Family of Technical Strategies for different Software Maintenance Purposes

Pablo Becker, María Fernanda Papa, and Luis Olsina

GIDIS_Web, Facultad de Ingeniería, UNLPam, General Pico, LP, Argentina
[beckerp, pmfer, olsinal]@ing.unlpam.edu.ar

Abstract. Software maintenance is often a long and costly phase of the software system/product life cycle, encompassing activities such as defect correction, adaptation to the changing environment, code/documentation improvement, and enhancement of functional and quality features. Despite its criticality, terminological inconsistencies and variation continue to hinder effective communication between researchers and practitioners. To address this gap, we have developed the Software Maintenance Top-Domain Ontology (SwMaintTDO), which provides an extensible conceptual model for representing key concepts for maintenance processes, roles, resources, and artifacts. This paper proposes a family of technical strategies tailored for different maintenance purposes, unified by the common vocabulary of SwMaintTDO. In this context, a maintenance strategy as a project resource integrates a conceptual base (terminology), as well as the specification of processes and methods necessary to execute its constituent activities. Building upon our previous work on the family of strategies for measurement, evaluation, and testing, this research focuses on the process specifications for two of the four primary maintenance purposes: corrective, perfective, adaptive, and preventive. We place special emphasis on the process specifications of corrective and perfective strategies, detailing how the SwMaintTDO facilitates consistency and knowledge reuse across different maintenance scenarios. By leveraging this common ontological backbone, these strategies provide a systematic approach to handling software correction and evolution, ensuring that maintenance processes are well-defined, repeatable, and aligned with specific engineering purposes.

Keywords: Software Maintenance Ontology, Maintenance Purposes, Maintenance Strategies, Corrective Strategy Process, Perfective Strategy Process.

Familia de Estrategias Técnicas para diferentes Propósitos de Mantenimiento de Software

Resumen. El mantenimiento de software suele ser una fase larga y costosa del ciclo de vida del sistema/producto de software, que abarca actividades como la corrección de defectos, la adaptación a un entorno cambiante, la mejora del código/documentación y el perfeccionamiento de aspectos funcionales y de calidad.

A pesar de su criticidad, las inconsistencias y la variación terminológica continúan dificultando la comunicación efectiva entre investigadores y profesionales. Para abordar esta brecha, hemos desarrollado la Ontología de Dominio Superior de Mantenimiento de Software (SwMaintTDO), que proporciona un modelo conceptual extensible para representar conceptos clave en procesos, roles, recursos y artefactos de mantenimiento. Este artículo propone una familia de estrategias técnicas adaptadas a diferentes propósitos de mantenimiento, unificadas por el vocabulario común de SwMaintTDO. En este contexto, una estrategia de mantenimiento como recurso de proyecto integra una base conceptual, así como la especificación de los procesos y métodos necesarios para ejecutar sus actividades constitutivas. Basándonos en nuestro trabajo previo sobre la familia de estrategias para medición, evaluación y pruebas, esta investigación se centra en las especificaciones de proceso para dos de los cuatro propósitos principales de mantenimiento: correctivo, perfectivo, adaptativo y preventivo. Ponemos especial énfasis en las especificaciones de proceso de las estrategias correctivas y perfectivas, detallando cómo la SwMaintTDO facilita la consistencia y la reutilización de conocimiento a través de diferentes escenarios de mantenimiento. Al aprovechar esta estructura ontológica común, estas estrategias proporcionan un enfoque sistemático para gestionar la corrección y evolución del software, asegurando que los procesos de mantenimiento estén bien definidos, sean repetibles y estén alineados con propósitos específicos de ingeniería.

Palabras clave: Mantenimiento de Software, Propósitos de Mantenimiento, Estrategias de Mantenimiento, Procesos de Estrategias Correctiva y Perfectiva.

1 Introduction

The longevity and evolution of modern software systems depend heavily on the effectiveness of their maintenance processes. As systems continuously adapt to changing requirements and increasingly dynamic technical environments, software maintenance often becomes the most resource-intensive phase of the lifecycle, consuming up to 90% of the total cost (Cozzetti et al., 2005; Verdugo et al., 2024). This effort encompasses a wide range of purposes—corrective, perfective, adaptive, and preventive—each involving different activities, roles, and artifacts.

Despite its critical importance, the maintenance domain suffers from persistent conceptual inconsistencies and terminological heterogeneity. For example, the term “Maintenance Request” is used by Ruiz et al. (2004), whereas Cozzetti et al. (2005) employ “Modification Request,” and Kitchenham et al. (1999) refer to it as a “Maintenance Event.” Furthermore, these terms are often used interchangeably despite having distinct semantic implications. A “Modification Request” is a general term for any proposed change during development or maintenance, while a “Maintenance Request” is specifically tied to the support of an already released system. These variations are not merely syntactic; they reflect deeper semantic inconsistencies that persist even within recognized standards such as ISO/IEC/IEEE 14764 (International Organization for Standardization [ISO/IEC/IEEE], 2022).

To address this issue, we have developed the Software Maintenance Top-Domain

Ontology (SwMaintTDO) (Olsina et al., 2025c), which provides a formal and extensible conceptual model for representing key maintenance concepts, including processes, roles, resources, and artifacts. This ontology establishes a shared vocabulary that enables semantic consistency across different maintenance strategies.

Our previous research has shown that a robust engineering (technical) strategy can be defined through the integration of three fundamental capabilities (Olsina et al., 2017): (i) a domain vocabulary, (ii) a process specification, and (iii) a set of method specifications. Together, these elements define what activities must be performed, how they should be executed, and under which common terminology. Process specifications describe the set of activities, tasks, roles, and artifacts required to achieve a given purpose, and may consider different perspectives such as functional, informational, behavioral, and organizational (Curtis et al., 1992). Method specifications define how these activities are performed in practice, while vocabularies establish the domain terms and relationships needed to ensure consistency across process and method definitions. The use of a shared vocabulary is essential to reduce ambiguity and improve clarity in specifications.

Based on this triad, we have previously developed families of strategies (Olsina et al., 2017; Tebes et al., 2021) for different technical purposes in the domains of software measurement, evaluation, and testing. In this work, we extend this approach to the software maintenance domain by proposing a comprehensive family of maintenance strategies.

Although the proposed family addresses the four classical maintenance purposes, this paper focuses on the process specifications for corrective and perfective strategies due to space constraints. To ensure consistency and robustness, these processes are grounded in SwMaintTDO as a common terminological reference.

To achieve formalization, we use the Software & Systems Process Engineering Metamodel (SPEM) (Object Management Group [OMG], 2008) for the specification of these strategy processes. Unlike existing maintenance process models, this proposal allows for a clearer and more standardized definition of roles, work products, and activity sequences that remain consistent across different maintenance purposes. Furthermore, we illustrate specific maintenance scenarios to demonstrate the practical applicability of each strategy's process, ensuring that the theoretical framework translates effectively into real-world engineering tasks.

The main contributions of this paper are: i) A family of software maintenance strategies aligned with the four classical maintenance purposes; ii) The integration of these strategies using SwMaintTDO to ensure semantic consistency; iii) The formal specification of maintenance processes using SPEM; and iv) The illustration of the strategies through realistic maintenance application scenarios.

The remainder of this paper is structured as follows: Section 2 presents the conceptual framework, detailing SwMaintTDO and its role as the backbone of our proposal. Section 3 describes and illustrates the family of maintenance strategies, focusing on the process specifications for corrective and perfective maintenance. Section 4 discusses related work, and finally, Section 5 draws the conclusions and future work.

2 Conceptual Framework

This section presents the conceptual foundation of the proposed family of maintenance strategies. As discussed in the introduction, a robust engineering strategy requires a well-defined vocabulary to ensure consistency across process and method specifications. In our approach, this role is fulfilled by SwMaintTDO.

SwMaintTDO is not an isolated artifact; it is part of a broader multi-level ontological architecture that enables modularity, reuse, and semantic consistency across domains. In this section, we first situate SwMaintTDO within this architecture and then describe its key concepts and relationships that are directly relevant to the definition of maintenance strategies.

2.1 Location of SwMaintTDO within the Ontological Architecture

SwMaintTDO is developed within a multi-layered ontological framework called FCD-OntoArch (*Foundational, Core, and Domain Ontological Architecture for sciences*) (Olsina, 2023). This architecture organizes ontologies according to different levels of generality, enabling a clear separation of concerns and systematic knowledge reuse. Fig. 1 illustrates these five levels of generality/specificity on the left side, distinguishing between domain-independent and domain-dependent layers. In particular, the architecture distinguishes between: Foundational level, which defines the most general concepts; Core level, which captures domain-independent but engineering-relevant concepts (e.g., processes, projects, situations); and Domain level, which includes domain-specific ontologies. This last level is further divided into Top-domain ontologies, which define high-level domain concepts and relationships; and Low-domain ontologies, which provide more specialized and detailed concepts.

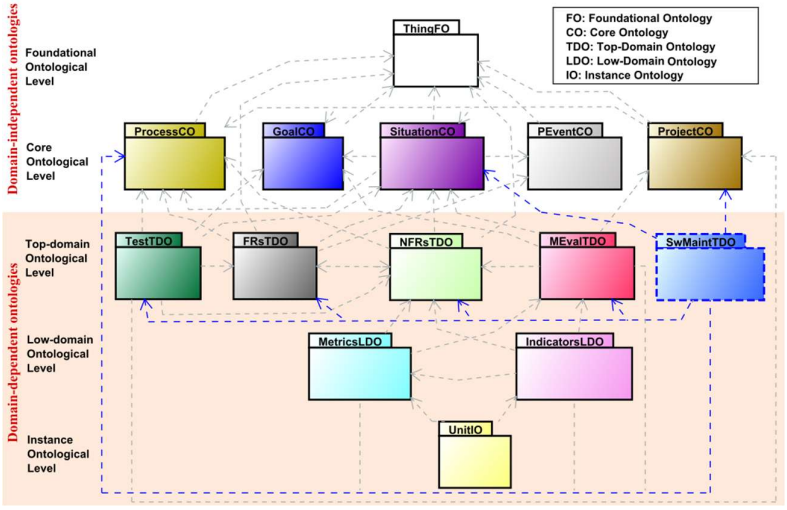

Fig. 1. Allocating the SwMaintTDO component in the context of the FCD-OntoArch.

At the foundational level, ThingFO (Olsina, 2023) serves as the single unified root, representing the world through particular Things, universal Things, and Assertions. Consequently, all subsequent lower-level ontologies inherit from and are semantically enriched (either directly or indirectly) by these foundational concepts.

Beneath the foundational ontology, FCD-OntoArch houses several core ontologies: ProcessCO (Becker et al., 2022), GoalCO, SituationCO, PEventCO (Particular Event), and ProjectCO (Olsina et al., 2024). Moving to the domain-dependent layers, the top-domain level includes TestTDO (Becker et al., 2025), FRsTDO (Functional Requirements) (Olsina et al., 2025a), NFRsTDO (Non-Functional Requirements) (Olsina et al., 2025d), MEvalTDO (Measurement and Evaluation) (Olsina et al., 2025b), and our recently proposed ontology called SwMaintTDO (Olsina et al., 2025c), while the low-domain level contains MetricsLDO and IndicatorsLDO.

At the top-domain level, SwMaintTDO defines terms specific to the software maintenance domain. It extends and specializes concepts from core ontologies (such as ProcessCO and ProjectCO), while also reusing elements from other domain ontologies (e.g., TestTDO, FRsTDO, and NFRsTDO).

As depicted by the blue dashed lines in Fig. 1, SwMaintTDO establishes dependency links with both higher-level and peer components. These relationships ensure that its specialized terms preserve semantic consistency within the broader architecture. To exemplify this semantic enrichment, see in Fig. 2 the term Maintenance Request. It is stereotyped as an <<Artifact>>, thus inheriting the Artifact semantics defined in ProcessCO. If we examine the ProcessCO ontology, we observe that Artifact is classified as a type of Work Product, which is, in turn, semantically enriched by the foundational term Thing from ThingFO.

Ultimately, this multilayered architectural approach ensures a rigorous separation of concerns by delineating clear ontological levels and properly locating conceptual components. This promotes modularity, extensibility, and systematic reuse of ontological knowledge across engineering domains.

2.2 SwMaintTDO overview

An excerpt from the SwMaintTDO conceptualization is shown in Fig. 2. The complete conceptualization and its detailed definitions can be found in (Olsina et al., 2025c). In the following paragraphs, we describe key elements of the SwMaintTDO using the following convention: top-domain terms are Capitalized, properties are *italicized*, and relationships are underlined. Furthermore, the UML diagram in Fig. 2 employs a color-coding scheme to enhance visual comprehension. Light green denotes an Assertion term. Conversely, three distinct colors are used to identify Thing (particular entity) terms: light red indicates the semantics of a Process or Activity; light orange signifies an Artifact or Outcome; and light yellow represents a Resource or an Organizational Entity, among others. Regarding the links between concepts, red lines denote taxonomic relationships, whereas black lines represent non-taxonomic relationships.

In Fig. 2, we can observe that the Maintained System Stakeholder is the one who triggers issue/need to the Software Maintenance Organization. This stakeholder encompasses any internal or external individual or group possessing a vested interest in the

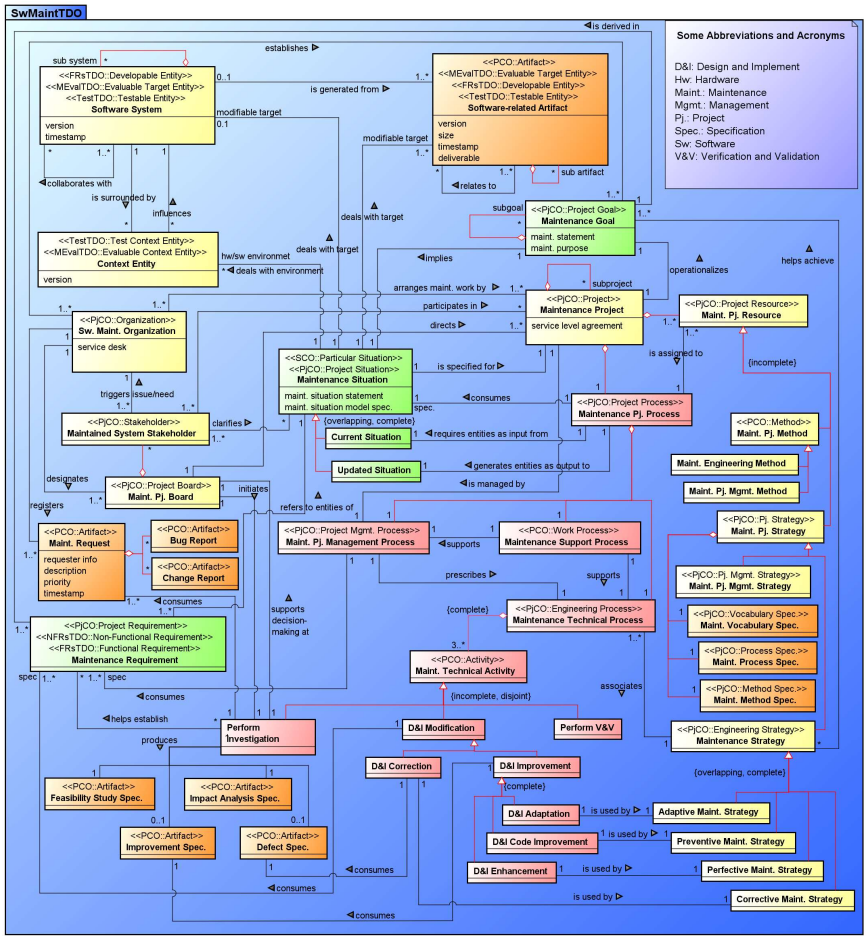

Fig. 2. Excerpt of SwMaintTDO v1.0: Top-Domain Ontology for Software Maintenance.

system's continued operation and upkeep. Upon receiving this input, the organization's *service desk* registers a Maintenance Request, which typically takes the form of a Bug Report or a Change Report document. Note that these latter three terms have the semantics of <<Artifact>>, a core concept defined in ProcessCO. While a Bug Report "*formally documents a software defect or system failure to provide a clear and reproducible account of the issue supporting its resolution*", a Change Report "*formally documents a required change to a Software System or related artifacts —such as an adaptation, enhancement, or code/documentation change— to support system evolution and long-term maintainability*" (Olsina et al., 2025c).

Following a typical workflow, the Maintenance Project Board, which is responsible for ensuring that system evolution aligns with business goals and long-term viability, initiates the activity Perform Investigation. The output of this activity assists the board

in making an informed decision to either authorize or halt the Maintenance Project. For example, the Feasibility Study Specification "*analyzes the feasibility of proposed solutions to a reported bug or proposed change, evaluating their technical, operational, and economic implications.*" and the Impact Analysis Specification "*identifies and assesses the effects of a proposed improvement or defect correction, including the scope of impact, affected entities, dependencies, and potential risks.*"

A Maintenance Project operationalizes a Maintenance Goal. The goal specification includes a *maintenance statement* and a *maintenance purpose* (e.g., 'adapt', 'prevent', 'enhance', and 'correct'). Moreover, a Maintenance Goal implies defining a Maintenance Situation, distinguishing two types: Current Situation (i.e., the current state of affairs and versioned baseline of the Software System and its related target and context entities) and Updated Situation (i.e., the new operational and versioned baseline of the Software System and its related target and context entities).

To support the realization of a Maintenance Project, it has assigned one or multiple Maintenance Project Resources, such as Maintenance Engineering Methods and a Maintenance Strategy. A Maintenance Engineering Method is a type of <<Method>> (a term from ProcessCO) "*available for use in technical activities, defining the particular way of performing the steps specified in its sub-activities and tasks*". A Maintenance Strategy (a type of <<Engineering Strategy>>) acts as a crucial resource within a Maintenance Project since it integrates the Maintenance Process Specification, the Maintenance Method Specification, and the Maintenance Vocabulary Specification, to help achieve the *maintenance purpose* of a Maintenance Goal.

Additionally, a Maintenance Project mandatorily has a Maintenance Project Process. This overarching process is composed of three distinct work processes: the Maintenance Project Management Process, the Maintenance Support Process, and the Maintenance Technical Process. In turn, the latter, for instance, is composed of at least three types of Maintenance Technical Activities: Perform Investigation, Design and Implement Modification, and Perform Verification and Validation. Particularly, Design and Implement Modification is the technical activity "*carried out to produce the required changes (corrections, adaptations, enhancements, or source code improvements) to the Software System, its Software-related Artifacts, or a Context Entity in the Current Situation.*" Note that this activity specializes in Design and Implement Correction, Design and Implement Adaptation, Design and Implement Code Improvement, or Design and Implement Enhancement, depending on the *maintenance purpose* to be achieved.

Finally, it is important to note that Fig. 2 demonstrates the mapping between specific activities and strategies: the Design and Implement Correction activity is used by a Corrective Maintenance Strategy; the Design and Implement Adaptation activity is used by an Adaptive Maintenance Strategy; the Design and Implement Code Improvement activity is used by a Preventive Maintenance Strategy; and the Design and Implement Enhancement activity is used by a Perfective Maintenance Strategy.

By establishing this ontological foundation, we have defined the first pillar of our technical strategies: a common reference vocabulary. This formal conceptualization ensures that all terms used in the subsequent sections maintain semantic consistency across different maintenance strategies and scenarios.

3 Family of Maintenance Strategies

The proposed family of maintenance strategies provides a systematic and unified approach to address the four classical maintenance goal purposes: corrective, adaptive, preventive, and perfective. Each strategy is defined as a specialized engineering strategy that integrates a process specification, a method specification, and a shared vocabulary grounded in SwMaintTDO.

A key feature of this family is that all strategies follow a common structural logic, while differing in the specific type of modification activity required to achieve the maintenance purpose. Although the family covers all four maintenance purposes, in this paper, we focus on the process specifications for corrective and perfective strategies. The remaining strategies (adaptive and preventive) follow the same structure and ontological mapping. Furthermore, it is worth noting that all strategy processes are specified using SPEM and are described from the functional and behavioral perspectives, enabling a clear representation of activities, work products, and control flow.

3.1 Corrective Maintenance Strategy

The Corrective Maintenance Strategy is applied when the *purpose* of the Maintenance Goal is to '*correct'* defects that cause system failures or deviations from expected behavior. This strategy provides a structured process to identify, analyze, and eliminate the root causes of defects.

As shown in Fig. 3, the Corrective Maintenance Process is specified as a workflow of three maintenance technical activities: Perform Investigation (A1), Design and Implement Correction (A2), and Perform V&V (A3).

The process begins with Perform Investigation (A1), which focuses on understanding the failure and its origins, as well as providing the technical and economic rationale needed for the Maintenance Project Board to authorize the intervention. Its first sub-activity, Analyze Bug Report and Root Causes, consumes the Maintenance Request, specifically the Bug Report artifact. This document typically encompasses problem descriptions, reproduction steps, execution context, behavior discrepancies, and severity levels, often supported by logs or screenshots. The primary output of this analysis is the Defect Specification artifact, which identifies affected entities in the Current Situation and defines the modifications required in the Software-related Artifacts to restore expected behavior, including defect locations and acceptance criteria. During this stage, additional information may be required to facilitate the analysis; therefore, the Request additional issue information sub-activity should be performed. Once the defect is specified, the sequence continues with Define Preliminary Corrective Solution, which consumes the Maintenance Requirements (i.e., project requirements, as well as functional and non-functional requirements). Then, the Perform Impact Analysis and Perform Feasibility Study are performed. These activities produce an Impact Analysis Specification and a Feasibility Study Specification, respectively, providing the Maintenance Project Board with the necessary data to make an informed decision to either authorize or cancel the project's initiation. For the sake of simplicity, the fine-grained interactions be-

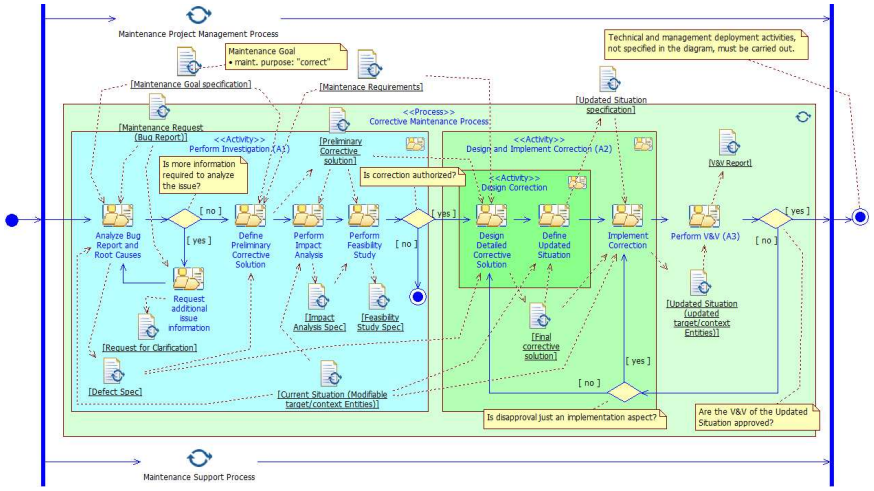

Fig. 3. Maintenance Process Specification for the Corrective Maintenance Strategy of the family considering the behavioral and functional perspectives.

tween these technical activities and management processes are not shown in the diagram.

Upon authorization, the Design and Implement Correction (A2) activity is performed. It is split into two main sub-activities: Design Correction and Implement Correction. The former involves, on the one hand, performing Design Detailed Corrective Solution, which builds upon the Preliminary Corrective Solution and takes the Defect Specification artifact into account. As a result, the Final Corrective Solution is produced. On the other hand, Design Correction also involves producing the Updated Situation Specification through the Define Updated Situation, using the Current Situation as a reference point alongside the Final Corrective Solution.

Subsequently, the Implement Correction sub-activity translates these design specifications into technical reality. By using the Final Corrective Solution as a roadmap, the maintenance engineer modifies the affected entities (such as source code, database schemas, or documentation) of the Current Situation. This execution results in the generation of the actual software artifacts that constitute the Updated Situation, which are then ready to be submitted for formal verification.

Finally, the Perform V&V (A3) activity evaluates the entities of the Updated Situation to yield a V&V Report, which is critical for managers to determine if the modifications are approved for deployment. The process follows a specific decision logic: if the Updated Situation is fully approved, the process concludes and moves toward deployment; if the rejection is due solely to implementation defects, the workflow reverts to Implement Correction; however, if design flaws are identified, the process returns to the Design Detailed Corrective Solution to redefine the fix.

Application Scenario: Correcting a Critical Defect in a university's Academic Records System. The scenario starts when a professor, acting as the Maintained System

Stakeholder, reports a "Null Pointer Exception" occurring whenever the system attempts to calculate the final average for students with more than 20 graded subjects. This issue is registered as a Bug Report (#404) within a Maintenance Request. For this scenario, the Maintenance Goal *statement* is "Correct the Null Pointer Exception that occurs when processing student records, ensuring the reliability of final average calculations for the Academic Records System." Therefore, the Goal purpose is 'correct' and the process to follow is that for the Corrective Maintenance Strategy (Fig. 3).

During the Perform Investigation (A1) activity, the Maintenance Engineer analyzes the Current Situation (Source Code v2.4.1) and identifies that the failure stems from a hardcoded array size in the AverageCalculator class. To support the investigation, the engineer requests additional execution logs from the stakeholder to confirm memory constraints. This stage ends in the production of a Defect Specification, which describes the fault and its location. Then, a Preliminary Corrective Solution is produced, defining the necessary modifications that must be made to the code, as well as the unit tests.

Subsequently, the engineer performs an Impact Analysis, identifying that the change from a static array to a dynamic list affects three reporting modules, and a Feasibility Study confirming that the fix can be implemented within four hours with minimal risk. These specifications are submitted to the Maintenance Project Board, which, after reviewing the technical and economic implications, provides the authorization to proceed.

Upon authorization, the Design and Implement Correction (A2) activity begins. The Design Detailed Corrective Solution is enacted, specifying the replacement of the fixed array with an ArrayList<Subject>, and the Define Updated Situation, establishing version 2.4.2 as the new baseline under configuration management. During the Implement Correction activity, the Java source code is modified, and the internal documentation is updated, effectively generating the entities for the Updated Situation.

Finally, the Perform V&V (A3) activity is executed. The engineer runs a regression test suite along with a specific test case for a student profile with 50 subjects. The resulting V&V Report indicates that while the primary defect is resolved, a minor UI alignment issue exists in the final grade report. Following the process logic, because the rejection is due solely to an implementation detail, the workflow reverts to the Implement Correction activity. Once the alignment is fixed and a new V&V Report is issued with an "Approved" status, the corrective process concludes, and the Updated Situation (v2.4.2) is formally accepted for deployment.

3.2 Perfective Maintenance Strategy

The Perfective Maintenance Strategy is applied when the *maintenance purpose* is defined as 'enhance' the system by improving its functionality or quality attributes. While the Corrective Strategy reacts to failures, this strategy focuses on improving entities of the Current Situation to meet evolving business needs. The Perfective Maintenance Process shares a similar high-level structure with the corrective one, being composed of three main technical activities: Perform Investigation (A1), Design and Implement Enhancement (A2), and Perform V&V (A3).

As shown in Fig. 4, the process initiates in Perform Investigation (A1) with the sub-

activity Analyze the proposed Change, which consumes a Change Request and produces an Improvement Specification artifact.

A Change Report artifact typically includes a brief description of the proposed change, its rationale, the expected benefits, and other relevant information to support further investigation. An Improvement Specification artifact defines the desired enhancements and typically includes the type and scope of the required modification, the rationale for the change, the affected entities or components, the proposed solution or additions, and criteria for acceptance, as well as references to related Maintenance Requests. Subsequently, the process follows a sequential flow: Define Maintenance Requirements, which updates the Maintenance Requirements list with new functional or non-functional requirements; Perform Impact Analysis; and Perform Feasibility Study. Similar to the corrective workflow, these sub-activities provide the Maintenance Project Board with the necessary data to authorize or cancel the project.

Upon authorization, the Design and Implement Enhancement (A2) activity is performed. The internal logic of this activity mirrors the corrective strategy but shifts its focus from "fixing" to "evolving". The Design Enhancement sub-activity involves the Design Detailed Enhancement (building upon the Improvement Specification) and the Define Updated Situation. Then, the Implement Enhancement activity translates these designs into actual code and documentation changes. Finally, Perform V&V (A3) evaluates the Updated Situation. The decision logic for re-work remains identical to the corrective process: implementation flaws trigger a return to Implement Enhancement, while design discrepancies require reverting to Design Detailed Enhancement.

Application Scenario: Enhancing the Academic Records System. This scenario involves the University Research Department, acting as the Maintained System Stakeholder, which requests a new feature to automatically export student transcripts into a specific XML format for national accreditation. This request is documented as a

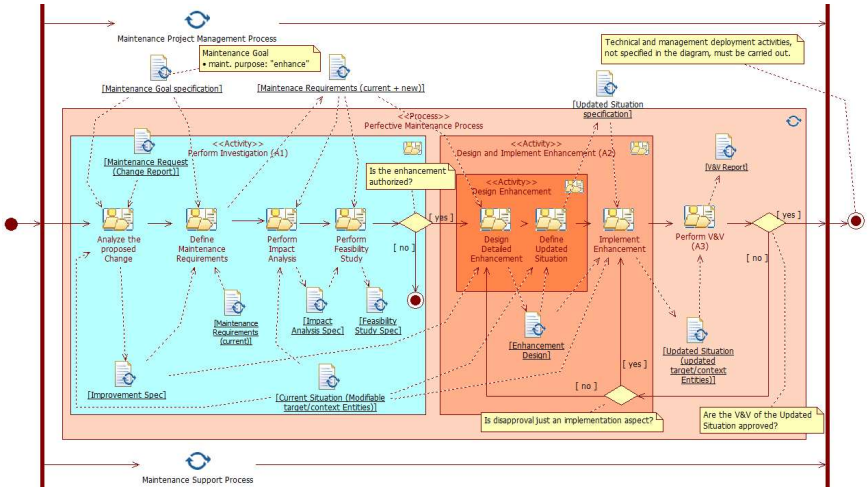

Fig. 4. Maintenance Process Specification for the Perfective Maintenance Strategy of the family considering the behavioral and functional perspectives.

Change Request. For this scenario, the Maintenance Goal *statement* is: "Enhance the Academic Records System by implementing an automated XML export feature for student transcripts, ensuring compliance with national accreditation standards." Therefore, the Goal purpose is 'enhance' and the process to follow is that for the Perfective Maintenance Strategy (Fig. 4).

During Perform Investigation (A1), the engineer analyzes the proposed change and generates an Improvement Specification detailing the XML schema requirements. Then, in the Define Maintenance Requirements, the Maintenance Requirements are updated to include this new export capability. Following an Impact Analysis that reveals that the existing reporting module can be extended without affecting core grade calculations, and a Feasibility Study confirms the availability of XML libraries, the Maintenance Project Board authorizes the project.

In the Design and Implement Enhancement (A2), the Design Detailed Enhancement is performed for creating the mapping logic between the database and the XML tags. The Updated Situation is defined as version 2.5.0. During the Implement Enhancement, the new export class is coded. Finally, in Perform V&V (A3), the QA testers verify that the generated XML files comply with the national standards. A minor schema error is found; since this is an implementation issue, the maintenance technician returns to Implement Enhancement to fix the XML tag nesting. Once the V&V Report confirms the output is correct, the Updated Situation (v2.5.0) is approved for deployment.

4 Related Work

The literature presents various maintenance process models, ranging from traditional and agile to educational approaches. For instance, Hardt (2024) proposed Co-Op, an agile maintenance process for classrooms; however, its process model lacks explicit input/output and does not provide specific activities based on maintenance purposes. Similarly, the models documented in the reviews by Masrat et al. (2021) and Yongchang et al. (2011) (not discussed here for space reasons) generally omit input/output and fail to differentiate activities according to the maintenance goal purpose.

The international standard ISO 14764 (ISO/IEC/IEEE, 2022) provides a high-level process framework. While it identifies activities, tasks, steps, and outcomes, it fails to map specific outcomes to their respective activities or offer specialized workflows for different maintenance purposes. The standard states that "*although not all steps may be necessary for all maintenance activities, especially when activities are focused on Modification request/Problem reports*," it does not specify exactly which ones are not mandatory. Furthermore, when describing the Perform Maintenance activity, this standard states that this activity consists of several tasks, including: 4) Implement the procedures for correction of flaws (defects) and errors, or for replacement or upgrade of system elements; 5) Perform preventive maintenance; and 6) Identify when adaptive or perfective maintenance is required. However, these activities should be executed selectively, depending on the specific maintenance purpose intended to be achieved.

Additionally, most existing process models lack a formal ontological foundation and do not utilize standard notations like SPEM. Finally, it is important to note that the

present work is similar to (Olsina et al., 2017) and (Tebes et al., 2021); however, while the focus here is on a family of strategies for software maintenance, those works model processes for strategies in the domains of measurement, evaluation, and testing.

5 Concluding Remarks

In this paper, we have presented a family of software maintenance technical strategies, specifically focusing on the process specifications for corrective and perfective goal purposes. The primary contribution of this research is the use of the SwMaintTDO ontology as a common terminological backbone. This ontological approach effectively mitigates the long-standing problem of terminological heterogeneity and conceptual inconsistency in the maintenance domain, providing a unified vocabulary that ensures semantic alignment across different maintenance scenarios.

By formalizing these strategies using SPEM, we have demonstrated that maintenance processes can be made repeatable, well-structured, and systematically aligned with specific engineering maintenance goal purposes, which act as the logic selector for the appropriate technical strategy. Furthermore, the application scenarios in the "Academic Records System" illustrate how the theoretical framework of SwMaintTDO and the strategy specifications translate into practical, traceable, and manageable engineering activities and artifacts.

Future work will follow two main directions. On the one hand, we will complete the family's documentation by detailing in another paper the process specifications for the remaining two strategies: adaptive and preventive maintenance.

On the other hand, we will develop the corrective, perfective, adaptive, and preventive low-domain ontologies that contain more specific terms and relationships for the family of Maintenance Strategies, some of which are used in Figs. 3 and 4. Finally, we aim to validate the complete family of strategies through case studies in diverse industrial environments to assess its impact on maintenance process efficiency and communication quality among stakeholders.

References

Becker, P., Papa, M.F., Tebes, G., and Olsina, L. (2022). Discussing the Applicability of a Process Core Ontology and Aspects of its Internal Quality. Software Quality Journal, Springer, 30:(4), pp. 1003–1038, https://doi.org/10.1007/s11219-022-09592-3

Becker, P., Tebes, G., Papa, M.F., and Olsina, L. (2025). Enhancing a Software Testing Ontology. ASSE'25, Argentine Symposium on Software Engineering, 11(2), 54th JAIIO, pp. 1–14, https://revistas.unlp.edu.ar/JAIIO/article/view/19521

Cozzetti, B., de Souza, S., Anquetil, N., and Marçal de Oliveira, K. (2005). A study of the documentation essential to software maintenance. 23rd annual international conference on Design of communication: documenting & designing for pervasive information (SIGDOC '05). Association for Computing Machinery, New York, USA, pp. 68–75, https://doi.org/10.1145/1085313.1085331

Curtis, B., Kellner, M., and Over, J. (1992). Process Modelling. Communications of the ACM, 35:(9), pp. 75–90.

Hardt, R. (2024). A software maintenance-focused process and supporting toolset for academic environments. 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, SA, Australia, pp. 360–370, doi: 10.1109/ICSME46990.2020.00042

International Organization for Standardization. (2022). Software engineering — Software life cycle processes — Maintenance (ISO/IEC/IEEE 14764:2022).

Kitchenham, B.A., Travassos, G.H., von Mayrhauser, A., Niessink, F., Schneidewind, N.F., Singer, J., and Yang, H. (1999). Towards an ontology of software maintenance. Journal of Software Maintenance: Research and Practice, 11(6), pp. 365–389.

Masrat, A., Makki, M. A., and Gawde, H. (2021). Software Maintenance Models and Processes: An Overview. SSRN. https://doi.org/10.2139/ssrn.3838444

Object Management Group. (2008). Software & systems process engineering meta-model specification (Version 2.0). https://www.omg.org/spec/SPEM/2.0/

Olsina, L. (2023). The Foundational Ontology ThingFO: Architectural Aspects, Concepts, and Applicability. In: Fred, A. et al. (eds.) Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2021, Communications in Computer and Information Science, vol 1718, Springer, Chapter 5, pp. 73–99.

Olsina, L. and Becker, P. (2017). Family of Strategies for different Evaluation Purposes. In: 20th Ibero-American Conference on Software Engineering (CIbSE'17) held in the framework of ICSE, CABA, Argentina, Published by Curran Associates 2017, pp. 221–234.

Olsina, L., Becker, P., and Papa, M.F. (2024). The Project Management Ontology called ProjectCO: Architectural Aspects, Concepts, and Usefulness. In: CIbSE 2024, Curitiba, Brazil, SBC_OpenLib, pp. 61–75, https://doi.org/10.5753/cibse.2024.28439

Olsina, L. and Becker, P. (2025a). FRsTDO v1.2's Terms, Properties, and Relationships -- A Top-Domain Functional Requirements Ontology. Preprint in Research Gate, pp. 1–6, https://doi.org/10.13140/RG.2.2.11781.00486

Olsina, L., Becker, P., and Papa, M.F. (2025b). An Ontological Conceptualization of Measurement and Evaluation Activities and Methods and their Applicability in Evaluation Strategies. 51st Latin American Computer Conference (CLEI'25), Valparaiso, Chile, IEEE Xplore, pp. 1–10, https://doi.org/10.1109/CLEI67442.2025.11420334

Olsina, L., Becker, P., and Papa, M.F. (2025c). SwMaintTDO v1.0's Concepts, and Relationships – A Top-Domain Ontology for Software Maintenance. Preprint in Research Gate, pp. 1–48, https://doi.org/10.13140/RG.2.2.26543.19367

Olsina, L., Becker, P., Papa, M.F., and Rossi, G. (2025d). Ontological Concepts of Non-Functional Requirements Applied to Particular Situations in Measurement and Evaluation Projects. CLEI Electronic Journal, vol. 28, issue 3, paper 7, http://dx.doi.org/10.19153/cleiej.28.3.7

Ruiz, F., Vizcaíno, A., Piattini, M., and García, F. (2004). An Ontology for the Management of Software Maintenance Projects. International Journal of Software Engineering and Knowledge Engineering, 14:(03), pp. 323–349. https://doi.org/10.1142/s0218194004001646

Tebes, G., Peppino, D., Becker, P., and Olsina, L. (2021). A Situation-aware Scenario-based Testing Strategy. SCCC 2021 - 40th International Conference of the Chilean Computer Science Society, In: IEEE Xplore, La Serena, Chile, pp. 1–8.

Verdugo, J., Oviedo, J., Rodriguez, M., and Piattini, M. (2024). Connecting Research and Practice for Software Product Quality Evaluation and Certification: A Software Laboratory's 25-Year Journey. IEEE Software, vol. 41, no. 03, pp. 33–40, https://doi.org/10.1109/MS.2024.3357119

Yongchang, R., Tao, X., Zhongjing, L., and Xiaoji, C. (2011). Software Maintenance Process Model and Contrastive Analysis. 2011 International Conference on Information Management, Innovation Management and Industrial Engineering, Shenzhen, China, pp. 169–172, https://doi.org/10.1109/ICIII.2011.324