МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 06.00.00.000 ПЗ

Група ІПм-23-1

Бігун Сергій

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Бігун Сергій Анатолійович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Методологія модель-базованого підходу до аудиту та тестування якості програмного забезпечення

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Бігун С.А.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Шекета Василь Іванович, д.т.н., професор

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Бігуну Сергію Анатолійовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Методологія модель-базованого підходу до аудиту та тестування якості програмного забезпечення ”

керівник проекту (роботи) Шекета Василь Іванович, д.т.н., професор

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних технологій тестування ПЗ

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження та огляд стандартів процесу тестування якості програмного забезпечення
2. Моделі та принципи оцінювання якості програмного забезпечення
3. Методика покращення мета-моделей для процесу оцінювання якості
4. Імплементація методології модель-базованого підходу до тестування якості ПЗ

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Структура Capability Maturity Model Integration (рис. 1.1)
2. Огляд областей процесу в CMMI-DEV (рис. 1.2)
3. Рівні зрілості та області процесу, присутні в TMMi (рис. 1.3)
4. Структура та компоненти, наявні в TMMi (рис. 1.4)
5. Переваги Testing Maturity Model integration (TMMi) (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Дослідження та огляд стандартів процесу тестування якості програмного забезпечення	15.10.2024	виконано
4	Моделі та принципи оцінювання якості програмного забезпечення	08.11.2024	виконано
5	Методика покращення мета-моделей для процесу оцінювання якості	20.11.2024	виконано
6	Імплементація методології модель-базованого підходу до тестування якості ПЗ	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 75 с., 34 рис., 5 табл., 52 джерел.

Тема: Методологія модель-базованого підходу до аудиту та тестування якості програмного забезпечення

Об'єкт дослідження: процеси аудиту та тестування якості програмного забезпечення в контексті застосування стандартів та моделей зрілості.

Мета роботи: розробка модельно-орієнтованого підходу до аудиту та тестування якості програмного забезпечення на основі сучасних стандартів якості та моделей зрілості процесів.

Предмет дослідження: методи та моделі оцінювання якості програмного забезпечення на основі стандартів, а також інструменти для їх реалізації.

Результати дослідження:

В роботі виконано дослідженні мета-моделей для оцінювання якості програмного забезпечення та процесів тестування та визначенні ключових елементів процесів розробки та тестування, які впливають на якість програмного забезпечення, з подальшою їх інтеграцією в мета-моделі.

Висновок

Розроблені мета-моделі та інструменти оцінювання можуть бути впроваджені в промислових проєктах для забезпечення високої якості програмного забезпечення, підвищення його надійності та продуктивності.

МОДЕЛЬНО-ОРІЄНТОВАНИЙ ПІДХІД, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, АУДИТ ЯКОСТІ, СТАНДАРТИ ЯКОСТІ МЕТА-МОДЕЛІ, ОЦІНЮВАННЯ ЯКОСТІ.

ABSTRACT

Master Thesis: 75 pp., 34 fig., 5 tab., 52 sources.

Thesis Subject: Methodology of a model-based approach to auditing and software quality testing

The object of the study: the processes of auditing and testing the quality of software in the context of the application of standards and maturity models.

The purpose of the work: development of a model-oriented approach to auditing and software quality testing based on modern quality standards and process maturity models.

Research subject: methods and models of software quality assessment based on standards, as well as tools for their implementation.

Research results

In the work, researched meta-models for evaluating the quality of software and testing processes were performed, and key elements of the development and testing processes that affect the quality of software were identified, with their further integration into the meta-model.

Conclusion

The developed meta-models and evaluation tools can be implemented in industrial projects to ensure high quality of software, increase its reliability and productivity.

MODEL-ORIENTED APPROACH, SOFTWARE TESTING, QUALITY AUDIT, META-MODEL QUALITY STANDARDS, QUALITY ASSESSMENT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ADAS - Advanced Driver Assistance Systems

ASPICE - Automotive Software Process Capability dEtermination

CMMI - Capability Maturity Model Integration

TMMi - Testing Maturity Model integration

CAPPMM - Common Automotive Process and Product Meta-Model

MDSE - Model Driven Software Engineering

PA - Process Area

ВСТУП

Актуальність теми.

Актуальність теми дослідження полягає в необхідності розробки ефективних методів і моделей для оцінювання та підвищення якості програмного забезпечення, що є критично важливим на сучасному етапі розвитку інформаційних технологій. Зростання складності програмних систем вимагає чіткого визначення стандартів якості та процесів тестування для забезпечення їх надійності та функціональності.

З розвитком кількості автомобілів, оснащених системами ADAS (Advanced Driver Assistance Systems), зростає залежність від складного вбудованого програмного забезпечення в автомобілях. Програмне забезпечення є більш критичним, ніж будь-коли раніше, для продуктивності, безпеки та ефективності.

Через високі вимоги до високоякісного програмного забезпечення з'явилися різні глобальні стандарти програмного забезпечення, які визначають процеси, що слід дотримуватися для розробки та тестування програмного забезпечення, а також стандарти якості програмного продукту. Прикладами цього є ASPICE (Automotive Software Process Capability dEtermination) та CMMI (Capability Maturity Model Integration) для процесів розробки програмного забезпечення та TMMi (Testing Maturity Model integration) для тестування програмного забезпечення. Прикладами стандартів якості програмного продукту є ISO/ IEC 9126 та ISO/ IEC 25010.

Компанії використовують різні процеси для розробки та тестування систем і програмного забезпечення. Ці процеси повинні відповідати певним стандартам, щоб зробити програмне забезпечення надійним. На основі дотримання цих стандартів процесів компанії сертифікуються. Оцінювачі проводять інтерв'ю та збирають дані від компаній для надання цієї сертифікації. У деяких випадках можуть бути розроблені інструменти для підтримки цього процесу оцінки.

Використовуючи інженерію програмного забезпечення на основі моделей, можуть бути розроблені інструменти для підтримки цього процесу оцінки. У минулому був побудований такий інструмент оцінки процесу та продукту програмного забезпечення, CAPPassessor [12], на основі CAPPMM Common Automotive Process and Product Meta-Model [12]. Цей інструмент може бути використаний для проведення оцінок та перевірки впливу процесу програмного забезпечення на характеристики якості програмного продукту. Однак інструмент CAPPassessor потрібно покращити, оскільки в ньому є можливість вплинути на розрахунок балів у тому ж редакторі, який використовується для реєстрації даних оцінки. Це пов'язано з проблемами в метамоделі, що впливає на інструмент оцінки. Метамодель об'єднала рівні визначення стандартів, визначення оцінок та проведення оцінок в одному.

Інструмент довелося покращити, щоб вирішити цю проблему. Покращення інструмента починається з перевизначення метамоделі(ей) для підтримки розділення проблем, що є властивим процесу оцінки. Як рішення цієї проблеми було створено три нові метамоделі, кожна з яких має різну мету. В результаті дані рівня визначення оцінки чітко відокремлені від даних про бали. Ці 3 метамоделі разом називаються CAPPTMM (Common Automotive Process and Product Testing Meta-Model). На основі CAPPTMM побудований графічний редактор дерев, який допомагає у проведенні оцінок шляхом побудови дерев аудиту. Це корисно для збору даних та аналізу впливу дотримання стандартних процесів на характеристики якості програмного продукту, такі як надійність, безпека, підтримуваність тощо.

Інструмент оцінки валідується двома промисловими кейсами, щоб досягти того ж результату, що і при використанні електронних таблиць Excel, які є поточним способом проведення оцінок.

Впровадження модельно-орієнтованого підходу дозволяє більш структуровано та системно оцінювати якість програмного забезпечення, зокрема через застосування міжнародних стандартів, таких як ISO 25010. Це дослідження є актуальним для вдосконалення методів аудиту і тестування

якості програмного забезпечення, особливо в контексті інтеграції моделей зрілості процесів розробки, таких як CMMI та TMMi.

Мета дослідження - розробка модельно-орієнтованого підходу до аудиту та тестування якості програмного забезпечення на основі сучасних стандартів якості та моделей зрілості процесів.

Об'єкт дослідження - процеси аудиту та тестування якості програмного забезпечення в контексті застосування стандартів та моделей зрілості.

Предмет дослідження - методи та моделі оцінювання якості програмного забезпечення на основі стандартів, а також інструменти для їх реалізації.

Відповідно до мети роботи було сформовано наступні **задачі:**

- проаналізувати сучасні стандарти якості програмного забезпечення, зокрема ISO 25010.
- дослідити вплив процесів розробки та тестування на якість програмного забезпечення.
- адаптувати мета-моделі для оцінювання якості програмного забезпечення та процесів тестування.
- провести налаштування та випробування інструменту для оцінювання на основі мета-моделей.
- оцінити ефективність інструменту на прикладах промислового використання стандартів.
- розробити рекомендації щодо вдосконалення процесів тестування та оцінювання якості програмного забезпечення.

Методи дослідження.

В магістерській роботі використано наступні методи:

- Аналіз літературних джерел щодо стандартів якості програмного забезпечення.

- Модельно-орієнтоване моделювання процесів та продуктів.

- Методи системного аналізу для розробки мета-моделей.

- Емпіричні методи для тестування та перевірки інструменту оцінювання.

Наукова новизна отриманих результатів полягає в дослідженні мета-моделей для оцінювання якості програмного забезпечення та процесів тестування та визначенні ключових елементів процесів розробки та тестування, які впливають на якість програмного забезпечення, з подальшою їх інтеграцією в мета-моделі.

Практичне значення магістерської роботи.

Результати дослідження можуть бути використані для вдосконалення процесів аудиту та тестування якості програмного забезпечення в організаціях, що займаються розробкою програмних продуктів. Розроблені мета-моделі та інструменти оцінювання можуть бути впроваджені в промислових проєктах для забезпечення високої якості програмного забезпечення, підвищення його надійності та продуктивності.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 75 сторінок, і містить 34 рисунки 5 таблиць, список використаних джерел із 52 найменувань.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД СТАНДАРТІВ ПРОЦЕСУ ТЕСТУВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Огляд предметної області дослідження

У зв’язку зі зростанням вимог до комфорту та ефективності, сьогодні в автомобілях є більше електронних та електричних систем, ніж будь-коли. З цією залежністю приходить більша залежність від програмного забезпечення заради вбудованих систем, присутніх в автомобілях. Advanced Driver Assistance Systems (ADAS) — це новітнє застосування складного програмного забезпечення, яке допомагає водієві транспортного засобу. Приклади ADAS включають адаптивний круїз-контроль (ACC), системи допомоги при паркуванні та системи попередження про виїзд зі смуги руху. Щоб задовольнити швидко зростаючий попит споживачів на розширені функції, ефективність і зручність, розробляється все більше і більше систем, що потребують інтенсивного програмного забезпечення. Програмне забезпечення також все частіше застосовується для систем, важливих для безпеки електроживлення. Збій програмного забезпечення в автомобільній сфері може мати катастрофічні наслідки, а також серйозні фінансові втрати для автомобільних компаній через судові позови та відкликання транспортних засобів.

Щоб запобігти подібним подіям, необхідно оцінювати та покращувати якість програмного забезпечення. Виробники та постачальники дотримуються різних процесів для виробництва обладнання, частин і програмного забезпечення, які відповідають вимогам безпеки, функціональним вимогам тощо.

Компанії мають різні інженерні процеси, які вони використовують для розробки систем програмного забезпечення. Ці процеси повинні відповідати певним стандартам. На основі дотримання цих стандартів процесів процеси

компаній сертифіковані. Оцінювачі проводять співбесіди та збирають дані від цих компаній, щоб надати цю сертифікацію. У деяких випадках існують інструменти для підтримки організації в зборі й обробці даних. За допомогою методів, заснованих на моделі, ці процеси можна полегшити.

Ідея підходу в цій роботі базується на використанні керовану моделлю розробку програмного забезпечення для побудови концептуальних метамоделей процесу програмного забезпечення та стандартів продукту, щоб підтримувати аудит процесів для різних характеристик якості продукту за стандартом ISO 25010 [10]. На основі метамоделей була побудована загальна метамодель автомобільних процесів і продуктів (CAPPMM) [12]. Використовуючи CAPPMM, було створено два графічні інструменти: CAPPeditor і CAPPassessor [12], обидва з яких були підтверджені тематичними дослідженнями. CAPPeditor використовується для побудови предметно-спеціальних моделей (DSM) стандартів. CAPPassessor дозволяє користувачам створювати таблиці аудиту процесів на основі цих DSM. Крім того, користувачі можуть збирати та аналізувати дані, необхідні для якості програмного продукту, з точки зору процесу. У результаті CAPPassessor дає оцінку процесу та оцінку проекту на основі характеристик якості продукту з точки зору процесу, зокрема для CMMI [17] та ASPICE [15].

CAPPassessor і CAPPeditor змішали різні рівні, тому що: визначення стандарту, визначення оцінки, а також проведення оцінювання та обробка його результатів були побудовані на одному рівні, все в одній мета-моделі, CAPPMM. Це призводить до змішування різних етапів процесу оцінювання в одному й тому самому кінцевому інструменті. Хоча намагаються розділити етапи шляхом створення різних (табличних) редакторів, ефективність інструменту знижується через небажані побічні ефекти, що є наслідком цієї фундаментальної проблеми.

Розглянемо три випадки використання проекту:

• Визначення знання стандартів у мета-моделі. Для цього необхідно вивчити документацію стандартів CMMI, TMMi та ASPICE.

• Визначення знання про оцінку, необхідне чітке уявлення про те, на які аспекти якості продукції впливають різні процеси стандартів, як CMMI (для CAPPassessor), так і TMMi (над яким ми збираємося працювати). Це включає дві частини, як описано у визначенні проблеми вище. По-перше, фіксуючи рівень наслідків для таблиць аудиту процесу, не даючи оцінювачу можливість змінювати його для конкретного проекту під час проведення оцінювання. Ці зміни також мають бути відображені в метамоделі. Ми можемо визначити оцінку для конкретного випадку. Для цього ми можемо звернутися до оцінок якості та тестування.

• Оцінку можна виконати, надавши новий інструмент/документацію для оцінювання, включаючи покращений інструмент для оцінювання CMMI, TMMi та ASPICE, модифікуючи CAPPassessor і CAPPeditor. Крім того, оскільки стандарт TMMi подібний до стандарту CMMI, представляючи інструмент, який також може повторити оцінку TMMi. Цей крок передбачає використання наявної попередньої оцінки і отримання того самого кінцевого результату за допомогою інструменту, отриманого з цього проекту.

1.2. Опис основних характеристик моделі Capability Maturity Model Integration

CMMI (Capability Maturity Model Integration) – це інтегрована модель зрілості можливостей, яка використовується для вдосконалення процесів організацій, зокрема в галузі розробки програмного забезпечення, управління проектами, інженерії, постачання продукції тощо. CMMI надає структуру для оцінки зрілості процесів та їх постійного вдосконалення. Вона використовується для того, щоб підвищити ефективність, якість продукції та задоволеність клієнтів через покращення бізнес-процесів. Модель розробляється завдяки об’єднанню зусиль груп продуктів у галузі спільно з Інститутом розробки програмного забезпечення (SEI) [18].

Основні характеристики CMMI:

- Зрілість процесів: Модель визначає п’ять рівнів зрілості, які показують, наскільки ефективно організація управляє своїми процесами:

Рівень 1 – Початковий (Initial): Процеси неорганізовані, хаотичні. Залежність від окремих людей висока, і процеси можуть не повторюватися.

Рівень 2 – Керований (Managed): Процеси вже є, і вони управляються, але можуть бути не повністю стандартизовані чи повторювані. Організація реагує на проекти і зміни, однак часто діє без належного планування.

Рівень 3 – Визначений (Defined): Процеси стандартизовані, задокументовані та інтегровані по всій організації. Всі проекти використовують стандартизовані процеси.

Рівень 4 – Керований на основі кількісних показників (Quantitatively Managed): Організація використовує кількісні метрики для вимірювання ефективності процесів та покращення їх на основі даних.

Рівень 5 – Оптимізуючий (Optimizing): Організація постійно вдосконалює процеси за рахунок інновацій та аналізу ефективності.

- Інтеграція кількох дисциплін: CMMI розроблена для різних дисциплін і охоплює не лише розробку програмного забезпечення, але й інженерію, управління постачаннями, обслуговування і роботу з клієнтами.

- Поліпшення процесів: Модель CMMI акцентує увагу на безперервному покращенні процесів організації шляхом встановлення структурованих та стандартизованих підходів.

Моделі CMMI:

- CMMI для розробки (CMMI for Development): Фокусується на вдосконаленні процесів розробки продукції та послуг. Практики в цій моделі зосереджені на діяльності з розробки високоякісних продуктів/послуг для кінцевих користувачів.

- CMMI для обслуговування (CMMI for Services): Спрямована на вдосконалення процесів надання послуг.

- CMMI для закупівель (CMMI for Acquisition): Застосовується до управління закупівельними процесами. Практики в цій моделі зосереджені на

діяльності, яка починає та керує придбанням продуктів і послуг для задоволення вимог клієнтів.

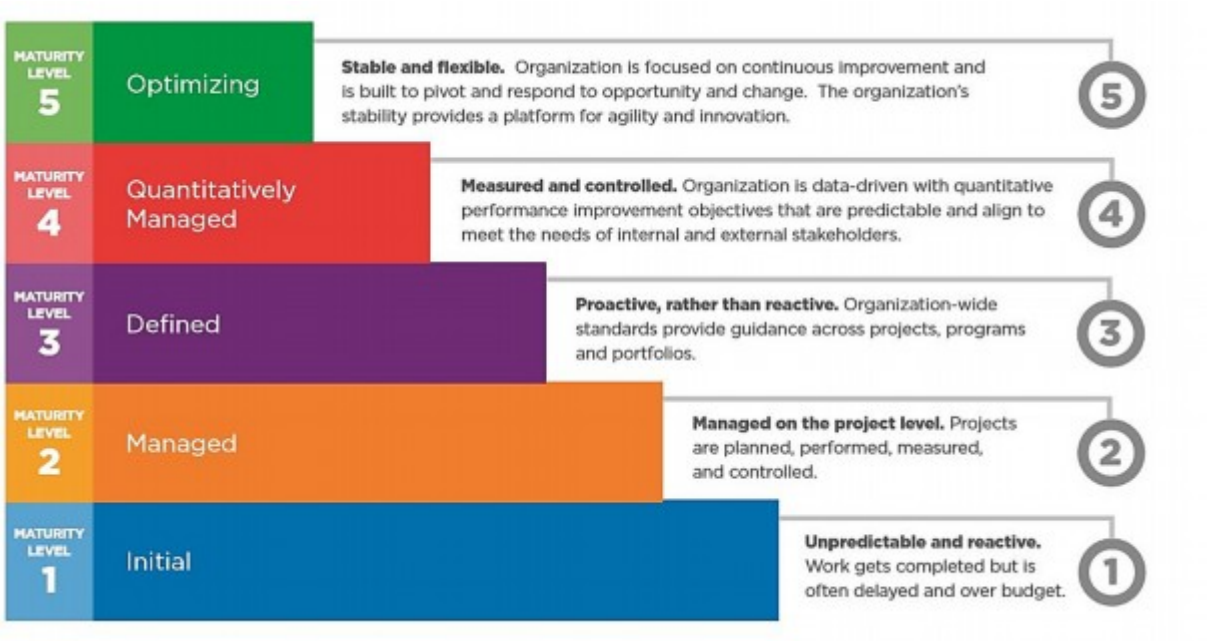


Рис. 1.1. Структура Capability Maturity Model Integration

Набір продуктів CMMI може забезпечувати перевірку того, що процеси встановлені, підтримуються та впроваджуються відповідно до найсучаснішого розуміння того, що таке «продуктивність світового класу». Однак CMMI не надаватиме підтвердження того, що організація має правильний продукт. Тому для використання CMMI слід здійснювати лише разом із повним розумінням бізнес-цілей і завдань, ринку та клієнтської бази, а також загальної стратегії продукту.

Практики CMMI: Модель містить практики і підходи до управління проектами, інженерії, забезпечення якості, управління ризиками та інші аспекти управління бізнесом.

Переваги CMMI:

- Підвищення якості продукції.
- Зниження ризиків і кількості дефектів.
- Поліпшення управління проектами.
- Стандартизація та повторюваність процесів.

- Підвищення задоволення клієнтів через надійніші продукти і послуги.

CMMI дозволяє організаціям систематично підходити до вдосконалення своїх процесів, досягаючи вищої продуктивності і якості робочих результатів.

Інтеграція моделі зрілості можливостей (CMMI) базується на чотирьох парадигмах:

- Можливість — дієздатний процес постійно виробляє вихідні дані, які відповідають його специфікаціям. Виконання ефективного процесу завжди дає передбачувані результати.

- Зрілість — зрілість означає, що все, що компанія робить, вона робить це добре задокументовано, коли кожен знає, чого від нього очікують, і діє відповідно, де ефективність не залежить від героїв і де рішення приймаються правильний аналіз ситуації.

- Модель — це структура або спосіб виконання чогось. CMMI надає лише структуру для визначення процесів у вашій організації, але не надає точних деталей процесу. Це лише керівництво для реалізації ефективних та ефективних процесів.

- Інтеграція — з історичної точки зору CMMI був поєднанням перевірених практик із низки різних моделей зрілості можливостей розробки програмного забезпечення та систем.

У таблиці 1.1 показано випадки, коли організації повідомили про зниження вартості робочих продуктів, витрат на процеси, а також загалом більше заощадили за допомогою вдосконалення процесів на основі моделі.

Таблиця 1.1.

Приклад збільшення рентабельності при використанні CMMI [7]

Result	Model
33% decrease in the average cost to fix a defect (Boeing, Australia)	CMMI
20% reduction in unit software costs (Lockheed Martin M&DS)	CMMI
15% decrease in defect find and fix costs (Lockheed Martin M&DS)	CMMI
4.5% decline in overhead rate (Lockheed Martin M&DS)	CMMI
Improved and stabilized Cost Performance Index (Northrop Grumman IT1)	CMMI

Таблиця 1.2.

Короткий опис переваг та впливу CMMI

Result	Model
Reduced by half the amount of time required to turn around releases (Boeing, Australia)	CMMI
60% reduction in work and fewer outstanding actions following pre-test and post-test audits (Boeing, Australia)	CMMI
Increased the percentage of milestones met from approximately 50% to approximately 95% (General Motors)	CMMI
Decreased the average number of days late from approximately 50 to fewer than 10 (General Motors)	CMMI
Increased through-put resulting in more releases per year (JP Morgan Chase)	CMMI
30% increase in software productivity (Lockheed Martin M&DS)	CMMI
Improved and stabilized Schedule Performance Index (Northrop Grumman IT1)	CMMI
Met every milestone (25 in a row) on time, with high quality and customer satisfaction (Northrop Grumman IT2)	CMMI

Таблиця 1.2 показує покращення розкладу у двох аспектах: скорочення часу, необхідного для виконання завдань, а також можливість кращого прогнозування розкладів.

1.3. Дослідження архітектурних компонентів моделі

Модель CMMI надає користувачам варіанти поетапного та безперервного представлень, і архітектурні компоненти [17] присутні в обох.

1. Сфери процесу. Сфера процесу – це набір пов’язаних практик у сфері, які, якщо вони задоволені, також задовольняють набір цілей, важливих для цієї сфери.

2. Конкретні цілі - «Описує унікальні характеристики, які повинні бути присутніми, щоб задовольнити область процесу».

3. Загальні цілі. Ці цілі є «загальними», тому що одна і та ж ціль застосовується до кількох областей процесу».

4. Специфічна практика – «Опис діяльності, яка вважається важливою для досягнення відповідної конкретної мети».

5. Загальна практика. Ці практики називаються «загальними», тому що одна й та сама практика застосовується до кількох сфер процесу».

Модель CMMI пропонує поетапне та постійне вдосконалення процесів, використовуючи відповідно рівні зрілості та рівні можливостей. CMMI

підтримує вдосконалення за допомогою цих двох шляхів. Обидва представлення мають однакові архітектурні компоненти. Поетапне представлення, рівні можливостей використовуються для перевірки вдосконалення процесу організації в окремих областях процесу, а чотири рівні можливостей пронумеровані від 0 до 3. Рівні зрілості можуть застосовуватися до вдосконалення процесу організації в кількох областях процесу. Кожен рівень зрілості є визначеним еволюційним плато і містить певний набір процесів, які готують його до переходу до наступного рівня зрілості. У поетапному представленні є 5 рівнів зрілості [17].

Таблиця 1.3.

Рівні можливостей у CMMI

Capability Level 0	Incomplete
Capability Level 1	Performed
Capability Level 2	Managed
Capability Level 3	Defined

У таблиці 1.3 можна побачити рівні можливостей у безперервному представленні CMMI. Безперервне представлення дозволяє організації вибрати конкретну область процесу (PA - Process Area) і вдосконалити її шляхом досягнення конкретних і загальних цілей у цій конкретній PA. Після досягнення 3-го рівня спроможності в певних сферах процесів організації можуть ще більше вдосконалювати свої процеси, запроваджуючи сфери процесів вищого рівня зрілості (шляхом досягнення відповідних цілей).

Таблиця 1.4.

Рівні зрілості в CMMI

Maturity Level 1	Initial	Process Unpredictable, Poorly Controlled
Maturity Level 2	Managed	PP, PMC, SAM, REQM, MA, PPQA, CM
Maturity Level 3	Defined	IPM, RSKM, RD, TS, PI, VER, VAL, DER, OPF, OPD, OT
Maturity Level 4	Quantitatively Managed	QPM, OPP
Maturity Level 5	Optimizing	CAR, OPM

У таблиці 1.4 можна побачити рівні зрілості в поетапному представленні CMMI. На кожному рівні зрілості в стандарті є певні області процесу. Ці зони процесу також можна побачити на рисунку 1.2. Рівень зрілості досягається шляхом завершення конкретних цілей і загальних цілей ключових сфер процесу, визначених на цьому відповідному рівні.

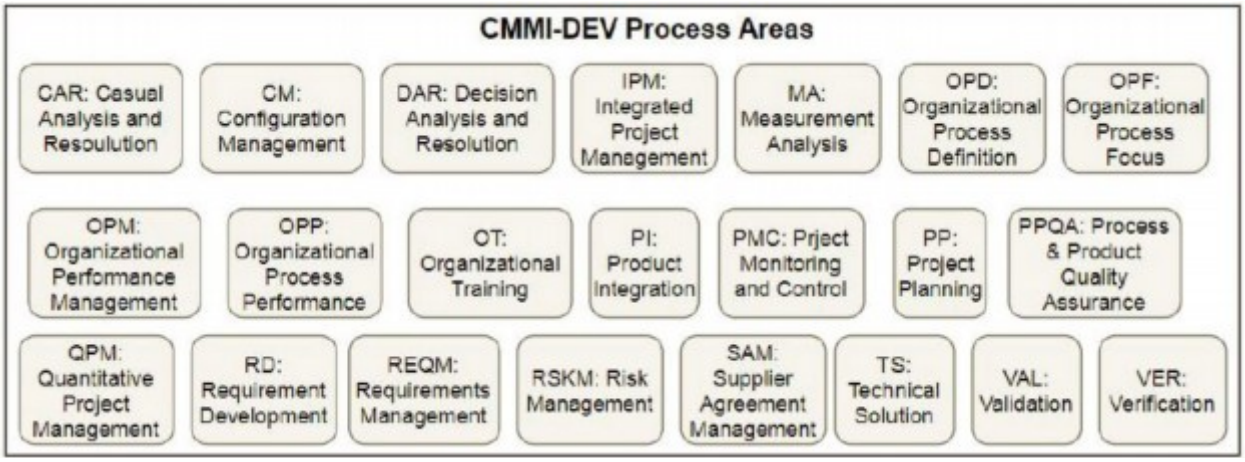


Рис. 1.2. Огляд областей процесу в CMMI-DEV [12]

На рисунку 1.2 показано огляд 22 ключових областей процесу (PA) [17], які містяться в моделі CMMI-DEV.

1.4. Особливості моделі зрілості процесів тестування

Хоча відомо, що на тестування припадає 30-40 відсотків витрат на проект, дуже мало уваги приділяється тестуванню в моделях вдосконалення програмного забезпечення, таких як CMMI. Тому спільнота тестувальників запропонувала власні моделі для вдосконалення, однією з яких є інтеграція моделі зрілості тестування (TMMi - Testing Maturity Model integration) [19]. Це детальна модель для вдосконалення процесу тестування та розроблена як доповнення до CMMI. Фреймворк TMMi був створений як довідкова основа та рекомендації для вдосконалення процесів тестування для питань, важливих для менеджерів тестування, інженерів тестування та професіоналів, які працюють у сфері якості програмного забезпечення.

Згідно з TMMi, тестування визначається як «процес, що складається з усіх дій життєвого циклу, як статичних, так і динамічних, пов'язаних із плануванням, підготовкою та оцінкою програмних продуктів і пов'язаних робочих продуктів, щоб визначити, що вони задовольняють заданим вимогам, щоб продемонструвати, що вони придатні для призначення та для виявлення дефектів» [19].

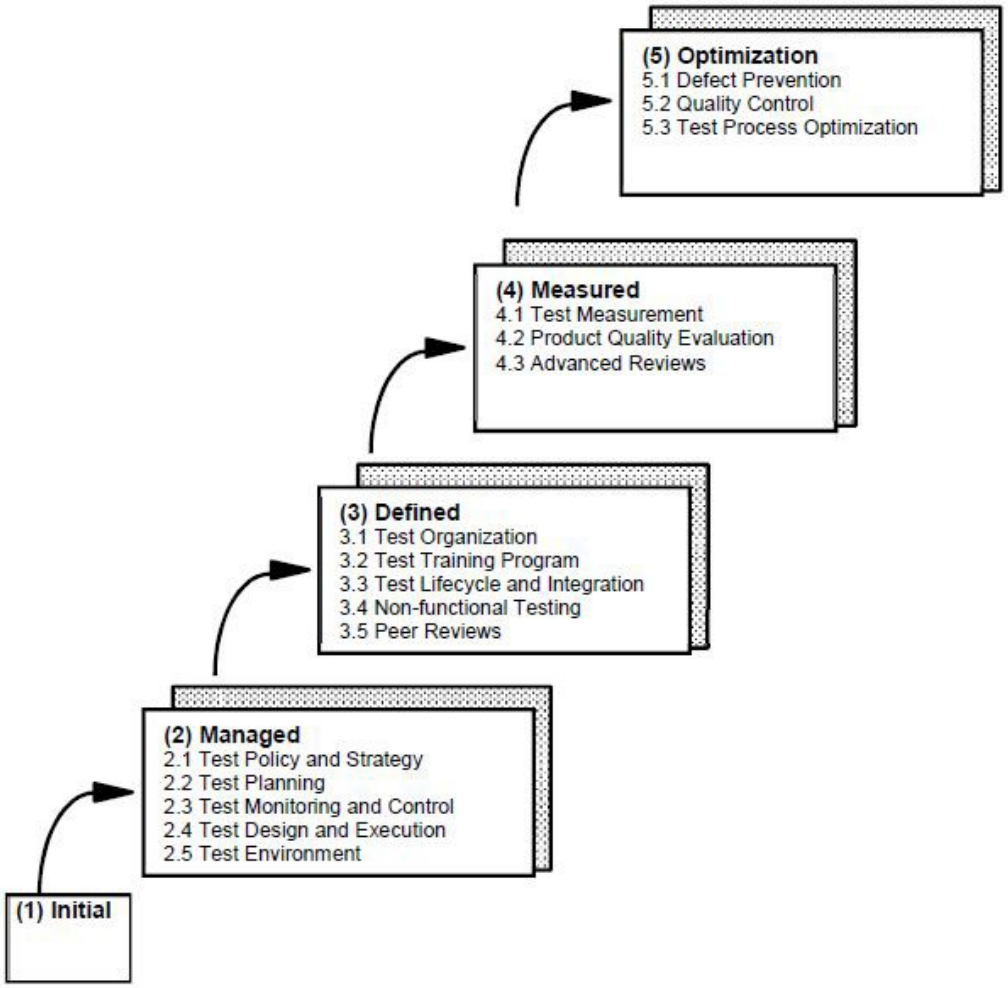


Рис. 1.3. Рівні зрілості та області процесу, присутні в TMMi [19]

CMMI має як етапне, так і безперервне представлення, і оскільки розробка TMMi керувалася роботою, виконаною над CMMI, TMMi було розроблено як модель етапу. Поетапна модель складається з рівнів/етапів, які організація проходить у процесі тестування. Якщо дотримуватися моделі TMMi, вона починається з некерованого процесу та переходить до процесу,

яким керують, визначають, вимірюють і, нарешті, на стадії постійного вдосконалення, відомої як оптимізація. На рисунку 1.3 можна побачити області процесу для кожного рівня зрілості TMMi. У TMMi є 5 рівнів, які демонструють еволюційний шлях до вдосконалення процесу тестування. Кожен із цих рівнів має низку процесів, які організація має реалізувати, щоб досягти зрілості на цьому рівні.

Рівні зрілості TMMi: TMMi складається з п'яти рівнів зрілості, які показують еволюцію процесів тестування від хаотичного до оптимізованого:

Рівень 1 – Початковий (Initial):

- Процеси тестування є неорганізованими і непередбачуваними.

- Тестування не стандартизовано і часто залежить від окремих осіб або проєктів.

Рівень 2 – Керований (Managed):

- Основи тестування визначені та повторювані.

- Тестування відокремлене від розробки, і процеси починають управлятися.

- Акцент на плануванні, моніторингу та управлінні тестуванням.

Рівень 3 – Визначений (Defined):

- Процеси тестування стандартизовані та інтегровані в організацію.

- Тестування стає частиною всього життєвого циклу розробки програмного забезпечення.

- Акцент на профілактичних заходах і тестуванні на основі вимог.

Рівень 4 – Керований на основі кількісних показників (Measured):

- Тестування управляється на основі кількісних показників та метрик.

- Використовуються інструменти для вимірювання ефективності процесів тестування, таких як дефектність, покриття тестами і якість продукту.

Рівень 5 – Оптимізуючий (Optimizing):

- Процеси тестування постійно вдосконалюються через впровадження інновацій та нових технологій.

- Акцент на профілактиці дефектів, автоматизації тестування та оптимізації часу і витрат.

1.4.1. Компоненти моделі тестування

Нижче наведені компоненти, присутні в моделі TMMi:

1. Рівні зрілості – їх можна розглядати як ступінь якості процесу організаційного тестування. Це визначається як еволюційне плато вдосконалення процесу тестування». Кожен рівень зрілості визначає, що потрібно зробити, щоб отримати цей рівень. Чим вищого рівня зрілості досягає організація, тим вищою є зрілість процесу тестування цієї організації. Щоб досягти конкретного рівня зрілості, організація повинна досягти всіх загальних і конкретних цілей сфери процесу цього рівня разом з цілями всіх нижчих рівнів. Усі організації вже мають мінімальний рівень 1, оскільки цей рівень не містить жодних цілей, які потрібно досягти.

2. Сфери процесу – вони «визначають проблеми, які необхідно вирішити, щоб досягти рівня зрілості. Кожна область процесу визначає кластер дій, пов’язаних з тестуванням». Крім рівня 1, кожен рівень зрілості включає низку областей процесу, які показують, на що організації слід звернути увагу, щоб покращити свій процес тестування. Для досягнення рівня зрілості всі області процесу цього рівня зрілості, а також нижчі рівні зрілості повинні бути задоволені. Наприклад, щоб отримати сертифікат рівня 3, він повинен задовольняти технологічні області, присутні в TMMi рівня 2, а також TMMi рівня 3.

3. Специфічні цілі - описує унікальні характеристики, які повинні бути присутніми, щоб задовольнити сферу процесу. Конкретна мета є обов’язковим компонентом моделі та використовується в оцінках, щоб допомогти визначити, чи задоволена область процесу».

4. Загальні цілі - «описує характеристики, які повинні бути присутніми для інституціоналізації процесів, які реалізують сферу процесу». Вони

називаються загальними, тому що ці цілі застосовуються до всіх областей процесу в TMMi.

5. Специфічні практики – це «опис діяльності, яка вважається важливою для досягнення відповідної конкретної мети». Якщо дії, описані в конкретній діяльності, виконуються, це означає, що ця конкретна мета в цій області процесу досягнута.

6. Загальні практики – вони «з'являються ближче до кінця області процесу та називаються «загальними», оскільки одна й та сама практика з'являється в усіх областях процесу». Загальні практики — це дії, які після завершення призводять до досягнення пов'язаної загальної цілі.

На рисунку 1.4 узагальнено компоненти та взаємозв'язок між ними.

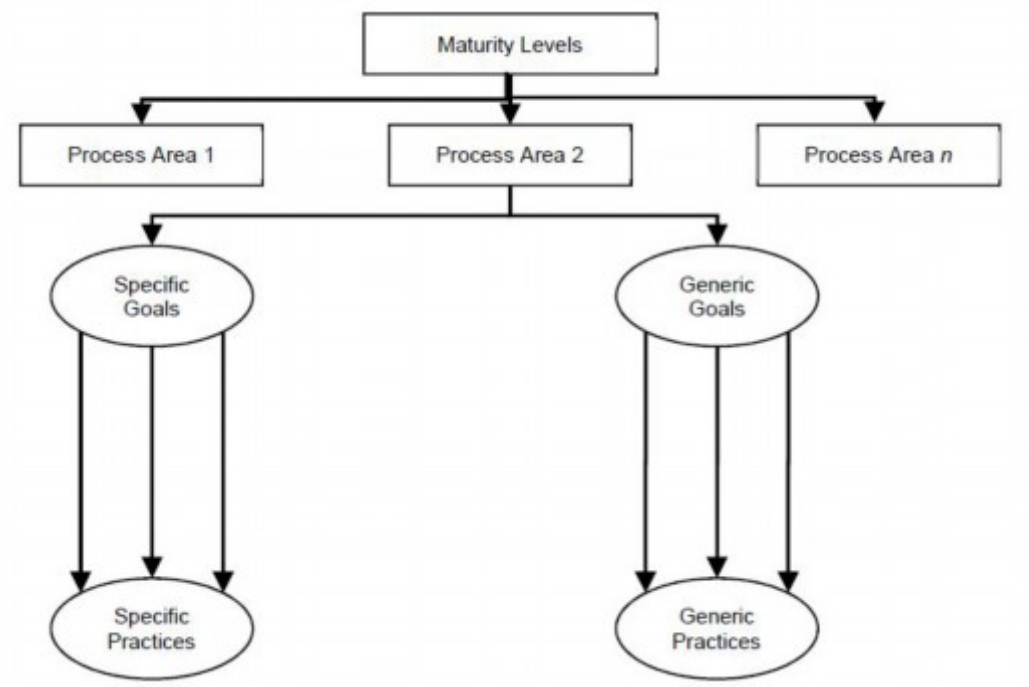
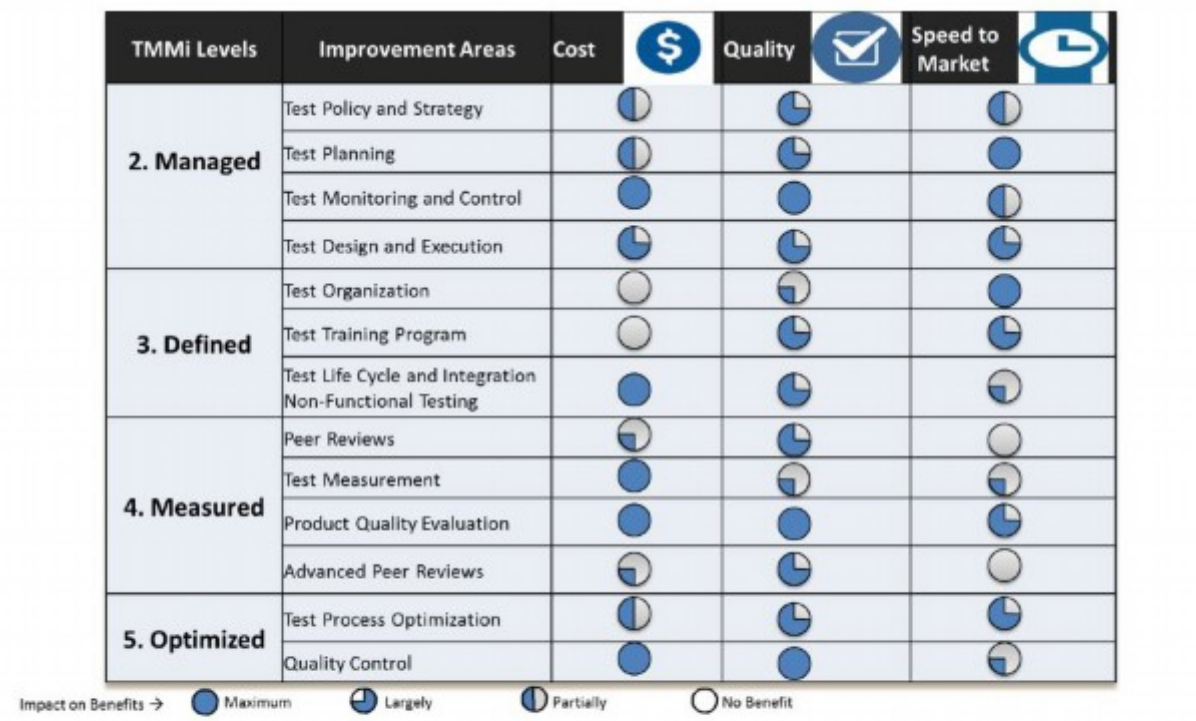


Рис. 1.4. Структура та компоненти, наявні в TMMi [17]

1.4.2. Переваги моделі

Причини, за якими організація може отримати користь від оцінки TMi, можна загалом згрупувати таким чином: удосконалення, зазначені на рисунку 1.5, базуються на кількох прикладах проектів. TMMi зосереджується на методах тестування та дозволяє підвищити якість завдяки досконалості в

тестуванні, а також надає відчутні переваги організаціям, враховуючи той факт, що фаза тестування проекту враховує від 40 до 50 відсотків зусиль і вартості проекту [3].



	Level	1. Initial	2. Managed	3. Defined	4. Measured	5. Optimized
Savings & ROI	Total Return on Investment %	0.0%	26.1%	55.7%	61.8%	64.9%
	ROI over previous Maturity Level	0.0%	26.1%	40.1%	13.8%	8.2%

Рис. 1.5. Переваги Testing Maturity Model integration (TMMi) [3]

1.4.3. Порівняння моделей

TMMi розроблений і розміщений, щоб доповнити CMMI. У деяких випадках даний рівень TMMi потребує підтримки саме з боку процесних областей на відповідному рівні CMMI або нижчих рівнях CMMI. За деякими винятками вони навіть пов’язані з вищими рівнями CMMI.

Подібності між двома моделями вже було розглянуто. Оскільки TMMi був розроблений як доповнення до CMMI, вони мають однакові архітектурні елементи. У таблиці 1.5 можна побачити основні відмінності між двома моделями.

Таблиця 1.5.

Порівняння двох моделей

Capability Maturity Model Integration	Test Maturity Model integration
Limited focus on test improvements	Limited focus on non-testing improvements
Has both staged and continuous representation, so uses both maturity and capability levels to measure improvement	Has only staged representation, so only uses maturity levels to measure improvement
CMMI Version 1.3 has 3 frameworks- CMMI for Development CMMI for Acquisition CMMI for Services	No additional TMMi frameworks
Developed by the Software Engineering Institute at the Carnegie Mellon University	Developed by the TMMi foundation

1.4.4. Стандарт для оцінки якості програмного забезпечення

ISO 25010 – це міжнародний стандарт, який визначає модель якості систем і програмних продуктів. Він надає чіткий і структурований підхід до оцінки та вимірювання якості програмного забезпечення, що дозволяє компаніям та організаціям забезпечувати високу якість своїх програмних продуктів.

Стандарт ISO 25010 визначає кілька характеристик якості, які можна використовувати для оцінки програмного продукту:

- Функціональність: Наскільки добре продукт виконує свої призначені функції.

- Ефективність: Наскільки ефективно продукт використовує ресурси (час, пам'ять тощо).

- Сумісність: Наскільки добре продукт взаємодіє з іншими системами та програмним забезпеченням.

- Безпека: Наскільки добре продукт захищений від несанкціонованого доступу та інших загроз.

- Зручність використання: Наскільки легко користувачам користуватися продуктом.

- Підтримуваність: Наскільки легко модифікувати та підтримувати продукт.

- Переносимість: Наскільки легко продукт адаптувати до різних середовищ.

Переваги використання ISO 25010:

- Покращення якості продуктів: Систематичний підхід до оцінки якості дозволяє виявляти та усувати проблеми на ранніх етапах розробки.

- Збільшення задоволеності клієнтів: Висока якість програмного продукту призводить до більшої задоволеності клієнтів і підвищення їхньої лояльності.

- Зменшення витрат: Виявлення та усунення дефектів на ранніх етапах розробки дозволяє знизити витрати на виправлення помилок.

- Підвищення конкурентоспроможності: Високоякісне програмне забезпечення є важливою конкурентною перевагою на ринку.

ISO/IEC 25010 є оновленою стандартною моделлю ISO/IEC 9126 [4]. Ця версія ISO/IEC 25010 була випущена в березні 2011 року. Ця модель містить вісім характеристик якості продукції. Модель містить 31 підхарактеристику якості на відміну від ISO/IEC 9126 21 підхарактеристику якості характеристики.

Рис. 1.6. Модель якості продукції в ISO/IEC 25010

ISO 25010 – це незамінний інструмент для організацій, які прагнуть забезпечити високу якість своїх програмних продуктів. Він надає чіткий і структурований підхід до оцінки якості, дозволяючи компаніям підвищити свою конкурентоспроможність і задовольнити вимоги клієнтів.

ISO/IEC 25010 — це міжнародний стандарт, що визначає модель якості для оцінювання характеристик програмного забезпечення та систем. Ця модель є частиною серії стандартів SQuaRE (Software product Quality Requirements and Evaluation), яка фокусується на вимогах до якості програмних продуктів та процесів їх оцінки.

Основні особливості стандарту ISO 25010:

1. Дві моделі якості:

- Модель якості продукту — застосовується для оцінки внутрішніх та зовнішніх характеристик програмного забезпечення.

- Модель якості використання — призначена для оцінки впливу якості програмного забезпечення на кінцевих користувачів під час його експлуатації.

2. Модель якості продукту: Включає вісім основних характеристик, які використовуються для оцінки якості програмного продукту:

- Функціональна придатність (Functional suitability): здатність програмного забезпечення надавати функціональність, що відповідає вимогам.

- Продуктивність (Performance efficiency): ефективність використання ресурсів системи, таких як час відгуку, швидкість та споживання енергії.

- Сумісність (Compatibility): здатність програмного забезпечення працювати в різних середовищах, з іншими системами або версіями.

- Юзабіліті (Usability): простота та ефективність використання системи кінцевими користувачами.

- Надійність (Reliability): здатність програмного забезпечення виконувати свої функції впродовж визначеного часу та умов експлуатації.

- Безпека (Security): здатність захищати дані і забезпечувати конфіденційність, цілісність і доступність.

- Підтримуваність (Maintainability): легкість обслуговування, зміни, виправлення помилок та вдосконалення.

- Переносимість (Portability): здатність програмного забезпечення переноситися на інші платформи без втрати функціональності.

3. Модель якості використання: Включає п’ять основних характеристик, які визначають ефективність використання програмного забезпечення з точки зору користувача:

- Ефективність (Effectiveness): здатність користувачів досягати своїх цілей за допомогою системи.

- Продуктивність (Efficiency): рівень ресурсів, що витрачаються для досягнення цілей (час, зусилля).

- Задоволеність (Satisfaction): рівень задоволення користувачів результатами взаємодії з системою.

- Безпека (Freedom from risk): захист користувачів від ризиків, таких як втрати даних або неправильне використання системи.

- Контекст охоплення (Context coverage): здатність системи ефективно працювати в різних умовах використання.

4. Інтеграція з іншими стандартами: ISO 25010 тісно пов'язаний з іншими стандартами серії SQuaRE (наприклад, ISO/IEC 25040 для процесу оцінювання якості), що дозволяє комплексно оцінювати як технічні, так і користувацькі аспекти програмного забезпечення.

5. Гнучкість і адаптованість: Стандарт дозволяє адаптувати модель якості під конкретні потреби проєктів або організацій. Характеристики та підхарактеристики можуть бути визначені або змінені залежно від специфічних вимог продукту або системи.

6. Оцінка внутрішніх і зовнішніх характеристик: ISO 25010 дозволяє оцінювати якість як під час розробки (внутрішня якість), так і після її завершення (зовнішня якість), а також якість використання в реальних умовах.

7. Підтримка сучасних підходів до розробки: Стандарт може використовуватися в контексті сучасних методологій розробки, таких як

DevOps, Agile, та інших, забезпечуючи гнучкість в оцінюванні якості на різних етапах життєвого циклу програмного забезпечення.

ISO 25010 є важливим інструментом для структурованої та всеохопної оцінки якості програмного забезпечення, забезпечуючи стандартизовані підходи до вимірювання та контролю різних аспектів його якості.

1.4.5. Зв'язок між CMMI/TMMi та моделлю якості

CMMI та TMMi є стандартами процесів, і спосіб виконання процесу розробки програмного забезпечення або процесу тестування програмного забезпечення матиме значний вплив на характеристики програмного продукту, такі як надійність, підтримуваність, зручність використання тощо.

Для проведення своїх оцінок використовують матрицю для аудитів якості програмного забезпечення, яка використовується для проведення оцінок. Використовуючи цю матрицю, можна отримати рівень/оцінку надійності (за шкалою 1-4) практики процесу на основі рівня наслідків (який визначається заздалегідь перед оцінкою під час його визначення) та рівня задоволеності (який знаходиться як результат оцінки) цієї конкретної практики.

Consequence / Satisfaction	Insignificant	Minor	Moderate	Major	Extreme
Not Satisfied	3	2	1	1	1
Partially Satisfied	4	3	2	1	1
Largely Satisfied	4	3	3	2	1
Fully Satisfied	4	4	4	4	4

Рис. 1.7. Матриця надійності аудиту програмного забезпечення [12]

Метою цього проекту є підтримка аудиту процесу та тестування для оцінювання якості програмного забезпечення. Для досягнення цієї мети необхідно оцінити вплив процесів розробки та тестування програмного забезпечення на якість продукції. Пам'ятаючи про часові обмеження цього

проекту, ми розглянули певні конкретні частини стандартів CMMMI (Capability Maturity Model Integration) та TMMi (Testing Maturity Model integration) та їхній вплив на якість програмного продукту ISO 25010 щодо надійності.

Висновки до розділу

Розділ присвячений дослідженню взаємозв’язку між стандартами процесу (CMMI, TMMi) та стандартом продукту (ISO 25010) у контексті оцінювання програмного забезпечення. Аналізується, як характеристики процесу розробки, що визначаються стандартами CMMI та TMMi, впливають на якість готового продукту, яка оцінюється за стандартом ISO 25010. Наведено приклади практичного застосування цих стандартів та їхньої інтеграції в процесі розробки програмного забезпечення

РОЗДІЛ 2. МОДЕЛІ ТА ПРИНЦИПИ ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Особливості процесу оцінювання

У цьому розділі показано те, яка підтримка потрібна для всього процесу оцінювання. Це робиться шляхом опису того, як відбувається процес оцінювання в компанії з його етапами, і чому старішому інструменту бракує певних аспектів цього.

Оцінка [11] – це діяльність зі збору та перевірки доказів узгодженості та впровадження щодо еталонного стандарту, еталонної моделі чи еталонної структури. Довідкові стандарти для цього інструменту можуть варіюватися від CMMI до ASPICE і TMMi. Результатом оцінки є звіт про оцінку. Звіт містить докази дотримання стандарту або його невідповідності разом із пропозиціями щодо покращення. Він також може або не може надати остаточну кількісну оцінку для звіту, щоб дати чітке уявлення читачеві, але він також дає короткий підсумок загального стану відповідності.

Нижче наведено різні етапи процесу оцінювання з точки зору роботи над програмним інструментом:

• Визначення відповідних стандартів. Це робиться один раз для кожного стандарту відповідно до наявної текстової документації, виданої відповідним керівним органом стандартів.

• Визначення оцінювання, яке буде проводитися, та підрахунок балів (один раз для клієнта або типу оцінювання). Ці кроки, серед іншого, вибирають відповідні частини стандартів для цілей оцінювання та пов’язують їх, де це необхідно для клієнта. Це робиться один раз для кожного клієнта або типу оцінювання.

• Проведення оцінювання. Це можна зробити кілька разів для того самого визначення оцінки.

Для проведення оцінки компанія призначає аудиторську групу з головним оцінювачем. Оцінювач відповідає за проведення загальної оцінки. Компанія бере участь як незалежна сторона, щоб надати інформацію про аспекти якості програмного забезпечення для різних компаній. Це може бути оцінка процесу, або оцінка коду, або навіть тестова оцінка. Це може включати такі кроки, як вивчення документації, співбесіди та звітування. Часто оцінка допомагає визначити зони ризику та дає рекомендації щодо покращення.

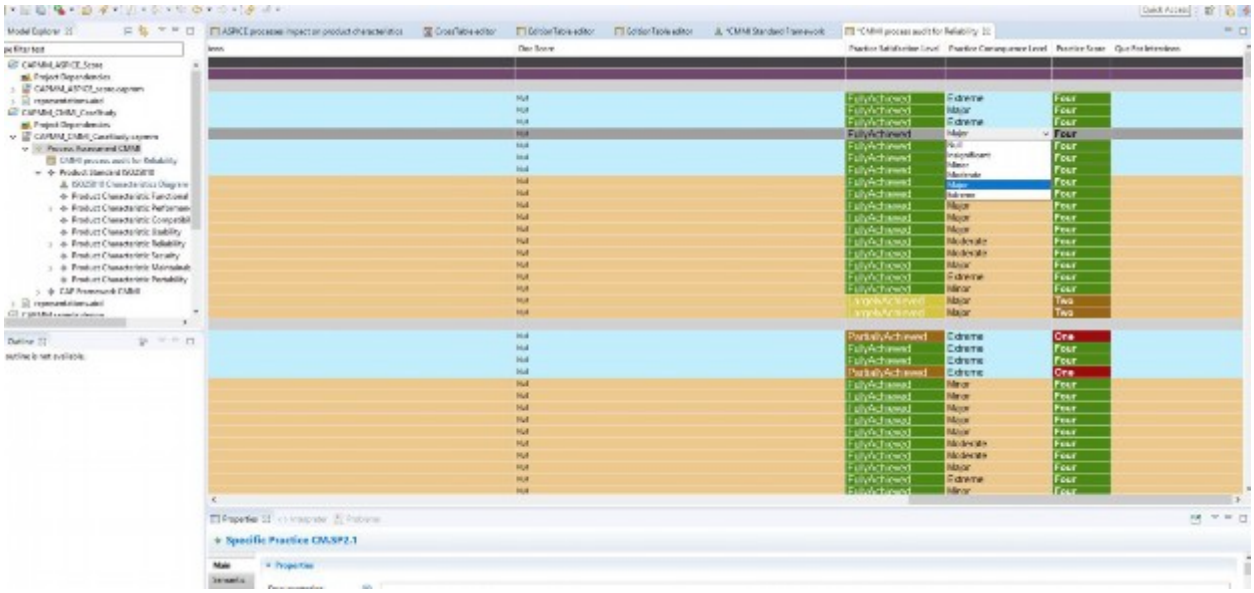


Рис. 2.1. Рівні наслідків у таблиці аудиту процесу CMMI [12]

Таблиці аудиту процесу CAPPassessor мають рівні наслідків. Ці рівні є показником того, наскільки певний процес у межах стандарту процесу впливає на якість продукту. У поточній версії CAPPassessor рівні наслідків можуть бути змінені користувачем інструменту. Оцінювач може вибрати один із рівнів наслідків: нульовий, незначний, незначний, помірний, серйозний і екстремальний. Це слід заздалегідь змінити до фіксованого рівня для кожного процесу. Оцінювач не повинен вирішувати під час оцінки, наскільки конкретна практика або загальна практика (процесу) впливає на певну якість продукту. Небажаним ефектом цього можуть бути необ’єктивні

результати. Цей рівень наслідків має бути заздалегідь визначений відповідно до знань оцінки. На рисунку 2.1 наведено таблицю з аудитом процесу CMMI для надійності, у якій користувач може вибрати рівень наслідків у стовпці «Рівень наслідків практики».

У CAPPMM рівні не були чітко розмежовані з точки зору визначення відповідних стандартів, визначення оцінки та, нарешті, виконання самої оцінки. Визначення оцінки та її виконання виконується провідним оцінювачем та групою оцінювання. Реалізація цих кроків була об'єднана в один рівень з точки зору мета-моделі, і це може заплутати та бути незручним для тих, хто хоче визначити нову оцінку для завдання компанії, або навіть для тих, хто хоче включити новий стандарт у інструмент. У цьому проекті ми спробували роз'яснити плутанину та включити більше відстеження між метамоделями та нашою версією інструменту оцінювання.

Проблеми в метамоделі впливають на весь набір інструментів. Удосконалення інструментарію починається з перевизначення метамоделі (метамоделей) для підтримки поділу проблем, властивих процесу оцінювання.

Щоб покращити процес оцінювання шляхом використання програмного засобу замість використання Excel, цього рішення «єдиної метамоделі» недостатньо. У наступному розділі представляємо нові метамоделі, які вирішують вищезгадані проблеми.

2.2. Дослідження програмних інструментів для роботи із структурованими моделями даних

EMF (Eclipse Modeling Framework) [8] — це структура моделювання та інструмент для генерації коду для створення програмних засобів та інших програм на основі структурованої моделі даних. EMF походить від специфікації моделі, описаної в XMI, і надає інструменти та підтримку для виконання для створення класів Java для моделі разом із набором класів

адаптерів, які дозволяють користувачам переглядати та редагувати на основі команд модель, а також базовий редактор.

До інструментів моделювання Eclipse відноситься Ecore, ядро/центр EMF, який використовується для створення метамоделей. Концепції, які користувач хоче включити у свій проект, можна змоделювати під EMF у файлі Ecore. Структура зберігається у файлах із розширенням «ecore», тобто закінчується на .ecore. Eclipse має майстер створення файлів, який допомагає користувачам створювати такі файли. За замовчуванням, коли такий (новий) файл відкривається, він відкривається в редакторі на основі дерева, що дає змогу будувати структуру елементів моделі Ecore. Інші доступні опції включають редактор діаграм інструментів Ecore, редактор Ecore на основі форм EMF [14].

Підводячи підсумок, Ecore є об’єктно-орієнтованою мовою EMF. Коли будується модель Ecore (незалежно від того, який редактор використовується), по суті, користувач будує структуру об’єкта, структуру «метаоб’єктів». Причина, чому їх називають мета, полягає в тому, що вони знаходяться на концептуальному рівні вище екземплярів, які вони моделюють [14].

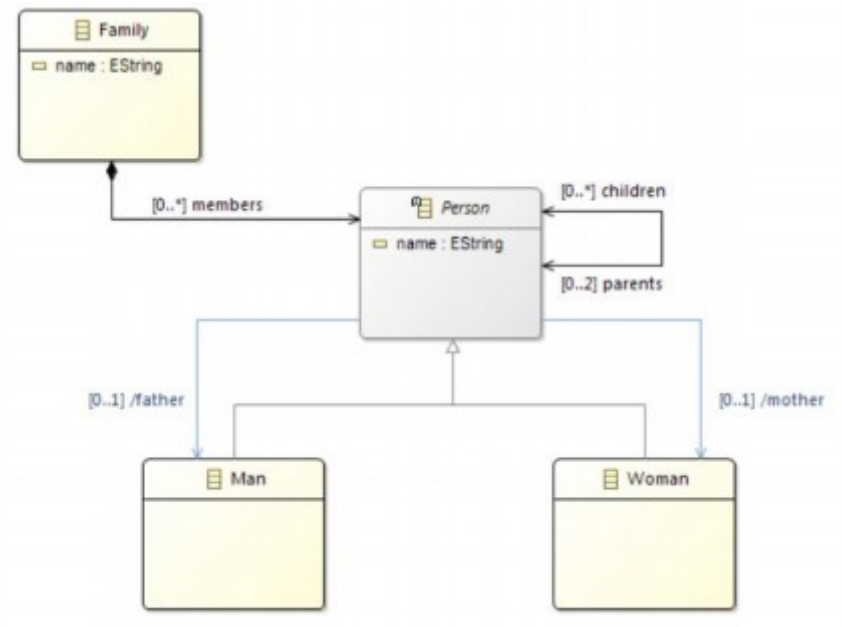


Рис. 2.2. Приклад метамоделі

На рисунку 2.2 показаний приклад моделі Ecore, що представляє структуру сімейства. Основне поняття сім'ї полягає в тому, що вона складається з кількох осіб. Це показано з композиційним зв’язком між класом Family і класом Person. Особа може бути чоловіком або жінкою, що відображається з супертипом/наслідуванням із класу Person. Кожна особа також має атрибут імені та такі посилання: textitmother, батько, двоє батьків і діти.

Концепції та зв’язки родини були змодельовані тут за допомогою палітри, використовуючи компоненти інструментів Класифікатор, Відношення та Характеристика.

Подібним чином Вимоги до будь-якого проекту можна моделювати за допомогою EMF у файлі ecore як метамодель. У наступних частинах цього розділу буде пояснено, як ми побудували різні мета-моделі, щоб створити програмний інструмент, який можна використовувати для оцінювання.

2.3. Методика покращення мета-моделей для процесу оцінювання якості

Перше, що було зрозуміло на початку цього проекту, полягало в тому, що потрібно було більше чіткості між мета-моделями та інструментом, за допомогою якого здійснюватиметься оцінювання. Початково була побудована лише одна мета-модель, у яку все було інтегровано. З його інструментом важко розрізнити рівні потреб/вимог, які інструмент може задовольнити. У нашому інструменті для вирішення цієї проблеми «Об’єднаної однієї моделі» використовуються такі рівні:

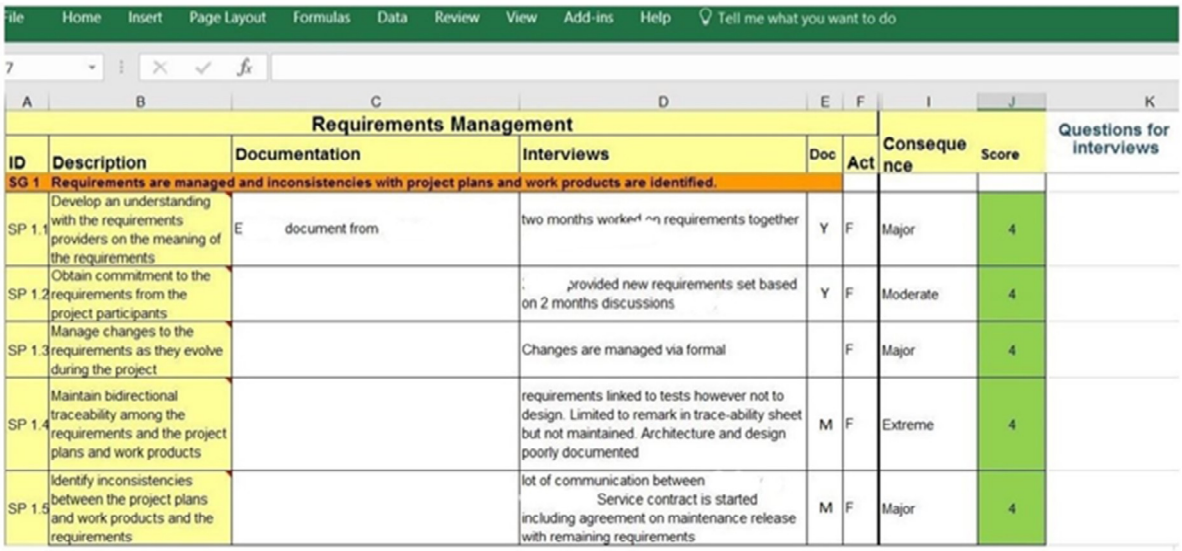
1. Визначення стандартів. Оцінки, які проводяться, завжди проводяться з посиланням на певний стандарт (наприклад, CMMI, ASPICE тощо). Різні класи в цій моделі можуть бути лише концепціями, які вже визначені в таких стандартах, як CMMI і TMMi. Ці стандарти мають бути чітко визначені в екземплярі метамоделі, щоб особа, яка визначає оцінку на наступному кроці,

мала доступ до матеріалу/інформації, наданої у стандартах. Ця мета-модель рівня має справу з визначенням стандартів. У нашому проекті ми назвали це «стандарти».

2. Визначення оцінок. На цьому рівні моделі йдеться про визначення оцінки перед тим, як звернутися до клієнта для проведення оцінки. Це передбачає підготовку «шаблону», який полегшує роботу оцінювача (або команди з оцінювання, яка виконуватиме фактичне завдання з оцінювання). Ця модель отримала назву «оцінка».

3. Виконання фактичного оцінювання. Цей рівень моделі стосується оцінювання, яке проводиться, тобто збирання даних, проведення опитувань, інтерв'ю тощо, щоб отримати остаточну оцінку. Цю модель назвали «фактичною».

На рисунку 2.3 наведено аркуш оцінювання клієнта з шаблону Altran Excel. У таблиці 2.1 пояснюються різні поняття з аркуша Excel. Усі ці концепції необхідно брати до уваги під час створення метамоделей. Це потрібно зробити, щоб будь-які дані, представлені в цьому шаблоні Excel, також можна було відтворити в нашому інструменті оцінки програмного забезпечення.



			Requirements Management					
ID	Description	Documentation	Interviews	Doc	Act	Consequence	Score	Questions for interviews
SG 1	Requirements are managed and inconsistencies with project plans and work products are identified.							
SP 1.1	Develop an understanding with the requirements providers on the meaning of the requirements	E document from	two months worked on requirements together	Y	F	Major	4	
SP 1.2	Obtain commitment to the requirements from the project participants		provided new requirements set based on 2 months discussions	Y	F	Moderate	4	
SP 1.3	Manage changes to the requirements as they evolve during the project		Changes are managed via formal		F	Major	4	
SP 1.4	Maintain bidirectional traceability among the requirements and the project plans and work products		requirements linked to tests however not to design. Limited to remark in trace-ability sheet but not maintained. Architecture and design poorly documented	M	F	Extreme	4	
SP 1.5	Identify inconsistencies between the project plans and work products and the requirements		lot of communication between Service contract is started including agreement on maintenance release with remaining requirements	M	F	Major	4	

Рис. 2.3. Аркуш в інструментів Excel, який підтримує аудит процесу для характеристики якості

Таблиця 2.1.

Концепції в процесі аудиту якості продукції в шаблоні засобами Excel (для оцінювання)

Concept	Description	EMF Relationship
ID	Identification	Attribute
Description	Description of the Process Group/ Process/ Practice that is being evaluated	Attribute
Documentation	A short description of the documentation relating to the practice being evaluated.	Attribute
Interviews	A note on the response received in the interview	Attribute
Doc (Documentation Score)	Represents the score on the documentation: Y: Documentation is available, compliant X: Documentation is available, but not compliant M: Documentation is missing	Enumeration
Act (Satisfaction Level)	Represents the satisfaction level of the process or practice, this is decided by the assessor at the time of the assessment. Fully satisfied: practices consistently executed, appropriate artifacts are present, no major weaknesses found. Largely satisfied: practices almost always executed and almost all appropriate artifacts present, some improvements possible. Partially satisfied: the practices are frequently (or recently) executed, some artifacts are present. Not satisfied: the practices are not executed, no artifacts are present.	Enumeration
Consequence (Consequence Level)	Represents the impact of the process on the software product quality characteristic Insignificant: Process or practice may affect the software quality characteristic in exceptional circumstances (may happen after 10 years). Minor: Process or practice might affect the software quality characteristic in 5-10 years. Moderate: Process or practice might affect the software quality characteristic in 2-5 years. Major: Process or practice could affect the software quality characteristic in 1-2 years. Extreme: Process or practice may affect the software quality characteristic within a short period.	Enumeration
Score	Represents the software Product quality assessment for each process/ practice/ goal	Enumeration

2.4. Мета-модель стандартів для полегшення проведення оцінювання

Рисунок 2.4 представляє метамодель першого рівня, яка використовується для полегшення визначення стандартів, необхідних для проведення оцінювання. Різноманітні додані класи, а також зв’язки та типи даних у них можна побачити на цьому рисунку.

Здійснимо опис класифікаторів. У таблиці 2.2 перераховані різні класифікатори ЕМП (класи, перерахування тощо) з рисунка 2.4 разом із зв’язками, які вони мають один з одним.

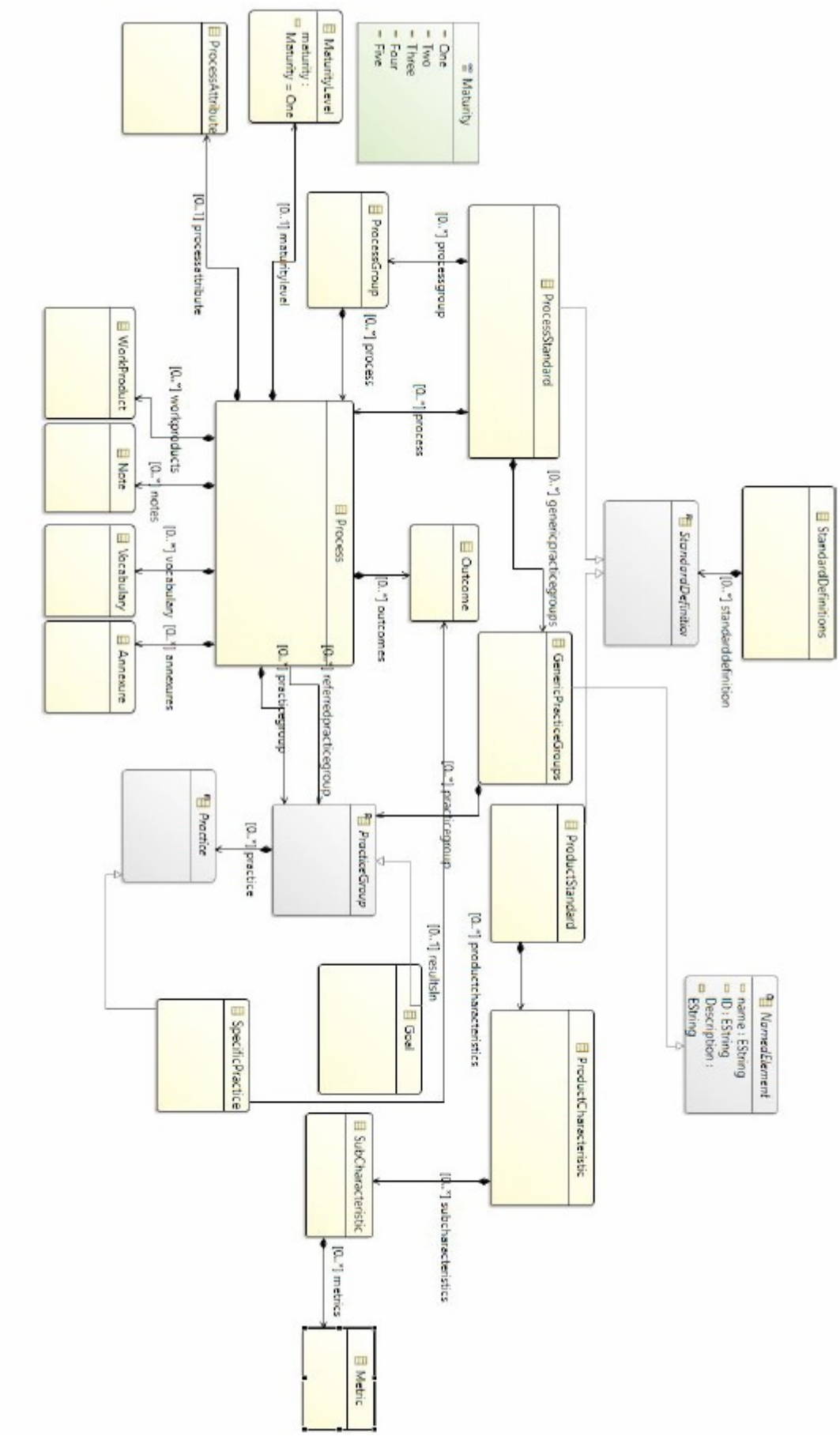


Рис. 2.4. Мета-модель стандартів

Клас StandardDefinitions є батьківським класом усієї діаграми, і його названо так, оскільки саме через цю мета-модель ми хочемо визначити всі відповідні стандарти для оцінювання якості процесу та продукції. Він містить абстрактний суперклас StandardDefinition, дочірні підкласи якого можуть представляти типи стандартів процесу та продукції, отже, це абстрактний клас.

Класи під ним є конкретними, оскільки це класи, екземпляри яких потрібно створити [2]. Крім того, для цілей цього інструменту кожне повторення стандартного визначення є або стандартом продукту, або стандартом процесу. З цієї причини класи ProcessStandard і ProductStandard є спеціалізаціями StandardDefinition.

Клас ProductStandard об’єднує стандарти продуктів, які мають характеристики та підхарактеристики, а також показники, тому відповідні класи є в метамоделі. Як видно на рисунку 2.4, він містить клас ProductCharacteristic, який містить клас SubCharacteristic, який, у свою чергу, містить клас Metric.

Клас ProcessStandard використовується для представлення стандартів процесів, таких як CMMI, TMMi та ASPICE. На даний момент є деякі аспекти ASPICE, які були включені в цю модель, але через брак часу та даних ми припинили роботу над ASPICE, але майбутні модифікації можна легко внести в мета-модель, щоб включити інші стандарти, які мають іншу структуру.

Класи GenericPracticeGroups, Process і ProcessGroup містяться в класі ProcessStandard. Як вже пояснювалося, Generic Goals — це цілі, яких необхідно досягти для кожного процесу в CMMI і TMMi, тому ми розмістили цей клас безпосередньо в Process, замість GenericPracticeGroups. Клас PracticeGroup міститься в класі Process, а також у класі GenericPracticeGroup. Він є абстрактним і є узагальненням класу Goal. Оскільки цілі в стандартах є способом групування практик, ми зробили це узагальнення. ProcessGroup спочатку створювався так, щоб нагадувати групи процесів, як описано в

ASPICE. Однак його також можна використовувати для оцінювання CMMI та TMMi, щоб групувати процеси під час визначення стандартів. Прикладом цього може бути групування процесів для CMMI-DEV (розробка) або CMMI-SVC (служба).

Клас MaturityLevel міститься в Process і використовується для надання кожному процесу рівня зрілості. Він містить атрибут зрілості Etype Maturity, який використовується для розподілу рівнів зрілості для процесів, як попередньо визначено в стандартах CMMI і TMMi. У метамоделі є Перелік (вищезгаданий Etype) під назвою Зрілість, який містить 5 літералів One, Two, Three, Four, Five, які описують 5 рівнів зрілості, визначених у CMMI та TMMi для процесів.

Такі атрибути, як Name, Description, ID, є загальними для майже всіх класів у цій метамоделі, тому клас NamedElement є супертипом для всіх класів, тому класи успадковують ці атрибути. Отже, екземпляри цих класів можуть мати власну назву, ідентифікатор та опис. Оскільки існує зв’язок узагальнення з усіма іншими класами, на рисунку 2.4 буде ще кілька стрілок, які показуватимуть ці зв’язки. Щоб уникнути цього безладу на схемі, ці зв’язки були приховані.

Клас Vocabulary використовується для опису значень нових термінів або для скорочень. Клас Note використовується для додавання деталей, вирівнювання або фону до будь-якого іншого компонента з моделі CMMI. Клас annexure призначений для додавання додаткової інформації про процес, якщо потрібно.

Клас WorkProduct походить зі стандарту CMMI і є вихідним результатом практики. Клас ProcessAttribute пов’язаний зі стандартом ASPICE та використовується як спосіб оцінки процесів у цьому стандарті. Однак цей аспект не був детально розроблений, оскільки аспекти ASPICE не були додані через часові обмеження в проекті.

Таблиця 2.2.

Класифікатори в мета-моделі стандартів

Classifier Name	EMF Relationship to Other Classes	Type of Classifier
StandardDefinitions	Composition (Owner) of StandardDefinition	Concrete Class
StandardDefinition	Supertype/Generalization of ProcessStandard and ProductStandard Composition (owned by) with StandardDefinitions	Abstract Class
ProcessStandard	Specialization of StandardDefinition, Composition (owner of) with ProcessGroup, Process and GenericPracticeGroups	Concrete Class
ProductStandard	Specialization of StandardDefinition, Composition (owner of) ProductCharacteristic	Concrete Class
ProductCharacteristic	Composition (owner of) with SubCharacteristic, Composition (owned by) with ProductStandard	Concrete Class
SubCharacteristic	Composition (owner of) Metric, Composition (owned by) with ProductCharacteristic	Concrete Class
Metric	Composition (owned by) with SubCharacteristic	Concrete Class
ProcessGroup	Composition (owned by) ProcessStandard	
Process	Composition (owned by) ProcessStandard and ProcessGroup Composition (owner of) with MaturityLevel, ProcessAttribute, WorkProduct, Note, Vocabulary, Annexure, PracticeGroup, Outcome.	Concrete Class
Outcome	Composition (owned by) Process Reference from Specific Practice	Concrete Class
GenericPracticeGroups	Composition (owned by)with ProcessStandard Composition (owner of) with PracticeGroup	Concrete Class
SpecificPractice	Specialization of Practice Reference to Outcome	Concrete Class
PracticeGroup	Composition (owned by) GenericPracticeGroup Supertype of Goal Composition (owner of) with Practice	Abstract Class
Practice	Composition (owned by) with PracticeGroup	Abstract Class
WorkProduct	Composition (owned by) with Process	Concrete Class
Note	Composition (owned by) with Process	Concrete Class
Vocabulary	Composition (owned by) with Process	Concrete Class
Annexure	Composition (owned by) with Process	Concrete Class
MaturityLevel	Composition (owned by) with Process	Concrete Class
Maturity	Etype that is used by Maturity Level	Enumeration
ProcessAttribute	Composition (owned by) with Process	Concrete Class
NamedElement	SuperType of every Class in this meta-model except MaturityLevel and ProcessAttribute	Abstract Class

Термін динамічний екземпляр пов’язаний з тим фактом, що механізм не покладається на згенеровані класи Java, а натомість використовує спеціальний підклас EObject, який підтримує всі аспекти Ecore, включаючи EAttributes і EReferences на основі визначення метамоделі .ecore. Отже, вони мають таку ж поведінку, що й екземпляри згенерованих класів Java [5]. Динамічний екземпляр — це простий спосіб створити екземпляр метамоделі

на початкових етапах розробки. Таким чином можна перевірити поведінку метамоделі [1, 16].

Загальний редактор екземплярів Ecore (приклад редактора моделей Ecore) є одним із способів створення динамічних екземплярів. У свою чергу, загальний редактор називається Sample Reflective Model Editor і може редагувати екземпляри моделі Ecore. Це об’єктні графи, які забезпечують деревовидне редагування основної ієрархічної структури елементів, що відповідають EClass, знайденому у відповідній моделі Ecore [5]. Є навіть команди для створення, видалення, копіювання та вставлення елементів, а також аркуш властивостей для редагування деталей. За межами редактора дерева знаходиться вікно властивостей, яке використовується для редагування.

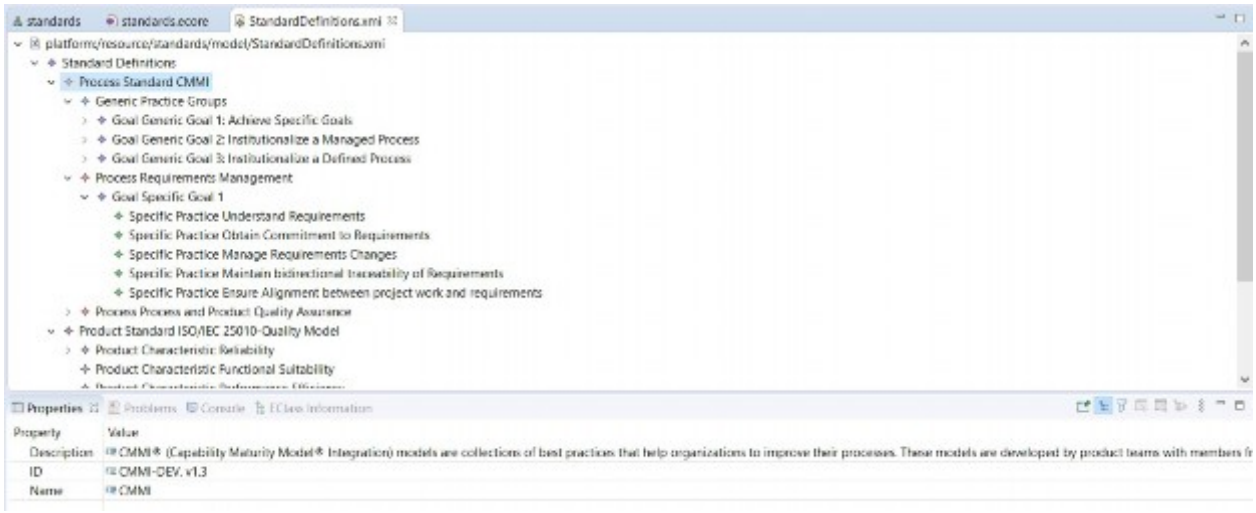


Рис. 2.5. Динамічний екземпляр стандартів, що показує редактор і вікно властивостей

Приклад редактора та перегляду властивостей показано на рисунку 2.5. Кореневий вузол представляє файл, який містить стандартне визначення, у цьому випадку стандарт процесу CMMI, а пізніше стандарт продукту — модель якості ISO/IEC 25010. Стандарт процесу CMMI містить загальні практичні групи та різні елементи процесу (керування вимогами, процес і забезпечення якості продукції). Елементи процесу містять елементи цілі

(наприклад, конкретна ціль 1 у розділі «Управління вимогами»), а в них — елементи конкретної практики (розуміти вимоги, отримати зобов'язання щодо виконання вимог тощо). У стандарті на продукцію можна побачити ряд характеристик продукту, а саме надійність і функціональну придатність. Оскільки вибрано Process Standard, його атрибути (успадковані від NamedElement) відображаються у вікні Properties. Щоб відредагувати його значення, потрібно просто натиснути на стовпець Значення.

Щоб створити нові елементи, можна клацнути правою кнопкою миші на батьківському (майбутньому) і вибрати New Child і тип елемента, який потрібно додати. Показано лише допустимі типи елементів, напр. можна додати групи загальних практик або процес до групи процесів, але не процес до групи загальних практик. Ця «законність» базується на мета-моделі та різних зв'язках між класифікаторами, як показано на рисунку 2.4 і в таблиці 2.2. Новий елемент вставляється внизу списку дочірніх елементів під батьківським, тому вам, можливо, доведеться перемістити його за допомогою перетягування. Якщо ви хочете розмістити новий елемент у середині списку дочірніх елементів, ви також можете клацнути правою кнопкою миші на дочірньому елементі над тим місцем, де ви хочете створити новий елемент, і вибрати «Новий однорідний» і тип елемента [5].

2.5. Представлення мета-моделі другого рівня для процесу оцінювання якості програмного забезпечення

На рисунку 2.6 представлена мета-модель другого рівня, яка була створена для того, щоб мати можливість визначити оцінювання, які необхідні для проведення оцінювання.

Опис класифікаторів

У таблицях 2.3 і 2.4 можна представлено різні класифікатори в мета-моделі (рис. 2.6) разом із відповідними співвідношеннями ЕМП.

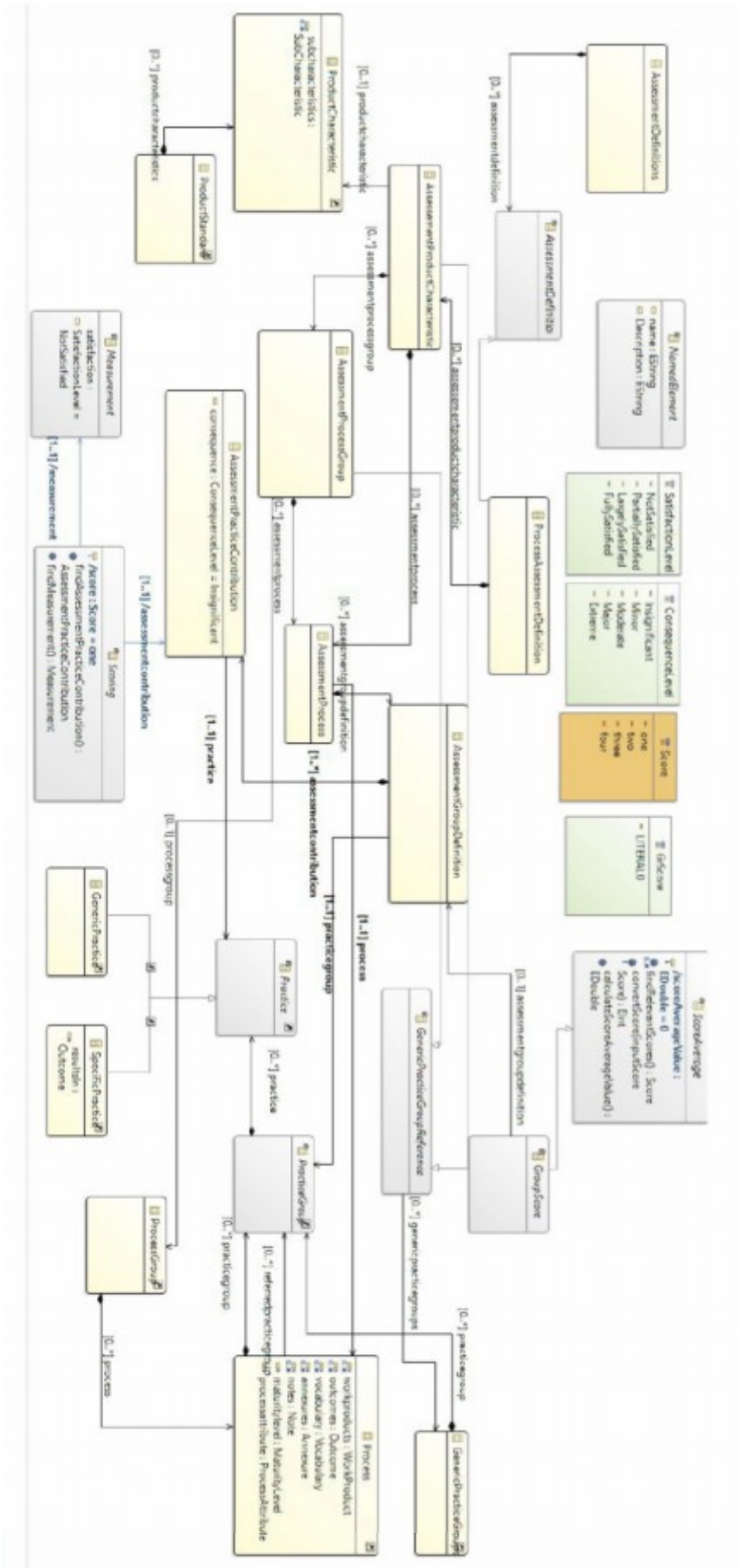


Рис. 2.6. Мета-модель другого рівня для оцінювання якості ПЗ

Клас AssessmentDefinitions є батьківським класом цієї метамоделі. У ньому міститься клас AssessmentDefinition, який є абстрактним і містить усі визначення оцінки процесу під ним. Клас ProcessAssessmentDefinition використовується для визначення всіх оцінок. Наприклад, якщо для компанії наближається оцінка CMMI/TMMi, а за кілька тижнів або місяців до неї команда оцінювання вирішує, які процеси вибрати зі стандартів, оскільки вони вважають, що саме вони впливають на конкретні характеристика продукту (і ця характеристика продукту є тією, на яку клієнт хоче орієнтуватися). Цей клас є спеціалізацією AssessmentDefinition.

У класі ProcessAssessmentDefinition міститься Assessment ProductCharacteristic, тому що на цьому рівні під час визначення оцінки оцінювач та його команда повинні вибрати характеристику продукту, яку компанія-клієнт сказала їм, що вони хочуть оцінити. Цей клас має посилання на ProductCharacteristic, який міститься в класі ProductStandard.

Крім того, клас AssessmentProductCharacteristic містить Assessment Process і AssessmentProcessGroup. Оцінювач може або безпосередньо додавати процеси після вибору характеристики продукту, або може спочатку додавати групи процесів, а потім процеси під ними. Групи процесів додаються, щоб можна було визначити, на якій основі додаються наступні процеси. Наприклад, «Процеси в CMMI-DEV + процеси в CMMI-SEV, що впливають на надійність». У цьому прикладі ми групуємо процеси, які впливають на якість продукту, характеристику надійності.

Клас AssessmentProcessGroup є узагальненням GenericPracticeGroup Reference. Цей клас, у свою чергу, відноситься до класу GenericPractice Groups, який було завантажено з метамоделі стандартів, і особа, яка визначає стандарти, може використовувати цей клас для додавання визначених у них загальних цілей.

Клас AssessmentProcess використовується для додавання процесів зі стандартів. Він містить клас AssessmentGroupDefinition. Він містить посилання на клас Process, який було завантажено з попередньої мета-моделі

Standards. Клас AssessmentGroupDefinition є середнім класом, який використовується для посилання на «практичні групи». Вони були названі так тому, що в стандартах, з якими ми маємо справу, практики згруповані разом під цілями. Він також містить клас AssessmentPracticeContribution, який має атрибут, який називається наслідок, Etype ConsequenceLevel. Цей клас використовується командою, що визначає оцінку, щоб визначити, які рівні наслідків має конкретна практика на характеристику якості продукту. Цей етап «Визначення оцінювання» було пояснено в попередніх розділах. Цей клас має посилання на Practice, яке міститься в класі PracticeGroup. Клас PracticeGroup міститься в класі GenericPracticeGroups. По суті, «Практичні групи» — це «Цілі», визначені в стандартах. Цей клас має посилання на PracticeGroup, щоб користувач під час визначення оцінювання міг шукати цілі. Цей клас містить Practice, який є узагальненням SpecificPractice та GenericPractice, усі три з яких були завантажені з попередньої мета-моделі.

Клас підрахунку балів – це той клас, у якому оцінювач обчислюватиме бали індивідуальної практики, він також використовується у фактичній мета-моделі, яка є нашою мета-моделлю третього й останнього рівня. У ньому є посилання на класи AssessmentPracticeContribution і Measurement, оскільки для розрахунку індивідуальних балів практики необхідно заздалегідь визначити як рівні наслідків для практик, так і рівні задоволеності в результаті оцінювання. Клас Measurement має буквальне задоволення Etype SatisfactionLevel, а клас AssessmentPracticeContribution має буквальний наслідок Etype ConsequenceLevel. Клас Scoring має дві операції findAssessmentPracticeContribution і findMeasurement, які пояснюються далі в цьому розділі. Він також має літеральний бал Etype Score.

Перерахування SatisfactionLevel — це те, що використовуватиметься під час фактичного оцінювання, тобто в наступній метамоделі, але ми визначили його на цьому рівні, оскільки саме тут визначаються оцінки. Це те, що оцінювач має ввести під час проведення оцінювання, і він може вибрати між 4 рівнями відповідно до матриці надійності, показаної на рис. 1.7.

Таблиця 2.3.

Опис класів в мета-моделі другого рівня

Classifier Name	EMF Relationship to Other Classes	Type of Classifier
AssessmentDefinitions	Parent class of the meta-model Composition (owner of) with AssessmentDefinition	Concrete Class
AssessmentDefinition	Owned by AssessmentDefinition	Abstract Class
ProcessAssessmentDefinition	Generalization of AssessmentDefinition Composition (owner of) AssessmentProductCharacteristic	Concrete Class
AssessmentProductCharacteristics	Composition (owner of) AssessmentProcessGroup Reference to ProductCharcteristic Composition (owner of) AssessmentProcess	Concrete Class
GenericPracticeGroupsReference	Generalization of AssessmentProcessGroup Generalization of AssessmentProductCharacteristic	Abstract Class
AssessmentProcessGroup	Specialization of GenericPracticeGroupsReference Composition (owner of) AssessmentProcess	Concrete Class
ProcessGroup	Reference from AssessmentProcessGroup Composition (owner of) Process	Concrete Class
Process	Composition(owner of) PracticeGroup Reference from AssessmentProcess	Concrete Class
AssessmentProcess	Reference to Process Composition (owner of) AssessmentGroupDefinition Composition (owned by) AssessmentProductCharacteristic	Concrete Class
ProductCharacteristic	Reference from AssesmentProductCharacteristic	Concrete Class
ProductStandard	Owner of ProductCharacteristic	Concrete Class
AssessmentGroupDefinition	Owner of AssessmentPracticeContribution Reference from GroupScore	Concrete Class
AssessmentPracticeContribution	Composition (owned by) AssessmentGroupDefinition Reference to Practice Reference from Scoring	Concrete Class

Перерахування ConsequenceLevel також було додано, оскільки часто використовуються рівні наслідків для своїх оцінок. Він має 5 рівнів, і на відміну від попереднього переліку, він фактично використовується на цьому рівні під час визначення оцінок. Команда, яка визначає оцінки, призначає рівень наслідків для окремих практик. У цьому переліку є 5 літералів (незначний, незначний, помірний, великий, екстремальний), які взяті з матриці оцінки (рис. 1.7).

Таблиця 2.4.

Опис класифікаторів в мета-моделі другого рівня

Classifier Name	EMF Relationship to Other Classes	Type of Classifier
Scoring	Reference to AssessmentPracticeContribution and Measurement	Concrete Class
Measurement	Reference from Scoring	Concrete Class
Practice	Generalization of SpecificPractice and GenericPractice Owned by PracticeGroup	Abstract Class
PracticeGroup	Reference from AssessmentGroupDefinition	Abstract Class
GenericPracticeGroups	Reference from GenericPracticeGroupReference	Concrete Class
GenericPractice	Specialization of Practice	Concrete Class
SpecificPractice	Specialization of Practice	Concrete Class
SatisfactionLevel	Etype used in Measurement	Enumeration
ConsequenceLevel	Etype that is used in AssessmentPracticeContribution	Enumeration
Score	Etype that is used by Scoring	Enumeration
GrScore	Etype used in ASPICE standard	Enumeration
GroupScore	Specialization of ScoreAverage	Abstract Class
ScoreAverage	Supertype of Groupscore	Abstract Class
NamedElement	Supertype (Generalization) of every class in this meta-model	Abstract Class

Клас GroupScore є абстрактним і містить посилання на Assessment GroupDefinition, а також є спеціалізацією класу ScoreAverage, який містить 3 операції/функції, які використовуються для обчислення балів окремих практик і процесів. Він також містить літерал scoreAverageValue Etype EDouble. Отже, підрахунок оцінок використовується для отримання остаточної оцінки кожної практики, визначеної в оцінках. Це відповідно до матриці скорингу, як показано на рисунку 1.7. У цьому переліку є 4 значення один, два, три, чотири, які є балами з оціночної матриці. Значення балів розраховуються на основі комбінації рівнів наслідків і рівнів задоволення.

Висновки до розділу

В даному розділі представлено підхід, застосований для розробки нових метамоделей, необхідних для створення програмного інструменту оцінювання. Представлено метамоделі, з докладними поясненнями щодо складових елементів кожної метамоделі та їх взаємозв'язків. Окремо розглянуто функціональні можливості, інтегровані у класи.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МЕТОДОЛОГІЇ МОДЕЛЬ-БАЗОВАНОГО ПІДХОДУ ДО АУДИТУ ТА ТЕСТУВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Операції та обмежені відношення в мета-моделі

В цьому розділі будуть досліджені різні операції, які були реалізовані в цій мета-моделі (розділ 2), а також розглянемо поняття обмеженого відношення. Обмежене відношення використовується для вибору конкретних екземплярів класу, які будуть використані. На рисунку 3.1 наведено простий приклад обмеженого відношення в мета-моделі. Покупець може придбавати лише товари, які доступні в магазині, де він здійснює покупку. Отже, відношення "товари магазину" обмежене відношенням "магазин".

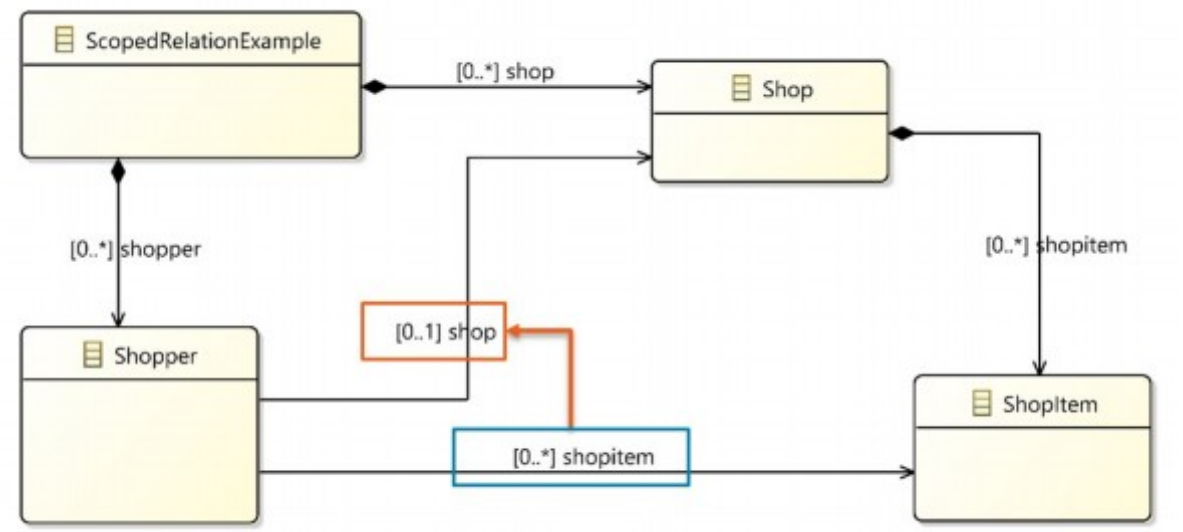


Рис. 3.1. Приклад використання Scoped відношення

На рисунку 3.2 представлено дерево Ecore метамоделі прикладу "Покупець" з рисунка 3.1. Розширення аннотації зі словом "scoped" пов'язує обмеження "scoped" (яке попередньо запрограмовано в EcoreExt) з класом "Покупець". Оскільки відношення "shopItem" обмежене відношенням "shop", до посилання "shopItem" всередині класу "Покупець" необхідно додати

EAnnotation. Ecore EAnnotation вказує, які обмеження перевіряти для класу. Це список обмежень, куди потрібно додати "scoped", щоб обмеження "scoped" було оцінено для класу "Покупець".

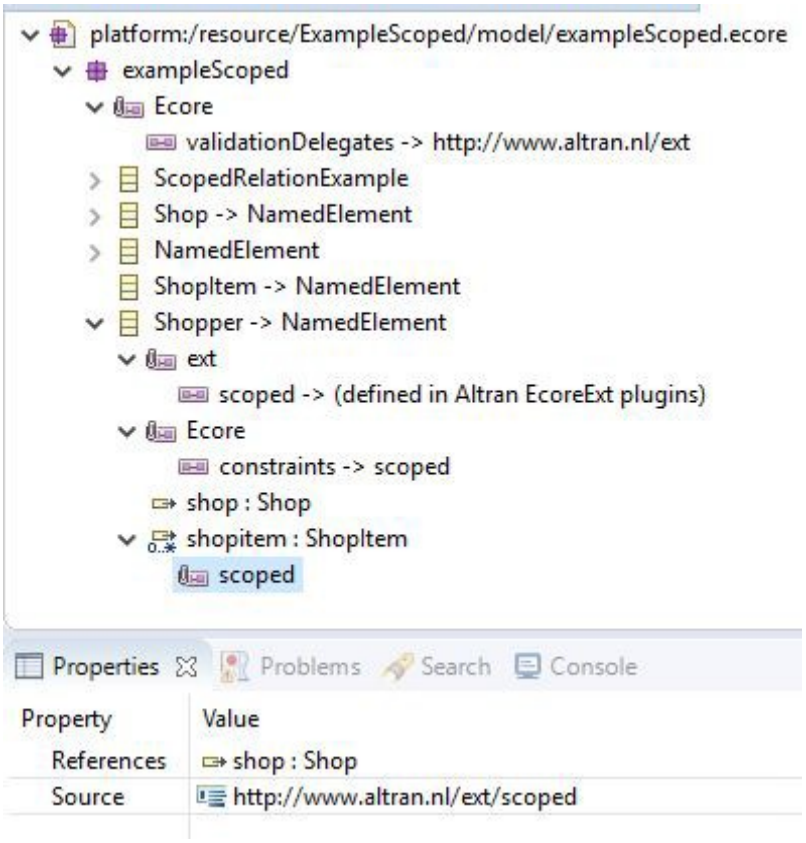


Рис. 3.2. Перегляд Ecore прикладу Scoped Shop

Це стало можливим завдяки плагіну, який дозволяє Ecore (мова мета-моделювання, яка використовується для визначення структури інших моделей) обробляти такі обмежені відношення. У наступній частині пояснюється, як обмежене відношення присутнє в різних відношеннях у метамоделі оцінки.

На рисунку 3.3 представлено дерево Ecore обмежених відношень, використаних у метамоделі оцінки.

Практика відношення в AssessmentPracticeContribution визначається практикоюgroup у AssessmentGroupDefinition, як показано на рисунку 3.3. Відношення області дії використовується в цьому класі для вибору

конкретних практик, які були визначені в стандарті на мета-моделі попереднього рівня.

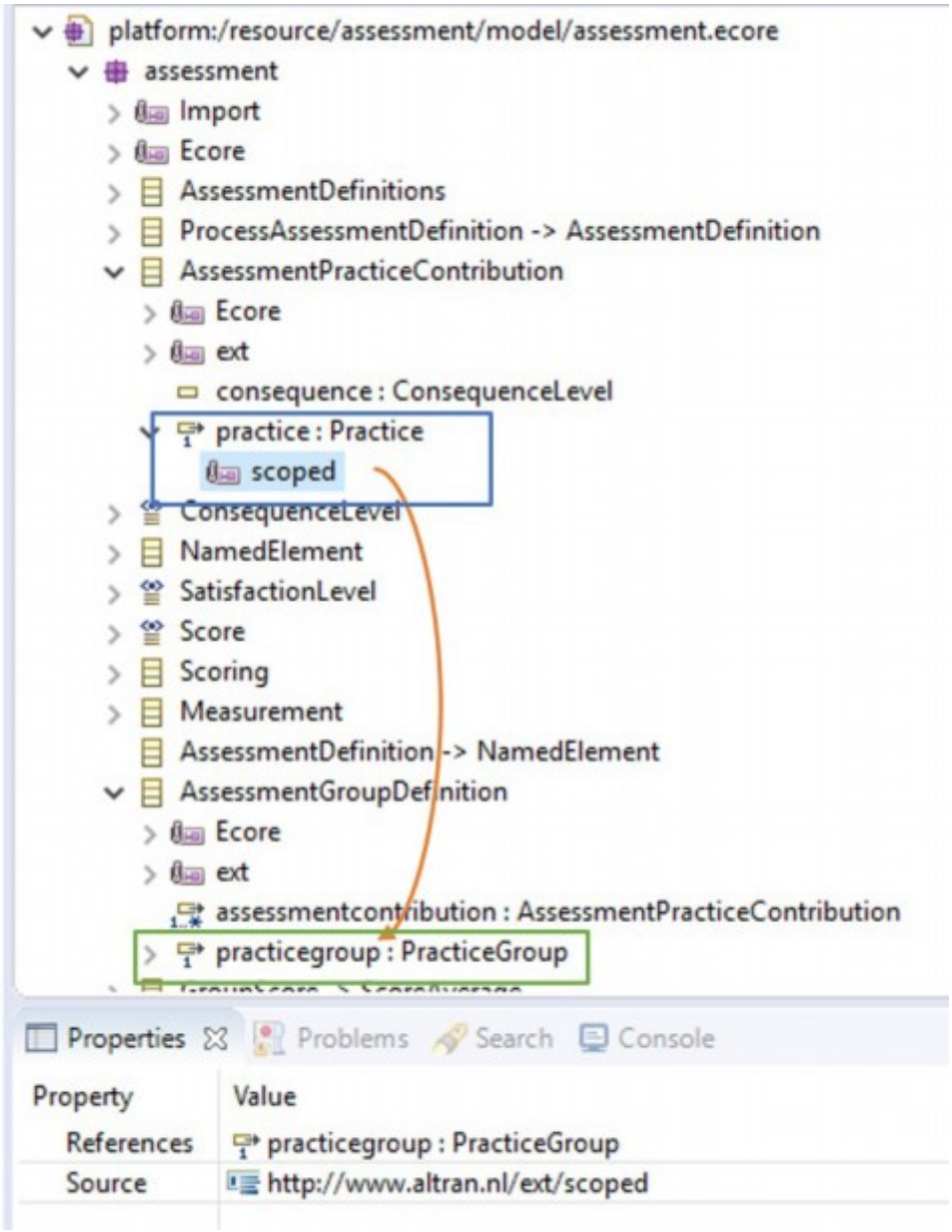


Рис. 3.3. Перегляд Ecore відношення Scoped у класі AssessmentPracticeContribution

На рисунку 3.4 ліворуч наведено стандарт CMMI, визначений у екземплярі метамоделі стандартів, а праворуч — екземпляр метамоделі оцінювання, де було визначено оцінювання. Лише практики, які були визначені в екземплярі стандарту в рамках Goal PPQA SG1, можуть бути обрані як практики в екземплярі оцінювання.

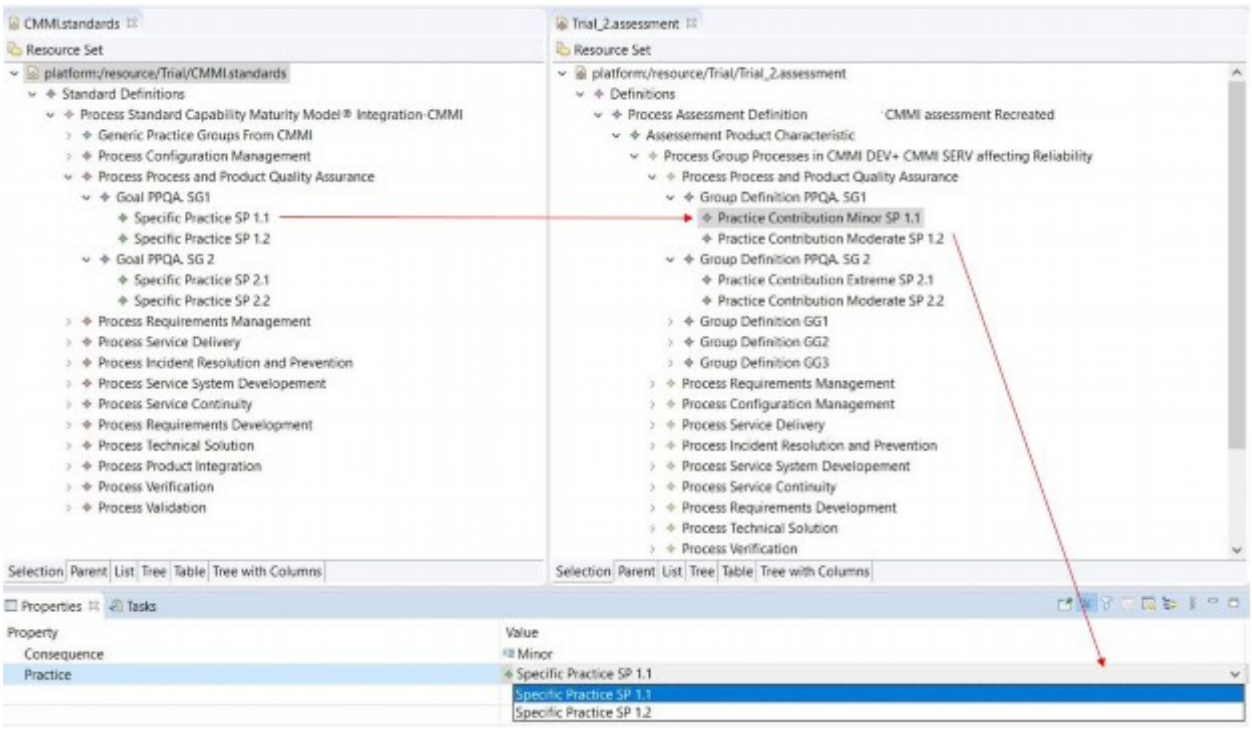


Рис. 3.4. Екземпляр оцінювання, що показує відношення області дії, що використовується для елемента AssessmentPracticeContribution

Відношення області дії використовується в класі AssessmentGroupDefinition, як показано на рисунку 3.5, для вибору з практичних груп, які були визначені в стандарті на мета-моделі попереднього рівня. Практична група в AssessmentGroupDefinition обмежена наступними посиланнями:

• genericpracticegroups у конкретних підтипах: GenericPracticeGroup Reference (AssessmentProductCharacteristic або AssessmentProcessGroup)

• процес у AssessmentProcess

На рисунку 3.6 ліворуч наведено стандарт CMMI, визначений у екземплярі метамоделі стандартів, а праворуч — екземпляр метамоделі оцінювання, де було визначено оцінювання. Лише цілі/групи практики, які були визначені в екземплярі стандартів у рамках процесу надання послуг, можуть бути обрані як PracticeGroups у екземплярі оцінювання.

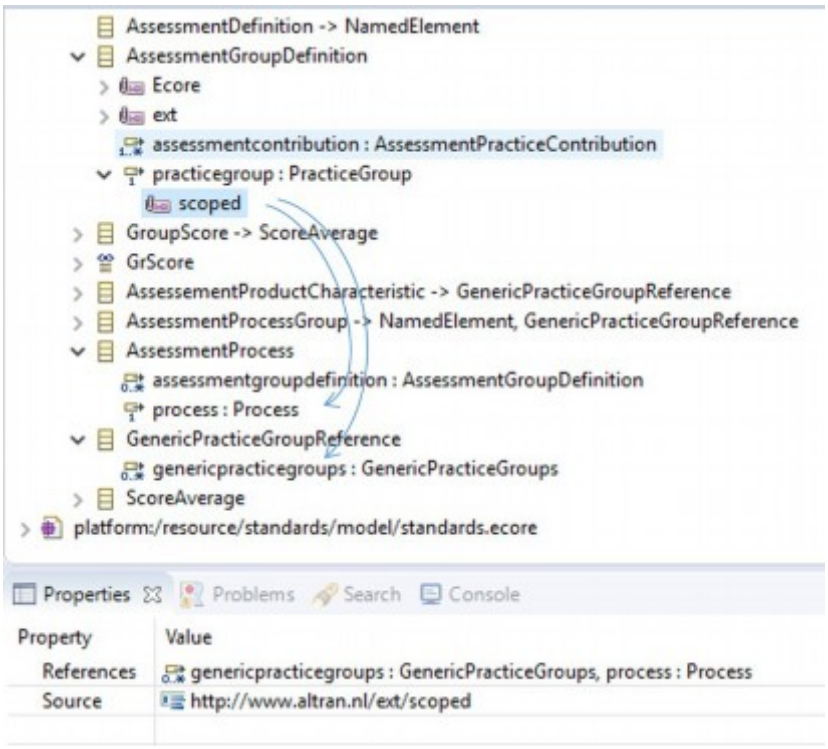


Рис. 3.5. Перегляд Ecore відношення Scoped у класі AssessmentGroupDefinition

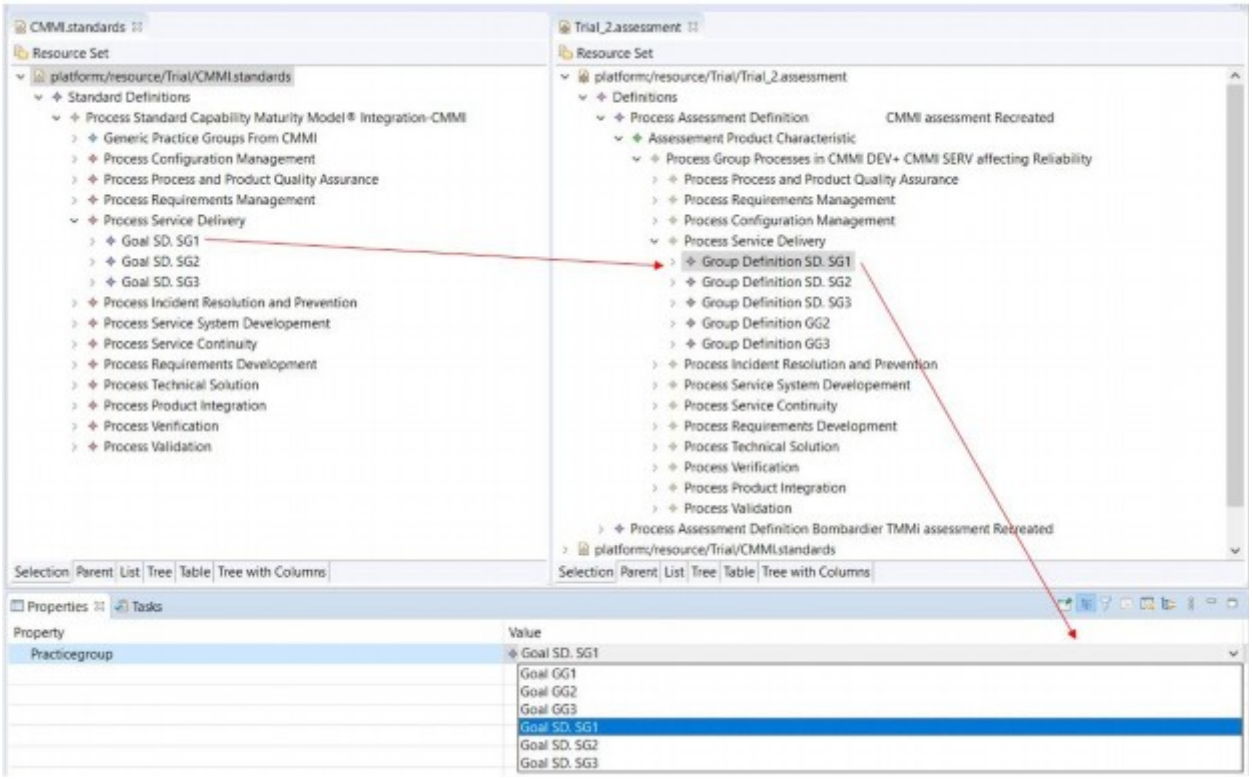


Рис. 3.6. Екземпляр оцінки під час виконання, що показує відношення області дії, що використовується для елемента AssessmentGroupDefinition

Дальше розглянемо операції обчислення в класах Scoring і Score Average. Клас Scoring має дві операції: findAssessmentPracticeContribution і findMeasurement, які пояснюються далі в цьому розділі. Тіло обох цих операцій є «нульовим», яке пізніше буде перевизначено у фактичній мета-моделі, як показано на рисунку 3.7.

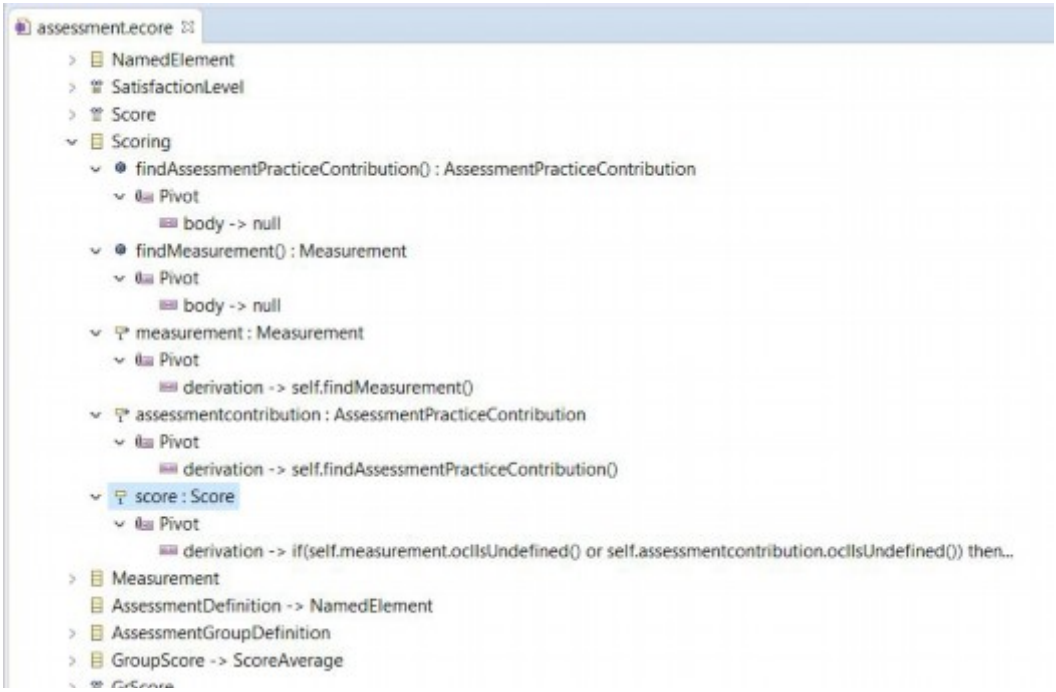


Рис. 3.7. Клас Scoring в оцінюванні в редакторі дерева Ecore

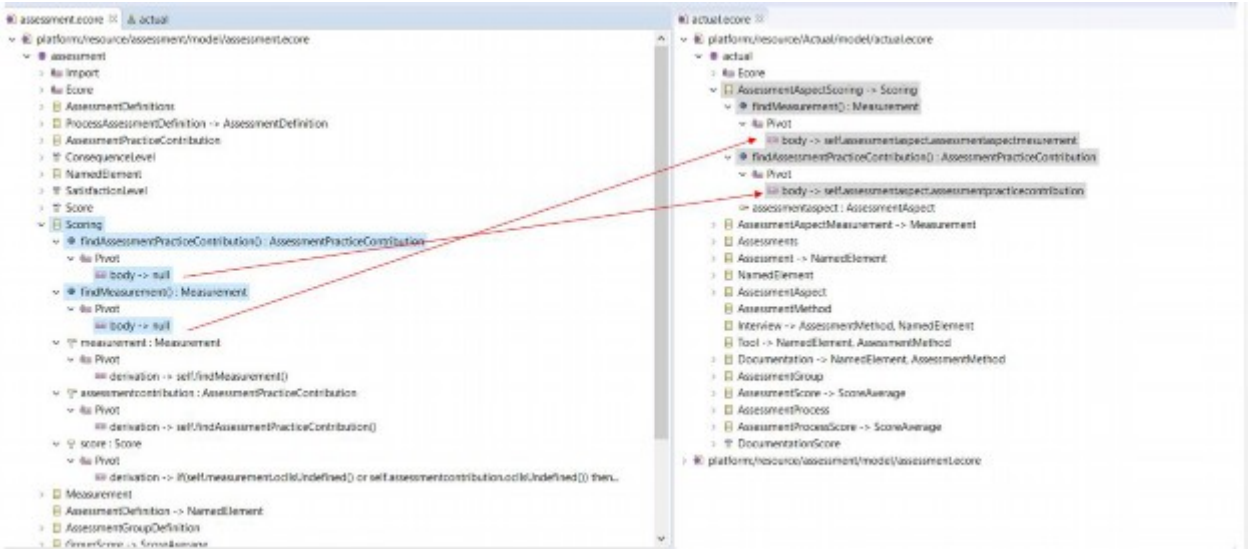


Рис. 3.8. Клас Scoring і його перевизначена форма у фактичній мета-моделі

У класі Scoring є 3 EStructuralFeatures (як показано на рис. 3.7):

• вимірювання – це атрибут Eattribute Etype AssessmentPractice Contribution;

• AssessmentContribtion - це EReference Etype Measurement;

• оцінка – це EReference Etype Score.

Оцінка розраховується на основі рівня задоволення від вимірювання та рівня наслідків AssessmentPracticeContribution. Пошук відповідного вимірювання та відповідного внеску оцінки залежить від того, де в метамоделях знаходяться конкретні підтипи оцінки. Таким чином, у фактичній мета-моделі тіло операцій findMeasurement() і findAssessmentPracticeContribution() потрібно перевизначити (або навіть можна визначити лише в конкретних підтипах Scoring у Actual).

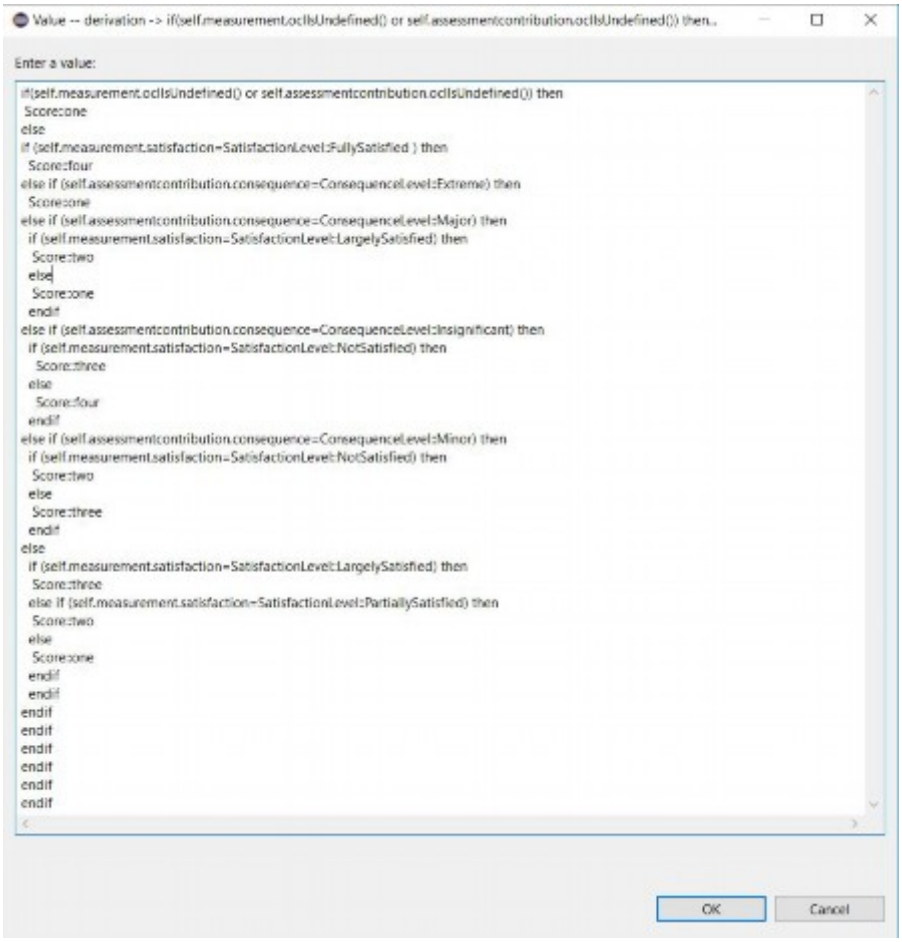


Рис. 3.9. Значення оцінки, що міститься в опорній частині оцінки в класі оцінювання

На рисунку 3.9 можна побачити, як розраховується оцінка для окремих вправ за допомогою синтаксису OCL. Значення обчислюється за допомогою ConsequenceLevel і SatisfactionLevel відповідно до матриці оцінки (рис. 1.7).

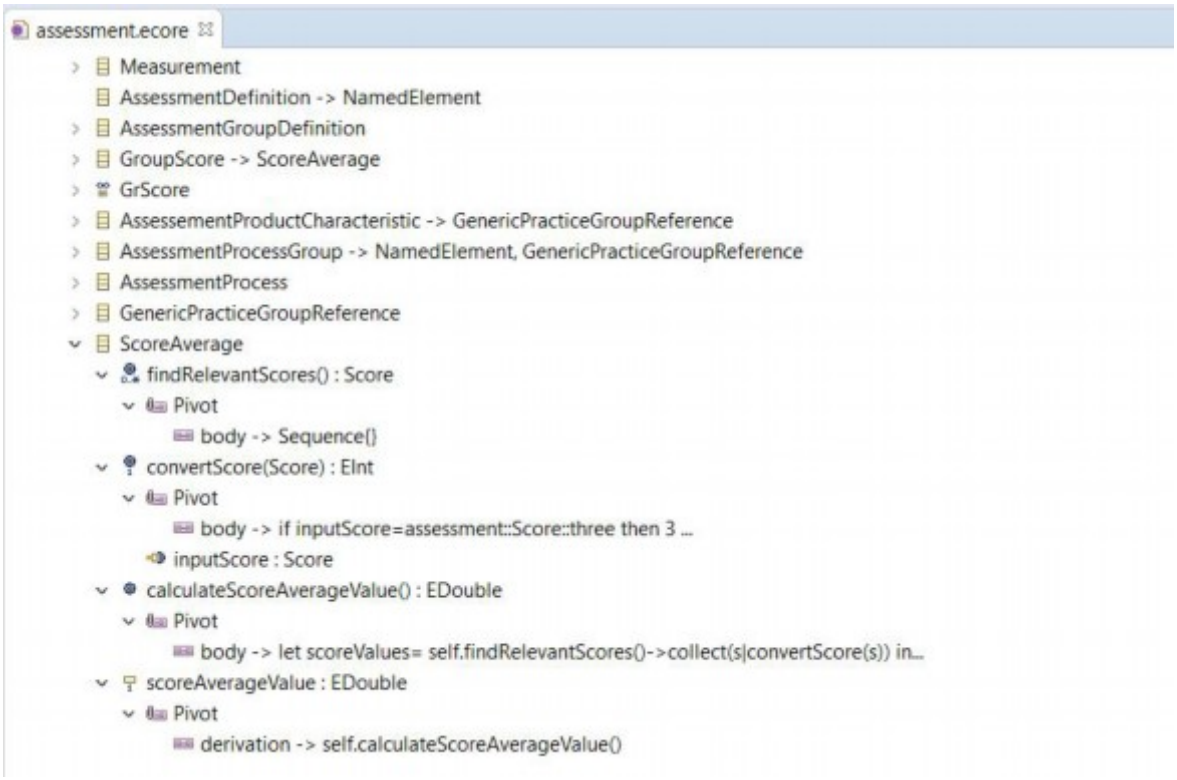


Рис. 3.10. Клас ScoreAverage в оцінюванні в редакторі дерева Ecore

На рисунку 3.10 можна побачити різні функції в класі. У тілі функції findRelevantScores() знаходиться Sequence, який є попередньо визначеним типом колекції в OCL. Ця функція має Etype Score. По суті, ця функція відповідає за пошук релевантних балів, з яких потрібно обчислити середнє значення балів. Він має два використання залежно від контексту:

• Щоб визначити, які бали враховувати під час розрахунку індивідуальних балів процесу.

• Щоб визначити, які бали враховувати під час розрахунку загальної оцінки процесу.

Далі функція convertScore() має EParameter під назвою inputScore або Etype Score (який раніше було визначено в метамоделі). Ця функція призначена для перетворення результатів індивідуальної практики з Etype

Score в Etype Eint або цілі числа. Тіло функції, щоб зрозуміти, як вона працює, можна побачити на рисунку 3.11.

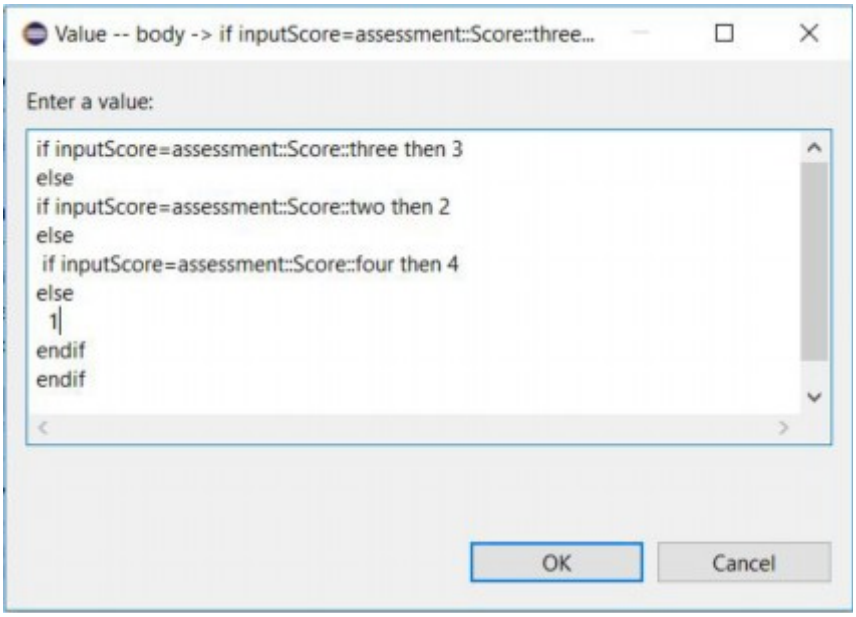


Рис. 3.11. Тіло функції convertScore

Далі йде функція calculateScoreAverage(). Як випливає з назви, вона використовується для обчислення середнього значення балу, і вона використовує функцію convertScore для перетворення відповідних балів спочатку в цілі числа. Тіло функції, написане на OCL, можна побачити на рисунку 3.12. Це обчислення цієї функції за замовчуванням.

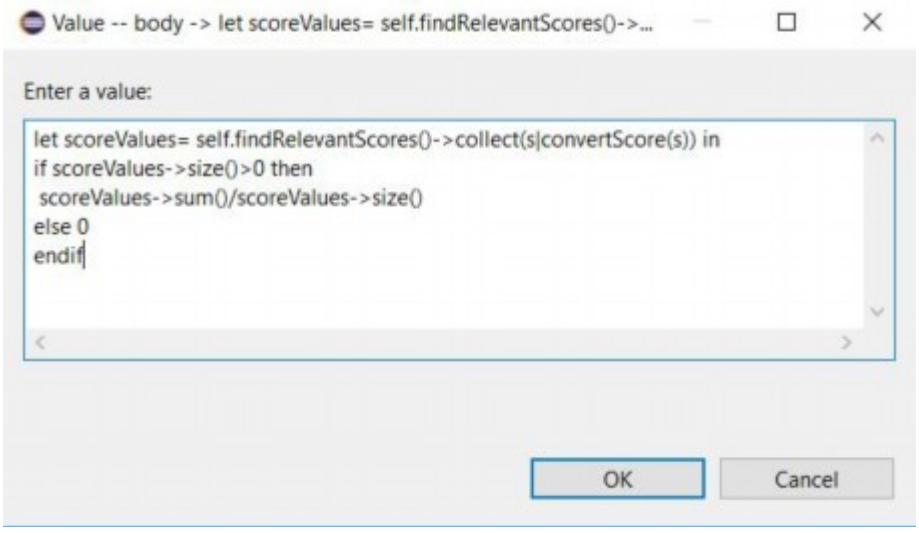


Рис. 3.12. Тіло функції calculateScoreAverageValue

На рисунку 3.13 можна побачити два види, ліворуч і праворуч. Вони являють собою оцінку та фактичну мета-моделі, відповідно, у перегляді дерева Ecore. У кінцевій мета-моделі третього рівня класи AssessmentScore та AssessmentProcessScore є спеціалізаціями (нащадками) класу ScoreAverage. Завдяки цьому вони успадковують функції класу. На малюнку можна побачити, як тіла обох функцій перевизначені у фактичній мета-моделі, як показано червоними стрілками.

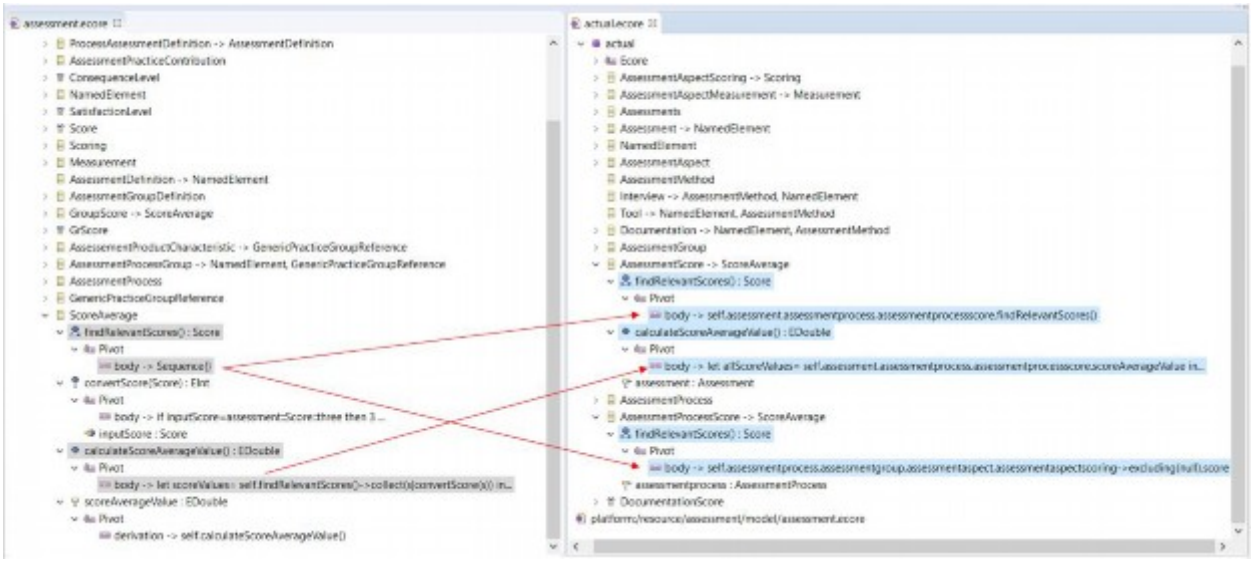


Рис. 3.13. Клас ScoreAverage та його перевизначена форма у фактичній мета-моделі

Існує різниця в обчисленні між AssessmentProcessScore, який усереднює окремі бали (використовуючи обчислення за замовчуванням у функціїkalculateScoreAverageValue()), і AssessmentScore, який усереднює AssessmentProcessScores, що потребує перевизначення тіла CalculateScore AverageValue().

3.2. Тестування екземпляра мета-моделі

Здатності відображення EMF, які використовуються в динамічних екземплярах, є цінними для грубої оцінки метамоделей, але щоб мати

можливість створювати та адаптувати інструменти оцінки та отримати доступ до більш розширених функцій, таких як співвідношення масштабів, нам потрібно працювати з програмні версії моделей. Для їх тестування потрібно запустити отриману програму як те, що в Eclipse називається екземпляром середовища виконання. На основі файлів .genmodel можна створити код Java. EMF може генерувати плагіни, які надають майстри для створення нових екземплярів моделі та дозволяють користувачам вводити інформацію про модель [20]. На цьому етапі проекту, оскільки була створена друга модель, потрібно було створити екземпляр часу виконання, щоб перевірити поведінку класів і операцій у них. На рисунку 3.14 можна побачити загальний вигляд екземпляра мета-моделі оцінювання під час виконання. Це приклад оцінки CMMI для характеристики продукту «Надійність», що визначається.

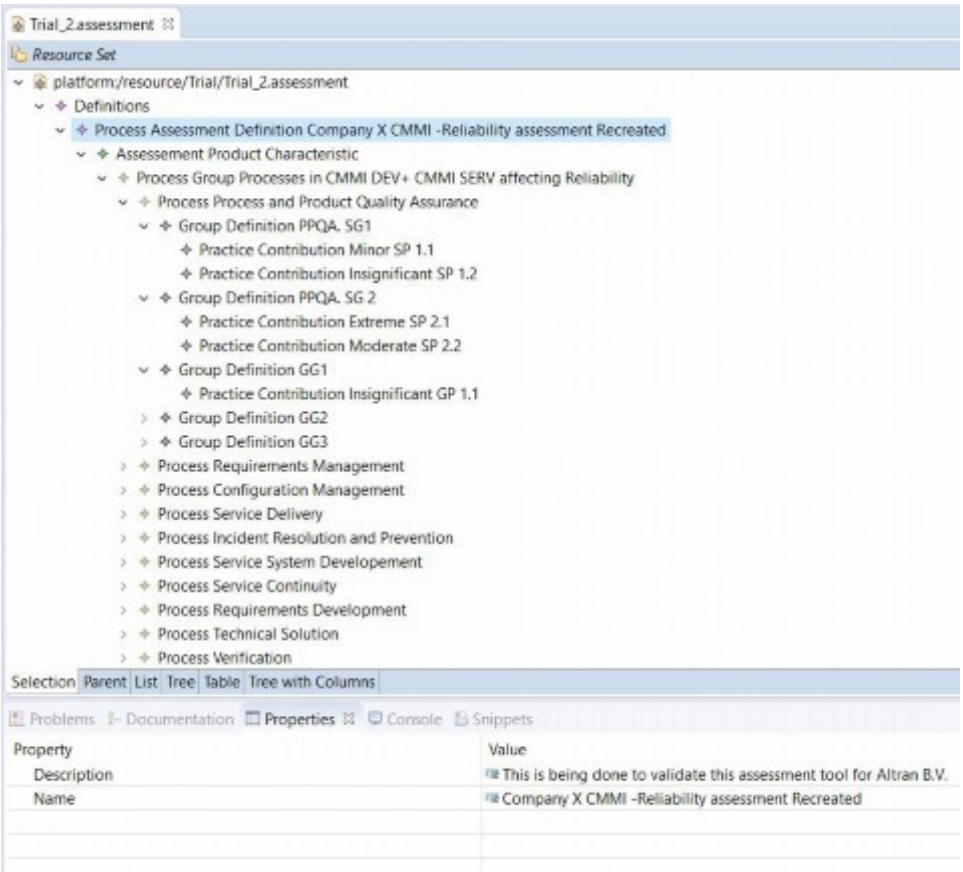


Рис. 3.14. Екземпляр оцінювання під час виконання

На рисунку 3.4 показано, як можна вибрати практики під час визначення оцінки. Це стало можливим завдяки тому, що ці практики вже визначено в екземплярі стандарту, а також завдяки використанню відношення області видимості, як пояснено вище. Під час визначення оцінки наступним кроком буде визначення рівня наслідків (або впливу), який кожна конкретна практика має на характеристику продукту, що оцінюється. У цьому випадку характеристикою продукту є надійність.

Цей крок можна побачити на рисунку 3.15. Це велика перевага порівняно з інструментом Tummalapalli, оскільки рівні наслідків визначаються заздалегідь під час створення шаблону оцінювання. Оскільки існує чітке розмежування між ресурсами, стає легше обмежити такі можливості. Через це експерт/команда з оцінювання не може змінити рівні наслідків під час проведення оцінки на пізнішому етапі. Це виключить будь-які шанси отримати необ’єктивні результати.

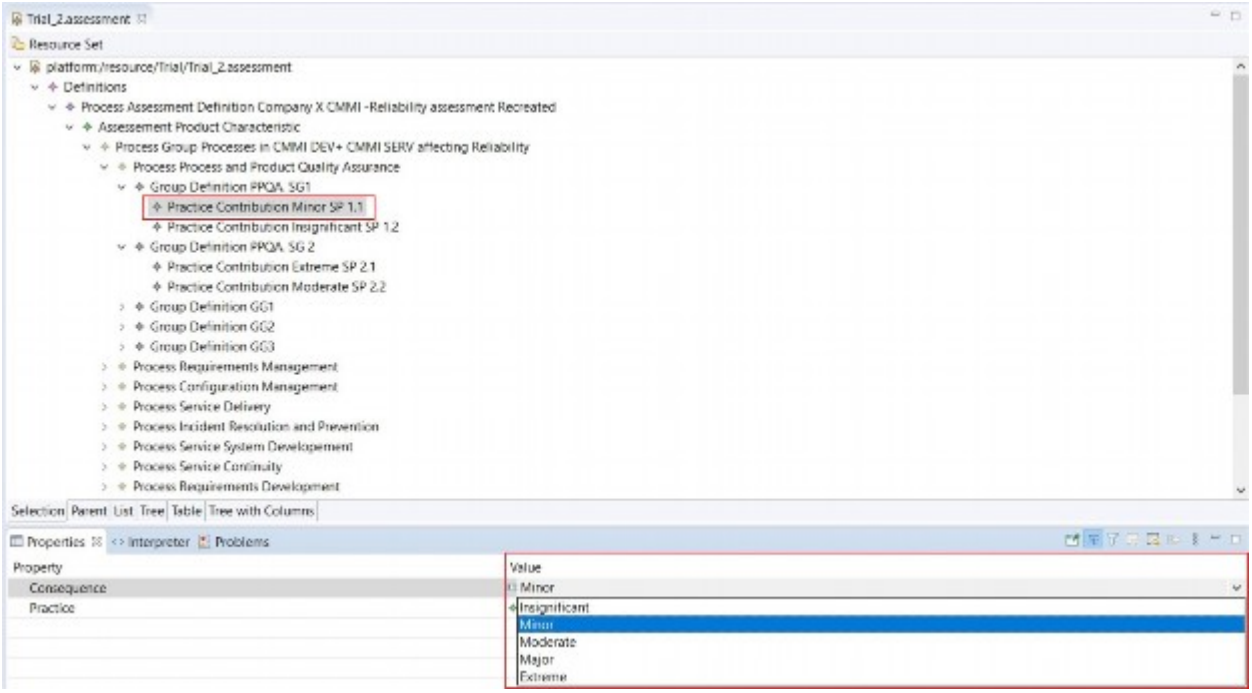


Рис. 3.15. Екземпляр оцінювання під час виконання, що показує можливість вибору рівнів наслідків для конкретної практики

На рисунку 3.15 можна побачити можливість вибрати, який рівень наслідків, спеціальна практика 1 має на надійність. Ця конкретна конкретна практика належить до конкретної цілі 1 процесу процесу та забезпечення якості продукції.

3.3. Перевірка адекватності мета-моделей

В цьому розділі розглядається перевірка метамоделей, які були побудовані. Дві попередні оцінки, що були проведені, використовуються як контрольні випадки для підтвердження мета-моделей та інструменту оцінки. Для перевірки інструменту проводиться оцінка CMMI та оцінка TMMi для конкретної якості продукту.

3.3.1. Випадок використання стандарту оцінки Capability Maturity Model Integration

На рисунку 3.16 показано оцінку CMMI (Capability Maturity Model Integration), яку використано для характеристики надійності. Як видно на рисунку, для оцінки було обрано 12 областей процесу, деякі з CMMI-SEV, а інші з CMMI-DEV. Області процесу вказані в першому рядку аркуша Excel. Їх було вибрано, оскільки команда з оцінки вирішила, що ці зони процесу є тими, які впливають на надійність. Областями процесу є управління вимогами (REQM), забезпечення якості процесів і продукції (PPQA), управління конфігурацією (CM), надання послуг (SD), вирішення та запобігання інцидентам (IRP), розробка системи обслуговування (SSD), безперервність обслуговування (SC)), розробка вимог (RD), технічне рішення (TS), інтеграція продукту (PI), верифікація (VER) і валідація (VAL). Як видно в нижньому правому куті малюнка, загальний бал становить 3,24. За винятком REQM, усі інші бали практики окремих областей процесу були розмиті, щоб захистити конфіденційні дані клієнтів. Оцінка процесу або можливості REQM становить 3,24.

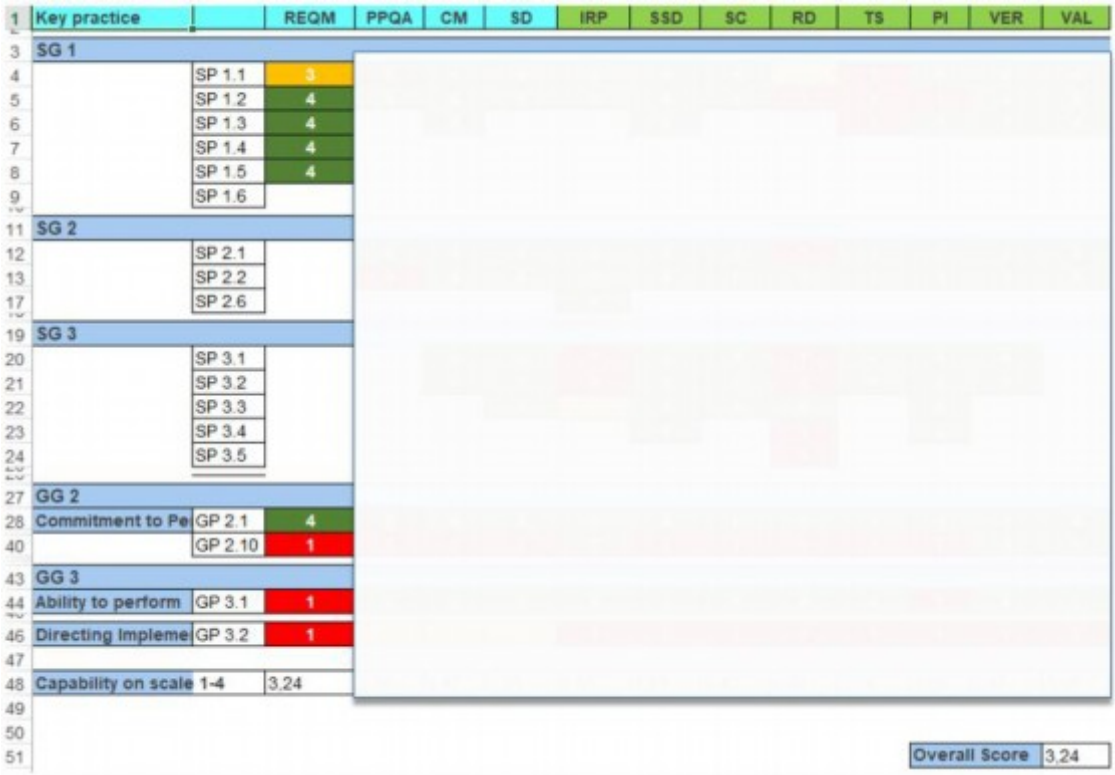


Рис. 3.16. Таблиця для оцінки CMMI

На рисунку 3.17 можна побачити екземпляр часу виконання всіх трьох мета-моделей, а також різні екземпляри, які були потрібні для завершення оцінки CMMI і, таким чином, перевірки новий інструмент. Раніше компанія Altran проводила оцінку CMMI для характеристик продукту щодо надійності. Оцінку було проведено повторно за допомогою інструменту, і було досягнуто того самого результату, що й компанія, яка робила за допомогою книги Excel.

Дві панелі у верхній частині зображення показують стандарти, які необхідно було визначити для проведення оцінювання. Обидва вони є файлами .standards і зображують стандарт CMMI та модель якості ISO 25010. На нижній лівій панелі є визначення оцінки, де оцінку було визначено для конкретного клієнта. Оскільки клієнту потрібна була оцінка характеристики надійності, це було зазначено там. Крім того, команда оцінювачів вибрала лише ті процеси з CMMI, які, на її думку, вплинули на цю характеристику. Це файл .assessment.

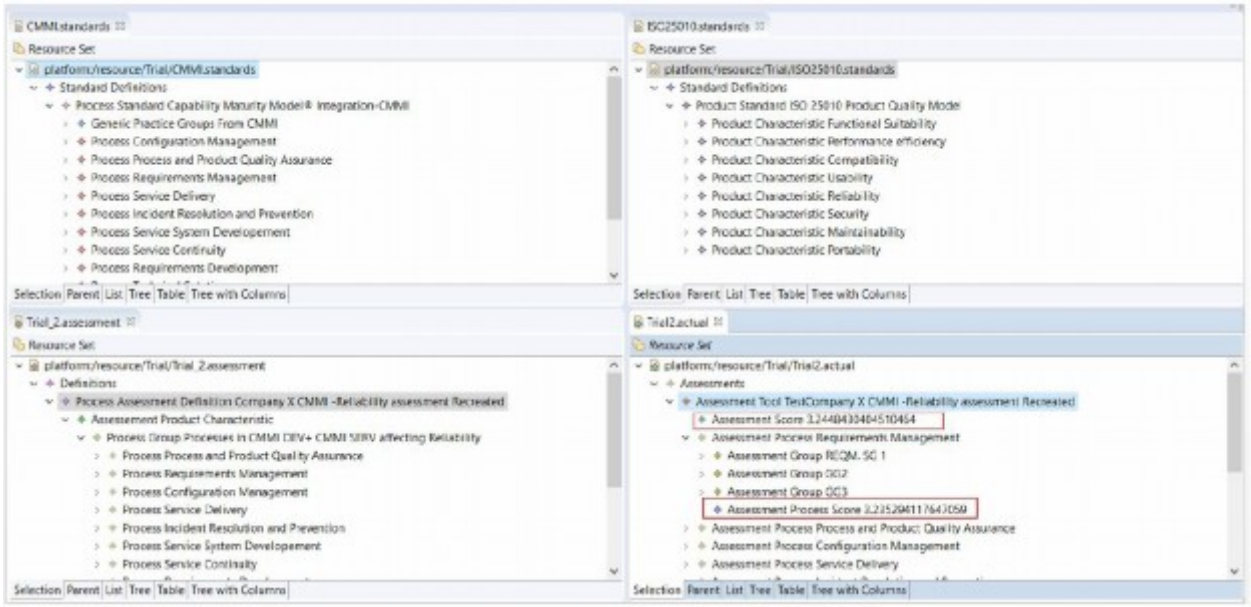


Рис. 3.17. Екземпляри середовища виконання редакторів, необхідні для оцінки надійності CMMI

На правій нижній панелі є екземпляр фактичної метамоделі під час виконання, яка є остаточною оцінкою CMMI на основі визначеної оцінки. Підсумкова оцінка виділена червоною прямокутною рамкою, оцінка 3,24, яка є такою ж, як отримана за допомогою Excel на рисунку 3.17. Крім того, оцінки окремих процесів також можна переглянути за допомогою опції спадного меню, наприклад оцінка процесу оцінки процесу керування вимогами, яка становить 3,23, така ж, як оцінка можливостей REQM із таблиці Excel на рисунку 3.17.

3.3.2. Випадок використання Testing Maturity Model integration

На рисунку 3.18 наведено оцінку TMMi (Testing Maturity Model integration), яку обрано для характеристики надійності. Як видно на рисунку, для оцінки було обрано вісім областей процесу за стандартом TMMi. Області процесу вказані в першому рядку аркуша Excel. Їх було вибрано, оскільки команда з оцінки вирішила, що ці зони процесу є тими, які впливають на надійність. Вісім областей процесу: стратегія обробки тестів (TPS), планування тестів (TP), проектування та виконання тестів (TDE), інтеграція

життєвого циклу тестів (TLE), нефункціональне тестування (NFT), організація тестів (TO) і експертні оцінки (PR). Загальна оцінка оцінювання – 3,30. Кожна з областей процесу має власні індивідуальні бали процесу. Середнє з восьми балів процесу дає експерту загальну оцінку. За винятком TPS, усі інші показники практик окремих областей процесу були розмиті, щоб захистити конфіденційні дані клієнтів. Оцінка процесу або можливостей TPS становить 3,33.

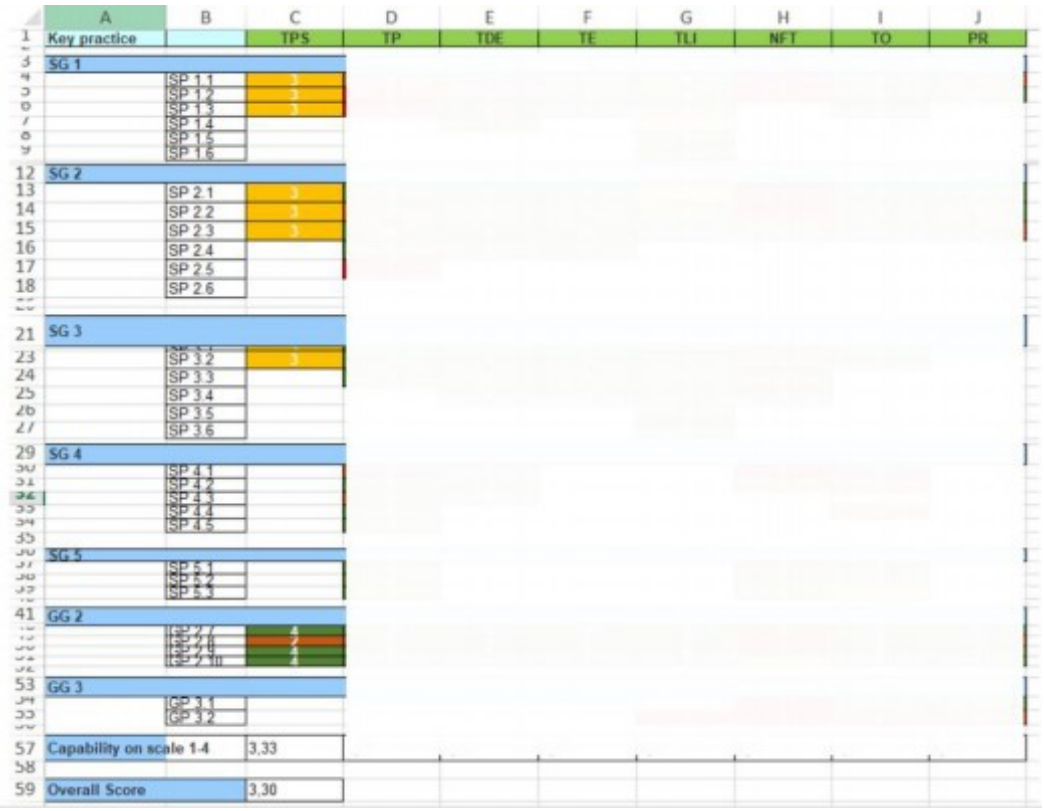


Рис. 3.18. Таблиця для оцінки Testing Maturity Model integration

Для оцінки TMMi на рисунку 3.18 показано чотири панелі, що показують екземпляри часу виконання на основі трьох створених метамоделей.

Дві панелі у верхній частині малюнка показують стандарти, які необхідно було визначити для проведення оцінювання. Обидва вони є файлами .standards і зображують стандарт TMMi та модель якості ISO 25010. Як показано червоними стрілками, дані з двох верхніх панелей разом

допомагають визначити оцінку на нижній лівій панелі. Це файл .assessment, який, у свою чергу, використовується для фактичного оцінювання, як вказано на панелі праворуч червоною стрілкою. Фактична оцінка – це файл .actual, який базується на фактичній метамоделі. У екземплярі оцінки оціночна оцінка та оцінка процесу оцінювання були виділені червоними прямокутними рамками. Порівнюючи з рисунком 3.18, можна порівняти кінцеві результати. Інструмент оцінювання надає точніші значення замість округлення, тому значення оціночного бала становить 3,29 у порівнянні із загальним балом у таблиці Excel, який становить 3,30. Індивідуальні оцінки процесу також можна порівняти з таблицею Excel. У фактичному екземплярі часу виконання процес TPS має оцінку процесу оцінювання 3,33, яка також є такою ж на рисунку 3.18. Отже, редактор дерева може замінити аркуш Excel для завершення оцінювання.

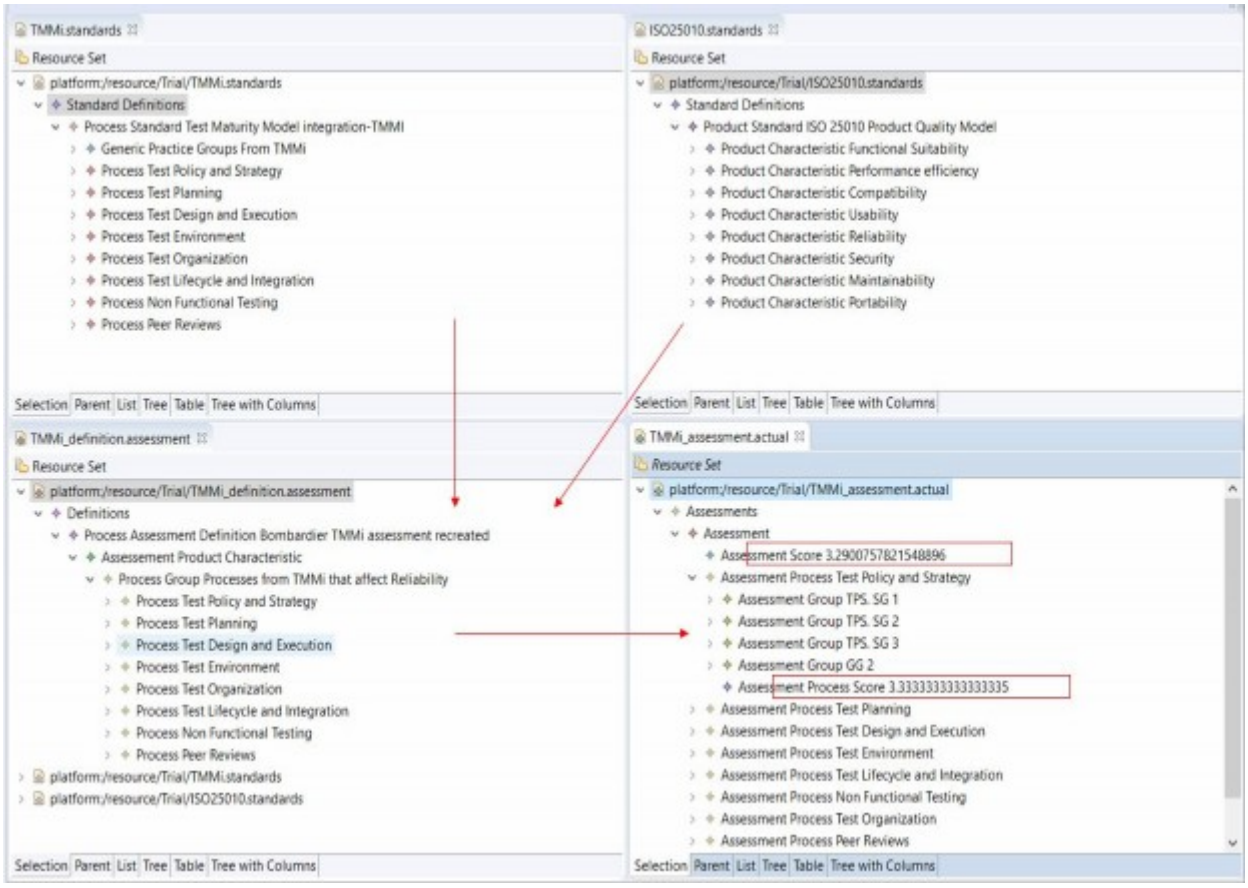


Рис. 3.19. Екземпляри середовища виконання редакторів, необхідні для оцінки надійності TMMi

Проведене дослідження показало, що результати, отримані за допомогою розробленого інструменту оцінювання, статистично не відрізняються від результатів, отриманих за допомогою традиційних шаблонів Excel для оцінки моделей зрілості CMMI та TMMi.

Висновки до розділу

Отже, в цьому розділі детально аналізуються динамічні екземпляри побудованих мета-моделей, що використовувалися для тестування інструменту, а також екземпляри під час виконання, які демонструють редактор дерева для кожної метамоделі, тим самим надаючи реальні приклади практичного застосування цих моделей. Виконано перевірку пропонованих мета-моделей.

ВИСНОВКИ

У магістерській роботі представлено методологію модельно-орієнтованого підходу до аудиту та тестування якості програмного забезпечення. Виконано огляд різних стандартів, що застосовуються для відповідних процесів, зокрема стандарту ISO 25010, який широко використовується в промисловості для оцінки якості програмного продукту. Розглянуто також вплив процесів розробки та тестування на характеристики програмного забезпечення. Описано налаштування процесу оцінювання, що включає етапи визначення відповідних стандартів, формування критеріїв оцінки, розрахунку результатів і виконання оцінювання. Детально досліджено загальну мета-модель процесів і продуктів, яку було модифіковано та адаптовано для створення мета-моделі тестування, включаючи розробку нових мета-моделей, що замінюють попередні.

Представлено інструмент оцінювання, що перевірено на двох промислових прикладах: застосуванні стандарту оцінювання Capability Maturity Model Integration та Testing Maturity Model Integration. Для підвищення точності процесу оцінювання необхідно чітко розмежувати окремі дії, оскільки недоліки мета-моделі впливають на всю систему інструментів. Удосконалення інструментарію починається з перевизначення мета-моделей, щоб забезпечити підтримку поділу проблем, характерних для процесу оцінювання. Це завдання виконано через розробку мета-моделей загального процесу тестування продуктів та програмного забезпечення.

Запропонована мета-модель може використовуватися для фактичного проведення оцінювання, де оцінювач може вводити рівні задоволеності, а розрахунки здійснюються на основі введених рівнів і функцій, які забезпечують точність розрахунків.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Kermata 2. Make instances models, 2020. http://www.kermeta.org/docs/kermeta2/html/Kermeta-FSMDslTutorial.docb/ch03s02.html. 20
2. M. Blaha. UML Database Modeling Workbook. Technics Publications, 2013. 18
3. Suresh Chandra Bose and Ganesh Bose. Transforming organizations to achieve tmmi certication. In Thirty-Fourth Annual Pacic Northwest Software Quality Conference, 2016. v,9
4. John Estdale and Elli Georgiadou. Applying the ISO/IEC 25010 Quality Models to Software Product: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings, pages 492{503. 01 2018.
5. Eclipse Foundation. Editing ecore model instances, 2020. https://www.ntnu.no/wiki/display/tdt4250/Editing+Ecore+model+instances. 20, 21, 34
6. Eclipse Foundation. Sirius/tutorials/startertutorial, 2020. https://wiki.eclipse.org/Sirius/Tutorials/StarterTutorial#Launch_a_new_runtime_from_your_Eclipse. v, 15
7. Dennis Goldenson and Diane L Gibson. Demonstrating the impact and benets of cmmi: an update and preliminary results. 2003. vii, vii, 5
8. Eclipse Foundation Inc. Eclipse modeling framework, 2020. https://www.eclipse.org/modeling/emf/14 CMMI Institute. What is cmmi. 4
9. iso25000.com. Iso/iec 25010, 2020. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0. v, 2,
10. Rajendra Khare. Cmmi assessment, 2020. = http://cmmi-assessment.com/12
11. Tummalapalli.bnvv.Kiran Kumar. A model based approach and tool support for software process audit on software product quality in automotive domain. Master's thesis, Eindhoven University of Technology, 2018. i, i, v, v, v, 2, 6, 11, 13

12. Pushkar Venkat Narayanan. Applying model based techniques for process improvement and assessment in the automotive domain. Master's thesis, Eindhoven University of Technology, 2017. 2
13. NTNU. Eclipse modeling framework, 2020. https://www.ntnu.no/wiki/display/tdt4250/Ecore
14. VDA QMC Working Group 13 / Automotive SIG. Automotive spice R process assessment/reference model for development, version 1.3. 2015.
15. SourceForge. Make instances models, 2020. http://dynamicgmf.sourceforge.net/create_xmi.html. 20
16. CMMI Product Team. Cmmi R for development, version 1.3. 2010. v, vii, vii, 1, 2, 5, 6, 9
17. Carnegie Mellon University. About the sei, 2020. https://www.sei.cmu.edu/about/index.cfm.
18. E van Veenendaal. Test maturity model integration (tmmi): Guidelines for test process improvement, release 1.2. TMMi Foundation. Ireland, 2018. v, 7
19. vogella GmbH. Eclipse modeling framework (emf) - tutorial, 2020. https://www.vogella.com/tutorials/EclipseEMF/article.html.
20. Ось список з 30 англомовних літературних джерел, які стосуються методології модель-базованого підходу до аудиту та тестування якості програмного забезпечення:
21. Utting, M., & Legeard, B. (2010). *Practical Model-Based Testing: A Tools Approach*. Elsevier.
22. Zhu, H., Hall, P. A. V., & May, J. H. R. (2014). *Software Unit Test Coverage and Adequacy*. ACM Computing Surveys, 29(4), 366-427.
23. Musa, J. D. (2004). *Software Reliability Engineering: More Reliable Software Faster and Cheaper*. McGraw-Hill.
24. 4. Binder, R. V. (1999). *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley.
25. Pretschner, A., Malek, M., & Wagner, S. (2007). *Model-Based Testing for Real: The In-the-Loop Approach*. IEEE Software, 24(4), 46-53.

26. Utting, M., & Pretschner, A. (2008). *A Taxonomy of Model-Based Testing*. Journal of Software Testing, Verification & Reliability, 22(4), 297-312.
27. Bochmann, G. V., & Petrenko, A. (1994). *Protocol Testing: Review of Methods and Relevance for Software Testing*. ACM Computing Surveys, 29(4), 254-288.
28. Hierons, R. M., & Nuñez, M. (2009). *Testing from a Model: I/O and Other Models*. Springer.
29. Cheng, B. H., & Atlee, J. M. (2007). *Research Directions in Requirements Engineering*. IEEE Software, 24(2), 44-52.
30. El-Far, I. K., & Whittaker, J. A. (2001). *Model-Based Software Testing*. Encyclopedia of Software Engineering. John Wiley & Sons.
31. Briand, L. C., & Labiche, Y. (2002). *A UML-Based Approach to System Testing*. IEEE Transactions on Software Engineering, 31(1), 4-18.
32. Whittaker, J. A. (2000). *What is Software Testing? And Why is It So Hard?*. IEEE Software, 17(1), 70-79.
33. Gros, C., & Muccini, H. (2008). *Automated Model-Based Testing: Current Challenges and Future Directions*. Journal of Software Testing, Verification & Reliability, 18(3), 219-230.
34. Fraser, G., & Arcuri, A. (2014). *EvoSuite: Automatic Test Suite Generation for Object-Oriented Software*. ACM Transactions on Software Engineering and Methodology, 24(2), 1-42.
35. Bai, X., Tsai, W., & Chen, Y. (2008). *Ontology-Based Test Modeling and Partition Testing of Web Services*. IEEE Transactions on Services Computing, 1(4), 1-12.
36. Wegener, J., & Grochtmann, M. (1998). *Test Case Design Using Classification Trees and the Classification-Tree Editor CTE*. Proceedings of the 8th International Conference on Quality Software.
37. Harel, D. (1987). *Statecharts: A Visual Formalism for Complex Systems*. Science of Computer Programming, 8(3), 231-274.

38. Broy, M. (2001). *Formal Methods for Software Engineering: Specification, Verification, and Testing*. Springer.
39. Aichernig, B. K., & Joakimsson, P. (2003). *Model-Based Testing in Practice: Lessons Learned from Two Case Studies*. Proceedings of the 9th International Conference on Formal Methods Europe.
40. Larsen, K. G., Mikucionis, M., & Nielsen, B. (2005). *Online Testing of Real-Time Systems Using UPPAAL*. IFIP International Conference on Testing of Software and Communicating Systems.
41. Chow, T. S. (1978). *Testing Software Design Modeled by Finite-State Machines*. IEEE Transactions on Software Engineering, SE-4(3), 178-187.
42. Bertolino, A. (2007). *Software Testing Research: Achievements, Challenges, Dreams*. Future of Software Engineering (FOSE '07), IEEE, 85-103.
43. Baresi, L., & Pezzè, M. (2006). *On the Automatic Generation of Test Cases for Web Services*. Journal of Systems and Software, 79(3), 303-319.
44. Aichernig, B. K., & Maibaum, T. (2003). *Model-Based Testing in Practice*. Journal of Software Testing, Verification & Reliability, 13(3), 133-145.
45. Levi, Z., & Yahav, E. (2016). *Regression Verification: Proving the Equivalence of Similar Programs*. ACM Transactions on Software Engineering and Methodology, 25(4), 1-35.
46. Gaudel, M.-C. (2005). *Testing Can Be Formal Too*. ACM SIGSOFT Software Engineering Notes, 30(4), 41-45.
47. Mingshu, L., Zhang, J., & Min, Y. (2004). *Software Quality Management: A Survey of Methods and Tools*. Journal of Systems and Software, 79(3), 1-12.
48. Tretmans, J. (2008). *Model-Based Testing with Labelled Transition Systems*. Formal Methods and Testing. Springer.
49. Briand, L. C., & Labiche, Y. (2010). *A UML-Based Approach to System Testing*. Journal of Systems and Software, 79(3), 34-58.

50. Utting, M., Pretschner, A., & Legeard, B. (2012). *A Taxonomy of Model-Based Testing Approaches*. Journal of Software Testing, Verification and Reliability, 22(4), 297-312.
51. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. International Organization for Standardization, 2011.
52. Garousi, V., Felderer, M., & Mäntylä, M. V. (2019). The need for multi-method empirical software engineering: An introduction to the special issue on multi-method research. Information and Software Technology, 105, 1-6.