

COSMIC - ISO 19761

Planning a Software Upgrade: the SimSE Java v8 to v17

Case Study

COSMIC Version 5.0

September 2026

Foreword.

Software upgrades triggered by portability concerns are often managed as technical code-fixing tasks and do not use software functional size measurement standards for planning or collecting historical size-effort data for later use in estimation and planning contexts. This Case study presents a COSMIC ISO 19761-based approach to size a portability-driven software upgrade and illustrates it with the case of a Java upgrade of SimSE, an educational software engineering game generator.

- This Case Study is based on the paper “A Software Version Upgrade sized with the COSMIC ISO standard: Management Lessons Learned from a Case Study” [1]. Texts, Figures and Tables are reproduced in this case study with authors’ permission.
- For a description of SimSE, see “SimSE, an educational software engineering game generator” [2].
- For COSMIC sizing terminology see the Measurement Manual Part 1 [3].
- For the COSMIC measurement guidelines see the Measurement Manual Part 2 [4].
- For the SimSE Games Generator Sizing Case Study see [5].
- For COSMIC and system portability requirements see [6].
- For fundamentals of software estimation see [7].

Editors:

Ling Guo, Ecole de technologie supérieure (Canada),

Alain Abran, Ecole de technologie supérieure (Canada),

Arlan Lesterhuis (The Netherlands).

Other members of the COSMIC Measurement Practices Committee:

Jean-Marc Desharnais, Ecole de technologie supérieure – University of Quebec (Canada),

Peter Fagg, Pentad (UK),

Andrés Gutierrez, Leda MC (Colombia),

Dylan Ren, Measures Technology LLC (China),

Bruce Reynolds, Tecolote Research (USA),

Hassan Soubra, ECE Lyon - Ecole d'Ingénieurs - Engineering School (France),

Sylvie Trudel, Université du Québec à Montréal - UQAM (Canada),

Francisco Valdés Souto, Spingere (Mexico),

Frank Vogelezang, Adviescollege ICT-toetsing (The Netherlands).

Copyright 2026. All Rights Reserved. The Common Software Measurement International Consortium (COSMIC). Permission to copy all or part of this material is granted provided that the copies are not made or distributed for commercial advantage and that the title of the publication, its version number, and its date are cited and notice is given that copying is by permission of the Common Software Measurement International Consortium (COSMIC). To copy otherwise requires specific permission.

Table of Contents

1 INTRODUCTION. 4

2 MEASUREMENT STRATEGY PHASE. 4

2.1 Software upgrades and evolution effort estimation. 4

2.2 COSMIC, early sizing, and generated applications. 5

2.3 Portability requirements as measurable software functional changes. 5

3 CASE STUDY CONTEXT AND UPGRADE CONVERSION ACTIVITIES. 6

3.1 SimSE environment: an educational games generator and its set of generated software games. 6

3.2 Study design and phases 6

3.3 Technical diagnosis and related upgrade work 7

3.4 Post-upgrade COSMIC measurement procedure for the functional processes modified 9

4 UPGRADE SIZING ANALYSIS 11

4.1 Technical conversion contributions 11

4.2 Post-conversion COSMIC measurement contribution. 11

4.3 Key Learning Points – collecting a baseline for estimation and planning purposes. 12

5 REFERENCES AND DOCUMENTATION. 12

1 INTRODUCTION.

The upgrade analyzed in this case study is based on a project to move from Java 8 to Java 17 and to support portability across Microsoft Windows and macOS/Sonoma platforms. The software applications to be upgraded were the SimSE educational software environment, which includes a model builder, code generator, execution environment, and supporting tools. COSMIC guidance has also shown that generated applications can be measured and reasoned systematically [6].

The motivation for this case study was using a functional size-based approach to estimate the software development effort in the context of system non-functional requirements allocated to software functions and in particular how a COSMIC-based sizing approach can support the planning of a portability-driven software upgrade and produce baseline data for later efforts modelling.

The upgrade was organized in two phases:

(1) observation and analysis of the initial upgrade planning stage and subsequent technical conversion activities, and

(2) post-upgrade identification of the software functions modified during the upgrade, their COSMIC functional sizing, and analysis of sizing information for planning contributions.

In the upgrade planning: the Java v7 to v18 diagnosis identified hundreds of errors and warnings, of which 93 were critical and related to wrapper constructors deprecated for removal; this defined the upgrade conversion scope.

In the post-upgrade phase: the COSMIC measurement showed that 47 of the 65 SimSE functional processes had been modified, yielding 220 COSMIC Function Points (CFP) of functional changes relative to its pre-upgrade size corresponding to 77% of the SimSE baseline functional size.

Such quantitative data, based on an ISO standard, can be used to produce baseline size-effort data for future effort estimation models for planning and benchmarking purposes of software upgrades projects.

2 MEASUREMENT STRATEGY PHASE.

2.1 Software upgrades and evolution effort estimation.

Software upgrade work belongs to a broader family of adaptive and perfective evolution. Studies have shown that functional size-based measures and their simpler variants are useful for software development and maintenance effort estimation. These studies also caution that estimation accuracy depends on dataset quality, project context, and model choice. In parallel, functional change impact analysis research emphasizes that making the likely scope and propagation of functional change explicit can reduce evolution costs and support more disciplined planning. Migration studies likewise describe modernization work as long-running and multi-faceted, requiring not only technical execution but also planning of supporting artifacts and intermediate transition steps.

In portability-driven upgrades, for example, the trigger for change is usually clear, but the eventual functional impact is unclear. This makes early planning difficult when it is not known whether functional size-based measurement methods are relevant to this context in terms of both applicability and usefulness. A functionalized-based approach can be initiated and is useful even before an organization has enough observations to calibrate an effort-prediction model.

2.2 COSMIC, early sizing, and generated applications.

When detailed requirements are not fully available, the COSMIC community recommends approximate sizing techniques to support feasibility studies and early requirements analyzes. The Practitioners Guide describes several early sizing techniques, including Early & Quick Approximation, the Software Iceberg Analogy, and Easy COSMIC Function Points for feasibility-stage use, as well as Average Size of Functional Processes, Fixed Size Classification, Equal Size Classification, and Average Use Cases for early requirements-stage estimation. For environments that include application generation, dedicated COSMIC guidance also shows how the generated applications and application generator can be sized consistently. Such guidance is relevant for SimSE, which is not a simple standalone application but a generator-centered environment.

When historical size-effort data of comparable projects are available in the post-upgrade analysis, regression analysis ('Ordinary Least Squares') can be used in the planning stage.

2.3 Portability requirements as measurable software functional changes.

The COSMIC practitioner's guide for sizing non-functional requirements explains that system-level non-functional requirements may be allocated to software functionality and then sized. This is particularly relevant for the variety of portability functions for which COSMIC sizing is illustrated.

This argument is directly relevant to the present case study. Such a Java 8-to-17 upgrade and the need to keep the software executable across operating environments are not merely technical settings: they imply software modifications that must be identified upfront and sized.

A high-level view of a standards-based model of the key characteristics and functions of portability is presented in Figure 1, which defines and details the portability of Software Components, Data Components, Hardware Components and Isolation of System Calls. The 'upgrade' functions sized with COSMIC fall under Component 1 (Software Component Portability), more specifically under IPLVMF (Independence of the Programming Language Virtual Machine Function), as upgrading to a new version of Java involves a change in the virtual machine.

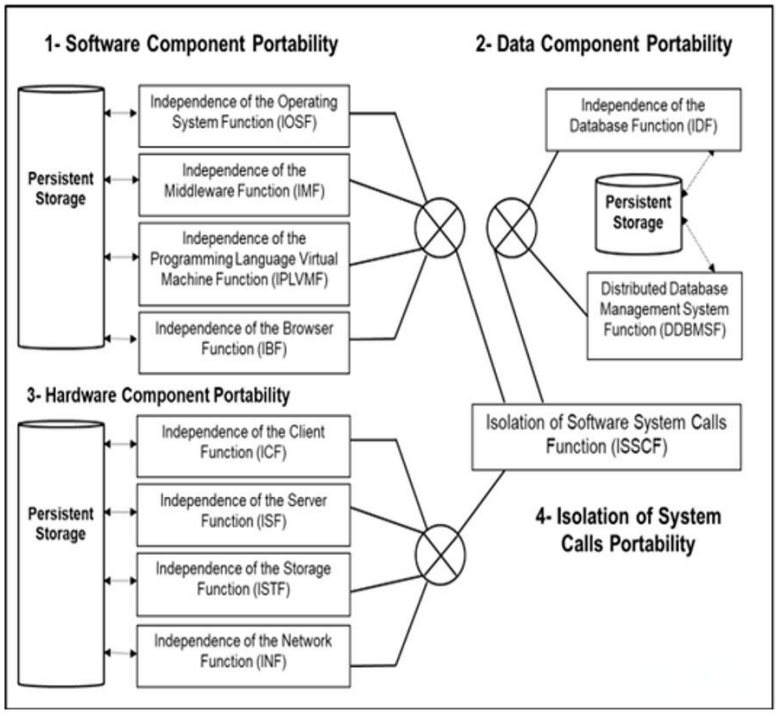


Figure 1: Conceptual view of portability requirements allocated to software [6].

3 CASE STUDY CONTEXT AND UPGRADE CONVERSION ACTIVITIES.

3.1 SimSE environment: an educational games generator and its set of generated software games.

SimSE is an educational game environment for teaching software engineering processes. It includes a model builder that allows instructors to define game models, a code generator that transforms those models into executable games, the generated simulation environment itself, and explanatory tools used for analysis and learning support. Figure 2 shows the architecture and the relationship between the game generator and the generated game applications. Six educational models have been generated using SimSE.

The measured environment is the Application Generator. It includes eight main functional user requirements (FURs) and 65 functional processes, with a total baseline functional size of 287 CFP. The eight FURs are: Model Initialization; Object Type Definition; Initial State Configuration; Action Specification; Rule Definition; Graphical Configuration; Execution Prioritization; and Game Generation. For the full measurement see [5].

Table 1 summarizes the fixed baseline context of this case study. It identifies the starting conditions used to interpret the upgrade functional sizing results later: the number of generated educational models already available at the time of this case study and the measured functional scope of the environment described in Figure 2.

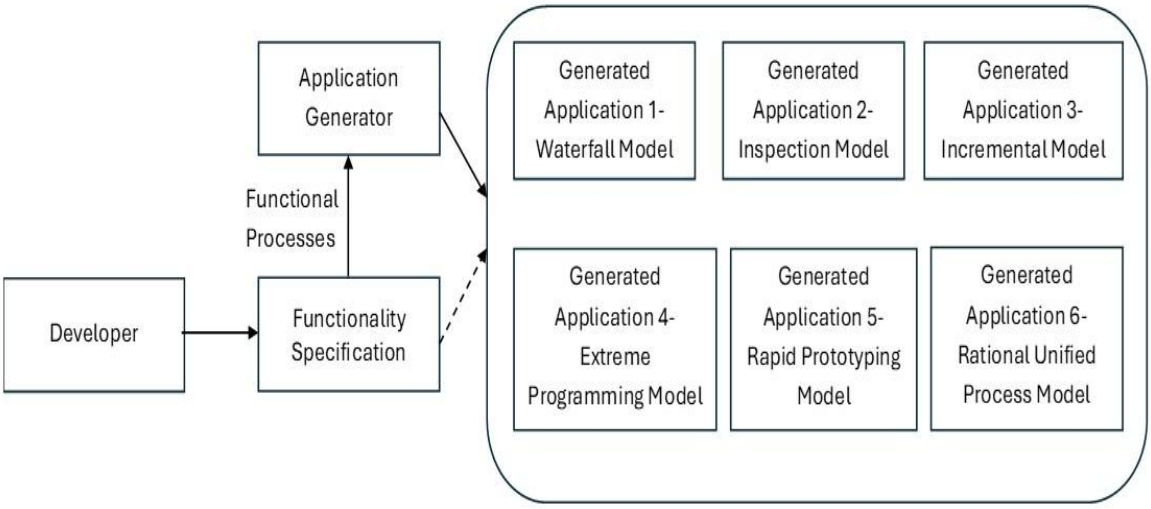


Figure 2: SimSE application generator and generated applications [1].

Table 1. Baseline context of the SimSE case study.

Context element	Value
Generated educational game models already available	6 Games
Main Functional User Requirements (FURs)	8 FUR
Functional processes in the SimSE environment	65
Functional size of the SimSE pre-upgrade environment	287 CFP

3.2 Study design and phases

A SimSE upgrade project from Java 8 to Java 17 had to be planned and initiated. The case study was organized in two phases:

1. The observation and analysis of the initial upgrade planning stage and subsequent technical conversion.
2. The post-upgrade identification of the software functions modified during the upgrade, their COSMIC sizing, and their analysis from a planning perspective.

The separation of the phases was deliberate. Since no reference information was available for planning up-front, it distinguishes the development work to complete the upgrade from the size measurement work and the managerial interpretation of sizing results. Figure 3 summarizes this approach: the two phases, their inputs, the main actions performed, and the outputs.

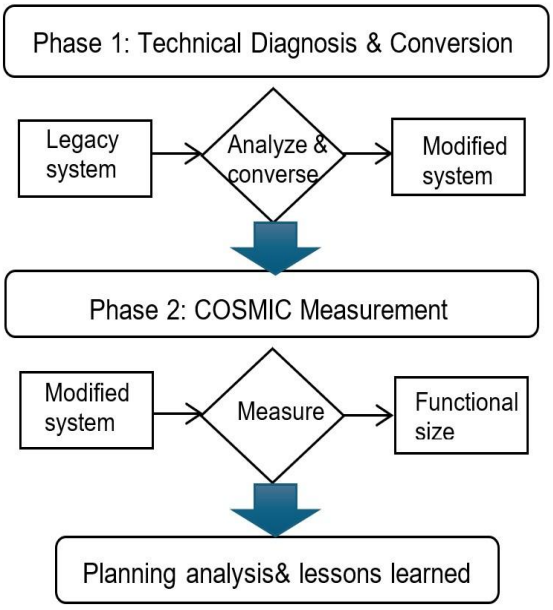


Figure 3: The case study phases [1].

3.3 Technical diagnosis and related upgrade work

The existing SimSE code base was compiled, and the resulting warnings and errors were classified manually. The diagnosis produced 567 warnings, as shown in Table 2. The team selected as upgrade scope the 93 critical warnings associated with wrapper constructors for removal because they directly affected the move to newer Java versions (the other 474 warnings are not part of the upgrade scope).

Table 2. Summary of the warnings and errors identified before the upgrade

Category	Warnings	Share
Critical warnings related to code marked for removal	93	17%
Warnings related to better code writing and unsafe parts	438	77%
Other warnings	36	6%
Total	**567**	**100%**

Therefore, the selected upgrade scope focused on replacing the usages of constructors such as Integer(int), Double(double), and Boolean(boolean) with the recommended alternatives based on valueOf(...) or autoboxing. The code changes impacted both the model builder and the code generator, because the generator logic had to stop emitting code that relied on deprecated constructors.

To resolve the critical warnings and ensure the project's compatibility with future Java versions, the following code changes were implemented:

1. Model Builder Module:
    - Review of the module to identify and replace all occurrences of the deprecated constructors.
    - Replacement of the deprecated constructors Integer(int), Double(double), and Boolean(boolean) with their recommended alternatives, such as Integer.valueOf(int), Double.valueOf(double), and Boolean.valueOf(boolean), respectively.

2. Code Generator Module:
    - Analysis of the generated code strings to identify any usage of the deprecated constructors.
    - Modification of the code generation logic to ensure that the generated code utilizes the recommended alternatives instead of deprecated constructors.
    - Testing was performed to verify that the generated code was free of critical warnings and functions as expected.

During and after the code upgrade, comprehensive testing was performed to ensure that the modified code functions as expected. The following tests were performed.

- System Testing: The entire project was tested as a whole to validate its end-to-end functionality and verify that the code upgrade did not introduce any unintended consequences or regressions.
- Generated Code Testing: A comprehensive testing approach was employed to ensure that the code upgrade did not affect the generated code for existing games. All the generated code for the old games was collected and subjected to thorough testing. The generated code for each game was compiled and executed to verify that no critical errors were introduced due to the code upgrade. The functionality and behavior of the generated code were compared with the expected results to ensure consistency.
- Compatibility Testing: The project was tested on Java versions 17, 16, and 15 to ensure compatibility and identify any version-specific issues.

Table 3 illustrates the type of technical evidence extracted from the git-diff analysis. Although only a sample of five entries is shown in Table 3 for readability, the full review covered 51 code modifications and was used to connect low-level code changes to the functional processes later sized using COSMIC. Two factors explain the gap between the number of warnings (93) and the number of modified files (51):

1. Several warnings per file. The deprecated constructors (new Integer/Double/Boolean) were concentrated in a few files, e.g. ModelFileManipulator.java had 19 occurrences, ActionFileManipulator.java18, RuleFileManipulator.java17. The 93 occurrences were spread over roughly 27 files.

2. The Code Generator module. Of the 51 modified files, 25 are code generators. The compiler only flagged 4 warnings there, because the generators emit Java code as string literals, which javac does not analyze. When I inspected the generated output, the same deprecated constructors appeared in the templates, so I fixed them in the generator sources as well. Those 23 additional files had no compiler warnings but still required changes.

So 93 warnings and 27 Model Builder files, plus 24 generator files corrected by inspection of the generated code, giving 51 modified files in total.

Table 3. Sample of the 51 code modifications identified in the git-diff report

Index ref.	Path	Class	Type of change	Observation
61f34a7..491b71e	src/simse/codegenerator/enginegenerator/EngineGenerator.java	EngineGenerator	Import update	Generator-side adaptation
7b7c596..39d324c	src/simse/codegenerator/explanatorytoolgenerator/ActionGraphGenerator.java	ActionGraph Generator	Constructor replacement	valueOf/autoboxing introduced
843bec6..4876c43	src/simse/codegenerator/explanatorytoolgenerator/ExplanatoryToolGenerator.java	ExplanatoryTool Generator	Constructor replacement	List usage and wrapper fixes
2cd7082..f942cc4	src/simse/codegenerator/explanatorytoolgenerator/MultipleTimelines BrowserGenerator.java	MultipleTimelines BrowserGenerator	Constructor replacement	Integer wrapper fix
c8ebdfc..dffe6e5	src/simse/codegenerator/explanatorytoolgenerator/ObjectGraphGenera-tor.java	ObjectGraph Generator	Import update	Generator-side adaptation

3.4 Post-upgrade COSMIC measurement procedure for the functional processes modified

The **measurement purpose** was to determine the functional size of the software changes made for the implementation of the Java 8-to-17 upgrade and the associated portability objective.

The **measurement scope** was the functionality modified by the upgrade.

The **functional user** was the instructor/model builder.

The granularity was the level of functional processes, and **the layer** was the single application environment represented by SimSE.

Based on the COSMIC measurement rules for sizing functional changes, the functional processes modified in the upgrade must be identified. As this was an upgrading project to Java 17, no new functional processes were added.

After identifying the existing **functional processes**, the associated classes for each functional process were listed. After filtering out the modified classes, the influenced functional processes can be traced back to the following.

Based on the modified classes, **data groups** related could be identified, while after picking out influenced functional processes, for each functional process, the current existing **data movements** could also be identified. Hence, the modified data movements could be sized for this SimSE upgrade. An example of the mapping between a modified class and a functional process is the modified class ModelBuilderGUI (in developer perspective) and functional process FP1-1 (File New) (in measurement perspective), leading to modification of the data groups Object Type and Start State Object.

The measurement procedure combined project documentation, execution of the software, data model, and git-diff report. For each modified class, the analysis traced back the affected functional processes and identified the changed data movements. Table 4 reports the detailed COSMIC CFP sizing of the upgrade modifications to the software environment.

Table 4. Functional size of the modifications to the software environment [1]**.**

No	Functional processes of the game builder	Entry	Read	Write	Exit	CFP Size
1	File new	1	1	–	1	3
2	File open	1	1	–	1	3
3	File close	1	1	1	–	3
4	File save	1	–	1	–	2
5	File save as	1	–	1	–	2
6	Triggers prioritizer update	1	1	1	1	4
7	Destroyers prioritizer update	1	1	1	1	4
8	Continuous rules prioritizer update	1	1	1	1	4
9	Estimate Project trigger rule prioritizer update	1	1	1	1	4
10	End Estimate trigger rule prioritizer update	1	1	1	1	4
11	Analyze trigger rule prioritizer update	1	1	1	1	4
12	Analyze destroyer rule prioritizer update	1	1	1	1	4
13	Generate the model	1	22	22	22	67
14	Define new object type	1	1	1	1	4
15	Add attribute (under code artifact Attributes) (entity = object type)	1	1	1	–	3
16	Edit attribute	1	1	1	–	3
17	Remove attribute (via object type)	1	1	1	–	3
18	Attribute up and down	1	1	–	1	3
19	Define object type (Object, attribute)	–	–	–	–	–
20	Rename object type	1	2	1	–	4
21	Remove object type	1	2	2	–	5
22	List of object type	1	1	–	1	3
23	Start State modify (edit starting value)	1	1	1	–	3
24	Actions Create	1	1	1	–	3
25	Actions remove	1	1	1	–	3
26	Actions rename	1	1	1	–	3
27	Actions move up and down	1	1	–	1	3
28	Add participant	1	1	2	–	4
29	Edit participant	1	1	1	–	3
30	Remove participant	1	1	1	–	3
31	Move up/down participant	1	1	–	1	3
32	View Action options	1	2	2	–	5

(continued on next page)

(Table 4 continued)

No	Functional processes of the game builder	Entry	Read	Write	Exit	CFP Size
33	Edit actions option	1	2	1	–	4
34	Add new effect rule	1	2	2	–	5
35	Add new create objects rule	1	2	2	–	5
36	Add new destroy objects rule	1	2	2	–	5
37	Rule configuration	1	–	1	–	2
38	View rule	1	1	–	1	3
39	Edit rule	1	1	1	–	3
40	Rename rule	1	1	1	–	3
41	Remove rule	1	1	1	–	3
42	View Effects	1	1	–	1	3
43	Button Pad	1	1	1	1	4
44	Add new rule	1	1	1	–	3
45	Annotation	1	–	1	–	2
46	Graphics Choose an Image	1	1	–	1	3
47	Map	1	1	–	1	3
	Total					**220 CFP**

4 UPGRADE SIZING ANALYSIS

4.1 Technical conversion contributions

The technical conversion replaced the selected deprecated constructors in the model builder and modified the code generator so that the newly generated code would no longer emit those constructs. The baseline functionality has not changed as a result of this software upgrade.

The project team also reported that the resulting configuration used a single build file that could serve both Microsoft Windows and macOS/Sonoma. This practical outcome reinforces the interpretation of the project as a portability-driven upgrade rather than a narrow syntax update.

The main contributions of the technical phase are:

- a compilable Java 17 code base, and
- an explicit conversion scope that could then be sized in COSMIC terms.

4.2 Post-conversion COSMIC measurement contribution.

The COSMIC sizing analysis showed that 47 of the 65 functional processes were modified by the upgrade, whereas 18 functional processes were not affected. Most of the changes involved data movements associated with model manipulations, rule handling, and code generation support.

Table 4 provides the detailed functional sizes for the modified functional processes, and Table 5 summarizes the distribution of modified Entries, Reads, Writes, and Exits.

The total size of the modified functionality was 220 CFP. Relative to the 287 CFP baseline, the upgrade represented approximately 77% of the measured baseline functional size. This result should be interpreted carefully; it does not mean that 77% of the software was rewritten from scratch, but it does show that the upgrade affected a large share of the measurable functional

scope. These results indicate that in this empirical case study, a portability-driven upgrade affected a substantial share of the measurable functional scope, even when no new user-visible business functionality was introduced.

Table 5 aggregates the functional sizes of modified data movements across all affected functional processes. The predominance of modified Reads and Writes is consistent with the observed propagation of changes through model manipulation, rule handling, and generator-side access to stored information. Such propagation across dependent artifacts is also consistent with change-impact analysis research and with migration and legacy-modernization studies showing that modernization work often affects supporting artifacts in addition to direct code fixes.

Table 5. Functional size summary of the modified data movements.

Entry modified	Read modified	Write modified	Exit modified	Total modified
46 CFP	70 CFP	63 CFP	41 CFP	220 CFP

4.3 Key Learning Points – collecting a baseline for estimation and planning purposes.

The main implication for project planning is that portability-driven upgrades can represent a substantial and measurable scope.

This case study illustrates the feasibility of quantifying the scope and size of the changed functions with an ISO standard.

The technical portability non-functional requirement is quite large. What might have initially appeared as a small burden is large enough that having treated such an upgrade as a minor technical clean-up would have understated the scope of the work effort involved.

The sizes resulting from this case study can contribute to kick-starting an organization's baseline and can be increased over time to provide sufficient quantitative evidence to support later estimation work. Additional upgrade cases sized with the same procedure can be collected to relate the changed size, upgrade driver, effort, and productivity within a compatible dataset.

5 REFERENCES AND DOCUMENTATION.

[1] IWSM-MENSURA_2026_- Portability-Upgrade COSMIC sizing - v final.pdf.

[2] E. Navarro, "SimSE: A Software Engineering Simulation Environment for Software Process Education" Ph.D. dissertation, Donald Bren School of Information and Computer Sciences, Univ. of California, Irvine, 2006. Available via Dissertation.pdf (tip: retrieve by AI).

[3] Guideline Measurement Manual v5 Part 1 Principles, Defs. & Rules.

[4] Guideline Measurement Manual v5 Part 2 Guidelines.

[5] SimSE Games Generator Sizing Case Study.

[6] A. Abran, K. T. Al-Sarayreh, and J. J. Cuadrado-Gallego, "A standards-based reference framework for system portability requirements," Computer Standards & Interfaces, vol. 35, no. 4, pp. 380–395, 2013, doi: 10.1016/j.csi.2012.11.003.

[7] Abran, A., Software Project Estimation, John Wiley & Sons, 2015.