the airport name. The design is to use an existing code data with all airports and IATA airport codes, and to add validation rules both for the airport abbreviation and airport name. Three EPs were identified (order a flight, amend order, cancel order) using this code data validation. One nesting level and one DET are used for SNAP counting.
- c) Data entry validation enabled using logical checks on the DETs—adding an employee to an organization. In the processing logic of adding an employee to an organization, the number of validations performed during data entry on a screen and the complexity level of these validations, are considered non-functional. Employee ID, in this example, is not generated automatically but manually entered besides employee name, department, date of birth and more. Validating that the set of data is correct is sized using this subcategory. (These are considered to be technical requirements.) As per ISO/IEC 20926:2009, some operations, which are performed in the EP to ensure valid data, are added to the data information. The data elements in these operations are not visible to the user (although agreed with the user) and not counted as DETs in FP. Data elements that are involved in these hidden operations should be counted using SNAP model.

## 5.1.2 Sub-category 1.2: Logical and mathematical operations

### 5.1.2.1 Extensive mathematical operations

SNAP defines an “extensive mathematical operation” as a mathematical operation that includes one or more algorithms. An algorithm is defined for SNAP as a series of mathematical equations and calculations executed in conjunction with, or according to, logical operators to produce results identifiable to the user. Examples of extensive mathematical operations include using the Program Evaluation Review Technique (PERT) to calculate the expected completion date of a project, calculating the optimal profit for a business process using linear programming, determining the way to formulate the fastest flowing waiting lines using queuing theory, and finding the shortest route through a network. Examples of other algorithmic mathematical operations fitting the definition of “extensive” include solving calculus integration formulas, calculating federal income taxes, GPS calculations, gaming, weather forecasting, and perhaps calculating retirement pensions.

The DETs counted are the set of those required to operate the extensive mathematical operation, such as values for an algorithm’s variables and settings maintained by the algorithm’s control information. These values and settings are not necessarily stored in a single physical file; they may be stored in various locations such as settings of the value of variables located in the code or as DETs in various physical files. However they

<sup>14</sup>See 5.5: Sizing code data.

are located, as a set(s), this satisfies the requirements for either an internal logical file(s) or external interface file(s) because they are the logical grouping(s) of data necessary to operate the algorithm.

“Simple” or “routine” mathematical operations are defined here as not using algorithms. Examples can include adding a column of numbers, balancing a checking account, totaling daily sales, and calculating averages. Also, an application may require a simple or routine mathematical operation to be iterated many times. For example, a fast-food restaurant manager may need to place an order for ketchup packets from a supplier. The manager first counts the current inventory of ketchup packets, forecasts the expected usage, and places an order to make up for the expected resulting shortfall. This is a simple or routine mathematical operation. If the manager has 100 types of items in inventory and must perform this calculation 100 times to complete the total order with the supplier, then this is still defined as being a simple or routine mathematical operation because the simple or routine mathematical operation is iterated 100 times—“extensive” refers to the depth of the algorithm(s), not to the number of simple or routine calculation iterations needed.”

Sizing an EP is determined by the type of processing logic used by the external input (EI), external output (EO), or external inquiry (EQ). While this can give a higher size to the EP that contains mathematical operations, it does not necessarily correlate to the effort needed to produce extensive mathematical operations. SNAP size compensates for the additional complexity of extensive mathematical operations.

### 5.1.2.2 Extensive logical operations

SNAP defines an “extensive logical operation” as a logical operation either containing a minimum of four nesting levels, containing more than 38 DETs required to operate the logical operation, or both. These DETs do not necessarily have to cross the application boundary.

The outcome of a logical operation may be a decision or set of decisions or evaluating a condition using data that exist in one or more logical files. The SCU is the EP. If more than one logical operation can be executed within the EP, then count either the combined number of DETs in the operations containing a minimum of four nesting levels, or count the sum of the DETs involved in all of the logical operations assuming that the sum is more than 38 (whichever is larger).

### 5.1.2.3 SNAP Counting Unit (SCU)

The EP.

### 5.1.2.4 Complexity parameters

SCU complexity parameters are as follows:

- FTR density of the logical file being accessed to do the business logic processing (see [Table 3](#))

**Table 3—FTR density, logical and mathematical operations**

	FTR density		
	0–3 FTR	0–3 FTR	0–3 FTR
Complexity Level	Low	Average	High

- Processing logic type of the EP (logical/mathematical) (see [Table 4](#))

**Table 4—EP type for logical and mathematical operations**

EP Type	Main Purpose of the EP
Logical	Decision making or evaluating a condition using data that exist in one or more logical files (internal and/or external) Example: Exception processing
Mathematical	Transformation of data and/or use of control information that exists in one or more logical files (internal and/or external) that is used for an extensive mathematical operation. Example: Complex tax calculation

— Number of data elements types (#DETs) (see Table 5)

**Table 5—SNAP sizing for logical and mathematical operations**

	Complexity Level		
	Low	Average	High
	SP=	SP=	SP=
EP type: Logical	4 × the number of DETs	6 × the number of DETs	10 × the number of DETs
EP type: Mathematical	3 × the number of DETs	4 × the number of DETs	7 × the number of DETs

FTR density factor is measured as follows: Type of the EP (logical/mathematical). Identify the type of the EP.

### 5.1.2.5 Calculation method

Identify FTR density based on the number of FTRs. Calculate size based on FTR density, EP type, and the number of DETs.

NOTE—When the main purpose cannot be clearly identified, select “Logical” (do not count it as one logical and one mathematical).

Examples of extensive logical and mathematical operations include the following:

- Project scheduling critical path analysis
- Complex tax calculations
- Linear programming algorithms
- Calculus integration formulas
- Financial return on investment calculations for a large industrial machine
- Statistical analysis of variance calculations
- Business sales forecasting using the ensemble forecasting method

### 5.1.3 Sub-category 1.3: Data formatting

A requirement that deals with structure, format, or administrative information in a transaction not directly relevant to functionality that is seen by the user.

#### 5.1.3.1 SNAP Counting Unit (SCU)

The EP.

### 5.1.3.2 Complexity parameters

Sub-category 1.3 complexity parameters include the following:

- a) Transformation complexity
  - 1) **Low:** Data type conversions or simple formatting such as byte padding, or data substitution, using a maximum of 2 operators (Celsius to Fahrenheit, single integer to double integer).
  - 2) **Average:** Involves encryption/decryption, which is a characteristic of the application and applies to almost all processes that is provided through a library—API interface.
  - 3) **High:** Involves local Encryption/Decryption.
- b) Number of data elements types (#DETs) transformed.

### 5.1.3.3 Calculation method

Identify the complexity based on transformation. Calculate SPs based on the constant factor and the number of DETs (#DETs) (see Table 6).

**Table 6—SNAP sizing for data entry validations**

	Transformation complexity		
	Low	Low	Low
SP=	2 × the number of DETs	3 × the number of DETs	5 × the number of DETs

NOTE 1—Data elements refer to all types of data

NOTE 2—The encryption algorithm is complex for:

- Design specifically allowed to several key lengths
- Provide a method to check data integrity for high volume data
- Formatting medical images
- Restructure huge volume database etc.

### Examples:

- a) Simple transformations:
  - 1) Convert text to value or value to text; convert strings to values.
  - 2) Data formatting required for reporting requirements.
  - 3) An application shows the date (MMDDYYYY), time (GMT), current atmospheric temperature (degrees Fahrenheit) in a standard format. However, due to regulations, the date is required to be displayed as ‘YYYYMMDD, the time should always show the local time zone, and temperature should be displayed as “Degrees Kelvin.”
  - 4) The display formats need to be converted to adhere to the standards prescribed.
- b) Complex transformations:
  - 1) Enabling multi-lingual support for an application by using code data.
  - 2) Encryption/ decryption, compression–decompression.

- 3) Compliance with standards for the electronic transfer of data in a healthcare environment. The data packets are sent and received in a particular format called electronic data interchange (EDI). For example: change the structure of the transactions (add headers and footers); the transaction format is changed per HIPAA (Health Insurance Portability and Accountability Act of 1996) [B1] requirements with no changes in the functionality.
- 4) Data interchange formats—XML to other formats, or other means of data interchange between two computer systems.
- 5) Preparation of metadata for various screen requirements or data warehouse views.
- 6) Transformations in data warehouse.

#### 5.1.4 Sub category 1.4: Internal data movements

Data movements from one partition to another within application boundary with specific data handling.

##### 5.1.4.1 SNAP Counting Unit (SCU)

The portion of the EP that crosses from one partition to another.

The SCU is identified by the EP and the two partitions crossed. If an EP crosses more than two partitions, use the formula below per each partition crossing [in Figure 2, an EP may move from Component 1 to Component 2 (labeled “A”), and then to Component 3 (labeled “B”). In such a case, SPs shall be calculated at each partition crossing].

NOTE—An EP is the smallest unit of activity that is meaningful to the user. While ISO/IEC 20926:2009 (IFPUG FSM) refers to transactions that cross the application boundary—for Sub-category 1.4 SCU—this standard refers to their internal processes/functions, which move from one partition to another.

##### 5.1.4.2 Complexity parameters

Sub-category 1.4 complexity parameters include the following:

- a) The number of unique data elements types (#DETs) transferred from one partition to the other, and are processed and/or maintained.
- b) The number of unique FTRs either read or updated by the EP at both partitions crossed.

##### 5.1.4.3 Calculation method

Identify the complexity level based on the number of FTRs read/updated and number of DETs transferred. Calculate size as per Table 7 for each partition crossing.

**Table 7—SNAP sizing for internal data movements**

	Complexity Level		
	Low	Low	Low
	(0–3 FTR)	(0–3 FTR)	(0–3 FTR)
SP =	4 × the number of DETs	6 × the number of DETs	10 × the number of DETs

NOTE 1—Internal data movements sub-category sizes internal transactions within the boundary of an application. These transactions are used in case they cross partitions.

NOTE 2—Internal data movements are counted by SNAP for functional transactions as well as non-functional transactions.

For example: Querying fields on a front-end application by using data that is stored in the back-end application.

NOTE 3—Any data transaction (functional) that crosses partitions, generates SPs in addition to FPs.

NOTE 4—When an EP crosses the partition in both directions, SNAP is sized as follows:

- One SCU, if the transactions are synchronous
- Two separate SCUs, if the transactions are a-synchronous

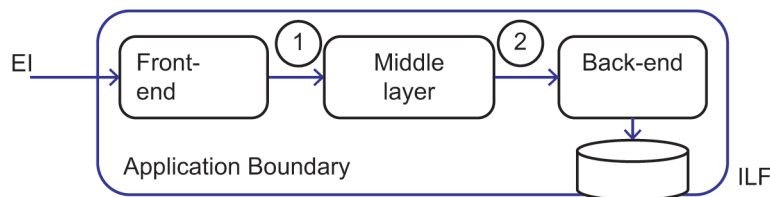
The following are examples of EPs that may have data crossing partitions:

- a) Data backup within application boundary, crossing partitions.
- b) Data copy/movement between tables within application boundary, crossing partitions.
- c) Realign data in temporary storage.
- d) Transactions between application and middleware that do not cross functional boundaries.
- e) Service-oriented architecture (SOA) solutions. (When SOA functionality are within the application boundary.)
- f) Data movements between tiers that do not cross functional boundaries.
- g) Data formatting transactions that use data that crosses partitions.
- h) Internal transactions for logical /mathematical operations.
- i) Reference data maintenance.

Sub-category 3.3 (Batch Processes) covers batch jobs. Batch jobs may be executed within a single partition. If the batch job crosses partitions, then it may need to be sized for additional impacts by this sub-category.

Example:

An EP “process invoice” has two partition crossings as shown in Figure 2:



**Figure 2—Example: Internal data movement**

Each component references/updates a unique set of FTRs during the processing as follows:

- Partition 1—2 FTRs
- Partition 2—4 FTRs
- Partition 3—3 FTRs

For this process six DETs are crossing from Partition 1 to Partition 2 and five (5) DETs are crossing from Partition 2 to Partition 3.

For the SCU “1 crossing” (from front-end to middle layer):

- Number of FTRs =  $2 + 4 = 6$  (Average complexity)
- Number of DETs = 6
- SP =  $6 \times$  the number of DETs = 36

For the SCU “2 crossing” (from middle layer to back-end):

- Number of FTRs =  $4 + 3 = 7$  (Average complexity)
- Number of DETs = 5
- SP =  $6 \times$  the number of DETs = 30

### **5.1.5 Sub-category 1.5: Delivering added value to users by data configuration**

Additional unique business value to users that is provided by adding, changing or deleting reference data/ code data information from the database or data storage with no change in software code or the database structure.

#### **5.1.5.1 SNAP Counting Unit (SCU)**

The EP per logical file.

NOTE—The SCU is the EP to consume the added value in the logical file and not the process to create or modify the configuration

Example: A new service is defined by adding its attributes to reference tables. The application is flexible enough to provide the new service with no code changes. EPs to be counted may be: add the new service; change this service; cease the service. The process to add the service attributes to the reference tables (i.e., writing and using scripts) should not be counted.

In case the configured data impacts several EPs, each EP is counted separately.

#### **5.1.5.2 Terms**

##### **5.1.5.2.1 Attribute**

An independent parameter that has a unique business meaning and contains a set of different values.

##### **5.1.5.2.2 A record**

One row in a logical file.

##### **5.1.5.2.3 A Logical file**

A user recognizable group of logically related data or control information.



### 5.1.5.3 Complexity parameters

The complexity parameters for Sub-category 1.5 are as follows:

- a) Number of unique attributes involved in the EP, that are added/modified/deleted
- b) Number of records configured

### 5.1.5.4 Calculation method

Identify the complexity level based on #Records. Calculate size based on the constant factor and the #Attributes (see [Table 8](#)).

**Table 8—SNAP Sizing for delivering added value by data configuration**

	Complexity level	Complexity level	Complexity level
	Low	Low	Low
	1–10 records	1–10 records	1–10 records
SP=	$6 \times$ the number of attributes	$8 \times$ the number of attributes	$12 \times$ the number of attributes

NOTE 1—New services, products, price plans etc., can be added to the application by adding or changing reference data and not by writing code.

NOTE 2—Functionality by data configuration brings added value to the user, also adds effort to configure and test the new functionality.

Examples of delivering added value to users by data configuration are as follows:

- a) An application requires granting access to a specific role in the application. To meet this requirement, the developer does not write any separate code and instead updates a configuration file, and associates the user or set of users into some property file(s). Such additions or changes are made to meet user requirements, which affect the functionality at the EP level.
- b) The process to configure the data in the database is not sized separately. Only the user's processes should be counted.
- c) An application requires configuring a new product ("Product X" here below) or a component that can be sold using the application. The new product and its price plan are defined in reference data. The project effort may be creating the data, by migrating it to the reference files and testing that the application functions with the new data. The sizing process identifies many SCUs here.
  - 1) Change product Y to product X.
  - 2) Provide a new product X.
  - 3) Change price of product X, etc.

## 5.2 Category 2: Interface design

The interface design category relates to the end user experience. This category assesses the design of graphical UI processes and methods that allow the user to interface with the application.

## 5.2.1 Sub-category 2.1—User interfaces

Unique, user identifiable, independent graphical user interface elements added or configured on the user interface that does not change the functionality of the system but affects non-functional characteristics (such as usability, ease of learning, attractiveness, accessibility).

### 5.2.1.1 SNAP Counting Unit (SCU)

Set of screens as defined by the EP.

### 5.2.1.2 Terms

#### 5.2.1.2.1 UI element

UI element (user interface element) is a unique user identifiable structural element that makes up the user interface. It includes elements such as:

- Window (which can be container, child, text or message window)
- Menus
- Icons
- Controls
- Pointer (or mouse cursor)
- Text box
- Button
- Hyperlink
- Drop-down list
- List box
- Combo box
- Checkbox
- Radio button
- Cycle button
- Datagrid
- Tabs
- Interaction elements like a cursor
- Labels

The above controls are used to display or manipulate data objects. The aspect that adds to the complexity of the design, configuration and testing time of a user interface is the configuration of each of these elements. NFR may involve changing the properties of these UI elements. Depending upon the type of the user-interface element, varying number of properties can be configured to produce the desired output. For example, button could be set to “locked” or “highlighted” or “colored” or placed at a particular location on the screen.

#### 5.2.1.2.2 UI element properties

Each UI element is associated with certain properties, which define the behavior and look and feel of the user interface element. The following examples come from Clause 18 of the W3C Recommendation on Cascading Style Sheets [B13]:

- A window would have properties like background color, active border, active caption, etc.
- A button can have properties like button highlight, button text, background color, etc.
- Tool tips can have properties like info text, info background, etc.

#### 5.2.1.2.3 UI element set

A UI element set is the collection of all the UI elements of the same type in the SCU.

Example: All the text boxes in the set of screens (SCU).

#### 5.2.1.3 Complexity parameters

The complexity parameters for Sub-category 2.1 include the following:

- a) The sum of the number of unique properties configured for each UI element in the SCU
- b) The number of unique UI elements impacted

#### 5.2.1.4 SNAP Points (SPs) calculation

Identify the complexity based on the number of properties of UI element set. Calculate size as the product of the constant factor and the number of unique UI elements (See Table 9).

**Table 9—SNAP sizing for user interface**

	UI Type complexity		
	Low	Average	High
	< 10 properties added or configured	10 to 15 Properties added or configured	16+ Properties added or configured.
SP =	$2 \times$ the number of unique UI elements	$3 \times$ the number of unique UI elements	$4 \times$ the number of unique UI elements

#### 5.2.1.5 Rules

The following rules for Sub-category 2.1 apply:

- If the process for adding/changing of UI is FP counted, then do not duplicate the count; however, changing the contents and appearance of a GUI element needs to be sized as non-functional. Aesthetic changes in UI screen, static or dynamic UI pages, rearranging of the screen and printed reports should be sized under user interfaces sub-category.
- The sets of screens within one process shall be counted as the SCU.

NOTE 1—UI elements added may be of any form, such as text, sound, picture, logos, color, keys, controls, navigation, animating or enabling/disabling the above (when such enabling/disabling is not functional).

NOTE 2—Added operations that support: function keys; auto fill; short cut keys; common keys; navigation screen/page level.

NOTE 3—Screens that are not considered as functional (such as Administrator’s screens) and hence are not counted using FPs, are counted by SNAP using 2.1—User Interfaces Sub-category.

**Example:**

Some text on users’ screens is hard coded. Due to a new policy of the company, the request is to replace the word “customer” with the words: “business partner.”

The analysis found that the word “customer” should be replaced in the following UI elements (note that since SNAP counts the number of unique UI elements, there is no need to estimate the number of occurrences of each unique UI element).

- SCU 1: Acquire a new business partner: header, labels, radio button, drop-down list
- SCU 2: Modify details of a business partner: header, labels
- SCU 3: Send a message to a business partner: header, labels,
- SCU 4: Cease services to a business partner: header, labels

Changing the text in these UI elements is considered one property. UI type complexity is low.

$SP = 2 \times \text{the number of unique UI elements per each SCU:}$

$$SP = 2 \times (4 + 2 + 2 + 2) = 20 \text{ SP}$$

## **5.2.2 Sub-category 2.2—Help methods**

Information provided to the users, which explains how the software provides its functionality or other supportive information provided to users.

### **5.2.2.1 SNAP Counting Unit (SCU)**

The Help Object (see 5.2.2.2.1).

#### **5.2.2.2 Terms**

##### **5.2.2.2.1 Help Object**

A “Help Object” is a particular part of the software for which the Help Item is provided.

##### **5.2.2.2.2 Help Item**

A “Help Item” is the smallest, unique, user identifiable information topic, which provides the user with supportive information or details about a particular part of the software.

Examples of Help Objects can be a DET in an EP, a report, a static web page, an icon or a picture on a screen. A few examples are described in the [Table 10](#).

**Table 10—Examples of Help Objects and Help Items**

Help Object	Help Item example
A report	A framed text explaining how to interpret a report, which is displayed by pressing a “?” mark near the header of the report
A DET (e.g., an attribute entered or displayed, a control)	A pop-up text box which is displayed when hovering over a field
A menu item	A pop-up text box which is displayed when hovering over a field
A static web page (a static web page is one that is delivered to all the users exactly as stored, displaying the same information to all users, and is not generated by an application).	The static web page
A link to an explanation page (e.g., to a “Change password” screen, explaining how to select a strong password)	A Section in the “Help” window, explaining the rules for a password strength
A screen	A framed text explaining the purpose of the EI
A link or an icon to open a multimedia item such as a voice message or a movie*	A movie
An FAQ	The FAQ text

\*SNAP does not refer to the size and the costs to produce the multimedia item, only the size for addressing those Help Items in the software.

### 5.2.2.3 Complexity parameters

The inclusion of screenshots in the Help Item.

### 5.2.2.4 SP calculation

Count the number of Help Objects and divide by 16 to calculate the SPs. However, if there is at least one screen shot accompanying the Help Object, count the number of Help Objects, divide by 16, and add 2 (see Table 11).

**Table 11—SNAP sizing for Help Items**

	With no screen shots	With at least one screen shot
SP=	$\frac{\# \text{ of Help Objects}}{16}$	$\frac{\# \text{ of Help Objects} + 2}{16}$

NOTE 1—The media type of the Help Item is immaterial for SNAP. For example, information provided to the user may be provided by a drop down, static web page, paper, or other media type. However, if the referenced Help Objects are the same, then the SNAP size of each media type is the same.

NOTE 2—Do not count multiple instances of the same Help Items, regardless of media type, with the exception of a general user Help Manual. A general user Help Manual is countable in itself.

NOTE 3—Do not add SNAP size for Help that is a help data function per ISO/IEC 20926:2009 (IFPUG FSM).

NOTE 4—This standard does not currently attempt to size help delivered by a person or persons (i.e., help that is not part of the automated software being delivered). This includes phone calls to a help desk, chat rooms, forms of video conferencing, training programs, etc. It also does not address any form of help provided by online videos or similar products. It does not address help for an index, glossary, table of contents, etc.

### Examples:

Below is a hypothetical screen representing an EI for students registering for a University summer program. Three examples of help are identified for this screen (EI) as described in examples 1, 2, and 3. The following examples refer to [Figure 3](#) to [Figure 7](#).

Last name:	XXXXXXX	Residency address:	XXXXX
First name:	XXXXXXX	Building	XXXXX
Room #	XXXXXXX	Telephone #	XXXXX
Courses registered for: XXXXXXXXXXXX XXXXXXXXXXXX			
Home address:			
# and street	XXXXXXXXXX		
City	XXXXXXXXXX		
State	XX		
ZIP	XXXXX		
ENTER			ERROR MESSAGE

**Figure 3—A screen example**

#### Example 1—Screen-level help

Suppose that help is provided which describes the help is provided describing the purpose of the screen as a whole but does not go into detail about the meaning of the individual fields (or DETs). The screen is the “Help Object” and the information provided which explains the purpose of the screen is the “Help Item.” The non-functional size of the help is based on the number of Help Objects affected. We count one “Help Object” in this example.

#### Example 2—Pop-up text box help

A pop-up text box is displayed when hovering over each of the fields that provided information for the meaning of the corresponding field. This includes the ENTER key and the ERROR MESSAGE DET. In this situation, each DET is a Help Object and each pop-up text box is an individual Help Item. If the user recognizes 14 DETs here, there are 14 Help Objects for this example.

#### Example 3—User manual help

A user manual provides information for the meaning of the 14 DETs; this includes the ENTER key and the ERROR MESSAGE DET. In addition, a screen shot of this input screen is included in the User Manual to aid in the explanation. In this situation, count 14 Help Objects and count one screen shot. A screen shot was also found to correlate with work effort to develop help.

#### Example 4. Help for other SNAP subcategories.

Under SNAP 1.2, Logical and Mathematical Operations, SNAP recognizes an algorithm containing 16 countable DETs under SNAP rules. Help is provided which describes the theory of this algorithm in detail to include how the algorithm uses each of these 16 DETs. Count 16 Help Objects for the algorithm. Suppose this help is provided as several pages of text in a user’s manual. Then this text is one Help Item addressing 16 Help Objects.

#### Example 5—Menu help

Pop-up balloon help is provided for buttons at the top of a word processor screen, such as the “Bold” button or the “Italics” button. Count one Help Object for each button having balloon help and count 1/16 SP for each Help Object.

#### Example 6—Login screen help

An application has a typical login screen that requires a username, password, and other fields of data, countable using FPs as five DETs. A Help Item is provided in the user manual, which describes the general purpose of the of the login screen but does not provide any significant detail on what each of the DETs mean or about the data validation logic incorporated in the login process. Count one Help Object for the login screen as a whole, and 1/16 SP.

#### Example 7—Help in the form of a video

An icon is added to a login screen as described in Example 6, in which an instructor explains the general purpose of the screen and how to fill the 5 fields. Count 1 Help Object for the general explanation and 5 Help Objects for the explained fields, which provides 6/16 SPs.

NOTE—There are many cases in which adding a Help Item involves UI effort. In such cases, it is not expected to size this activity twice, as SPs for help sub-category and additional SPs for user interfaces. When the primary intent of the activity is creating Help Item, only the help methods sub-category should be counted.

### 5.2.3 Sub-category 2.3—Multiple input methods

The ability of the application to provide its functionality while accepting multiple input methods

#### 5.2.3.1 SNAP Counting Unit (SCU)

The EP.

#### 5.2.3.2 Terms

##### 5.2.3.2.1 Input method

A technique or media type, which is used to deliver data into the assessed application, such as bar code reader, fax, pdf, office document, screen, voice message, SMS, smart mobile device, etc.

The assessed application may need to identify the input method in order to interpret and use the received information.

##### 5.2.3.3 Complexity parameters

- The number of data element types (DETs) in the SCU
- The number of additional input methods

##### 5.2.3.4 SNAP Points (SPs) calculation

Identify the complexity based on the number of DETs. Calculate SP based on the constant factor and the number of input methods (see [Table 12](#)).

**Table 12—SNAP sizing for multiple input methods**

	Input methods complexity		
	Low	Average	High
	1–4 DETs	5–15 DETs	16+ DETs
SP=	3 × the number of additional input methods	4 × the number of additional input methods	6 × the number of additional input methods

### 5.2.3.5 Rules

This category should be used to when there are multiple types of inputs used to invoke the same functionality. If the different input types differ in terms of DETs, FTRs, and processing logic, then they would already have been accounted as separate functions in FP counting process.

If they are same, then multiple input methods should be used.

Check the following:

- Approach taken for FP counting - single instance or multiple instance.
- The multiple methods of input for the same functionality (same DETs, FTRS and processing logic) have not been included for FP size calculation. In other words, if the FP count has been done using single instance approach for different media types, then the additional input method of same data entry needs to be accounted for using SNAP. For example, the same input can be provided via a smart phone or a web screen.

If multiple input methods are already accounted for in the FP count or the multiple instance approach has been taken for FP counting, then it should be excluded from the SNAP assessment.

#### Example:

See Example in [5.2.4](#).

## 5.2.4 Sub-category 2.4—Multiple output methods

The ability of the application to provide its functionality while using multiple output methods.

### 5.2.4.1 SNAP Counting Unit (SCU)

The EP.

#### 5.2.4.2 Terms

##### 5.2.4.2.1 Output method

A technique or media type, which is used to deliver data from the assessed application, such as Fax, PDF, Office document, screen, voice message, SMS etc.

The assessed application may need to manipulate the sent information in order to send it to the various outputs.



#### 5.2.4.2.2 Complexity parameters

- The number of DETs in the SCU
- The number of additional output methods

#### 5.2.4.3 SNAP Points (SPs) calculation

Identify the complexity based on the number of DETs. Calculate size based on the constant factor and the number of output methods (see [Table 13](#)).

**Table 13—SNAP sizing for multiple output methods**

	Output Methods complexity		
	Low	Average	High
	1–5 DETs	6–19 DETs	20+ DETs
SP =	$3 \times$ the number of additional output methods	$4 \times$ the number of additional output methods	$6 \times$ the number of additional output methods

NOTE—When counting a new development project, the number of output methods should include the additional ones only, assuming one of the output methods is the base method. For example, if the new development uses four output methods, the number of additional output methods is three.

#### 5.2.4.4 Rules

This category should be used to when there are multiple types of outputs used for the same functionality. If the different output types vary in terms of DETs, FTRS, and processing logic, then they would already have been counted as separate functions in function point counting process.

If they are same, then multiple output methods should be used.

Check the following:

- Approach taken for FPA—single instance or multiple instance.
- The multiple methods of output for the same functionality (same DETs, FTRS, and processing logic) have not been included for FP size calculation. In other words, if the FP count has been done using single instance approach for different media types, then the additional output method of same data entry needs to be accounted for using SNAP.
- For example, the same output can be provided to a smart phone or to a web screen.
- If multiple output methods are already accounted for in the FP count or the multiple instance approach has been taken for FP counting, then it should be excluded from the SNAP assessment

#### Example:

A banking software application supports five different processes (in FP terms EPs): Create Account, Modify Account, Make Payment, End-of-Day (EOD) Account Creation summary report, EOD Credit Debit report.

At present, the three EPs of ‘Create Account,’ ‘Modify Account,’ and ‘Make Payment’ take input by keying in data from the keyboard. The bank wants to enhance the software to be able to accept input for these three processes in the form of scanned documents and by reading a barcode as well.

(The “Create Account” and “Modify Account” processes 20 DETs each, and “Make Payment” process processes 15 DETs).

The “EOD Account Creation Summary Report” and “EOD Credit Debit Report” are currently sent out in printed CSV format. The bank wants to enhance the software to be able to produce the output for these processes in the form of printed PDF as well as inline mail to the recipients.

(The EOD Account Creation Summary Report has 15 DETs and EOD Credit Debit Report has 10 DETs).

The design solution for this requirement involves two subcategories: 2.3 (multiple input methods, see [Table 14](#)) and 2.4 (multiple output methods, see [Table 15](#)).

**Table 14—Example: SNAP sizing for multiple input methods**

2.3 Multiple input methods					
No.	SCU Description	Complexity Level	# Additional Input Methods	Formula	SP =
1	Create Account	High	2	$6 \times$ the additional input methods	12
2	Modify Account	High	2	$6 \times$ the additional input methods	12
3	Make Payment	Average	2	$4 \times$ the additional input methods	8

**Table 15—Example: SNAP sizing for multiple output methods**

2.4 Multiple Output Methods					
No.	SCU Description	Complexity Level	# Additional Output Methods	Formula	SP =
4	EOD Account Creation Summary Report	Average	2	$4 \times$ the additional input methods	8
5	EOD Credit Debit Report	Average	2	$4 \times$ the additional input methods	8

Total SNAP size for the project =  $\sum$  SP for Sub-category 2.3 +  $\sum$  SP for Sub-category 2.4 = 12 + 12 + 8 + 8 + 8 = 48

### 5.3 Category 3: Technical environment

The technical environment category relates to aspects of the environment where the application resides. It assesses technology as well as changes to internal data and configuration that do not provide added or changed functionality from a function points perspective

#### 5.3.1 Sub category 3.1: Multiple platforms

Operations that are provided to support the ability of the software to work on more than one platform. In order for software to be considered multi-platform, it should be able to function on more than one computer architecture or operating system. This can be a time-consuming task given that different operating systems have different application programming interfaces or APIs (for example, Linux uses a different API for application software than Windows does).

##### 5.3.1.1 SNAP Counting Unit (SCU)

The EP.

### 5.3.1.2 Terms

Computing platform includes a hardware architecture and a software framework (including application frameworks), where the combination allows software, particularly applications software, to run. Typical platforms include a computer's architecture, operating system, programming languages and related user interface (run-time system libraries or graphical user interface).

Software Platforms: Software Platform is a framework used for the software application development. Different programming languages can be grouped into several platforms based on the programming language family.

A programming language can belong to a particular software language family like the following:

- Object Oriented: Java, C++, C#, Javascript, Python, Smaltalk, VB, VB.NET etc
- Procedural: C, FORTRAN, PHP, COBOL etc.
- Declarative: SQL, XQuery, BPEL, XSLT, XML etc.

Hardware Platforms: A hardware platform can refer to a computer's architecture or processor architecture. For example, the x86 and x86-64 CPUs make up one of the most common computer architectures in use in general-purpose home computers

### 5.3.1.3 Complexity parameters

The complexity parameters for Sub-category 3.1 include the following:

- Nature of platforms (i.e., software, hardware)
- Numbers of platforms to operate

### 5.3.1.4 Calculation method

Identify the different software and hardware platforms involved and the number of platforms to operate. Calculate size based on the platform category row from [Table 16](#) below and the number of platforms. If more than one row is applicable, then the size is the sum of constant factors obtained from each category applicable.

**Table 16—SNAP sizing for multiple platforms**

	SP=		
	2 platforms	3 platforms	4+ platforms
<b>Category 1: Software Platforms: Same Software Family</b>	20	30	40
<b>Category 2: Software Platforms: Different Family</b>	40*	60*	80*
<b>Category 3: Software Platforms: Different Browsers</b>	10	20	30
<b>Category 4: Hardware Platforms: Real time embedded systems</b>	TBD**	TBD**	TBD**
<b>Category 5: Hardware Platforms: Non- Real time embedded systems</b>	TBD**	TBD**	TBD**
<b>Category 6: Combination of Hardware and Software: Non-Real time embedded systems</b>	TBD**	TBD**	TBD**

\*For category 2: Software platforms: Different family, count number of different families (2 families, 3 families, 4+ families).

\*\*TBD: To be defined.

### 5.3.1.5 Platform examples

The following are examples of platforms from [Table 16](#):

- Software Platform: .NET, Java
- Operating System Platform: MS Windows, Linux, IBM/Microsoft Operating System 2, Mac OS
- Hardware Platform: Mainframe computers, Midrange computers, RISC processors, Mobile device architecture, Mac systems.
- Browsers: Internet Explorer, Firefox, Google Chrome, etc.

NOTE 1—Working on one platform is a basic requisite for the system to operate; therefore, a single platform does not generate SPs using this sub-category.

NOTE 2—Building the software on multiple platforms generates SPs for two platforms or more according to [Table 16](#).

NOTE 3—Software platforms can be added or removed, not changed. For example, upgrading from Firefox 3.1.x to 3.2.a is considered as adding a platform, not changing one.

NOTE 4—Upgrading a software platform from one version to another is counted as adding a platform only. The old platform is not counted as deleted nor as changed.

NOTE 5—When adding or removing platforms during an enhancement project, size the multiple platform sub-category for each impacted SCU at the end of the enhancement project and use that as the SNAP size for the enhancement.

NOTE 6—For enhancement projects, count the total number of platforms after the project is complete and use that to calculate SNAP size (do not size the changes, additions or deletions of platforms during the enhancement project).

NOTE 7—When adding a mixture of platforms from different categories, count SPs for each platforms category.

Examples of multiple platforms include the following:

a) Example 1

Two platforms of Family 1 and 2 platforms of Family 2:

Use category 1 (similar platforms) for each software family (20SP for the two platforms of family 1) + (20SP for the two platforms of family 2)

Use category 2 (different platforms—SPs for 2 different platform families (= 40 SP)

b) Example 2

Three platforms of Family 1 and 2 platforms of Family 2:

Use category 1 (similar platforms) for each software family (30SP for the 3 platforms of family 1) + (20SP for the two platforms of family 2)

Use category 2 (different platforms)—SPs for 2 different platform families (= 40 SP)

c) Example 3

Three platforms of Family 1 and 1 platform of Family 2:

Use category 1 (similar platforms) for each applicable software family (30SP for the 3 platforms of family 1) + (0 SP for the 1 platform of family 2)

Use category 2 (different platforms)—SPs for 2 different platform families (= 40 SP)

d) Example 4

Three platforms of Family 1, 1 platform of Family 2 and 2 browsers:

Use category 1 (similar platforms) for each software family (30SP for the 3 platforms of family 1) + (0 SP for the platform of family 2)

Use category 2 (different platforms) - SPs for 2 different platform families (= 40 SP)

Use Category 3 for the browsers support (10 SP)

If an application is built on JAVA and COBOL and requires multiple (more than 4) browser support, then SNAP size would be 40 SP per each SCU that is built on both Java and Cobol, (Java and COBOL are considered as a different family plus 30 SP per each SCU that is to work on multiple browsers

NOTE—Currently the platforms considered in the calibration of the model are only software type platforms.

Please note that this category should be used only if the same set of functionality is being delivered on multiple platforms. This is the case where business functionality is the same but it needs to be delivered in two different environments. For example, same application functions are built on JAVA and also on VC++ to suit client requirements, and then this category can be used.

If the architectural framework itself consists of different platforms to deliver part of the functionality, then this category should not be used. This is a usual case where different technical components interact with each other to deliver application functions. No duplication of effort takes place to rebuild the same functionality in a different environment.

### 5.3.2 Sub category 3.2: Database technology

Features and operations that are added to the database or to the statements to read/write data to and from the database to deliver NFR without affecting the functionality that is provided

#### 5.3.2.1 SNAP Counting Unit (SCU)

The EP.

#### 5.3.2.2 Terms

##### 5.3.2.2.1 Database changes

Each of the following sub items is considered as one change.

- a) Creating or changing a business table or a reference table, such as:
  - 1) Adding tables or adding columns for non-functional purposes only
  - 2) Rearranging the order of column in a table
  - 3) Changing or adding relationships using referential integrity features
  - 4) Changing the primary key without dropping and adding the primary key
- b) Creating or updating code data table
  - 1) Adding tables or adding columns for non-functional purposes only
  - 2) Rearranging the order of column in a table
  - 3) Changing the primary key without dropping and adding the primary key

- c) Adding, deleting or changing an index, such as:
  - 1) Changing the columns used for indexing
  - 2) Changing the uniqueness specification of an index
  - 3) Clustering the table data by a different index
  - 4) Changing the order of an index (ascending or descending)
- d) Adding or changing database views (see definitions) and partitions, such as:
  - 1) Changing or adding database partitioning
  - 2) Adding, changing or removing a database view
- e) Changing database capacity, such as:
  - 1) Tables' space
  - 2) Enhancing the performance features
- f) Changing a query or insert, such as changes to queries or data selection or inserts to the database without adding, deleting or changing functionality. For example, changing the primary key and adding relationship is counted as one change

### 5.3.2.3 Complexity parameters

Complexity parameters for Sub-category 3.2 include the following:

- FTR complexity
- The number of database-related changes

Changes to the database might be done for any non-functional requirement such as performance, capacity management, data integrity etc. The complexity of implementing any such change would depend on the complexity of the Logical File as well as the # of changes (see [Table 17](#)).

**Table 17—FTR complexity, database technology**

		DETs		
		1–19	20–50	> 50
Record element types (RETs)	1	Low	Low	Average
	2–5	Low	Average	High
	>5	Average	High	High

### 5.3.2.4 Calculation method

Calculate size as the product of the constant factor and the number of changes ([Table 18](#))

**Table 18—SNAP sizing for database technology**

	FTR Complexity Factor		
	Low	Low	Low
SP=	6 × the number of changes	9 × the number of changes	12 × the number of changes

If there are multiple FTRs being impacted for the NFR, which are all impacting the same EP, then the higher complexity of the FTR should only be considered as the Complexity Factor, not the individual FTRs separately.

In such a case, perform the following steps:

- Identify the impacted FTRs for the EP
- Determine the complexity of the FTRs impacted
- Choose the highest complexity

NOTE—Use this sub-category for new development/new requirement as well as enhancement. For a new development or new requirement, separate the requirement into its functional aspects and its non-functional aspects.

**Examples:**

An EP “Create Order” is designed for performance improvement. To achieve this, a “read-only” database view is created on “Customer” FTR having 18 DETs and 3 record element types (RETs) (FTR complexity is “Low”).

In addition, an index is created on “Order Placed” FTR having 30 DET and 3 RET (FTR complexity is “Average”).

The highest FTR complexity is “Average”; therefore, for the two changes:

$$SP = 9 \times 2 = 18$$

### **5.3.3 Sub category 3.3: Batch processes**

Batch jobs that are not considered as functional requirements (they do not qualify as a transactional function) can be considered in SNAP. This sub-category allows for the sizing of batch processes, which are triggered within the boundary of the application, not resulting in any data crossing the boundary.

NFR associated with batch jobs such as improving the job completion time, increasing the capacity of the job to process higher volumes of transactions, or performance improvement requirements may be sized using SNAP Sub-categories 3.2, 1.1, or 1.2, as applicable.

However, if an NFR related to batch processing is not covered under these sub-categories, it may be considered in Sub-category 3.3.

#### **5.3.3.1 SNAP Counting Unit (SCU)**

User identified batch job.

NOTE—When several batch jobs are automated (run always as a whole) and only the end result is user identifiable, count these batch jobs as an individual SCU

#### **5.3.3.2 Complexity parameters**

The complexity parameters for Sub-category 3.3 are as follows:

- Number of DETs processed by the job
- Number of FTRs either read or updated by the job

### 5.3.3.3 Calculation method

For each user identified job, identify the complexity level by counting the number of FTRs read or updated. Calculate size as per Table 19.

**Table 19—SNAP sizing for batch processes**

	Complexity level		
	Low (1–3 FTR)	Average (4–9 FTR)	High (10+ FTR)
SP =	4 × the number of DETs	6 × the number of DETs	10 × the number of DETs

NOTE—User specified one-time data loads to logical tables, can be counted using this category. Please note that these data loads should not be migration-related data loads, which are counted as conversion FP using function points.

#### Examples:

- Different processes are merged into one batch: count the DETs and FTRs in the merged batch
- Intermediate data for job validation that are in code data
- Scheduler data instructs to perform subsequent process steps, which are in code data

## 5.4 Category 4: Architecture

The architecture category relates to the design and coding techniques utilized to build and enhance the application. It assesses the complexities of modular and/or component-based development.

### 5.4.1 Sub category 4.1: Component-based software

Pieces of software used within the boundary of the assessed application to integrate with previously existing software or to build components in the system.

#### 5.4.1.1 SNAP Counting Unit (SCU)

The EP.

#### 5.4.1.2 Terms

##### 5.4.1.2.1 A Software component

A piece of software offering a predefined service and which is able to communicate with other components via standard interfaces.

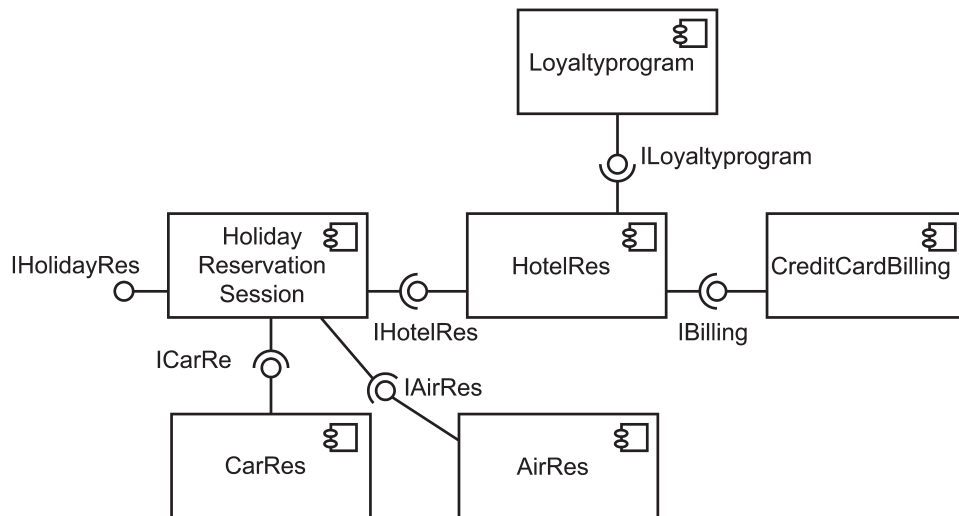
An individual software component is a software package, a Web service, or a module that encapsulates a set of related functions (or data). The essence of a “component” is the encapsulation of business logic or technical functionality that admits a standard interface. The software component is the element that conforms to a component model and can be independently deployed and composed without modification, according to a composition standard. A component model defines specific interaction and composition standards. A component model implementation is the dedicated set of executable software elements required to support the execution of components that conform to the model.



Criteria for software components:

- a) Performs a specific functionality
- b) Capable of parallel execution: multiple-use
- c) Exchangeable: Non-context-specific
- d) Composable with other components (can be selected and assembled in various combinations to satisfy specific user requirements)
- e) Encapsulated i.e., non-investigable through its interfaces
- f) A unit of independent deployment and versioning with well-defined interfaces and communicates via interfaces only
- g) Has a structure and behavior that conforms to a component model like COM, CORBA, SUN Java, etc.

Example: [Figure 4](#) shows simple components interacting with each other (source: Wikipedia).



**Figure 4—Components model for holiday reservation system**

#### 5.4.1.3 Complexity parameters

The complexity parameters for Sub-category 4.1 include the following:

- Third-party component or in-house reuse
- The Number of unique components that are involved in the EP

#### 5.4.1.4 SNAP Points (SPs) calculation

Calculate size based on the constant factor and the number of unique components as per [Table 20](#).

**Table 20—SNAP sizing for component-based software**

Type	SPs calculation
In-house components	$SP = 3 \times \text{the number of unique components}$
Third party components	$SP = 4 \times \text{the number of unique components}$

NOTE 1—This sub-category does not size the functionality of the component. Follow the instructions of ISO/IEC 20926:2009 to count the functional size of the components.

NOTE 2—Reuse of components may be applied to meet NFR such as maintainability (“the capability of the software product to adhere to standards or conventions relating to maintainability”), changeability, maturity or replaceability.

#### 5.4.2 Sub-category 4.2—Multiple input/output interfaces

Applications required supporting multiple inputs and outputs interfaces (user files with the same format) are covered in this subcategory. For example: due to a growing number of users and volume of data over a period of time.

Adding more input/output interfaces without changing the functionality is not considered functional change and hence such changes are not sized by FP. This sub-category should be used to size such changes in an application.

NOTE—If the project/organization considers adding new input/output interfaces as a functional change, then function points would be used for sizing, and SNAP should not be used.

##### 5.4.2.1 SNAP Counting Unit (SCU)

The EP.

##### 5.4.2.2 Complexity parameters

The complexity parameters for Sub-category 4.2 include the following:

- The number of DETs in the SCU
- The number of additional input and output interfaces

NOTE—Count the number of additional input and output interfaces

##### 5.4.2.3 SP calculation

Identify the complexity based on the number of DETs in the SCU.

The size is the product of the factor derived from the number of DETs specified in [Table 21](#) and the number of added interfaces.

**Table 21—SNAP sizing for Multiple Input /Output Interfaces**

	Complexity Level		
	Low	Average	High
	1–5 DETs	6–19 DETs	20+ DETs
SP =	$3 \times \text{the additional number of interfaces}$	$4 \times \text{the additional number of interfaces}$	$6 \times \text{the additional number of interfaces}$

## Examples

- a) Example 1: Adding interfaces to external application A1 without adding or changing the functionality. One SCU is shown in Figure 5, data flows to and from the boundary.

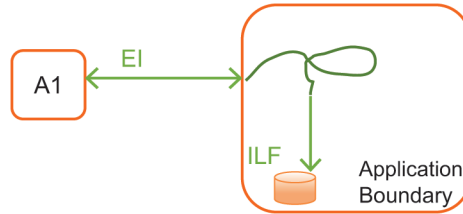


Figure 5—Example 1, before the change

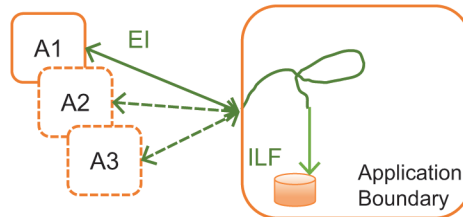


Figure 6—Example 1, after the change

NOTE—Dotted lines in Figure 6 through Figure 9 indicate the change to the existing configuration as follows:

- The number of DETs in the EP = 6 (average complexity)
  - Two additional interfaces
  - $SP = 4 \times 2$  interfaces = 8
- b) Example 2: After the change in Example 1 was delivered, it is required to add another interface to external applications A1, A2, A3.
- One SCU is shown in Figure 7, data flows to and from the boundary.

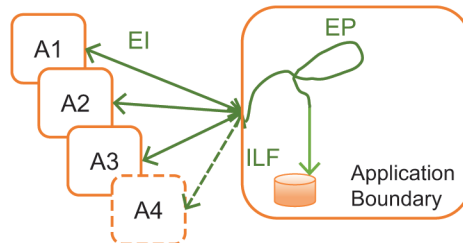


Figure 7—Example 2, after the second change

- The number of DETs in the EP = 6 (average complexity)
  - One additional interface
  - $SP = 4 \times 1$  interfaces = 4
- c) Example 3: Adding interfaces to external applications A1 and B1 without adding or changing the functionality

Two SCUs are shown in Figure 8, data flows to and from the boundary.

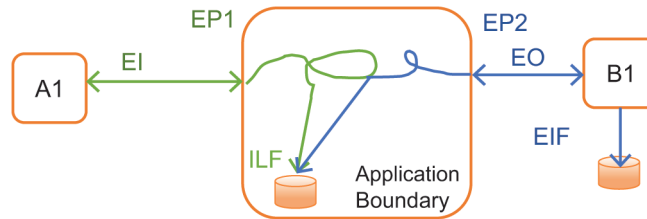


Figure 8—Example 3, before the change

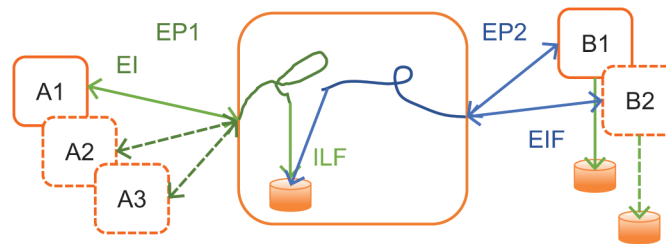


Figure 9—Example 3, after the change

The number of DETs in EP1 flowing to and from application A1 = 5 and the number of DETs in the EP2 flowing to and from application B1 = 8.

SCU 1 = EP1

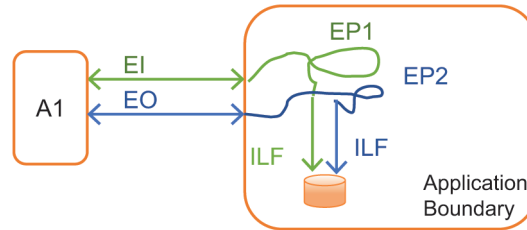
- 5 DETs = Low
- 2 Additional interfaces
- $SP = 3 \times 2$  interface = 6

SCU 2 = EP2

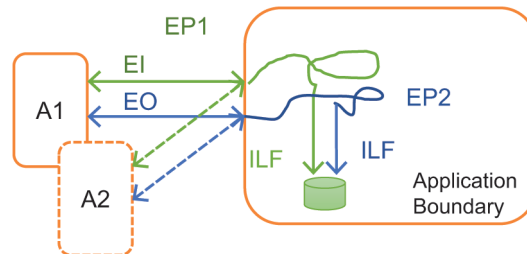
- 8 DETs = Average
- 1 Additional interface
- $SP = 4 \times 1$  interface = 4

$SP = 6 + 4 = 10$

- d) Example 4: Adding interfaces to external application A1 without adding or changing the functionality, two SCUs (see Figure 10 and Figure 11)



**Figure 10—Example 4, before the change**



**Figure 11—Example 4, after the change**

The number of DETs in the EI = 5 and in the EO = 10

SCU 1 = EP1

— 5 DETs = Low

— 1 Additional interface

—  $SP = 3 \times 1 \text{ interface} = 3$

SCU 2 = EP2

— 10 DETs = Average

— 1 Additional interface

—  $SP = 4 \times 1 = 4$

Total  $SP = 3 + 4 = 7$

NOTE—A key difference between this sub-category and 2.3 / 2.4 multiple input /output methods is that in 4.2, the existing interface is replicated with the same technology to give all users the same level of performance and same experience.

## 5.5 Sizing code data

Code data is a type of data entities, used for software sizing (in addition to business data and reference data).

According to ISO/IEC 20926:2009, “code data” usually exists to satisfy NFR from the user (for quality requirements, physical implementation and/or a technical reason).

The user does not always directly specify code data (sometimes referred to as “list data” or “translation data”). In other cases, it is identified by the developer in response to one or more NFR.

Code data provides a list of valid values that a descriptive attribute may have. Typically, the attributes of the code data are code, description and/or other ‘standard’ attributes describing the code (e.g., standard abbreviation, effective date, termination date, audit trail data, etc.). The different categories of data are outlined below to assist in identification.

When codes are used in the business data, it is necessary to have a means of translating to convert the code into something more recognizable to the user. In order to satisfy NFR, developers often create one or more tables containing the code data. Logically, the code and its related description have the same meaning. Without a description, the code may not always be clearly understood.

The key differences between code data and reference data are:

- With code data, one can substitute one for the other without changing the meaning of the business data; e.g., airport-code versus airport-name, color-ID versus color-description.
- With reference data, one cannot substitute (e.g., tax code with the tax-rate).

### **5.5.1 Code data characteristics**

Code data has most of the characteristics outlined in the following subclauses.

#### **5.5.1.1 Logical**

Logical characteristics include:

- Data is mandatory to the functional area but optionally stored as a data file
- Not usually identified as part of the FUR; it is usually identified as part of the design to meet NFR
- Sometimes user maintainable (usually by a user support person)
- Stores data to standardize and facilitate business activities and business transactions
- Essentially static—only changes in response to changes in the way that the business operates
- Business transactions access code data to improve ease of data entry, improve data consistency, check data integrity, etc.
- If recognized by the user
  - is sometimes considered as a group of the same type of data
  - could be maintained using the same processing logic

#### **5.5.1.2 Physical**

Physical characteristics include:

- Consists of key field and usually one or two attributes only
- Typically has a stable number of records
- Can represent 50% of all entities in third normal form
- Sometimes de-normalized and placed in one physical table with other code data
- May be implemented in different ways (e.g., via a separate application, data dictionary, or hard-coded within the software)

### **Examples**

Examples of code data include:

- a) State
  - 1) State code
  - 2) State name
- b) Payment type
  - 1) Payment type code
  - 2) Payment description

### 5.5.2 Handling code data from non-functional sizing perspective

For the purpose of FPA, code data cannot be counted as logical files. They are not considered as Internal Logical File (ILF) or External Interface File (EIF), and cannot be considered Record Element Types (RETs) or DETs on an ILF or EIF. Code data cannot be considered a File Types Referenced (FTR) while assessing the complexity of a transactional function (EI, EO, and External Inquiry—EQ).

For the purpose of SNAP, code data which is maintained within the application boundary by use of screens or by formal enhancement requests by the customer is counted as follows:

- Irrespective of the number of code data physical tables, the code data would be grouped as 1 data group (1 FTR) under SNAP
- Code data is classified as in [Table 22](#),

**Table 22—Types of code data**

Substitution	Static or constant	Valid values
Code + Description	One occurrence Static Data Default value	Valid Values Range of valid values

For SNAP analysis of the complexity of code data group, the number of RETs of the code table would depend upon the type of occurrences of code data types.

#### Example

In a banking application, the following code tables were created:

- [Table 1](#): Having state name and state code
- [Table 2](#): Having branch code, branch name, branch city
- [Table 3](#): Having one single data entry of the bank name and logo, which is used for printing letterheads

Three types of occurrences exist for code data:

- Substitution—[Table 1](#) having state code and state name, [Table 2](#) having branch code and branch name
- Valid values—[Table 2](#) having a range of bank branch cities
- Static data—[Table 3](#) having one single data entry of the bank name and logo, which is used for printing letterheads

Hence the number of RETs for the code data group is 3.

NOTE—If the data is not of the above code data sub types, then it may be the data in system tables and not the application tables required for supporting business. This is not sized under code data.

### 5.5.3 How code data is sized using SNAP

Count the creation/maintenance and the utilization of code data.

The creation of code data is always counted as “3.2: Database technology.”

The maintenance of code data is checked under the following subcategories depending on the case applicable:

- If the code data is maintained in hard coded tables that are not viewable via screens, but can be updated by the administrator using script/code change in response to a formal user request, then it is counted using Sub-category 3.2: Database technology. For example, a code table has the bank name and logo stored as static data, which is referred by different processes. When a change request is raised to modify the logo, then it is sized using this category.
- When the code data is used for reasons such as entry validations, data formatting, batch process management, any changes to code data values (add /change/ delete) should be counted using the proper sub-category.

The utilization of code data is counted in the following sub-categories, according to the purpose of the data: “1.1 Data Entry Validation”; “1.2 Logical and Mathematical Operations”; “1.3 Data Formatting”; “1.5 Delivering Added Value to Users by Data Configuration”; and “3.3 Batch Processes.”

When NFR use code data and transactions cross partitions, Sub-category “1.4 Internal Data Movements” should be used.

Examples of SNAP sizing of code data are shown in [Table 23](#).

**Table 23—Example of SNAP sizing of code data**

Examples:	Sub-categories for utilizing code data
Create a code data table for address validation	1.1 Data Entry Validation for enabling the validation 3.2 Database Technology for code table creation
Same as above. The screens with the address are on the Front-End application, the data is in the Back-End application	1.1 Data Entry Validation for enabling the validation 3.2 Database Technology for code table creation 1.4 Internal Data Movements
Using multi-language screens, the translation is in new code data tables	1.3 Data Formatting 3.2 Database Technology for code table creation
Add scheduling data to perform batch files	3.3 Batch Processes 3.2 Database Technology for code table creation

## 6. The sizing process

The non-functional sizing process uses the collection of information, which is needed for defining what is measured and then measuring its non-functional size.

The sizing process contains the following steps:

- a) Gather available documentation



- b) Determine the purpose, scope, boundary, and partition of the measurement
- c) Identify the NFR
- d) Associate NFR with sub-categories and identify the SCU
- e) Determine the SNAP size for each sub-category
- f) Calculate non-functional size
- g) Document and report

## 6.1 The timing of the non-functional sizing

Non-functional sizing can be assessed at any time in the development lifecycle to aid in project estimating, monitoring project change of scope, and evaluating delivered NFR.

Prior to beginning a non-functional sizing, users should determine whether they approximate or measure the size, and document any assumptions.

Approximating permits assumptions to be made about unknown non-functional aspects, to determine an approximate non-functional size.

Measuring includes the identification of all applicable non-functional sub-categories and their complexity.

At an early stage, NFR may not be fully defined. Despite the disadvantages, this sizing can be useful to produce an early estimate. Uses of the non-functional sizing for approximating or measuring non-functional size at the various life cycle phases are presented in [Table 24](#).

**Table 24—The timing of approximating or measuring the non-functional size**

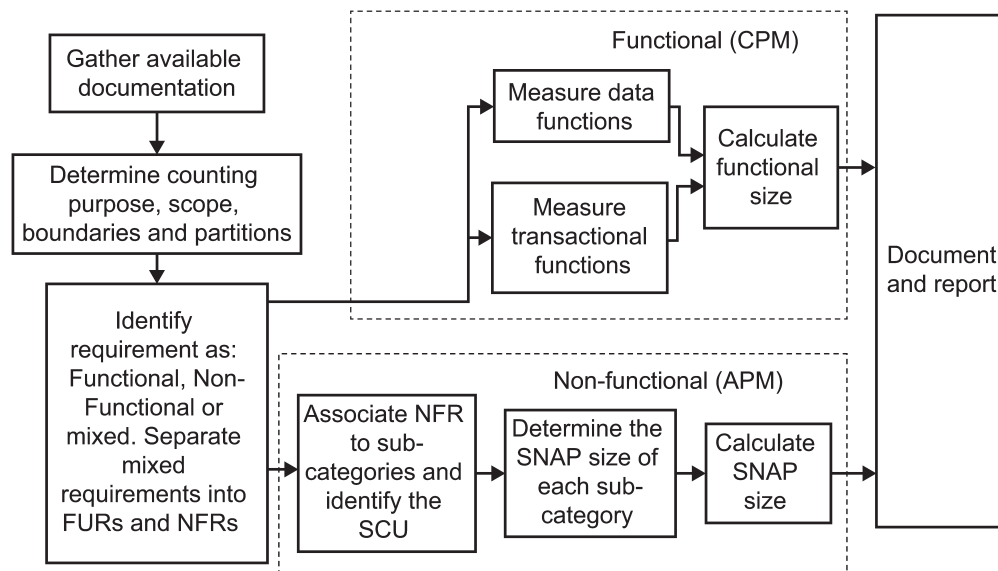
Life cycle phase	SPs can be approximated	SPs can be measured
Proposal: users express needs and intentions	Yes	No
Requirements: developers and users review and agree upon expression of user needs and intentions	Yes	No
Design: developers may include elements for implementation	Yes	Yes
Construction	Yes	Yes
Delivery	Yes	Yes
Maintenance (adaptive, perfective, or preventive)	Yes	Yes

NOTE—No specific development life cycle is implied. The lifecycle phases above may be used as steps of any software development lifecycle

## 6.2 Non-functional sizing and FSM

Non-functional size can be used in conjunction with functional size to provide an overall view of the project or application including both functional and non-functional sizing.

[Figure 12](#) describes the joint process.



**Figure 12—The sizing process**

The steps listed in 6.3 describe the non-functional sizing process.

## 6.3 Steps to determine the non-functional size

### 6.3.1 Step 1: Gather available documentation

Documentation to support the measurement process describe the NFR delivered by the software or the non-functional aspects that are impacted by the software project that is being measured. Non-functional characteristics may also be embedded in the functional documentation.

Sufficient documentation, or access to subject matter experts, who are able to provide additional information to address gaps in the documentation, should be obtained. Standards and International guidelines (such as WCAG 2.1 accessibility guidelines [B15]) should be considered if applicable.

NOTE 1—NFR may be implicit or explicit. When not explicitly specified, verify that expectations are understood and documented.

NOTE 2—NFR are subjected to configuration management.

### 6.3.2 Step 2: Determine the sizing purpose, scope, boundary, and partition

Subclauses 6.3.2.1 through 6.3.2.4 contain required activities to determine the non-functional size.

#### 6.3.2.1 Identify the purpose

Sizing the non-functional size may be conducted to:

- Provide an accurate estimation of effort and time to deliver a software project
- Assess the value of the installed base of applications to determine the support costs
- Assess the developed size and determine the cost of the project based on size

Examples of purposes are to provide the non-functional size.

### 6.3.2.2 Identify the type

The functional size and the non-functional size can be measured for either projects or applications. The type of assessment is determined, based on the purpose, as one of the following:

- Development project assessment
- Enhancement project assessment
- Application assessment

Figure 13 illustrates the types of assessments and their relationships. (Project A is completed first, followed by Project B.)

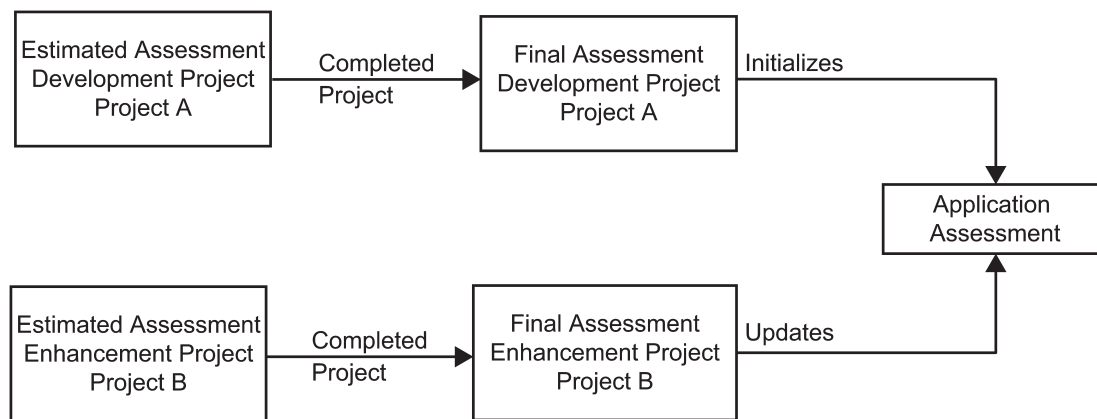


Figure 13—Types of assessments

### 6.3.2.3 Identify the scope

The scope defines the set of NFR to be included in the assessment.

- a) The scope:
  - 1) Is determined by the purpose of performing the non-functional assessment.
  - 2) Defines a set of partition(s).
  - 3) Identifies which non-functional categories and sub-categories should be included in the non-functional size measurement to measure the size of NFR for the development and delivery of the software product.
  - 4) Could include more than one application.
- b) Types of sizing:
  - 1) Sizing a development project includes all NFR for the development and delivery of the software product.
  - 2) Sizing of an installed base of applications includes all NFR for the support of the installed applications.

- 3) Sizing an enhancement project includes all NFR for the development and delivery of the enhancement project.
- 4) Sizing a maintenance project includes all NFR for a selected scope.
- c) The following hints can help in identifying the assessment scope:
  - 1) Review the purpose of the non-functional assessment to help determine the assessment scope.
  - 2) When identifying the scope for measuring the non-functional size of the installed base of applications, include all of the non-functional categories supported by the maintenance team, eventually distinguished by partition within each application's boundary.

#### **6.3.2.4 Determine the boundary**

Determine the boundary of each application within the sizing scope, based on the user view.

##### **6.3.2.4.1 User view**

In order to establish the boundary, the user view should be defined. The following hints can help to identify the user view:

- a) A user is any person or thing (application, device, etc.) that communicates or interacts with the software at any time.
- b) A user view consists of the functional and NFR as perceived by the user.
- c) A user view represents a formal description of the user's needs in the user's language.
- d) A user view can be verbal statements made by the user as to what their view is.
- e) A user view should be approved by the user.
- f) A user view can vary in physical form (e.g. user stories, catalog of transactions, proposals, requirements document, external specifications, detailed specifications, user handbook, quality or non-functional specifications).
- g) A partition may act as an "internal user" for another partition within the same application boundary, in terms of data exchange or data sharing; consequently, different non-functional sizing might be created for each partition.

##### **6.3.2.4.2 Boundary**

The boundary is a conceptual interface between the software under study and its users.

The boundary (also referred to as application boundary):

- Defines what is external to the application
- Indicates the border between the software being measured and the user
- Acts as a "membrane" through which data processed by transactions pass into and out of the application
- Is dependent on the user's external business view of the application; it is independent of non-functional and/or implementation considerations

The positioning of the boundary between the software under investigation and other software applications may be subjective. It is often difficult to delineate where one application stops and another begins. Try to place the boundary from a business perspective rather than based on technical or physical considerations.

For example, Figure 14 shows boundaries between the Human Resources application and the external applications, Currency and Fixed Assets. The example also shows the boundary between the human user (User 1) and the Human Resources application. The Human Resource application may in turn internally satisfy the functional, technical and quality requirements specified by the user.

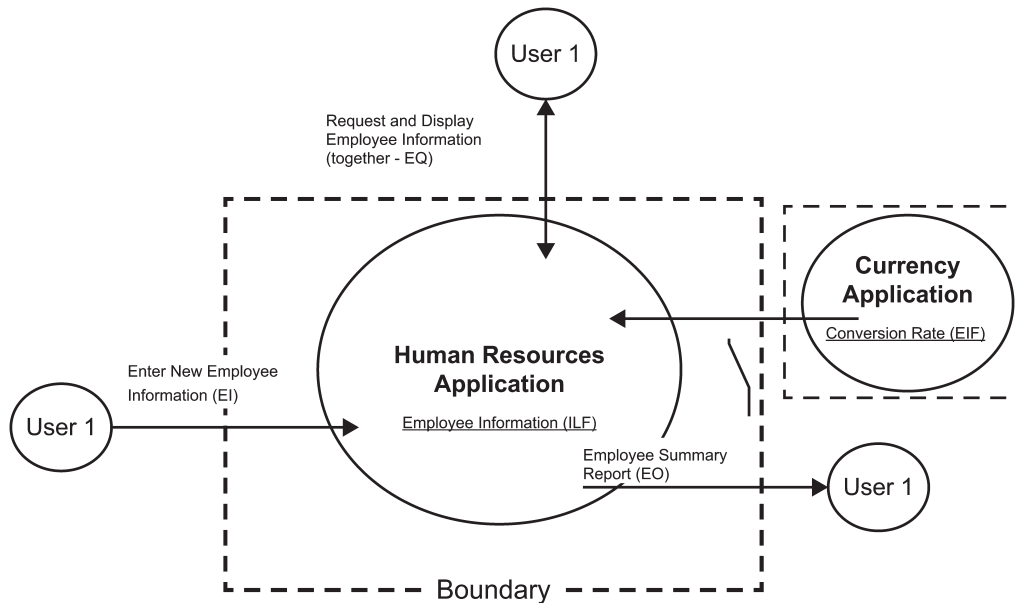


Figure 14—Boundary example

The following rules shall apply for boundaries:

- The logical application boundaries need to be consistent between the FPA and SNAP processes.
- The boundary is determined based on the user view; the focus is on what the user can understand and describe.
- The initial boundary already established for the application, or applications being modified, is not influenced by the assessment scope.
- There may be more than one application included in the sizing scope. If so, multiple application boundaries would be identified.
- When the boundary is not well-defined (such as early in the analysis), it should be located as accurately as possible.

#### 6.3.2.4.3 Multiple-instance approach versus single-instance approach

Different organizations may take different approaches for sizing similar functionality being delivered on different media. They use the single instance or the multiple instance approaches to specify the same. Single instance approach is said to be when the same functionality is delivered via different mediums (input or output), but is counted only once. Organizations using the single instance approach for the FP size can size the other methods of delivery using SNAP. The single instance approach does not recognize the medium for delivery for a transaction function as a differentiating characteristic in the identification of unique transaction functions. If two functions deliver the same functionality using different media, they are considered to be the same function for functional sizing purposes. The multiple instance approach specifies that instance functional size is taken in the context of the approach objective of the count, allowing a business function to be recognized

in the context of the medium in which it is required to operate. The multiple instance approach recognizes the medium for delivery for a transaction function as a differentiating characteristic in the identification of unique transaction functions.

#### **6.3.2.5 Identify the partitions, if applicable**

When identified, partitions may add non-functional size. Sub-category 1.4 (Internal Data Movements) is used to provide the additional non-functional size for the application being assessed.

The following hints can help to identify the boundary and the partition of the application(s):

- a) Use the system external specifications or a system flow chart and draw a boundary around it to highlight which parts are internal and which are external to the application.
- b) Look at how groups of data and software partitions are being maintained.
- c) Identify functional areas by assigning ownership of certain types of analysis objects (such as entities or EPs) to a functional area; non-functional categories are determined by the identification of the functional boundaries (application boundaries as determined by FPA) and eventually partitions within them.
- d) Look at the associated measurement data, such as effort, cost, and defects; the boundaries for measurement should be the same, or eventually, those measurement data might be distinguished by partitions within a single boundary.
- e) Interview subject matter experts for assistance in identifying the boundary.
- f) Interview software analysts for assistance in identifying the partitions, if any.

The following hints can help to identify the boundary and the partition of the application(s):

- g) Use the system external specifications or a system flow chart and draw a boundary around it to highlight which parts are internal and which are external to the application.
- h) Look at how groups of data and software partitions are being maintained.
- i) Identify functional areas by assigning ownership of certain types of analysis objects (such as entities or EPs) to a functional area; non-functional categories are determined by the identification of the functional boundaries (Application boundaries as determined by FPA) and eventually partitions within them.
- j) Look at the associated measurement data, such as effort, cost, and defects; the boundaries for measurement should be the same, or eventually, those measurement data might be distinguished by partitions within a single boundary.
- k) Interview subject matter experts for assistance in identifying the boundary.
- l) Interview software analysts for assistance in identifying the partitions, if any.

#### **6.3.3 Step 3: Identify the NFR**

The following guidelines may be used to identify all NFR:

- a) NFR may be explicitly documented, but may also be implicit. Users should make sure that they identify all requirements, implicit or explicit.
- b) NFR may be mandated by law or by customer's policy, and may not be documented as part of the project's documentation.

- c) It is important to separate NFR and functional requirements. In actual project documentation, a requirement may contain both functional and non-functional aspects.
- d) Such a requirement should be broken down into its functional components and non-functional components, and the segregation should be agreed by both the user/customer and development teams. Rules for segregating the functional aspect of the requirements from its NFR are provided in [7.1](#).

#### **6.3.4 Step 4: Associate NFR with sub-categories and identify the SCU**

Identify the subcategories as follows:

- a) Identify the applicable sub-categories for the NFR. Use the description of the sub-categories and the examples in this standard to associate NFR with sub-categories.
- b) Identify the Counting Units (SCUs).

The SCUs are unique to each sub-category; they are determined by the nature of the sub-category. The SCU is part of the sub-category definition.

A non-functional requirement may consume several SCUs. For example, a requirement for improved time behavior (performance) may involve several EPs, each EP is an SCU.

Sizing is done separately per each SCU.

#### **6.3.5 Step 5: Determine the SNAP size for each sub-category**

For each SCU:

- a) Identify the complexity parameters of the SCU.
- b) Use [Clause 5](#) of this standard to determine the non-functional size of the SCU.

The non-functional size for the sub-category is the sum of the SPs of all SCUs.

#### **6.3.6 Step 6: Calculate the non-functional size**

The non-functional size is the sum of the sizes of all SCUs identified in all sub-categories

#### **6.3.7 Step 7: Document and report**

The sizing results shall be documented as follows:

- a) The purpose and type of count
- b) The counting scope and the boundary of the application
- c) The date of the count
- d) A list of all SCUs, including their type and complexity and number of SPs assigned
- e) The result of the count
- f) Any assumptions made and issues resolved

The documentation can also include the following:

- g) Identification of the source documentation on which the count was based
- h) Identification of the participants, their roles and qualifications

NOTE 1—Negotiate the level of documentation with the client, and inform the client about the related costs and benefits.

NOTE 2—A fully documented count should facilitate traceability, usability and maintainability; however, a client may be interested only in the results.

NOTE 3—The document should have versioning control.

NOTE 4—The document should state the conventions to be adopted when reporting size such that it is qualified with:

- The units of the size
- The name of the sizing method

Example: Non-functional size = 300 SPs (APM V2.4).

## 6.4 Calculating the Non-functional size

### 6.4.1 Formula approach

During an assessment of a non-functional requirement, one or many of the sub-categories can be assessed depending on the specification of the requirement. For each non-functional requirement, it is possible to determine the non-functional size in four steps:

- **Step 1:** For each requirement, identify the categories and sub-categories that are associated with the requirement.
- **Step 2:** For each of the sub-categories, identify the SCUs.
- **Step 3:** Determine the non-functional size (SPs) for each SCU within the sub-category, by using the equation or the table for the sub-categories.
- **Step 4:** Determine the SPs for a specific project or application by using the formula for the project type in question.

### 6.4.2 Determine the non-functional size of each sub-category

The non-functional size of each sub-category shall be determined using the defined measure for the SCU for each sub-category.

There is one definition of the SCU for each of the sub-categories. These assessment criteria are defined in the sub-category definition.

The size of each sub-category is determined by using the defined equation or table for each sub-category.

### 6.4.3 Determine the non-functional size of a development project

The size of the NFR is equal to the sum of SP sizes of each category. A development project non-functional size shall be calculated using the development formula



The formula for development project:

$$\text{DSP} = \text{ADD}$$

where

DSP	is the development project SNAP size
ADD	is the size of the NFR delivered to the user by the development project
ADD	is the sum of SP for all sub-categories

The application non-functional size is equivalent to the non-functional size of the development project.

NOTE—For non-functional size, converted functionality is not identified

#### 6.4.4 Determine the non-functional size of an enhancement project

Enhancement projects can involve additions, changes to, and deletion of existing non-functional features.

Enhancement project is a project to develop and deliver maintenance. It may be adaptive, preventive or perfective type

The enhancement project non-functional size is a measure of the non-functional characteristics added or deleted at the completion of an enhancement project as well as changes made to existing non-functional characteristics as measured by the enhancement project SNAP size.

##### Rules:

Enhancement NFR shall be measured in accordance with the following:

- a) Do not modify the boundary or partition already established for the application(s) being modified.
- b) Assess requirements that are added, changed or deleted.
- c) The application non-functional size may be updated to reflect:
  - 1) Added NFR, which increase the application non-functional size.
  - 2) Changed NFR, which may increase, decrease or have no effect on the application non-functional size.
  - 3) Deleted NFR, which decrease the application non-functional size.

NOTE—This rule is analyzed and counted per each sub-category. An enhancement project non-functional size should be calculated using the formula in [6.4.4.1](#).

##### 6.4.4.1 The formula for enhancement project

$$\text{ESP} = \text{ADD} + \text{CHG} + \text{DEL}$$

where

ESP	is the enhancement project SNAP size
ADD	is the size of the non-functional characteristics being added by the enhancement project
CHG	is the size of the changes made to existing sub-categories by the enhancement project
DEL	is the size of the non-functional characteristics deleted by the enhancement project

For a sub-category SC

$$ESPSC = ADDSC + CHGSC + DELSC$$

For the enhancement project

$$ASP = \sum \text{of ESP for Sub-category 1.1} + \sum \text{of ESP for Sub-category 1.2} + \dots + \sum \text{of ESP for Sub-category 4.2}$$

The formula for the application size after an enhancement project

An application non-functional size after an enhancement project shall be calculated using the formula:

$$ASPA = ASPB + (ADD + CHGA) - (CHGB + DEL)$$

where:

ASPA	is the application SP after the enhancement project
ASPB	is the application SP before the enhancement project
ADD	is the size of the NFR being added by the enhancement project
CHGA	is the size of the NFR being changed by the enhancement project as they are/will be after implementation
CHGB	is the size of the NFR being changed by the enhancement project as they are/were before the project commenced
DEL	is the size of the NFR deleted by the enhancement project

NOTE—The size of changes made to sub-categories in an enhancement project (i.e., CHG) does not provide an assessment of the overall size of the sub-categories before (CHGB) or after (CHGA) the enhancement project—CHG measures the size of the change in the non-functional characteristics being changed.

A demonstration of a situation in which the size of the change in the project (CHG) is 12 SP, but the overall size of the sub-category is 0 (CHGA = CHGB) is shown in Item b) in 6.4.5.

Therefore, in order to maintain the application's non-functional size after each enhancement project, one should assess, in addition to the size of the changes made, the size of the sub-category before and after the enhancement.

#### 6.4.5 Calculation examples

- a) An application has an EP in which there are 10 DETs. Three DETs are already encrypted (encryption complexity is average).

The enhancement project requires to change the encryption type (new encryption is local).

- 1) Counting the enhancement project:
  - i) Type of requirements: CHG
  - ii) Transformation complexity: High

$$ESP = ADD + CHG + DEL$$

$$= 0 + 5 \times \text{the number of DETs} + 0 = 15$$

- 2) Counting the application:

$$ASPB = 3 \times \text{the number of DETs} = 9 \text{ (Assuming this is the only NFT in the application)}$$

$$\text{CHGB} = 3 \times \text{the number of DETs} = 9$$

$$\text{CHGA} = 15$$

$$\begin{aligned}\text{ASPA} &= \text{ASPB} + (\text{ADD} + \text{CHGA}) - (\text{CHGB} + \text{DEL}) \\ &= 9 + (0 + 15) - (9 + 0) = 15\end{aligned}$$

- b) A “Search” application has an entry screen with one entry: a string to search. To keep the look-and-feel of the application fresh, each month the background color is changed, the size and location of the search field are changed and the shape and location of the search button are changed.

To build this screen, the sum of the number of unique properties is 20.

- 1) Counting the enhancement project:

Type of requirements: CHG

UI Type complexity: High

Number of unique UI elements: 3 (Screen, field, control);

$$\begin{aligned}\text{ESP} &= \text{ADD} + \text{CHG} + \text{DEL} \\ &= 0 + 4 \times 3 + 0 = 12\end{aligned}$$

- 2) Counting the application:

$$\text{ASPB} = 3 \times 4 = 12 \text{ (Assuming this is the only NFT in the application)}$$

$$\text{CHGB} = 3 \times 4 = 12$$

$$\text{CHGA} = 3 \times 4 = 12$$

$$\begin{aligned}\text{ASPA} &= \text{ASPB} + (\text{ADD} + \text{CHGA}) - (\text{CHGB} + \text{DEL}) \\ &= 12 + (0 + 12) - (12 + 0) = 12\end{aligned}$$

(No change in application size)

## 7. Complementarity of the functional and the non-functional sizes

Documented requirements, in actual projects, may combine functional and non-functional aspects. Such a requirement should be broken down into its functional components and non-functional components, and the segregation should be agreed by both the user/customer and development teams. Use FP for the functional parts of the requirements and SPs for the non-functional parts of the requirements.

[Table 25](#) is a guideline. To define NFR, ISO standard or a similar standard may be used.

**Table 25—IFPUG FPA and SNAP interrelations**

Case #	Circumstance	Description	Guideline
1	Requirements are functional only	The users do not have any explicit or implicit NFR	Count function points only
2	Requirements are clearly marked as NFR	Parties agree on clear segregation between functional requirements and NFR. Requirements classified as NFR cannot be sized with function points	Count SPs only
3	Requirements involve both functional and non-functional aspects	Functional requirements have additional NFR which can be clearly identified:	See 7.1
4	Requirements are functional only, transactions cross partitions	Functional requirements may involve single or multiple flows. In case of multiple flows, and using ISO/IEC 20926:2009 (IFPUG FSM) guidelines, each flow might not qualify as a separate EP.	Count function points to size the new/enhanced functionality for the main EP as per ISO/IEC 20926:2009, add SNAP size for the transactions/flows within the application's boundary, that cross the partitions
5	Requirements are functional, but they are provided without any software change	Functionality (or any business value) that is added or modified by changing reference data or other means that cannot be sized by FPs, according to ISO/IEC 20926:2009 guidelines or FP counting practices of the organization.	Count SPs using Sub-category 1.5—Delivering Functionality by Data Configuration

The following subclauses contain guidelines that should be used to determine how FP and SPs should be counted.

## 7.1 Requirements involving functional and non-functional requirements

### 7.1.1 Sub-category 1.1 data entry validation

#### 7.1.1.1 What to check

The EP.

#### 7.1.1.2 Rules

- a) Count SPs when:
  - 1) Adding a new DET with validation (count SPs in addition to FPs).
  - 2) Changing a DET (e.g., Masking, length, format) and changing the entry validation.
  - 3) Changing the validation logic of an existing DET.

NOTE 1—For an EP with a DET added to current functionality, count FPs per ISO/IEC 20926:2009 (IFPUG FSM) for the functional aspect of adding the DET and SPs for the complexity change of the validation.

NOTE 2—When adding or changing a validation on an existing DET, count FPs per ISO/IEC 20926:2009 for the functional aspect of changing the validation logic and SPs for the impacts to change in validation complexity.

NOTE 3—Data entry validation is counted in this subcategory only if it involves adding, changing or removing validation logic related to what data is being entered.

For example, adding a rule to validate that a numeric entry is not negative, in addition to validating that the entry is numeric and is decimal—is SP-counted. Adding a valid value to an existing list without changing the

validation logic does not generate SPs in this sub-category (adding a valid value should be counted under Sub-category 3.2 Database Technology, as a database change). Adding a new list of valid values for an existing DET to validate values against is counted in SPs both in Sub-category 1.1 and 3.2.

### 7.1.1.3 Examples

- a) A change in a date field in an enhancement project: former validation for date entry checked the format of DD/MMM/YY and now format should be DD/MMM/YYYY. For the EP that enters the date, this change is counted in SPs using Sub-category 1.1 in addition to FPs counting. (Count FPs if there is a change in the processing logic, as per ISO/IEC 20926:2009.)
- b) In an enhancement project of a travel application for international flight booking function, passport type field is being added. A new code table is added to store passport type (regular, diplomatic, etc.) Based on passport type, validation rules exist to check the format of passport number being entered is correct or not. Two EPs are impacted: “Enter traveler’s details” and “modify traveler’s details”. For both EPs, count FPs for adding the new DET, SPs (Sub-category 1.1) for the data entry validation required for passport type field, and SPs (Sub-category 3.2) for adding the code table.
- c) In a development project of a telecommunication self-service application, payment screen (one EP, “enter payment details”): validate that the mobile number is numerical, and is structured as “0XX-XXXXXXX” (the first digit must be zero, following 9 digits only) is SPs-counted using this sub-category in addition to FPs counting.
- d) In both development and enhancement projects of a health insurance application, entering a new claim process has many fields. Validating that all fields are populated with the right format and that all mandatory fields are not empty is SP counted using this sub-category in addition to FPs counting.
- e) A change is requested to the values in a drop-down list with five values, change one value and add two new values: if data entry validation logic does not change, use Sub-category 3.2 and not this sub-category to count SPs.
- f) An existing EI was enhanced by adding a validation to an existing field to confirm that the value is between 8 and 12 digits long. This new validation was the only change made to the EI. Size this change using FPs since there is a change in the processing logic as per ISO/IEC 20926:2009 (IFPUG FSM), and size it using SPs since it involves a non-functional aspect as well.

## 7.1.2 Sub-category 1.2 logical and mathematical operations

### 7.1.2.1 What to check

The EP

### 7.1.2.2 Rules

- a) Count the functional requirement using FPs for any EO/EI/EQ per ISO/IEC 20926:2009 (IFPUG FSM); in addition, count SPs when processing logic includes extensive mathematical and/or extensive logical operations as defined in 5.1.2. The complexity of the algorithmic calculations and/or the logical processing required is the non-functional aspect of the requirement.
- b) The above is true either for a development of a new request or for an enhancement.

### 7.1.2.3 Example

A project management software application currently allows the user to compute project completion time using the algorithm “Critical Path Method.” The user now wants an enhancement to allow the new, optional use of

the “Program Evaluation Review Technique” (PERT) algorithm to compute project completion times. New DETs are added to capture the attributes needed for calculating project completion time and the probability to meet the completion time.

In this case, count both FPs and SPs for each EP that is affected by the enhancement. Function points are impacted for processing logic change as there is a change in the calculation as per definition in ISO/IEC 20926:2009. SPs need be counted to assess the complexity change in the algorithmic processing being done.

### 7.1.3 Sub-category 1.3 data formatting

#### 7.1.3.1 What to check

The EP

#### 7.1.3.2 Rules

- a) When formatting is requested, break the requirements into its functional part (The “what”) and its non-functional part (the formatting, such as encryption /decryption, byte padding, adding header/footer information on a transaction).
- b) Enhancements may have functional aspect only (no requirements for any format) or both functional and non-functional aspects; enhancement may have functional aspect only (such as a change in the data transferred) or non-functional aspect only or both.

#### 7.1.3.3 Examples

- a) Example 1: Billing report: To improve the look and feel of the report, the user requests an enhancement: all the fields should apply bytes padding, to help ensure:
  - 1) a 20 character display field length for name field
  - 2) a 15 character display field length for the bill values (amounts) and the PID<sup>15</sup>. The patient name should be displayed as First Name.Middle Name.Last name instead of FirstName\_LastName.

In this case, the requirement should be broken into two parts:

- 3) Functional: the patient name should be displayed as First Name.Middle Name.Last name.
- 4) Non-functional: byte padding is added to help ensure a 20-character display field length for name field and 15-character display field length for amount and PID.

Count FPs to size the transaction for part a): Display Report; count SPs to size part b), using Sub-category 1.3.

- b) Example 2: A new HR system is being built to replace two outdated applications. To protect sensitive information, the user requests that the employee’s social security number be masked on all display screens. When updating employee information, the user is provided with an option to unmask the social security number.

In this case, the requirement should be broken into two parts:

- 1) Functional: display screens and Employee Update screen should be counted under FPs
- 2) Non-functional: The masking<sup>16</sup> and unmasking of the social security number are considered non-functional but should only be counted one time under SNAP.

<sup>15</sup>PID – Patient Identifier

<sup>16</sup>Data masking or data obfuscation is the process of hiding original data with random characters or data. The main reason for applying

- c) Example 3: The user has requested changes to the existing display screens that contain employee data and the “Employee Update” screen. In order to protect sensitive information, the user requests that the employee’s social security number be masked on all display screens. When updating employee information, the user is provided with an option to unmask the social security number.

In this case, there is only a non-functional requirement for the masking and unmasking of the social security number. Since there is no addition/deletion of DETs or change in the processing logic for the screens, there is no change to the system’s user functionality, therefore no FPs are counted.

Impacted EPs: “Add new employee,” “View employee personal details,” and “Edit employee personal details.” For each EP:

- 1) Functional: none.
- 2) Non-functional: the masking and unmasking of the social security number are considered non-functional and should only be counted using SNAP.

#### 7.1.4 Sub-category 1.4 internal data movements

##### 7.1.4.1 What to check

The EP

##### 7.1.4.2 Rules

- a) Count SPs when partitions are defined and the EP crosses partitions. Use this sub-category whether a new process is added that cross partitions, or when an existing process is either enhanced or deleted, and it crosses partitions. In addition, count FPs if the processing logic of the EP was modified as per ISO/IEC 20926:2009). The FP counting measures the data movements in and out of the boundary, and the SP counting measures the internal data movements between partitions.
- b) Count SPs when a data flow between partitions is added, changed or deleted to meet the non-functional requirement. (See Example 2 in 7.4.3.)

##### 7.1.4.3 Examples

- a) Example 1: An HR application consists of one application boundary with the front end and the back end as two partitions. A new data entry field is added and is used by three EPs: Add, change and enquire employee. All the employee-related transactions receive input from the front-end application and processed with data from the backend. Since there is a new field added, count FPs for the functional change in the three EPs. In addition, count SPs for the change in internal data movements in these three EPs.
- b) Example 2: An HR application consists of one application boundary with the front end and the back end as two partitions. A screen displays a value D when D is calculated as  $A/(B+C)$ . A, B and C are retrieved from the back-end to the front-end; the calculation is performed by the front-end. To improve performance, it is requested that the value D is calculated at the back-end; data flow to the front end is changed from (A, B and C) to (D).

This screen is used by three EPs.

Count SPs for the data flow change using Sub-category 1.4 per each EP.

---

masking to a data field is to protect data that is classified as personal identifiable data, personal sensitive data or commercially sensitive data, however the data must remain usable.

### 7.1.5 Sub-category 1.5 delivering added value to users by data configuration

#### 7.1.5.1 What to check

The EP

#### 7.1.5.2 Rules

- a) A software application may be designed in a way that user value can be added or changed by adding or changing data in reference tables, without any change to the processing logic. According to ISO/IEC 20926:2009, FPs cannot be counted in this case. Count SPs in such cases.
- b) When functionality is added or enhanced using application source code change along with application configuration or addition of data to reference tables, and it generates FPs, do not add SNAP size.
- c) However, if the user value added requires application source code change and the user value added using configuration are independent of each other, then count the functional change developed by changing the source code with FPs, and the change added by configuration using SPs.

#### 7.1.5.3 Examples

- a) Example 1: CommWorld Company is a telecommunications company. Its BSS application enables CommWorld employees to manage products by geographical location. A new location for product setup can be enabled by adding data to the “Products Profile” table. This table stores data on which product setup functions are available for each location. The company has now expanded into the Asia Pacific Area; for the India location it needs to enable its product setup web pages for DSL packages; for Singapore, it needs to enable its product setup web pages for DSL and Cable networks.

The application maintenance team creates one entry in the “products profile” table for the India location and two entries for the Singapore location corresponding to each product type. Since this change is enabled only by adding values to the table, FPs cannot be counted. Use SNAP for sizing this change.

- b) Example 2: For adding or changing the product setup, the system is required to check internally that if same product package is being created in new geography as “already for sale in another location,” then the corresponding row in the “product package popularity” master table should be updated.

For the second part of the requirement, the new validation and update to product popularity master is a functional change to “add new product package” transaction. Hence, it is FP countable. Since the reference table updates are accompanied by application source code change and they are not independent of each other, SPs cannot be counted here, count only FPs for the entire requirement.

### 7.1.6 Sub-category 2.1 user interfaces

#### 7.1.6.1 What to check

Set of screens as defined by the EP.

#### 7.1.6.2 Rules

- a) Creation of a new user interface (UI) element in order to add, create or change functionality, which is countable using FPs: count FPs and SPs (FPs for the functional requirement and SPs for configuring



the UI element to meet NFR). Creation of a new UI element that does not add or change functionality (such as adding a static menu): count SPs only.

- b) In case of modification of a UI element:
  - 1) If functionality is added or changed and properties of the UI element are changed (see definition of properties in 5.2.1 of this standard), separate the requirements into its functional aspects (counted using FPs) and its non-functional aspects (counted using SPs). SPs would assess non-functional impacts of the change in UI elements.
  - 2) If functionality is not changed and only properties of the UI element are changed (see definition in 5.2.1) then count the change using SPs.

### 7.1.6.3 Examples

- a) ABC Company merged with StarY Company. UI standards followed by the application development team have changed for font size, logo and background color. The application software needs to be modified to meet the latest version of the UI standards. Identify all affected EP. Per each EP, count SPs only to size the UI enhancement requirement.
- b) A new field “Customer Loyalty Points” is added to the “View Bill” transaction. Count FPs for the change (EO) to “View Bill” transaction for the newly added field. Count SPs for the UI element impacts for the new field “Customer Loyalty Points.”
- c) Code data is changed often in the CRM application of Ship-IT Company. To improve the time to implement these changes and avoid user's errors, it is requested to add an admin's screen. This screen shall be used directly by business experts without the need for IT personnel.

Use SNAP sub-category to size the development of the new screen. Use Sub-category 3.2 to size the creation and the maintenance of the code data.

### 7.1.7 Sub-category 2.2 help methods

#### 7.1.7.1 What to check

The help object.

#### 7.1.7.2 Rules

Count SP for any types of “help” that is not FP counted: static text, static web pages<sup>17</sup>, tool tips, dynamic help on mouse over (context help) help window (such as F1 in MS word). See example “Help application” in ISO/IEC 20926:2009 (IFPUG FSM) for FP-counted Help.

NOTE—An EP may have functional and non-functional help types—count functional help per ISO/IEC 20926:2009 rules at the EP level, and the non-functional help at the application level.

#### 7.1.7.3 Examples

- a) Example 1: A photo editing software application “ZoomX” provides a number of photo editing options, which are either available free of cost to users or under a paid license. The requirement is to add tool tips to the existing photo editing tool icons as well as display a message about the corresponding tool usage on mouse hovering. Count SPs to size this requirement for adding the tooltips.

<sup>17</sup>A static web page is a web page that is delivered to the user exactly as stored, in contrast to dynamic web pages, which are generated by a web application. Consequently, a static web page displays the same information for all users, from all contexts.

- b) Example 2: Photo editing application has the pages “About ZoomX” and “Contact us,” which are text only static pages containing ZoomX’s history and company contact information respectively. Count SPs to size these static pages.

### 7.1.8 Sub-category 2.3 multiple input methods

#### 7.1.8.1 What to check

The elementary process.

#### 7.1.8.2 Rules

- a) Organizations that are using the single instance approach (see 3.1).
  - 1) Count FPs for the first input method only.
  - 2) Count SPs for each additional input method.
  - 3) For new development requiring  $n$  input methods, count FPs for one input method and SPs for  $(n-1)$  input methods.
- b) The organization is using the multiple instance approach (see 3.1)
  - 1) Count FPs per for each input method.
  - 2) Do not count SPs.

#### 7.1.8.3 Example

A health care claim insurance software application receives claim inputs via web screens (maintained by another application outside the boundaries) or via batch inputs. Organizations using single instance approach should count claim input via the screen (first instance of same input type) using FPs. All other similar inputs, like batch input, would be counted using SPs.

Organizations using multiple instance approach already accounted for this complete functionality using FPs. So SPs cannot be applied in those cases.

### 7.1.9 Sub-category 2.4 multiple output methods

#### 7.1.9.1 What to check

The EP.

#### 7.1.9.2 Rules

- a) The organization is using the single instance approach (see 3.1).
  - 1) Count FPs for the first output method only.
  - 2) Count SPs for each additional output method.
  - 3) For new development requiring  $n$  output methods, count FPs for one output method and SPs for  $(n-1)$  output methods.<sup>18</sup>
- b) The organization is using the multiple instance approach.
  - 1) Count FPs per for each output method.
  - 2) Do not count SPs.

<sup>18</sup>Only in case that they are identical EP that don’t meet the definition of uniqueness in ISO/IEC 20926:2009.

### 7.1.9.3 Example

The health care claim insurance application sends the processed claim output report via web screens, as a paper output or as a downloadable .pdf. Organizations that use the single instance approach should count the claim report output via the screen (first instance of same output type) using FPs but all other similar outputs would be counted using SPs. Organizations that use the multiple instance approach have already accounted for this complete functionality using FPs. SPs should not be applied in those cases.

### 7.1.10 Sub-category 3.1 multiple platforms

#### 7.1.10.1 What to check

The EP.

#### 7.1.10.2 Rules

- a) For  $n$  platforms of any category (as defined in 5.3.1) while  $n > 1$ , count SPs per 5.3.1.
- b) If  $n = 1$ , do not count SPs.

#### 7.1.10.3 Examples

- a) For a banking application written in JAVA, the “my account view” transaction is required to be available on three different browsers—Internet Explorer, Chrome and Firefox in the same format. The first platform deployment is FPs counted; count SPs for required transaction compatibility on the two other platforms, using this category.
- b) For a banking application written in JAVA, the “export customer data” transaction is executed nightly. This transaction is a functional requirement and is counted using FPs. SPs cannot be counted. If the same transaction had to be re-written in VB to support any legacy system for the same application boundary, then SPs should be used to size the additional language support.

### 7.1.11 Sub-category 3.2 database technology

#### 7.1.11.1 What to check

The EP.

#### 7.1.11.2 Rules

- a) A database change to meet a non-functional requirement (such as adding a technical field or a change to improve performance), is not counted as a functional change. Such a database change may be done in addition to functional changes or to meet a pure non-functional requirement. In both cases, count the database change itself in SPs only. See the definition of database changes in 5.3.2.

#### 7.1.11.3 Examples

- a) A system requires exception handling, where the error code and description are maintained in a code table. Count SPs for the table changes.
- b) The user has requested a performance improvement for the “search business partner” EP to support B2E (Business-to-Enterprise) transactions. The transactional function is an EO. The development

team creates a secondary index file for the “Business Partner Name” column on the “Partner” table to improve the search response time. Count SPs using this category for the new index file being created.

### **7.1.12 Sub-category 3.3 batch processes**

#### **7.1.12.1 What to check**

The user identified batch job.

#### **7.1.12.2 Rules**

- a) When a user identifiable batch job does not cross the application's boundary and, therefore, is not qualified as any of the transaction functions per FPA rules because it does not cross the boundary, use SNAP to size the batch jobs.
- b) When an existing batch job is divided into multiple jobs and the reason to do so is to meet NFR, count SPs to size the batch jobs.

#### **7.1.12.3 Examples**

- a) The marketing data warehouse has a technical requirement for automatic archival of marketing campaigns older than 5 years, to improve performance. (The archiving is not triggered by a control EI.) An end-of-month job archives data based on this criterion. The job reads data from within the database and after applying the necessary logic, moves the data to archive tables within the same database without any data crossing the boundary as input or output. Count SPs for this requirement.
- b) In a travel application, the requirement is that the user can cancel a ticket. Post-cancellation, the system automatically does a background refund calculation and initiates a new entry (task) to the service module for customer service agent to issue the credit note manually. As per in ISO/IEC 20926:2009 (IFPUG FSM), any transaction which does not cross the application boundary is not FP counted. Counting teams may have assumed auto triggers as the DET crossing the boundary and accounted for this transaction in FP. In such case, where the auto trigger is assumed as part of local FPs guidelines, do not count SPs. Otherwise, count the above requirement using SPs.

### **7.1.13 Sub-category 4.1 component-based software**

#### **7.1.13.1 What to check**

The EP.

#### **7.1.13.2 Rules**

- a) This subcategory adds non-functional size to the counted functionality when the software comprises of re-usable components.
- b) Count the functional requirement using FPs per ISO/IEC 20926:2009, per each EP.
- c) In addition, count SPs when the EP uses components within the application boundary.

### 7.1.13.3 Example

A Travel Application is required to build software components, which can be re-used across different travel applications for booking product types (such as hotel reservation, flight reservation and car rental). All the components are built in-house. Reusable components include the following:

- GetPassengerDetails.
- ProcessPayments.
- SendUserNotification.

Each of the product types would have its own EP for booking (book a flight, book a hotel) which would use the above re-usable components along with other processing logic, which is unique to that process. Count FPs to size the respective EP for each product type. Count SPs to size the requirement for reusable components to be created.

### 7.1.14 Sub-category 4.2 multiple input/output interfaces

#### 7.1.14.1 What to check

The EP.

#### 7.1.14.2 Rules

- a) Count the functional requirement using FPs for sizing the required added, changed and deleted functionality. If a new development or a new requirement includes multiple inputs and /or outputs, count FPs for the functional aspect and SPs for the additional interfaces.
- b) When a functional enhancement is required, count FP for the change (as if one interface exists) and add SPs for the additional number of interfaces (according to 5.4.2).
- c) When there is no functional change in the counted application and the only change involves adding or removing interfaces that have same DETs and FTRs and do not require any functional change, count SPs using this sub-category for only the added or removed interfaces.

#### 7.1.14.3 Example

An online retail application is required to support an increased volume of user transactions for its mega discount offers on Christmas. Application maintenance team decided to add new servers to support the increased volume. (Software changes are required to support it). Use SPs for sizing the setup required for all processes enabled via use of additional servers.

### 7.1.15 Additional notes

Having the same person count both FPs and SPs for a requirement can help to avoid double counting and provide a more efficient and accurate count of each type.

If an organization has developed their own organization guidelines based on ISO/IEC 20926:2009 (IFPUG FSM), and some non-functional aspects have been included in the FP counting, the organization should revise the guidelines to count both FPs and SPs based on ISO/IEC 20926:2009 and this standard.

## 8. Use of non-functional software sizing

This clause describes some uses for SNAP. It is neither intended to be a manual for the use of SNAP nor is it intended to be exhaustive.

The uses of SNAP are organized into two parts: uses for project management and uses for performance management.

### 8.1 Functional size and non-functional size

Software can be classified into two types of categories as experienced by the software user: functional and non-functional.

The functional software is the software that generates the flow and storage of data for a software application. According to ISO/IEC 20926:2009, there are five different ways that data flows and is stored. Data enters the application through EIs. This data is then stored internally in some kind of file structure. Although there are many different programming languages and programming styles for these file structures, the user is usually indifferent to these. ISO/IEC 20926:2009, therefore, recognizes logical file structures, which are called internal logical files (ILFs). Data leaves an application in one of two ways. Data “reports” sent to users on a read-only basis are classified as external inquiries (EQs). Data reports sent to users that require calculations are classified as EOs. Sometimes the application needs data that belongs to another application; the part of the logical file in that other application that is needed is an external interface file (EIF). The total volume of data flow, called “functional software size,” is measured in FPs. For example, an application of 500 FPs in size has half of the functionality as another application of 1000 FPs.

The non-functional software is basically all of the other types of software that do not directly involve the flow and storage of data as defined by ISO/IEC 20926:2009. For example, the software may contain embedded help, which may be extensive. Much software requires data validation, such as validation of logins, passwords, or other identifying codes. Also, some software contains complex algorithms or numerous algorithms. The Help, data validations, and algorithms all require work effort to develop and these are sized using SNAP. The standard unit of non-functionality is called the “SP.” An application of 500 SPs in size has half of the non-functionality as another application of 1000 SPs.

Therefore, if Project A has 500 FP and 100 SPs, and Project B has 400 FP and 250 SPs, it is not enough data to determine which project is bigger, but it is enough information (provided that the two projects have the same projects’ characteristics) to estimate cost, resources, and other parameters of these projects

### 8.2 Project management

This description of the uses of SNAP addresses how SNAP could be applied to the management and control of software projects.

#### 8.2.1 Project resource forecasting

For new development and enhancement projects, an algorithmic forecasting model can be constructed from various types of data collected from a sample of completed projects, together with FSM. For example, the following information can be used as an input into the model:

- a) The functional size of the software
- b) The non-functional size of the software
- c) Project’s characteristics (e.g., project type, software development life cycle type, location of development)
- d) Additional project’s constraints

### 8.2.2 Software cost estimation

Software development contracts, especially their statements of work, can specify at least three standards to which the project team can be measured. These are software cost, development schedule, and software quality.

When possible, it is advisable to set prices or costs by the unit of item purchased. For example, gasoline may cost \$2.50 per “gallon,” meat may be sold by the “pound,” and electricity may be sold by the “kilowatt hour.” In like manner, both functional and non-functional software can also be sold or purchased by the unit of item developed or purchased. The functional software could be priced in terms of “dollars per FP,” and non-functional software could be priced in terms of “dollars per SP.” This is not to say that each FP in a particular application will cost the same to develop, but it is to say that a statistically large number of FPs will cost, on average, the same amount. The larger the application, the more representative the average cost per FP becomes. The same applies to SPs.

A software application may likely have both functional and non-functional components. Suppose one wants to estimate the cost to develop an application and uses metrics to be the base for the cost estimate. If the application has 400 FPs costing an average of \$700 each, the functional component of the software should cost about \$280 000. If the application also has 300 SPs costing an average of \$200 each, then about \$60 000 should be allocated to the SNAP development cost. The total application cost would then be estimated at about \$340 000. This is usually understood to include overhead costs such as management, analysts, testers, team meetings, etc.

The organization can estimate its costs per FP and SP based on one of at least two methods. The best method is for an organization to calculate its own costs per FP and SP using organizational data and a statistically large number of FP counts and SP counts. The second-best way is to use industry benchmarks, recognizing that the organization’s cost structure may be somewhat different than benchmark.

It is important to recognize that software size should be included in cost estimation (cost per FP or SP) to normalize the metric. A million-dollar cost estimation may be prohibitively large for an 800-FP project, but very favorable for a 1400-FP project.

### 8.2.3 Software project duration estimation

Software development schedule can be estimated based on an average of total FPs delivered per development day. For example, if the life cycle duration of a certain project was 300 workdays, and the project generated 900 FPs, then the metric would normally calculate an average of about three FPs per day during the lifetime duration of the project. To generalize, an organization can calculate its average number of FPs per day from a statistically large number of projects to get its internal duration benchmark. It can do the same for its SP production rate. A number of FPs and SPs per day can be used in Statements of Work to set the duration of a software development project as a standard for measuring the normalized timeliness of project completion.

### 8.2.4 Software quality estimation

Software quality was initially defined by IFPUG in 1991 using the metric.

$$\frac{\text{Defects}}{\text{Function points}} \quad (1)$$

For example, if applied at system testing, if an application was found to have 10 defects and was 300 FPs in size, then its defect ratio would be calculated as:



$$\frac{10 \text{ defects}}{800 \text{ function points}} \quad (2)$$

Or 0.033 defects per FP. If this was satisfactory to project management at system testing in the life cycle, then this could be used as a future standard for future development efforts, or for use in statements of work.

Since 1991, and especially since the development of SNAP, this metric has been updated. One update is to incorporate both FP and SP sizes into the ratio's denominator. The resulting IFPUG formula is:

$$\frac{\text{number of defects}}{\sqrt{\text{function points}^2 + \text{SNAP points}^2}} \quad (3)$$

where the denominator is now called “equivalent size.” For example, if an application was 400 FPs and 300 SPs in size, then the value of the denominator would be 500. If 10 defects were found at system testing, the quality measure would be:

$$\frac{10 \text{ defects}}{\sqrt{400^2 + 300^2}} = \frac{10}{500 \text{ (equivalent size)}} \quad (4)$$

If there are no SPs, then the ratio reverts to the original defects/FP.

The second update is to recognize that not all defects have the same severity. Some defects might introduce a virus into the network and are intolerable defects. These could be classified as “Level 1” defects. A defect that caused an application to become dysfunctional could be classified as a “Level 2” defect. A defect that was a misspelled word that did not affect the operation of the application or other minor cosmetic defects might be classified as “Level 5” defects. Levels 3 and 4 would be defined as intermediate severity levels.

A quality standard would therefore use the formula above for each level of defect. For example, for each of Level 1 and Level 2 quality standards at system testing, the acceptable ratio might be:

$$\frac{0(\text{Level 1 acceptable defects})}{\sqrt{\text{function points}^2 + \text{SNAP points}^2}} \quad (5)$$

At Level 5, the acceptable number of defects might be a number higher than 0, as requiring perfection at each level at system testing may be unreasonable or too costly.

Alternatively, an organization could develop other quality standards based on level, with the equivalent size still being the denominator. The equivalent size is needed for normalization.

Software quality standards can be set for each level of severity. Based on company records, such defect rates can be estimated for various testing points in the life cycle such as unit testing, system testing, or defects found after perhaps 90 d (or some other time frame) after release. Industry benchmarks can also be useful.

### 8.2.5 Updating estimates

When a software development project is completed, it is important to compare the cost, schedule, and quality estimates with the actuals realized. Project estimation algorithms can subsequently be adjusted for future projects. Some areas to consider (and there are probably many others) are described in 8.3.

Estimates are often based on the “typical” project, and if the project turned out to be not typical, then lessons learned can be compiled to help calibrate future project estimates based on the degree to which a future project differs from the “typical” project.



Sometimes early assumptions turn out to be in error. For example, one may have assumed little or no requirements creep, but the project turned out to have, say, 15% more FPs delivered than originally forecast due to requirements creep. The delivery of these 15% more FPs realized a corresponding increase in schedule and cost.

A given project may have had numerous requirements changes. A lesson learned may consider adding a change factor to scheduling algorithms for future project schedules.

Software quality can be measured at certain times in the life cycle, and at certain times after release. A lesson learned from an existing project may be to measure software quality at different times for future projects.

Early in a software metrics program, achievement of costs/schedule/quality metrics may be based on industry benchmarks. These industry benchmarks may or may not reflect the organization. After many organization projects are completed, an organization should shift its standards based on organizational benchmarks.

### **8.2.6 Tracking the progress of a project**

The project management can track and communicate the progress of the project by noting the functional size and the non-functional size that have been developed. The project's progress can then be communicated as the percentage of target sizes, which have passed a milestone or have been completed.

### **8.2.7 Managing scope change**

Together with tracking the changes in the functional size (adding, changing or deleting FPs), similar tracking of SPs can be used to measure scope changes. For example, improving the time behavior of the system after performing load and stress testing (testing the system with over-capacity or when many users approach the system concurrently) can be measured using SNAP, and an estimation model can be used to estimate the extra effort and duration needed to improve the performance of the system.

### **8.2.8 Post-mortem analysis**

The actual cost, resources, and effort related to the development and enhancement of delivered functional size and non-functional size should be recorded to make them comparable with other projects.

### **8.2.9 Conclusion**

Three metrics need to be included in any software development statement of work: cost, schedule, and quality. This is to prevent, for example, an economic incentive to produce low cost yielding poor quality, short schedules resulting in low cost but poor quality, or exceptional quality resulting in intolerable cost and schedule overruns. These three metrics promote balanced outcomes, as set by either the project management or the organization. They can set standards for contractors and for project management.

This estimation of these metrics should include normalization for software size, both in terms of FPs and SPs. For example, a project yielding 50 defects at system testing is not necessarily of higher quality than a project yielding 100 defects. The quality depends in part on the size of the software. If the 50-defect project was for an application of 100 FPs, then the defect per FP ratio would be 1/2. If the 100-defect project contained 1000 FPs, then the defect per FP ratio would be much less—1/10; the second project would have much higher quality than the first. Similar lines of reasoning apply to both cost and schedule, and also to SP metrics.

A large portion of software is functional, but estimation metrics based only on functional size are not complete. Non-functionality should also be included in the estimates to allow a more complete view of the software application and development effort.

## 8.3 Performance management

This clause addresses how functional size and non-functional size could be applied to forecasting resource usage and the management of performance. This typically involves the use of both FPs and SPs as normalizing factors and the collection of a large amount of data to create models.

### 8.3.1 Productivity

Software development productivity is a measure of the volume of data processing capacity developed by a software development team per unit of time. It can also refer to the volume of other requirements, which supplement data processing capacity per unit of time. Software development productivity is not measured per individual team member.

Organizations seeking to continuously improve their productivity need to have productivity performance standards in place from which to measure progress. These standards often need to be normalized according to the basic measure or driver of production. For example, a brewery bottle shop might have a productivity measure in terms of cases of beer produced per shift—the normalization factor being “cases” of beer. A fast food restaurant might have a measure of its productivity in terms of the median time to wait on a customer—the normalization factor being the “customer.” For software development, two normalized measures of productivity can be used.

Much software is written to facilitate the flow and storage of data in and through an application. The units of volume of this type of software data processing capacity can be measured in terms of FPs. Productivity of data processing capacity software development by a software development team over the software development life cycle can be normalized by the “FP.” For example, if a software development action for data processing capacity required 100 staff days to complete its full life cycle, and delivered 80 FPs, then its productivity could be measured as 100 d/80 FPs, which could be expressed as 1.8 d to deliver a FP. Organizations wanting to measure the trend in productivity can plot its “days to develop a FP” over time to measure to determine if new work practices yielded fewer days to deliver a FP over the project life cycle. There are numerous other potential productivity metrics based on FPs published by the International Function Points Users Group in *Function Points as Assets Reporting to Management* [B4], Section 3.0, Productivity Metrics, and these can be adapted to focus on functional requirement development.

There is also software that is required by users that does not directly involve the flow and storage of data in and through an application. This is non-functional software. One example of non-functionality is data validation. When a driver attempts to enter a military installation, that driver may be stopped at a security gate for identification validation. If the validation is verified, then the driver is allowed passage into the installation, but if not verified, the driver must leave. Suppose that operating a software application requires a user to enter a password. In principle, when a password is entered, its flow into the application is “stopped” at a “gate” so that it can be verified. If verified, then the user is allowed further access to the application. If not verified, then the user is “bounced out” of the application. This process of data validation occurs when the flow of the password data is “stopped at the gate” and is therefore non-functional. Where a standard unit of functionality is the FP, a standard unit of non-functionality is the Software Non-functional Assessment Process (SNAP) point. Productivity metrics normalized by FPs can also be suitably normalized by SPs. For example, SNAP productivity can be measured as the total software development life cycle staff hours focusing on NFR divided by SPs.

It may seem to be intuitive that the total size of software can, therefore, be measured by adding the number of FPs to the number of SPs—such as 400 FPs added to 200 SPs yielding 600 “size points” for the total software size. Therefore, the total productivity could be measured as total software lifecycle staff hours/total “size points.”

Unfortunately, this is mathematically unsound because FPs and SPs measure two different aspects of software. Adding them together would be a mathematical operation in error similar to adding a person’s height in inches

or centimeters, to their weight in pounds or kilos, to get a “size of the person” measure; height and weight are two different aspects of a person and their measures are not additive (i.e., centimeters do not convert into kilos). So to measure software productivity, it is best to measure the functional productivity as one number, and the non-functional productivity as a different number, and have management programs to improve the productivity of each separately.

There are several moral hazards when using other types of metrics to measure productivity. One metric commonly used is normalizing by the number of lines of code. Imagine a college professor grading term papers by counting the number of sentences—900 sentences yielding an “A,” 800 sentences yielding a “B,” etc. The economic incentive for students would be to generate at least 900 sentences to achieve the A grade, but both quality of those sentences and the overall quality of the papers’ academics could certainly be in question. Measuring student productivity progress by how many sentences produced per week would be equally questionable. That being said, a sentence to a language is like a line of code to a programming language. Many organizations measure productivity by normalizing using some form of lines of code count, and measure progress by the number of lines of code generated per month. Measuring software development performance using some form of lines of code metric or similar metric puts the manager in much the same position as the college professor using the metric “number of sentences” to indicate student academic performance (IFPUG 2016) [B6].

### 8.3.2 Quality

Software quality is an indicator of the degree to which the delivered software meets customer and stakeholder requirements. A full software quality determination program might include quantitative metrics based on FPs and SPs, and also associated qualitative customer satisfaction surveys. This entry focuses on measurements by FPs and SPs. Much of the following discussion is adapted from the IFPUG white paper *Integrating Procedures for Function Point Analysis and the Software Non-functional Assessment Process (SNAP)*, Part 2 [B9].

One of the first metrics for measuring software quality was from the IFPUG publication “Function Points as Assets,” (IFPUG 1992 [B4]) described on Page 24 as the defect ratio. This has been adapted to the following formula:

$$\text{Quality} = \frac{\text{Number of defects}}{\text{Function points}}$$

The normalization by FPs allows easy comparison between the qualities of different applications. For example, a 1000-FP application having 100 defects at release would be measured as having 100 defects per 1000 FPs, or 0.1 defect per FP. Another application having 200 FPs, and 50 defects, at release would have its quality measured at 50 defects per 200 FPs, or 0.4 defects per FP. One can initially conclude that the quality of the first application is four times better than the quality of the second application.

Reflecting technology improvements since 1992, it is recognized that there are different levels of software defect severity. For example, an organization might define a “Level 1” defect as a defect, which places malware onto an organization’s network thereby rendering it unusable. A “Level 2” defect might be defined as a defect which renders the application unusable. In contrast, a “Level 5” defect might be defined as one which does not interfere with the application’s operation – such as misspelling a name in an output field on a report. So an organization might improve the original metric by some form of:

$$\text{Quality} = \frac{\text{Number of defects at severity level } x}{\text{Function points}}$$

and have quality performance standards for each level. One such standard might be “0 Allowable defects at Severity Level 1/FP”

The above discussion assumes that all defects found relate to functional software – that which involves the flow and storage of data in the software. Other defects are non-functional, such as non-compliance with performance requirements or non-compliance with accessibility standards. It is possible to, therefore, design a quality metrics improvement with two sets of metrics, such as:

$$\text{Quality} = \frac{\text{Number of functional defects}}{\text{Function points}} + \frac{\text{Number of non-functional defects}}{\text{SNAP points}}$$

The difficulty with this approach, although valid and mathematically sound, is that it requires the software quality assurance inspectors to be able to distinguish between functional and non-functional defects – in other words, in the “best” circumstances to add professional certification in both FPs and SPs to their skill sets. Moreover, defects that are related to non-functional aspects, as described in this standard, may be interpreted as functional defects. For example, errors in internal data movements, or errors in one of the outputs, in case of multiple outputs methods. This is very impractical and if possible would be very time-consuming.

It may seem intuitive to, therefore, add the FP and SNAP counts into a metrics like below.

$$\text{Quality} = \frac{\text{Number of defects}}{\text{Function points} + \text{SNAP points}}$$

Unfortunately, this is mathematically unsound because FPs and SPs measure two different aspects of software. Adding them together would be a mathematical operation in error similar to adding a person’s height in inches or centimeters, to their weight in pounds or kilos, to get a “size of the person” measure; height and weight are two different aspects of a person and their measures are not additive (i.e., centimeters do not convert into kilos). Therefore, to measure software quality, it is best to measure using the following formula based on linear algebra:

$$\text{Quality} = \frac{\text{number of defects}}{\sqrt{FP^2 + SP^2}}$$

In this way, total defects (per severity level) can be counted and the total number of FPs and SPs can be used for the denominator.

### 8.3.3 Organizational maturity and process capability

FSM and NFSSM can provide a base of quantitative measurement necessary to support higher levels of organizational maturity or process capability.

### 8.3.4 Accounting for an organization’s software asset

Both functional size and non-functional size could be measured for part or all of an organization’s application portfolio and entered into an estimating model to determine the software asset value or the total cost of replacement, re-engineering, or outsourcing.

### 8.3.5 Budgeting for maintenance

NFSSM can assist with budgeting for the maintenance of an organization’s software portfolio. The maintenance cost or effort per SP could be monitored and be used to forecast maintenance budgets.

### 8.3.6 Contract management

NFSSM can be used as part of managing the cost and schedule of software development by software suppliers.

## Annex A

(informative)

### NFSSM strengths

This annex describes several value-added reasons for using non-functional sizing methods in a software development project metrics program.

There is a strong commercial industry of software developers serving the needs of its customers. Customers typically release a set of requirements (a “request for proposal”) to the industry of developers, and those developers bid on the proposals in part competing on price. A moral hazard faced by customers can occur when the proposal asks developers to bid based on “price per FP.” “Price per FP” adds value to the contracting process because it requires customers to both describe the software’s requirements and to declare the approximate size of the software to be developed. Then users can compete based on their prices per FP. It is a straightforward matter at the end of the development to count the delivered FPs, and then pay the developer based on the agreed-upon rate. The moral hazard enters with this type of contract when the customer requests enhancements which are non-functional—such as upgraded password protection or new reports based on new algorithms. These new enhancement requirements may be the result of a law change or a regulation change and must be developed. The password protection upgrade probably requires a more sophisticated data validation, and the new report may need a new computational algorithm. Neither the data validation nor the algorithms are countable using IFPUG FPs (but are countable under SNAP), so the developer should allocate work effort to deliver these new requirements. The developer then faces a choice of either providing these new software features, but not get paid for the associated work, or to not produce these features and risk delivering an application which does not conform to the new law or regulation. The solution to this dilemma is to also incorporate a “price per SP” into the contract. In this way, any new legal or regulatory requirement measurable by SNAP can be developed and paid for.

Part of the vendor industry bid on a software development request for proposal is based on the amount of software expected to be developed, and based on the risk inherent in not knowing exactly how much software is to be actually developed. For example, if a customer releases a proposal and declares that the software requirements translate into between 500 and 600 FPs, and 200 to 250 SPs, then the industry can bid on that size of software. However, if the number of FPs and SPs are not declared in the proposal, then the vendor industry should “guess” at the total size. They have an economic incentive to estimate the actual volume of software to develop, and then to add a “management reserve” (in this example of perhaps 200 extra FPs and 100 extra SPs to their original estimate) to protect themselves from underbidding. All vendors in the industry have this economic incentive when the functional and non-functional sizes are undeclared by the customer. This situation can artificially increase the basic bids from the vendors. However, if the proposal includes the estimated size ranges for both FPs and SPs, and makes provision for a penalty to be experienced by the customer if the final size is above the range, then this greatly provides an economic incentive to reduce or eliminate this “management reserve” needed by industry and there is an economic incentive for all bidders to reduce their cost per FP and SP bids. See the IFPUG case study “How to Use Function Points and SNAP to Improve a Software Acquisitions Contract” (IFPUG 2014) from which this paragraph was adapted.

Part of the reason for conducting and recording FP and SNAP counts is to maintain an inventory of such assets. Adding the non-functional assets to the inventory makes such accounting more complete.

## Annex B

(informative)

### Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

[B1] HIPAA (Health Insurance Portability and Accountability Act of 1966).<sup>19</sup>

[B2] IEEE Std 1062™-2015, IEEE Recommended Practice for Software Acquisition.

[B3] IEEE 1175.2™-2006 IEEE Recommended Practice for CASE Tool Interconnection—Characterization of Interconnections.

[B4] International Function Point Users Group, Function Points as Assets Reporting to Management. Princeton Junction, NJ: International Function Point Users Group, 1992.

[B5] International Function Point Users Group, How to Use Function Points and SNAP to Improve a Software Acquisitions Contract. Princeton Junction, NJ: International Function Point Users Group, 2014.

[B6] International Function Point Users Group, Integrating Procedures for Function Point Analysis and the Software Non-functional Assessment Process (SNAP), Part 2. Princeton Junction, NJ: International Function Point Users Group, 2016.

[B7] ISO/IEC/IEEE 12207:2017 Systems and software engineering—Software life cycle processes.

[B8] ISO/IEC 14764:2006 Software Engineering—Software Life Cycle Processes—Maintenance.

[B9] ISO/IEC 19770-1:2017 Information technology—IT asset management—Part 1: IT asset management systems—Requirements.

[B10] ISO/IEC/IEEE 24765:2017 Systems and software engineering—Vocabulary.

[B11] ISO/IEC/IEEE 26513:2017 Systems and software engineering—Requirements for testers and reviewers of information for users.

[B12] Software Non-functional Assessment Process (SNAP) Assessment Practices Manual, Release 2.4, 2017.

[B13] Tichenor, C., “A New Software Metric to Complement Function Points—The Software Non-functional Assessment Process (SNAP),” Crosstalk, vol. 26, no. 4, pp. 21–26, 2013.<sup>20</sup>

[B14] Tichenor, C., “Industry Best Practices V. Moral Hazard in Software Development,” MetricViews, vol. 10, no. 2, pp. 15–16, August 2016.<sup>21</sup>

[B15] WCAG 2.1 (W3C Recommendation 05 June 2018) Web Content Accessibility Guidelines.<sup>22</sup>

<sup>19</sup>Available at: <https://aspe.hhs.gov/report/health-insurance-portability-and-accountability-act-1996>.






<sup>20</sup>Available at <http://static1.l.sqspcdn.com/static/f/702523/23063507/1373252377207/201307-0-issue.pdf?token=UBF6yih09HnRi%2B14oZWPWFUhpA%3D>

<sup>21</sup>Available at: <https://www.ifpug.org/Metric%20Views/MetricViewsAugust2016.pdf>

<sup>22</sup>Available at: <https://www.w3.org/TR/WCAG21/>

# RAISING THE WORLD'S STANDARDS

## Connect with us on:

-  **Twitter:** [twitter.com/ieeesa](https://twitter.com/ieeesa)
-  **Facebook:** [facebook.com/ieeesa](https://facebook.com/ieeesa)
-  **LinkedIn:** [linkedin.com/groups/1791118](https://linkedin.com/groups/1791118)
-  **Beyond Standards blog:** [beyondstandards.ieee.org](https://beyondstandards.ieee.org)
-  **YouTube:** [youtube.com/ieeesa](https://youtube.com/ieeesa)

[standards.ieee.org](https://standards.ieee.org)  
Phone: +1 732 981 0060