الجمهوريّة العربيّة السوريّة
جامعة دمشق
كلية الهندسة المعلوماتيّة
قسم هندسة البرمجيات ونظم المعلومات

تحسين جودة إجراءات هندسة البرمجيات المُقادة بالنماذج

Improvement of Quality for MDE (Model Driven Engineering) Processes

رسالة أُعدت لنيل درجة الماجستير في الهندسة المعلوماتيّة
قسم هندسة البرمجيات ونظم المعلومات

إعداد
م . لبنى منصور

إشراف
د. باسم قصيبة

2015

إهداء

إلى الحكاية التي لا تنتهي .. إلى موطن الياسمين ...

سوريا بلدي

إلى من فرشا لي طريق العلم ورود اً .. ووقدا شموع الأمل في حياتي..

أمي وأبي

إلى من شاركوني أشيائي.. فرسموا بسمتي ومسحوا دمعتي ..

إخوتي الأحباء

إلى من فرقتني عنهم الحدود وباعدتني عنهم المسافات .. لكن جمعني معهم الدعاء والأمل .. أرى يوم لقائنا قريب ..

أخوالي مصطفى وعمر

إلى الذي يختبئ في رماديته فيجعل من الصخرة وردة حمراء ..

إلى من كان في غير أرض وغير مكان واشتاق لحفنة من تراب الوطن ليضمها ويلثمها ويجدها أغلى من كنوز الدنيا وأثمن ما في الوجود ..

إلى العطاء الذي أفخر به وأخجل منه ..

إلى ذكراك الطيبة ..

أهدي عملي الأول مع فائق الاحترام والتقدير

د. عمّار المصري

ملخص

يزداد الاهتمام بجودة إجرائيات تطوير البرمجيات مما يجعل من السهل صيانة وتحديث هذه الإجرائيات لأنّ التغيير لابدّ منه. إنّ الهندسة المُقادة بالنماذج هي منهجيّة لبناء إجرائيات تطوير البرمجيات، فقد أثبتت نجاحاً باهراً حيث توسّعت أفقيّاً في كل مجالات الهندسة المعلوماتيّة.

على الرغم من التوسع الأفقي لاستخدام منهجيات الهندسة المُقادة بالنماذج، لم يرافق ذلك زيادة في جودة الإجرائيات المطورة بهذه المنهجية، إذ أنّ عدد الأبحاث يكاد يكون معدوماً.

إنّ النماذج المترفعة التي تقود إجرائيّة الهندسة المقادة بالنماذج هي المسؤولة عن توليد الأدوات (النماذج)، هذه الأدوات هي المسؤولة عن انتاج البرمجيّة، وجودة البرمجيّة مرتبطة ارتباطاً وثيقاً بجودة الأداة التي تولدها، فكلما كانت الأداة أعلى جودة، كلما كانت البرمجية الناتجة عالية الجودة أيضاً.

وجدنا أنه من الأهمية بمكان إعادة النظر ومراجعة تصميم هذه النماذج المترفعة قبل توليد الأدوات (النماذج)، والتأكد من خلوّه من العيوب والعمل على تصحيحها إن وُجدت.

تتصدر العيوب البرمجيّة في السويات المنخفضة (مستوى التصميم والرماز) مفاهيم الحجم الكبير وضعف الترابط، لذا قمنا بنظرة توسعيّة بتوسيع مفهوم العيوب البرمجيّة لتشمل السويات المرتفعة البرمجيّة (مستوى النماذج المترفعة).

قمت باقتراح منهجيّة لاكتشاف هذه العيوب وتصحيحها بتلافي الحجم الكبير والتماسك الضعيف باقتراح التقسيم المناسب للنماذج المترفعة بما يحقق نماذج جزئيّة أصغر حجماً وأعلى ترابطاً وأكثر تخصيصاً، تطلّب ذلك القيام بدراسة تجريبيّة على عدد كبير من الطلاب والعاملين في مجال الهندسة المُقادة بالنماذج ثمّ أخذ النتائج والتوثيقات منهم مرتقين بذلك من المنهج اليدوي الضيق المعتمد على حالات دراسية المحدودة، إلى منهج آلي شامل غير معتمد على السياق بهدف زيادة الجودة. وتمكنا من المصالحة ما بين المنهجيتين المنهج المعتمد على البنية والمنهج المعتمد على المنطق والتحليل القواعدي، وتمكنا من اقتراح أدوات لقياس الجودة وأرفقنا ذلك بنسب وإحصاءات شكلت بنظرنا الدليل القوي والبرهان على تحسّن الجودة وتجاوز العيوب.

بهذه الطريقة يمكن أن نولد أدوات كفوءة ومؤهلة، وبالتالي برمجيات كفوءة ومؤهلة من خلال ضمان جودة إجراءات النماذج المترفعة، هذا الاتجاه يمثل بنظرنا التوسع الشاقولي للهندسة المُقادة بالنماذج.

Abstract

The quality of software development processes importance increase which makes maintenance and update of these procedures are more easily, because the change must be done.

Model Driven Engineering (MDE) is an approach to build software development processes by proving a large success through wide expansion in all information technology fields.

Although of the wide extension of using Model Driven Engineering Procedures, It doesn't accompany with quality increasing for the developed procedures by this approach. Researches count could be null.
Meta Models (M2), which drives The Model Driven Engineering (MDE) approach, is the basics for the production of modeling tools. Modeling tools allow us to define Models (M1) in one step in the process. Software quality highly relates with development process. Development process is affected by tools quality that supports its steps. Whenever the highest quality tools increases the quality of development process that uses these tools.
We find that it's so important to review the design of Meta-Model before generating tools (Models). We have to check that there is no smells and refactor them if there are found.
The software smells in the low levels (Code and Design) levels centers of large size and less cohesion concepts. So, we try to extend software smell concepts to contain the high levels of software (Meta-Models).
I have suggested an approach to detect the smells and refactor them to be avoiding the size and cohesion problems by part the meta-model to many partial Meta-Models which are smaller, more cohesion and more specializing. I have to achieve an empirical study with large number of students and MDE workers. After that I took the results and feedbacks to expand the handle approach which depends on stakeholders' views and limited case studies to general automatic approach which doesn't depend on the context to increase quality. I was able to approve between two approaches, the structural one and the semantic or logic one in the same algorithm. I was able to suggest new tools to measure the quality. I supplied this research results with statics and mathematical ratios which are the strong proof of enhance and improvement of Quality and pass the smells. This approach represents the vertical expandition in MDE. We supply our work with empirical studies with 100 students from Damascus University –Faculty of Information Technology.

جدول المحتويات

مقدّمة

يُعتبر موضوع جودة البرمجيات من المواضيع الهامة بسبب تعقيده وتغلغله بالأنظمة البرمجيّة، علاوةً على ذلك فالاتجاه الحالي من التطوير والصيانة يتطلب قياس الجودة مع الكثير من التفاصيل. إنّ الجودة العالية للبرمجيّة تسمح بالتخفيف من كلفة التصحيح، والاختبار، وإعادة استخدام المنتج، وتتأثر الجودة بشكل عكسي مع وجود العيوب البرمجية.

فالعيوب هي مؤشر إلى وجود مشكلة ما في مكان محدد من الرماز أو التصميم، تؤدي إلى تراجع جودة المنتج البرمجي وأدائه، وهدر الوقت لتعديله، وتمديده، وفهمه. وإنّ الكشف المبكر عن العيوب وتصحيحها سيفيد في إجراءات التطوير والصيانة.

إنّ النماذج المترفعة التي تقود إجرائيّة الهندسة المقادة بالنماذج هي المسؤولة عن توليد الأدوات (النماذج)، هذه الأدوات هي المسؤولة عن انتاج البرمجيّة، وجودة البرمجيّة مرتبطة ارتباطاً وثيقاً بجودة الأداة التي تولدها، فكلما كانت الأداة أعلى جودة، كلما كانت البرمجية الناتجة عالية الجودة أيضاً.

على الرغم من الانتشار الواسع (أو التوسع الأفقي) لاستخدام منهجيات الهندسة المُقادة بالنماذج، لم يرافق ذلك زيادة في جودة الإجرائيات المطورة بهذه المنهجية، إذ أنّ عدد الأبحاث يكاد يكون معدوماً.

وجدنا أنه من الأهمية بمكان إعادة النظر ومراجعة تصميم هذه النماذج المترفعة قبل توليد الأدوات (النماذج)، والتأكد من خلوّها من العيوب والعمل على تصحيحها إن وُجدت.

بهذه الطريقة يمكن أن نولد أدوات كفوءة ومؤهلة، وبالتالي برمجيات كفوءة ومؤهلة من خلال ضمان جودة إجراءات النماذج المترفعة، هذا الاتجاه يمثل بنظرنا التوسع الشاقولي للهندسة المُقادة بالنماذج.

إنّ مشكلة الحجم الكبير وضعف الترابط تتصدر عيوب الرماز والتصميم، ولا تقتصر هذه المشكلة على مستوى الرماز والتصميم بل تتعداها إلى النماذج، فالنماذج الجيدة هي كالصفوف تخضع لقيود معينة من حيث الحجم، وعدد المفاهيم الموجودة فيه، فكلما كانت أصغر كلما كانت أكثر ترابطاً وتماسكاً، وبالتالي أمكن استخدامه كوحدة هيكلية أكثر مرونة، وأكثر قابلية للفهم، والتوسع، والمراجعة، وذلك يتطابق مع معايير الجودة العالمية. وهذا ما تقوم عليه أسس هندسة البرمجيات من خلال اتباع منهجيّة "فرّق تسد"، إذ تقسم فضاء العمل إلى مجموعة من الرزم، بحيث تجمع في كل رزمة العناصر المتشابهة، مما ينظم العمل ويرفع جودته.

لتحقيق جودة أفضل، يتوجب علينا تحديد عدد أمثلي للعناصر المُحتواة في الوحدة البرمجيّة، الوحدة البرمجيّة يُمكن أن تكون صفاً، أو رزمةً، أو نموذجاً، أو .. ..، وبهذه الطريقة نتجنب مشاكل الحجم الكبير وضعف الترابط قدر المُستطاع.

غير أنّ مثل هذه القيود لايمكن إسقاطها على مفاهيم النماذج المترفعة التي تُعد أساساً لتوليد النماذج والأدوات، إذ يلحق بها تبايناً كبيراً من حيث الحجم تبعاً للغرض الذي تعمل على نمذجته، فنمذجة لغة برمجة غرضية التوجّه تختلف عن نمذجة منصة عمل خاصة بمعايير الجودة، إذ أنّ لكل منهما معاييره الخاصة بما يتناسب مع المتطلبات.

من هنا كانت أهمية العمل على تقسيم النماذج المترفعة إلى عدة نماذج جزئية، وفصلها بناءً على معايير وأسس علمية بغض النظر عن المجال الذي تنمذجه، لتعمل على توليد أدوات أكثر تخصصاً وأكثر ترابطاً.

وبهذه الطريقة نكون قد وسّعنا مفهوم العيوب من مستوى الرماز والتصميم لتشمل مستوى النموذج، واعتبار النموذج الحاوي على عدد كبير من المفاهيم عيباً من العيوب لابدّ من تصحيحه، وتقسيمه لعدة نماذج جزئية.

وبهذا نكون قد وصلنا إلىتحقيق هدف البحث وهو تحسين جودة إجراءات هندسة البرمجيات المُقادة بالنماذج.

جرى البحث في هذا السياق على أربع مراحل أساسيّة، تتمحور المرحلة الأولى حول الدراسة المرجعيّة التي تمّ فيها البحث في مختلف جوانب الهندسة المُقادة بالنماذج وأهم العيوب البرمجيّة التي يتعرض لها المُنتج البرمجي، أما المرحلة الثانية فتتركز حول طرح نماذج مترفعة للتقسيم اليدوي بعد ممارسة التطبيق العملي عليها من قبل الطلاب، وتتمحور المرحلة الثالثة حول تمديد مفهوم عيوب السويات الدنيا (الرماز والتصميم) لتشمل السويات العليا (النماذج والنماذج المترفعة)، واقتراح منهجيّة آليّة للتقسيم بما يضمن سهولة الاستخدام والفهم والصيانة والمراجعة .. ..، واقتراح تمديد لهذه المنهجيّة بما يضمن المصالحة ما بين المنهجيتين الهيكليّة والمنطقيّة، أما المرحلة الرابعة فتشمل اختبار النماذج المترفعة المُقسّمة من قبل الطلاب والتقييم وفقاً للمعايير الدوليّة لجودة البرمجيات، وشكلت بدورها البرهان القوي على تحقيق أهداف المشروع في زيادة الجودة لإجراءات هندسة البرمجيات المُقادة بالنماذج.

تمّ تقسيم التقرير النهائي للرسالة إلى أربعة أجزاء رئيسة تندرج ضمنها خمسة عشر فصلاً كالتالي:

الجزء الأول، ويتناول مفاهيم عامة حول الهندسة المُقادة بالنماذج ويندرج ضمنه ثلاثة فصول كالتالي:

- **الفصل الأول ... مفاهيم أساسيّة في الهندسة المُقادة بالنماذج.**

يهدف هذا الفصل إلى التعريف بالمنهج ذي الأربعة مستويات وأهميتها، ويقدم أهم التجارب في مجال الهندسة المُقادة بالنماذج.

- **الفصل الثاني ... عيوب السويات الدنيا (الرماز والتصميم).**

يقدم هذا الفصل مفهوم العيوب البرمجية في السويات المنخفضة (سوية الرماز والتصميم) ويبيّن آثارها السلبية على البرامج، ويقدم أهم النماذج التي تعمل على اكتشاف هذه العيوب وتصحيحها.

- **الفصل الثالث ... دراسات مرجعيّة في طريقة التصحيح وإعادة البناء.**

نهتم في هذا الفصل بتوضيح أهمية إعادة البناء والتصحيح من الأخطاء والعيوب البرمجيّة، ويعرض أهم التقنيات الموجودة لذلك، وأهم النماذج التي تعمل على اكتشاف العيوب وتصحيحها.

الجزء الثاني، ويتناول الدراسة المرجعيّة لأهم العيوب وتأطير للحل ويندرج ضمنه ستة فصول كالتالي:

- **الفصل الرابع ... مفهوم عيوب الصفوف الكبيرة.**

نعرض في هذا الفصل تعريف عيب الصف الكبير، وصفاته، والسلبيات التي تؤثر على البرنامج، وأهم الدراسات السابقة في اكتشافه وتصحيحه.

- **الفصل الخامس ... مفهوم عيب العنصر.**

نخصص هذا الفصل لتقديم عيب العنصر وأسبابه، مساوئه واقتراحات تصحيحه، بالإضافة للدراسات السابقة حوله.

- **الفصل السادس ... مفهوم عيب الوراثة.**

نبحث في هذا الفصل في مفهوم عيب الوراثة، تعريفه وسلبياته، طريقة اكتشافه وتصحيحه، وأهم الدراسات السابقة حوله.

- **الفصل السابع ... مفهوم عيب التكرار.**

نلخص في هذا الفصل عيب التكرار وعيب إهمال العنصر وعدم استخدامه، وأهم سلبياته وتأثيراته كحمل زائد على النظام، بالإضافة لأهم الدراسات السابقة حول هذا المجال.

- **الفصل الثامن ... مفهوم عيب الحلقة.**

نشرح في هذا الفصل أثر الحلقات على البرامج، وما تؤدي إلى تراجع في سوية النظام ككل، ويظهر أهم الدراسات السابقة في هذا المجال.

- **الفصل التاسع ... مفهوم الجودة البرمجيّة.**

نتوقف في هذا الفصل عند معايير الجودة العالمية ونستعرض أهمها ونقوم بإسقاط عناصرها على منهجيتنا ونشرحها بشكل دقيق.

الجزء الثالث، ويتناول النموذج المُقترح وتمديده ونتائجه ويتناول أربعة فصول تندرج كالتالي:

- **الفصل العاشر ... منهجيّة التقسيم المُقترحة.**

نقترح في هذا الفصل منهجيّة لتلافي أسباب العيوب السابقة (الحجم والترابط) من خلال تقسيم النماذج المترفعة إلى عدد من النماذج الجزئية، بما يضمن سهولة الاستخدام، والتعديل، والفهم، والتمديد والصيانة، اعتماداً على الدراسات السابقة والتجربة العملية مع طلاب السنة الخامسة في كلية الهندسة المعلوماتيّة. وقمنا بالتعديل على المنهجيّة المُقترحة للتقسيم، من خلال المصالحة ما بين المنحى الهيكلي، والمنحى المنطقي، من خلال التحليل القواعدي لأسماء المفاهيم وقياس مدى تشابهها بالاعتماد على الأنطولوجي، وذلك في محاولة لترقية المنهجيّة وجعلها أكثر دقة.

- **الفصل الحادي عشر ... تمديد منهجيّة التقسيم بما يشمل المنحى المنطقي.**

نتابع في هذا الفصل تفصيل خوارزمية المُقارنة والتشابه المُعتمدة على الأنطولوجي، وإظهار كيفية الحسابات العددية لهذا التشابه مع إيضاح سلبيات وإيجابيات الخوارزميّة المُعتمدة.

- **الفصل الثاني عشر ... التطبيق العملي للمنهجيّة مع الخطوات التفصيليّة.**

ننتقل في هذا الفصل إلى إيضاح الخطوات التفصيلية للمنهجيّة المُقترحة موضحين الحسابات العددية لذلك بشكل دقيق.

- **الفصل الثالث عشر ... الإحصاءات والتقييم للمنهجيّة المُقترحة.**

نسعى في هذا الفصل إلى إظهار الإحصاءات والمخططات البيانية الناتجة عن التجريب العملي للنماذج قبل التقسيم وبعده، ونسعى إلى تقييمها من عدة وجهات نظر وفقاً لعدة نقاط تضمن سهولة الاستخدام واختصار الوقت ووضوح النماذج.

الجزء الرابع ويتناول التطبيق العملي وواجهات التنفيذ ويندرج تحته فصلان كالتالي:

- **الفصل الرابع عشر ... أداة النمذجة المستخدمة.**

نسهب في هذا الفصل في شرح الأداة التي سنستعملها لننمذج ما نريد، ونورد الكثير من الأشكال والصور لتوضح آلية العمل.

- **الفصل الخامس عشر ... التحقيق العملي وواجهات التنفيذ**

هذا الفصل يتضمن صور الواجهات البيانيّة للمنهجيّة المطوّرة، نبيّن فيها تسلسل التنفيذ.

ننهي هذه الرسالة بخلاصة توجز أهم النتائج والتوصيات التي تمكنّا من الوصول إليها، سواءً من الدراسة المرجعيّة أو من الدراسة العمليّة والتجريبيّة المطروحة في كلية الهندسة المعلوماتيّة في جامعة دمشق. ونفتح آفاق مستقبليّة لهذا العمل بشكل موجز.

دمشق – 1 تشرين الأول / 2015

الجزء الأول

مفاهيم عامة حول الهندسة المُقادة بالنماذج

- الهندسة المُقادة بالنماذج
- عيوب الرماز والتصميم
- التصحيح وإعادة البناء

الفصل الأول
الهندسة المقادة بالنماذج

1.1. مقدمة

يُعتبر موضوع جودة البرمجيات من المواضيع الشائكة التي يجب تحقيقها في مجال صناعة البرمجيات، لأنّ الجودة العالية للبرمجيّة ممكن أن تخفف من كلفة التصحيح، والاختبار، وإعادة استخدام المنتج، ولمّا كانت جودة البرمجيّة مرتبطة بجودة الأداة التي تولدها، كان لابدّ من إعادة النظر في تصميم النماذج المترفعة (Meta Model) التي تقود إجرائيّة الهندسة المقادة بالنماذج، ((Model Driven Engineering (MDE) المسؤولة عن توليد هذه الأدوات، إذ لابدّ من التأكد من خلوّ التصميم من عيوب النموذج (Smells) والعمل على تصحيحها إن وُجدت، لتلافي أخطار صعوبة الفهم والتوسع والتعديل والصيانة.

يستعمل مصطلح الهندسة المُقادة بالنماذج (MDE) للتعبير عن الاستخدام النظامي للبنى البرمجيّة أو النماذج كحقائق أو عناصر أوليّة خلال عمليّة تصميم البرمجيات. فهي منهج لمطوري البرمجيات يؤكد على ضرورة استعمال النماذج في عمليات التحديد، التطوير، التحليل، التحقيق وإدارة الأنظمة البرمجيّة، وتكمن فائدتها الأساسيّة في استخدام النماذج والنماذج المترفعة (Meta Models) وتحويلات النماذج لتنظيم فعالية نظام ما.

2.1. تعريف النماذج المترفعة

المعنى الحرفي لكلمة **Model** هو النموذج، ويأخذ عدة معاني:

- تجريد لشيء في العالم الحقيقي.
- هيكل خارجي يستعمل لإنتاج شيء في العالم الحقيقي.

والهدف منه التأكد من عمل النظام قبل إنشائه بشكل كامل، وبالتالي توفير الكلفة، إذ يمكن للنموذج أن يجيب عن الأسئلة المهمة التي تتطابق أجوبتها مع النظام الذي يتم بناؤه.

أما المعنى الحرفي لكلمة **Meta** باللغة اليابانيّة هو "بعد". والمواضيع المتعلقة بـ Meta في علوم الحاسوب تستخدم بكثرة وبمعان مختلفة:

- في **قواعد البيانات،** مصطلح Metadata: يعني "بيانات عن البيانات"، ويشير إلى قواميس البيانات وغيرها.

- في **لغات البرمجة**، مصطلح Meta Interpreter يعني "مترجم لمترجم (برنامج)" .
- أما في **النمذجة المفاهيميّة** Conceptual Modeling فإنّ مصطلح Meta Model يعني "نموذج لنموذج بيانات"، مثلاً نموذج . E-R للنموذج العلاقاتي E-R.

وبالتالي إنّ النماذج المترفعة **(Meta Model)** يمكن تعريفها بأنها: النماذج التي يمكن أن تُعرّف عند مستوى أعلى من التجريد بشكل يسهل اندماجهم فيما بعد. وإنّ البيانات المنمذجة باستخدام النماذج المترفعة مثل المخططات، الصفوف.. . هي أجزاء من نماذج مترفعة أخرى التي بدورها أيضاً هي أجزاء من Meta Meta-Model، وهكذا.

وأما المعنى المُراد في **هندسة البرمجيات** للنماذج المترفعة فهي مجموعة من القواعد والشروط التي تمكننا من تعريف المفاهيم اللازمة لبناء النماذج المجردة، ويكمن هدفها بالدمج بين النماذج المختلفة لوصف النظم البرمجيّة.

3.1. أهمية الهندسة المقادة بالنماذج

إنّ تعقيد البرمجيات وتغلغلها في المجتمع ينمو بشكل أسي، وبالتالي هناك إجماع بأن الطريقة الواقعية الوحيدة لإدارة هذه التعقيدات واستمرار تحسين البرمجيات، هي تطوير البرمجيات باستخدام النماذج المجرّدة والهندسة المقادة بالنماذج.

يُمكن القول بأنّ الهندسة المقادة بالنماذج أصبحت ذات انتشار واسع على المستوى الأكاديمي والصناعي، وهي الطريقة المُثلى للتعامل مع التعقيدات الناتجة عن البرمجيات الحديثة، الكثيرون يروا أن الخطوة المستقبليّة للأبحاث والمشاريع البرمجيّة هي التوسع في بناء مستوى التجريد لكي تتم عمليّة تصدير وضم البيانات بشكل سهل وفقاً للمعايير وهذا بدوره يفرض على الشركات والمطورين أن يستعملوا النماذج كعناصر أوليّة في التطوير.

تتضمن الهندسة المُقادة بالنماذج مناهج تقود النماذج فهي تتضمّن (معماريّة البناء، النمذجة المحدودة المجال، حساب تكاملات النماذج)، وبالتالي يمكن القول بأنها:

- تلعب دوراً متنامياً في هندسة البرمجيات غرضيّة التوجه، فمعظم المناهج تستخدم النماذج المترفّعة بطريقة عمليّة؛
- تُستخدم كتقنية عامة لدمج وتعريف نماذج من مختلف المجالات، وإنّ الجوانب المشتركة لهذه الرؤى المختلفة يمكن أن تُعرّف وتُشارك، وبالنتيجة فإنّ تقنية النمذجة المترفّعة يمكن أن تطبق في مختلف مجالات التطبيقات، وخاصة لأغراض التوحيد القياسي والمعياري، ولذلك ينبغي تعريفها بدقة، فضلاً عن كونها بديهيّة ومهيكلة بشكل جيد؛

- تدعم توصيفات حبيبيّة متنوعة؛
- تدعم وسائل لتعريف المفاهيم الأوليّة، مثل: الكيانات، النشاطات، والأهداف، ضمن نطاق النماذج المترفّعة؛
- تدعم مجموعة متنوعة من العلاقات المرجعيّة، مثل: يعرّف، يدل، يذكر، يشمل، وغيرها؛
- يسمح برأسملة التوصيفات البدائية للنظام ورأسملة معرفة تطوير هذا النظام إذ يعرفها ضمن تحويلات النماذج بحيث يسمح بامتلاك وثائق محدثة لرماز النظام؛
- الميزة الهامة هي أنّ فعالية التطوير لبرمجية ما بإمكانها أن تتكيف بسرعة مع أي تغيير في شروط تحقيق النظام، والآثار المترتبة على هذا التغيير يمكن حصرها بسهولة، بينما يمكن إعادة استخدام الأجزاء الأخرى غير المتضررة؛
- يرفع مستوى التطوير من التجميع المباشر للأغراض نحو تحويلات النماذج، هذه التحويلات مقادة بالنماذج المعرّفة في مستوى تجريد أعلى، مما يسمح بأتمتة عملية تجميع الأغراض، وهكذا نتجه نحو جعل تطوير البرمجيات قابلة للتصنيع.

وبالتالي تقوم على فكرة استخدام النماذج وتحويلاتها لتنظيم فعاليّة نظام ما، فهي تسمح بتوصيف منهجيّة لتعريف المشكلة وكيفية الذهاب إلى حلها، إذ تقسم فعاليّة تطوير البرمجيات إلى عدة مستويات من التجريد، ويتم نمذجة كل جانب بشكل مستقل عن الآخر عن طريق النماذج المترفّعة. فهي تحقق مبدأ الفصل بين اهتمامات النظام، وتزيد من إمكانية إعادة استخدام النماذج. هذا وتختلف المناهج في كل مستوى عن الآخر، بدقتها المهنيّة أو التقنيّة، والانتقال بين هذه المستويات يتم عن طريق تحويلات النماذج، وعلاوةً على ذلك إمكانية توليد أداة آليّاً بحيث ندعم النمذجة في كل اهتمام.

هذا وقد شهد قطاع الهندسة المعلوماتية توسّعاً ملحوظاً في استخدام الهندسة المُقادة بالنماذج (MDE) يُمكن اعتباره (توسّعاً أفقيّاً) لاستخدام هذه المنهجيّة، أهمها:

- الوكلاء المتعددين.
- تطوير تطبيقات الويب.
- تطوير تطبيقات النظم المضمّنة.
- تطوير أدوات الـ CMMI.
- تطوير نظم الزمن الحقيقي و المضمنة.

(غير أنّ هذا التوسع الأفقي في الاستخدام لم يرافقه زيادة في جودة الإجرائيات المطوّرة بهذه البرمجيّة، لذا كان لابدّ من إعارة الانتباه لمسألة جودة الإجرائيات المطوّرة بهذه المنهجية وتقنياتها، حيث أنّ عدد الأبحاث يكاد يكون معدوماً، وهو يشكل بنظرنا التوسع الشاقولي في مجال الهندسة المقادة بالنماذج.)

تؤكد مناهج الهندسة المُقادة بالنماذج على تحسين الإنتاجيّة، النقل، التشغيل البيني، والصيانة. فبشكل عام أوضحت العديد من الدراسات [1] أن الإنتاجيّة ترتفع من (20 ← 800%) [2] عند استعمال هذا النوع من الهندسة، وأنّ أسباب انخفاض الإنتاجيّة التي قد تحصل أحياناُ يعود للافتقار إلى الأدوات التقنيّة، وحدوث فائض في النماذج، وزيادة التعقيد، وضرورة التزامن مابين الرماز والنماذج. وإنّ التجارب المختلفة هي ذات نتائج متناقضة منها الناجح، ومنها الفاشل، وبالتالي تشكل عائقاً صعباً أمامنا وتحديّاً كبيراً حول إمكانية وضع القواعد العامة لاختيار الهندسة المقادة بالنماذج وتبنيه كمنهج للتطبيق، وبالتالي فهي لها نشاطاتها ذات التأثير السلبي والإيجابي، فعلى سبيل المثال: توليد الرماز يعتبر للوهلة الأولى ذو تأثير إيجابي لتوفيره الوقت والجهد على المبرمج لأنه يكفيه عناء تعلم التقنيّة، غير أنّه يخلق صعوبات كثيرة في تكامله مع الأنظمة الموجودة، وقد يخلق صعوبات كثيرة في الصيانة.

أما كيفيّة الموازنة ما بين التأثيرات السلبيّة والإيجابيّة فهي بحد ذاتها لاتزال تعتبر تحدياً يعتمد على السياق.

لذا لا نغفل عن ضرورة وجود برهان على زيادة الجودة، فيما إذا نجحت الحزمة الحالية من البرمجيات التي تتبع الهندسة المُقادة بالنماذج في تطويرها في تحقيق أداء أفضل وجودة أعلى أم لا، والبرهان لايكون إلا باستعراض النتائج التجريبيّة.

وبالتالي سيتم دعم النتائج التي سنتوصل إليها وذلك بتجريبها على يد مجموعة كبيرة من المبرمجين والطلاب وسيتم رفدنا بكافة التقارير التي تعمل على تقييم الأداء والجودة الخاص بنموذجنا.

4.1. الجودة في الهندسة المقادة بالنماذج

إنّ التركيز على سمات الجودة في النماذج البرمجية، وتطوير بيئة النمذجة يعود إلى ضرورة تحسين جودة الأدوات المستخدمة في النمذجة في سياق الهندسة المُقادة بالنماذج لما لها من أثر مباشر على تحسين جودة المنتج النهائي.

تَعد الهندسة المقادة بالنماذج بتحقيق تواصل أفضل مابين أصحاب المنفعة، وزيادة المرونة في تطبيق الحلول البرمجيّة في منصات مختلفة، وبالتالي تُصبح البرمجيّة أقل عرضة للخطأ، وأقل كلفة من التطوير اليدوي، وتتحسّن جودتها. وبالتالي تقوم بتغطية أهداف الجودة المُعرّفة في نماذج الجودة المختلفة، إذ بدأ الباحثون عملهم بتحديد قضايا الجودة في الهندسة المقادة بالنماذج، كتعريف خصائص النموذج المطلوبة لتحقيق الجودة البرمجيّة. فجودة النماذج تتأثر بالعوامل التالية:

- جودة لغة النمذجة، النماذج، الانتقالات مابين النماذج، الأدوات، عمليات النمذجة؛
- المعرفة المكتسبة وخبرة المنمذج ومدى معرفة المطورين بالمجال (أدواته، لغاته ..)؛
- تقنيات ضمان الجودة المطبقة.

هناك العديد من الأعمال التي تساعد على تعريف صفات الجودة وطرائقها بهدف تحسينها على نطاق الهندسة المقادة بالنماذج، إذ تسمح النماذج المترفعة بمشاركة اللغات العامة عند مناقشة الجودة وتسمح بتحويل النماذج العامة إلى نطاقات محددة، فهي بالنهاية نموذج صريح لبناء القواعد الخاصة ببناء نموذج ضمن نطاق محدد، بهدف زيادة جودة نماذجنا في سياق الهندسة المقادة بالنماذج.
وبالتالي يمكن القول بأنّ النماذج المترفعة تتجلى في 3 نقاط أساسيّة، هي:

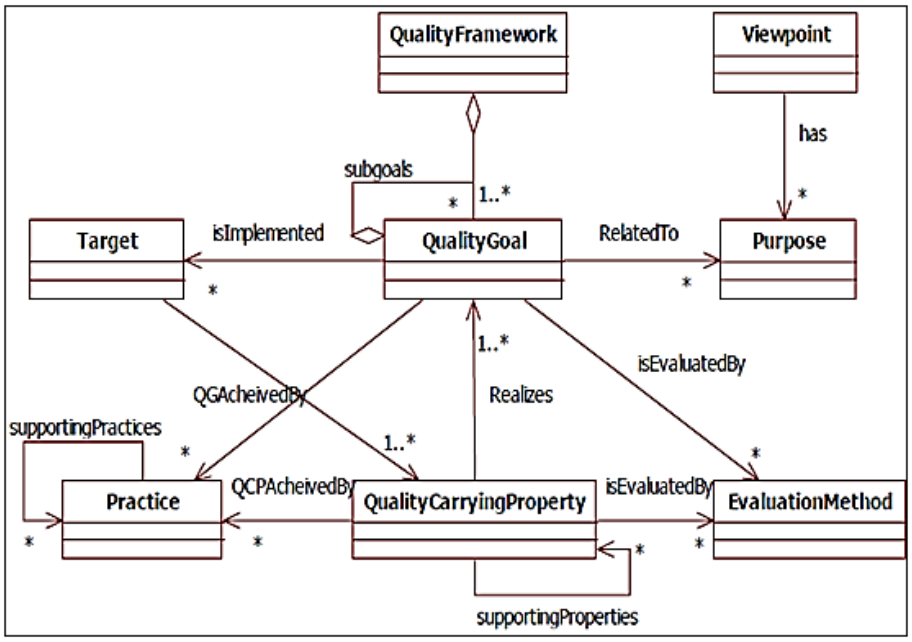
- مجموعة وحدات بنائيّة و قواعد لبناء النموذج؛
- نموذج من أجل اهتمامات في نطاق محدد؛
- نسخة من نموذج آخر.

وبالتالي الهدف من تعريف وتحقيق جودة النماذج المترفعة هو تحقيق الجودة العامة أو الجودة ضمن نطاق محدد للنموذج.
الشكل (1−1) يوضح النموذج المترفع الخاص بنموذج الجودة [3].

شرح المخطط

- **إطار عمل الجودة Quality Framework**

أو كما تُعرف نماذج الجودة، وهي تجميع لكل كيانات الجودة والارتباطات فيما بينها ويمكن أن تستخدم للمجالات العامة أو المحدودة النطاق، ويمكن أن تستخدم في مجال التطبيقات، مثل الاتصالات وأنظمة العمل، أو للتحسين مثل النمذجة والتواصل مع أصحاب المنفعة.



الشكل(1-1) النموذج المترفع الخاص بالجودة.

- **أهداف الجودة Quality Goal**

هي أن يكون هناك تعريف واضح ومفهوم لما تعنيه جودة النماذج لأصحاب المنفعة كالمستخدمين والمطورين والمدراء. فالتعريف الجيد يسمح لنا بقياس الجودة بطرق مفهومة، وواحدة من سمات أهداف الجودة هي "النوع" الذي يسمح بالتصنيف إلى :

- ✓ **أهداف خشنة** تتحقق بالفعاليات؛
- ✓ **أهداف سهلة** تتأثر إيجابياً أو سلبياً بالفعاليات؛
- ✓ ويمكن تصنيفها إلى أهداف جودة **(العمليّة، المشروع، المنتج)**.

- **وجهة النظر Viewpoint**

يستخدم للإشارة إلى وجهة نظر أصحاب المصلحة لتحقيق هدف الجودة وتقسيمهم إلى نماذج، مثل نموذج المستخدمين ، نموذج المطورين، ونموذج المدراء .. . الخ.

- **الأهداف Purpose**

وهي تمثل أهداف أصحاب المصلحة، على سبيل المثال الهدف من النمذجة قد تكون توليد الرماز أو التوثيق وهي على علاقة وثيقة بأهداف الجودة.

- **الخصائص المطلوبة لتحقيق الأهداف Target**

هي فعاليّة تحتوي الخصائص المطلوبة لتحقيق أهداف الجودة، على سبيل المثال: جودة artifact المطورة بسياق الهندسة المقادة بالنماذج، والتي تعتمد على جودة النماذج المترفعة، والأدوات، واللغات، وعمليات الانتقال، وعمليات النمذجة وخبرة الأشخاص. هذه بمجملها تمثل عناصر الهدف في جودة إطار عمل الهندسة المقادة بالنماذج.
يملك الهدف عنصر إضافي وهو الطور، وهو بدوره يشير إلى طور التطوير البرمجي، فمن أجل النماذج نستطيع تعريف أنواع مثل (CIM/ PIM/PSM) أو التحديد، التحليل، التطوير، التحقيق، التوثيق أو الهيكل و السلوك.

- **خصائص الفعاليّة المحققة لأهداف الجودة QualityCarryingProperty**

وهي خصائص الفعاليات التي تحقق أهداف الجودة، فهي تعمل على تحويل الخصائص من شكل غير ملموس إلى شكل ملموس وواضح وقابل للتحقيق، فعلى سبيل المثال:
قابلية الفهم: تعني كون النموذج بسيط (لا يتضّن العديد من العناصر)، منظم بشكل جيد، هاتان الصفتان هما الخصائص الحاملة لنموذج جودة "قابلية الفهم".

- **التطبيقات Practice**

نعرف وسائل تسمح بتحقيق الخصائص المطلوبة، على سبيل المثال: تطبيق الاتفاقيات والمعاهدات يُعتبر كممارسة عملية تقود إلى التطوير الصحيح والمتوافق للنماذج.

- **منهجيات التقييم Evaluation Method**

تتضمن مقاييس وطرق أخرى للتقييم، كتقييم الخبراء، الاختبارات، أو الاستبيانات مما يقلل من نسبة الأخطاء، وبالتالي يساعد على اختصار الكلفة والوقت.
إذ تعد الهندسة المقادة بالنماذج بتوفير الكلفة وزيادة الإنتاجيّة، ولكن من المستحيل التعبير عن الانتقالات التي تطرأ على النماذج بشكل وصفي وتفصيلي بهدف التوليد التلقائي للتعليمات البرمجيّة، وبالتالي كان لابدّ من التخصيص، ويحدث هذا التخصيص إمّا من قبل مطور اللغات القياسيّة، أو من قبل مطور اللغات المحدودة النطاق (DSL) . هذا وإنّ استخدام اللغات المحدودة النطاق ما هو إلّا تدعيم التواصل وترميم الفجوات بين الخبراء المتشابهين بالخبرة التقنية ومفاهيم المجال، بالتالي هذه

الفوائد تستخدم للتحفيز من أجل استعمالها، وتظهر الدراسات أن أغلب المطورين يفضلون استعمال اللغات المحدودة النطاق على استعمال لغات النمذجة القياسيّة.

ولكن يكمن الخطر في عدم وجود الخبرة الكافية لدى المطورين في هندسة اللغة، بل وتطويرها بأسلوب رديء بعيداً عن الجودة، فلن تعود اللغة قابلة للتطوير والصيانة، وترتفع الخطورة طالما لايوجد مجال مُنظم ينطوي على معايير للمفاهيم الموجودة بهذه اللغات.

(من هنا كانت ضرورة اكتشاف أخطاء وعيوب النماذج المسؤولة عن انتاج الأدوات التي تنتج البرمجيات وذلك للحفاظ على الجودة الخاصة بالمنتج، انطلاقاً من الحفاظ على جودة الأداة التي تولدها.)

5.1. التقنية التي تعتمدها الهندسة المقادة بالنماذج

تعتمد الطريقة المستخدمة في هذه التقنية على منهج ذو أربع مراحل، بحيث يتم توصيف العلاقات البنيويّة بين الكيانات في النمذجة المترفّعة، كما يتم تعريف القيود البنيويّة والسلوك الديناميكي للكيانات.

1.5.1. المنهج ذو الأربع مستويات

الشكل التالي يوضح المستويات المختلفة للتجريد بالنسبة للمنهج ذو الأربع مستويات.

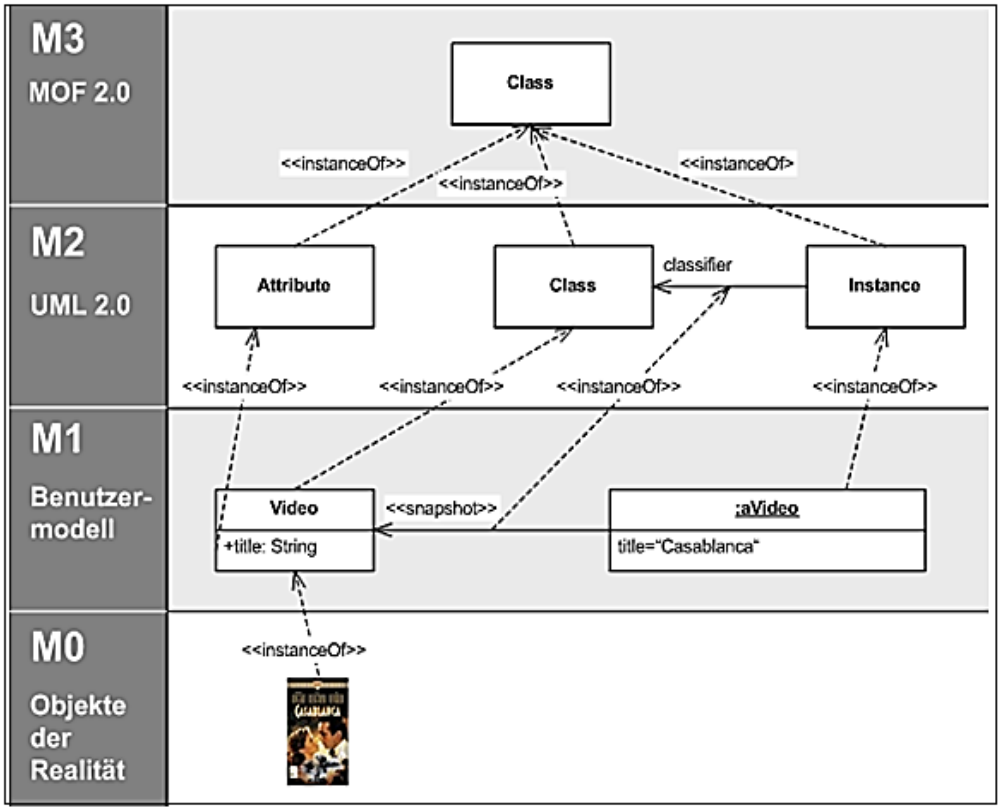
- **المستوى الأول:** مستوى البيانات أو كما يُعرَف M0.

  الكيانات هي كائنات زمن التنفيذ، مثلاً حالات (Instances) من الصفوف والعمليات التنفيذيّة في النظام الفعلي؛
- **المستوى الثاني:** مستوى النماذج، أو كما يُعرَف M1.

  يوجد نماذج مختلفة تقوم بتوصيف النظام والعلاقات الفعليّة، مثلاً: Employee, Employer هي صفوف لتصف العلاقة بين الموظف ورب العمل؛
- **المستوى الثالث:** مستوى التجريد الأول أو كما يُعرف النموذج المترفع M2.

  وهو يصف نموذج الكيانات والتي هي هنا على شكل صفوف وأغراض ومابينها من علاقات، حيث نعتبر أنّ Employee, Employer هما حالتان من الصف (Class) التابع لمستوى Meta Model أو M2؛
- **المستوى الرابع:** مستوى التجريد الثاني أو كما يُعرَف النموذج المترفع المترفع M3.

وهذا المستوى الإضافي للتعبير عن المفاهيم التي تعرّف اللغة المستخدمة، ويدعى Meta Meta-Model أو M3؛



الشكل (1-2) المستويات المختلفة للتجريد.

إذ تشكل طبقات النمذجة المترفعة المترفعة Meta Meta-modeling أساس معمارية النمذجة المترفعة، وأهم وظائف هذه الطبقة المجردة أن تقوم بتحديد اللغة المعتمدة لتعريف النماذج المترفعة، حيث أن النماذج المترفعة عن المترفعة تقوم بتوصيف النموذج على سوية تجريدية أعلى مما يفعل النموذج المترفع، وهي مضغوطة أكثر من النماذج المترفعة التي توصفها نموذجياً.

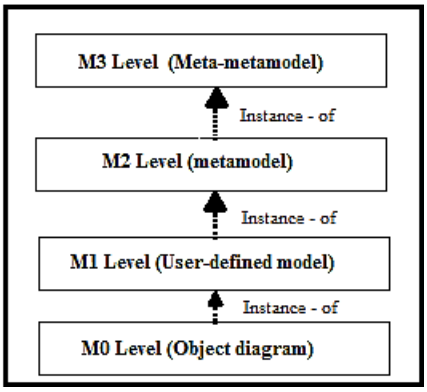
يمكن للنموذج المترفع عن المترفع أن يقوم بتوصيف عدة نماذج مترفعة في آن واحد، وبالعكس فإنه من الوارد أن يوصّف أحد النماذج المترفعة بأكثر من نموذج مترفع عن المترفع، وعادة في حال لم يكن لدينا نموذج صريح مترفع عن المترفع فإنه يكون ضمنيّ بحيث أن كل نموذج مترفع يكون محققاً لنموذج مترفع عنه أعلى منه تجريداً، ورغم أنه من المرغوب أن يكون للنماذج المترفعة meta models والنماذج المترفعة عنها meta meta-model مبادىء تصميمية واحدة لبنائها إلا أن هذا الأمر ليس قسريّاً، ولكن على كل طبقة من طبقات النمذجة أن تحافظ على سلامتها التصميمية وكمالها.
من الأمثلة المطروحة عن طبقات النمذجة المترفعة عن المترفعة :

- MetaClass ;
- MetaAttribute ;
- MetaOperation.

وهكذا، فإن كل نموذج model سيكون عبارة عن نسخة من النموذج المترفع الموافق له، والمسؤولية الرئيسة لطبقة النموذج أن تقوم بتحديد لغة مناسبة بغية توصيف نطاق معلومات ما Information Domain.

اتسع استخدام النمذجة المترفعة شيئاً فشيئاً في مجالات هندسة البرمجيات، بخلاف تقنيات تطوير البرمجيات الكلاسيكية المعتادة والمعتمدة على الرماز المصدري بالدرجة الأولى، وفي هذه الحالة يكون لدينا نموذج متكيف ومتطابق تماماً مع النموذج المترفع الموافق، وأحد أكثر أفرع الهندسة المقادة بالنماذج نشاطاً فرع (MDA) Model Driven Architecture أي المعماريّة المُقادة بالنماذج والمقترحة من قبل اتحاد (OMG) Object Management Group (والهندسة المقادة بالنماذج MDE هي نهج تطوير برمجي يركز على إنشاء واستثمار نماذج النطاق Domain Models بدلاً من التركيز على الحوسبة والمفاهيم الخوارزمية)، وتعتمد هذه المبادرة في لبها على اللجوء إلى معيار مخصص من أجل كتابة وصياغة النماذج المترفعة ويدعى(MOF) Meta Object Facility، وقد تم تصميم النموذج المترفع الخاص بلغة النمذجة الموحدة UML بحيث يمكن استنساخه من النموذج المترفع عن المترفع الخاص بـ MOF المقترح من قبل اتحاد .OMG، وهي طريقة قياسيّة تسمح بالتوصيف التدريجي للمشكلة، ومن ثم تصف الذهاب إلى كيفيّة حلها. وهي متوافقة تماماً مع المنهج ذو الاربع مستويات الذي استعرضناه سابقاً. والشكل (1–3) يوضح الفكرة.

ونشير أن من النماذج المترفعة المقترحة من قبل اتحاد OMG: نماذج UML و SysML و SPEM المدعو أيضاً CWM.



الشكل (1–3) الطريقة القياسيّة المعتمدة MOF

2.5.1. مستويات التكامل

إنّ الفائدة الأساسيّة للنماذج المترفّعة القائمة على التكامل أنّ القيود العامة الثابتة يمكن أن تتوضح في هذا المستوى، إذ يتم في هذا المستوى تعريف عناصر النمذجة، وقيود التكامل، والسلوك الديناميكي لعناصر النمذجة.

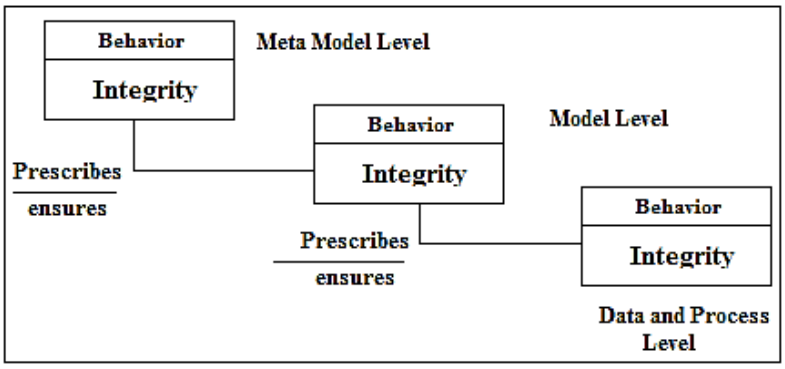
وبما أنّ النموذج الفعلي هو حالة من النماذج المترفّعة فإن القيود والسلوك مصانة، لذلك بشكل عام فإنّ القيود العامة الثابتة مثلها مثل عناصر النمذجة يمكن تعريفها وفهمها في مستوى النماذج المترفّعة، هذه القيود يجب ان تعرض مرة واحدة فقط، وتبقى لكل النماذج المحتملة.

الشكل التالي يوضح مستويات التكامل.

يمكن أن نميّز مابين قيود التكامل الثابتة وقيود التكامل الديناميكيّة،

القيود الثابتة: تعرّف مجموعة من الحالات المحتملة للنموذج الفعلي؛

القيود الديناميكيّة: فهي تقيّد سلوك كيانات النموذج.



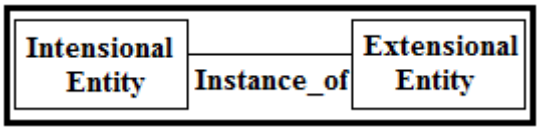
الشكل (1-4) مستويات التكامل.

3.5.1. الكيانات الداخليّة والخارجيّة

من المهم التمييز بين الكيانات الداخلية والخارجية، **فالكيانات الداخليّة** تملك نسخة مطابقة من القواعد اللغوية للنموذج مثل صف أو علاقة، بعكس الكيانات الخارجية تستخدم من أجل تخزين معلومات زمن التنفيذ الأساسية، مثل غرض أو رابطة. على حين أنّ **الكيانات الخارجيّة** عادة تكون متقاربة بشكل ثنائي مع الكيانات الداخليّة، بحيث أن هيكل الكيان الخارجي يكون موصف عن طريق كيان داخلي، مثال: هيكلية الغرض تعطى من قبل صف.

وإنّ واحدة من أهم ميزات النمذجة المترفّعة، هي القدرة على تعريف العلاقة بين الكيانات الخارجيّة والداخليّة باستخدام مستوى واحد. في هذه الحالة يكون المفهوم الدلالي للكيان الداخلي قابل للتفسير عن

طريق مجموعة من الكيانات الخارجيّة المقابلة. وإنّ تعريف المفهوم الدلالي عن طريق مستوى واحد يعتبر ميزة مهمة في النمذجة المترفّعة.



الشكل (1-5) العلاقة مابين الكيانات الداخليّة والخارجيّة.

6.1. سلبيات الهندسة المقادة بالنماذج

1- نتائج التجريد غير مضمونة، إذ ليس من الضرورة أن يحقق برامج أفضل. فهو قد يؤدي إلى نتائج سلبية، وذلك لأنّ التفكير بالتجريد يعتبر من الأمور الأكثر صعوبة خاصة مع ميل المبرمجين والمهندسين إلى استعمال الأمثلة والمحاكاة بشكل عملي أكثر من ميلهم للمفاهيم المجرّدة؛

2- تمتلك الهندسة المقادة بالنماذج سلسلة من النشاطات المتتابعة ذات التأثيرات السلبيّة والإيجابيّة، فمثلاً مسألة توليد الرماز تعتبر ذات تأثيرات إيجابيّة للوهلة الأولى بالنسبة لتوفير الوقت والجهد على المبرمج، إذ أنّه يكفيه عناء تعلّم التقنيّة، لكن بالمقابل هناك جهد إضافي مطلوب لتطوير النماذج لتجعل توليد الرماز ممكناً، وجهداً أكبر لتوفير إمكانية تماشيه مع التعديلات اليدويّة، ناهيك عن صعوبة تكامله مع الأنظمة الموجودة. وبالتالي الموازنة ما بين التأثيرات السلبيّة والإيجابيّة يعتبر تحديّاً يعتمد على السياق، والذي قد يؤدي إلى حمل إضافي لايمكن تقديره في البداية؛

3- هناك الكثير من الأنواع المختلفة من الهندسة المقادة بالنماذج، وأغلب الشركات تواجه مصاعب في اختيار النموذج المناسب لها وفقاً لأغراضها التجاريّة، وبالتالي فوائدها وعوائقها غير واضحة وهي في الواقع تعتمد على السياق بشكل رئيسي، وبالتالي الاختيار الصحيح سيكون حاسماً لنجاح المشروع.

الجدول التالي يلخص الإيجابيات والسلبيات [1] لاستعمال الهندسة المقادة بالنماذج.

الجدول (1-1) الإيجابيات والسلبيات للهندسة المُقادة بالنماذج

العوامل المؤثرة	التأثيرات الإيجابيّة	التأثيرات السلبيّة
الانتاجيّة وتقاس بـ:		
الزمن اللازم للتطوير	ينقص الزمن الازم للتطوير بسبب تقنيات توليد الرماز التلقائي.	بالمقابل: عملية توليد نماذج مقروءة من قبل الحاسب وتحقيق انتقالات النماذج تستهلك وقتاً كبيراً.
الزمن اللازم لاختبار الرماز	ينقص الزمن اللازم للاختبار لأن نسبة الأخطاء تقل بسبب التوليد التلقائي للرماز وتكون مجرد أخطاء عابرة يمكن تلافيها بسهولة.	بالمقابل: يزداد الوقت في عملية اختبار انتقالات النماذج والتأكد من صلاحيتها.
عائد استثمار جهود النمذجة (ROI)	يزداد المردود أو العائد بشكل كبير إذا شكلت عملية النمذجة بالنهاية المزيد من الحلول المبدعة التي تسمح للمطورين برؤية الصورة المتكاملة لأنظمتهم.	هناك ما يسمى بشلل النموذج أو (Model Paralysis) وهي نماذج فاشلة تعيق العمل ولا تعطي مردود جيد.
المحموليّة وتقاس بـ:		
الزمن اللازم لتكامل النموذج مع منصات عمل جديدة.	ينقص الزمن اللازم للتكامل وذلك لأننا فقط نطبق مجموعات جديدة من الانتقالات.	بالمقابل: يزداد الوقت والجهد اللازم لتطوير مجموعات جديدة من الانتقالات أو لتخصيص مجموعات موجودة مسبقاً.
الصيانة وتقاس بـ:		
الزمن الذي يتطلبه أصحاب	ينقص الزمن إذ أنه من	يزداد الوقت في عملية فهم

المنفعة ليتم فهم التطبيق.	السهولة بمكان فهم التطبيق والأنظمة بالنسبة للكوادر الجديدة خاصة أنّ الأنظمة عادةً تكون موثقة.	الرماز من قبل الكوادر الجديدة لأنه مولد بشكل آلي.
الزمن المطلوب لصيانة البرمجيّة.	ينقص زمن الصيانة طالما هي في سوية النماذج ويتم توليد روابط للتتبع بشكل تلقائي.	يزداد زمن الصيانة حالما انتقلنا إلى سوية الرماز وذلك لحاجتنا إلى إجراء تزامن ما بين السويتين (الرماز والنموذج)

(وكنتيجة لهذه العوامل فإنّه لايوجد إطار عمل واضح لصنع القرار، إذ لا يمكن أن ينبئنا فيما إذا كانت الهندسة المقادة بالنماذج ستنجح أم ستفشل دون الرجوع إلى السياق ودراسة العوامل المؤثرة وحجم العمل.)

7.1. التجارب السابقة في استخدام الهندسة المقادة بالنماذج

لا يوجد حتى الآن مستويات موسّعة من العمل البرمجي والنظاميّ حول دراسة تأثير الهندسة المقادة بالنماذج.

على سبيل المثال: لا يوجد استبيانات واسعة الانتشار حول تنظيمها في مجال الاستخدام الصناعي، على الرغم من أهمية الاستبيانات في إزالة عملية الالتباس وسوء الفهم [4].

من أهم **الاستبيانات المطروحة** في هذا المجال:

- استبيان حول دخول الـ UML وأدواتها إلى السوق التجاريّة [5]، وهي في لب موضوع الهندسة المقادة بلنماذج تمّ نشر الاستبيان عام 2005؛
- أجرى كل من Forward و Lethbridge استبياناً OnLine [6] لمعرفة آراء ومواقف المهنيين حول النمذجة.

أما بالنسبة **للتقارير والأبحاث المنشورة** فهي ذات عدد متواضع جداً من أهمها:

- تمّ توثيق "حالة استخدام" للانتقال من البرمجة المركّزة على الرماز إلى البرمجة المركّزة على النماذج من قبل Afonso [7]؛
- أجرت Anda [8] تقريراً سرديّاً حول ميزات النمذجة، مثل: تحسين إمكانيّة التتبع، ولكنها أشارت إلى النقاط السلبيّة حول زيادة الوقت لنكامل الرماز القديم أو الموروث مع النماذج، وإجراء التغيرات التنظيميّة لتلائم النمذجة؛
- هناك عدة أبحاث حول تطوير UML [9, 10, 11, 12] تدرس محاولة للقياس التجريبي لوضوح نماذج UML [13, 14, 15, 16, 17, 18, 19]، إذ تُعتبر كمجال تجريبي يعمل على تقييم تقنيات هندسة البرمجيات بالتعاون مع الـ UML، غير أنّ الـ UML ماهي إلاّ مثال عن لغة النمذجة،
- زوّد كل من France و Rumpe [20] بنظرة عامة حول البحث في الهندسة المقادة بالنماذج، وأشارا إلى بعض المشاكل "wicked problems" المضمّنة بعيداً عن التطبيقات الصناعيّة.
- قام كل من Mohagheghi و Dehlen [21] بتحليل مترفع (Meta-Analysis) للهندسة المقادة بالنماذج حول ماهية التقنيات المطبّقة في الشركات ذات المجالات المختلفة، ركزت دراستهما على توليد النماذج والرماز والعناصر الأوليّة أكثر من كونها تركز على الهندسة المقادة بالنماذج أكثر من كونها تركز على النمذجة بشكل عام إذ عمدوا إلى تقييم الرماز فتراوحت التقييمات مابين درجات 65% –100 %.

أما **<u>الدراسات التجريبيّة</u>** في هذا المجال [1] فقد اعتمدت على ثلاثة منهجيات تمّ اتباعها لأداء التقييم:

- ✓ استبيان نوعي موجّه لشريحة كبيرة من العاملين في مجال الهندسة المقادة بالنماذج؛
- ✓ مقابلات مع خبراء للهندسة المقادة بالنماذج في عدة شركات؛
- ✓ المراقبة والمتابعة للشركات العاملة في مجال الهندسة المقادة بالنماذج.

ويتوقف هدف الدراسة التجريبيّة عادةً على نقطتين أساسيتين هما:

- ✓ فهم آلية عمل الهندسة المقادة بالنماذج وتطبيقها في المجال الصناعي؛
- ✓ تعريف أهم العوامل المؤثرة على الهندسة المقادة بالنماذج.

من أهم الدراسات التجريبيّة هي:

- دراستان تجريبيتان [21,22] أجرتهما شركة "Middleware" باستعمال الحاسوب (2003–2004) قاست فيهما التطور وعدد مرات الصيانة لمخزن للحيوانات الأليفة على النت وذلك

باستخدام الـ MDA والـ IDE ، كلا الدراستان أكدتا على زيادة 35% على الانتاجيّة و37% على الصيانة.

- دراسة تجريبيّة رائدة من نوعها في تطبيق الهندسة المقادة بالنماذج في شركات السيارات، الاتصالات، وشركات الطباعة [1] حيث ركزت على الفائدة الناتجة عن استخدام الهندسة المقادة بالنماذج والتي أفادت بـ:
  - ✓ التدرج في حل المشكلة؛
  - ✓ الالتزام بحل المشكلة أي (دعم عدة مناهج بسلوك مستمر) ؛
  - ✓ التكيف مع المشاكل المختلفة بهدف حلها؛
  - ✓ يعمل على حل مشاكل الاختلاف في التقنيات؛
  - ✓ الفصل مابين اهتمامات النظام؛
  - ✓ إمكانيّة إعادة الاستخدام؛
  - ✓ تمثل المرجعيّة المعماريّة للنماذج والمشاريع؛
  - ✓ الاستجابة السريعة للتغيرات الحاصلة على المتطلبات؛
  - ✓ تعتبر الهندسة المقادة بالنماذج كحل لمواجهة التغيرات التنظيميّة والتجاريّة؛
- مشروع SaltNPepper [23]: وهو مشروع مفتوح المصدر مطور من قبل جامعة Humboldt في برلين وهو مشروع يستخدم تقنية النمذجة المترفّعة في التعامل مع البيانات الموصفة لغوياً. إذ توجد البحوث اللغوية بأنواع مختلفة من التنسيقات ، لكن لاتوجد طريقة موحدة في معالجتهم، لذلك تمّ تطوير نموذج مترفّع يدعى Salt والذي يعتبر تجريد للبحوث اللغوية، إذ يصف الحد الأدنى من الدلالات، ويعالج فقط بنية البيانات اللغويّة، فهو يقدم نموذج مترفع لتمثيل البيانات اللغوية في الذاكرة الرئيسيّة، والتعامل معها.
- تطبق الهندسة المقادة بالنماذج في مجال الشركات التجارية والمالية، شركات الاتصالات، وشركات الدفاع والطيران وتطبيقات الويب [1].

8.1. الخلاصة

- إنّ عدد الأبحاث التي تركز على معاملات تنظيميّة وهندسية حول تبني واستخدام الهندسة المقادة بالنماذج وتقييمه لايزال محدود جداً، إذ أنّ معظم الدراسات تركز على السمات التقنيّة لها وتفتقر إلى الخبرة العمليّة، والخبرة الصناعيّة، وتفاصيل حول استخدامها، وبالتالي هناك 3 فجوات في المنهج الحالي للهندسة المقادة بالنماذج:

- نقص المعرفة عن كيفيّة استخدامها في المجالات الصناعيّة.
- نقص في الفهم عن كيفيّة تأثير العوامل الهندسيّة على استخدامها.
- فشل UML في تقييم سماتها مثل:
  - فوائد توليد الرماز؛
  - تجريد المجال المحدد؛
  - انتقالات النماذج.

- هناك تركيز واضح على مناهج UML والتي بدورها ما هي إلاّ مثال عن لغة نمذجة، على حين الهندسة المقادة بالنماذج أكثر شمولاً، إذ تشمل:
  - لغات نمذجة متعددة (كل واحدة لمجال مختلف)؛
  - تعمل على أتمتة انتقال النماذج مابين لغات النمذجة؛
  - تؤكد على تعدد وجهات النظر للمشروع الواحد:
    - النموذج المعتمد على إطار العمل Platform Specific Model (PSM)؛
    - النموذج المستقل عن إطار العمل Platform Independent Model (PIM).

فبينما معظم العمل الموجود يركز على ملائمة لغة UML، لايحدث أي عمل للتركيز على الهندسة المقادة بالنماذج باعتبارها ذات قضايا أوسع وأشمل، مثل:

- فيما إذا كان الرماز المولّد يعمل أم لا ؟
- ماهي العلاقة بين اللغات المحددة السياق (المجال) ومابين لغات الأهداف العامة ؟
- هل تتوافق الهندسة المقادة بالنماذج مع الهيكليّة التنظيميّة الموجودة ؟

- وفي سياق جودة البرمجيات نحاول تخفيض العيوب والتقليل منها، وإنّ هذه العمليّة تتطلب تفتيش الرماز وصيانته مراراً، القليل من الأوراق البحثية المنشورة تناقش حالات التحليل للرماز والتصميم، وهي تؤمن إثبات تجريبي قوي وتأثير إيجابي على الإنتاجيّة.

رغم أنّ الدراسة التحليليّة والاهتمام بالتفاصيل قد يخلق تحيّزاً وقد يؤدي إلى تصادم مع إمكانية إثبات الوثوقيّة. غير أننا سنحاول تعميم الاستنتاجات ونتائج التجارب حتى تكون ممكنة الاستعمال في الحالات المشابهة، إذ سنعمل على الدمج الانتقائي لمناهج الأبحاث الخاصة بالهندسة المقادة بالنماذج، والاستفادة من النتائج السابقة ودعم النتائج التي سنتوصل إليها من خلال:

- استبيان مكوّن من مجموعة أسئلة يوجّه للخبراء والعملين بمجال الهندسة المقادة بالنماذج؛

- منهج لوضع الملاحظات وهي ستشكل التغذية الراجعة؛
- كتابة التقارير من قبل شريحة كبيرة من الطلاب في مرحلة ما قبل التخرج لشرح العقبات والتسهيلات التي تواجههم أثناء التعامل مع المنهج المقترح؛
- الحصول على حالات استخدام نصف مهيكلة مستنتجة من المقابلات المباشرة.

وبذلك نتزود بالتفاصيل بشكل دقيق، ونحصل على وصف سياقي ونحلل ظواهر دورة الحياة الحقيقيّة، **ويكون لدينا البرهان القوي على تحسّن الجودة باستخدام الهندسة المقادة بالنماذج.**

الفصل الثاني
عيوب الرماز والتصميم

1.2. مقدمة

تُعتبر العيوب Smells بأنها المؤشر إلى وجود مشكلة ما في مكان محدد من الرماز أو التصميم، تؤدي إلى تراجع في جودة المنتج البرمجي وهدر الوقت لتعديله أو صيانته وصعوبة التعامل معه، لذا لابدّ من تعقب هذه المشكلة بهدف حلها وتلافي وقوع مشاكل أخرى وبالتالي تحسين المنتج البرمجي.

2.2. أنواع العيوب

- عيوب الرماز (Code Smells): وهي العيوب ذات السوية المنخفضة تتعامل على نطاق الرماز فقط؛
- عيوب التصميم (Design Smells): وهي العيوب ذات سوية منخفضة أيضاً، تتعامل مع مستوى الصفوف، ولا علاقة لها بالرماز. إنّ تجمّع عدد كبير من عيوب الرماز يسبب بدوره عيب في التصميم.

3.2. دراسات سابقة

1.3.2. دراسات سابقة حول عيوب الرماز

أول من قدّم تعريف لعيوب الرماز هو Kent Beck، وأما Fowler فقد عرّف ما يقارب 22 من عيوب الرماز، وعرّف David Jonsson ثلاثة من أهم العيوب، وكذلك Marco Zanoni فقد عرّف ستة من هذه العيوب.

تؤثر عيوب الرماز على أداء النظام بشكل عام وتؤدي إلى انخفاض في الجودة العامة له، وبطئ في عملياته، وصعوبة في فهمه [24]، ولها الكثير من الأنواع نذكر أهمها:

- **الرماز المكرر Duplicated Code**

وهو تكرار نفس البنية في أكثر من مكان وبالتالي فمن الأفضل توحيدهم في بنية واحدة لتحسين البرنامج؛

- ☒ أبسط مشكلة من الرماز المكرر عندما يتواجد نفس التعبير في تابعين ينتميان لصف واحد والحل هو باستخراج تابع واستدعاؤه من مكاني التكرار؛
- ☒ مشكلة أخرى شائعة هي تواجد نفس التعبير في صفين أبناء بينهما علاقة (Sibling) يمكن إزالة هذا التكرار باستخدام استخراج التابع (Extract Method) ثم سحب الحقول المشتركة إليه؛
- ☒ إذا كان الرماز متشابه لكن غير مطابق نستخدم استخراج التابع لفصل التشابه عن الاختلاف ثم نستخدم Form Template Method؛
- ☒ إذا تواجد نفس الرماز في صفين مستقلين عن بعضهما نقوم باستخراج صف في واحدة من الصفوف ثم نضع المكون الناتج بالصف الآخر.

- **التوابع الطويلة Long Method**

ترتبط جودة البرامج بتوابعها القصيرة، حيث كلما كانت الإجرائية طويلة كلما كان فهمها صعباً، إن المفتاح الأساسي لفهم الإجرائيات الصغيرة هو بالتسمية الجيدة، حيث يتيح لك الاسم الجيد لتابع عدم الاضطرار للنظر للرماز الداخلي له.

- ✓ لتصغير الإجرائية نستخدم استخراج التابع، أي نجمع الأجزاء الملائمة مع بعضها ونضعها في تابع جديد؛
- ✓ إذا تضمن التابع الكثير من الوسطاء والمتحولات المؤقتة سينتهي بنا الأمر عند استخدام استخراج التابع بتابع جديد يحوي المتحولات المؤقتة بالإضافة للوسطاء الأصلية وبالتالي نادراً ما يكون قراءته أسهل من التابع الأصلي؛
- ✓ من التقنيات الجيدة النظر إلى التعليقات، فكتلة الرماز المشار إليها بتعليق يوضح ما تفعله، نستبدلها بتابع له اسم معتمد على التعليق يقدم نفس الغرض وكذلك السطر الواحد الذي له تعليق نعامله بنفس الأسلوب. أما بالنسبة للحلقات فنحول الحلقة وما بداخلها الى تابع خاص بها.

- **قائمة وسطاء طويلة Long Parameter List**

في البرمجة غرضية التوجه قائمة الوسطاء تكون أصغر مما هي عليه في البرمجة التقليدية، وهذا جيد لأن قائمة الوسطاء الطويلة صعبة الفهم لأنها تصبح متناقضة وصعبة الاستعمال، ولأن

المستخدم يغيرهم كلما احتاج إلى المزيد من البيانات. أغلب التغيرات تحدث بتمرير الأغراض وذلك لأننا كمستخدمين نفضل أن ننفذ فقط طالبين للحصول على البيانات التي نحتاجها.

- **الصف الكسول Lazy Class**

إنّ إنشاء صف يتطلب بعض الجهد والوقت، وإذا لم نعد بحاجة لأن نستخدم هذا الصف يصبح عبء أكثر منه فائدة وهذا ما يُعرف باسم Lazy Class؛

- **الحقول المؤقتة Temporary Field**

يجب أن تكون جميع المتحولات ضمن الغرض مستخدمة ومفيدة، تحدث حالة الحقل المؤقت عندما تحتاج خوارزمية معقدة كثير من المتحولات ولا تريد تمريرهم كوسائط فتضع هذه المتحولات في حقول مؤقتة؛

- **صف المعطيات Data Class**

يعرّف صف البيانات بأنّه الصف الذي يحوي عناصر وتوابع تهيئة ووصول لها بنسبة 90% من عملياته على حين يجب أن يحوي الصف توابع سلوك تختلف عن توابع التهيئة والتعبئة؛

- **وجود بيانات غير مهيكلة Un Capsulated Attributes**

وهذا يشكل ثغرة أمنية في النظام، إذ لابدّ من كبسلة كل العناصر بعمليات Set/ Get بهدف ضبط عمليات الوصول إليها؛

- **عمليات اختبار الأنماط بالعملية Switch**

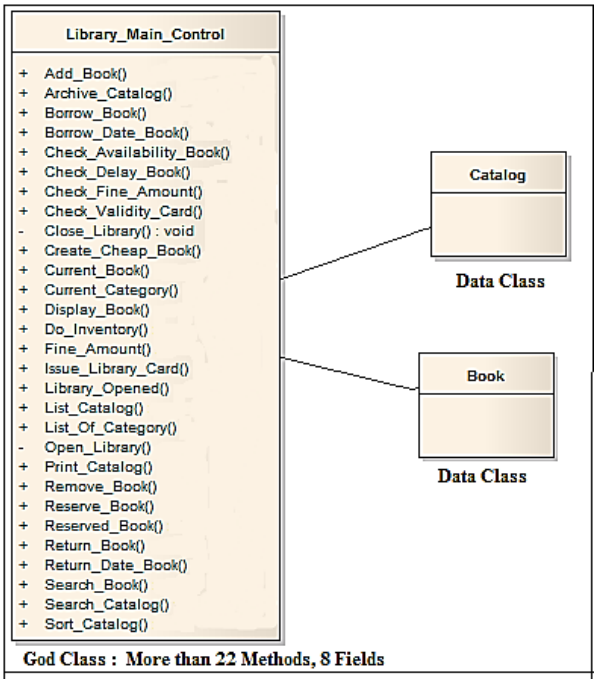
وهذا يُعتبر مثالاً للبرمجة الضعيفة الهيكلة، إذ لابدّ من استبدالها بعمليات Polymorphism المخصصة لتعدد الأنماط.

2.3.2. دراسات سابقة حول عيوب التصميم

إنّ عيوب التصميم هي عيوب قاتلة، تلعب دوراً حاسماً بالتأثير على جودة النظام، وإنّ تجمع عدد كبير من عيوب الرماز يؤدي بدوره إلى عيوب التصميم، وهناك عدد واسع من هذه العيوب [25] ، نذكر أهمها:

- **الصف الكبير God Concept**
  - صف وحيد يحوي عدد كبير من الحقول والعمليات يتجاوز عددها 30؛
  - نقص عام في تماسك الحقول والعمليات حيث تمثل مجموعة متباينة غير مرتبطة مغلفة في صف واحد؛
  - صف معقد من حيث إعادة الاستخدام والاختبار؛
  - صف مكلف من حيث التحميل إلى الذاكرة، بسبب استخدامه موارد مكلفة حتى للعمليات البسيطة [26].

الشكل التالي يوضّح المفهوم.

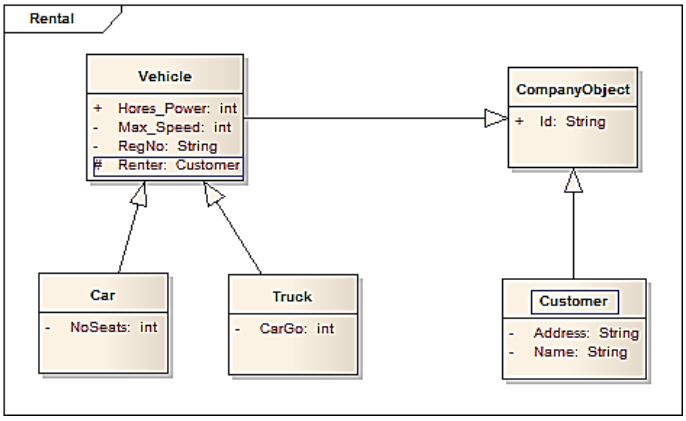


الشكل (2-1) مفهوم الصف الكبير

- **مفهوم العنصر Attribute Concept**

وهي أن يحوي أحد المفاهيم مفهوماً آخر بداخله ، هذا يعتبر من عيوب النظام ويشكل حملاً ثقيلاً [27]. مثال: لو كان لدينا نموذج لشركة تأجير السيارات فهي تحوي على مفاهيم من نوع سيارة، زبون، شركة...

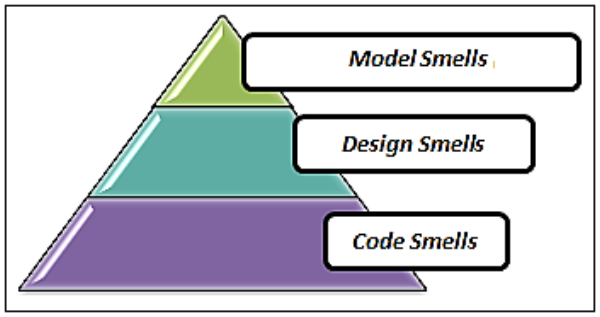
الشكل التالي يوضح المفهوم.



الشكل (2-2) التصميم الخاص بشركة السيارات

لابدّ من الاستغناء عن العنصر Renter الذي هو من نوع Customer في المفهوم Vehicle والتصحيح يكون بجعل علاقة Association بين Vehicle و Customer. ففي هندسة البرمجيات نعمد دوماً إلى تفصيل العمل وتقسيمه والابتعاد عن التداخلات البرمجية والتصميميّة.

وهناك العديد من عيوب التصميم الأخرى سنأتي على ذكرها بشكل موسّع في الفصول التالية، لكن الجدير بالذكر أنّ تجمع عيوب الرماز تسبب عيوب التصميم وتجمع عيوب التصميم تسبب عيوب النموذج فالعلاقة هرميّة يوضحها الشكل التالي.



الشكل (2-3) سويات العيوب

(لمّا كانت سوية النماذج المترفعة هي الأعلى من هذه السويات وهي المسؤولة عن توليد النماذج (الأدوات) والتي بدورها تعمل على انتاج البرمجيّة، رأينا أنّه من الضرورة بمكان ضبط هذه العيوب في السوية العُليا أي سوية النماذج المترفعة، وذلك في محاولة لإحكام السيطرة على النماذج والأدوات البرمجيّة المولّدة، وبالتالي نضبط عملية التوليد والإجرائيات ونضمن عدم وجود مثل هذه العيوب في السويات الدنيا، لأننا نكون قد تلافيناها من مصدرها.)

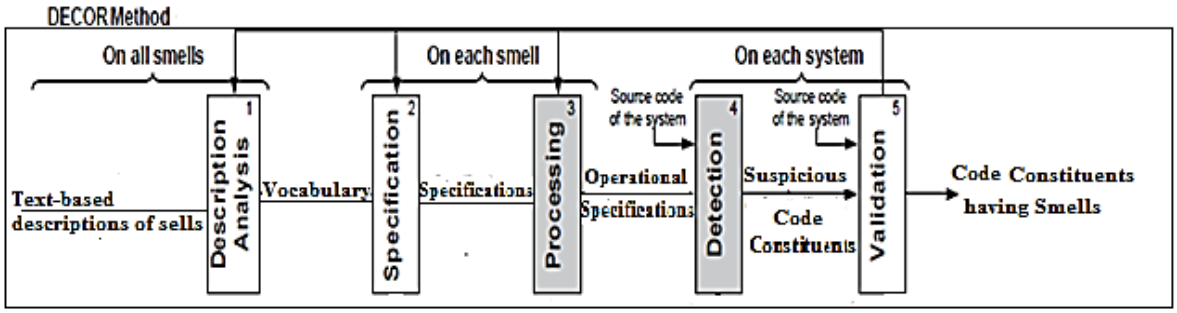
لذا سنعتمد على دراسة هذه العيوب بشكل موسع في السويات الدنيا (الرماز + التصميم) ثم نعمل على تمديدها لتشمل السويات العُليا (النماذج المترفعة).

3.3.2. نموذج لكشف عيوب الرماز والتصميم (DECOR)

وهو نموذج يعمل على نمذجة العديد من عيوب التصميم والرماز معاً[27]، وتتدرج إجرائياته وفق عدد من الخطوات، أهمها:

1- يتم تعريف المفاهيم المفتاحية للعيوب؛
2- تحديد مواصفات العيوب؛
3- المعالجة ونقلها إلى خوارزمية؛
4- اكتشاف العيوب من خلال الاستعانة بالقواعد السابقة ومصادقتها.

والشكل التالي يبيّن المخطط العام لخطوات عمل نموذج DECOR



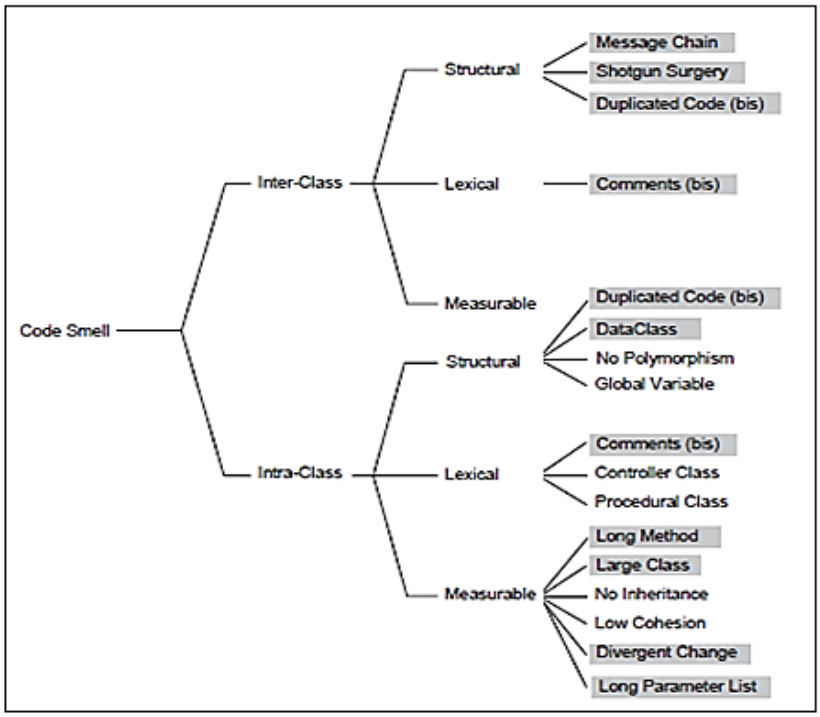
الشكل (4.2) المخطط العام لخطوات عمل نموذج DÉCOR

- **مجموعة عيوب الرماز المعرّفة في DÉCOR**

يعرف مجموعة واسعة من عيوب الرماز تتدرج تحت عدة مستويات [27]:

A. **العيوب الهيكيلية:** ولها علاقة ببنية الصفوف؛
B. **العيوب المعجمية:** وتشمل التوصيفات كالتعليقات وصفوف التحكم والإجرائيات؛
C. **العيوب التي تتعلق بالمقاييس:** كعدد العناصر وطول التوابع والتماسك مابين العناصر.

الشكل التالي يوضح عيوب الرماز المعرّفة وأنماطها.



الشكل (2-5) عيوب الرماز المعرّفة وأنماطها.

- **مجموعة عيوب التصميم المعرّفة في DÉCOR**

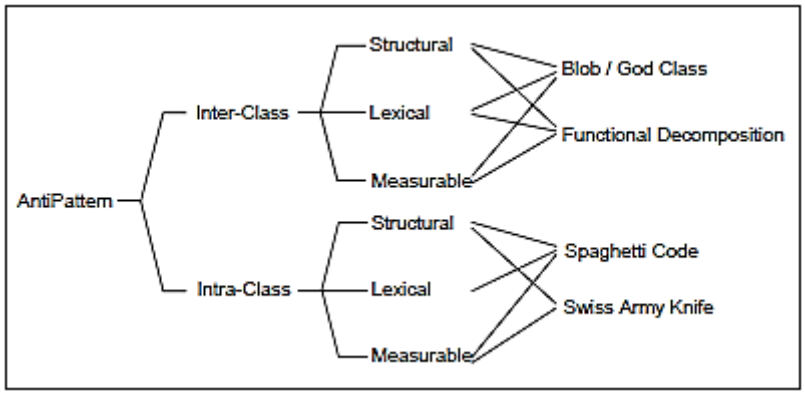
وأما عيوب التصميم أو النماذج التصميمية السيئة تندرج ضمن نفس التقسيم

A. **العيوب الهيكيلية:** ولها علاقة بتصميم الصفوف تشبه God / Blob Class.

B. **العيوب المعجمية:** مثل Functional Decomposition.

C. **العيوب التي تتعلق بالمقاييس:** كعدد العناصر وطول التوابع والتماسك مابين العناصر.

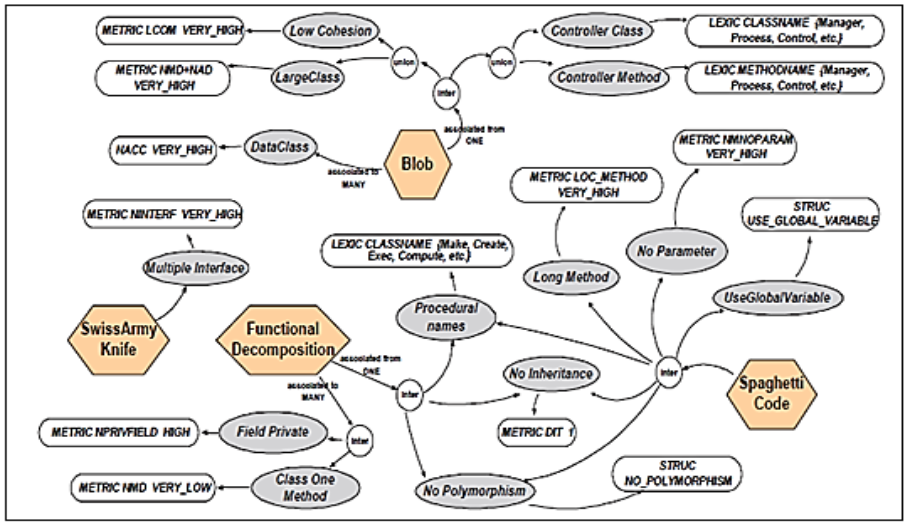
الشكل التالي يوضح عيوب التصميم المعرّفة وأنماطها.



الشكل (2-6) عيوب التصميم المعرّفة وأنماطها

- **المخطط الكلي للعيوب التي يكتشفها DÉCOR**

إنّ تجمع مجموعة من عيوب الرماز يؤدي إلى عيوب التصميم، والشكل التالي يوضح مجموعة العيوب كاملة التي يكتشفها نموذج DECOR.



الشكل (2-7) مجموعة العيوب كاملة التي يكتشفها نموذج DÉCOR

4.2. الخلاصة

(إن عيوب النظام سواءً كانت على مستوى الرماز أو على مستوى التصميم تؤثر سلباً على الجودة، لذا كان لابدّ من إزالتها ومعالجتها بهدف زيادة الجودة، وتحسين الإنتاجيّة. سنعمد في مشروعنا إلى اكتشاف وتصحيح العيوب لكن من مستوى النماذج المترفعة لضبط كامل عملية توليد البرمجيات وضمان خلوها من العيوب.)

الفصل الثالث

التصحيح وإعادة البناء

1.3. مقدمة

إنّ عملية إعادة البناء والتصحيح هي عملية التغيير التي نجريها على البنية الداخلية للبرمجيات ليست فقط لتخليص الرماز والنماذج من الأخطاء، بل هي أبعد من ذلك فهي لجعل المنتج البرمجي أسهل للفهم وأقل كلفة للتعديل، مع بقائه يلبي الوظيفة التي صُمِم من أجلها دون تغيير السلوك الخارجي له، فهذه العملية تزودنا بالتقنيات اللازمة لتخليص المنتج البرمجي من الأخطاء بطريقة احترافية وأكثر فعالية إذ يمكن التحكم بها يدوياً وآلياً.

2.3. أهمية عملية إعادة البناء والتصحيح

تكمن أهمية عملية التصحيح في مجموعة من النقاط، أهمها:

- تحسين تصميم البرمجيات Refactoring Improves the Design of Software

تسهم عملية التصحيح وإعادة البناء في ترتيب بنية الرماز والتصميم وإزالة الأجزاء التي ليست في مكانها الصحيح، فهي تساعد على استعادة الشكل النموذجي والحفاظ على التصميم من خلال إزالة الأجزاء المكررة منه، وإجراء عمليات التقسيم المناسب للنماذج، وهذا يقلل من حجمه مما يسرع من تنفيذه و يزيد من إمكانية تعديله، والذي يؤدي إلى تصميم جيد.

- يساعد على اكتشاف الـ Bugs في البرنامج Refactoring Helps You Find Bugs

بعد إجراء عملية التصحيح يصبح التصميم أسهل للفهم إذ نستطيع معرفة الـ Bugs وإدراكها ونصبح أكثر خبرة وفعالية في عملية البرمجة.

- يساعد على البرمجة السريعة Refactoring Helps You Program Faster

عملية التصحيح التي يخضع لها التصميم تساعد على تطوير الرماز بسرعة أكبر، إذ يسهل عملية كتابة الرماز ويقلل عدد الـ Bugs بشكل ملحوظ، فنحن لسنا بحاجة لإضافة خصائص

جديدة أو إجراء تغيرات تحتاج لوقت طويل لفهم كيفية عملها. فالتصميم الجيد ضروري لتسريع عملية التطوير ومنع تصميم النظام من التدهور وتحسين تصميم البرنامج مما يزيد من سرعته على المدى البعيد.

3.3. دراسات سابقة

تمّ تقديم مصطلح إعادة البناء والتصحيح لأول مرّة عام 1992 من قبل Opdyke [28] حيث قدّم 26 تصحيح منها 3 مركبة تأخذ حيّز التصحيح التصميمي والباقي 23 تأخذ حيّز التصحيح الرمازي.

وعرّف Martin Flower مجموعة من التصحيحات اليدويّة [29] واقترح تصنيف لهذه التصحيحات.
أمّا Kent Beck فأسهم في اقتراح طرق اكتشاف العيوب الخاصة بالرماز[29] .
و سعى Tichelaar إلى إيجاد لغة مترفعة النماذج أسماها FAMIX لكي تمثّل التصحيح الذي يطرأ على النماذج والبرامج الغرضيّة التوجه [30,31].
وهناك العديد من الآليات للتصحيح مثل Refactoring Browser [32] و XRefactory [33].
تخضع عملية التصحيح وإعادة البناء إلى طرق وأساليب متنوعة تتدرج بشكل رئيسي تحت نوعين أساسيين، وهما:

- التصحيح اليدوي؛
- التصحيح الآلي.

ويتم تطبيقها على عدد كبير من المستويات، أهمها:

- مستوى قواعد البيانات؛
- مستوى الواجهات؛
- مستوى الرماز؛
- مستوى التصميم.

1.3.3. مستويات إعادة البناء والتصحيح

- **إعادة البناء على مستوى قواعد البيانات**

من أصعب التحديات التي تواجه المطورين هي عملية تغيير قواعد المعطيات لأن معظم التطبيقات مرتبطة بها وتدعمها، والأصعب هو ترحيل بياناتها (Migration)، فحتى لو أمكننا

تغيير مخطط قاعدة المعطيات وترتيب العلاقات فيها، مازلنا بحاجة لترحيل البيانات السابقة وهذه عملية متعبة ومكلفة زمنياً. يضاف إليها كلفة إجراء التعديلات اللازمة على الصفوف المرتبطة بقاعدة المعطيات في حال كان نموذج قاعدة المعطيات مرتبط بنموذج الأغراض وهذا من المحتمل أن يزيد عدد الأخطاء (Bugs) أثناء التعديل.

- **إعادة البناء على مستوى الواجهات**

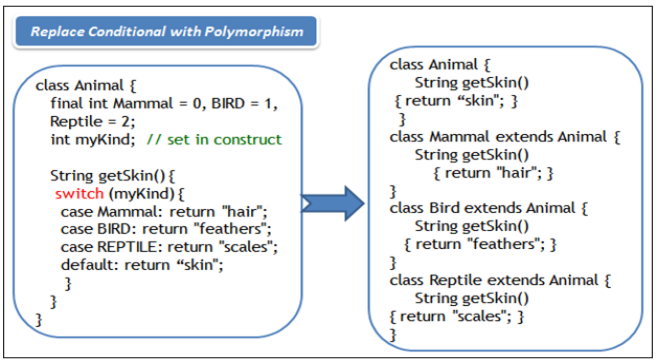
المشكلة تكمن عندما نريد تغيير واجهة عامة تستخدم رماز لا نستطيع إيجاده أو تعديله، كأن نعدل غرض ما بتغيير اسم التابع الذي يستدعيه ونحن لا نعلم في أي جزء من الرماز يتم استدعاء هذا التابع، لأنه إذا عدّلنا على اسم التابع سوف تبقى الواجهة القديمة تستدعي الواجهة الجديدة مما يتطلب إعادة تعديل الصفوف المرتبطة بهذه الواجهة كي يستطيع المستخدم التفاعل فقط مع الواجهة الجديدة.

- **إعادة البناء على مستوى الرماز**

قد تزيد عملية التصحيح نسبة مخاطر العمل، وهي مكلفة من حيث الوقت، لكنها تسعى نحو زيادة الجودة واتباع المعايير الدولية. من أهم الأمثلة عليها:

- **استبدال حالة Switch بحالة Polymorphism**

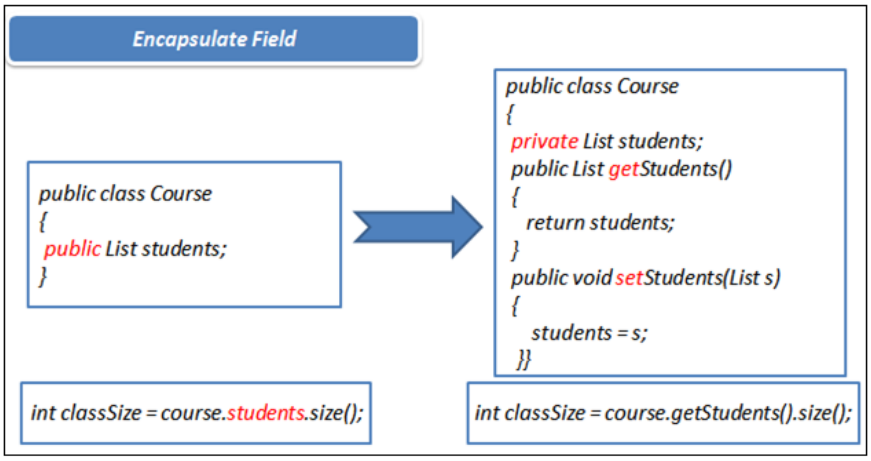
وهي الحالة المعيارية التي تضمن الجودة العالية للرماز، الشكل التالي يوضح المفهوم.



الشكل (3-1) حالة اختبارات Switch

- **ضرورة تغليف العناصر**

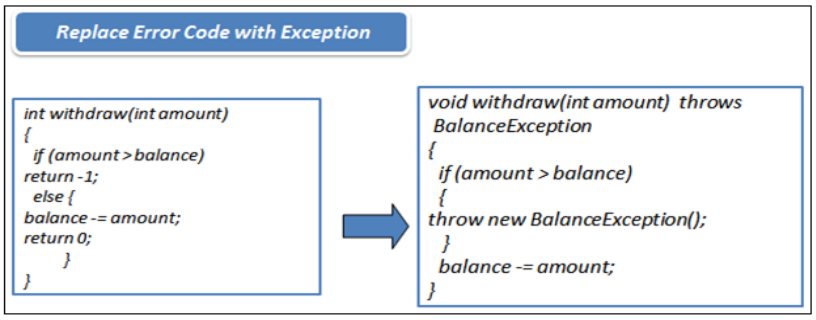
وذلك لعدم إتاحة للتعامل بشكل مباشر والهدف حمايته، الشكل التالي يوضح المفهوم.



الشكل (3-2) تغليف العناصر

- **استبدال خطأ الرماز باستثناء (Exception)**

وهي الطريقة المعيارية للعمل، وتضمن الجودة العالية الشكل التالي يوضح المفهوم.



الشكل (3-3) التصحيح المتضمن وضع استثناء

- **إعادة البناء على مستوى التصميم**

من الصعب إجراء التعديلات المطلوبة على البرنامج بعض الأحيان، كأن نغير تصميم نظام يفتقر إلى متطلبات الحماية إلى آخر جيد الحماية، عندها تكون عملية إعادة البناء صعبة كون التعديل يتطلب إعادة بناء النظام بشكل شبه كامل، ولكنها تُعتبر الأجود من حيث الإنتاجيّة، فالتصميم الصحيح يؤدي إلى جودة أعلى على كافة أصعدة النموذج والبرامج. والأمثلة عليها طويلة وسيتم الشرح عن التصحيح الخاص بها في الفصول اللاحقة.

2.3.3. أنواع اكتشاف العيوب والتصحيح المُقترح

- ✓ **التصحيح اليدوي**

إنّ المراجعات المنتظمة للرماز والتصميم والتي تكون على شكل تقارير تُعتبر معلومات هامة لفريق التطوير، تساعد على الارتقاء بالنموذج وجعله أكثر دقة وثقة. كما أن هذه المراجعات

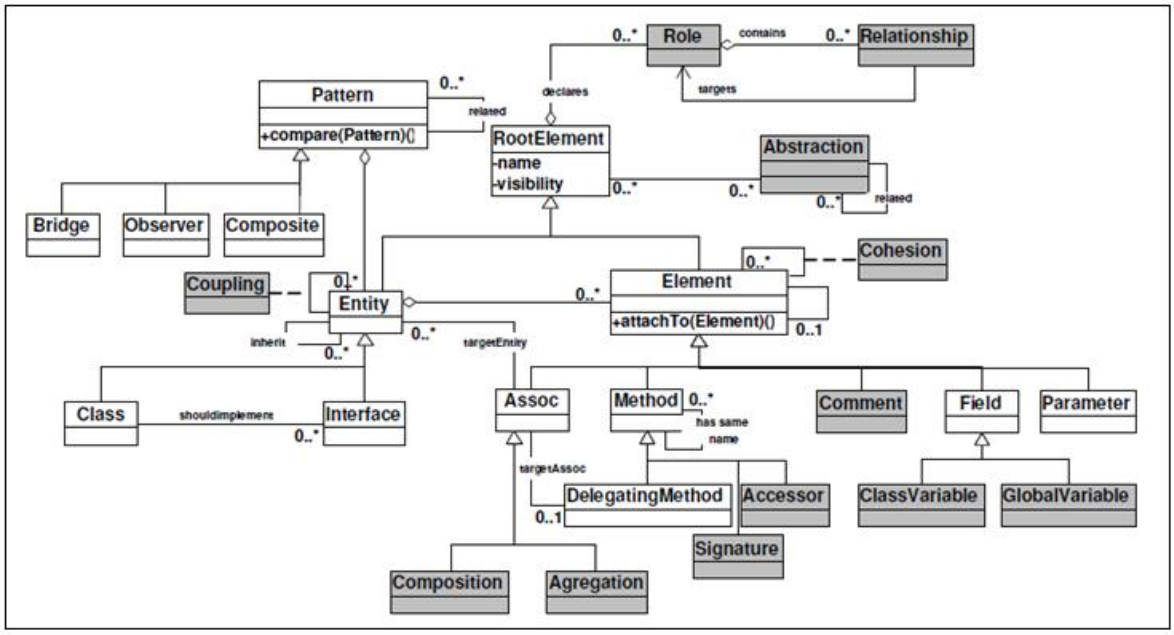
التصميمية تحتاج لفريق تطوير خبير فالتصميم السليم يساعد على توليد رماز سليم، وكثيراً ما تأخذ عملية المراجعة والتصحيح اليدوي وقتاً طويلاً وجهداً كبيراً.

- ✓ **التصحيح الآلي**

غالباً ما تتم الدراسة اليدوية لعيب من عيوب التصميم ويقترح فريق المطورون تصحيحه ولاحقاً تتم عملية صياغة هذا التصحيح على شكل خوارزمية ليتم تطبيقها على كل النماذج والحالات المشابهة.

3.3.3. نموذج كشف العيوب وتصحيحها PADL

وهو نموذج مترفع يعرف لغة لكشف العيوب التصميمية في المستويات المجرّدة، فهو نموذج يعمل على توصيف البرنامج أياً كانت العلاقات التي يحويها والصفوف التي يؤمنها والعناصر والنماذج والكيانات التي يقدمها، والشكل التالي يوضّح صورة مُفصّلة عنه [34].



الشكل(3-4) النموذج المترفع PADL

أما الكشف عن العيوب التصميمية يتم وفق خوارزمية من الخطوات المتتالية [35]:

1- **التحليل Analysis:** نستخدم المفاهيم الأساسية من التوصيفات النصية كعيوب التصميم، هذه المفاهيم تتضمن القياس المعتمد على المقاربات heuristics بالإضافة إلى المعلومات المهيكلة والدلالية. والمفاهيم الأساسية تشكل مفردات ثابتة قابلة للاستعمال ثانية لوصف عيوب التصميم بدون مرادفات أو جذور؛

2- **التصنيف Taxonomy:** نعرّف تصنيف لتوصيف عيوب التصميم وذلك بتصنيف المفاهيم الأساسية في أصناف منفصلة، هذا الصنف هو النموذج المرجعي لتمييز عيوب التصميم وتوضيح الأشياء المشتركة، والمختلفة، و ذلك لتجنب سوء الفهم؛

3- **التخصيص Specification:** نحدد عيوب التصميم كمجموعات من القواعد في بطاقات القاعدة (Rules Cards) نستخدم المفردات لتصنيف أنواع عيوب التصميم، توضح بطاقات القاعدة أدبية توصيفات عيوب التصميم بشكل آلي مع مفردات موحّدة وبشكل دقيق تتبع القواعد من نمط BNF، بطاقات القاعدة تصف بنية العلاقات بين الصفوف، الطرائق، والمتحولات ... وتميز الأدوار من خلال الأسماء والبنية وخصائص قابلية القياس؛

4- **نمذجة النماذج Meta-Modeling:** يتم تزويد النموذج PADL ببطاقات قاعدة لكشف عيوب التصميم والتي نأخذها من نماذج كشف عيوب التصميم والتي تولد آلياً تقنيات كشف العيوب، ونموذج كشف العيوب يجسد المفاهيم الأساسية التي تستخدم لتحديد العيوب كبطاقات قاعدية؛

5- **النمذجة Modeling:** نستخدم بطاقات قاعدية لبناء نموذج صحيح خالٍ من العيوب، والذي يمكن أن يعالج برمجياً، إنّ مجموعة كشف العيوب المنمذجة تشكل دليل لقاعدة تقنيات كشف العيوب، وهنا نضمن أننا قمنا بنمذجة العيوب وحصرها بشكل صحيح وهذه الخطوة هامة لتنقية المفاهيم الأساسية بشكل تكراري.

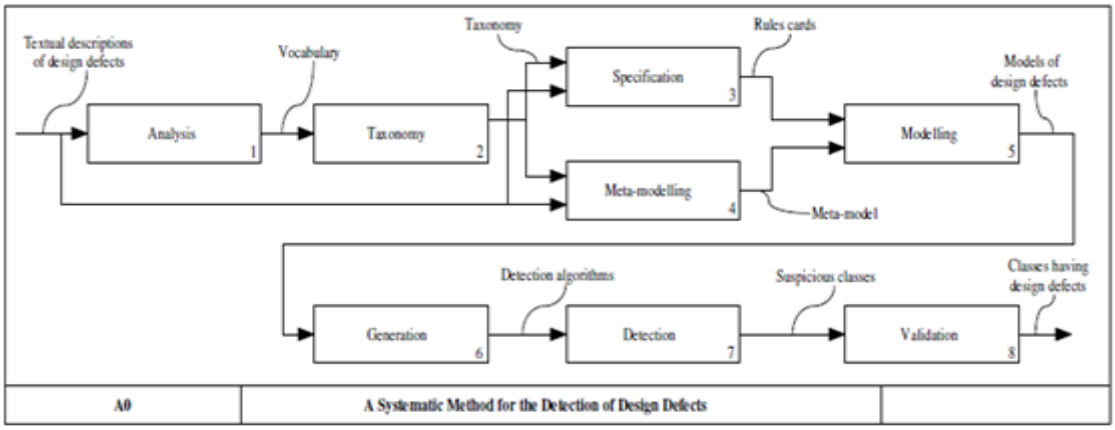
في نهاية هذه الخطوات الخمس نحصل على التعاريف الأساسية الدقيقة على شكل بطاقات قاعدية ونماذج برمجية لأي نوع من عيوب التصميم، متضمنة عيوب الرماز وعيوب التصميم من ناحية القياس والمعلومات المهيكلة.

6- **التوليد Generation:** نقوم بتوليد خوارزميات لكشف عيوب التصميم في البرامج باستخدام الـ Meta-Model هذه الخوارزميات تعتمد على قيم القياسات، خصائص المفردات، و على بنية البرنامج. إما أن تكون يدوية أو آلية؛

7- **كشف العيوب Detection:** خوارزمية كشف العيوب تعيد قوائم من الصفوف التي تحوي عيوباً؛

8- **التحقق Validation:** نتحقق من صحة خوارزميات كشف العيوب من خلال البحث عن عيوب التصميم في برامج مفتوحة المصدر وبتحليل النتائج يدوياً. هذا التحقق يتم يدوياً لأن المطورين هم الوحيدين الذين يمكنهم أن يقيّموا صلاحية التصميم الذي تم كشف عيوبه؛

الشكل التالي يوضح الخطوات.



الشكل (3-5) إجرائية الكشف عن الأخطاء

وأما عن طريقة التصحيح التي ينهجها PADL [35] فهي كالتالي:

1- التحليل لمعرفة أي التصحيح هو الأنسب لعيب النظام؛

2- تصنيف تصحيح العيوب من خلال التمييز بين التصحيحات الأولية والمركبة؛

3- نقترح قواعد للغة تعمل على تصحيح العيوب؛

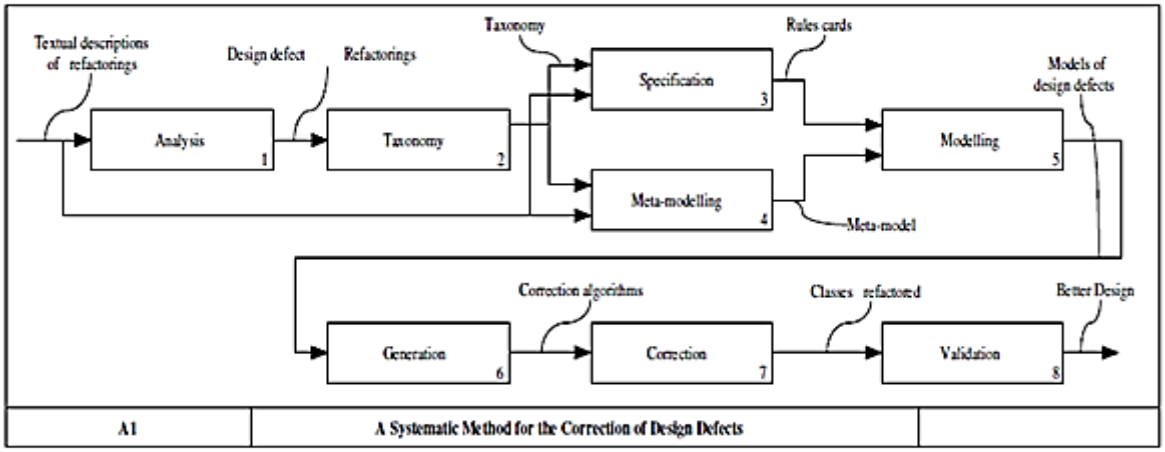
4- النمذجة المترفعة إذ نعمل على إغناء النماذج المترفعة عن طريق تهيئة البطاقات القواعدية بطريق تصحيح عيوب النموذج وتلقائياً يتم توليد تقنيات وآليات لتصحيحها؛

5- النمذجة؛

6- التصحيح، إذ تتم هذه الخطوة من خلال تطبيق خوارزميات التصحيح؛

7- المصادقة على عمليات التصحيح، إذ يمكن للمستخدم أن يوافق أو لا يوافق على العملية التصحيحية.

الشكل التالي يوضح المفهوم.



الشكل (3-6) إجرائية تصحيح الأخطاء

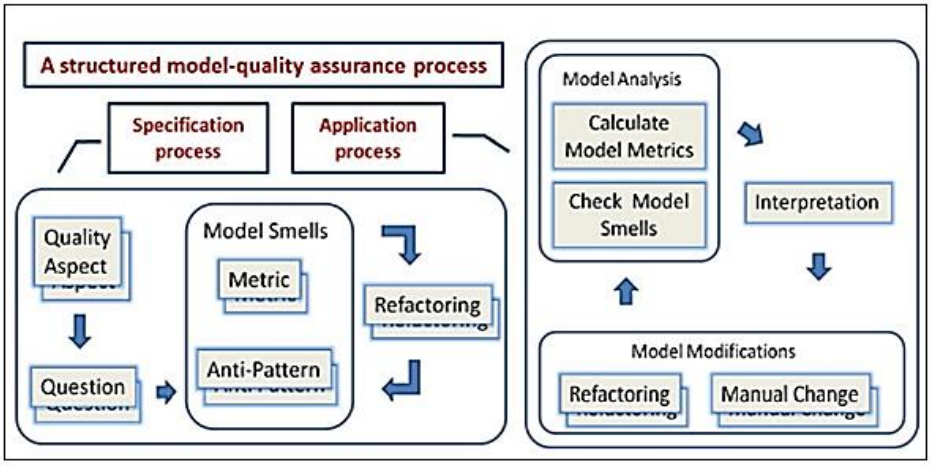
4.3.3. نموذج EMF للتصحيح الآلي

يعتمد على نوعين من الآليات لاكتشاف العيوب [36,37,38,39].

1- **المقاييس Metrics:** وهي نسب وبيانات إحصائيّة أو ديناميكيّة يقوم بها النموذج لمعرفة فيما إذا كان بحاجة لإعادة البناء والتصحيح أم لا، أما الإحصائيّة فتكون مقاييسها ثابتة يتم حسابها حتى قبل تشغيل البرنامج، أما الديناميكية فهي مجموعة من المقاييس تُحسب أثناء زمن التنفيذ (كالصلاحية وزمن الاستجابة ...)؛

2- **نماذج العيوب Antipatterns:** المعرّفة مسبقاً والتي تمّ كشفها بشكل يدوي وتمّ تحويلها إلى نماذج وخوارزميات ليتم الكشف عنها.

الشكل التالي يوضّح تسلسل العمل [40].



الشكل (3-7) آلية العمل

ولدعم هذه الأدوات نستخدم أدوات جودة مرنة من أجل النمذجة النصية لتتكامل مع الأداة EMF وكل هذه الأدوات المتاحة نستخدمها معاً بالشكل التالي:

- ✓ Xtext تزودنا بالبنية التحتية للغة؛
- ✓ EMF Refactoring تزودنا بأدوات ضمان جودة النموذج؛
- ✓ OCL لتحديد تقنيات ضمان الجودة.

5.3.3. نموذج الويب البسيط SWM

وهو نموذج خاص بنمذجة الويب البسيط [41]، تمّ تطبيق عمليات التصحيح الآلي عليه، يتألف بشكل أساسي من طبقتين:

- **طبقة البيانات Data Layer**

وهي تمثل نمذجة الكيانات التي لابدّ أن تكون موجودة في قاعدة المعطيات؛

- **طبقة الارتباطات التشعبيّة Hypertext Layer**

وهي الطبقة الحاوية على صفحات التطبيق.

تتجلى عيوب التصميم في مجموعة من النقاط هي:

- الكيانات الفارغة: وهي التي لا تحوي عناصر ولا توابع؛
- الصفحات الغير المشار لها من الصفحة الجذر؛
- الكيانات الغير المستخدمة؛
- ضياع الرابط بين الصفحات.

يتم اكتشافها من خلال مجموعة من المقاييس (Metrics) [42] كالتالي:

- Number of Entities in the Model (NEM): عدد الكيانات في النوذج؛
- Number of Dynamic Pages in the Model (NDPM): عدد الصفحات الديناميكية في النموذج؛
- Average number of Attributes in Entities of the Model (AvNAE): عدد العناصر الوسطي في الكيانات؛
- Average number of Reference in Entities of the Model (AvNRE): عدد الروابط الوسطي في الكيانات.

وهناك العديد من المقاييس الأخرى، أما عن طريقة تصحيحها وضبطها فهي تستمل لغة القيود OCL والشكل التالي يوضح الشروط التي يضعها على النموذج ويعرف النماذج السيئة ليتم التعرف عليها حال ورودها [43].

```
context WebModel
def: noDynamicPages(): Set(Entity) =
 Entity.allInstances() -> excludesAll
  (DynamicPage.allInstances() -> collect(entity))
```

الشكل (3-8) الشروط والقيود على النموذج

4.3. الخلاصة

(تصحيح العيوب واكتشافها كان لها جذور طويلة، وهي تساعد على رفع جودة المنتج البرمجي وتجاوز المشاكل التي تسببها العيوب. رأينا من الضرورة بمكان اكتشاف العيوب وتصحيحها، لكن ليس في سوية الرماز أو التصميم أو النموذج، إنما في سوية النماذج المترفعة المسؤولة عن توليد النماذج، فاعتمدنا في كشف العيوب وتصحيحيها على المنهجين اليدوي والآلي، إذ طرحنا 5 حالات دراسيّة من النماذج المترفعة على 100 طالباً في كلية الهندسة المعلوماتية ليتم اكتشاف العيوب فيها وتصحيحها، وتمّ رفدنا بالتقارير الموائمة، ثمّ تمّ صوغها على شكل منهجيّة آلية تأخذ الحسابات الرياضية والمقاييس العددية بعين الاعتبار لتكتشف العيوب تلقائياً.)

الجزء الثاني
الدراسة المرجعيّة لأهم العيوب وتأطير الحل

- ❖ مفهوم الصف الكبير
- ❖ مفهوم خطأ العنصر
- ❖ مفهوم عيوب الوراثة
- ❖ مفهوم عدم الاستخدام المباشر وتكرار تعريف المفهوم
- ❖ مفهوم العلاقة الحلقية
- ❖ مفهوم الجودة

الفصل الرابع
مفهوم الصف الكبير

1.4. مقدمة

إنّ وجود العيوب في الرماز (Code Smells) والتصميم (Design Smells) له آثار شديدة وحاسمة على جودة البرنامج، فهي تعتبر من النماذج الخاصة بالأنظمة الغرضيّة التوجه، والتي تقود إلى صعوبات في صيانة هذه الأنظمة.

وبالتالي نالت عملية اكتشاف هذه العيوب و تصحيحيها على اهتمام الكثير من الباحثين الذين قاموا باقتراح طرق للكشف عنها بهدف تحسين الصيانة، لذا كان لابدّ من إزالة هذه العيوب من خلال تصحيحها وإعادة بنائها "Refactoring".

إذ أنّ الصفوف الحاوية على العيوب "Smells" تكون أكثر عرضة للتغير والتعديل من غيرها من الصفوف.

2.4. تعريف الصف الكبير

يستخدم مصطلح الصف العظيم أو الكبير "God Class" للتعبير عن نمط محدد من الصفوف الكبيرة، والتي تقوم بأفعال ومهام كثيرة جداً، فهو يُعتبر صف يتصدر النظام، ويقوم بأغلب المهام، ويتحكم بكم كبير من البيانات المُخزنة في الصفوف المحيطة به (صفوف البيانات)، في حين أنّ باقي الصفوف لاتقوم إلّا بأعمال ثانويّة بسيطة [44].

إذ تكون هذه الصفوف مركزيّة، ويتم الزيادة عليها بشكل دوري ومتزايد، وهي تُعتبر كأمثلة عن التصميم السيء الذي لابدّ من اكتشافه وإزلته لضمان جودة البرمجيات.

3.4. سلبيات الصف الكبير

بعض الباحثين وجدوا بالصفوف الكبيرة (God Classes) حلاً أمثلياً في تنظيم الصفوف مثل Li and Shatnawi [45] ومثل Rahman [25]، كما في حالة الـ Parser Implementation إذ ستكون صفوفها بهيكليّة الصفوف الكبيرة، وتمر بالتغيرات ثمّ تستقر في باقي أجزاء النظام، غير أنّ دراسات أخرى أثبتت أنّ الصفوف الكبيرة تؤدي إلى مشاكل بسبب أنّه:

- ☒ عالي التعقيد؛
- ☒ ذو تماسك ضعيف ما بين الصفوف الداخليّة؛

- ☒ يحقق نفاذ كبير إلى بيانات الصفوف الخارجيّة Foreign Class؛
- ☒ يتجاوز مبادئ التصميم الغرضي التوجه وهي أن يكون كل صف مسؤول عن مهامه.
- ☒ كبير جداً؛
- ☒ صعب الفهم؛
- ☒ له الدور الأكبر في النظام؛
- ☒ يتغيّر أكثر من غيره بكثير وتطرأ عليه تعديلات حاسمة في حال أردنا القيام بعمليّة الصيانة، وهذا سيزيد نسبة الخطورة لظهور أخطاء وعيوب أخرى، وسيزيد الجهد عند الصيانة، وستتعرقل عملية تطوير النظام؛
- ☒ يسبب مشاكل في الصيانة والمراجعة؛
- ☒ يسبب إرهاق في عملية البناء واستدعاء الأغراض لكثرة المتحولات فيه، وبالتالي يسبب عبء إضافي وحمل زائد على النظام ككل.

4.4. الأعمال السابقة في مجال الكشف عن الصف الكبير

تُركز فرق تحليل الجودة على تعريف أجزاء من مشاكل التصميم خلال تطوير البرمجيات، ويكون تركيزها بالبداية على الأجزاء الهامة الأكثر خطورة في النظام كنواة التصميم مثلاً، ومن هنا يتصدّر الصف الكبير (God Class) الدراسات التي قام بها الباحثون.

اتجهت الدراسات السابقة إلى **دراسة** الصف الكبير من خلال منهجيتين:

- إمّا عن طريق دراسة تطور دورة حياة الصف الكبير (God Class) من خلال مراحل متعددة وأخذ أكثر من نسخة من التطبيق لدراسة التطورات التي تطرأ عليها [46]؛
- أو من خلال نسخة واحدة من التطبيق تكون نسخة ساكنة غير متغيّرة [44].

هذا وهناك منهجيتان **للتقييم**:

- فإما أن تكون نتائج خرج النموذج المقترح أحكاماً صارمة (0/1) فهي لا تحتمل الأرجحيّة وبالتالي إمّا أن يكون الصف كبير أو لا [44]؛
- أو أن تكون نتائج خرج النموذج المقترح ضمن المجال [0,1] وبالتالي فهو مجال يحتمل الأرجحيّة وتكون هناك عتبة لتحديد درجة عظمة الصف الكبير (God Class) [47].

أمّا **استراتيجيات الكشف** فهي أيضاً على نوعين:

- الكشف اليدوي:

كمراجعة الرماز والتصميم يدوياً [48,49]، وهو الأكثر وثوقيّة لكنّ سلبياته تتجلى بـ:

☒ استهلاك الوقت؛

☒ عدم القدرة على التمديد والتكرار.

وذلك وفقاً لـ Mantyla , Marinescu والدراسات المبنية في [52] ,[51] ,[50].

- الكشف التلقائي:

أي تطبيق إرشادات الكشف الآلي (مثل الاعتماد على المقاييس).

وتمت مقارنة أداء آليات الكشف التلقائي بالكشف اليدوي وكانت النتيجة بأنّ:

(آليات الكشف التلقائي هو بديل جيد ونافع عن آليات الكشف اليدوي كما في كل الدراسات الواردة [50,51,53,27,54,55,56] إذ أنّ استراتيجيات الكشف التلقائي ذات دقة مماثلة لليدوي بالإضافة إلى أنّه يمكن تمديدها بشكل أفضل.)

1.4.4.دراسات سابقة تعمل على دراسة دورة حياة الصف الكبير

- **تعريف بدراسة فريق Ptidej**

كما أنّ الأمراض تؤثر على صحة الإنسان، كذلك العيوب تؤثر على جودة البرمجيات، وبحسب المثل القائل: "الوقاية خير من العلاج"، لابدّ للوقاية من معرفة الأسباب وراء العيوب وبالتالي وضع القيود التي تضمن الجودة والسلامة وتبعدنا عن الإصابة، وهذا ما عمد إليه فريق (Ptidej Team) في جامعة (Montreal) في كندا [46]. إذ قاموا بدراسة العوامل والأسباب، وطرق الانتشار، والعيوب المكررة التي قد تؤدي إلى حدوث عيب من نمط الصف الكبير.

وتقوم **منهجيّة العمل** على مايلي:

- تعريف الصف الكبير تعريفاً واضحاً؛
- تصنيف الصف الكبير حسب درجة العظمة خلال دورة حياة النظام.

ويتم ذلك من خلال تقنيات التصنيف المعتمدة على زمن الدورة الديناميكيّة أي:

Dynamic Time Warping (DTW)، ثمّ استخدامه لمعرفة متي يصبح الصف هو GC ومتى يجب تصحيحه.

تعتمد تقنيات الكشف والتصحيح على أساليب متطورة، إذ يقوم النموذج المقترح بمعرفة فيما إذا كان الصف هو GC أم لا من خلال الأمور التالية:

1- حجم الصف (عدد التوابع والعناصر)؛
2- التحليل المعجمي لأسماء الصفوف والتوابع.

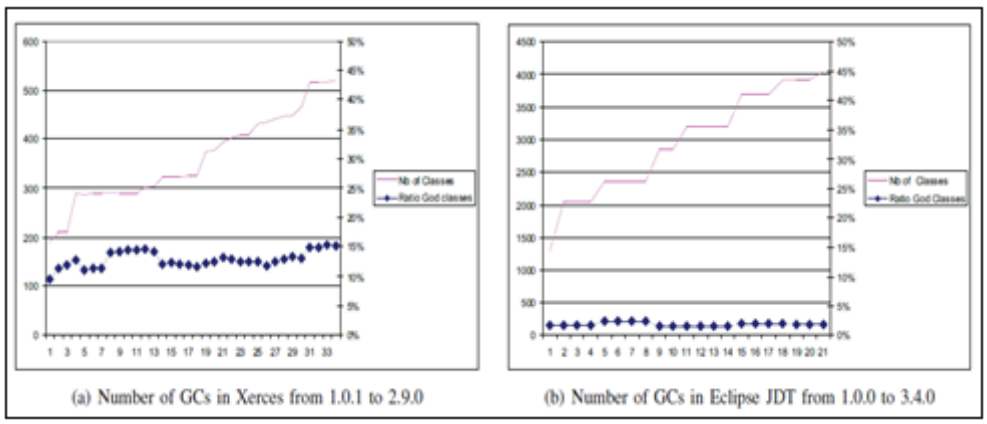
ولقد اعتمدت الدراسة على إثبات وجود الـ GC من خلال الجداول الاحتماليّة ذات البيانات الناتجة عن الاختبارات اليدويّة التي قام بها 4 طلاب اثنان متخرجان واثنان غير متخرجان، استطاعت الدراسة من خلالها جمع النتائج وصياغة البيانات في جداول احتماليّة.

خرج النموذج سيكون قيمة ما بين المجال الرياضي [1- 0] كلما اقتربت القيمة من الواحد فهذا يعني أنّ الصف أكثر احتمالاً بأن يكون GC والعكس بالعكس.

وهذا يمكننا من ترتيب الصفوف بشكل تدريجي بحسب احتماليتها بأن تكون GC.

تمّ اتخاذ العتبة هي 45% ففي حال تجاوزها ==> هو GC وكانت نسبة الدقة لهذا العمل هو 77% وهي نسبة لا بأس بها.

الشكل التالي يوضّح نسبة التعاظم للصفوف العادية خلال دورات تطور النظام فنجد أنّ النسبة تقريباً خطيّة بينهما.



الشكل (4-1) نسبة الـ GC للصفوف العادية خلال دورات تطور النظام

والجدول التالي يُترجم المخططين وقد تمّ قياس كل منهما على بيئتين مختلفتين هما Xerces و Eclipse JDT.

الجدول (4-1) ترجمة المخططين

	Xerces	Eclipse JDT
Nb of GCs (%)	138 (18%)	144 (3%)
Nb of GCs from introduction (%)	97 (70%)	88 (61%)
Nb of GCs deleted (%)	41 (30%)	27 (19%)

- **زمن الدورة الديناميكيّة Dynamic Time Warping (DTW)**

يمر الصف بعدة نسخ قبل الاستقرار وسيتم في كل نسخة تحديد احتماليّة في كون تحوله إلى GC وبالتالي يمكن تمثيله بشعاع يبيّن التغيرات التي طرأت على احتماليته.

$S = (s1; s2; s3\ldots.; sn)$ و حيث si هي الاحتماليّة، $n > i > 1$.

استعملت الدراسة تقنيات التصنيف المبنيّة على أساس DTW، وأول من عرفها هما Kruskal و Lierberman والهدف حساب الوقت اللازم للمقارنة مابين شعاعين يمثلان صفين في النظام.

هذه التقنية تعمل على إيجاد مسافة طوبولوجيّة مابين الإشارتين من خلال التعديل على محور الزمن لكل إشارة، فعلى سبيل المثال يمكن أن نجمع مابين إشارتين ذو قمتين مختلفتين في زمنين مختلفين. وهذه خاصيّة تعديل الزمن مهمّة، لأنّ التغيرات في الصفوف تحدث بشكل غير نظامي عند الانتقال من نسخة إلى أخرى خلال دورة حياة الصف وصولاً إلى الاستقرار.

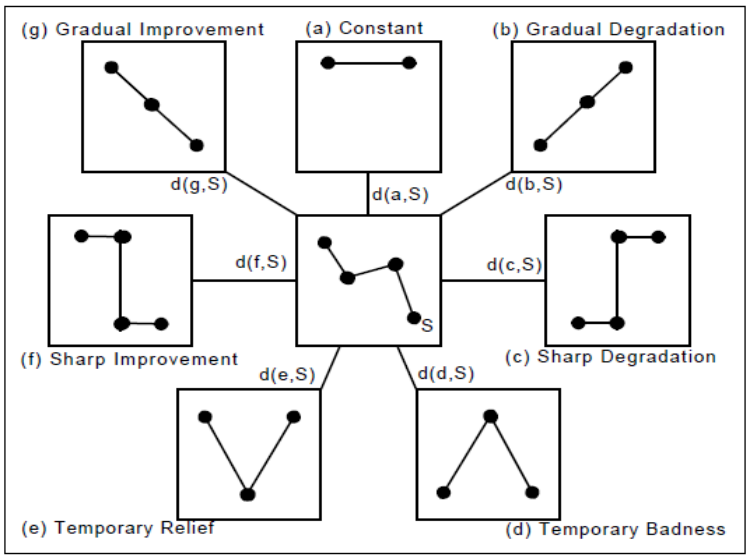
يتم تصنيف إشارة الصف بحسب قربها من النماذج الثابتة المعرفة مسبقاً.

- **تصنيف اتجاهات التطوير**

تمّ تقديم نتائج التصنيف لعمليّة تطور الصفوف من خلال بيئتي عمل هما:

Xerces و Eclipse JDT وذلك لتحديد فيما إذا كانت الصفوف GC أم لا.

الشكل التالي يبيّن تطور صف على مدى 7 مراحل (دورات) من a→g



الشكل (4-2) تطور الصف على سبعة عدة مراحل

حيث S تمثل إشارة صف الـ GC، d(x,S) هي المسافة ما بين الصف الذي لدينا والنموذج الثابت للـ GC، أما المركز فهو يمثل GC[1]

يمكن تعريف النماذج المختلفة من خلال نقطتين أو ثلاثة، بحيث تكون كل نقطة إمّا أن تملك قيمة منخفضة أو مرتفعة أو متوسطة.

فالقيم المرتفعة والمنخفضة تتطلب القيمة الأعلى والقيمة الأخفض بحسب الإمكانيّة الإحتماليّة، أما القيمة المتوسطة فيتم حسابها وفق: (High +Low)/2 .

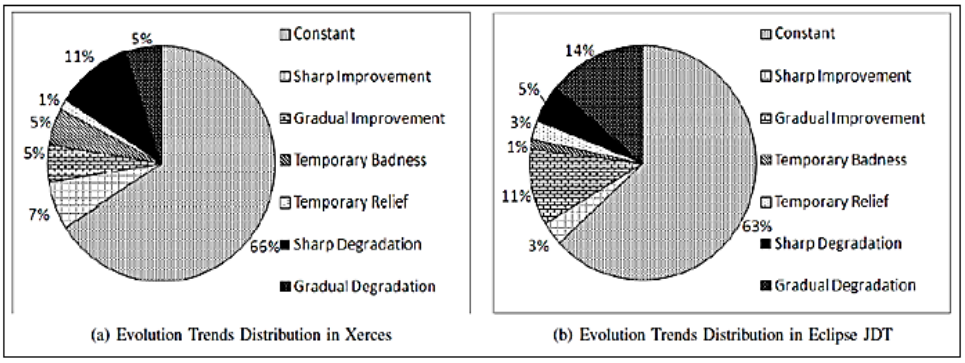
نحن بحاجة فقط 3 نقاط لأنّ DTW يمكن أن تمدد الإشارة حسب الحاجة.

لدينا عدة أنواع من النماذج والتغيرات:

- **نماذج ثابتة:** بحيث تصنف دائماً بأنّها GC، فهي تتفق مع مبدأ التغير التدريجي للصف الذي لدينا وهو أنّ GC تتطلب قيمة احتماليّة ابتدائيّة مرتفعة وتتدرج إلى قيمة متوسطة قبل أن تنحدر نحو القيمة المنخفضة.
- **نماذج التغير المفاجئ:** يكون بشكل حاد مفاجئ إذ تنحدر الإشارة من مرتفعة إلى منخفضة بشكل مباشر، إنّ عملية تخفيض التغير التدريجي وتخفيض التغير المفاجئ تتم معالجتهما بنفس الظاهرة باستثناء عملية وصف تخفيض التصميم.
- وأخيراً **نموذج حالتي الارتياح المؤقت و الاستياء المؤقت:** وهما نماذج لصفوف مؤقتة في GC.

لتصنيف الصفوف نقوم بحساب المسافات مابين النماذج السابقة وتساعدنا على ذلك خوارزميّة DTW التي توجد المسافة الأقصر مابين النماذج و GC.

يمثل الشكل التالي توزع الصفوف من خلال الانتقالات الحادة أو التدريجيّة خلال دورة حياة النظام.



الشكل (4-3) توزع الصفوف

[1] - org.apache.xerces.impl.XMLVersionDetector.

هذا يوضح وجود ثلاثة أنماط رئيسيّة من التطويرات:

1- المُحسّن؛

2- المُخفّض؛

3- الثابت GC.

تحليل كل مجموعة من هذه المجموعات يمكن أن يكشف لنا أسباب تعقيدها وكيفيّة تحسينها، سنبدأ بالمجموعة الأكبر وهي الثابتة.

❖ **النماذج الثابتة Constant**

وهي المجموعة الأكبر وتتضمن الصفوف التي تُعرّف في البداية على أنها GC وتبقى GC دون أن تتغيّر خلال دورة حياة النظام.

نركز هنا على الصفوف الكبيرة ذات الوظائف المتعددة التي يخلقها المستخدم، وبالتالي نعمل على تنبيهه، وبشكل مستقل نحاول أن نلفت اهتمام المطوّر لاستعمال النماذج الجيدة Design Patterns ليضمن الجودة. إنّ حوالي 82% من الصفوف التي يبنيها المطور تحقق أحد النماذج التالية:

1- Abstract;
2- Factory;
3- Adapter;
4- Observer;
5- Prototype.

❖ **نماذج التخفيض Degradation**

إنّ الصف الكبير المُخفّض في هذا التصنيف له سببان:

1- إمّا لأنّه لاحقاً سينمو ويزداد حجمه وتعقيده؛

2- أو لأنّه يُحدث صفوف بيانات Data Class كثيرة وجديدة.

وبالتالي الصف الذي يتعامل مع عدد كبير من صفوف البيانات بشكل مركزي يكون أكثر عرضة ليكون GC، في حالة التخفيض الحاد: يكون لدينا الزيادة الوسطيّة حوالي 150% من حجم التعليمات و حوالي 86% من حجم التوابع خلال النسخة الواحدة.

أما في حالة التخفيض التدريجي: يكون لدينا الزيادة الوسطيّة حوالي 65% من حجم التعليمات وحوالي 41% من حجم التوابع وذلك مابين نسخة لأخرى. والجدول التالي يوضّح المفاهيم.

الجدول (4-2) الزيادة الوسطية ونسب الحالات

Degradation trend	Growth rate instructions/version	Growth rate methods/version	Nb of versions
Sharp	363.64%	162.50%	1
Sharp	138.44%	283.33%	1
Sharp	214.98%	113.64%	1
Sharp	513.33%	100.00%	1
Gradual	40.99%	16.67%	2
Gradual	142.90%	6.67%	3
Gradual	9.46%	15.38%	5
Gradual	65.04%	6.25%	2
Gradual	34.28%	155.56%	3
Gradual	63.46%	42.86%	2
Gradual	103.33%	45.00%	4

إذ لابدّ أن ينتبه المطور للتعديلات الحاصلة على الصفوف والحجم الذي سيتغير وبالتالي قد تنخفض جودة الصف.

- **تصحيح الصفوف الكبيرة حسب دراسة فريق Ptidej**

تشير الدراسة إلى أنّه هناك عوامل تعمل على تصحح الـ GC، وهناك مجموعة قواعد لابدّ من اعتمادها للتصحيح، وهي كالتالي:

1- يقوم النموذج بالتحذير من التعديلات على الصفوف الهامشيّة (الصفوف التي لديها أعراض عالية لتكون GC) لأنّ إضافة أعداد كبيرة من التعليمات سيؤدي إلى انحدار الصفوف إلى GC؛

2- أي تعديل على الصفوف لا يقلل من تعقيدها (يُقاس التعقيد بمفردات الحجم) سيؤدي إلى زيادة الخطورة في جعلها GC؛

3- أي تغيرات أخرى لن تؤثر على الـ GC

الشكل (4-4) يوضّح القواعد.

أمّا من الحلول المقترحة لمعالجة الصف الكبير فهو ما يُعرف بالتصحيح أو إعادة البناء (Refactoring)، تتناوب مابين:

- إقتراح إضافة توابع إلى صفوف البيانات؛
- استخراج توابع جديدة من التوابع الكبيرة.

```
RULE_CARD : God Classes {
RULE 1: {
  ( Class status = Borderline )  AND
  ( Ratio of instructions
    added and/or deleted = high )
     AND  ( Instruction Change Ratio = high )
       => Godliness = Increase (83 %)
};
RULE 2: {
  ( Class status = Healthy ) AND
  ( Instructions Deleted = none )
      => Godliness = Increase (66 %)
};
RULE 3: {
  ( Default  => Godliness = Stable (74 %)
};
}
```

الشكل (4-4) قواعد التنبؤ بالصف الكبير

وتختلف طريقة التصحيح حسب طريقة الكشف عن الصف الكبير GC، فعند دراسة دورة حياة الصف سنكون قادرين على تحديد نموذج الصف فيما إذا كان نموذج الصف ذو تغيرات حادة أو ثابتة أو تدريجيّة وبالتالي: سيتراوح التصحيح مابين نقل توابع أو استخراجها إلى صفوف آباء او أبناء، والجدول (4-3) يوضح المفهوم.

الجدول (4-3) تصحيح الملائم للصفوف الكبيرة

Improvement	Refactoring	Nb (%)
Sharp	Move Method to data class	5 (31%)
Gradual	Move Method from GC	2 (13%)
Sharp	Extract Superclass* from GC	4 (25%)
Sharp	Extract Class from GC	1 (6%)

وفيما يلي مجموعة من الحالات التي يكون فيها الصف هو GC:

1- تواجد صفوف بيانات بشكل كبير: وهي تشكل حامل للبيانات بدون أي سلوك (Behavior)، وبالتالي أي تصحيح لهذه الأعراض سيتضمن إضافة توابع (سلوك). إذ يتوجب على المطور بدايةً أن يُغلّف العناصر وثمّ ينقل عمليات السلوك إلى صفوف البيانات، ولدينا الجدول التالي الذي يُعرّف المقاييس التالية:

- NPF : Number Of Public Field (عدد العناصر العامة المعرّفة).
- NPFr : Number Of Public Field Removed (عدد العناصر العامة الملغيّة)
- NMD : Number Of Method Declared (عدد التوابع المعرّفة)
- NMDa : Number Of Method Added To GC (عدد التوابع المضافة إلى الصف الكبير)
- NMDm : Number Of Method Moved From GC (عدد التوابع المنقولة من الصف الكبير)

- DC : Data Class (عدد صفوف البيانات)

الجدول (4-4) تصحيح صفوف البيانات

Refactoring	Data class	GC
1) Encapsulate	$\#NPF - NPF_r$	
2) Move method	$\#NMD + NMD_a$	$\#NMD_m - NMD_m$
3) Result		#DC - 1

2- تواجد عدد كبير من التوابع (السلوك) في الصف: يُستحسن هنا في هذه الحالة نقل عدد من هذه التوابع إلى صفوف البيانات.

بتطبيق المقاييس يكون:

كلما زاد عدد العناصر والتوابع المُنتقلة، كلما زاد الترابط (cohesion) .

إذا تمّ استخراج صفوف كـ (super/subclass) عندها موقع الصف في شجرة الوراثة سيُقاس بـ عدد الأبناء Number Of Children (NOC) أو بعمق شجرة الوراثة Depth of the Inheritance Tree (DIT). والجدول التالي يوضح الفكرة.

الجدول (4-5) التصحيح القائم على استخراج الصفوف

Refactoring	GC
1) New class/sub/superclass	#Assoc./$NOC + 1$/$DIT + 1$
2) Move methods out	$\#NMD - NMD_m$
3) Move attributes out	$\#NAD - NAD_m$
Results	Cohesion is better, Size is smaller

وبالتالي النموذج بشكل متكامل سيكون قادراً على تحديد حالة الـ GC وتصحيحها، مثل:

1- الصف يرتبط بعدد كبير من صفوف البيانات بعلاقات Association عندها الحل الملائم نقل التوابع إلى صفوف البيانات؛

2- إذا كان الصف كبير وغير متماسك فالحل يكمن باستخراج صفوف جديدة؛

3- إذا كان الصف يحوي على عدد كبير من التوابع والعناصر عندها يجب استخراجها.

2.4.4. دراسات سابقة تعتمد على المقاييس

وسندرس التجربة المشتركة مابين الجامعة الألمانيّة (Kaiserslautern) والجامعة النروجيّة (Trondheim) الاعتماد على مقاييس ثابتة وقيم معياريّة في عملية اكتشاف الصف الكبير.

أهم المقاييس المعتمدة:

1- تحديد تماسك الصف من خلال مقاييس Henderson-Sellers' LCOM5

ويمكن أن نعرف هذا المقياس على الشكل التالي:

ليكن الصف C يحوي على التوابع M1 , M2 ,... وكل تابع من هذه التوابع يستخدم نسخة من العناصر لتكن I1,I2, ... فيمكن أن نعرّف المجموعة P بأنها مجموعات العناصر الغير متقاطعة مابين التوابع، أما Q فنعرفها بأنها العناصر المتقاطعة مابين التوابع I1,I2,... وبالتالي I1,I2,.. يكون لدينا LCOM كالتالي:

$$LCOM = |P| - |Q|, \text{ if } |P| > |Q|$$
$$= 0 \text{ otherwise}$$

2- عدد صفوف البيانات التي ترتبط مع الصف بعلاقة Association .

هناك ثلاث قضايا هيكليّة أساسيّة في الـ GC يجب أن نعيرها جزء كبير من اهتمامنا عند العمل على مسألة المقاييس، وهي:

1- الحجم؛

2- التماسك أو الترابط؛

3- مدى الاعتماد على صفوف البيانات.

ويُمكن أن نعتمد على المقاييس في عمليّة تحديد GC مثل:

عدد التوابع :(NMD) Number Of Method

القاعدة التالية تصف حالة وجود الصفوف الكبيرة بالاعتماد على مجموعة مقاييس وعتبات [44]

$$GC(C) = \begin{cases} 1, & ((WMC(C) \geq 47) \wedge (TCC(C) < 0.\overline{3}) \wedge \\ & (ATFD(C) > 5)) \\ 0, & else \end{cases}$$

حيث

- C: Inspected Class (الصف الذي يتم فحصه)
- WMC(C): Weight Method Count

(مجموع التعقيد الحلقي لكل التوابع الموجودة في الصف)

- TCC(C): Tight Class Cohesion (العدد النسبي للروابط المباشرة)

نقول عن تابعين أنهما مترابطين بشكل مباشر إذا كان يتم استدعاءهما من قبل نفس النسخة (Instance).

- ATFD(C): Access To Foreign Data

عدد العناصر من الصفوف الأخرى التي تتصل مباشرة أو عن طريق توابع الوصول بالصف المدروس.

- **التعريف بالـ Brain Class**

يشبه الصف الكبير (God Class) من حيث المركزيّة وتعقيد المهام، وبالتالي يسبب صعوبة في الفهم والصيانة، لكن على عكسه، فهي لا تستعمل بيانات من صفوف خارجيّة، لذا فهي اكثر تماسكاً ولكن هناك احتماليّة لوجود الأخطاء والعيوب فيها بنسبة أعلى وأكبر من باقي الصفوف [44]. ولكن إذا بُنيت Brain Class بشكل متقن فسوف تستقر بدون تعديلات وبالتالي قد لا تسبب مشاكل في النظام. ولكي يكون الصف هو Brain Class لابدّ أن يتحقق القاعدة التالية:

$$BC(C)=\begin{cases}1, & \neg GC(C)\wedge((WMC(C)\geq 47)\wedge(TCC(C)<0.5))\wedge \\ & (((NBM(C)>1)\wedge(LOC(C)\geq 197))\vee \\ & ((NBM(C)\equiv 1)\wedge(LOC(C)\geq 2\cdot 197)\wedge \\ & \quad (WMC(C)\geq 2\cdot 47))) \\ 0, & else\end{cases}$$

حيث:

- NBM(C): Number Of Brain Methods (عدد التوابع).

أما لاكتشافها التوابع الـ Brain فنتبع القاعدة التالية:

$$BM(M)=\begin{cases}1, & ((LOC(M)>65)\wedge \\ & (CYCLO(M)/LOC(M)\geq 0.24)\wedge \\ & (MAXNESTING(M)\geq 5)\wedge(NOAV(M)>8)) \\ 0, & else\end{cases}$$

حيث:

- LOC (M): Line Of Code Of Method (عدد اسطر الرماز متضمناً التعليقات).
- CYCLO(M): Cyclomatic Complexity Method (التعقيد الحلقي للتابع).
- MAXNESTING(M): Maximum Nesting Level

(الحد الأعظمي من التغليف والتضمين في هيكليّة الصفوف)

- NOAV(M): Number Of Access Variable

(العدد الإجمالي لمتحولات الوصول المباشر عبر التابع)

- **نموذج EvAn التجريبي**

تمّ تطوير أداة اسمها (Evolution Analyzer EvAn) هذه الأداة تعمل على تجميع وتحليل تطبيقات الجافا [44]، الشكل (4-5) يُبيّن مثال عن توليد نموذج البيانات، إذ يمثل نموذج مترفع (Meta Model) لتطوير النظام واكتشاف الـ Smells فيه.

يُمكن تلخيص العمل وفق الشكل السابق بالخطوات التالية:

1- تجميع البيانات من مخازن البيانات؛

2- توليد الـ Model الموافق لها؛

3- استخراج الـ Meta Information من كل إصدار، مثل: (تاريخ التعديل، المؤلف، تاريخ الإصدار...)؛

4- حساب المقاييس Metrics؛

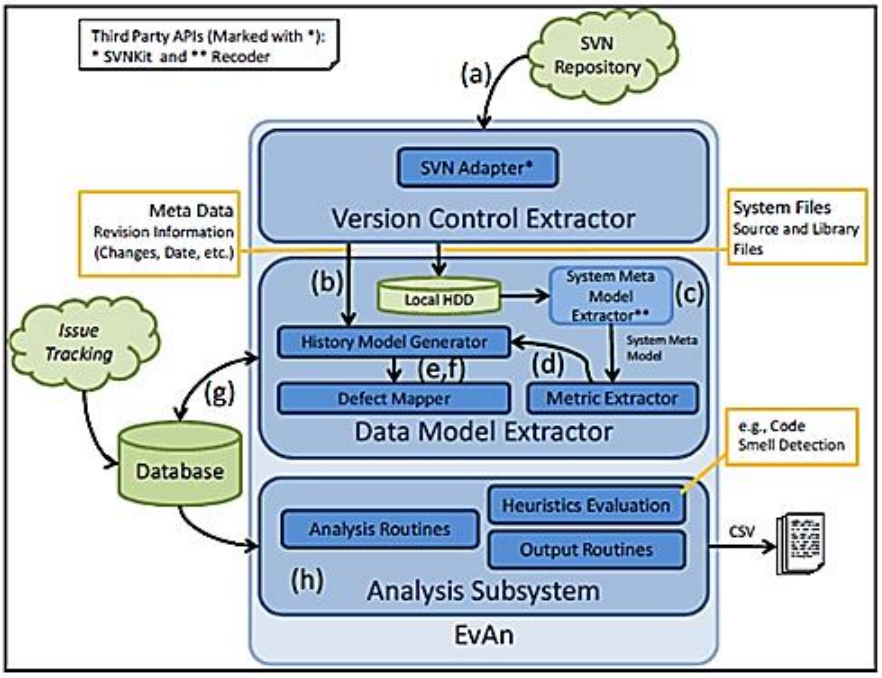
5- اغناء الـ Meta Model الخاصة بالنظام بنتائج مقاييس الرماز المستخرجة من الصفوف والطرائق؛

6- ربط التغيير مع العناصر في نظام النماذج الترفعة؛

7- مزاوجة كل عنصر في الإصدار (الصفوف- الطرائق- الملفات المصدريّة ...) مع مايقابلها في الإصدارات السابقة؛

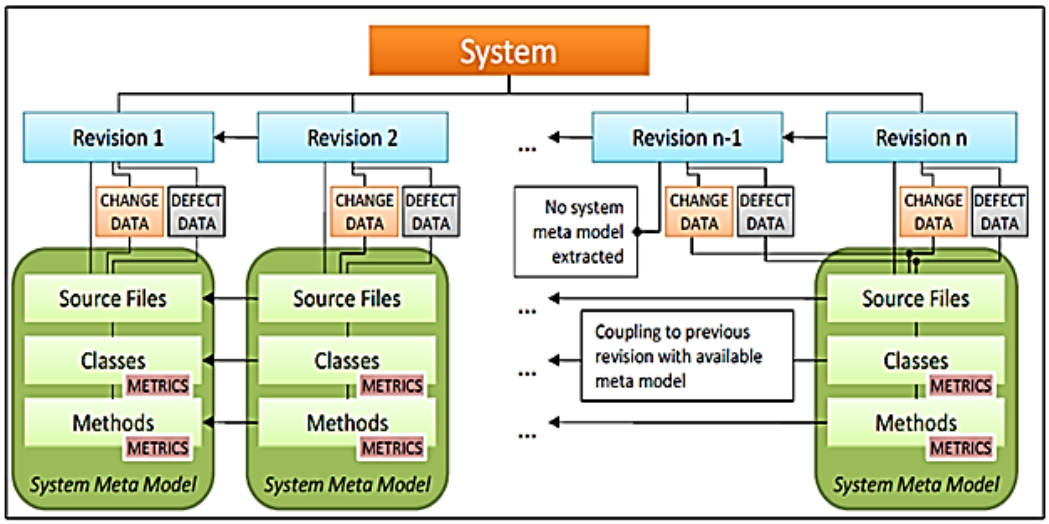
8- تخزين البيانات في قواعد البيانات المحليّة؛

9- إنجاز عمليات المعالجة والتحليل والتطوير وإظهار النتائج.



الشكل (4–5) أداة لتوليد نموذج البيانات

والشكل (4–6) يبيّن تسلسل العمليات على الصف الواحد ولكن خلال عدة دورات:



الشكل (4–6) تسلسل العمليات على الصف خلال عدة دورات

أمّا لحساب تغيرات الحجم ومعدل التغيير بين نسخة وأخرى من الصف الكبير مقارنة مع غيرها من الصفوف، يمكن تطبيق القوانين التالية [44].

$$CF(C_t)/LOC(C_t) = \frac{(NC(C_t)/PDIST(C_t)*100)}{LOC(C_t)}$$

$$CS(C_t)/LOC(C_t) = \frac{(CSIZE(C_t)/PDIST(C_t)*100)}{LOC(C_t)}$$

$$WDR(C_t)/LOC(C_t) = \frac{(SUMWD(C_t)/PDIST(C_t)*100)}{LOC(C_t)}$$

حيث:

NC(Ct) : Number Of Changes

(عدد التغيرات الحاصلة على الصف مابين الدورة والدورة التي تسبقها)

CSIZE(C) : Change Size Of code

(التغير في حجم الرماز الخاص بالصف ما الدورة والدورة التي تسبقها)

SUMWD(C) : Sum Of The Weight

(مجموع عدد العيوب الموجدة في الصف مابين الدورتين)

PDIST(C) : Distance (البعد الرقمي مابين النسخ)

LOC(C) : Number Line Of Code (عدد أسطر الرماز في الصف)

CF(C) : Change Frequency (تغير تواتر الصفوف)

CS(C) : Change Size (تغير حجم الصف)

WDR(C) : Weight Wrong (الأخطاء الموزونة)

وكل هذه القيم تضرب بالعدد 100 لتجنب الحالات الصغير جداً.

من خلال **الدراسة التجريبيّة** تبيّن أنّه:

مجرد المقارنة مابين النسخة الحالية والنسخة السابقة لها وحساب هذه المقاييس سيتبيّن معنا مباشرةً فيما إذا كان الصف هو صف كبير أو صف تشابك God Class Or Brain Class أم لا.

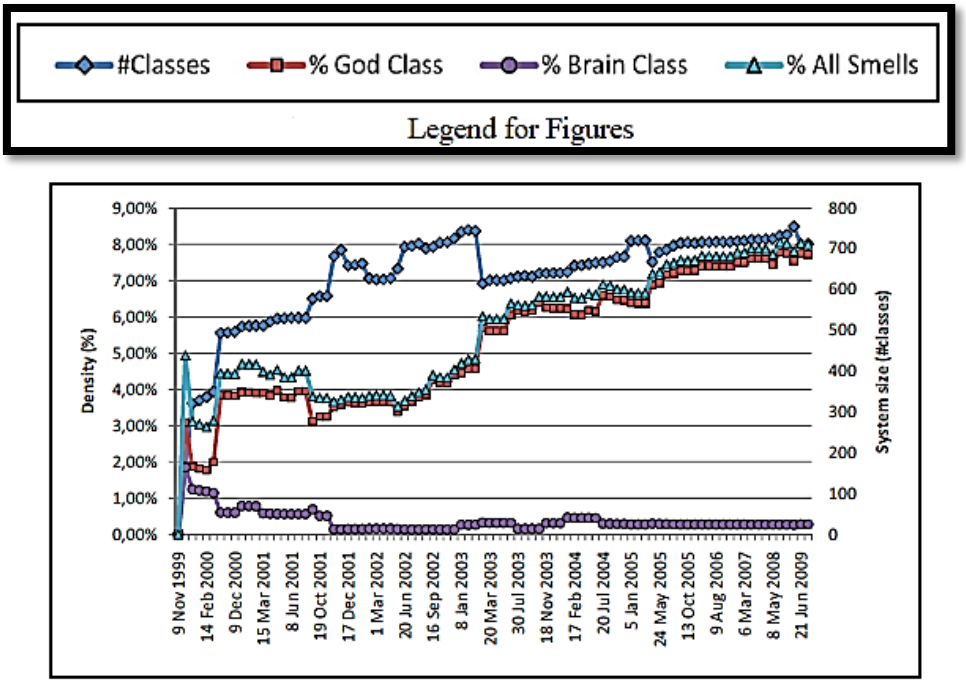
من خلال دراسة 3 أنظمة مفتوحة المصدر خلال فترة زمنيّة طويلة [44] (Xerces, Lucene, Log4j) تبيّن معنا الإحصائيات التالية:

Xerces يحوي على أكثر نسبة من الـ God Class وتساوي 8% وهذه النسبة تزداد بشكل بطيء سنة بعد سنة.

ويحتوي على 1.9%من صفوف التشابك (Brain Classes) .

ويحوي 8 صفوف كبيرة (God Classes).

الشكل التالي يوضح المفهوم، إذ أنّ محور الـ X يدل على الامتداد الزمني، ومحور الـY يدل على الكثافة بالنسبة لحجم الصف، ومحور الـ Y اليميني يدل على الكثافة بالنسبة لعدد الصفوف.

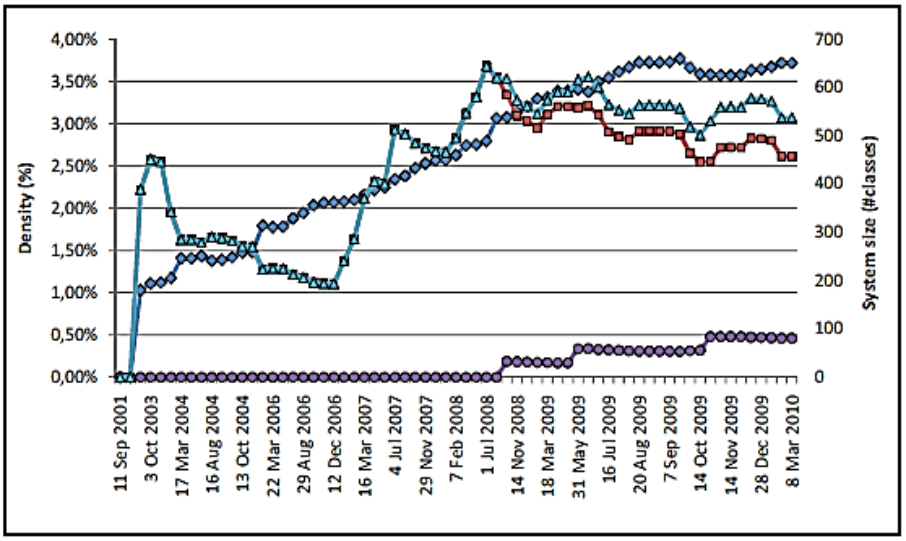


الشكل (4-7) كثافة كل من صفوف التشابك والصفوف الكبيرة في نظام Xerces

Lucene يحوي على نسبة 4% من الصفوف الكبيرة وهذه النسبة تتناقص سنة بعد سنة.

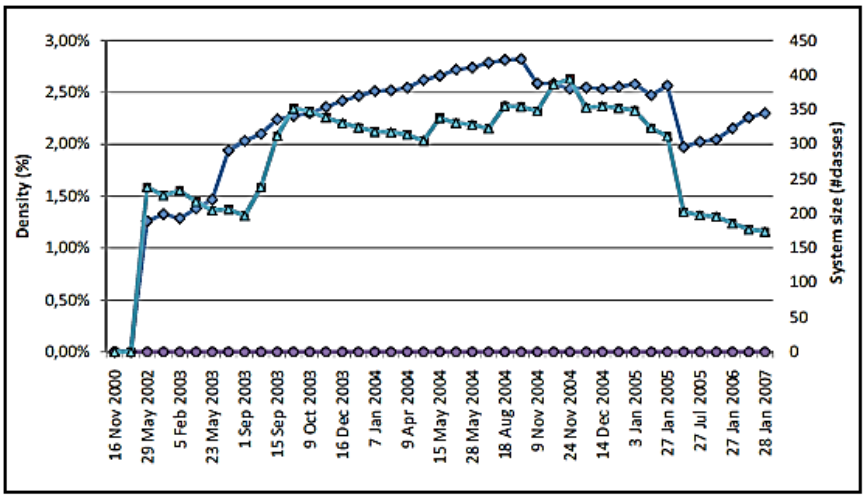
ويحتوي على صفوف تشابك بنسبة 0.5% من صفوف التشابك علماً أنّه في البداية لم يكن يحتوي على أي منها. ويحوي صف واحد مشتركة مابين الصفوف الكبيرة وصفوف التشابك.

الشكل التالي يوضح المفهوم.



الشكل (4-8) كثافة كل من صفوف التشابك والصفوف الكبيرة في نظامLucene

Log4j يحوي على نسبة 3% من الصفوف الكبيرة وهذه النسبة تتناقص سنة بعد سنة فقد وصلت مؤخراً إلى 1.2%، ولا يحتوي على صفوف تشابك. والشكل التالي يوضح المفهوم.



الشكل (4–9) كثافة كل من صفوف التشابك والصفوف الكبيرة في نظام Log4j

وبالتالي يمكن توضيح الإحصائيات والمقارنة مابين البرامج الثلاثة من خلال الجداول التالي.

الجدول (4–6) مقارنة مابين الأنظمة بالنسبة للصفوف الكبيرة وصفوف التشابك

CLASS SIZE COMPARISON GOD CLASS

		Lucene		*Xerces*		*Log4j*	
		GC	¬ GC	GC	¬ GC	GC	¬ GC
	n :	714	26214	2526	33690	256	10166
LOC	$\overline{X}$:	**881.3908**	93.6075	**768.0867**	111.2898	**677.4453**	62.7319
	s :	808.2458	144.6259	593.6356	212.7042	478.0222	82.0332

CLASS SIZE COMPARISON BRAIN CLASS

		Lucene		*Xerces*	
		BC	¬ BC	BC	¬ BC
	n :	57	26225	177	33695
LOC	$\overline{X}$:	**939.0**	93.7139	**655.4972**	111.3175
	s :	359.5675	144.9027	261.8377	212.7329

حيث: N : تمثل الحجم، X : تمثل الوسطي، S : الانحراف المعياري.

يُمكن تلخيص الاستنتاج كالتالي:

- ✓ إنّ تغير الـ Frequency أو CF يكون في الصفوف الكبيرة وصفوف التشابك أعلى بكثير ممّا هو عليه في الصفوف الأخرى؛
- ✓ إنّ تغير الـ Frequency أو CF Per LOC يكون في الصفوف الكبيرة وصفوف التشابك مختلف عمّا هو عليه في الصفوف الأخرى؛
- ✓ إنّ تغيّر الحجم CS يكون في الصفوف الكبيرة وصفوف التشابك أعلى بكثير ممّا هو عليه في الصفوف الأخرى؛
- ✓ إنّ تغيّر الحجم CS Per LOC يكون في الصفوف الكبيرة وصفوف التشابك أعلى بكثير ممّا هو عليه في الصفوف الأخرى؛
- ✓ إنّ نسبة الأخطاء الموزونة(WDR) تكون في الصفوف الكبيرة وصفوف التشابك أعلى بكثير ممّا هي عليه في الصفوف الأخرى؛
- ✓ إنّ نسبة الأخطاء الموزونة في سطر الرماز(WDR Per LOC) تكون في الصفوف الكبيرة وصفوف التشابك أعلى بكثير ممّا هي عليه في الصفوف الأخرى؛

وبالتالي يمكن اعتماد النتائج السابقة كقاعدة لتمييز الصفوف الكبيرة وصفوف التشابك من غيرها من الصفوف.

فبشكل عام نسبة تغيرات الحجم وتغير التردد (التكرار) ونسبة الأخطاء في الصفوف الكبيرة وصفوف التشابك هي أعلى ممّا هي عليه في باقي الصفوف هذا إذا لم نقم بعمليّة Normalization أما في حال القيام بها ستكون النتيجة معاكسة بسبب الحجم الكبير الذي تعاني منه كل من الصفوف الكبيرة وصفوف التشابك.

فعمدوا إلى حساب المقاييس .. CF, CS, WDR وبأخذ عتبات معينة واعتماد استراتيجيات للكشف تمّ إيضاحها سابقاُ حكمنا على الصف بأنّه من الصفوف الكبيرة وصفوف التشابك وجاءت الدراسة للبرامج المفتوحة المصدر وما صدر عنها من تقارير لتؤكد أنّها فعلاً مقاييس ناجحة.

3.4.4. دراسات سابقة حول استعمال BBN في كشف الصفوف الكبيرة

تستعمل شبكة الـ Bayesian Belief Network (BBN) بنجاح في نمذجة حالة الشك أو عدم اليقين في عدة حقول متنوعة، كالطب [57]، وإدارة المخاطر [58]، وعلم الحاسوب [59]، هذا وإنّ النجاحات التي حققتها في كل من المجالات السابقة تجعلها خياراً جيداً لكشف عيوب الرماز والتصميم.

- **التعريف بدراسة جامعة Montreal في كندا**

وهي أيضاً من تجربة فريق (Ptidej Team) في جامعة Montreal في كندا [47]، تعتمد مبدأ الشبكة BBN ويتم توصيفها كالتالي:

- دخل الشبكة: هو الصف المطلوب تحديد حالته؛
- خرج الشبكة: سيكون احتمال كون الصف جزء من عيوب التصميم؛
- يتم تعليم الشبكة وتدريبها بالاعتماد على نتائج سابقة، إذ تربط مابين الدخل والخرج ويوجد العلاقات بينها، وبالتالي يتحسّن الأداء.

فهي بيان حلقي موزّع، يمثل توزع الاحتماليّة، في هذا البيان كل متحول Xi يشير إلى عقدة i وكل قوس موجّه بين عقدتين يشير إلى احتماليّة انتقاليّة مابين الأب والابن. لذا اقترحنا هيكليّة للشبكة تجعل من كل عقدة فيها تعتمد على أباها بشكل شرطي، وبالتالي لكل عقدة جدول احتمالي شرطي يمثّل احتماليّة ارتباطها بعقد أخرى.

وبالتالي **تحليل الجودة** يتطلب أمرين خاصين من المعلومات حول بناء شبكة الـ BBN وهي:

- هيكليّة الشبكة التي تمثل العقد والأقواس السببيّة، وهنا يتأكد محللوا الجودة من صلاحية عمليات القرار؛
- جدول احتمالي شرطي يصف عمليات القرار بين العقد، وهذا الجدول قابل للتعلّم من البيانات التاريخيّة أو البيانات المدخلة مباشرة من قبل المحللين عندما تكون البيانات مفقودة.

وقد اعتمدوا في تطبيق الـ BBNS على نموذج Blob Antipatterns كما ورد في المرجع [46] إذ قدّم قواعد دقيقة للكشف عن عيوب النموذج. وقد عملوا على نقل هذه القواعد إلى منهج BBNS وطوروا على أدائها، فكانت النتائج أفضل ممّا هي عليه في القواعد العاديّة.

- **القواعد المتبعة في الكشف عن الصفوف الكبيرة**

هناك أربع أنماط من القواعد [47] تتجلى بـ:

- قواعد صفوف التحكم؛
- قواعد الصفوف الكبيرة؛
- قواعد الترابط الضعيف؛
- الارتباط مع صف أو أكثر من صفوف البيانات.

❖ أمّا **صفوف التحكم**: فنتعرف عليها بحسب إرشادات Riel وذلك إمّا من اسمها او اسم التوابع التي فيها إذ لابدّ أن تحتوي على مصطلحات مثل (Process, Control, . . .)؛

❖ أمّا **قواعد الترابط الضعيف**: فنتعرف عليها من خلال مقياس

LCOM: (Lack of Cohesion of Methods)

❖ وبالنسبة **لقواعد الصفوف الكبيرة**: فتعتمد على مقاييس عدد التوابع المعرّفة

NMD: (Number Method Declared)

وعلى مقاييس عدد العناصر المعرّفة

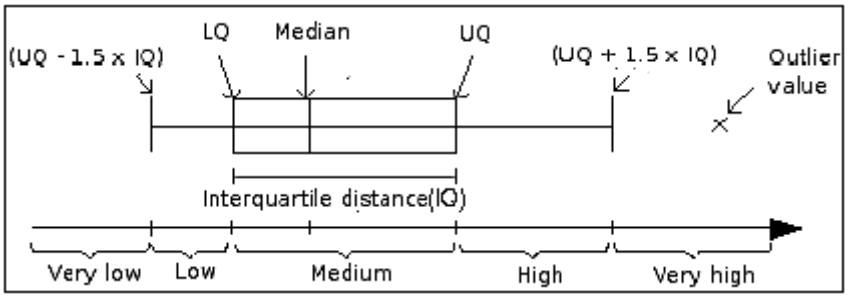
NAD: (Number Attribute Declared)

وبالتالي تعتبر الصفوف كبيرة بالنسبة لوسطي حجم الصفوف في البرنامج يتم حساب الوسطي لكل برنامج وتحديد العتبة بشكل تلقائي.

❖ **حالة الارتباط بصفوف البيانات**: تُعرّف صفوف البيانات بأنها الصفوف التي تحوي حوالي 90% من توابعها فقط عمليات set/get for its attribute.

وبالتالي المقاييس تقسم إلى خمس سويات من الاحتمالات:

(منخفض جداً، منخفض، متوسط، مرتفع، مرتفع جداً) والشكل التالي يوضح العتبات.

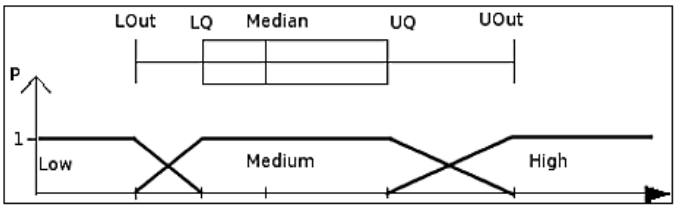


الشكل (4-10) توضيح العتبات

➢ **البناء الهيكلي لشبكة BBN**

عمد المشروع إلى تقسيم المقاييس إلى عدة سويات وهنا عمدنا إلى جعلها 3 سويات (منخفض ، متوسط ، مرتفع) مستثنين حالتين هما منخفض جداً و مرتفع جداً كون هذه الحالات نعتبرها حالات يقينية لا تحتمل الارتياب أو الشك، وسنستعمل التقريب الخطي لوضع النتائج.

والشكل التالي يوضح المفهوم



الشكل (4-11) توضيح العتبات

أما عن مسألة الهيكليّة التي تحدد فيما إذا كان الصف مرتبط مع صفوف بيانات أخرى، فتحسب من خلال عدد الارتباطات مابين الصف وصفوف البيانات فكلما ارتبط بعدد أكبر من الصفوف كلما زادت احتماليته ليكون صفاً كبيراً.

أما بالنسبة لتحديد فيما إذا كان الصف هو صف بيانات أم لا فيعود ذلك إلى عدد عمليات Set/Get في الصف فإذا تجاوزت 90% من العمليات عندها سيكون صفاً للبيانات، ويمكن جعلها على شكل مقياس (NoDC). حيث تتراوح القيم مابين (1--> n) حيث n تمثل القيمة العليا من حدود NoDC.

اعتمد المشروع في بناء الهيكليّة لشبكة BBN على عاملين أساسيين [47]:

- ينقل عناصر الدخل الموجودة في القواعد Rule Cards إلى شبكة الـ BBN كعقد دخل لها Input Node مع التوزيعات الاحتماليّة.
- ينقل العمليات Operation الموجودة في القواعد Rule Cards إلى شبكة BBN كعقد قرار Decision Nodes مع جدول للاحتمال الشرطي.

الشكل التالي يوضّح القواعد.

```
1  RULE_CARD : Blob {
2     RULE : Blob { ASSOC: associated FROM: mainClass ONE TO: DataClass MANY };
3     RULE : MainClass { UNION LargeClassLowCohesion ControllerClass };
4     RULE : LargeClassLowCohesion { UNION LargeClass LowCohesion };
5     RULE : LargeClass { (METRIC: NMD + NAD, VERY_HIGH, 0) };
6     RULE : LowCohesion { (METRIC: LCOM5, VERY_HIGH, 20) };
7     RULE : ControllerClass { UNION
8               (SEMANTIC: METHODNAME, {Process, Control, Ctrl, Command, Cmd,
9                                       Proc, UI, Manage, Drive})
10              (SEMANTIC: CLASSNAME,  {Process, Control, Ctrl, Command, Cmd,
11                                      Proc, UI, Manage, Drive, System, Subsystem}) };
12    RULE : DataClass { (STRUCT: METHOD_ACCESSOR, 90) };
13 };
```

الشكل (4-12) القواعد الخاصة بكشف الصفوف الكبيرة

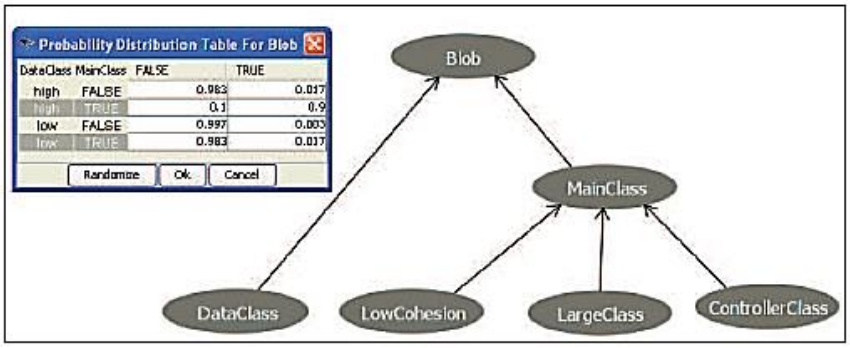
<u>أمّا عن العمليات:</u>

هناك عمليتان متاحتان لتجميع المعلومات من القواعد وهما: الاجتماع والتقاطع.

يتم تعريف الصف الأساسي (Main Class) بأنه عملية اجتماع لكل من الخصائص (الحجم الكبير، الترابط المنخفض، أسماء محددة) وأما الصف الكبير يُعرّف بأنها تقاطع الصفوف الأساسيّة مع صفوف البيانات.

وأمّا عقد الخرج:

من خلال تطبيق قواعد Rule Card على شبكة BBN ينتج العقد التالية الموضحة بالشكل التالي.



الشكل (4-13) العقد المشكلة للصف الكبير

حيث تمّ تجميع الصفوف الأساسية مع صفوف البيانات ليعطي الصف الكبير (Blob Class / God Class). وعند توافر بيانات وتصانيف سابقة يمكن لنا تدريب شبكة BBN، وبالتالي يؤدي إلى تخفيف نسبة الأخطاء في النتائج الناتجة عن التشويش في القيم.

- **النتائج التجريبيّة**

الهدف من التجريب هو إثبات طريقة Bayesian في كشف عيوب التصميم وإعطاء مجموعة مرتبة حسب الاحتماليّة المتناقصة لكون الصف فيه عيب من نوع (Blob Class)، وبالتالي زيادة جودة البرامج .

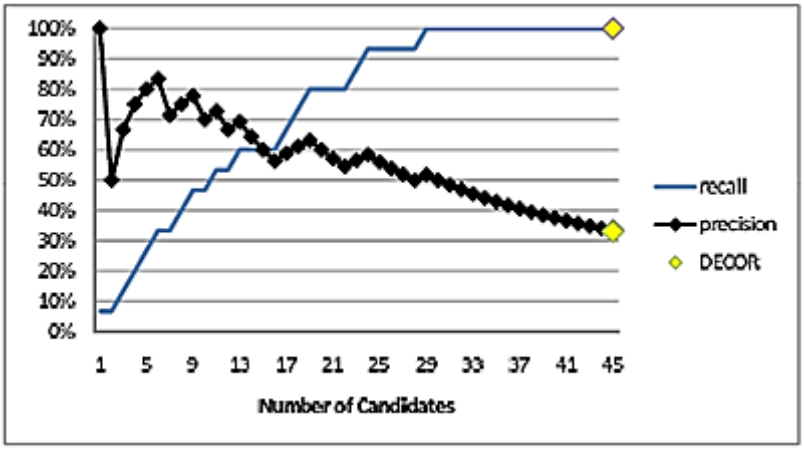
- ✓ فالنموذج يستفيد من البيانات التاريخيّة في تدريب الشبكة.
- ✓ هو أكثر مرونة من DECOR ذي النتيجتين المحددتين (Smell / Not Smell) إذ يعطي نسبة احتماليّة وهي ذات إنجاز أعلى من DECOR بشكل عام وهذا ماتظهره فيما بعد حسابات الدقة والاستدعاء. الجدول التالي يوضح النتائج بعد العمل على برنامجين مفتوحي المصدر.

الجدول (4-7) نتائج العمل على البرنامجين

Systems	♯ classes	KLOC	♯ Blobs
GanttProjectv1.10.2	188	31	4
Xerces v2.7.0	589	240	15
Total	777	271	19

وبالتالي يمكن تعريف المقياسين التاليين:

- ✓ الدقة Precision (عدد الصفوف التي تحقق Blob Class من بين الصفوف المرشحة / عدد الصفوف المرشحة).
- ✓ الاستدعاء Recall (عدد الصفوف التي تحقق Blob Class من بين الصفوف المرشحة / عدد الصفوف التي تحقق Blob Class). والشكل التالي يبيّن القياسين المذكورين سابقاً.



الشكل (4-14) رسم بياني يوضح مقياسي الدقة والاستدعاء

5.4. الخلاصة

(رغم تعدد آليات ووسائل اكتشاف الصفوف الكبيرة ومنهجيات التصحيح إلّا أنها تدور حول محور واحد، وهو الحجم الكبير وضعف الترابط، وهذا سبب في عدم تماسك النموذج، وضياع المستخدم بين المفاهيم، وعدم القدرة على الإحاطة بكامل العمليات بالشكل السليم، ناهيك عن صعوبات الفهم والصيانة والتعديل، لذا فعملية اكتشاف هذه العيوب (عيوب الحجم وضعف الترابط) والعمل على إزالتها سيرتقي بالنموذج نحو جودة أعلى.)

الفصل الخامس
مفهوم خطأ العنصر

1.5. مقدمة

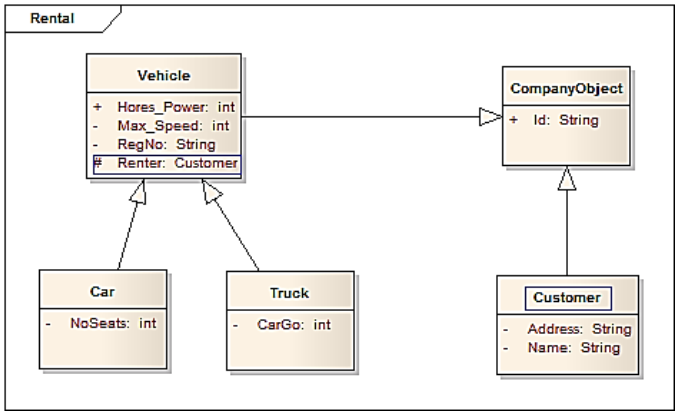
عيوب التصميم تطعن بالجودة بشكل كبير، لذا كان لابدّ من تلافيها، ويُعتبر مفهوم خطأ العنصر في صدارة هذه الأخطاء، ويمكن تعريف خطأ العنصر بأنه:

وجود عنصر من نمط صف كعنصر داخل صف آخر، وعليه يُفترض استبدال هذا العنصر مباشرةً بعلاقة ارتباط Association.

2.5. الدراسة المرجعية

1.2.5. شرح حالة الخطأ

إنّ حالة مفهوم خطأ العنصر تطعن بالجودة إذ تنفي صفة "الانسجام والتوافق" أو كما يُعرف Conformity إذ يشترط هذا البند أن يكون نمط كل عناصر الصفوف من الأنماط البسيطة غير المعقدة في التصميم ليسهل اندماج المستخدم معها ويسهل التعامل معها، وفي حال اقتضت الحاجة إلى وجود عنصر داخل صف من نمط صف آخر لابدّ من استبداله فوراً بعلاقة ارتباط تعبّر عن العنصر [40]. وفيما يلي نستعرض مخطط الصفوف الخاص بشركة تأجير السيارات والذي احتوى على عيب من نوع مفهوم خطأ العنصر. الشكل التالي يبيّن المفهوم.



الشكل (5-1) مخطط الصفوف الخاص بشركة تأجير السيارات

يحتوي المخطط على عدة صفوف مبينة في الجدول التالي.

الجدول (5-1) الصفوف الموجودة في مخطط شركة تأجير السيارات

اسم الصف	نوعه	العناصر	أنماط العناصر
CompanyObject	Abstract Class	Id	String
Customer	Concrete Class	name	String
		adress	String
Vehicle	Abstract Class	horsepower	Integer
		regNo	String
		maxSpeed	Integer
		renter	Customer
Car	Concrete Class	noSeats	Integer
Truck	Concrete Class	cargo	Integer

وكما هو واضح يكمن العيب في وجود عنصر من نمط Customer داخل الصف Vehicle.

2.2.5. شرح التصحيح المُقترح

كما أوردنا سابقاً لابدّ من إزالة هذا العيب لنضمن عنصر الجودة المعروف بـ "التوافق أو الانسجام" Conformity وإزالته تكمن بحذف العنصر ذي النمط المعقد وجعل الصف يتصل بالصف الآخر بعلاقة ارتباط Association مما يسهّل عملية الفهم ليحافظ على الجودة بأعلى مستوياتها.

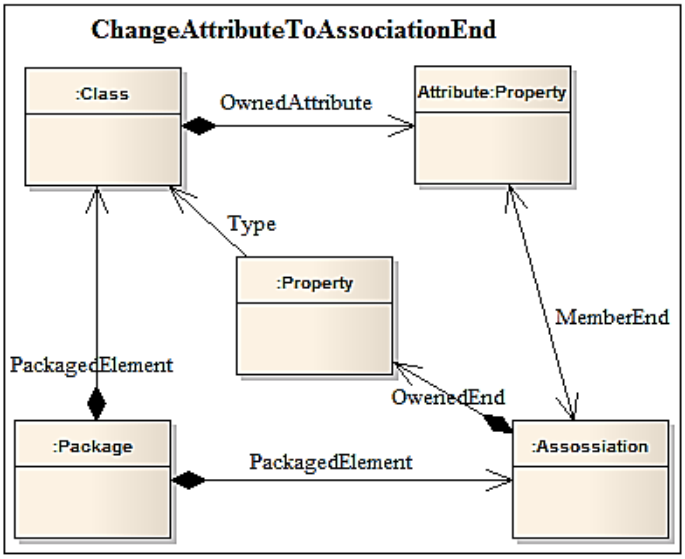
يقترح تصميم Henshin عدة قواعد لتعالج عيوب التصميم [40] وفيما يلي نورد القاعدة التي تمنع وجود عنصر في صف من نمط صف آخر، الشكل التالي يبيّن القاعدة.

الشكل (5-2) قاعدة Henshin لمنع وجود عيب من نوع Attribute Concept

وتشكل هذه القاعدة كشرط مسبق يمنع تعريف عنصر في صف من نمط صف آخر، وذلك لضمان جودة التصميم، أما التصحيح المُقترح فهو يكمن على الشكل التالي:

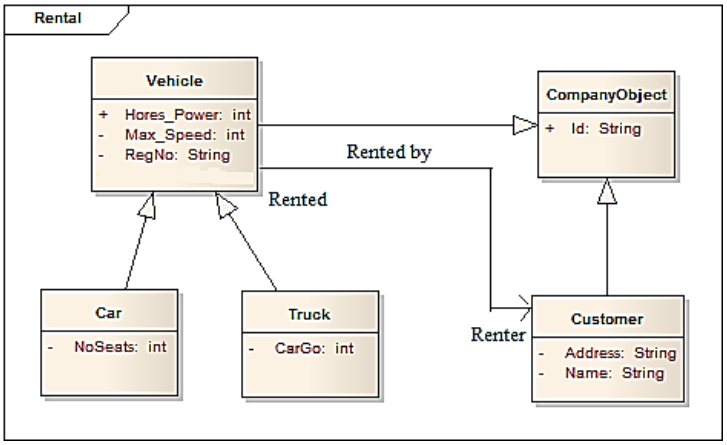
- ✓ حذف العنصر ذي النمط المُعقد؛
- ✓ استبداله بعلاقة ارتباط مابين الصفين.

الشكل التالي يبيّن قاعدة التصحيح حسب Henshin



الشكل (5-3) قاعدة التصحيح حسب Henshin

وأما مخطط الصفوف السابق الخاص بشركة تأجير المركبات بعد أن يخضع للتصحيح المُقترح من قبل Henshin يغدو كما في الشكل التالي:



الشكل (5-4) مخطط الصفوف بعد إحداث التصحيح المُقترح حسب Henshin

3.5. الخلاصة

(من الواضح أنّ مفهوم خطأ العنصر عند تواجده في التصميم يسعى إلى زيادة الحجم والتعقيد، لكن عند إزالته واستبداله بعلاقة التجميع نكون قد أنقصنا الحجم وزدنا من الترابط.)

الفصل السادس
مفهوم عيوب الوراثة

1.6. مقدمة

تعتبر الوراثة من أساسيات التصميم البرمجي لما لها من ميزات كبيرة، تعتمد على تقسيم التصميم إلى كتل أو مفاهيم متعددة، وكل كتلة تكون مستقلة بحد ذاتها وتربطها علاقة الوراثة بباقي الكتل، وبالتالي تزداد جودة التصميم، من خلال التركيز على كل كتلة على حدى.

2.6. ميزات علاقة الوراثة في التصميم

- جمع البيانات والإجراءات في كائن واحد وبالتالي نتعامل معهما كوحدة واحدة؛
- إخفاء المتغيرات التي لا نرغب في أن يصل إليها المستخدم وإتاحة مداخل محددة لتغيير وقراءة البيانات والمعلومات؛
- زيادة الأمن وضبط عمليات الوصول وتوحيد طريقة المعالجة وإعادة الاستخدام وبالتالي التوفير بالوقت والجهد على المصممين؛
- زيادة الترابط ما بين المفاهيم؛
- الابتعاد عن الحجوم الكبيرة للمفاهيم من خلال وجود صفوف أبناء وآباء تتقاسم الحمل فيما بينها.

3.6. الدراسات السابقة

1.3.6. الوراثة على مستوى النماذج المتسايرة

الوراثة مهمة من أجل إعادة استعمال الرماز وتحديد الهرمية في النظم المصمّمة. إنّ دمج مفهوم التساير والتزامن مع الوراثة يؤدي إل مشاكل أهمها ما يعرف بـ"شذوذ الوراثة" (Inheritance Anomaly) إذ تظهر هذه المشكلة عند وجود تنافر أو تعارض ما بين الرماز الموروث وقيود التزامن في الأغراض المتسايرة [60]، وإنّ أغلب النظم الحالية تحتاج إلى التساير من أجل أداء أفضل أو لنمذجة نظم متزامنة بطبيعتها، وعند استعمال الوراثة سنكون أمام مشكلة عدم التوافق والشذوذ الوراثي [62],[61].

تمّ تعريف ثلاث أنماط للشذوذ الوراثي سابقاً وتسمى في مجملها حالات التحويل، وهي:

- حالة التقسيم والتجزيء؛
- حالة الحساسيّة للتنفيذ السابق؛
- حالة التعديل والتحرير.

تحدث هذه الحالات عند إضافة تابع جديد أو إضافة صف جديد في الوراثة.

يكمن الحل باستخدام شبكات بتري التي تحل مشاكلة الوراثة عن طريق الشروط الأولية والأفعال اللاحقة [61] وتدعم أيضاً النمو المتزايد للنموذج، فهي تؤمن إطار عمل موحّد للنمذجة من أجل التصميم الغرضي التوجه واختبار الصلاحيّة.

إذ أن مشكلة الوراثة تكمن في عدم التزامن مابين التابع الموروث وشروط التزامن في الأغراض المتسايرة، وبالتالي يكمن الحل [62] بأحد الخيارين التاليين:

- إعادة تعريف التوابع الموروثة مع ما يتوافق مع شروط التزامن؛
- التعديل على شروط التزامن أو إعادة تعريفها.

- **<u>حالة التقسيم والتجزيء</u>**

وهذه الحالة تواجهنا عند إضافة تابع جديد في الابن أو التعديل على التابع الموروث وفي هذه الحالة سنكون بحاجة إلى تمييز حالات إضافية لم يميزها التابع الأب.

مثال: لدينا الصف Buffer فيه التابعين:

()Set : لوضع عنصر في الـ Buffer، ()Get: لإخراج عنصر من الـ Buffer، وهنا سيكون لدينا حالات لابدّ من تمييزها،

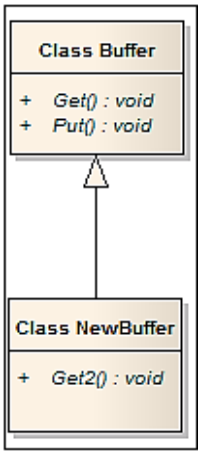
1. حالة الـ Buffer فارغ؛
2. حالة الـ Buffer ممتلئ؛
3. حالة الـ Buffer فيه عنصر واحد.

ولدينا الصف الابن NewBuffer فيه التابع ()Get2 وهي عملية إخراج عنصرين من الـ Buffer وهنا لابدّ من تمييز الحالات التالية:

1. حالة الـ Buffer فارغ؛

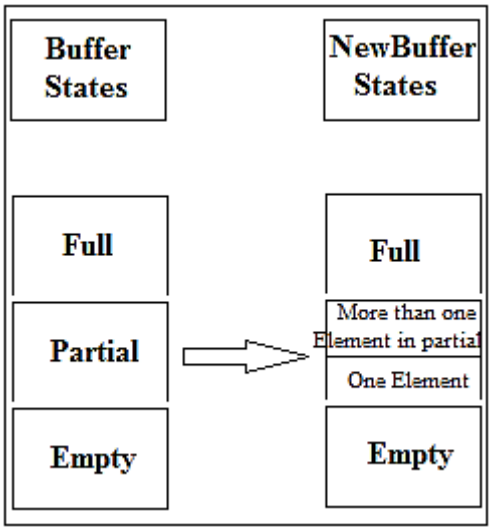
2. حالة الـ Buffer ممتلئ؛
3. حالة الـ Buffer يحوي على أكثر من عنصر؛
4. حالة الـ Buffer فيه عنصر واحد.

والشكل التالي يبيّن الوراثة [60].



الشكل (6-1) هرمية الوراثة

والشكل التالي يبيّن الحالات المميزة في كلا الصفين [60].



الشكل (6-2) الحالات المميّزة في كلا الصفين

وبالتالي إضافة تابع أدى إلى ضرورة تمييز حالة جديدة في التقسيم وهذه حالة من حالات شذوذ الوراثة.

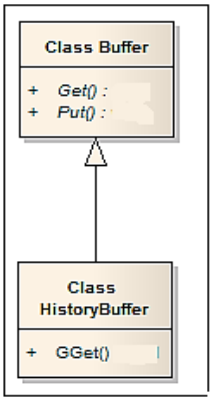
- **حالة الحساسيّة للتنفيذ السابق**

حالات التزامن تحتوي أحياناً على توابع تعتمد على تاريخ الاستدعاء، وتميّز مابين الحالات التاريخيّة المختلفة للتنفيذ [63]، وهذا يتطلب تعريف متحولات إضافية تستطيع تحديث قيمها حسب حالات التنفيذ المختلفة، وبالتالي التعريف بهذه المتحولات قد يسبب إعادة تعريف الرماز لتمثيل الشروط التاريخيّة وهنا سنكون أمام حالة الحساسيّة للتنفيذ السابق.

مثال: بالعودة إلى الصف Buffer السابق وتعريف صف ابن جديد له، وليكن الصف HistoryBuffer وفيه التابع GGet() الذي لا ينفذ إلّا إذا تمّ تنفيذ التابع Get() في الصف الأب Buffer لأنّ التابع GGet2() يعتمد على البيانات التاريخية لتنفيذ التابع Get(). وبالتالي سنكون أمام حالات جديدة لابدّ من تمييزها عند تنفيذ الصف الابن، إذ لابدّ من تمييز حالتين:

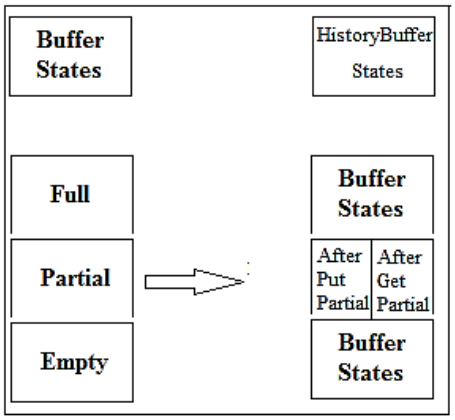
- قبل استدعاء التابع Get() في الأب؛
- بعد استدعاء التابع Get() في الأب.

الشكل التالي يبيّن هرميّة الوراثة بين الصفين [60].



الشكل (6-3) هرمية الوراثة

الشكل التالي يبيّن الحالات المميّزة لكلا الصفين [60].

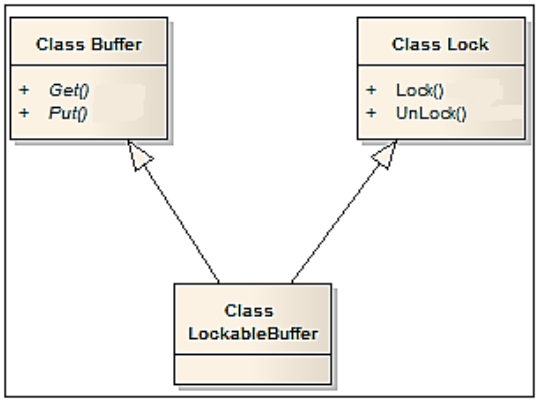


الشكل (6-4) الحالات المميّزة لكلا الصفين

- **<u>حالة التعديل</u>**

تحدث هذه الحالة نتيجة الوراثة المتعددة، إذ يرث الصف الابن من أكثر من صف أب وبالتالي سنكون بحاجة إلى التمييز مابين التوابع الموروثة وستتعدل الحالات.

مثال: ليكن لدينا الصف Lock وهو الصف الخاص بالقفل يحوي على تابعين ()Lock وهو التابع الذي يقفل أي يمنع جميع أنواع العمليات على العنصر، والتابع ()UnLock وهو التابع الذي يفك القفل عن العنصر. وليكن لدينا الصف LockableBuffer وهو سيرث من الصف Buffer ومن الصف Lock الشكل التالي يوضح هرميّة الوراثة [60].



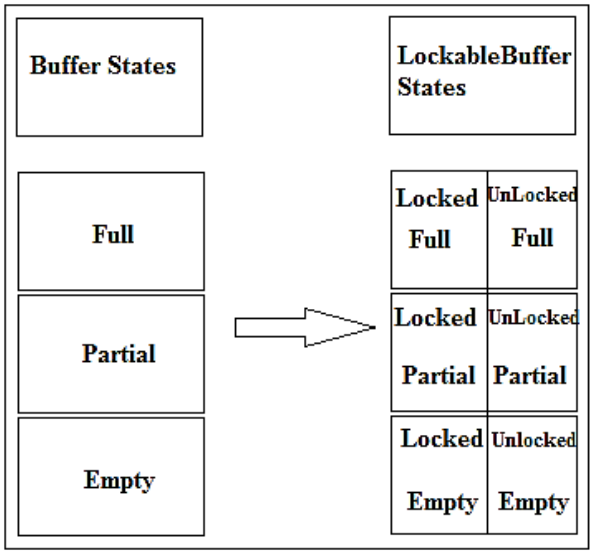
الشكل (6-5) هرمية الوراثة

وهنا ستتعدل التوابع الموروثة من الصف Buffer وسيكون لدينا بدلاً من 3 حالات، 6 حالات هذه الحالات كالتالي:

1. حالة الـ Buffer ممتلئ ومقفل؛
2. حالة الـ Buffer ممتلئ و غير مقفل؛

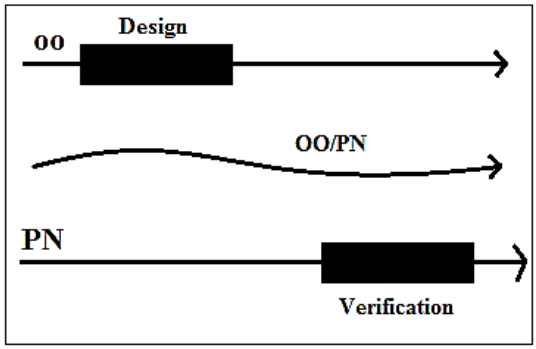
3. حالة الـ Buffer ممتلئ جزئيّاً ومقفل؛
4. حالة الـ Buffer ممتلئ جزئيّاً وغير مقفل؛
5. حالة الـ Buffer فارغ ومقفل؛
6. حالة الـ Buffer فارغ وغير مقفل.

الشكل التالي يبيّن الحالات المميّزة في كلا التابعين [60].



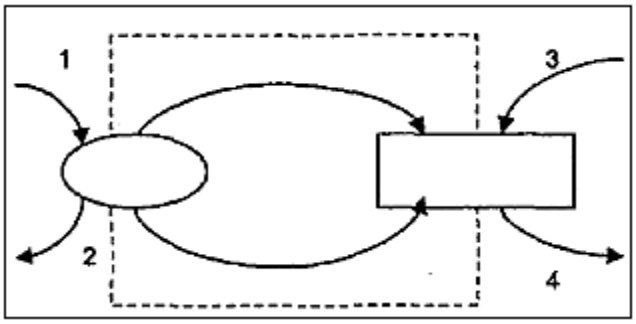
الشكل (6-6)الحالات المميّزة في كلا الصفين

يأتي الحل متمثلاً بشبكات بتري متكاملاً مع التصميم الغرضي التوجه، إذ أنّ التصميم الغرضي التوجه يعرّف هرميّة الصفوف والأغراض التعاونية، أما شبكات بتري فتعرّف تفاعل هذه الصفوف والأغراض وستسمح بالتحقق والتأكد من الصلاحية [64]. الشكل التالي يبيّن التكامل مابين المفهومين [60].



الشكل (6-7) التكامل مابين الغرضيّة التوجّه وشبكات بتري

وتمّ تمثيل شبكات بتري من خلال العقد المجردة، فالعقدة المجردة هي نموذج موحّد لشبكات بتري يمكن أن تترجم هذه العقدة إلى ثنائية (عقدة، انتقال). الشكل التالي يمثل العقدة المجرّدة [60].



الشكل (6-8) العقدة المجرّدة

إذ تُبنى من خلال اتصال مكان مع انتقال من خلال قوسين في حلقة وبالتالي القوسان (1و2) إذا استعملا لإدخال العقدة في الشبكة وبالتالي ستتصرف العقدة كحامل للبيانات، أما لو كان القوسان (3و4) سيستعملان لإدخال العقدة إلى الشبكة ستتصرف العقدة كانتقال وستعمل على معالجة البيانات. وبالتالي ما يحدد تصرّف العقدة هو الثنائية مابين الأقواس ويمكن أن تعرف الشبكة من خلال القوسين (1و4) فتصبح مكاناً ناقلاً للبيانات أو من خلال القوسين (2و3) وتصبح حاملاً للبيانات، ويتم ذلك كله عن طريق ألوان الممرات وبالتالي منذ تغليف العقدة المجرّدة لكلا البيانات والتوابع أصبح من الممكن استعمالها لتمثيل الأغراض والأنظمة الغرضيّة التوجه. ويصبح بالإمكان تمثيل مخطط الصفوف والمخطط التتابعي حيث كل غرض يمثل عقدة والأفعال تُمثل عن طريق الألوان في شبكات بتري [65].

2.3.6. الوراثة على مستوى النماذج المفاهيميّة

إنّ مفهوم الوراثة في النماذج المفاهيمية وإعادة الاستعمال للرماز هما مفهومان في تضداد دائم، إذ لزيادة قابلية إعادة الاستعمال سنخسر جزء من النمذجة والعكس بالعكس [66].

لذا التكامل مابين المفهومين مهم جداً خاصة في اللغات الغرضية التوجه، والتكامل بينهما يحتل سويتان وهما:

- إعادة استعمال الرماز: وهي على مستوى المكونات. وهي آلية تجريد مشابهة للصفوف؛
- النمذجة المفاهيميّة وهي على مستوى الصفوف.

تسمح الوراثة بإعادة استعمال صفوف موجودة سابقاً، وتستخدم في الحالتين التاليتين:

- في حال وجود علاقة هيكليّة مفاهيميّة مابين المفاهيم من النوع "Is -a" مثالها لغة Beta [67]؛

- الحاجة إلى استعمال الرماز الموجود لدى صف موجود سابقاً وذلك للتطوير السريع واختصار الوقت والجهد مثالها لغة Smalltalk [68] .

وبالتالي تخضع عمليّة الوراثة إلى وجهتي نظر:

الأولى تركز على المفاهيم، و**الثانية** تركز على الرماز.

وهناك لغات تأخذ بعين الاعتبار الوجهتين معاً الوراثة المفاهيميّة وإعادة استعمال الرماز مثل لغة Eiffel [69] ولغات لا تأخذ بعين الاعتبار أيّاً من وجهات النظر السابقة مثل Java [70] و C++ [71] فهي تتيح الوراثة دون ضوابط على البنية والرماز.

نعتقد بأنّ علاقة الوراثة التي تعكس التصنيف المفاهيمي والهيكلي أكثر فائدة من آليات الوراثة فقط، إذ تكون أسهل من حيث الفهم والاستعمال والصيانة والتمديد.

على سبيل المثال:

الوراثة في لغة Beta ولغة الـ Smalltalk غير مضبوطة وغير مقيدة فهي تسمح بالوراثة المرنة وإعادة تعريف التوابع بشكل غير مقيّد، وبالتالي لا يوجد منطق يضبط الوراثة والتوافق ما بين الصف الأب والصف الابن.

إنّ SubClass وهو الصف الابن يعكس فكرة إعادة استعمال الرماز تماماً. أمّا فكرة SubType هو النمط الابن أو النمط الجزئي فهو يعكس فكرة النمذجة المفاهيميّة.

تعتبر الوراثة تقنية إما للتخصيص أو التحقيق أو كليهما معاً فهي تقنية للتخصيص كون الصف الابن يخضع لقواعد الأب وهي تقنية للتحقيق كون الصف الابن يستعمل العمليات الموروثة من الصف الأب [72].

3.3.6. أبرز مشكلات الوراثة وحلولها المُقترحة

أبرز عيوب الوراثة التي تعرضت لها الدراسات السابقة هي كالتالي:

- **حالة عدم تطبيق الوراثة مع إمكانية تطبيقها**

ليكن لدينا المفهومين صفحة ويب تعليمية، وصفحة ويب ترفيهية.

صفحة الويب التعليمية تحوي على العناصر التالية:

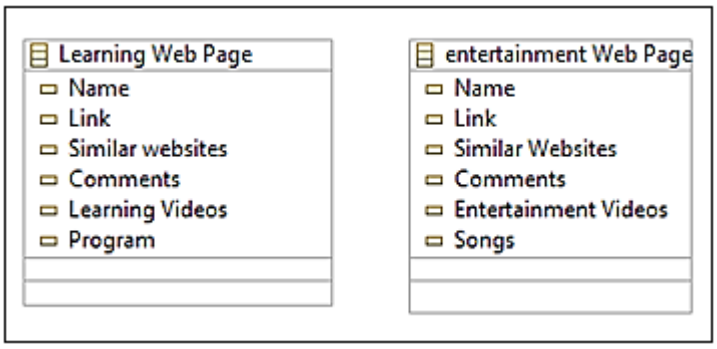
اسم الصفحة، الرابط، المواقع المشابهة، التعليقات، الفيديوهات التعليمية، البرامج الأسبوعية.

صفحة الويب الترفيهية تحوي على العناصر التالية:

اسم الصفحة، الرابط، المواقع المشابهة، التعليقات، الفيديوهات الترفيهية، الأغاني.

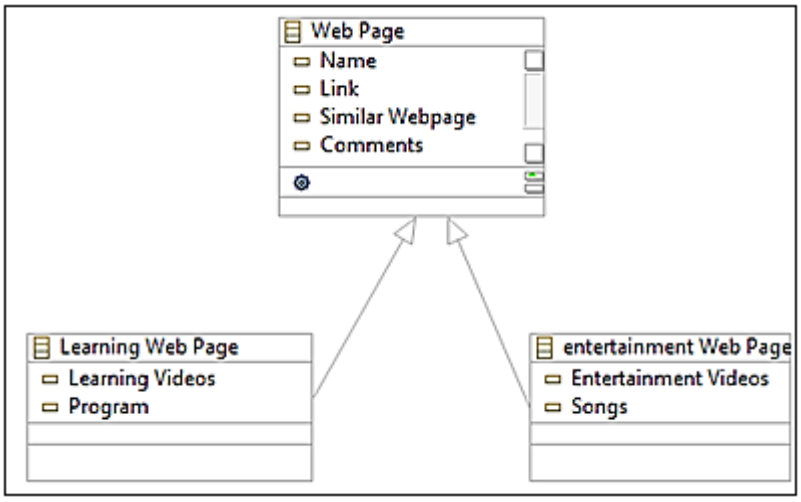
كلا الصفحتين من الممكن انشاء صفحة أب تحوي المفاهيم المشتركة وجعل العلاقة بينها وبين الصفحتين السابقتين علاقة وراثة.

الشكل التالي يبيّن المفهومين صفحة ويب تعليميّة وأخرى ترفيهية.



الشكل (6-9) مفهوم صفحتي الويب

أما التصحيح المقترح فهو إنشاء صفحة أب تحمل الاسم المشترك لكلا المفهومين (Web Page) وتحوي العناصر المشتركة في كليهما وهي: (Name , Link, Similar websites, Comments) إذ نحدد العناصر المشتركة من خلال خوارزمية تفحص الاسم والنمط وفي حال التطابق يتم رفع سوية هذه العناصر إلى الأب وحذفها من الأبناء والربط بين الأب والابناء بعلاقات الوراثة، والأمر نفسه مع العمليات والتوابع الخاصة بالعناصر. الشكل التالي يبيّن التصحيح المقترح.



الشكل (6-10) التصحيح المُقترح

- **حالة وجود عناصر مشتركة في الأبناء مع عدم رفع السوية إلى الأب.**

نفس المثال السابق لدينا:

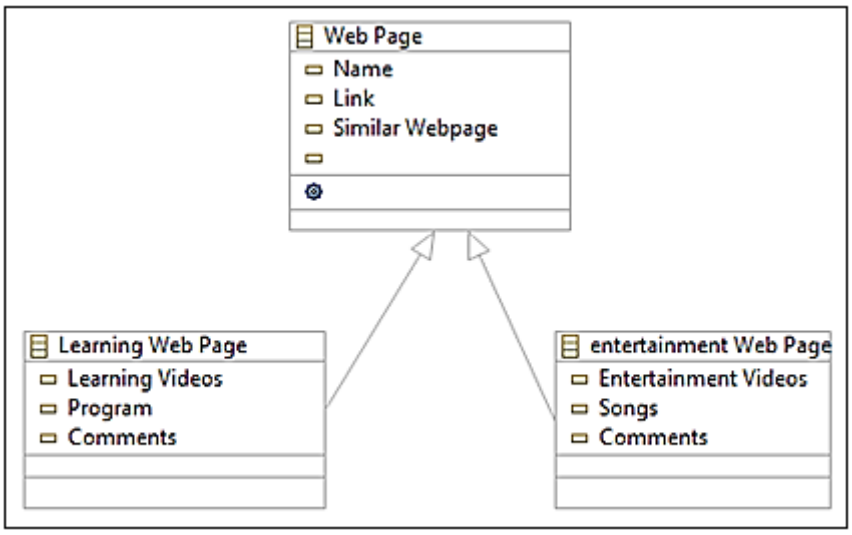
مفهوم صفحة الويب (Name, Link, Similar Webpage).

صفحة ويب تعليمية تحوي (Comments, Learning Webpage, Program).

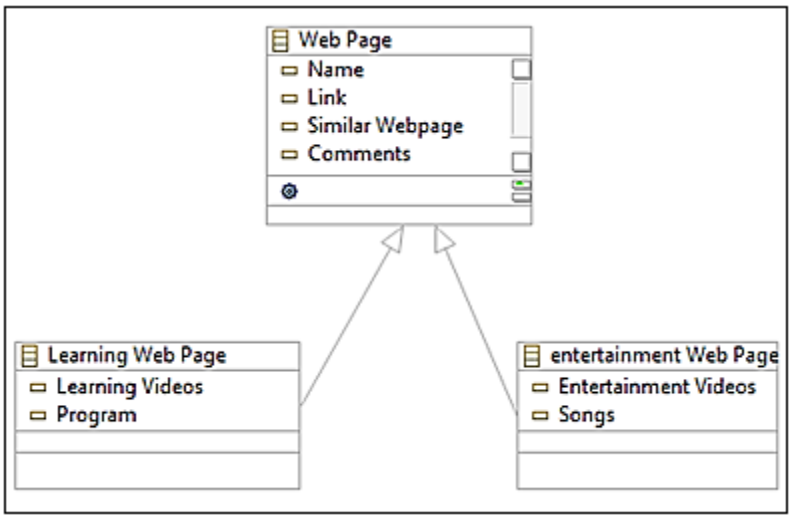
صفحة ويب ترفيهية تحوي (Comments, Entertainment Videos, Songs).

الشكل (6-11) يبيّن المفاهيم السابقة.

أما التصحيح المقترح فهو رفع سوية العناصر المتساوية بالاسم والنمط إلى الأب Web Page وحذفها من الأبناء. والشكل (6-12) يبيّن التصحيح المقترح.



الشكل (6-11) مفاهيم صفحات الويب



الشكل (6-12) التصحيح المُقترح

4.6. الخلاصة

(لم تحظ دراسة النماذج الوراثية بقدر كبير من الاهتمام في الدراسات الحديثة، بل كانت مقتصرة على عدد محدود جداً من النماذج والدراسات القديمة، لكن الجدير بالذكر أنّ المشاكل التي حاولت الوراثة أن تحلها هي مشاكل الحجم الكبير للمفاهيم وذلك باقتراح سوية عُليا تملك العناصر المشتركة وترتبط مع المفاهيم من خلال علاقة الوراثة، وبالتالي ينقص حجم الصفوف ويزداد الترابط.)

الفصل السابع
مفهوم عدم الاستخدام المباشر وتكرار تعريف المفهوم

1.7. مقدمة

يعتبر مفهوم عدم الاستخدام المباشر للصفوف المجردة من الثغرات التي تطعن بالجودة، فوجود مفهوم مجرّد Abstract Concept يرث منه مفهوم مجرّد آخر دون وجود مفاهيم ملموسة Concrete Concept ترث منه بشكل مباشر يؤدي إلى الهبوط بمستوى جودة التصميم، لذا كان لابدّ من تلافي هذه الثغرات، كونها تشكل هرمية طويلة قد نكون بغنىً عنها، وتزيد من طول سلسلة الوراثة، وتشكّل عبئاً على التصميم، وقد تؤدي إلى ضياع المستخدم أو المصمم بين المفاهيم الكثيرة والسلاسل العديدة للوراثة، أما مفهوم تكرار التعريف فهو خلق مفهومين متماثلين بالتسمية وهو حمل زائد على النظام، قد يؤدي إلى حدوث إشكاليات في الفهم خاصة للمستخدمين حديثي العهد، فلابدّ من حذف أحدهما أو استبدال اسم أحدهما لئلا يشكل عبئاً إضافياً على العمل، وتشويشاً للمستخدم.

2.7. دراسة مرجعيّة

1.2.7. شرح حالتي الخطأ

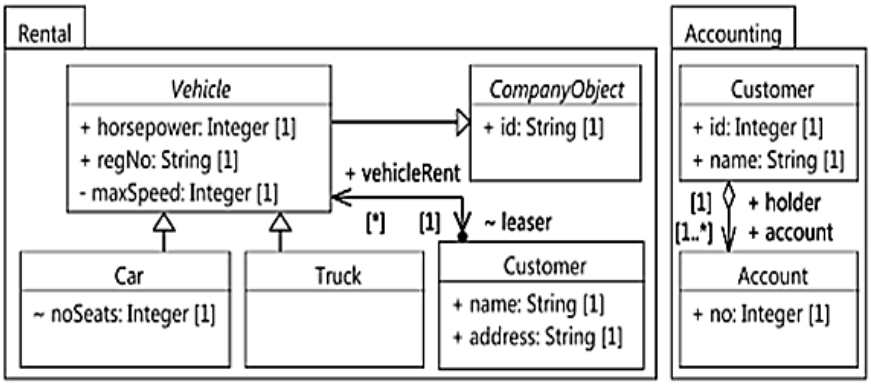
يُعتبر عدم الاستخدام المباشر للمفاهيم المجردة من عيوب الأنظمة، كونه يشكل سلسلة هرمية طويلة، وكذلك الأمر لتكرار التعريف بالنسبة للمفاهيم [43]، وفيما يلي يستعرض الشكل التالي مخطط الصفوف الخاص بشركة تأجير السيارات والذي احتوى على عيبين (2 Smells) أحدهما من نوع عدم الاستخدام المباشر، والثاني من نوع تكرار تعريف المفهوم.

ويمثّل الشكل (7-1) مخطط الصفوف لشركة تعمل على تأجير مركبات النقل للأفراد، فكان من أول العيوب التي يجب إزالتها هي:

- **أولاً:** عدم الاستخدام المباشر للمفاهيم، فالصف *CompanyObject* هو صف مجرّد يرث منه الصف *Vehicle* وهو صف مجرّد أيضاً، يرث منه صفان غير مجرّدان هما Car و Truck.

- **ثانياً:** فهو تعريف صفين بنفس الاسم وهو الصف Customer الموجود في الحزمة Rental والصف Customer الموجود في الحزمة Accounting.
- **ثالثاً:** هناك عيب آخر وهو وجود صف بدون عناصر وهو الصف Truck.

يحتوي المخطط على عدة صفوف مبينة في الجدول (7-1).



الشكل (7-1) مخطط الصفوف الخاص بشركة تأجير السيارات

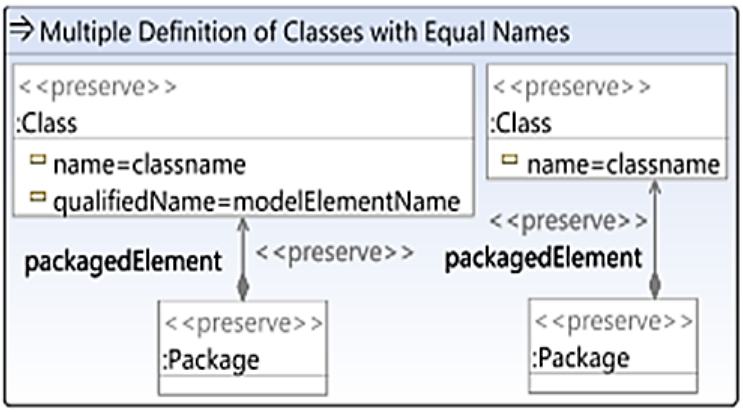
الجدول (7-1) الصفوف الموجودة في المخطط

اسم الصف	نوعه	العناصر	أنماط العناصر
Rental Package			
CompanyObject	Abstract Class	Id	String
Customer	Concrete Class	Name	String
		adress	String
Vehicle	Abstract Class	horsepower	Integer
		regNo	String
		maxSpeed	Integer
Car	Concrete Class	noSeats	Integer
Truck	Concrete Class	No Attributes	
Accounting Package			
Customer	Concrete Class	Id	Integer
		Name	String
Account	Concrete Class	No	Integer

2.2.7. شرح التصحيح المُقترح

نستعرض فيما يلي قواعد نموذج Henshin [43] لكشف العيوب وتصحيحها.

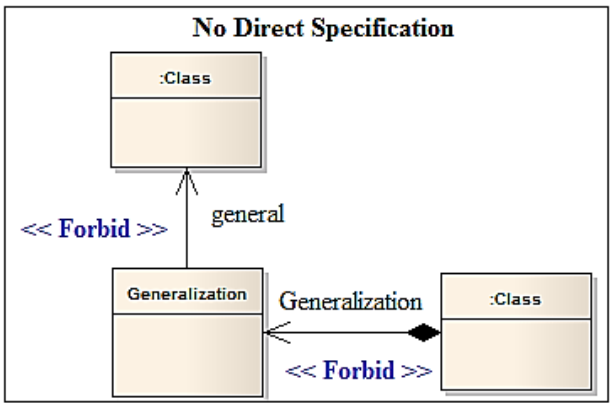
- نبدأ مع **القاعدة الأولى** لاكتشاف تكرار أسماء الصفوف أو المفاهيم عندما تكونان في حزمتين مختلفتين يبينها الشكل التالي.



الشكل (2.7) قاعدة Henshin لاكتشاف خطأ تكرار تسمية المفهوم وتصحيحه بين عدة حزم

تقترح القاعدة أن يتم إضافة اسم النموذج إلى اسم الصف لكي لا يكون هناك تشابه في التسميات.

- أما **القاعدة الثانية** فهي لاكتشاف وإزالة حالة عدم الاستخدام المباشر للصف المجرّد بيّنها الشكل التالي.

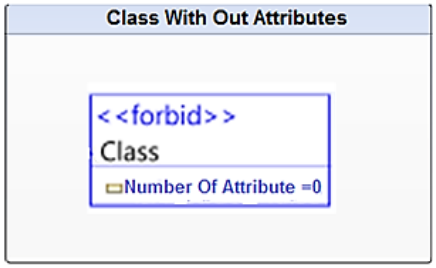


الشكل(7-3) قاعدة Henshin لاكتشاف خطأ عدم الاستخدام المباشر للمفهوم المجرّد

فلكي يكون لدينا هذا النوع من الخطأ لابدّ من عدم وجود علاقة وراثة مباشرة ما بين صف مجرد Abstract Class وصف غير مجرد Concrete Class.

أما التصحيح المُقترح هو إيجاد علاقة مباشرة ما بين الصف المجرد والصف الغير مجرد، طبعاً قد لايكون الكلام دقيق إذ أننا سنعمد إلى التصحيح الآلي الذي يعتمد على المقاييس، ويسعى لإيجاد الحل السليم بأقل كلفة ممكنة.

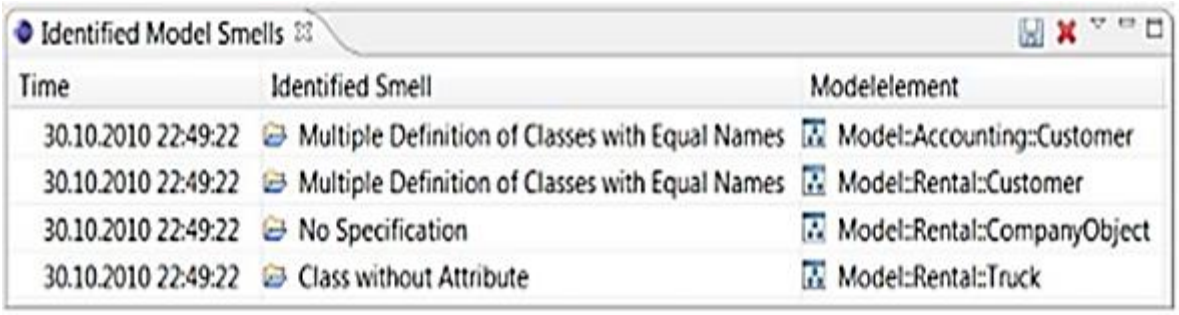
- أما **القاعدة الثالثة** فتشمل على اكتشاف الصفوف الخالية من العناصر Attribute وإزالتها. والشكل التالي يوضح المفهوم.



الشكل (7-4) قاعدة Henshin لاكتشاف الصفوف الخالية من العناصر

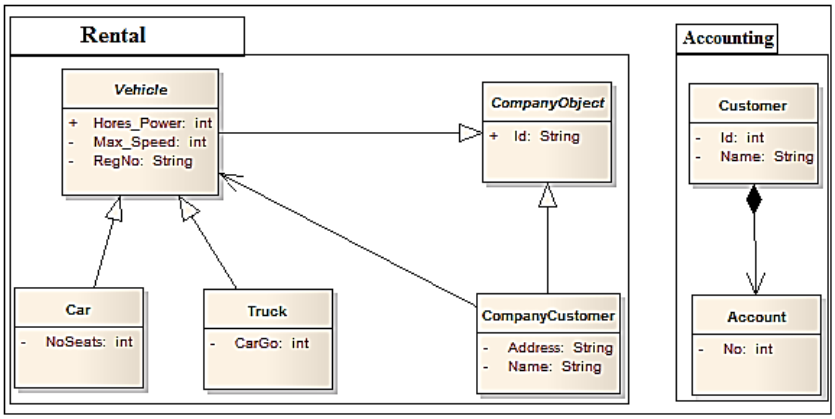
وأما التصحيح المُقترح فهو إزالته تماماً حسب Henshin.

والشكل التالي يوضح الأخطاء المكتشفة في التصميم السابق مجتمعةً.



الشكل (7-5) كشف العيوب حسب نموذج Henshin

وأما الشكل التالي يوضح مخطط الصفوف بعد إجراء التصحيح حسب نموذج Henshin عليه.



الشكل (7-6) مخطط الصفوف حسب نموذج Henshin بعد التصحيح

3.7. الخلاصة

(إنّ عملية تكرار تعريف المفهوم أمر مرفوض كونها تزيد من الحجم دون فائدة، على حين أنّ عملية عدم الاستخدام المباشر للمفاهيم تقلل من الترابط، لذا لابدّ من إزالة هذه الحالات كونها تسبب زيادة الحجم وضعف الترابط.)

الفصل الثامن
مفهوم العلاقة الحلقيّة

1.8. مقدمة

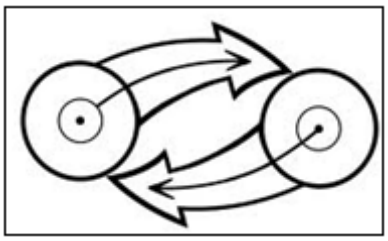
تعتبر العلاقات ما بين المفاهيم الموجودة في التصميم من أهم العوامل في نجاحه، لذا كان لابدّ من إعطاء أولويّة للعلاقات لأنها تعكس طبيعة المهام مابين المفاهيم.
تعتبر الحلقيّة أو العوديّة مابين المفاهيم من عيوب التصميم التي لابدّ من تجاوزها، سواءً كانت مباشرة أو غير مباشرة، فهي تعتبر علاقات متداخلة عيوبها كثيرة وتسبب إرباك للمبرمج أو المستخدم عند استخدام نسخ من هذه المفاهيم لما تضعه من قيود على الصفوف وارتباطات وما تفرضه من أولويات.

2.8. تعريف

العلاقة الحلقيّة: هي العلاقة الناتجة من تبادل علاقة ونظيرتها ما بين مفهومين أو أكثر، وبالتالي يتم فرض أولويات مرهقة على التصميم نحن بغنىً عنها، الكيان الأول يحوي غرض من الكيان الثاني، والكيان الثاني يحوي غرض من الكيان الأول، وتقسم إلى نوعين.

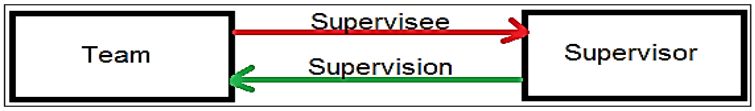
- **العلاقة الحلقيّة المباشرة**

وتكون مابين مفهومين مباشرين والشكل التالي يوضح المفهوم، وفي هذه الحالة يُفضل الحذف المباشر لأحدى هذه العلاقات لأنّ واحدة منها تؤدي مفهوم الاثنتين.



الشكل (8-1) العلاقة الحلقيّة المباشرة

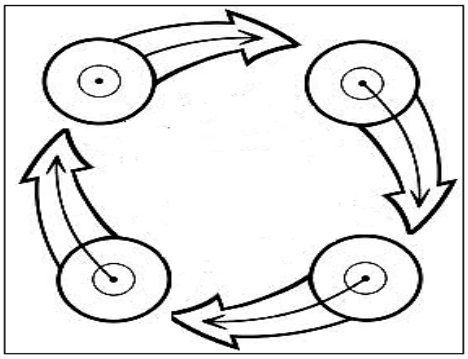
ومثالها علاقة الموظفين والمدير، فالمدير يقوم بالإشراف على فريق، والفريق يتم الإشراف عليه من قبل المدير، وبالتالي علاقة واحدة من هذه العلاقات تغني عن الثانية، الشكل التالي يوضح المفهوم.



الشكل (8-2) العلاقة الحلقية المباشرة مابين المشرف والفريق

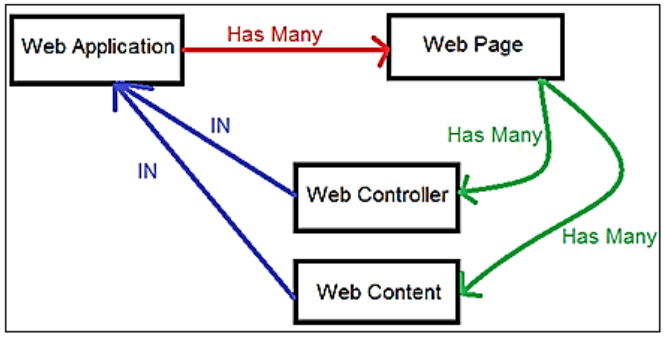
- **العلاقة الحلقيّة الغير مباشرة**

وفي هذه الحالة لاتكون مقتصرة مابين مفهومين بل تتعداها لأكثر من ذلك مشكلة ما يشبه الحلقة الدائريّة ولكن ذات اتجاه واحد أي عند الانطلاق من مفهوم معين باتجاه معيّن نعاود الوصول غليه ذاته بعد تخطي عدد من المفاهيم الأخرى دون تغيير الاتجاه، لذا لابدّ من الانتباه عندما نجد هذا النوع من الحلقات، الشكل التالي يوضح المفهوم.



الشكل (8-3) العلاقة الحلقيّة الغير مباشرة

ومثالها: العلاقة مابين تطبيق الويب الذي يحوي على عدد كبير من الصفحات، والصفحة التي بدورها تحوي على عدد كبير من المحتوى والمتحكمات، والمحتوى والمتحكمات المضمّنة في تطبيق الويب، وبالتالي علاقة حلقيّة. الشكل التالي يوضح المفهوم.



الشكل (8-4) العلاقة مابين تطبيق الويب والصفحات والمحتوى

3.8. الدراسات السابقة

حظيت عملية اكتشاف العلاقات الحلقيّة وإزالتها على قدر كبير من اهتمام الباحثين خلال الفترة الماضية.

- **على مستوى قواعد البيانات:** من التصاميم السيئة و المرفوضة بتاتاً وجود حلقة من العلاقات بين الكيانات وذلك لما تسببه من صعوبات في الإدخال، والحذف، والتعديل، بسبب المفتاح الرئيسي والثانوي والاختلاط بين العلاقات لذا لابدّ من إزالتها، والإبقاء على علاقة ذات اتجاه واحد بين الكيانات [73].
- **على مستوى إطار موارد البيانات الموحد RDF:** من غير الجيد أن يكون هناك حلقة مابين المفاهيم في الأنطولوجي، إذ تسبب زيادة في التعقيد وغياب لبعض الأفكار الأساسية، وصعوبة في عمليات الاستعلام، والتباس في الأفكار [74].
- **على مستوى المفاهيم:** وهنا يجب الحذر الشديد عند تصميم قالب أو نموذج، إذ لابدّ من تجنب الحلقات قدر المستطاع، وخاصة أنّ هذا القالب سيكون مقياساً أساسيّاً تتبناه كل النسخ (Instances) المشتقة عنه [75].

4.8. الخلاصة

(من خلال الدراسات السابقة تبيّن معنا مدى الأثر السلبي والسيء لوجود العلاقة الحلقيّة في التصميم، وبالتالي لابدّ من إزلتها بأي شكل كان سواءً علاقة حلقيّة مباشرة أو غير مباشرة.

لكن مثل هذه العملية التصحيحيّة تحتاج الكثير من الدراسة المُعمّقة قبل عملية اتخاذ القرار بشأن حذف عنصر أو رابط ما، فهي تحتاج لدراسة أهمية المفاهيم والروابط، وبنظرنا تحتاج إلى بحث منفصل، لذا سيتم التجاوز عنها مبقين عليها للآفاق والدراسات المستقبلية.)

الفصل التاسع
الجودة البرمجيّة

1.9. مقدمة

اختلف الباحثون في ايجاد تعريف موحد وشامل للجودة، فالجودة من منظور شخص لآخر تختلف في العديد من المواطن. ولكن نستطيع القول بأن أحد مفاهيم الجودة هي "أن يلبي المنتج كل المتطلبات المرجوة منه، إذ من الضروري أن يحقق التطابق مع المتطلبات الوظيفية ومتطلبات الأداء المعلنة، ومع مقاييس التطوير الموثقة بوضوح، ومع الخصائص الضمنية المتوقعة من المنتج البرمجي".
وأصبحت المنتجات البرمجية تتنافس في مجال الجودة، لمعرفة ما إذا كان المنتج البرمجي ذو جودة أم لا، لذا لابد من وجود مقاييس محلية وعالمية لتقييم وتصنف المنتج البرمجي.

2.9. الدراسات السابقة حول الجودة

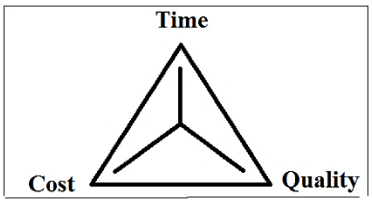
1.2.9. لمحة تاريخيّة

في الأيام الأولى لعصر الحوسبة (ما بين 1950 و1960)، كانت مسؤولية الجودة مُناطة بالمبرمجين فقط. أما مقاييس ضمان جودة البرمجيات Software Quality Assurance (SQA) فقد جرى إدخالها خلال السبعينات. ويقتضي ضمان جودة البرمجيات في الوقت الحالي وجود أقسام في المؤسسات البرمجية مسؤولة عن ضمان جودة البرمجيات، إذ يجتمع فيها مهندسو البرمجيات، ومديرو المشاريع، وممثلو الزبائن، ومسؤولو البيع، والأفراد الذين يخدمون ضمن مجموعة ضمان جودة البرمجيات.

والمثلث البرمجي يوضح أهمية الجودة [76] وهي تمثل طرف من أطرافه، إذ أنّ أي مشروع برمجي لابدّ أن تتمّ مناقشته من وجهات النظر الثلاث:

- الوقت: إذ يجب أن تنتهي المشاريع بوقت محدد فلا تتجاوز المدة الزمنية المحددة أبداً، وهناك طرق لحساب المدة الزمنيّة، (COCOMO 1,2).
- الكلفة: ويجب أن يكون هناك تقدير مسبق للكلفة العليا فلا نتجاوز الميزانية المحددة.
- الجودة: وهي تحقيق المطلوب بأفضل شكل وأداء والوصول إلى رضى المستخدم.

الشكل التالي يوضح المثلث البرمجي [76].



الشكل (9-1) المثلث البرمجي

2.2.9. نموذج Halstead

يعتبر من أقدم المحاولات لبناء مقاييس جودة البرمجيات، أثبتت الدراسات نجاح مؤشرات هذا النموذج في تقييم تعقيد المنتجات البرمجية وجوانبها الأخرى [77]. استندت مؤشرات نموذج هولستد للقياس على التركيب الداخلي للنص البرمجي. فقد اعتمد مجموعة من المقاييس وهي:

- عدد المؤثرات Operators المختلفة الموجودة في برنامج ما (n1).
- عدد الحدود Operand المختلفة المتأثرة بالعملياتفي برنامج ما (n2).
- العدد الكلي لحالات ورود مؤثر ما (N1).
- العدد الكلي لحالات ورود حد ما (N2).

يهدف نموذج Halstead إلى تقدير الطول الإجمالي للبرنامج، والحجم الأصغري المحتمل للخوارزمية وفق القانونين التاليين:

الطول الإجمالي N يعطى وفق:

$$N = n_1.\log_2 n_1 + n_2.\log_2 n_2$$

أما حجم البرنامج V يمكن تحديده وفق:

$$V = N.\log_2(n_1 + n_2)$$

ومن ثم عمد هذا النموذج إلى تحدد الحجم القياسي للنص البرمجي الذي يعتمد على الخوارزمية ولا يتأثر باللغة أو عمليات التحويل أو أي مؤثر آخر، وجميع مقاييس نموذج تستند على المقارنات بين الحجم القياسي والحجم المثالي للنص البرمجي، ومن أهم عيوبه [77]:

- ☒ إهمال العديد من مواطن التأثير في التعقيد والجودة للنص البرمجي مثل تأثير نقاط اتخاذ القرار (جمل التكرار والجمل الشرطية وغيرها);
- ☒ تباين تأثير البنى اللغوية المختلفة;
- ☒ تماسك وتخلخل النص البرمجي.

3.2.9. مقاييس الجودة حسب نموذج McCall

ركز هذا النموذج على ثلاثة وجوه مهمة في المنتج البرمجي [78]، هي:

- ✓ خصائص التشغيل.
- ✓ قابلية التغيير.
- ✓ التكيف مع بيئات جديدة.

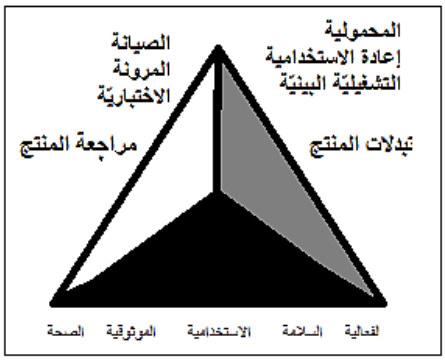
الجدول التالي يوضح **مقاييس الجودة** التي اعتمدها هذا النموذج في تقييم المنتج البرمجي.

الجدول (9-1) مقاييس الجودة في نموذج McCall

اسم عامل الجودة	الاسم الأجنبي	شرح مبسّط عنه
الصحة	Correctness	هي مدى تحقيق البرنامج للمتطلبات الموصّفة له، ومدى موافقته لأهداف مهمة للزبون.
الموثوقية	Reliability	هي مدى إنجاز البرنامج للوظائف المطلوبة منه بالدقة المطلوبة.
الفعالية	Efficiency	مقدار حاجة البرنامج من موارد حاسوبية ورماز لإنجاز وظائفه.
السلامة	Integrity	مدى التحكم في نفاذ المستخدمين غير المخولين إلى البرنامج أو المعطيات
الاستخدامية	Usability	هي الجهد المطلوب لتعلم برنامج وتشغيله، وإعداد دخله وفهم خرجه.
الصيانة	Maintainability	الجهد اللازم لتحديد الخطأ في البرنامج وإصلاحه
المرونة	Flexibility	الجهد اللازم لتعديل برنامج في الاستثمار

الاختبارية	Testability	الجهد اللازم لفحص برنامج والتوثق من إنجازه للوظائف المقررة
المحمولية	Portability	الجهد المطلوب لنقل برنامج من بيئة (عتادية / برمجية) إلى أخرى
إعادة الاستخدامية	Reusability	مدى إمكانية إعادة استخدام البرنامج أو أجزاء منه في تطبيقات أخرى مرتبطة بنفس الحزمة ولها أفق الوظائف التي يؤديها البرنامج
التشغيلية البينية	Interoperability	الجهد اللازم لربط نظام بآخر

والشكل التالي يوضح **مقاييس الجودة** التي عرّفها نموذج McCall وعلاقتها بالوجوه المختلفة للمنتج البرمجي [78].



الشكل (9-2) مقاييس الجودة والعلاقة مع المنتج البرمجي حسب نموذج McCall

4.3.9. عوامل الجودة FURPS

يبيّن الجدول التالي **عوامل الجودة** حسب FURPS

الجدول (9-2) عوامل الجودة حسب FURPS

عامل الجودة	الاسم الأجنبي	شرح مبسط عنه
التدقيقية	Audibility	سهولة التحقق من مطابقة المقاييس
الدقة	Accuracy	دقة الحسابات والتحكم
ألفة الاتصال	Communication Commonality	درجة استخدام الواجهات والميثاقات وعرض الحزمة القياسية

الاكتمال	Completeness	درجة تنجيز كامل الوظائف المطلوبة
الإيجاز	Conciseness	مدى تراص البرنامج مقاساً بعدد أسطر الرماز
الاتساق	Consistency	استعمال تقنيات منتظمة للتصميم والتوثيق خلال مشروع التطوير برمجي
ألفة المعطيات	Data Commonality	استخدام الأنماط وبنى المعطيات الأساسية في البرنامج
تحملية الخطأ	Error Tolerance	الأذى الذي قد يحث عندما يواجه البرنامج خطأً
فعّالية التنفيذ	Execution Efficiency	أداء البرنامج في زمن التنفيذ
التوسعيّة	Expandability	الحد الأدنى الممكن لتوسيع التصميم البياني وتصميم المعطيات والتصميم الإجرائي
العموميّة	Generality	سعة التطبيقات الممكنة باستخدام مكونات البرنامج
الاستقلال عن العتاديات	Hardware Independence	درجة فصل البرمجيات عن العتاديات التي تعمل عليها
التجهيز بأدوات قياس	Instrumentation	درجة مراقبة البرنامج لعمله وقدرته على تمييز الأخطاء التي تحدث
الاجتزائية	Modularity	الاستقلال الوظيفي لمكونات البرنامج
التشغيليّة	Operability	سهولة تشغيل البرنامج واستعماله

الأمن	Security	توفر الآليات التي تتحكم وتحمي البرنامج ومعطياته
التوثيق الذاتي	Self-Documentation	درجة احتواء الرماز المصدري على توثيق ذي دلالة
البساطة	Simplicity	مدى القدرة على استيعاب البرنامج دون عناء
استقلال البرمجيات عن النظام	Software System Independence	درجة استقلال البرنامج عن المزايا الغير قياسية في لغات البرمجة، وعن خصوصيات نظام التشغيل، وعن قيود بيئيّة أخرى.
التعقبيّة	Traceability	إمكان تعقب تمثيل التصميم أو مكوّن البرنامج الفعلي رجوعاً إلى المتطلبات
التدريب	Training	الدرجة التي تساعد فيها البرمجية على تمكين المستخدمين الجدد من تطبيق النظام

والشكل التالي يوضح **عوامل الجودة** وعلاقتها مع **مقاييس الجودة**

مقياس جودة البرمجيات / عامل الجودة	الصحة	المرونة	الفعالية	السلامة	الصيانة	المرونة	الاختبارية	المحمولية	إعادة الاستخدامية	التشغيلية البينية	الاستخدامية
الدقيقية				X			X				
الدقة		X									
ألفة الاتصال										X	
الاكتمال	X										
التعقد		X				X	X				
الإيجاز			X		X	X					
الاتساق	X	X			X	X					
ألفة المعطيات										X	
تحملية الخطأ		X									
فعالية التنفيذ			X								
التوسعية						X					
العمومية						X		X	X	X	
الاستقلال عن العتاديات								X	X		
التجهيز بأدوات قياس				X	X		X				
الاجرائية		X			X	X	X	X	X	X	
التشغيلية			X								X
الأمن				X							
التوثيق الذاتي					X	X	X	X	X		
البساطة		X			X	X	X				
استقلال البرمجيات عن النظام								X	X		
التعقية	X										
التدريب											X

الشكل (9-3) عوامل الجودة ومقاييسها

5.3.9. أبعاد الجودة حسب Kitchen ham

"الجودة صعبة التعريف مستحيلة القياس". فهي مفهوم غامض نوعي (Qualitative). ومناقشة غموض الجودة وتعريفها ميز اتجاه العديد من البحوث المهمة. وموارد الغموض التي تكتنف جودة البرمجيات وقياسها عديدة. ولعل أهم موارد الغموض يرجع إلى صعوبة بيان المقصود بتعبير "جودة" واستحالة حصر المؤشرات الكمية لقياسها: فمساحة الأرض يسهل تحديدها بدقة، بينما قياس جودتها قد يثير جدل كبير [79,80]. وأرجع الغموض في مفهوم الجودة لكونها مفهوم متعدد الأبعاد، وأبعاد الجودة تشمل النقاط التالية:

- ✓ **موضع الاهتمام** (Entity of Interest): إن موضع اهتمام المطور للمنتج البرمجي يرتبط ارتباطاً وثيقاً بحاجة المطور وموقعه وخبرته. حيث جودة النظام البرمجي للرواتب والأجور

قد يكون فيها تباين بين مهمته التي تقتصر على إدخال البيانات وبين الموظف المسؤول عن النظام في قسم الحسابات وبين مدير الحسابات وبين مدير المؤسسة. وذلك لتباين مواضع الاهتمام لكل من الفئات المشار إليها. حيث الموظف المسؤول عن إدخال البيانات قد يركز موضع اهتمامه لتعريف الجودة على الجوانب التشغيلية من شاشات منسقه وعدم تكرار أي مدخل لتقليل الجهد وغير ذلك من التسهيلات ذات العلاقة بمهمته بينما يهتم الموظف المسؤول عن النظام بالجوانب ذات العلاقة بموثوقية نتائجه والتقارير الناتجة عنه. في حين مدير الحسابات قد يهتم بالقيمة الفنية التي يساهم بها من خلال تفاعله وتعامله مع النظم الأخرى لدعم النظام المالي للمؤسسة و إعداد الميزانية وسهولة متابعتها. أما الإدارة العليا للمؤسسة قد تهتم بالدعم الذي يوفره النظام لتطوير خدمات المؤسسة ومدى فاعليته في إسناد التخطيط ودعم القرار. لذا فإن خبرة واتجاه كل من أشير لهم قد تؤثر على تحديد موضع الاهتمام؛

- ✓ **سمات الجودة** لذلك الكيان (Quality Attribute to That Entity): سمات الجودة هي التي تعكس موضع الاهتمام للجودة وتكون قابلة للقياس لتساهم مباشرة في تحديد مفهوم الجودة. و تحديد هذه السمات وتقويمها قد يختلف من شخص لآخر وإن اتفقا في موضع الاهتمام. فذلك يعتمد على عوامل عديدة منها خبرته في التقويم وعلاقته بالمنتج البرمجي. وعلى سبيل المثال تحديد السمات التي تتيح تقويم مدى استجابة المنتج للإدامة والتوسع (باعتبارها موضع اهتمام للمستفيد والشخص الذي يقوم بعملية الإنتاج) قد ينتج عنها خلاف. فالمستفيد قد يهتم بسهولة وسرعة خدمات الإدامة بينما الشخص المنتج للبرمجيات يضيف إلى ذلك الكلف المترتبة عليه في عمليات الإدامة والتوسع؛
- ✓ **وجهة النظر لموضع الاهتمام** (Viewpoint on That Entity): قد يتفق أشخاص على تحديد مواضع الاهتمام ولكن قد يختلفون في تحديد أسبقية أهميتها مما يقود إلى خلاف في قرار الجودة.

6.2.9. مواضع الجودة حسب Garvin

تمّ تحديد سبع مواضع اهتمام في الجودة [81] وفقاً لـ (Garvin):

1- الأداء (Performance) الممثل بالخواص التشغيلية.

1. السمات (Features).
2. الوثوقية (Reliability).

3. التطابق مع المتطلبات أو المعايير(Standards) المنتخبة مسبقاً.
4. قدرة التحمل أو قابلية المحافظة على البقاء (Durability).
5. سهولة وسرعة خدمات الإدامة.
6. الجمالية(Aesthetics).

ويمكن القول بأن مواضع الجودة الأساسية هي:

- ✓ جودة المنتج (Quality of Product)
- ✓ جودة العملية التي ينتج عنها المنتج (Quality of Process)
- ✓ جودة الخدمات والمردودات الأخرى التي يحققها المنتج في البيئة العملية (Quality in the Context of Business Environment).

أما بالنسبة لوجهة النظر فقد تختلف لعديد من العوامل ولعل أهمها علاقة المقوم بالنظام. فالمستفيد تعني له الجودة تحقيق متطلباته وتوقعاته لما يحصل عليه من النظام بأسلوب سهل في التعلم والتنفيذ. في حين المنتج تعني له الجودة رضاء المستفيد وسهولة تنفيذه لخدمات ما بعد البيع بكلف معقولة وإمكانية إعادة استخدام أجزاء منه في منتجات جديدة. أما بخصوص سمات الجودة فغالباً المستفيد يهتم بالسمات الخارجية بينما المنتج يضيف إلى اهتماماته السمات الداخلية.

كل مفهوم قد يحدد بمستويات متباينة من التجريد (Abstraction). فعند الكلام عن الجودة، قد يشار له بالمعنى الشامل أو بمعنى محدد. ويعني التجريد بناء تشكيل المفهوم من خلال تحديد العناصر الفاعلة في الواقع والمؤثرة فيه. لذلك قد يكون تقويمنا لجودة البرنامج من خلال تحديد عناصر عامة فاعلة أو عناصر تحتاج إلى تفاصيل أدق. ولتوضيح ذلك نطرح بعض مستويات التجريد التي يمكن أن نتبناها في حكمنا على جودة البرمجيات:

- ✓ جمالية واجهاته (شاشاته) و تنظيم مخرجاته وسهولة التحاور معه.
- ✓ حسن تنظيم وثائقه وسهولة متابعته و مقروئيته (Readability).
- ✓ أسلوب المواجهة معه ومدى استخدامه للدوال القياسية و تماسك (Cohesive) وحداته البرمجية ودرجة دقة نتائجه.
- ✓ تعبير الجودة متداول ضمن حديثنا اليومي وقد يختلف المعنى المتداول عما يقصد باستخدامه في المجال المهني لوجهة النظر الهندسية أو الإدارية.

3.9. المعايير الدولية لجودة البرمجيات

هي مجموعة القواعد والإجراءات والخطوات والعمليات التي وضعت من قبل جهات عالمية مختصة لكي تحقق في حالة الالتزام بها أن يتم التفكير والتخطيط للبرمجيات ثم تصميمها وإنتاجها بطريقة واضحة تحدد مدى كفاءة البرنامج في العمل وخلوه من العيوب والمشكلات المختلفة واتساقه التام مع الهدف الموضوع من أجله، واتباع معايير واضحة ومعتمدة في أسلوب التصميم والبناء وكتابة كود البرنامج بحيث يسهل بعد ذلك تصنيف البرنامج من حيث مستوى جودته وقدرته على العمل في الظروف المختلفة، وأن يكون البرنامج موثقاً توثيقاً كافياً وجيداً في مجموعة وثائق ورقية أو الكترونية بشكل يشرح كل تفصيلاته وطريقة بنائه بحيث يمكن لأي مطور أو مدير مشروع أن يعمل بهذا البرنامج دون اللجوء بشكل أساسي لمن قام بالتصميم، ولا تقتصر معايير الجودة علي البرنامج نفسه من الناحية التقنية بل تشمل كل شيء داخل شركة البرمجيات من العمليات الإدارية والتنظيمية المتبعة داخل الشركة وكيفية وضع مشروع لإنتاج برنامج معلومات وكيفية متابعته، وحتى كيفية التعامل مع المبرمجين في أمور التدريب الداخلي كالحضور والانصراف وغيرها.

1.3.9. معيار ISO

صدرت المعايير القياسية ISO بخصوص إنتاج وصيانة البرمجيات عام 1991. والقراءة السريعة لهذه المعايير تؤكد أن هناك الكثير مما يجب إنجازه من قبل الحاسوبيين العرب لتحقيق هذه المعايير خاصة في المجالات التالية: مسؤولية الإدارة –نظم الجودة – مراجعة العقود– تخطيط التطوير– تخطيط الجودة– الاختبار والمعايرة – التوريد و التركيب.

الاستلام – الصيانة– متابعة التوثيق – والقياسات. ويجب على الشركات الكبرى أن تولي اهتماماً بتكوين كوادر متخصصة في هذا المجال.

وقُسّمت الجودة حسب معيار ISO إلى أنواع: جودة خارجيّة وتهتم بزمن التنفيذ الديناميكي للرماز، وجودة داخليّة تهتم بالأجزاء الساكنة من البرنامج، وجودة الاستخدام وتهتم فيما إذا كان المنتج البرمجي يحقق متطلبات المستخدم في بيئة العمل [82].

- **جودة خارجيّة External Quality**

وتتمثل بما يلي:

- سرعة الاستجابة؛
- القوة/الشدة(Robustness):

وهي تمثل نجاح عمل البرنامج أطول فترة ممكنة دون حدوث انهيارات Fatal Errors تسبب في إيقاف مفاجئ للبرنامج. أحد أبرز الأسباب التي تسبب الأخطاء هي لحظة الحصول على مدخلات من المستخدمين User Inputs، فعشرات البرامج ظهرت انهياراتها بسبب خطأ في عملية الإدخال. كما أن توقع تغيير في ظروف بيئة عمل البرنامج (كعدم التحقق من أسماء الملفات، عدم وجود مكونات (Components) معينة ، ...) أمر ضروري لقوة برنامجك.

- سهولة الاستخدام (Usability): المستخدمين ليسوا مثلك (مبرمجين) وخبرتهم في التعامل مع البرامج بسيطة جداً، ولا تتوقع أن يعتمد المستخدمون على الوثائق Documentations لتعلم برامجك، فليس لديهم الوقت لذلك. تحقيق سهولة الاستخدام يعتمد بالدرجة الأولى والأخيرة على واجهة الاستخدام User Interface، والواجهة الناجحة هي التي تمكن المستخدم العادي من فهم برنامجك دون الحاجة إلى تعليم .
- الكفاءة والفعالية (Efficiency): مقياس لكفاءة التنفيذ يعتبر أبرز مقياس يؤدي إلى تغيير في مسالة اختيار القرار واعتماد برنامج معين، سرعة الانجاز تمثل الوحدة الرئيسية لقياس كفاءة البرنامج.
- الوثوقية (Reliability): للحصول على برمجيات موثوقة يجب أن يصل عدد الأخطاء في البرامج إلى أدنى قيمة و كذلك إنقاص النتائج السلبية الناتجة عنها إلى أقل ما يمكن .
- الإصلاح (Maintainability): معيار يمثل قدرة البرنامج على صيانة نفسه وتصحيح مشاكله دون الحاجة إلى اللجوء إلى الدعم من المنتج.
- إمكانية إعادة الاستخدام (Reusability): وهي إمكانية تعديل المحتوى بسهولة واستخدامه عدة مرات باستخدام أدوات ومنصات تشغيل متعددة، تشمل إعادة استخدام البرمجيات أكثر من مجرد الكود المصدري. فهناك مجموعة من المصنوعات artifacts يعاد استخدامها عند القيام بهندسة البرمجية. تشمل هذه المجموعة: المواصفات، نماذج

البنيان، التصميم، الرماز، الوثائق، معطيات الاختبار، وحتى المهمات المتعلقة بالإجرائية كتقنيات التدقيق.

- **جودة داخلية Internal Quality**

وتقوم بقياس الأجزاء الساكنة كما في كود البرنامج مثل: طول المسار، وتملك تأثير غير مباشر على الجودة الخارجية للبرنامج، وتتمثل بعدة نقاط أهمها:

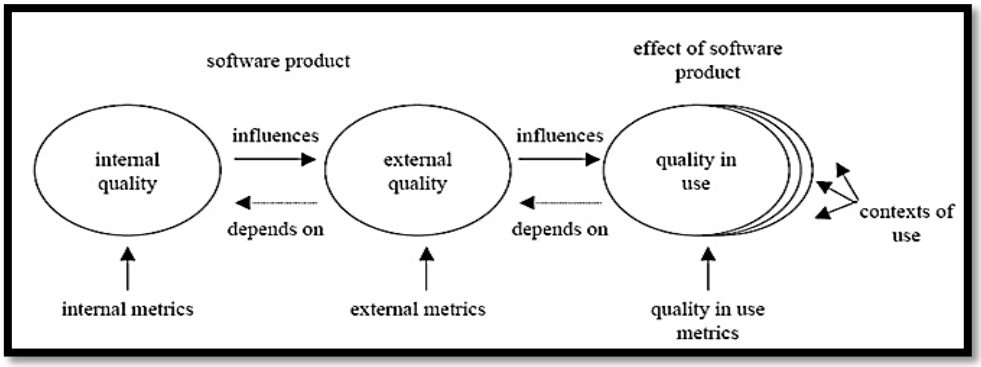
- المرونة (Flexibility): السهولة التي يمكن تعديل مكون نظام أو للاستخدام في التطبيقات أو بيئات أخرى غير تلك التي تم تصميمها خصيصا.
- القدرة على الفهم (Understandability).
- العمل مع أنظمة أخرى (Interoperability).
- نمط البرمجة وجودة وكمية التعليقات وتعقيد برنامج المنتج.
- الحجم: بزيادة الحجم (عدد الأسطر) يزداد التعقيد؛
- الإشارة إلى الأجزاء من البرمجية التي هي صعبة الفهم؛
- الاعتمادية: الإشارة إلى الأجزاء من البرمجية التي لها اعتماد كبير على أجزاء أخرى؛
- التشابه: تكرار أقل ما يمكن لعدم زيادة عدد الأسطر؛
- التعقيد: حسب قانون عدد الحلقات المستقلة (المغلقة) ضمن Control Graph وهو عبارة عن طريقة قياس التعقيد للتطبيق من الداخل.

- **جودة الاستخدام Quality in Use**

تقيس فيما إذا كان المنتج البرمجي يحقق متطلبات المستخدم في بيئة العمل وتشمل أربع خصائص

- الفعاليّة : Effectiveness؛
- الإنتاجيّة: Productivity؛
- الأمان: Safety؛
- رضاء المستخدم: Satisfaction.

والعلاقة بين هذه الأنواع موضحة بالشكل التالي.



الشكل (9-4) العلاقة بين أنواع الجودة حسب معيار ISO

4.9. تطور إدارة فكر الجودة

إن استخدام فكر الجودة قديم وتمتد أصوله في عمق تاريخ الحضارة الإنسانية. فأي عملية مقارنة لاختيار الأفضل لمتطلبات الحياة الأساسية أو تهيئتها أو تطويرها أو تنميتها يكون لمفهوم الجودة موقع أساسي فيها. فالجودة فكراً استراتيجياً مؤثراً في حياة المجتمعات وحاجاتها الأساسية. لذلك اهتم الكثير من الباحثين بإدارة فكر الجودة وتطويره. والجودة نظام تراكم من الخطط و الأنشطة والأحداث التي ينبغي توفيرها لضمان تحقيق المنتج أو العملية أو الخدمة للاحتياجات المتوقعة منها [3]. إنّ تطوير فكر الجودة يمكن أن نميّز أربعة مراحل هي:

1. فحص الجودة(Quality Inspection)

الفحص يستند ببساطة على اختبار خاصية أو اكثر للمنتج أو الخدمة أو النشاط وقياسها بالمقارنة مع مجموعة من المتطلبات باعتبارها معايير لتخمين المطابقة (Conformity).

2. ضبط الجودة (Quality Control)

ضبط الجودة هي خطوة متقدمة عن فحص الجودة ومطورة عنها. ففي ضبط الجودة تستخدم بعض الطرق والآليات والأدوات التي قد يكتنفها بعض التعقيد لإدارة الجودة. ففي نظم ضبط الجودة نتوقع الحاجة لبعض الاستعدادات مع تهيئة إجراءات الضبط و فحص بعض مراحل المنتجات الوسطية (Intermediate Products) في المنتجات البرمجية: مثل تعريف المتطلبات ومواصفاتها و التصميم و البرمجة وهكذا. إضافة لتهيئة البيانات والمعايير التي نحتاجها في فحص ومعالجة الأداء وغيرها مما له علاقة بإدارة الجودة وضبطها.

وإنّ ضبط الجودة لا تعني تطوير الجودة وتحسينها وإنما فقط تشخيص أو إنارة (Highlight) مواطن عدم تطابق أو انسجام المنتج أو الخدمة مع المتطلبات [85]. وأحياناً قد يعجز توجه

ضبط الجودة عن تحديد سبب عدم التطابق أو الانسجام. وضبط الجودة هو ضمان إتباع فريق العمل للإجراءات وللمعايير. فضبط الجودة تعني تنفيذ عدد من الإجراءات وإعداد عدد من التقارير خلال عملية تنمية البرمجيات.

3. ضمان الجودة Quality Assurance

توجه ضمان الجودة ينطلق من حقيقة أن الجودة ليست مسؤولية شخص واحد في المؤسسة، بل هي مسؤولية كل من له علاقة، مباشرة أو غير مباشرة، بعملية الإنتاج أو أداء الخدمة. لذلك فلابد أن يكون للعملية الإدارية دور واضح و مميز ضمن فكر الجودة. وإن الجودة تحتاج إلى نظام يضمن تطبيق جميع الإجراءات التي تم تنفيذها أو تخطيطها، وهذا بدقة دور وغرض ضمان الجودة. لذلك ينطلق ضمان الجودة من كون تشخيص مَواطن عدم التطابق مع المتطلبات ومعالجتها أسلوب غير فعال في إزالة الخلل الأساس المسبب للمشكلة [86]. ويكمن حل المشكلة في توجيه المؤسسات المعنية لجهودها في التخطيط لمنع حدوث عدم التطابق. وهذا التوجه يقودنا إلى توجه جديد في فكر الجودة أطلقنا عليه "ضمان الجودة". لذلك نجد أنه من المهم في خطط تنمية و إنتاج البرمجيات وخاصة في المشاريع الكبيرة ولابد من تخصيص خطة مفصلة للجودة ضمن الأبواب التفصيلية للخطة الشاملة. وضمان الجودة يقتضي تأسيس إجراءات مؤسسية ومعايير تقود إلى جودة عالية. ويتضمن ضمان الجودة بالتعريف كيفية توجه المؤسسة لإنجاز الجودة. فضمان الجودة يتضمن تعريف أو اختيار المعايير التي يستوجب تطبيقها على المنتجات البرمجية أو عملية تنميتها. لذلك يمكن تعريف "ضمان الجودة" بكونها جميع الأفعال المخططة (Planned) و الآلية (Systematic) الضرورية لتوفير الثقة بكون المنتج أو الخدمة سوف تحقق الحاجات الفعلية. فضمان الجودة، على سبيل المثال، في مرحلة تصميم المنتج يكون بمراجعة إجراءات التصميم وربما التدقيق لتحديد نوع المعلومات التي يجب توفرها في قسم التسويق لاستخدامها في تصميم المنتج.

3. الإدارة الشاملة للجودة (TQM)

يمكن اعتباره مرحلة متقدمة من مفهوم ضمان الجودة بدأ يتبلور و ينمو ليصبح مستقبلاً فكراً مستقلاً متميزاً. فالجودة كما وجدنا فيما تقدم مفهوماً متعدد الأبعاد يتأثر بعوامل و متغيرات كثيرة. وتحقيق الجودة للبرمجيات أسوةً بغيرها من المنتجات يحتاج إلى ربط بين حاجات المستفيد وتوقعاته وحاجات المنتج وما يساهم في تحقيق قوى تنافسية له في سوق البرمجيات. لذلك فإدارة الجودة تحتاج إلى شمول جميع العوامل والفعاليات والضوابط التي تساهم بشكل مباشر أو غير مباشر في المنتج (منتج برمجي أو سواه).

1.4.9. المرتكزات الأساسية لضمان الجودة حسب Pressman

تمت الإشارة إلى أن فكر ضمان الجودة يرتكز على [3]:

- ✓ توجه إدارة الجودة؛
- ✓ التقنيات الفاعلة لهندسة البرمجيات (طرق وأدوات)؛
- ✓ الأساليب الشكلية (Formal) للمراجعة التي تطبق خلال عملية تنمية البرمجيات؛
- ✓ استراتيجيات الفحص؛
- ✓ السيطرة على وثائق البرمجيات و تحديثها؛
- ✓ إجراءات ضمان التطويع (Compliance) لمعايير تنمية البرمجيات؛
- ✓ آليات القياس وإعداد التقارير؛
- ✓ الإدارة الشاملة للجودة (Total Quality Management TQM).

2.4.9. المعالم الأساسية لنظام الإدارة الشاملة TQM

يمكن أن نوجز العناصر الأساسية لنظام الإدارة الشاملة للجودة بأربعة معالم رئيسية هي [3]:

- التركيز على المستفيد وتحقيق متطلباته

نهدف إلى تحقيق الرضا الشامل للزبون، من خلال دراسة حاجاته ورغباته و توقعاته وتجميع متطلباته وقياس وإدارة رضاه، وتعرف هذه العملية بـ (Custom Focus).

- عملية الإنتاج (Process)

ويهدف إلى تقليص متغيرات عملية الإنتاج (Process Variation) وتحقيق استمرار تطويرها، وتتضمن العملية المهنية وتنمية المنتج وتحسين جودته.

- الجانب البشري للجودة (Human Side of Quality)

ويهدف إلى خلق تراث واسع للجودة في المؤسسة الإنتاجية، وجوهر ذلك يتضمن القيادة (Leadership) وجماعة الإدارة (Management Community) والمشاركة الشاملة (Total Participation) وتفويض العاملين (Employee Empowerment) وبقية العوامل الاجتماعية والنفسية والبشرية.

- التحليل والقياس (Measurement and Analysis)

ويهدف إلى قيادة التطوير بشكل مستمر في جميع معالم الجودة بنظم قياس هدفيه.

وبالتالي فالمؤسسة التي تمارس إدارة الجودة الشاملة يجب أن يكون لها إدارة تنفيذية ترتكز على بنية تحتية (Infrastructure) وتدريب و ثقافة ولها خطة جودة استراتيجية.

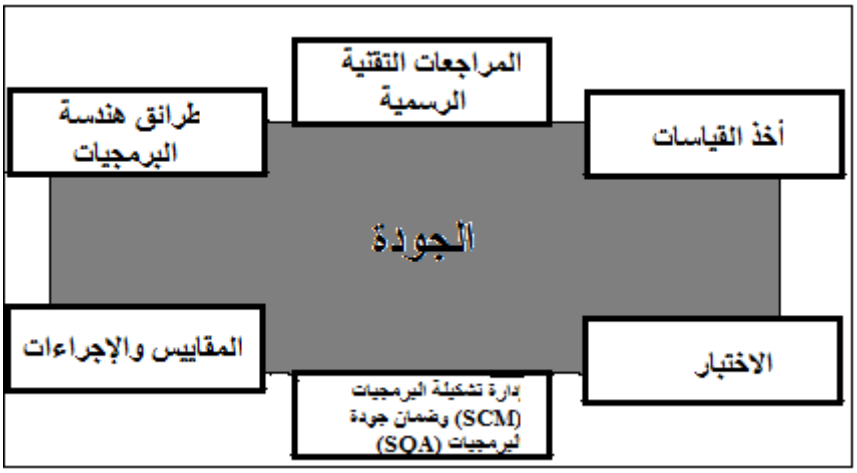
وإنّ الفعاليات اللازمة لتحقيق جودة البرمجيات هي مجموعة من المكونات، هي كالتالي:

طرائق هندسة البرمجيات، طرائق التحليل والتصميم والبناء، المراجعات التقنية الرسمية، المقاييس والإجراءات، الاختبار، وإجرائية SQA.

إذ أنّ طرائق هندسة البرمجيات تعطي الأساس الذي تبنى عليه الجودة، وتقوم طرائق التحليل والتصميم والبناء بتحسين الجودة عن طريق توفير تقنيات نظامية ونتائج متوقعة، تساعد المراجعات التقنية الرسمية (المسح السريع) على ضمان جودة منتجات العمل، التي تنتج عن كل مرحلة من مراحل هندسة البرمجيات.

خلال هذه الإجرائية يخضع كل عنصر من تشكيلة البرمجيات للقياسات والضبط، وتساعد المقاييس والإجراءات على ضمان الانتظام، وتعزز إجرائية SQA الرسمية "فلسفة الجودة الكاملة".

والشكل التالي يوضح مجموعة المكونات.



الشكل (9-5) يبيّن أساليب تحقيق جودة البرمجيات

5.9. المقاييس التي تستخدم لقياس تعقيد المنتج البرمجي

تهتم قياسات البرمجيات بتوفير قيمة رقمية لسمة من سمات منتج البرمجيات أو عملية، يسمح هذا بالمقارنة الموضوعية بين التقنيات والعمليات، فالقياس (Measurement) يمكن تعريفه بعملية اشتقاق أعداد أو رموز مرافقة لخصائص الأشياء الحقيقية لتسهيل شرحها وفق معايير واضحة تمكننا من التعبير عن الأشياء وخواصها بأرقام أو رموز ذات دلالة. والمقاييس هي مؤشرات دالة (Indicators) لخصائص الأشياء كمياً و يستوجب تحديد قيم واضحة ومحددة لها نتيجةً لعملية

القياس [3]، فالقياس يساعدنا في اتخاذ القرار الصائب واختيار البديل الأفضل، والمقاييس الكمية تساعد في توحيد القرار، و توحيد القرار في تصنيع البرمجيات يعني الحصول على منتجات برمجية عالية الجودة بأقل كلفة وجهد يسهل إدامتها وتنفيذها وتطويرها وفق المستجدات المتجددة للمستفيد وبالتالي الحصول على رضاء المستفيد وهو من أهم مقاييس الجودة للمنتج (Product) يسعى المصنع لتحقيقها. وكذلك يسمح التقييس بالتعبير الكمي عن البرمجيات وتقييس عمليات البرمجيات أو المنتج وقد تستخدم لتوقع سمات المنتج او التحكم في عملية البرمجيات.

وهناك العديد من المقاييس البرمجيّة للجودة، منها:

1. عدد الأشخاص والأيام المطلوبين لتطوير مكون؛
2. مقياس جودة البرمجيات من حيث المدى :

- **الزمن القصير (على المدى القريب):** هل البرمجية تحقق احتياجات الزبون الحالية. هل هذه البرمجية فعالة بشكل كاف من ناحية المعطيات الموجودة لدينا.
- **الزمن الطويل (على المدى البعيد):** هل البرمجية قابلة للإصلاح في حال ظهور أعطال، هل ممكن إضافة أشياء عليها في حال أراد المستخدم مستقبلاً الإضافة.

3. مقياس يطلق عليه (Line of Code (LOC)): حساب عدد أسطر الكود باعتبار حجم البرنامج أفضل مؤشر لتعقيده ولكن أثبتت الدراسات قصور هذا المقياس في العديد من المواضع، ومن عيوبه:

☒ عدم تناغم (Consistent) هذا المقياس مع لغات البرمجة والتطبيقات والمطورين، وقيمته تتأثر بهذه العوامل وليست موحدة.

☒ تعقيد البرنامج لا ينعكس من خلال هذا المقياس. فليس دائماً البرنامج الأصغر أقل تعقيداً والتعقيد قد يتسبب من جوانب أخرى كما سنرى لاحقاً، لذلك هذا المقياس لا يعتبر مؤشر جيد لتقويم الجودة.

1.5.9. الخصائص المميزة والمؤثرة في بناء المقاييس غرضية التوجه

1) التوجه المحلي (Localization) : التوجه المحلي، هو خاصية البرمجيات في التعبير والإشارة (Indicate) لسلوك المعلومات للتمركز ضمن البرنامج بينما كان تمركز المعلومات حول الوظائف في التوجه الوظيفي .

2) الاحتواء (Encapsulation): به امتلاك الكائنات لذاتها [83]، التي أطلقنا عليها تجريد البيانات (ADT) لذلك ولتقويم جودة البرمجيات الغرضية نحتاج لمقاييس تختبر احتواء الأصناف

على البيانات والإجراءات الضرورية والكافية لتمثيل الكائن الذي ينشأ بفعلها، تغير اهتمام وتركيز مبدأ القياس من الوحدة البرمجية إلى قياس رزمة من البيانات والإجراءات وتشجيع القياس كي يكون من مستوٍ عالٍ من التجريد: مثل قياس عدد العمليات (أي الوظائف) النافذة للبيانات المعرفة لصنف ما.

3) إخفاء المعلومات (Information Hiding): اخفاء التفاصيل الإجرائية لمكونات البرنامج والسماح فقط لما هو ضروري للتواصل مع أجزاء أخرى.
4) الوراثة (Inheritance): خاصية الوراثة تمنح القدرة على اشتقاق صنف جديد بإكسابه صفات و سلوك صنف أو أكثر(كلاً أو جزءاً) موجودة فعلاً. وهذه القابلية ممكن استثمارها في مختلف المستويات للفئات المنتظمة ضمن تشكيل هرمي بفعل خاصية الوراثة. وأثارت هذه الخاصية اهتمام الباحثين لبناء مقاييس لتقويمها.

ومن أمثلة هذه المقاييس: عدد الأصناف الأساس وعدد الأصناف المشتقة وعمق الاشتقاق في شجرة الوراثة.

5) أساليب تجريد الأغراض (Object Abstraction Technique): أسلوب التجريد آلية تتيح للمصمم التركيز على التفاصيل الأساس الضرورية لأجزاء البرامج والاستغناء عن الإسهاب والتفصيل. وكما أشار الباحثون "التجريد فكرة نسبية". فكلما ارتقينا إلى مستوى أعلى من التجريد، تجاهلنا التفاصيل شيئا فشيئاً. وكلما انحدرنا إلى المستويات الدنيا، حددنا تفاصيل الفكرة أو المبدأ بشكل أدق.

6.9. اختبارات تحقيق الجودة

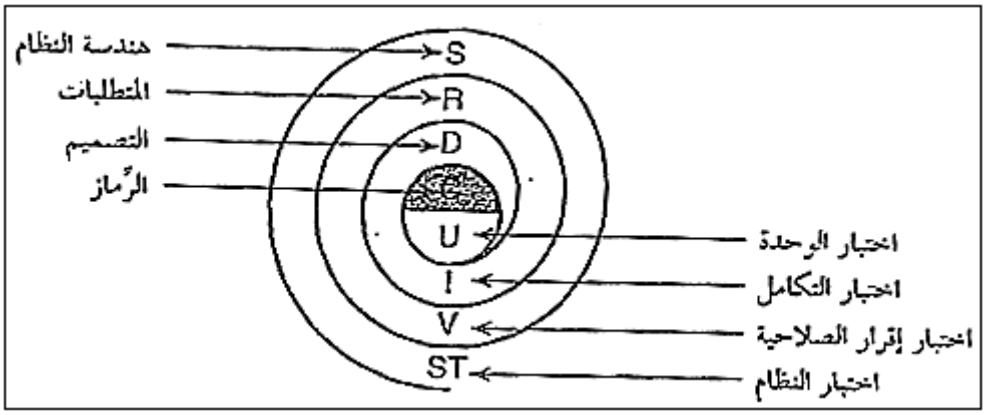
يُربَط اختبار البرمجيات بضمان الجودة بشكل وثيق إذ يمكن الاعتبار بأنّ الدافع الذي يحرّك اختبار البرنامج هو إثبات جودة البرمجيات بواسطة طرائق يمكن تطبيقها اقتصادياً وبفعالية، على الأنظمة الواسعة والضيقة النطاق على السواء [3].

وإنّ الاختبارات تقسم بشكل واضح إلى قسمين أساسيين:

- اختبارات التأكد من صحة العمل ويقوم بها مهندس البرمجيات أثناء بناء البرنامج وتكون هي اختبارات الصندوقين الأبيض والأسود.
- اختبارات تحقيق الجودة ويقوم بها الأشخاص العاديين أو المهندسيين الذين ليس لديهم دراية كاملة بالرماز والتصميم وهي اختبارات من نوع الصندوق الأسود.

إذ يمكن أن نرى إجرائية اختبار البرمجيات كالحلزون، يقوم بالبداية مهندس النظم بتعريف دور البرمجيات، ثم ينتقل إلى تحليل متطلبات البرمجيات، فيعرّف نطاق المعلومات، ووظيفة وسلوك وأداء وقيود ومعايير صلاحية البرمجيات، إذا اتجهنا نحو داخل الحلزون، فسوف نأتي على التصميم وننتهي بالرماز.

يبين الشكل التالي الحلزون البرمجي مع ما يناسبه من اختبارات



الشكل (9-6) استراتيجيّة الاختبار

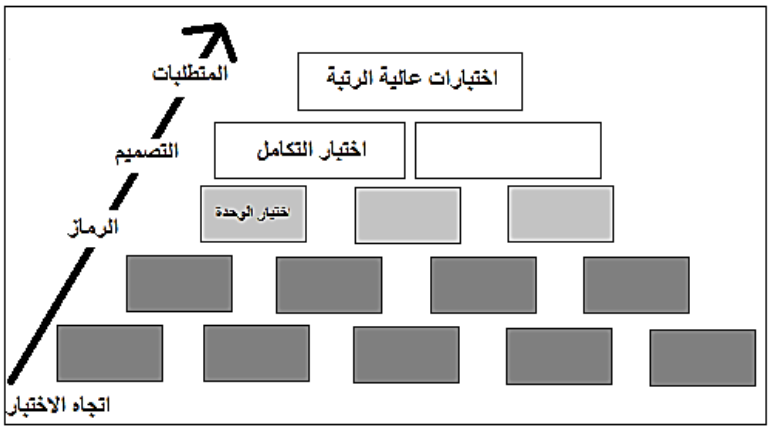
وبالتالي يمكن النظر إلى استراتيجية اختبار البرمجيات ضمن السياق الحلزوني، يبدأ باختبار الوحدة عند مركز الحلزون، ويركّز الاهتمام في كل وحدة برمجية طبقاً للتنجيز في الرماز المصدري، تتقدم الاختبارات بالتحرك على طول الحلزون خارجاً، نحو اختبار التكامل، إذ يركز في تصميم البنيان البرمجي وإنشائه، وبالدوران دورة أخرى خارجاً بالحلزون، نصادف اختبار إقرار الصلاحية، حيث يجري إقرار صلاحية المتطلبات للبرمجيّة التي أُنشأت، وأخيراً نصل إلى اختبار النظام حيث تُختبر البرمجيّة وعناصر النظام الأخرى دفعة واحدة.

ويمكن تقسيم الاختبارات إلى حسب السوية أربع مراحل تنجز بشكل متتالي:

- **اختبار الوحدة:** وهو اختبار منخفض الرتبة يجري على الرماز يستعمل اختبارات الصندوق الأبيض بكثافة.
- **اختبار التكامل:** وهو اختبار متوسط الرتبة يجري على التصميم ويستعمل اختبارات الصندوق الأسود بكثافة.
- **اختبار إقرار الصلاحية:** وهو اختبار عالي الرتبة يجري على البرمجية لمعرفة مدى توافقها مع المتطلبات وتستعمل اختبارات الصندوق الأسود حصراً.

- **اختبار النظام:** وهو اختبار عالي الرتبة يجري للتحقق من أنّ جميع العناصر(الأشخاص، العتاديات، قواعد المعطيات) تتشابك كما ينبغي وأنّ وظيفة وأداء النظام الكاملة منجزة.

الشكل التالي يبيّن مراحل الاختبار حسب الرتبة.



الشكل (9-7) يبيّن مراحل الاختبار حسب الرتبة

7.9. توجهات بناء مقاييس جودة البرمجيات

اهتم الفكر الهندسي للبرمجيات بدراسة القياس والمقاييس باعتباره الركن الأساس الذي ترتكز عليه صناعة البرمجيات. وتميز بشكل خاص العقد الأخير من الألفية الثانية ومنطلق الألفية الراهنة بهذا المحور الفاعل. وظهرت عبر هذه المرحلة توجهات فكرية جادة وناجحة في بناء المقاييس لجودة المنتج البرمجي. واهتمت هذه التوجهات بمجملها كل في جانب من جوانب الجودة وأهملت جوانب بالرغم من كون العديد منها أثبت نجاحه في الاستدلال على الجودة ودعم الصناعة. وهكذا تبرز الحاجة للفكر الإحصائي وبحوث العمليات لبناء مقاييس ذات استدلال أوسع تجمع دلالات عدد من المؤشرات الكمية (مقاييس مطلقة) لسمات متباينة للجودة. لذلك سنحاول ضمن هذه الفقرة التطرق إلى أكثر التوجهات شيوعاً في قياس جودة المنتج البرمجي عبر مسيرة تطورها.

المحاولات الأولى لبناء مقاييس الجودة للبرمجيات توجت نحو موثوقية (Reliability) المنتج البرمجي. والسبب هو وضوح هذه السمة وسهولة قياسها باعتبارها تتعلق باحتمالية الفشل الوظيفي والعلل التي تحدث في النظام البرمجي خلال تشغيله الفعلي لمدة طويلة. ولكن ماذا حول الجوانب الأخرى التي لا تقل أهمية في تحديد مفهوم الجودة للمنتج: مثل سهولة خدمات ما بعد البيع وانخفاض تكاليفها وسهولة تشغيله وتعلمه وغيرها. لذلك ظهر توجه حثيث نحو بناء مقاييس جادة لتعقيد المنتج البرمجي. والسبب في ذلك أن التعقيد يؤثر على معظم السمات الأخرى للجودة [3]. فالمنتج المرتفع التعقيد غالباً ما يكون صعب الاستخدام والتشغيل وموثوقيته منخفضة ومكلف في خدمات ما بعد البيع وقابليته للتطور

استجابة للتغيرات المتوقعة في حاجات الزبون الذي يفرضه الواقع المتطور للمجتمعات المعاصرة. ولكن ما المقصود بالتعقيد ؟. والتعقيد كما نستخدمه في لغتنا اليومية مفهوم مبهم شأنه شأن الجودة ذاتها.

8.9. أهداف الجودة في الهندسة المقادة بالنماذج MDE

جودة النماذج تتأثر بالعوامل التالية:

- ✓ جودة لغة النمذجة، الأدوات، عمليات النمذجة؛
- ✓ المعرفة المكتسبة وخبرة المنمذج؛
- ✓ تقنيات ضمان الجودة المطبقة.

ما يهمنا من مواصفات الـ MDE عند التحدث عن تحديد إطار الجودة هو [84]:

- استخدام النماذج في مراحل متعددة من عملية التطوير البرمجي: تستخدم النماذج منذ مراحل التطوير المبكرة حتى عملية الاختبار والمحاكاة وتوليد الكود، غالباً ما تكون النماذج غير مكتملة، وغير دقيقة أو منسجمة في بداية دورة حياة التطوير البرمجي وتدريجياً تحقق الاكتمالية والدقة، ومن الممكن أن تكون قابلة للتنفيذ أو غير قابلة للتنفيذ (حتى نماذج التحليل قد تكون قابلة للتنفيذ).
- النماذج على مستويات مختلفة من التجريد: العلاقات بين النماذج مهمة وذلك عند مرحلة التقييم لهذه النماذج من أجل تحقيق مواصفات جودة إضافية. مثال، النماذج المكررة لديها صفوف وتوابع كثيرة التي تزيد من تعقيد الخوارزميات.
- النماذج من وجهات نظر مختلفة: مثل النماذج الهيكلية مقابل النماذج السلوكية.
- توليد الكود من النماذج: هذا يعني أن تحقيق النماذج ذات أهمية أكبر في الـ MDE منها في تطوير البرمجيات التقليدي حيث توليد الكود يكون لتحقيق الجودة.
- تمثل المهام في النماذج بواسطة أدوات: تخضع النماذج للعديد من التحويلات والتدقيق.

9.9. الاستنتاج

1.9.9. حول تعريف فكر الجودة

رغم التطور الحاصل في فكر الجودة لكنه لا يزال مبهم، ورغم الإبهام الذي يكتنفه فهو مهم ولا يمكن تجاهله، لذلك فمسيرة تطوره مستمرة والحاجة لجهود حثيثة ودراسات جدية تزداد. وهذا بلا شك هو السبب الرئيسي في استمرار ظهور أدوات جديدة وتوجهات في حقل إدارة الجودة ومقاييس كثيرة وأساليب متعددة في القياس.

استناداً للعديد من البحوث التي تناولت الفكر الإداري للجودة فإن مفهوم إدارة الجودة الشاملة لا يزال في مرحلة الطفولة. كما تناولته الكثير من البحوث من ضمن تراث فكر ضمان الجودة واستخدمته العديد من الشركات بمسميات أخرى: مثل ضبط الجودة الشاملة.
فالإدارة الشاملة للجودة تتمحور حول ثلاثة مواضيع أو محاور رئيسية: المستفيد، العملية، والأفراد. إذ أنّ الهدف الأساسي رضاء المستفيد وتحقيق احتياجاته وتوقعاته، والعملية باعتبارها النظام الذي يؤدي إلى إنتاج المنتج، والأفراد هم الذين يساهمون بشكل مباشر أو غير مباشر في العملية الإنتاجية .

لو ناقشنا هذا الفكر الإداري لوجدناه ينطبق بشكل فاعل على إدارة جودة المنتج البرمجي، وإن السياسة العليا للمؤسسة الإنتاجية التي تحتل قلب هذه العملية الإدارية، برؤيتها وأهدافها والتزاماتها أو ثوابتها الإدارية، تقوم بربط المحاور الثلاثة (المستفيد، الأفراد، والعملية الإنتاجية) المؤثرة بشكل فاعل على جودة البرمجيات المنتجة. فالمستفيد له احتياجات و توقعات. ورضاء المستفيد يتحقق بشكل أساسي بتلبية حاجاته. أما تلبية توقعات المستفيد فليس بالضرورة أن تكون جميعها من ضمن حاجاته. وقد يكون جزء من التوقعات ما يضيف للمستخدم أعباء مادية إضافية قد لا تنال رضاه في حال كونها لا تمثل جزءاً من حاجاته. أما بخصوص الأفراد فتطوير قنوات التواصل بين أفراد كل فريق عمل وبين فرق العمل المختلفة للمنتجات الوسطية لعملية تنمية وتطوير المنتج البرمجي (مواصفات المتطلبات و المواصفات الفنية للتصميم وغيرها) وتصويب منح الصلاحيات، مما لاشك أنّ له أثر فاعل على جودة المنتج البرمجي النهائي. أما العملية الإنتاجية فتتأثر بشكل فاعل بالتغذية الراجعة من قبل البائعين (Venders) للمنتج البرمجي وسلوكهم وخبرتهم. لذلك فالتكامل مع البائعين يسهم في تطوير العملية الإنتاجية [87]. كذلك قد يكون للعملية الإنتاجية علاقة مع بعض المجهزين: قد تكون المؤسسة مرتبطة بعقد لتنمية برمجيات إدارة مواد لمؤسسة صناعية يتم تنفيذه على شبكة من الحواسيب ويرتبط مشروعها بشراء برمجيات إدارة الشبكة من قبل مجهز آخر. فالتكامل مع المجهزين يساهم بلا شك في إذكاء إدارة جودة منتجها. كذلك عمليات التحليل وتطويرها المستمر ودعمه بالآليات والأدوات البرمجية المطورة وتطوير قابليات التوجيه الذاتي لفرق العمل وتعدد مهاراته جزء فاعل في العملية الإدارية المدروسة. وتراث المؤسسة الناتج من تراكم الخبرة وتنميتها في إدارة الجودة و العملية الإدارية ككل وبناء المقاييس و وضع المعايير ونماذج تقويم نضج العملية الإنتاجية وغيره مما يساهم في تطوير فعاليات المؤسسة يحتاج إلى العناية به والحرص عليه ليساهم بدور فاعل في دعم خبرات الأفراد و إذكاء العملية الإنتاجية.
ويمكن تلخيص المرتكزات الفكرية و الفلسفية لإدارة الجودة الشاملة بما يلي:

- ✓ الجودة كما يراها المستفيد (المستهلك).

- ✓ القيادة.
- ✓ المشاركة وتطوير المستفيد.
- ✓ الاستجابة السريعة.
- ✓ تصميم الجودة والوقاية.
- ✓ التطوير بالمشاركة.

2.9.9. الجودة والمنهجيّة المُقترحة

سنعمل على تجاوز نموذج Halsted فهو يعتمد على الرماز في تحديد الجودة وعلى عدد العناصر، وهذا لا يتناسب مع طبيعة المشروع، إذ لابدّ لنا من البحث عن نماذج ومقاييس تقييم التصميم وجودته بدلاً من نماذج تقييم الرماز، **فنحن بصدد تقييم التصميم للنماذج المترفعة الذي يولّد الأدوات لا الرماز الناتج عنه**.

و**تتطابق المنهجيّة المُقترحة** مع قسم واسع من **عوامل FURPS ومقاييس الجودة حسب McCall** يبينها الجدول الآتي.

الجدول (9-3) مقاييس الجودة وعواملها التي تحققها المنهجيّة المُقترحة

مقاييس الجودة									
التشغيلية البينية	إعادة الاستخدامية	المحمولية	الاختبارية	الاستخدامية	الفعالية	الموثوقية	الصحة		عوامل الجودة
						✓		الدقة	
✓								ألفة الاتصال	
✓							✓	الاكتمال	
✓								ألفة المعطيات	
						✓		تحملية	

	الخطأ								
	فعالية التنفيذ			✓					
	التشغيلية			✓	✓				
	استقلال البرمجيات عن النظام						✓	✓	
	التعقبية	✓							
	التدريب								✓

هذا وإنّ **<u>المنهجيّة المُقترحة تستكمل عناصر الجودة حسب Kitchen Ham</u>** والاسقاط التالي يوضح مكان الجودة من المشروع.

1- **موضع الاهتمام:** يتجه تركيزنا إلى المستخدمين الذين يستعملون المنتج النهائي وبالتالي الجودة ستكون من موضع اهتمام المستخدمين الذين يستعملون التصميم النهائي ويخلقوا منها أدوات.

2- **سمات الجودة:** وهنا نعود إلى عواملها ومقاييسها ويمكن الإضافة على ماسبق خبرة المستخدم (وهنا خبرة المستخدمين لبرنامجنا واسعة كون الاختبارات أجريت في كلية الهندسة المعلوماتية في جامعة دمشق للطلاب ما قبل التخرج من السنة الأخيرة، وبالتالي اختباراتهم وتقييماتهم تعتبر مهمة جداً كون أفكارهم لم تأتي عبثاً بل عن دراسات مشابهة وأدوات متقاربة).

3- **وجهة النظر:** وقد تختلف من شخص لآخر حسب الاهتمام وهنا تجاوزنا وتلافينا هذه النقطة من خلال حصر الاهتمام بعدة نقاط (الزمن، قابلية الفهم، سهولة الاستخدام) وقمنا بتقييم (جودة المنتج ، جودة العملية، جودة الخدمات) وفقاً لهذه النقط.

<u>ومنهجيتنا تحقق الجودة حسب Garvin</u>

- **جمالية واجهاته (شاشاته)** و تنظيم مخرجاته وسهولة التحاور معه.
- **حسن تنظيم وثائقه وسهولة متابعته** و مقروئيته (Readability).
- **مدى استخدامه للدوال القياسية وتماسك (Cohesive) وحداته البرمجية ودرجة دقة نتائجه.**

بالإضافة لذلك فهي تتطابق مع المعايير العالمية الدولية (ISO)، ويبدو واضحاً وفق الجدول التالي.

الجدول (9-4) مطابقة المنهجيّة لعناصر الجودة حسب معيار ISO

عنصر الجودة	الشرح
القوة أو الشدة	عدم حدوث انهيارات أثناء استخدام النماذج المترفعة بعد التقسيم، تمّ اختبار المنهجيّة من قبل 90 طالب عدة مرات ولم يحدث أي انهيار أو خلل.
الوضوح وقابلية الفهم	لا حاجة للملفات المساعدة والتوثيق، انخفضت نسبة استخدامهم بشكل كبير بعد عملية التقسيم.
سهولة الاستخدام	انخفض عدد الضغطات الوسطي لإنشاء غرض بعد تقسيم النماذج المترفعة إلى ما يقل عن 6 ضغطات.
إمكانية إعادة الاستخدام	وهذا الشرط محقق كوننا نعمل في عالم النماذج المترفعة وهي موجهة حتماً لإعادة الاستخدام في السويات الأدنى منها.
اختصار الزمن	انخفض الزمن المطلوب لفهم النموذج واستعماله إلى أقل من ساعتين وسطياً بعد عملية التقسيم.

الجزء الثالث

النموذج المُقترح وتمديده ونتائجه

- ❖ المنهجيّة المُقترحة بهدف تجاوز العيوب
- ❖ تمديد المنهجيّة لتشمل البعد المنطقي
- ❖ النتائج والتقييم

الفصل العاشر
المنهجيّة المُقترحة

1.10. مقدمة

من خلال الدراسات السابقة لعيوب التصميم (الصف الكبير، عيب العنصر، عيب الوراثة، عيب التكرار، عيب عدم الاستخدام وعيوب العلاقات الحلقيّة) نجد أنّ مشكلتي الحجم الكبير وضعف الترابط تتصدر هذه المشكلات وتسببها، وكل الأعمال السابقة كانت تعمل على تلافي هاتين المشكلتين في مستوى التصميم (المستوى المنخفض)، لذا رأينا أننا لكي نضبط هذه العيوب ونمنعها من الوجود في المستويات المنخفضة لابدّ من ضبط الأمر في المستويات المرتفعة (النماذج والنماذج المترفعة).

فالنماذج المترفعة هي المسؤولة عن توليد النماذج (الأدوات) والنماذج هي المسؤولة عن إنتاج البرمجيات. وبذلك نضمن انتاج برمجيات خالية من عيوب الحجم الكبير وضعف الترابط.

2.10. تمديد مفهوم الحجم الكبير وضعف الترابط إلى السويات العُليا

إنّ عوامل الحجم والتماسك تتمثل في كل مستوى بشكل مختلف عن الآخر، فهي:

في السويات المنخفضة (سويات الرماز والتصميم):

- الحجم الكبير يعني عدداً كبيراً من العناصر والتوابع.
- الترابط الضعيف يعني التفكك ما بين توابع الصف الواحد، وبالتالي فالتقسيم مطلوب.

في السويات الأعلى (سويات النماذج) وسّعنا المفاهيم السابقة لتصبح كالتالي:

- الحجم الكبير يعني عدداً كبيراً من الصفوف.
- الترابط الضعيف يعني التفكك ما بين صفوف النموذج وقلّة الروابط التي تربط بينها، وبالتالي فالتقسيم مطلوب.

في السويات العُليا (سويات النماذج المترفّعة) وسّعنا المفاهيم السابقة لتصبح كالتالي:

- الحجم الكبير يعني عدداً كبيراً من المفاهيم.
- الترابط الضعيف يعني التفكك ما بين مفاهيم النماذج المترفّعة، وقلّة الروابط بينها، وبالتالي فالتقسيم مطلوب.

وبالتالي يتوجّب علينا قسم النموذج المترفّع الكبير والضعيف الترابط ما بين مفاهيمه إلى عدد من النماذج المترفّعة الجزئية، أصغر حجماً، وأعلى تماسكاً، وأكثر تخصصاً.

وطالما أجرينا عمليات اكتشاف العيوب وتصحيحها في السوية العُليا (النماذج المترفّعة)، فلا داعي لاكتشافها وتصحيحها في السويات الدنيا (سويات النماذج والتصميم والرماز)، إذ ستكون هذه السويات قيد التحكم من قبل السوية العُليا.

بهذه الطريقة، سيجمع النموذج المترفّع عدداً من النماذج المترفّعة الجزئية بشكل مشابه للرزم، ونضمن بالتالي أدواتاً مولّدة أكثر تخصصاً، وأكثر مرونة، وأكثر قابلية للفهم.

استخدمنا العديد من النماذج المترفّعة كحالات دراسيّة بهدف جمع المعلومات حول دراستنا في محاولة لوضع منهج لتقسيم النماذج المترفعة إلى عدة نماذج مترفّعة جزئية.

لا يمكننا وضع منهج لتقسيم النماذج المترفّعة إلى عدة نماذج مترفّعة جزئية بدون وجود بيانات تجريبيّة، وهذا ما تقوم عليه أغلب الدراسات المشابهة، إذ تعتمد بيانات تجريبيّة [2,89,5,90].

هذا المشروع يقدم مجموعة من نتائج الدراسة التجريبيّة البحثيّة حول تقسيم وفصل النماذج المترفعة، وقد استمرت لمدة ستة أشهر. كنا قد طرحنا عدة نماذج مترفعة كحالات دراسية ليتم تقسيمها إلى نماذج مترفعة جزئية من قبل 100 طالباً بشكل يدوي. النماذج الجزئية تحقق التجميع الأنسب للمفاهيم والذي بدوره يضمن سهولة الاستخدام.

تمّ رفدنا بوجهات نظر الطلاب حول التقسيم من خلال استبيان /1/. الاستبيان موجود في الملحق[2]. وانطلاقاً من هذه البيانات التجريبيّة وفي محاولة لنقل هذه التجارب من المنهج اليدوي إلى المنهج التلقائي (الأوتوماتيكي) قمنا باقتراح منهجيّة للتقسيم الأنسب للنماذج المترفعة كبيرة الحجم، ضعيفة الترابط، وذلك بما يضمن الحجم الأصغر، والترابط الأعلى، والتخصيص الأكبر.

3.10. تحليل الدراسة التجريبيّة

أوضحت الأشكال الموجودة في الملحق طريقة التقسيم التي تمّ اعتمادها من قبل أغلب الطلاب، ولنقل عملية التقسيم من المنهج اليدوي إلى المنهج الآلي، كان لابدّ من إيجاد بنية موحدة أياً كان شكل مخطط النموذج المترفع، بحيث يؤول إليه فيغدو دخلاً موحداً للمنهجيّة الخاصة بنا. نقترح البنية

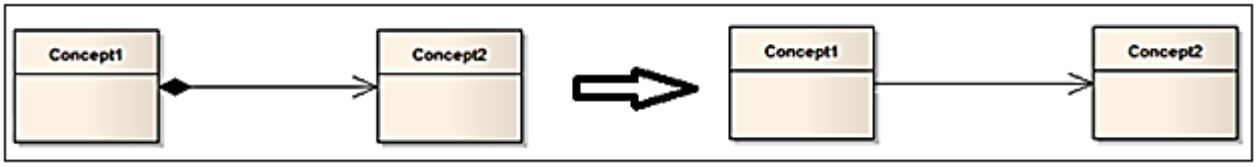
[2] - الملحق /أ/ يظهر مجموعة من النماذج المترفعة والتقسيم المقترح من الطلاب. أغلب تقسيماتهم تظهر بالألوان.

الشجرية بهدف تبسيط العمل والحسابات العددية. وفيما يلي نقترح خوارزميّة للتحويل إلى البنيّة الشجريّة ونبيّن أهم العقد.

1.3.10. خوارزمية التحويل إلى البنية الشجرية

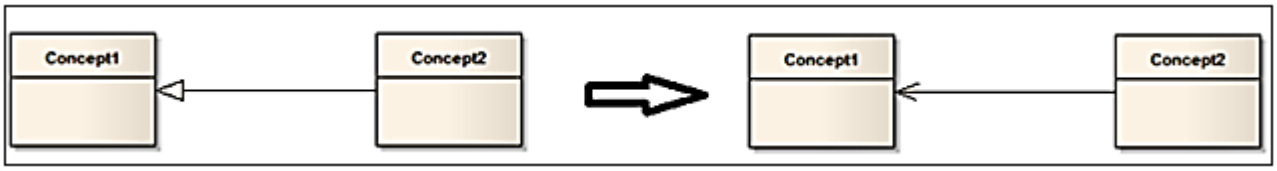
- تعاملنا خلال النماذج المترفّعة السابقة مع 3 أنواع من الروابط.

1- **علاقة التركيب:** ونحوّلها إلى سهم موجّه فقط من الكل إلى الجزء. الشكل التالي يوضح المعنى.



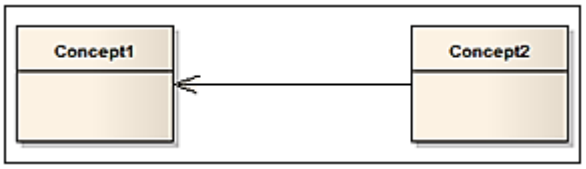
الشكل (10-1) تحويل علاقة التركيب

2- **علاقة الوراثة:** ونحوّلها إلى سهم فقط من الابن باتجاه الأب. الشكل التالي يوضح المعنى.



الشكل (10-2) تحويل علاقة الوراثة

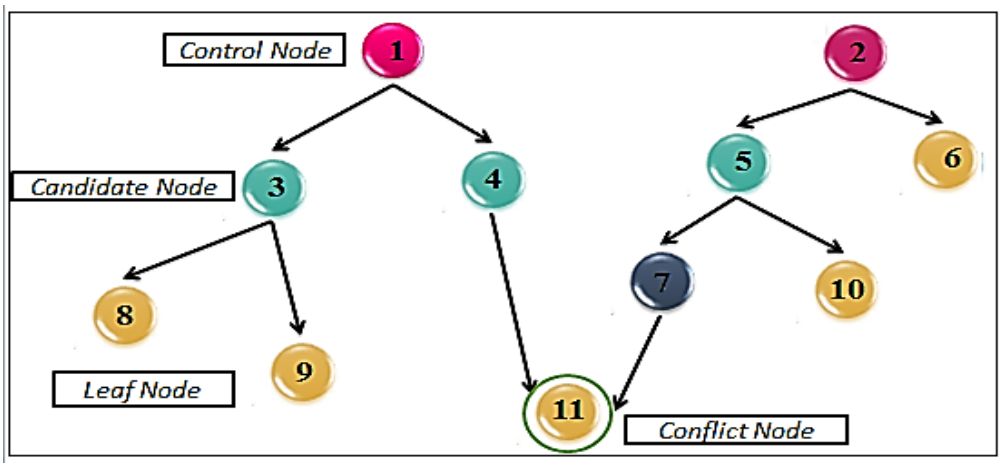
3- **علاقة التجميع:** ونبقي على حالها حسب اتجاه السهم. الشكل التالي يوضح المعنى.



الشكل (10-3) علاقة التجميع

- الأعداد على طرفي العلاقات والتي تحدد (واحد لواحد) أو (واحد لكثير) يتم إهمالها، فالهدف معرفة اتجاه العلاقة فقط.
- العلاقات المكررة يتم إهمالها.
- يتم إهمال العلاقة العكسيّة وهي علاقة المفهوم بنفسه.

الشكل التالي يبيّن المخطط العام الشجري الناتج بعد تطبيق خوارزمية التحويل.



الشكل (10-4) البنية الشجريّة

2.3.10. العقد الموجودة في البنية الشجريّة

نميز بين الأنواع التالية من عقد البنية الشجرية:

- **العقدة الورقة Leaf Node:** هي العقدة التي لا يصدر عنها أي رابط، هي فقط تعتبر مستقبل للروابط، كحال العقد
  (6, 8, 9, 10, 11) في الشكل السابق.
- **العقدة التحكمية Control Node:** هي العقدة الجذر في المخطط الذي يمسك بكل العقد ويمكن أن يحوي المخطط على أكثر من عقدة تحكميّة، كحال العقد (1,2) في الشكل السابق.
- **العقدة المرشحة Candidate Node:** هي العقدة التي ترتبط برابط مباشر مع العقدة التحكمية، ففي حال التقسيم، هي مرشّحة لأن تصبح عقدة تحكمية للنموذج الجزئي الناتج، كحال العقد (3, 4, 5, 6) في الشكل السابق.
- **العقدة المُتنازع عليها Conflict Node:** هي العقد التي ترتبط مع عقد تحكمية بشكل مباشر أو غير مباشر، فعند التقسيم تغدو العقدة محط تنازع مابين عدة نماذج جزئية، كحال العقدة (11) في الشكل السابق.

وفيما يلي اقتراح لمنهجيّة التقسيم بالاعتماد على المفاهيم السابقة.

4.10. المنهجيّة المُقترحة للتقسيم

في هذا القسم نقترح منهجيّة موحدة لتقسيم النماذج المترفعة مستندين على تحليل الدراسات التجريبيّة السابقة. نعتمد على عدد من العوامل والمفاهيم لتقسيم النماذج المترفعة.

1.4.10. مفهوم الرقم المحسوب للعقدة

نقترح مفهوم الرقم المحسوب للعقدة(Calculated Number of Node (***CNN***)) للدلالة على سوية العقد وعدد الوصلات الخارجة منها باتجاه السويات الدنيا. ونقترح التابع اللغرتمي كمعامل تخميد. نقوم بحساب عدد الوصلات الخارجة من العقدة باتجاه السوية الأدنى منها وتستمر العملية حتى وصولنا إلى أدنى مستوى (مستوى الأوراق).

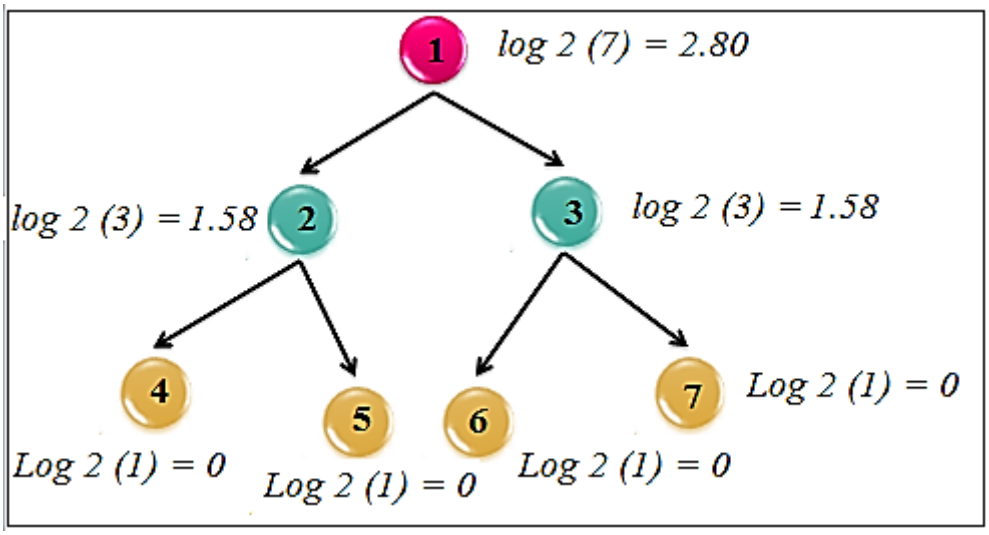
يتم احتساب ***CNN*** كما يلي:

$$CNN = Log\ 2\ (Number\ of\ Output\ Links\ to\ low\ levels + 1)$$

نعمد إلى إضافة العدد (1) إلى عدد الوصلات الخارجة من العقدة وذلك بهدف تجنب القيم اللانهائيّة. السوية الدنيا للعقدة الورقة معدومة، لذا ***CNN*** لها يُحتسَب كالتالي:

$$CNN\ (Leaf\ Node) = Log\ 2\ (0 + 1) = log\ 2\ (1) = 0$$

الشكل التالي يُقدم مثالاً حول حساب ***CNN*** للعقد.



الشكل (10-5) مثال حول حساب *CNN*

2.4.10. مفهوم وزن العقدة

نقترح هذا المفهوم (***WN***) (Weight for Node) للتعبير عن الحجم والترابط ما بين عقدة محددة وإحدى العقدة التحكميّة في النماذج المترفعة الجزئيّة، فكلما كانت ***WN*** أصغر، كلما كانت العقدة أكثر تلاحماً وإرتباطاً بالنموذج المترفع وأصعب من ناحية الفصل والتقسيم.

نحتاج إلى العديد من المفاهيم في هذه المرحلة.

- مفهوم **الترابط** ما بين العقدة المحددة والعقدة التحكميّة في أحد النماذج الجزئيّة، فعندما يكون لدينا حالة تنازع مابين عقدتين تحكميتين، لابدّ من تحديد أيهما أضعف ترابطاً مع العقدة المدروسة. إنّ الترابط الضعيف واحد من أهم العوامل المسببة للعيوب، لذا لابدّ من تجنبه.
- مفهوم **حجم وأهمية العقدة التحكميّة** في النماذج المترفّعة الجزئيّة، يُعتبر الحجم الكبير واحداً من أهم العوامل المسببة للعيوب، لذا لابدّ من تجنبه أيضاً.
- معامل **التخميد** وهو يساعد على جعل القيم أصغر ويساعدنا على وضع عتبة للتقسيم.

المنهجيّة المُقترحة تتكوّن من عدة معاملات هي:

- عدد الروابط ما بين العقدة والعقدة التحكميّة في النموذج المترفّع الجزئي ← وهو يمثّل الترابط.
- العدد المحسوب ***CNN*** للعقدة التحكميّة ← وهو يمثّل الحجم والأهميّة.
- Log 2 ← وهو يمثّل معامل التخميد.

وبالتالي الشكل النهائي للقانون كما يلي:

$$Weight\ of\ Node\ (WN) = Log\ 2\ (Number\ of\ Links\ between\ this\ Node\ and\ Controller\ Node) + CNN\ (Control\ Node)$$

وزن العقدة المرشّحة (التي يفصلها عن العقدة التحكميّة رابط واحد) هو كالتالي:

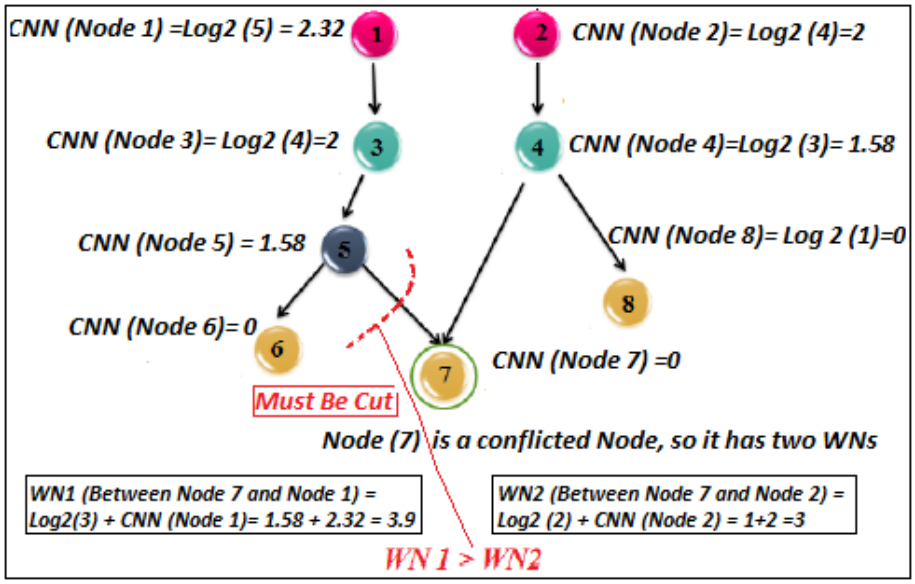
$$Weight\ CAndidate\ Node\ (WCAN) = Log\ 2\ (1) + CNN\ (Control\ Node) = CNN\ (Control\ Node)$$

لكل عقدة مُتنازع عليها أكثر من وزن واحد (***WN***)، وذلك بسبب وجود أكثر من عقدة تحكميّة تتنازع على العقدة المذكورة. لذا كان لابدّ من حساب أوزان هذه العقدة بالنسبة لكل واحدة من هذه العقد التحكميّة. فكلما كان هذا الوزن أصغر، كلما كانت العقدة أقرب للنموذج المترفّع الجزئي وصعبة الانفصال عنه.

نورد مثالاً في الشكل (10–6) لحساب الأوزان ***WN***.

عندما نقسّم النموذج المترفّع إلى عدة نماذج جزئيّة، فإنّ العقد المرشّحة تصبح عقداً تحكميّة في النماذج المترفّعة الجزئيّة. إذا كان لدينا عقدة نزاع كالعقدة رقم (7) في الشكل [10–6])، فإنّه يتوجب علينا حساب الأوزان لهذه العقدة مع كل العقد التحكميّة الموافقة. يظهر في الشكل عقدتان تحكميتان (1, 2) تتنازعان على هذه العقدة.

الوزن الأعلى يدل على حجم أكبر ومسافة أكبر وبالتالي ترابطاً أضعف، لذا يتوجب علينا اقتطاع الرابط من جهة الوزن الأعلى.



الشكل (10–6) حساب الأوزان *WN* لعقدة التنازع.

3.4.10. القواعد العامة المستنتجة من الدراسات التجريبيّة

- إذ كان العدد المحسوب للعقدة التحكميّة (Controller Node) ***CNN*** = < 4 عندها يكون النموذج كبير ويستوجب التقسيم إلى نماذج جزئية أصغر بحيث تكون أكثر ترابطاً، وإنّ العتبة 4 تعني أن لدينا 16 مفهوم أو ما يفوقها في نفس النموذج المترفّع لذا وجب التقسيم؛
- نقوم بالاقتطاع من عند العقدة المرشحة ذات الوزن الأكبر لأنها أقرب لعملية القسم وتحرير العقد المتنازع عليها من حالة النزاع، فالأولوية للعقدة المرشحة ذات العدد المحسوب الأعلى؛ فتغدو عقدة جذر مستقلة عن العقدة الجذر السابقة على حين تنخفض درجة العقدة الجذر السابقة كونها نقصت فرع من فروعها عند هذه العقدة.

- نسمح بوجود عقدة تحكميّة وحيدة حرّة، فهي ستشكل الحاوي والرابط ما بين النماذج الجزئيّة أما بقية العقد التحكميّة في النموذج فعند عملية القسم لابدّ من أن نراعي انتماءها لواحد من النماذج الجزئيّة، وذلك كي نحول دون تفكك النظام؛
- في حال وجود عقد مرشحة ترتبط ببعضها عن طريق علاقات معينة، سيكون لدينا الخيارين التاليين:
  - إذا كانت العقد ذات أعداد محسوبة كبيرة: عندها سيتم إلغاء العلاقات فيما بينها وجعل كل منها عقدة تحكمية مستقلة بذاتها وجذراً لنماذج جزئيّة لاحقة.
  - وإلّا سيتم الإبقاء على العلاقات وبالتالي سيكون لدينا عقدة تحكمية تنتمي إليها باقي العقد المرشحة ولن تعود عقداً تحكميّة.
- نبدأ باختبار العقد سوية تلو سوية؛
- في حال تساوي وزن العقدة بالنسبة لعقدتين تحكميتين فإن أولوية التقسيم تعود للمستخدم؛
- يتم إعادة حساب الأوزان والأعداد المحسوبة للعقد مباشرة بعد كل مرة من الاقتطاع؛
- في حال كان لدينا أكثر من عقدة تحكم، فإننا نحرص على الخطوات التالية:
  - إذا كانت العقدة تحوي رابطاً وحيداً وجميع العقد الأعلى منها سوية لا تحوي إلا رابطاً وحيداً، عندها لا نقتطعها، كي نحول دون تفكك النظام؛
  - العتبة الخاصة لنبدأ الاقتطاع إذا كان ***CNN*** للعقدة الجذر أكبر أو يساوي اللغرتم الثنائي لعدد مفاهيم النموذج المترفع / 3. أي:

$$CNN\ (\text{Control Node}) >= \text{Log2}\ (\text{Number of Concepts} / 3)$$

  - في أثناء حساب ***WN*** من أجل العقد المُتنازع عليها يتم احتساب عدد الوصلات مع أقرب عقدة تحكمية كون لدينا أكثر من جذر.

5.10. تمديد المنهجيّة المُقترحة

إنّ المنهجيّة التي قمنا باقتراحها تقوم فقط على البنية الهيكلية، وذلك بالاعتماد على الأوزان وعدد الروابط، ولا تأخذ بعين الاعتبار المنحى المنطقي والدلالي لأسماء المفاهيم.

لذا رأينا أنّه من الضرورة بمكان تمديد منهجيّة التقسيم المُقترحة لتشمل المنحى المنطقي من خلال التحليل القواعدي والمعجمي للأسماء في النموذج، واكتشاف نسبة تشابهها، ولمّا كانت مفردات اللغة كثيرة جداً كنا بحاجة لوثائق معياريّة، تخضع لنمذجة موحدة عالمية، من هنا لجأنا إلى استعمال الأنطولوجي، وهي وثائق مهيكلة على شكل ثلاثيات من RDF تساعد على قياس التشابه مابين المفردات. وبالتالي نكون قد صالحنا ما بين المنهجيتين الهيكليّة والمنطقية. الفصل العاشر يعرض نظرة موسّعة عن الأنطولوجي.

أما خوارزميّة المقارنة تعتمد على الخطوات التالية:

- يتم مطابقة اسم العقدة المُتنازع عليها مع أسماء العقد الموجودة في النموذج المترفّع كافة.
  - في حال التطابق الحرفي فإننا نجزم بحتمية التشابه ونعمد إلى إعطاء نسبة تامة للتطابق معها = 1.
  - في حال عدم التطابق الحرفي، يتم الرجوع إلى الأنطولوجي، وأخذ نسبة التشابه.
- يتم احتساب وسطي نسب تشابه اسم العقدة مع أسماء باقي العقد.

وبما أن المبدأ الذي نعمل عليه هو "**كلما كان الوزن أصغر، كلما كانت العقدة أقرب**"، لذا كان لابدّ من أخذ معكوس نسبة التشابه، لأن Ontology سترد نسبة التشابه بشكل طردي، أي الأكثر تشابهاً هو الأكبر قيمة وهي قيمة ضمن المجال [1−0]. وبالتالي سيتم قلب القيم لكي تتناسب مع المفاهيم المطروحة، وفقاً للتالي:

(1- Avg (Similarity (This Node, Nodes of Sub meta-models))

إن الأنطولوجي Word.net تتميّز بتغطيتها لعدد كبير من الكلمات والمفاهيم وغناها بالعلاقات المفرداتية [91].

أما القانون بعد التعديل عليه بهدف إدخال البعد المنطقي يصبح كالتالي:

Weight Node (WN) = Log 2 (Number of Links between this Node and Control Node) + CNN (Control Node) + (1- Avg (Similarity (This Node, Nodes of Sub meta-models)). Where 0<= Similarity <=1

6.10. الخلاصة

(من خلال اقتراح منهجيّة التقسيم نكون قد نقلنا الخبرة التجريبيّة التي استمرت لمدة ستة أشهر في كلية الهندسة المعلوماتية في جامعة دمشق من المنهج اليدوي إلى المنهج التلقائي، وأحطنا بمفهوم التقسيم بما يضمن الابتعاد عن الحجم الكبير وضعف الترابط ما بين المفاهيم في النماذج المترفعة (أعلى سويّة) وكاملنا ما بين المنهجين، المنهج الهيكلي المعتمد على الروابط والأوزان، والمنهج المنطقي المعتمد على التحليل المعجمي.)

الفصل الحادي عشر
دراسة التشابه باستخدام الأنطولوجي

1.11. مقدمة

إنّ عملية التحليل القواعدي للكلمات واكتشاف نسبة تشابهها يُعتبر عملاً بالغ الأهمية في إضفاء الاتجاه المنطقي على العمل، ولمّا كانت مفردات اللغة كبيرة جداً كنا بحاجة لوثائق معياريّة، تخضع لنمذجة موحدة عالمية، يتم اعتمادها من قبل مراكز علمية موثوقة، من هنا لجأنا إلى استعمال Ontology، وهي وثائق مهيكلة على شكل ثلاثيات من الـ RDF تساعد على قياس التشابه مابين المفردات.

2.11. منصة وصف المصادر RDF

إنّ ارتباط البيانات بعضها ببعض ارتباطاً بسيطاً غير كاف، فتوصيف الخصائص للمفردات والعلاقات بينها مهم لتحقيق التشغيل البيني Interoperability بين التطبيقات التي تتبادل بيانات غير مفهومة من قبل الآلة.

وهذا ما يقدمه معيار RDF الذي طورته W3C لوصف المصادر، والذي يحوي نموذجاً لوصف كل مصدر بيانات باستخدام تعبير مكوّن من ثلاثية RDF، كما يقدم المعيار أدوات تتيح المعالجة الآلية لموارد الويب التي تلغي أية افتراضات حول مجال تطبيق معيّن، بناءً على ذلك يعدّ RDF أساس توصيف البيانات على الوب [92].

تحوي ثلاثيّة RDF ثلاثة مكونات:

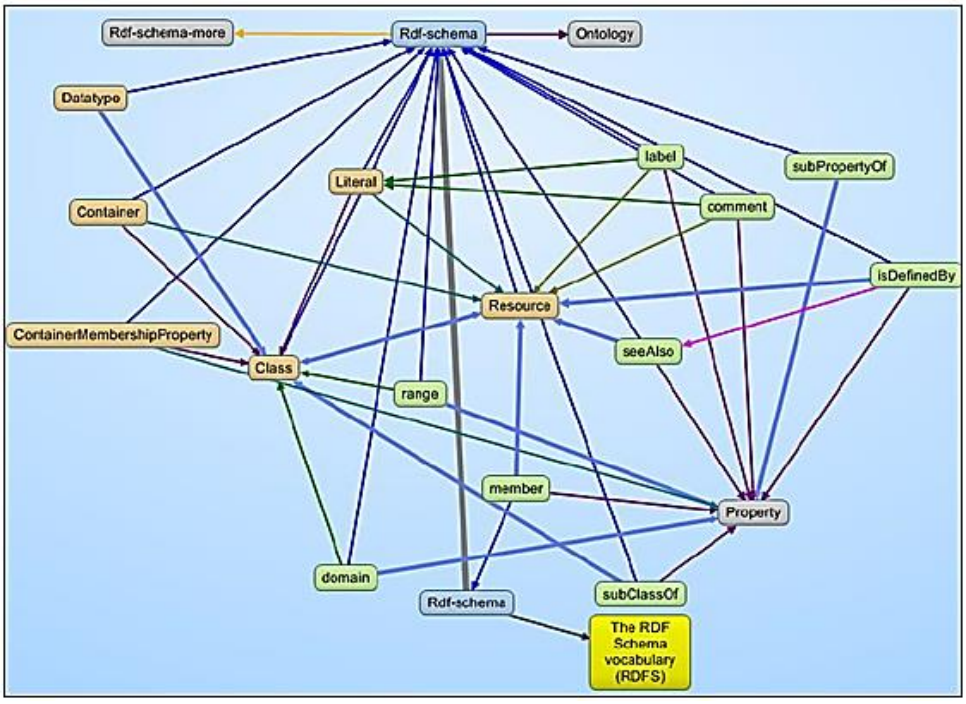
- **الموضوع** (Subject): وهو RDF URI أو عقدة فارغة؛
- **العلاقة** (Predication): هي RDF URI؛
- **الغرض** (Object): وهو RDF URI أو مصدر حرفي (Literal) أو عقدة فارغة.

وتتم كتابة الثلاثيات على النحو التالي: الموضوع – العلاقة – الغرض.

الشكل (11-1) ثلاثيّة RDF

3.11. مخطط منصة وصف المصادر RDFs

يضمّ مخطط RDFs (Recourse Description Framework Schema) مجموعة من مصادر RDF المستخدمة لوصف علاقات مصادر RDF الأخرى، وقد عُرّفت العلاقات والمفردات الأساسية في فضاء اسمي (Namespace) وسُميت RDFs وهي موضحة بالشكل التالي.



الشكل (11-2)علاقات مفردات RDFs الأساسية

تؤمن مخططات RDF للمطورين إمكانية تحديد علاقات خاصة ببيانات RDF وتحديد أنواع الأغراض التي تربط بينها هذه العلاقات. إذ تعرّف مخططات RDF مفاهيم النمط والصف واشتقاق الصف صورياً وتضيفها إلى RDF، تمتاز هذه المخططات بإتاحة إمكانية تعريف علاقات RDF الدلالية صراحةً، وتُنظم هذه العلاقات ضمن هرمية منطقية Hierarchy (Typed) من خلال مجموعة من المفاهيم، نذكر منها:

- Class;
- subClassOf;
- type;
- Property;
- subPropertyOf.

وبالتالي تجمّع عدد كبير من ثلاثيات RDF تشكل مخططات كحال Dublin Core [3] و vCard [4]، وقد تشكل Ontologies كحال Word NET [5].

4.11. لغات الأنطولوجيات

يحتاج تكامل البيانات إلى التوافق على المفردات وأصنافها والعلاقات بينها. بعد ذلك نستطيع استنتاج علاقات جديدة بين هذه البيانات. لكن المشكلة تبقى في إيجاد لغة تضمن تعريف المفردات وتحديد العلاقات بين البيانات بدلالة واضحة لاغموض فيها، قد يتبادر إلى الذهن استخدام RDFS لغة لهذا الغرض لأنها تحوي مفاهيم الأنماط والصفوف والاشتقاق مع إمكانية تعريف العلاقات تعريفاً هرمياً، لكن تبقى مشكلتها في محدوديتها من حيث المفردات.

1.4.11. تعريف WordNet

هي قاعدة بيانات قواعديّة للغة الإنكليزيّة، تعمل على تجميع كلمات اللغة في مجموعات من المترادفات، تزودنا بتعريف مختصر وسريع عن استعمال مفردات اللغة. بالإضافة إلى أنها تؤمّن عدد الارتباطات ما بين المجموعات، يمكن أن ننظر لها كتجمع لعدد من قواميس ومكانز اللغة، يمكن للأشخاص استخدامها بشكل مباشر من خلال مستعرض الوب، غير أنّ استخداماتها الأساسية هي بالطرق الآلية لتحليل النصوص وتطبيقات الذكاء الصنعي. لذا وجدنا أنّها حلاً أمثلياً لاستخدامها في كشف تشابه التسميات للمفاهيم الموجودة بالنماذج المترفعة بغرض فصلها عن بعضها [91].

[3] - http://dublincore.org/.
[4] - http://www.w3.org/TR/2013/WD-vcard-rdf-20130502/.
[5] - http://wordnet.princeton.edu/.

2.4.11. لمحة تاريخيّة

نشأت هذه الأنطولوجي في مخبر (Cognitive Science) في جامعة (Princeton) سنة 1985 وتلقى دعماً كبيراً من الوكالات الحكومية، وهي مجانية بالكامل. طرأ عليها الكثير من التعديلات منذ نشأتها وحتى يومنا هذا.

3.4.11. محتويات Word Net

الشكل التالي يبيّن أهم المفاهيم والمفردات والخصائص التي تحويها هذه الأنطولوجي.

Property	Domain	Range	Prolog clause
containsWordSense	Synset	WordSense	s
word	WordSense	Word	s
lexicalForm	Word	xsd:string	s
synsetId	Synset	xsd:string	s
tagCount	Synset	xsd:integer	s
gloss	Synset	xsd:string	g
frame	VerbWordSense	xsd:string	fr
hyponymOf	Synset	Synset	hyp
entails	Synset	Synset	ent
similarTo	Synset	Synset	sim
memberMeronymOf	Synset	Synset	mm
substanceMeronymOf	Synset	Synset	ms
partMeronymOf	Synset	Synset	mp
classifiedByTopic	Synset	Synset	cls
classifiedByUsage	Synset	Synset	cls
classifiedByRegion	Synset	Synset	cls
causes	Synset	Synset	cs
sameVerbGroupAs	Synset	Synset	vgp
attribute	Synset	Synset	at
adjectivePertainsTo	Synset	Synset	per
adverbPertainsTo	Synset	Synset	per
derivationallyRelated	WordSense	WordSense	der
antonymOf	WordSense	WordSense	ant
seeAlso	WordSense	WordSense	sa
participleOf	WordSense	WordSense	ppl
classifiedBy	Synset	Synset	cls
meronymOf	Synset	Synset	mm,ms,mp

الشكل (11-3) أهم المفردات الموجودة في WordNet

4.4.11. لغة SPARQL

تسمح لغة SPARQL (SPARQL Protocol And RDF Query Language) باستخراج القيم من البيانات المهيكلة وشبه المهيكلة الممثلة في RDF، فضلاً عن استكشاف بيانات RDF بالستعلام عن علاقات مجهولة، وتسمح هذه اللغة بتنفيذ استعلامات معقدة لبيانات من مخازن مختلفة وتحويل بيانات RDF من تمثيل إلى آخر[93]، مما سبق نستطيع القول: إنّ هذه اللغة تساعد على تطوير تطبيق عالي المستوىعبر منصات مختلفة، ويكون تركيب استعلام SPARQL على النحو الآتي، الشكل التالي يوضح المفهوم.

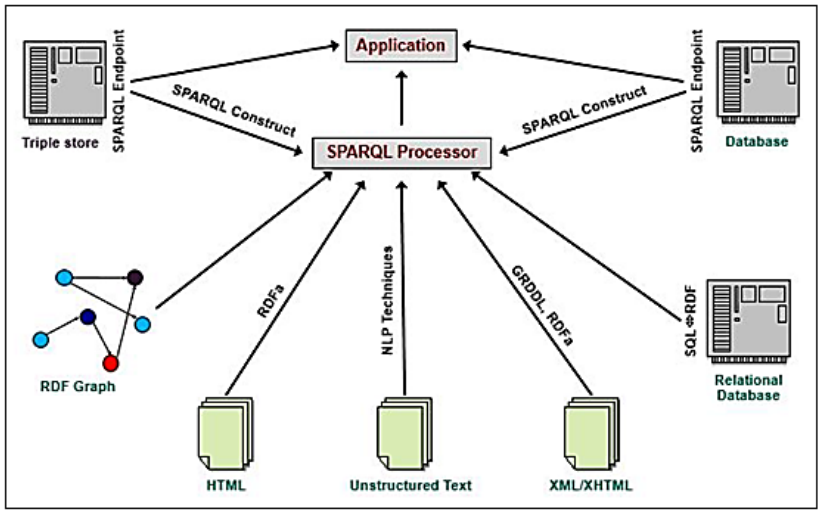
```
# التصريح بالبادئة
PREFIX example: <http://example.org/resources/> ...
# تعريف مجموعات البيانات التي سيُستعلم عنها
FROM ...
# النتائج المطلوبة
SELECT ...
# بنية الاستعلام
WHERE { ... }
# شروط الاستعلام
ORDER BY ...
```

الشكل (11-4) شرح استعمال لغة SPARQL

تُنفذ استعلامات SPARQL على مجموعة بيانات RDF، وتقبل نقطة الخدمة (Endpoint) في SPARQL الاستعلامات وتعيد النتائج من خلال وصلات وبصيغ مختلفة: (XML, JSON, RDF, HTML)، وتُعرّف بروتوكولات وتنسيقات الصيغ ضمن وثائق منفصلة على الوب، وتقدم مجموعة البيانات غالباً نقاط خدمة مختصة بها. بذلك تكون لغة SPARQL هي النقطة التي تجمع مصادر ومجموعات البيانات من جهة، مع التطبيقات التي تظهر نتائج الاستعلامات من جهة أخرى. كما يوضح ذلك الشكل التالي، وفيما يلي مثالاً حول الاستعلام عن الكلمة التي تحوي الجزء "bank".

```
PREFIX wn20schema: <http://www.w3.org/2006/03/wn/wn20/schema/>

    SELECT ?aSynset
    WHERE { ?aSynset wn20schema:containsWordSense ?aWordSense .
            ?aWordSense wn20schema:word ?aWord .
          ?aWord wn20schema:lexicalForm "bank"@en-US }
```



الشكل (11–5) لغة الاستعلام في الوب الدلالي SPARQL

5.11. خوارزميات Lesk لقياس التشابه

وهي واحدة من خوارزميات التصنيف المُقدمة من قبل Michael E. Lesk سنة 1986.

تعتمد هذه الخوارزمية على افتراض أنّ الكلمات المتجاورة تتشارك بتوصيف موضوع محدد، فهي تسعى لمقارنة التعريف المعجمي للكلمات الغامضة مع المفردات المجاورة لها [94]، وبالتالي احتساب عدد الكلمات المشتركة ما بين تعاريف كلمتين كالتالي:

لكل كلمة مجموعة تعاريف تتم مقارنة كل التعاريف بعضها ببعض وجمع عدد تقاطع الكلمات.

مثال:

PINE
1. kinds of evergreen tree with needle-shaped leaves
2. waste away through sorrow or illness

CONE
1. solid body which narrows to a point
2. something of this shape whether solid or hollow
3. fruit of certain evergreen trees

As can be seen, the best intersection is Pine #1 ⋂ Cone #3 = 2.

- **ميزات الخوارزمية:**

✓ غير مقيدة بالـ Part Of Speech للكلمات المدخلة، فالمصطلحات المدخلة للمقارنة لا حاجة لأن تكون من نفس POS.

✓ لا تنحصر بهرمية kind-of للـ WordNet كبعض قياسات التشابه Semantic Similarity Measures التي تعتمد على طول الطريق بين مصطلحين ضمن هرمية WordNet، مثل خوارزمية LCH.

- **سلبيات الخوارزمية:**

☒ تعتمد فقط على تقاطع تعاريف الكلمات المعنية بالقياس مباشرة، وهذا يشكل محدودية في الأداء لأن التعاريف في WordNet قصيرة نسبياً وبالتالي لا توفر ما يكفي من المفردات التي تستطيع عند مقاطعتها تمييز مدى ارتباط المعاني مع بعضها البعض.

☒ كما يوجد جانب سلبي آخر للخوارزمية إذ أنّ score يحسب بناء على عدد المفردات المشتركة بين التعريفين ، ولا يميز بين تقاطع كلمات بشكل منفرد وبين وتقاطع عبارات phrases مؤلفة من أكثر من كلمة، بل يعتبر أي تقاطع مجرد مجموعة من الكلمات : bag of words.

غير أنّ هذه الخوارزمية أمثليّة للعمل لأننا بصدد قياس تشابه تسميات للمفاهيم ولسنا بصدد قياس تشابه جمل أو شبه جمل، لذا لكي يناسبنا العمل نقترح إضافة البحث عن التسميات فيما إذا كانت ترتبط ارتباطاً مباشراً مع بعضها، ففي حال وجود الرابط المباشر (*hypernym, hyponym, meronym, holonym* ..) عندها ستكون نسبة التشابه =1، وإلا نلجأ عندها إلى المقارنة المذكورة سابقاً مع عمل Normalization للنتيجة من خلال قسمتها على عدد الكلمات الموجودة في كلا التعريفين.

6.11. الخلاصة

(تمت دراسة الأنطولوجي بشكل موسّع نوعاً ما، واعتماد العمل على Wordnet نظراً لغناها بالمترادفات، واعتماد خوارزميّة Lesk لقياس التشابه بين أسماء المفاهيم، وبذلك نكون أدخلنا البعد المنطقي على العمل من خلال التحليل القواعدي للأسماء وقياس تشابهها باستخدام الأنطولوجي.)

الفصل الثاني عشر
التطبيق العملي للمنهجيّة المُقترحة

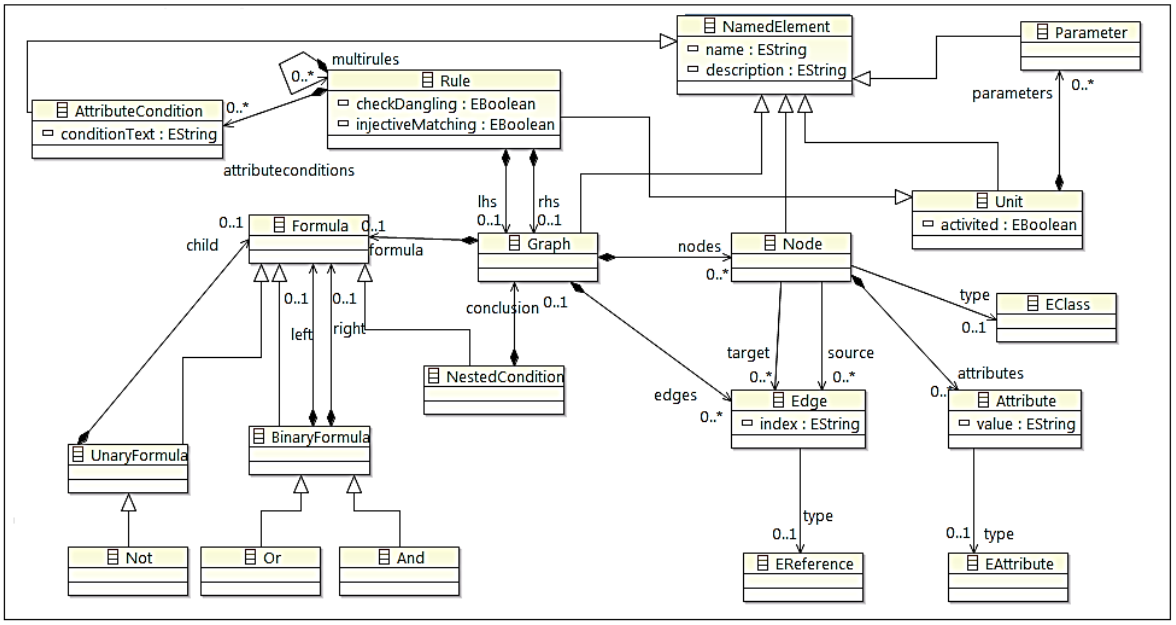
1.12. مقدمة

قمنا بتطبيق المنهجيّة المُقترحة على مجموعة من النماذج المترفعة المعياريّة العالميّة، لبيان كيفية عملها وكيفية التقسيم الذي تقترحه.

وفيما يلي نستعرض الخطوات التفصيليّة للمنهجيّة والحسابات العدديّة الموافقة.

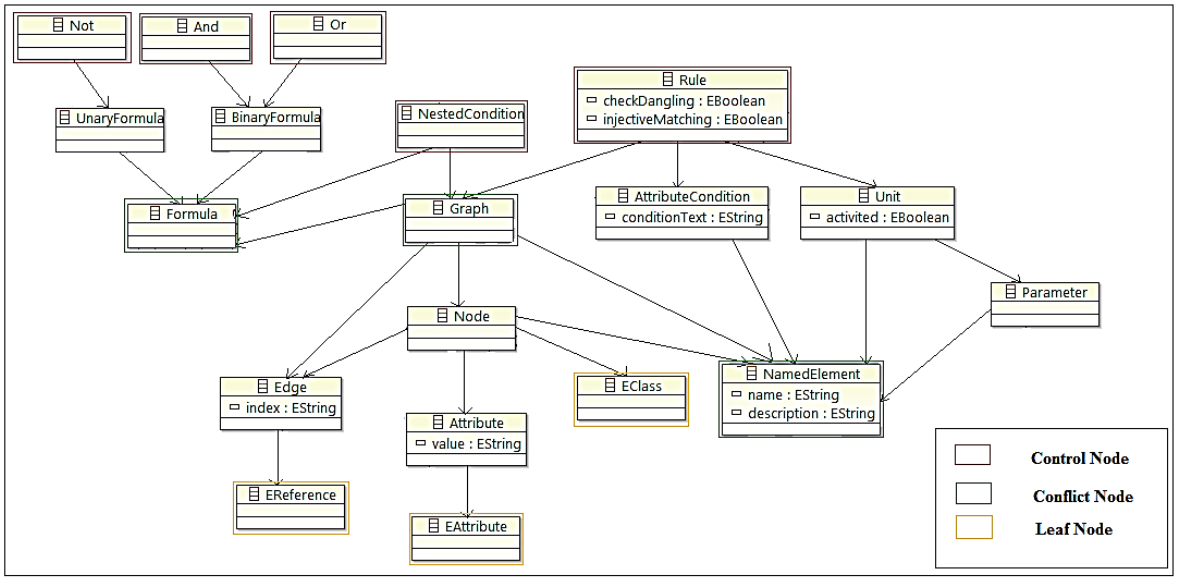
2.12. تطبيق المنهجيّة على النموذج المترفع Henshin

سنتناول جزء من النموذج المترفع Henshin [95]، وهو أداة للنمذجة ونموذج انتقال للغة، يعتمد على إطار عمل Eclipse [96]. الشكل التالي يبيّن النموذج المترفع له.



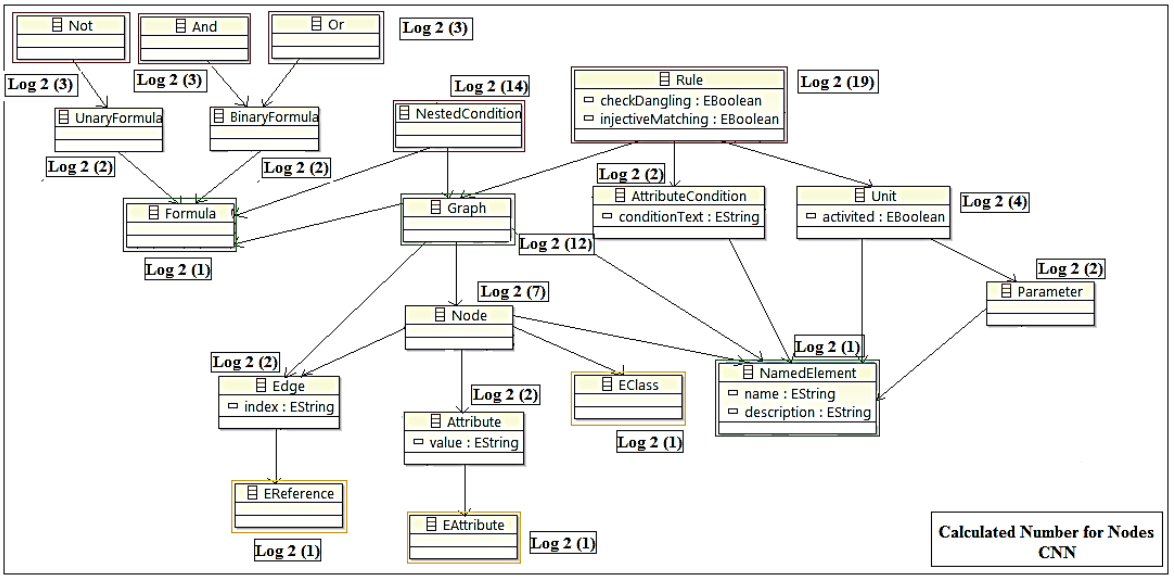
الشكل (12–1) جزء من النموذج المترفع Henshin

1- نقوم كخطوة أولى بالتحويل إلى البنية الشجريّة، وذلك بعد تحويل العلاقات حسب القواعد الموضحة سابقاً يصبح الشكل كما هو مبيّن في الشكل التالي.



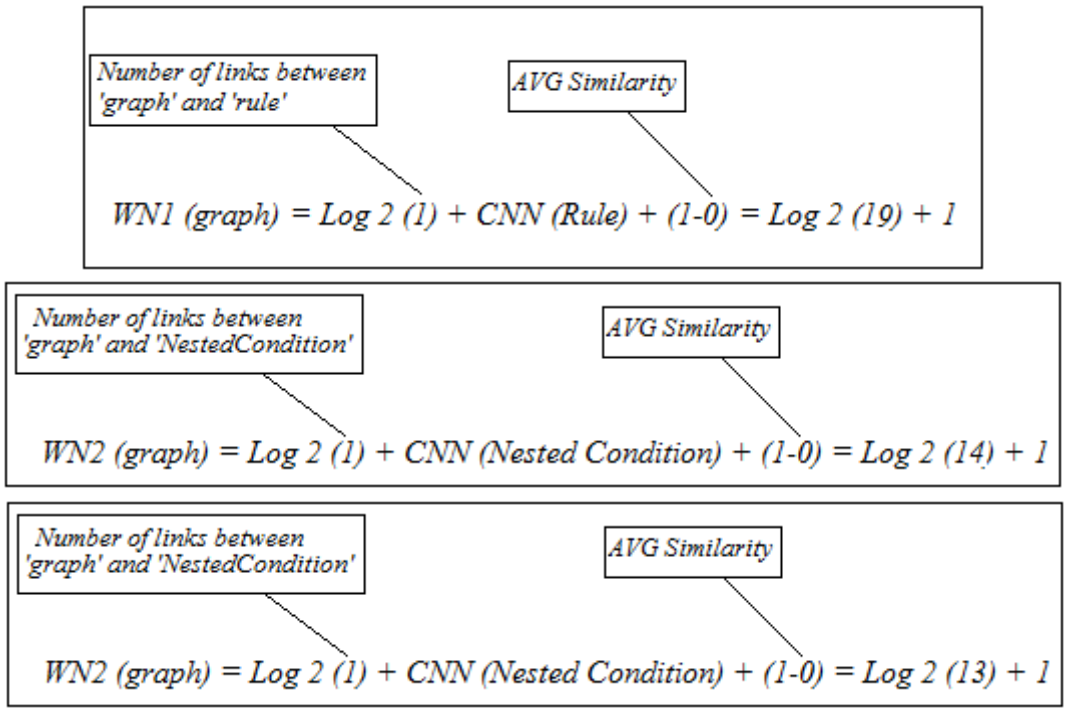
الشكل (12-2) البنية الشجريّة

2- حساب الأعداد المحسوبة لكافة العقد كما يبيّنها الشكل التالي.

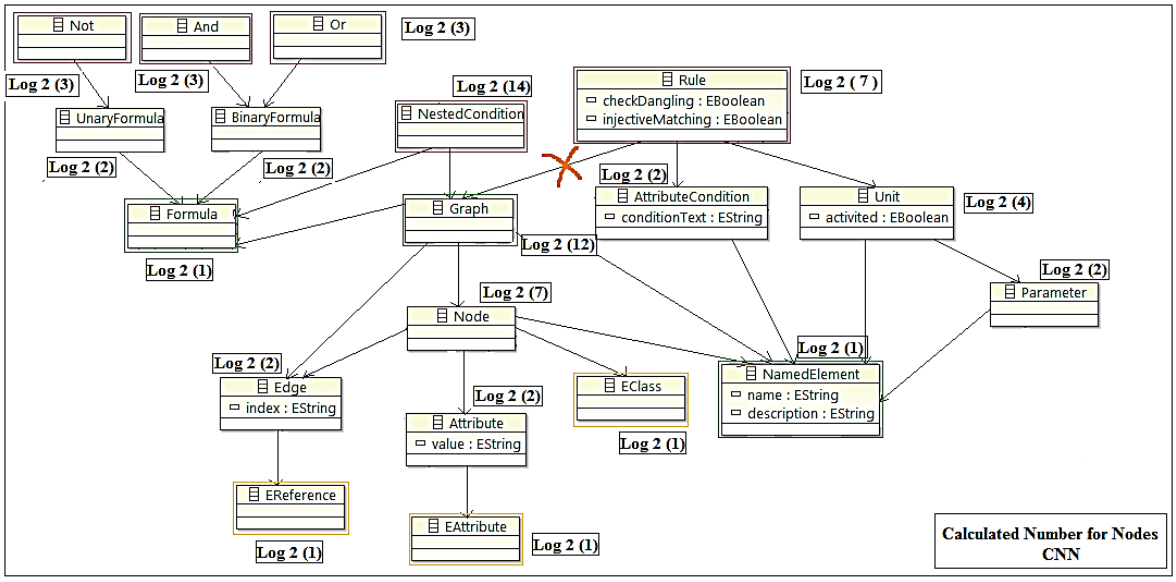


الشكل (12-3) الأعداد المحسوبة CNN

3- نبدأ من العقدة ذات العدد المحسوب الأكبر وهي Rule، نرى العقد المُتنازع عليها فنجد العقدة المرشّحة Graph، تتازع العقدتان التحكميتان Rule ، Nested Condition عليها. ويتم حساب وزن هذه العقدة WN بالنسبة لكليهما.



4- نقارن القيم فنجد 2 ***WN*** > 1 ***WN*** وبالتالي نقتطع من عند القيمة الأكبر أي نقطع الرابط ما بين Graph و Rule ونعيد حساب ***CNN*** من جديد للعقد. تتعدل القيم كما يبدو في الشكل التالي وتظهر العقدة Nested Condition هي أكبر قيمة من حيث العدد المحسوب فنبدأ منها لنجد أن هناك عقدة متنازع عليها هي Formula.



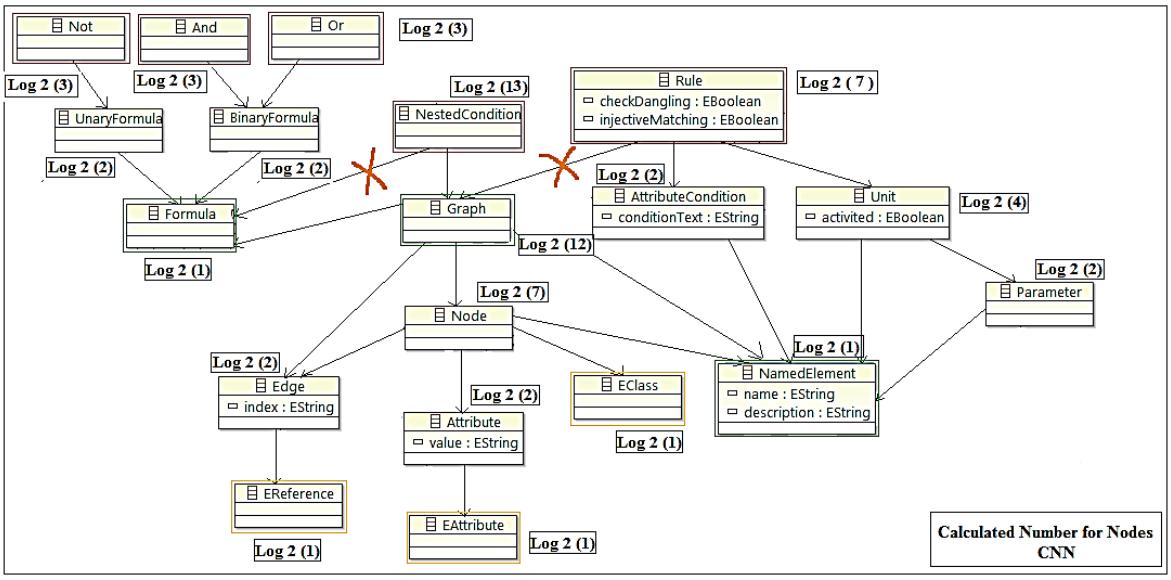
الشكل (12-4) الخطوات التفصيليّة للمنهجيّة

5- نحسب الأوزان للعقدة Formula بالنسبة للعقدتان التحكميتان Nested Condition و Or كما يلي:

$$WN\ 1 = Log\ 2\ (1) + Log\ 2\ (14) + (1\text{-}0) = Log\ 2\ (14) + 1.$$

$$WN\ 2 = Log\ 2\ (2) + Log\ 2\ (3) + (1\text{-}0) = Log\ 2\ (3) + 2.$$

6- نقارن الأوزان فنجد أنّ WN 2 < WN 1 وبالتالي نقتطع الرابط ما بين Nested Condition وما بين Formula ونعيد حساب الأوزان مجدداً. والشكل التالي يبيّن المفهوم.



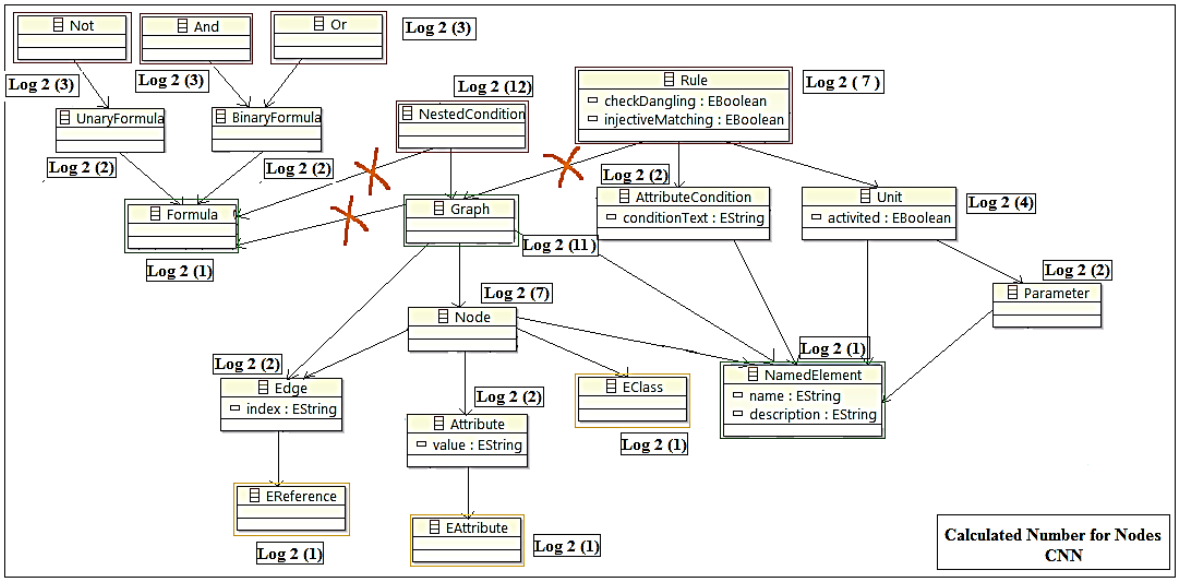
الشكل (12-5) الخطوات التفصيليّة للمنهجيّة

7- لاتزال العقدة Nested Condition هي ذات العدد المحسوب الأعلى لذا نتابع في عقدها والعقد التي تسير إليها لنصل إلى عقد متنازع عليها أخرى وهي Formula لكن ما بين العقدة Graph ومابين العقدة Or ونقوم بالحساب أيضاً.

$$WN\ 1 = Log\ 2\ (2) + Log\ 2\ (13) + (1\text{-}0) = 2 + Log\ 2\ (13)\ .$$

$$WN\ 2 = Log\ 2\ (2) + Log\ 2\ (3) + (1\text{-}0) = 2 + Log\ 2\ (3)\ .$$

8- نقارن الأوزان فنجد WN 2 < WN 1 وبالتالي نقتطع الرابط ما بين Graph و Formula ونعيد حساب الأوزان كما في الشكل التالي.



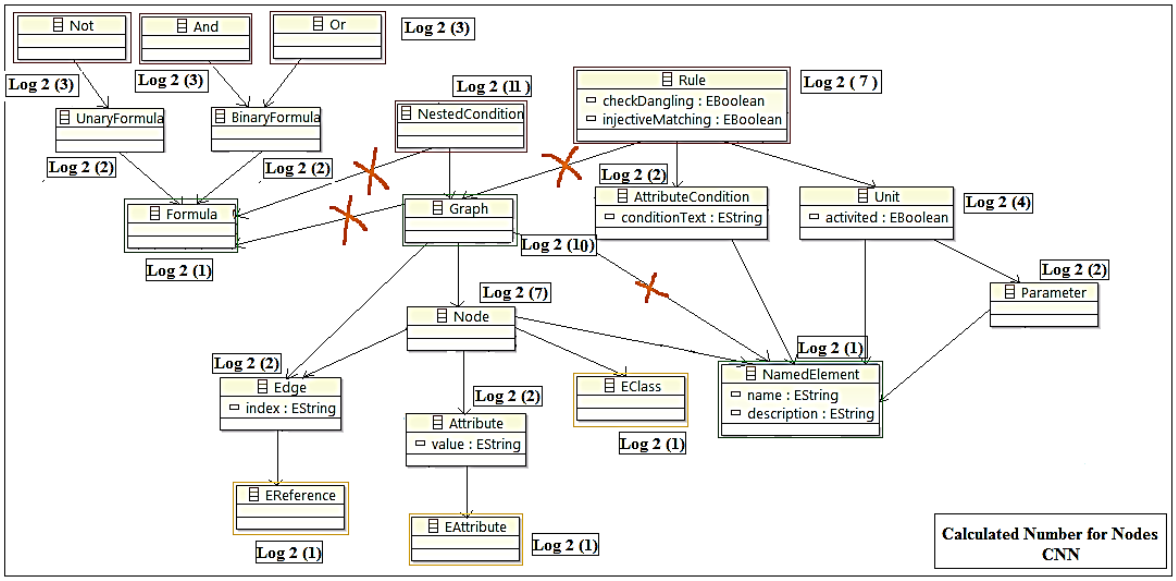
الشكل (12-6) الخطوات التفصيليّة للمنهجيّة المُقترحة

9- لا تزال العقدة Nested Condition أعلى العقد وزناً فنستمر في فحص عقدها لنصل إلى العقدة Nested Element وهي عقدة مُتنازع عليها مابين Nested Condition و Rule ، فيتم حساب الأوزان لها.

$$WN\ 1 = Log\ 2\ (2) + Log\ 2\ (7) + (1-0) = 2 + Log\ 2\ (7) = 4.48\ .$$

$$WN\ 2 = Log\ 2\ (2) + Log\ 2\ (12) + (1- 0.875) = 5.\ 459$$

10- نقارن بين القيم لنجد أنّ WN 1 < WN 2 وبالتالي نقتطع الرابط ما بين Nested Element و Graph ونعيد حساب الأعداد المحسوبة كما يبيّن الشكل التالي.



الشكل (12-7) الخطوات التفصيليّة للمنهجيّة المُقترحة

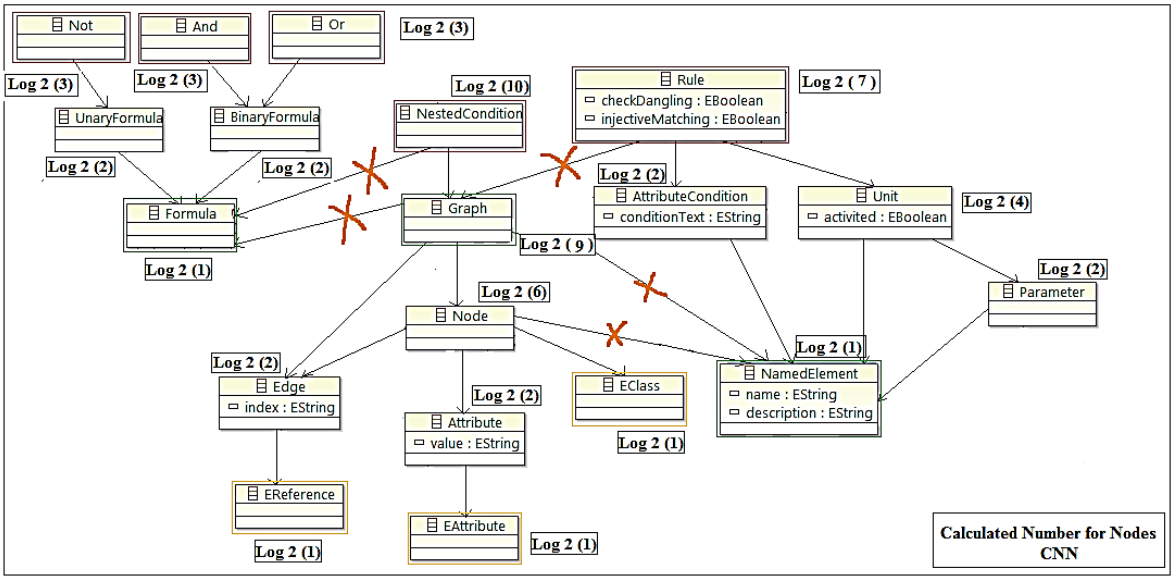
11- لا تزال العقدة Nested Condition أعلى العقد وزناً فنستمر في فحص عقدها لنصل إلى العقدة Nested Element وهي عقدة مُتنازع عليها مابين Nested Condition و Rule ، فيتم حساب الأوزان لها.

فنحسب القيم الجديدة للأوزان لنجد أنّ:

$$WN1 = Log\ 2\ (2) + Log\ 2\ (7) + \ (1\text{-}0) = 2 + Log\ 2\ (7) = 4.8$$

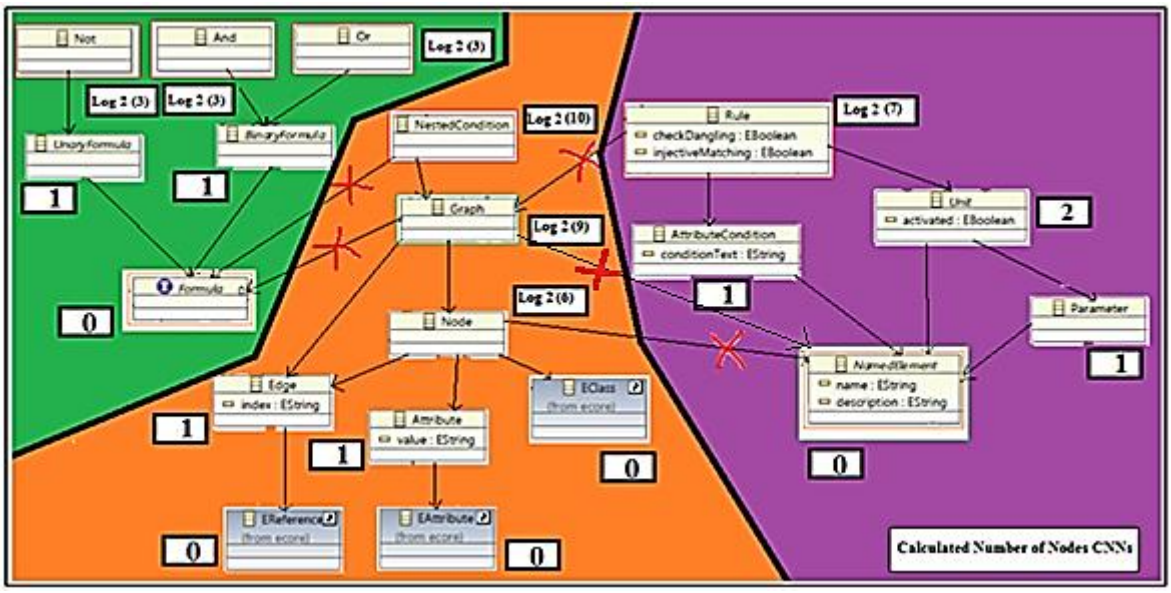
$$WN2 = \ Log\ 2\ (3) + Log\ 2\ (11) + (1\text{-}0) = Log\ 2\ (3) + Log\ 2\ (11) + 1 = 6.044$$

12- نقارن القيم لنجد أنّ WN 1 < WN 2 وبالتالي نقتطع الرابط ما بين Nested Element و Node ونعيد حساب الأعداد المحسوبة كما يبيّن الشكل التالي.

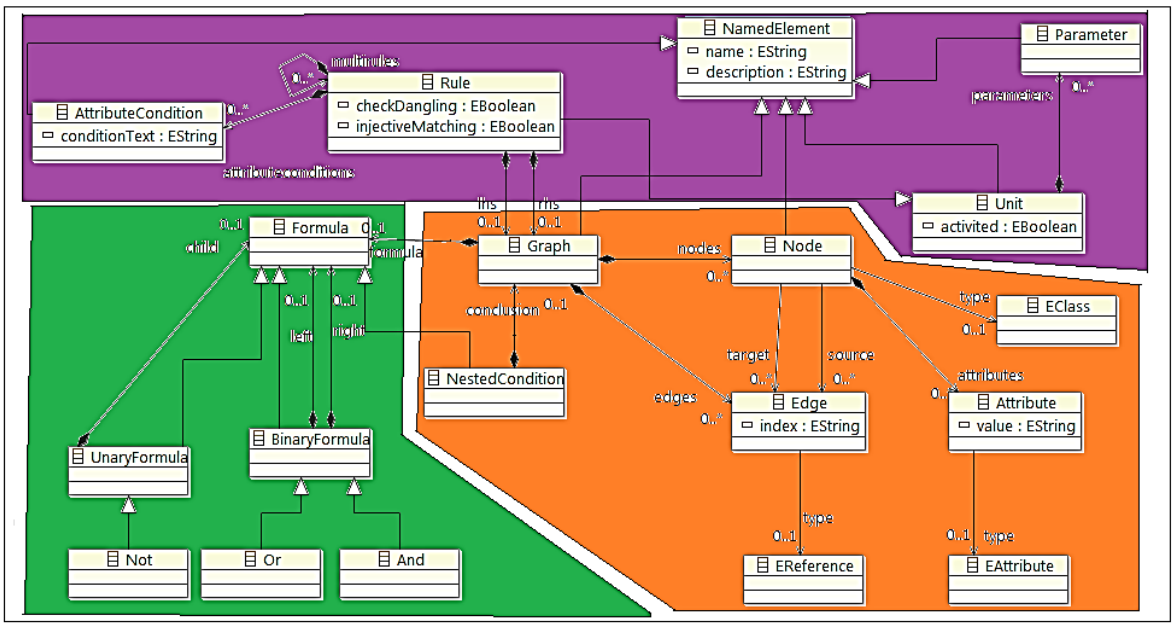


الشكل (12-8) الخطوات التفصيليّة للمنهجيّة المُقترحة

هناك العديد من العقد التحكمية (Not, And, Or) التي تتنازع على عقد أخرى. لكننا لا نقتطع هذه العقد كونها تملك رابطاً وحيداً مع عقد النموذج لأننا إن اقتطعناها أضحت عقداً حرّة. ونحن نريد أن نحول دون تجزء وتفكك النموذج. وهكذا تمّ تقسيم النموذج إلى 3 أقسام مستقلة. الشكل (12-8) يوضح الفكرة. وبالعودة إلى الشكل الأصلي للنموذج Henshin يبدو الشكل (12-9) يوضح المفهوم.



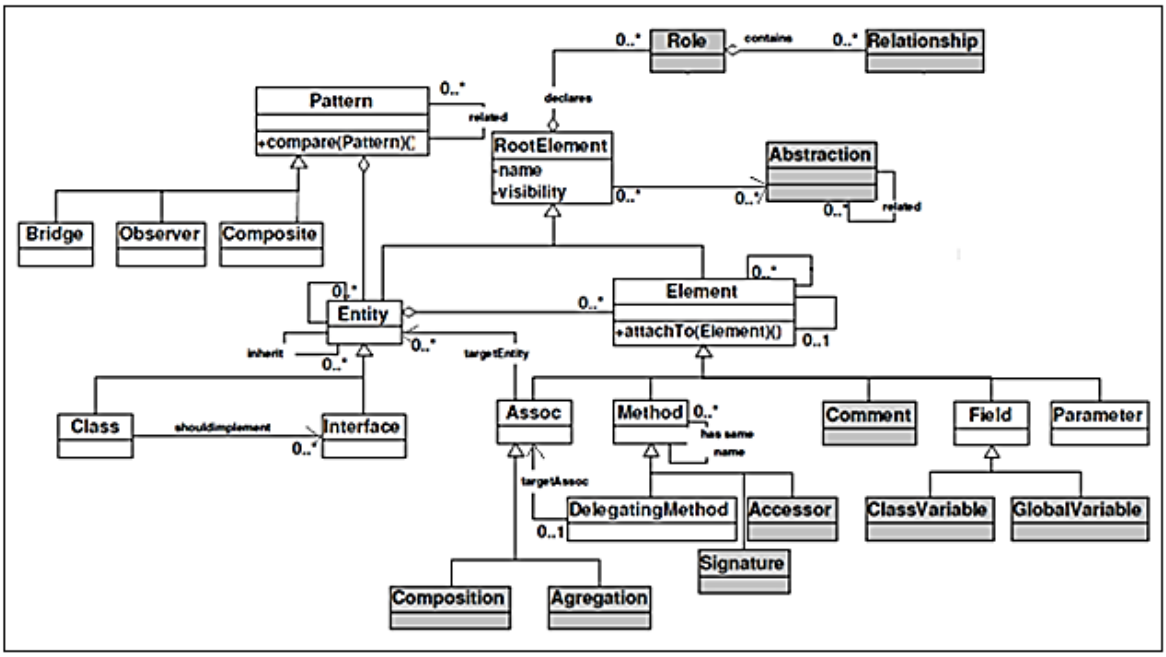
الشكل (12–9) الشكل الشجري النهائي لنموذج Henshin بعد التقسيم



الشكل (12–10) الشكل النهائي للنموذج المترفع Henshin بعد التقسيم

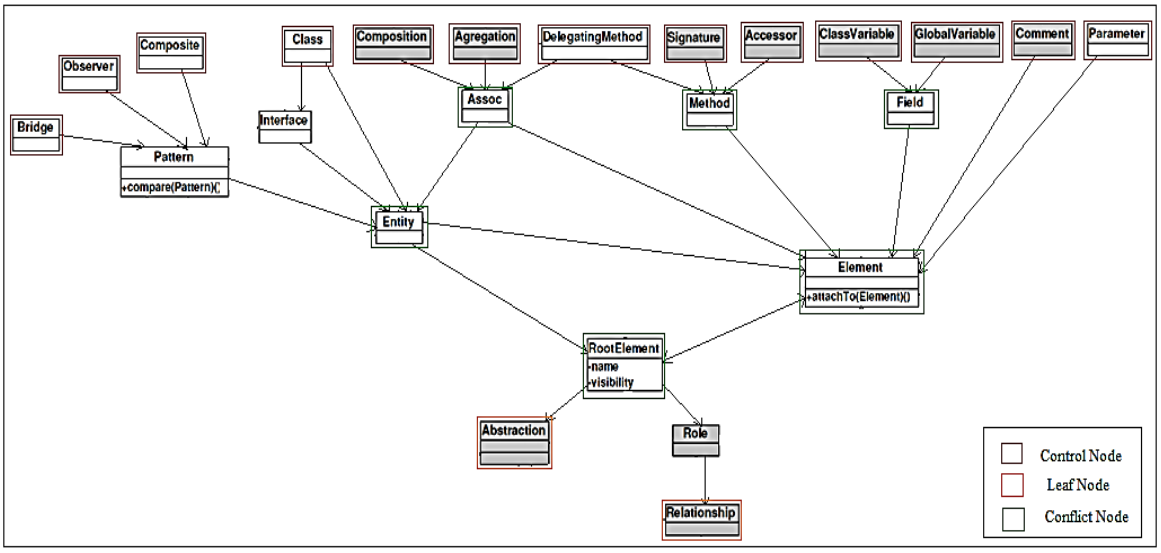
3.12. تطبيق المنهجيّة على النموذج المترفع PADL

وهو نموذج مترفع للغة غرضية التوجّه، تمّ اقتراحه عام 2002 وأدخل عليه عدة تعديلات في عام 2006، الشكل التالي يوضح النموذج [35].



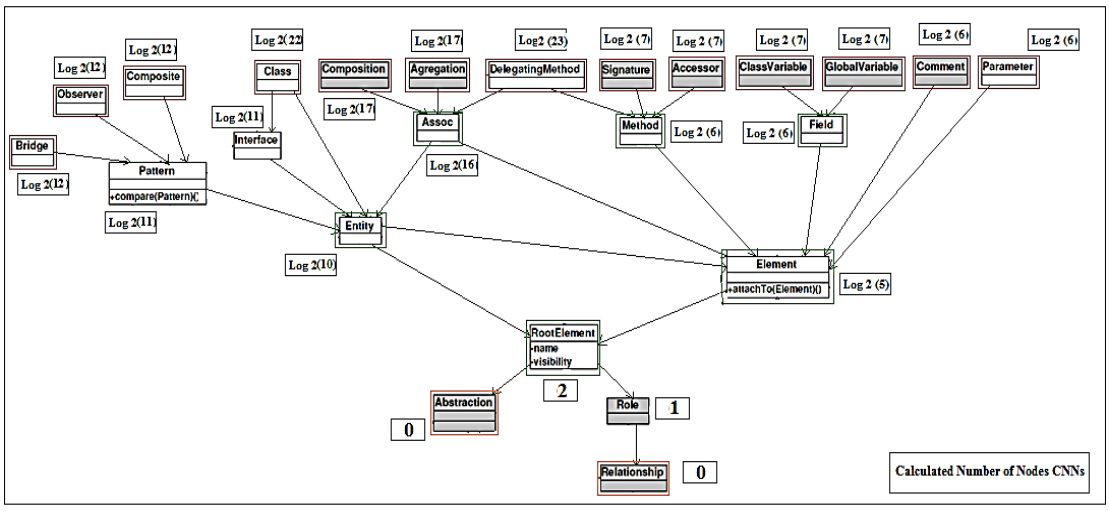
الشكل (12-11) النموذج المترفع PADL

1- نقوم كخطوة أولى بالتحويل إلى البنية الشجريّة، وذلك بعد تحويل العلاقات حسب القواعد الموضحة سابقاً يصبح الشكل كما هو مبيّن في الشكل التالي.



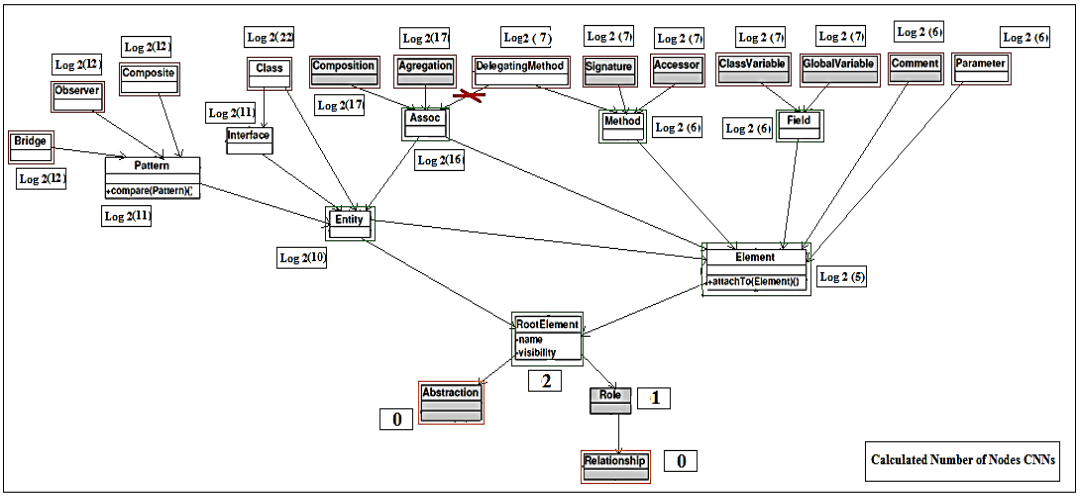
الشكل (12-12) البنية الشجريّة للنموذج المترفع PADL

2- نقوم بحساب الأعداد المحسوبة لكافة العقد كما يوضحه الشكل التالي.

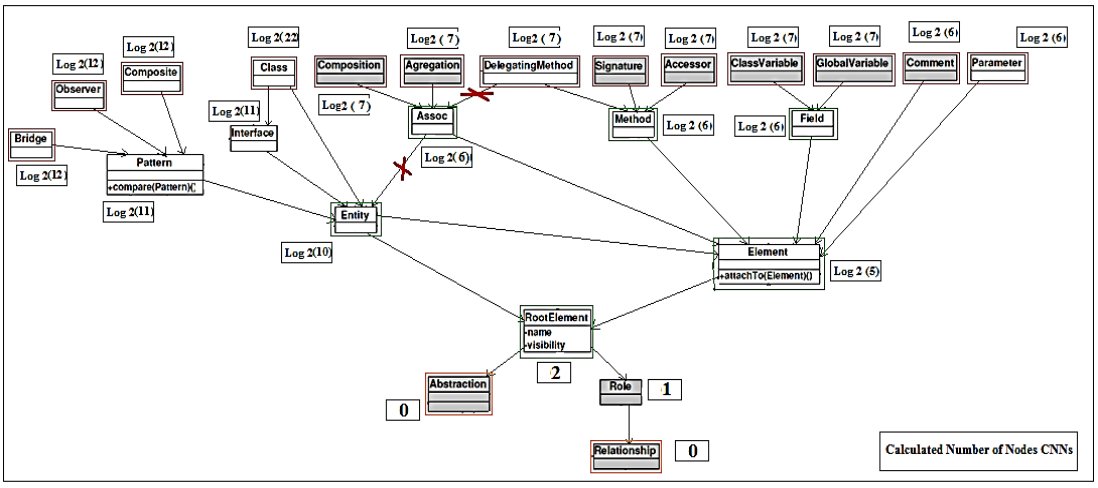


الشكل (12–13) الأرقام المحسوبة للعقد

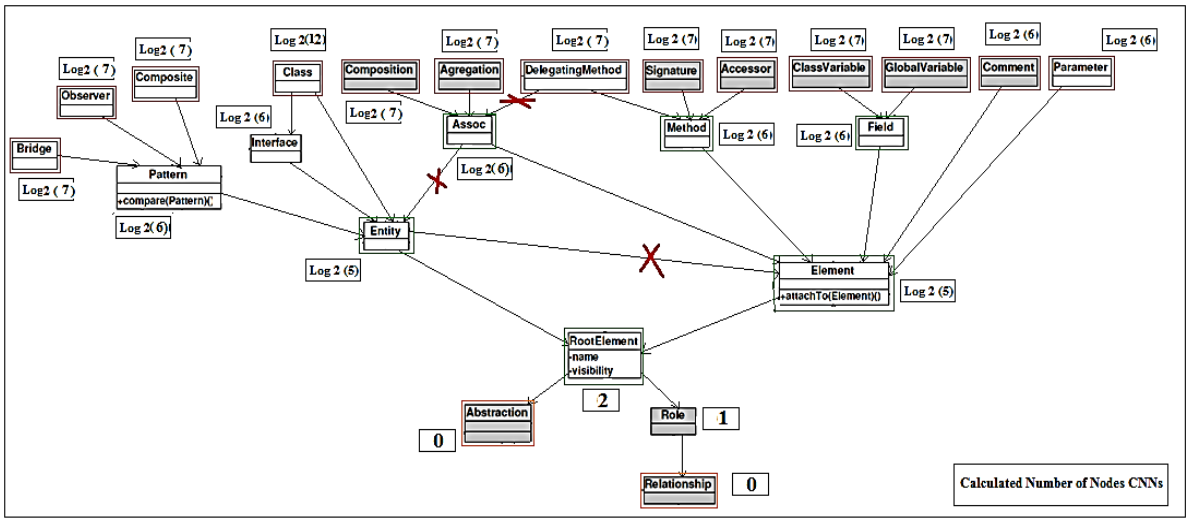
ونبدأ بسلسلة الحسابات للأوزان ابتداءً من العقد ذات الرقم المحسوب الأكبر. والأشكال التالية توضح سلسلة الحسابات.



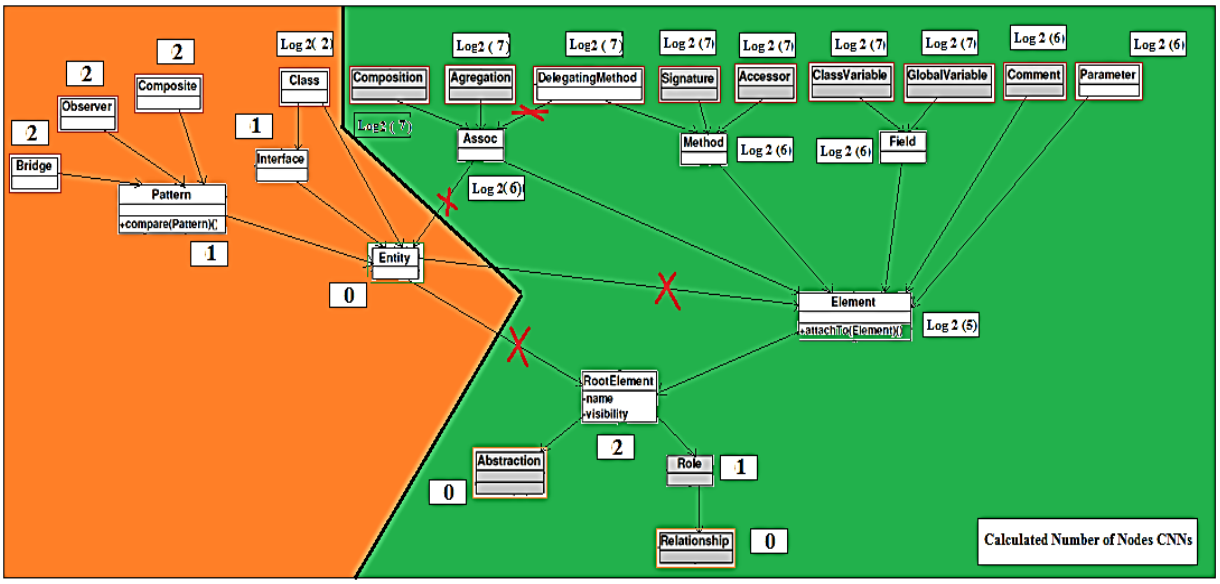
الشكل (12-14) سلسلة الحسابات



الشكل (12-15) سلسلة الحسابات

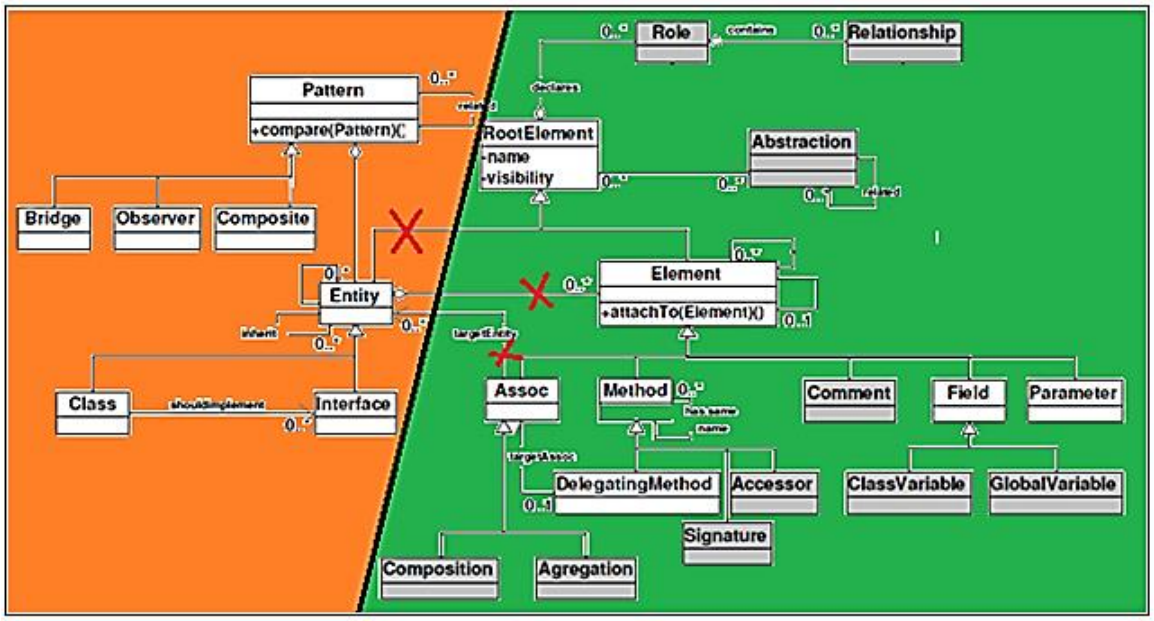


الشكل (12-16) سلسلة الحسابات



الشكل (12-17) نتائج التقسيم على البنية الشجرية لنموذج PADL

وبالعودة إلى الشكل الأصلي لنموذج PADL يكون شكل التقسيم كالتالي.



الشكل (12-18) نتائج التقسيم على النموذج المترفع PADL

4.12. الخلاصة

(قمنا بتطبيق المنهجيّة المُقترحة على نماذج معياريّة عالميّة، هما Henshin و PADL وكان التقسيم الناتج عن المنهجيّة المُقترحة منطقياً يضمن سهولة الفهم والاستخدام والصيانة والمراجعة والتعلّم، وسنأتي في الفصل القادم على دراسة معايير الجودة واخضاع المنهجيّة المُقترحة لها، وطرحها لأخذ آراء الطلاب بعد الممارسة العملية لها في كلا المرحلتين، قبل التقسيم وبعد التقسيم.)

الفصل الثالث عشر
الإحصاءات والتقييم للمنهجيّة المُقترحة

1.13. مقدمة

نالت عملية الاستبيانات والنسب الإحصائية أهمية كبيرة عبر الزمن في الدراسات العلمية، فقد كان لها أثر بالغ في حسم الكثير من الإشكاليات العلمية التي تخضع لمبدأ الارتياب العلمي، إذ أنّ الاستبيانات تعبّر عن رأي الجمهور وهو رأي الأكثرية، وكثيراً ماكانت الدراسات العلميّة تلجأ إلى مثل هذه النسب والاستبيانات لإيجاد الإثبات والدليل على صحة النتائج ولتأمين منصة عمل تُعتبر التجربة أساس عملها، وبالتالي تكون النتائج على درجة عالية من الثقة، كونها خضعت للتجريب وتكون الاستبيانات بمثابة استطلاع للآراء، وبمثابة تغذية راجعة للعمل.

فالاستبيان هو أحد وسائل البحث العلمي المستعملة على نطاق واسع من أجل الحصول على بيانات أو معلومات تتعلق بمسألة علميّة، وتأتي أهمية الاستبيان كأدوات لجمع فهو اقتصادي في الجهد والوقت إذا ما قورن بالمقابلة والملاحظة، فالاستبيان يتألف من استمارة تحتوي على مجموعة من الفقرات يقوم كل مشارك بالإجابة عليها بنفسه دون مساعدة أو تدخل من أحد.

وبالتالي رأينا فيه وسيلة نافعة لأخذ آراء الطلاب حول الممارسة العمليّة التي قاموا بها على استخدام النماذج المترفعة وتشكيل النماذج الخاصة، فقمنا بطرح مجموعة نماذج مترفعة قبل وبعد التقسيم على 90 طالباً في مرحلة ماقبل التخرج وذلك كجزء من مقرر هندسة البرمجيات /3 / من السنة الخامسة في قسم هندسة البرمجيات، في كلية الهندسة المعلوماتية في جامعة دمشق. وعمد الطلاب إلى تنفيذ هذه النماذج المترفعة وتشكيل النماذج الخاصة بهم بهدف تقييمها بطرق عملية. قام الطلاب بتزويدنا بملاحظاتهم من خلال عدة استبيانات تشرح النتائج. الاستبيانات في الملحق /ب/.

2.13. عناصر التقييم

تُعرّف الجودة غالباً بأنها ملائمة الأهداف، وهي خاصيّة مفتاحيّة لتحديد متى نقيّم النوعيّة. ومع زيادة أهمية النمذجة في تطوير البرمجيات، أصبح هناك المزيد من الاهتمام في موضوعات الجودة، في الهندسة المُقادة بالنماذج MDE [83].

الجودة صعبة التعريف، مستحيلة القياس، وغامضة نوعياً [80]. وهي أمر نسبي تابع لرضاء المستخدم. لا توجد أدوات لقياس الجودة بدقة. إنّ المعيار الدولي للجودة البرمجيّة ISO /IES 14598 يُعرّف نموذج الجودة كالتالي:

" مجموعة خاصيات وروابط فيما بينها، تؤمن الأساس لتحديد متطلبات الجودة وتقييم النوعية ".

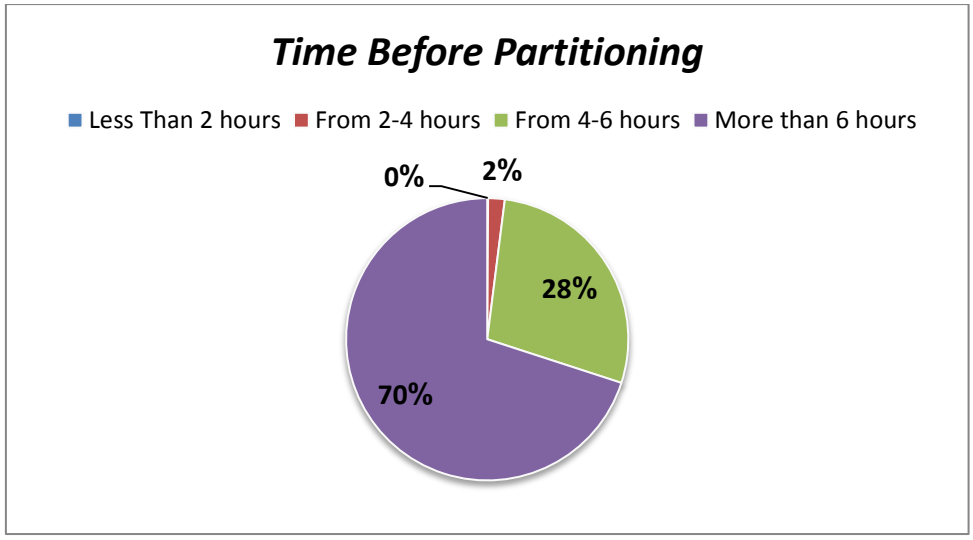
استناداً للأبحاث السابقة في الجودة، فإن الإدارة الشاملة للجودة تتمحور حول ثلاثة محاور رئيسية: المستفيد، العملية، والأفراد [3].

لقياس الجودة بدقة، نحن بحاجة لآراء المستخدمين وتقييماتهم حول منتجنا البرمجي. لذا نعتمد على 3 عوامل لقياس جودة النماذج المترفعة وتقييمها قبل وبعد التقسيم.

- **الزمن:** هو عامل مهم لقياس الجودة فزيادة زمن الاستخدام والفهم تدل على رداءة النموذج، والعكس بالعكس.
- **عدد الضغطات اللازمة لإنشاء عنصر:** فكلما قلّت، كلما دلّ ذلك على سهولة الاستخدام.
- **استخدام الملف المساعد والتوثيق:** فكلما أمكن المستخدم التعامل مع النموذج بعيداً عن التوثيق والملفات المساعدة، كلما دلّ ذلك على وضوح النموذج أكثر.

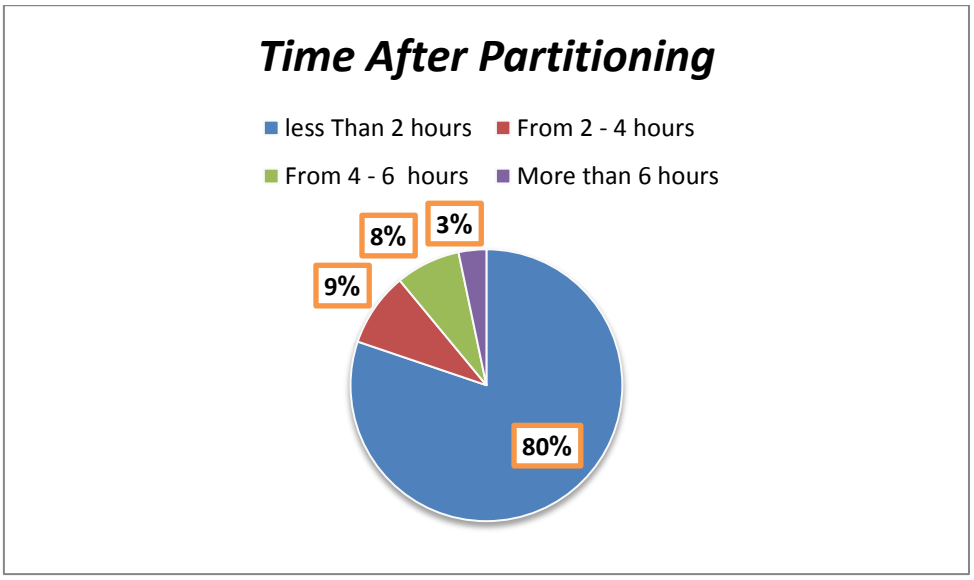
3.13. المخططات الإيضاحيّة

- المخطط الأول ويعكس نسبة الزمن المستخدم للتعامل مع التطبيق وفهمه قبل إجراء التصحيح، ونلاحظ أن النسبة العليا ترتكز على الوقت الطويل نسبياً أكثر من 6 ساعات.



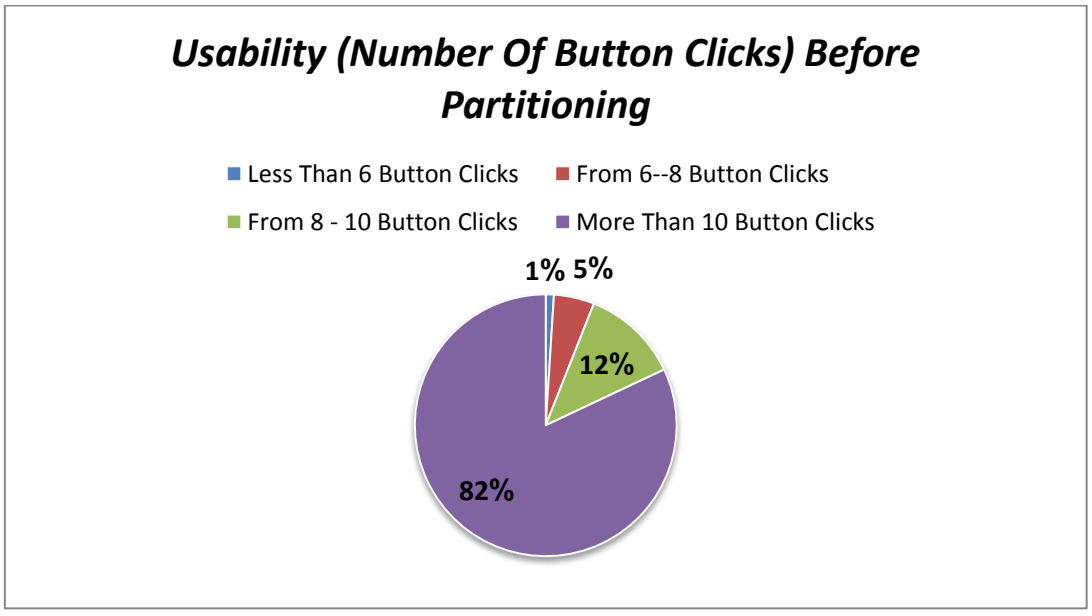
الشكل (13-1) الوقت المستهلك للتعامل النموذج قبل إجراء التصحيح

- المخطط الثاني ويعكس نسبة الزمن المستخدم للتعامل مع التطبيق وفهمه بعد إجراء التصحيح، ونلاحظ أن النسبة العليا ترتكز على الوقت القصير نسبياً أقل من 2 ساعة.



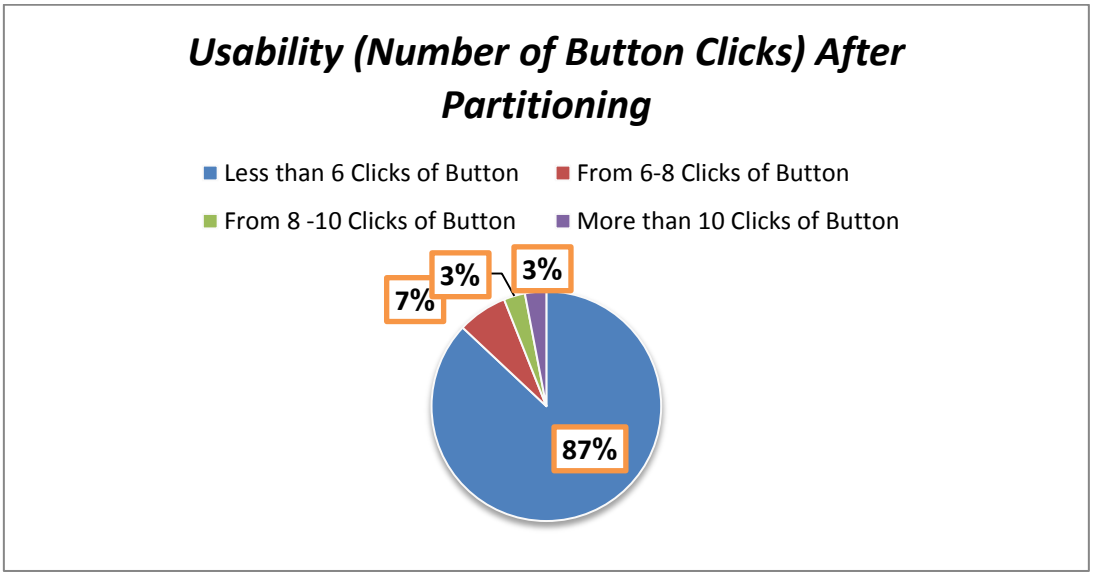
الشكل (13-2) الوقت المستهلك للتعامل مع النموذج بعد إجراء التصحيح

- المخطط الثالث ويعكس مدى صعوبة الاستخدام وذلك بتحديد عدد الضغطات للوصول إلى الكيان المطلوب وخلق علاقاته، وتتركز النسبة العليا في أن عدد الضغطات يزداد عن 10 ضغطات.



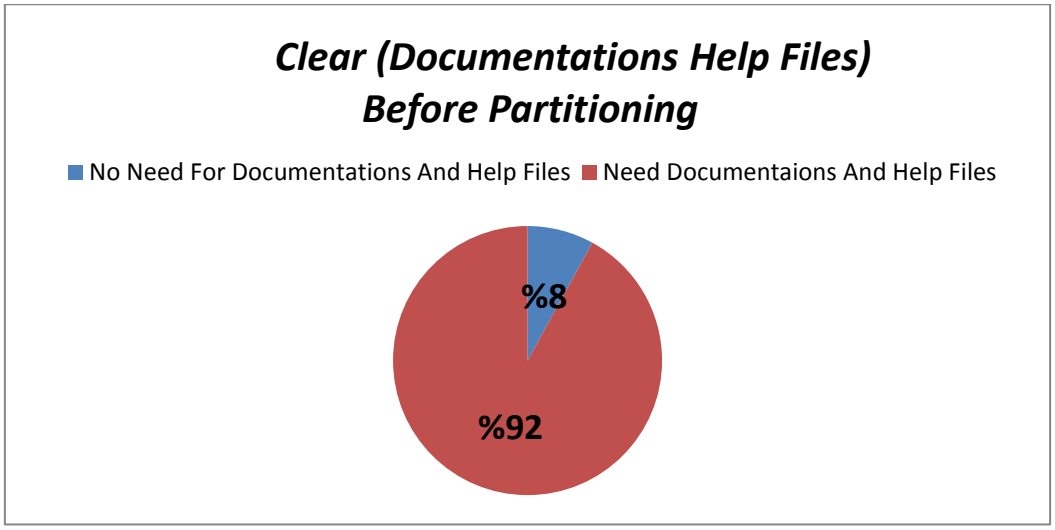
الشكل (13-3) مدى صعوبة الاستخدام من خلال عدد الضغطات قبل التصحيح

- المخطط الرابع ويعكس مدى سهولة الاستخدام وذلك بتحديد عدد الضغطات للوصول إلى الكيان المطلوب وخلق علاقاته، وتتركز النسبة العليا في أن عدد الضغطات يقل عن 6 ضغطات وهذا يُعتبر جودة عالية جداً قياساً مع تصاميم أخرى.



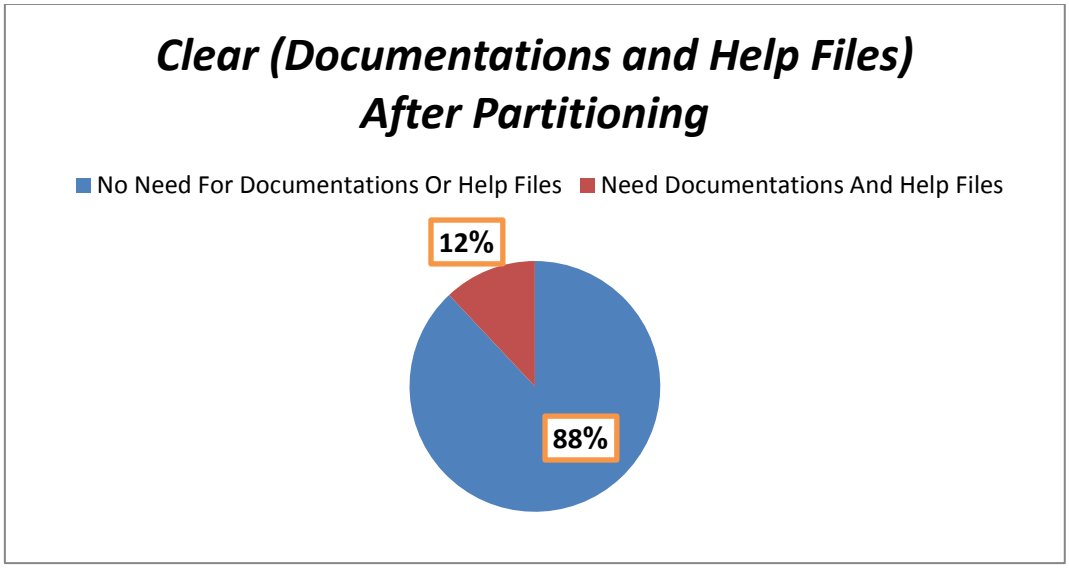
الشكل (13-4) مدى سهولة الاستخدام من خلال عدد الضغطات للوصول للهدف

- المخطط الخامس ويعكس مدى قابلية النموذج للفهم دو الاستعانة بالوثائق والملفات المساعدة أي يعكس مدى وضوح النموذج المطروح. ونجد أن النسبة مرتفعة عند عدم استعمال الملف المساعد أو التوثيق المرافق للتصميم.



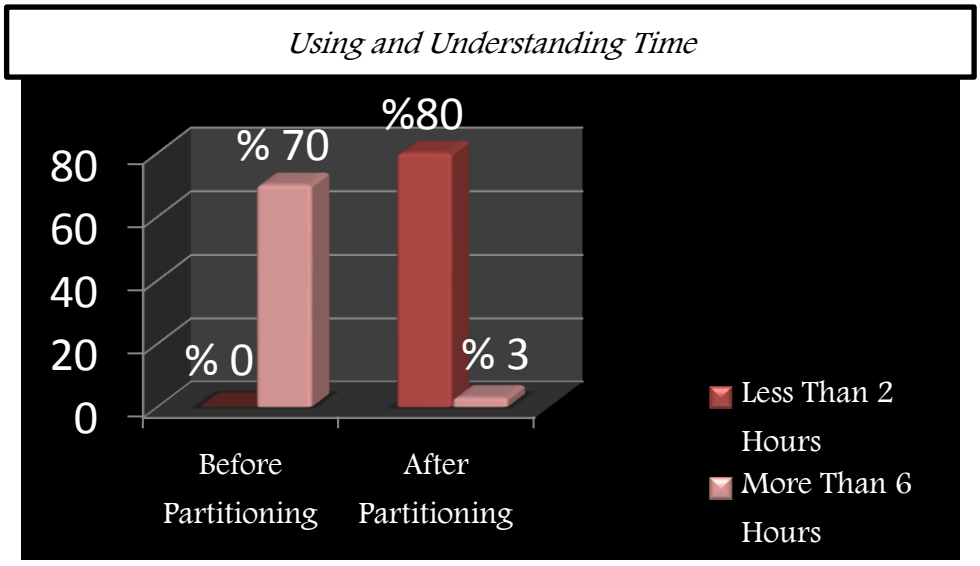
الشكل (13-5) مدى غموض النموذج واستعمال الوثائق والملفات المساعدة

- المخطط السادس ويعكس مدى قابلية النموذج للفهم دون الاستعانة بالوثائق والملفات المساعدة أي يعكس مدى وضوح النموذج المطروح. ونجد أن النسبة مرتفعة عند عدم استعمال الملف المساعد أو التوثيق المرافق للتصميم.



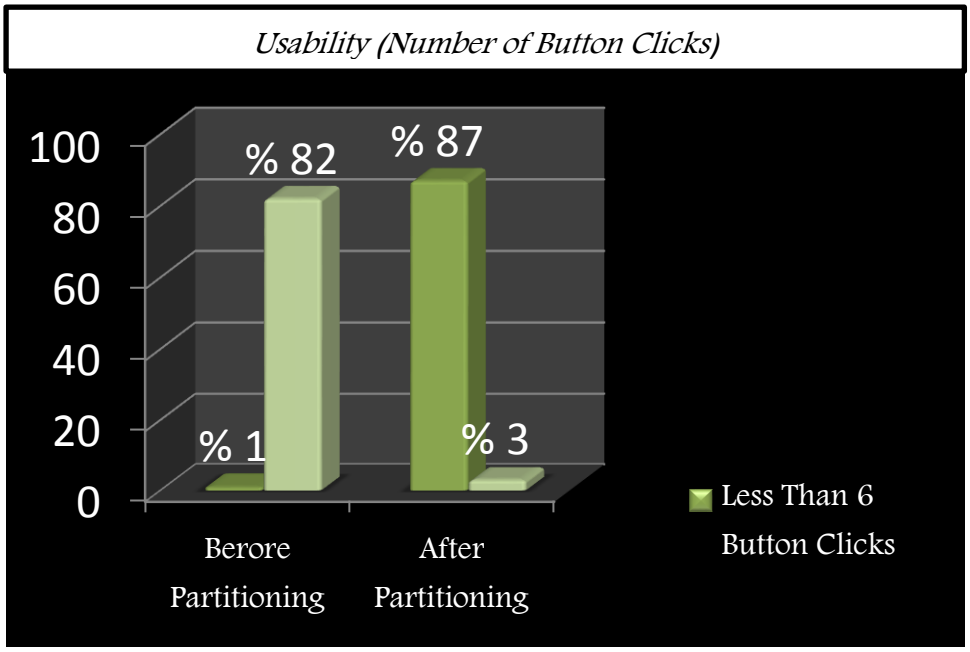
الشكل (13-6) مدى وضوح النموذج وعدم استعمال الوثائق والملفات المساعدة

وبالتالي تمّ ارتفاع نسبة التحسين من 0% إلى 80% وهي تؤكد اختصار الوقت من أكثر من 6 ساعات إلى أقل من 2 ساعة. المخطط التالي يوضح النسب.



الشكل (13-7) ارتفاع نسبة توفير وقت الفهم والاستخدام

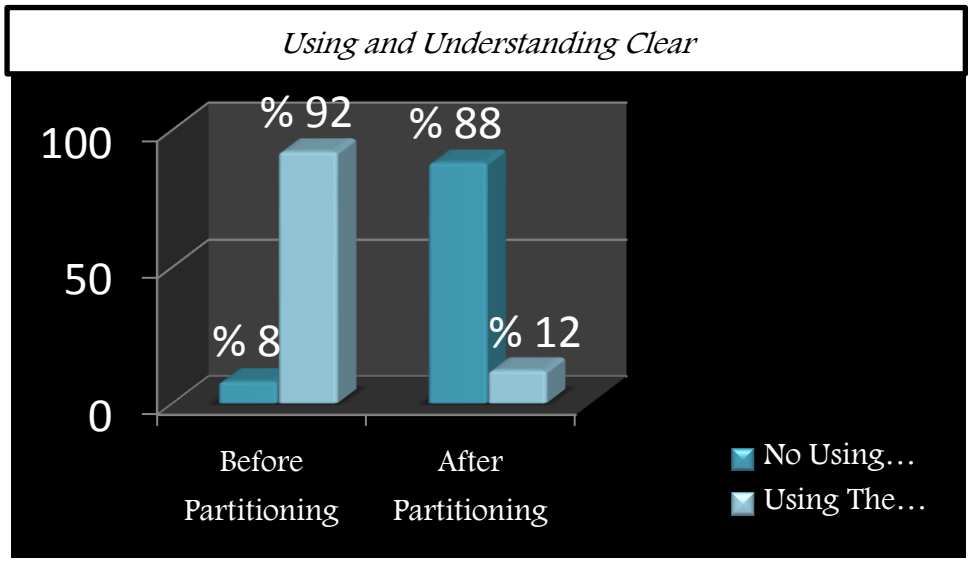
وتمّ ارتفاع نسبة سهولة الاستخدام من 1% إلى 87% وذلك بانخفاض عدد الضغطات المطلوبة، ويبيّن المخطط التالي ارتفاع نسبة سهولة الاستخدام. المخطط التالي يوضح المفهوم.



الشكل (13-8) ارتفاع نسبة سهولة الاستخدام

وتمّ ارتفاع نسبة وضوح التصميم من 8% إلى 88% ولم تعد هناك حاجة إلى استخدام الوثائق والملفات المساعدة، فالتصميم الجيد يغني عن التوثيق والشرح.

والشكل التالي يوضّح الفكرة.



الشكل (13-9) ارتفاع نسبة وضوح التصميم بعد إجراء التصحيح

4.13. الخلاصة

(تتحسّن عوامل الزمن وسهولة الاستخدام والوضوح بشكل واضح وكلها تعتبر البرهان القوي على تحسّن الجودة. وبذلك نكون قد حققنا أهداف المشروع بزيادة جودة النموذج المترفع بعد تقسيمه إلى نماذج مترفعة جزئية. تمّ نشر ورقة علميّة بالعمل، وهي في الملحق[6].)

[6] - الملحق /ج/ يحتوي على الورقة العلميّة التي تمّ تقديمها للنشر في المجلة العلميّة المكمة JCST.

الجزء الرابع

التطبيق العملي وواجهات التنفيذ

- التعريف بأداة النمذجة
- التحقيق وواجهات التنفيذ

الفصل الرابع عشر
التعريف بأداة النمذجة المستخدمة

1.14. مقدمة

سنستعمل أداة رسومية خاصة بـ Eclipse [96] مطورة عن مشروع Graphical Editing Framework (GEF) وقد تم بناؤها باستعمال Eclipse Modeling Framework (EMF) وهي اطار عمل يتيح امكانية تصميم نماذج Meta models انطلاقاً من مخططات UML حيث يتم بناء النموذج بمخطط UML ومن ثم توليد رمازالنموذج وملف يمثل المخطط بشكل شجري لاحقته Ecore ومنه يمكن توليد XML File. وبالتالي:

EMF: هي أداة للنمذجة وتوليد الرماز من أجل بناء أدوات وتطبيقات مبنية على نموذج البيانات الهيكليّة، حيث تقوم هذه الأداة بتوليد كود بلغة Java انطلاقا من ملف XML ، وتشكل البنية التحتية التي تستخدم لبناء الـ Meta model.

GEF: هي أداة مسؤولة عن توليد المحرر الرسومي والتعامل مع مكتبات البيانيات وهي من نموذج MVC. ثم تأتي الـ GMF لتولد المكونات الرسومية التي نحتاجها في المحرر الرسومي الذي نريد أن نبنيه ،وذلك بالإعتماد على EMF ، حيث نستطيع ان نحدد خصائص المكونات من (شكل ، لون ، حجم ، سحب وافلات ، شكل ايقونات و.....) ومن ثم بناء الـ Diagram الخاص بنا ،و إضافته كـ plug in ضمن الـ Eclipse.

2.14. مكونات GMF

تتكون من مكونين أساسيين هما :

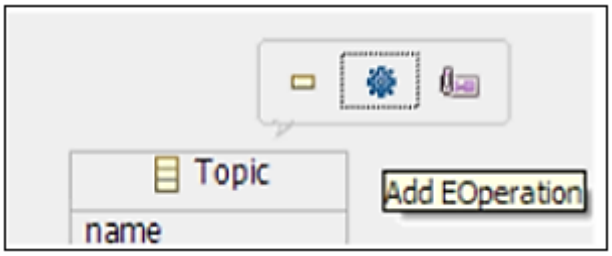
1.2.14. ساحة التنفيذ Run Time

وهي مجموعة من إطارات العمل التي تسهل عملية تطوير التحرير البياني:

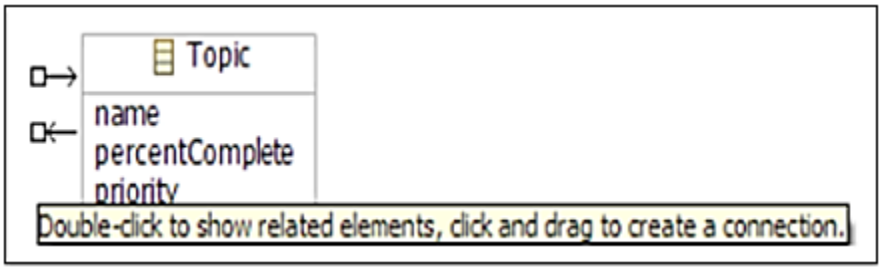
- المكونات القابلة لإعادة الإستخدام و المبينة مسبقاً .
- واجهة موحدة من أجل وصف الـ Diagram .
- التحكم بالأوامر التي تربط EMF مع GEF .

- تمكن من فتح Diagram مبني مسبقا و التعديل عليه.

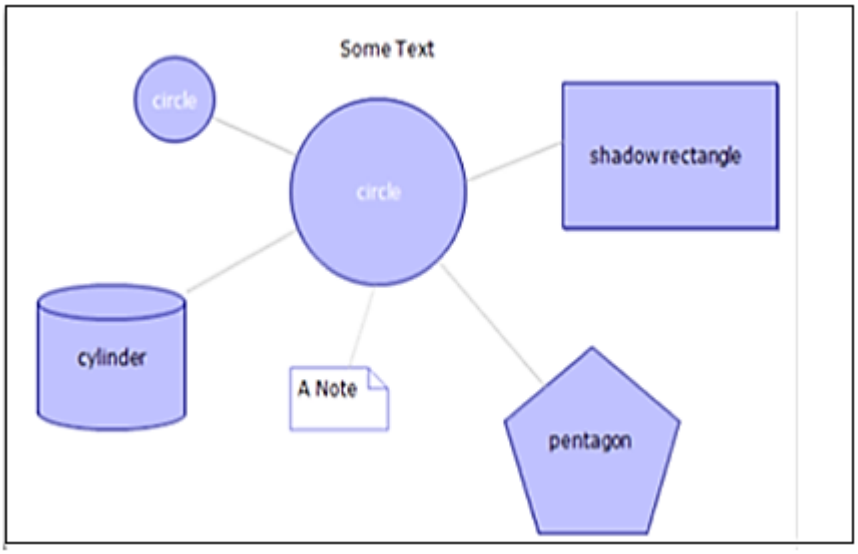
وفيما يلي بعض الأشكال التوضيحة.



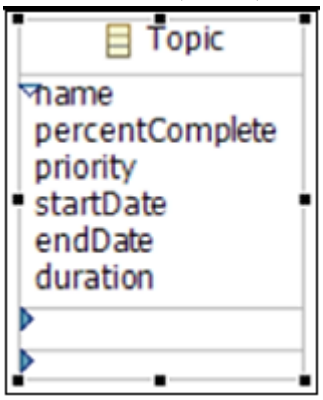
الشكل (14-1) شريط الأفعال



الشكل (14-2) معامل التواصل

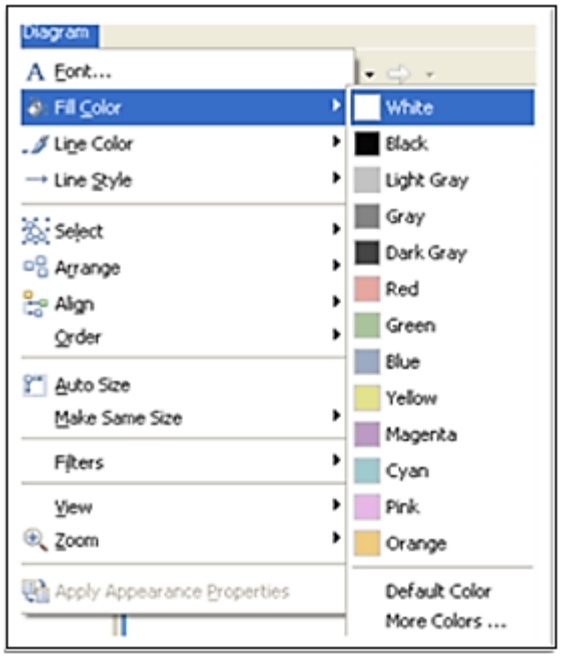


الشكل (14-3) الأشكال الهندسية

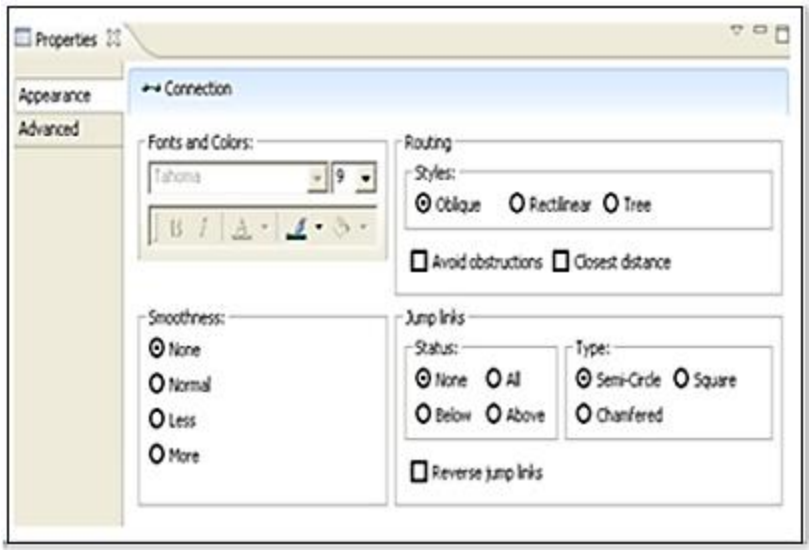


الشكل (14-4) إمكانية طي الأجزاء

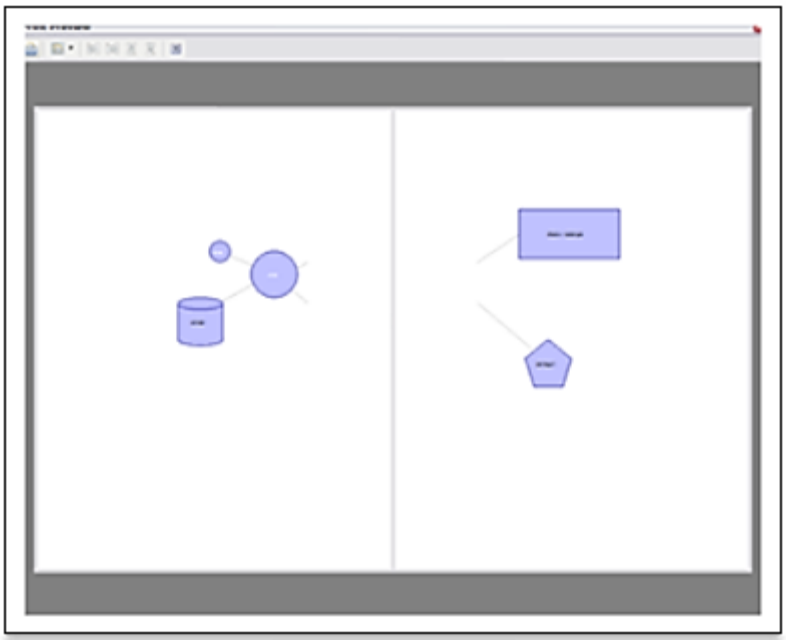
الشكل (14-5) شريط الأدوات



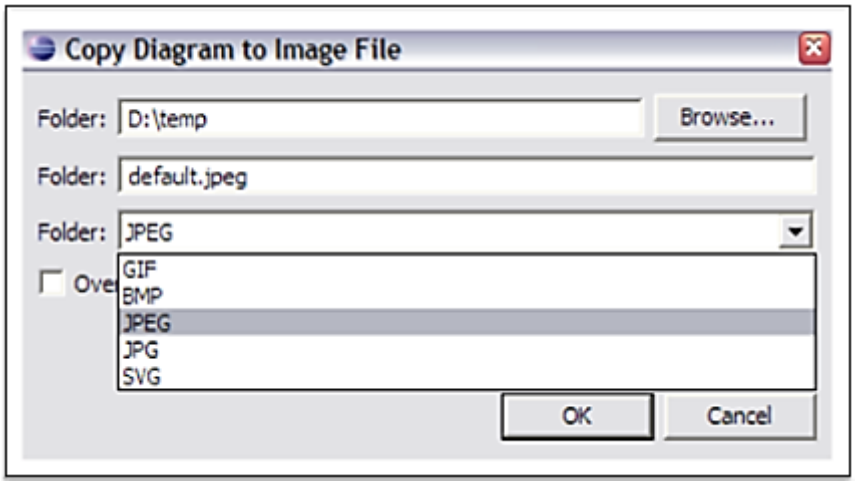
الشكل (14-6) الأفعال



الشكل (14-7) إظهار الخصائص



الشكل (14-8) معاينة ما قبل الطباعة



الشكل (14-9) نسخ المخطط كصورة

2.2.14. ساحة توليد الأدوات Generation Tools

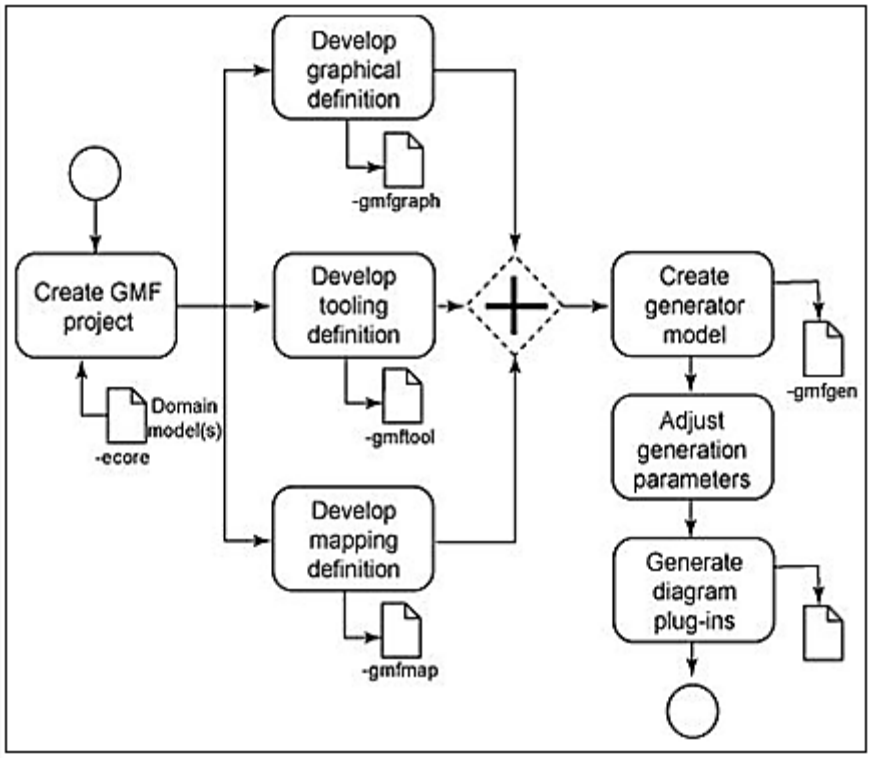
- النماذج بأنواعها التي ستستخدم من أجل بناء الـ Application.
- مرحلة توليد الرماز.
- اتاحة خيارات من اجل توليد الكود بحيث يتناسب مع المتطلبات.

3.14. الهدف من الأداة

- الإستفادة من منهج النماذج المُقادة بالنماذج واستخدامها على شكل محرر رسومي.
- المحافظة على الفصل ما بين سوية النماذج المترفعة والنماذج الخاصةببناء المحرر.

- تأمين مرونة في الرماز المولد، من خلال السماح بتحديد الهدف الموجه نحوه التطبيق مثلRCP.

4.14. المخطط العام للتوليد



الشكل (14–10) المخطط العام للتوليد

5.14. شرح المخطط

Domain Model (.ecore)

- هو نموذج مترفع يمثل القواعد الخاصة بالـ MDA .
- يوصّف مكونات التطبيق الذي نريد بناءه والعلاقات فيما بينها.
- يوصف القيود على المكونات.
- يشكل هذا النموذج العمود الفقري في عملية بناء الـ Application.

Graphical Definition (.gmfgraph)

يحدد هذا الملف الخصائص الرسومية للعقد والروابط وغيرها ويتم فيه تعريف المعلومات التالية:

- Figures: يحوي الأشكال والألوان والخطوط وغيرها من الخصائص التي تستعملها العقد والروابط.
- Nodes: كل صف من الصفوف يكون عقدة في هذا الملف.

- Links: الروابط بين الصفوف.
- Compartments: في حال إحتوت أحد العقد على عقد أخرى.

Tooling Definition (.gmftool)

يحدد هذا الملف الأدوات الرسومية ويتم فيه بناء ما يلي:

- Palette
- Tools (e.g. creation)
- Menus
- Actions
- Toolbars

Mapping Model (.gmfmap)

و هو نموذج لتحديد العلاقة ما بين:

- Domain elements (*.ecore)
- Graphical elements (*.gmfgraph)
- Tooling elements (*.gmftool)

أي أنه العقدة بالصف الموافق لها مع الأدوات الرسومية اللازمة لعرضها.

Generation Model (.gmfgen)

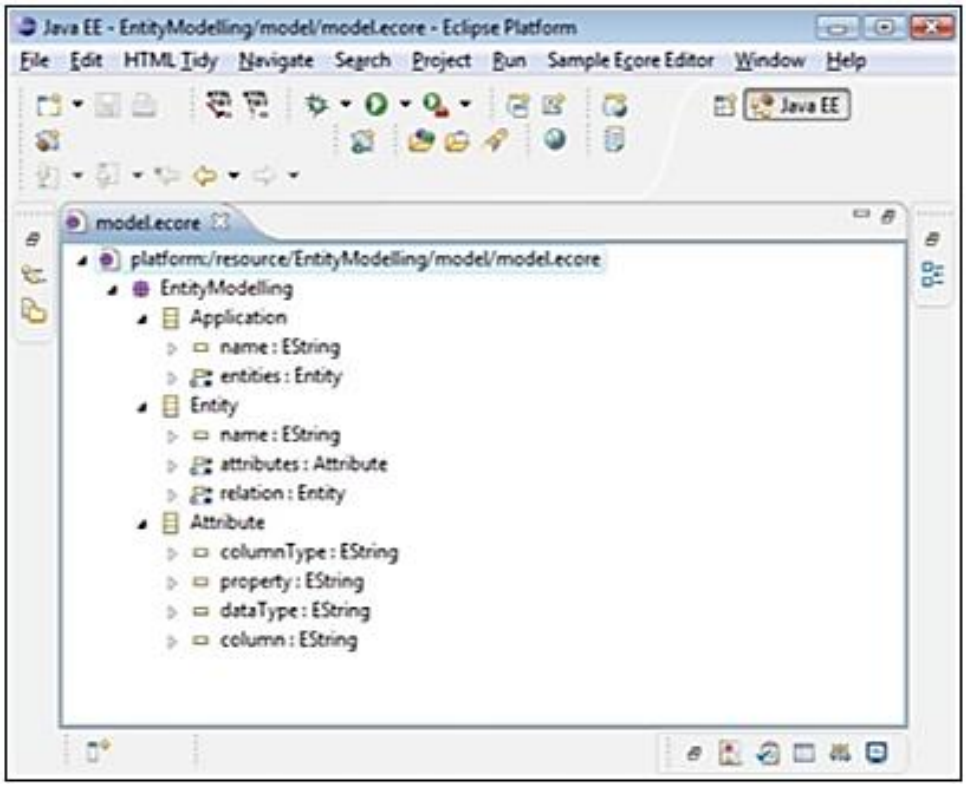
- وهو نموذج يستخدم من أجل تحديد code generation parameters.
- وهو ينتج عن Mapping Model.
- يحوي كل المعلومات من الملفات السابقة.

Diagram Code Generation

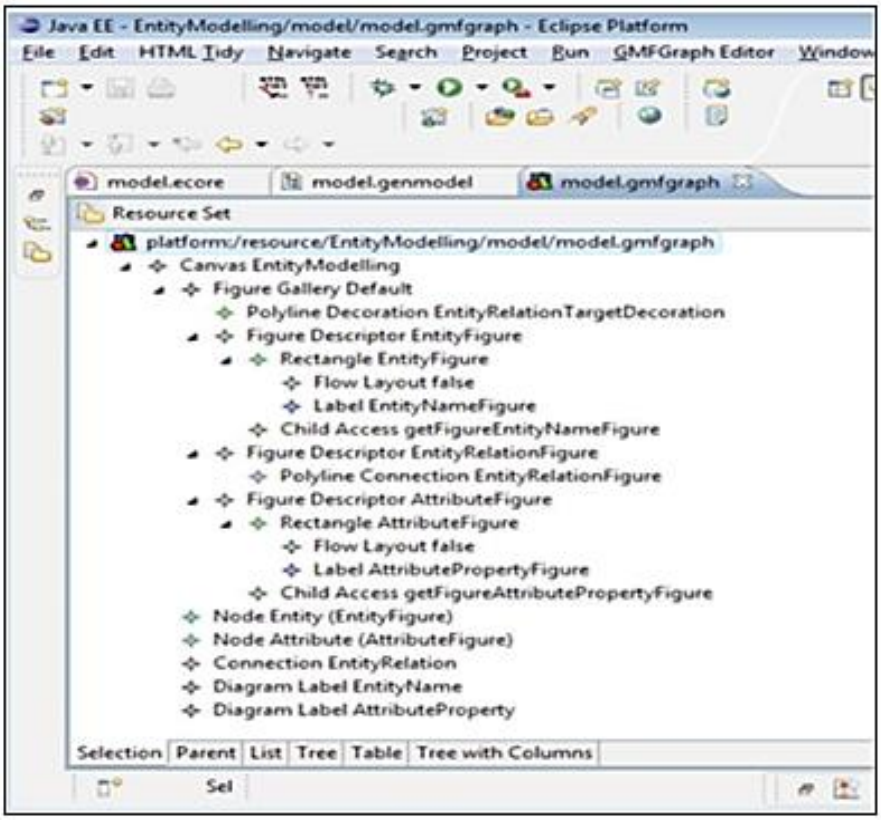
في النهاية وبعد بناء النموذج المولّد وتحديد خصائصه ، تتم عملية توليد الكود بلغة JAVA Emitter Templates (JET) ، وتسمى هذه العملية بـ model-to-text transformation .

6.14. مثال بسيط يشرح ما سبق

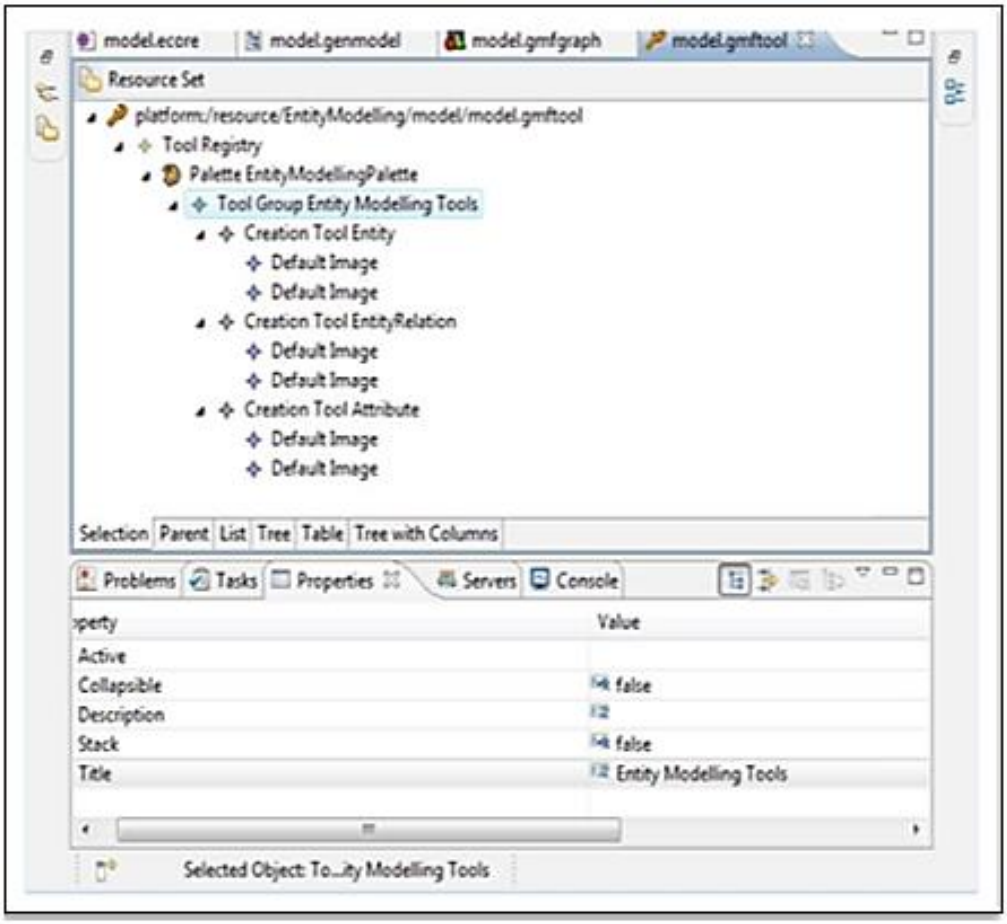
نريد بناء النموذج المترفع لعملية ربط كيانات (Entities) مع بعضها مع وجود واصفات ضمن هذه الكيانات وبالتالي يمكن الوصول إلى النموذج (Model) الذي نريد عن طريق تخصيص هذه الكيانات، وسيتم في مثالنا بناء كيانين هما الزبون والطلبية والربط بينهما بعلاقة بالإضافة إلى الواصفات لكل منهما. وفيما يلي سلسة من الأشكال التي تظهر تسلسل العمل.



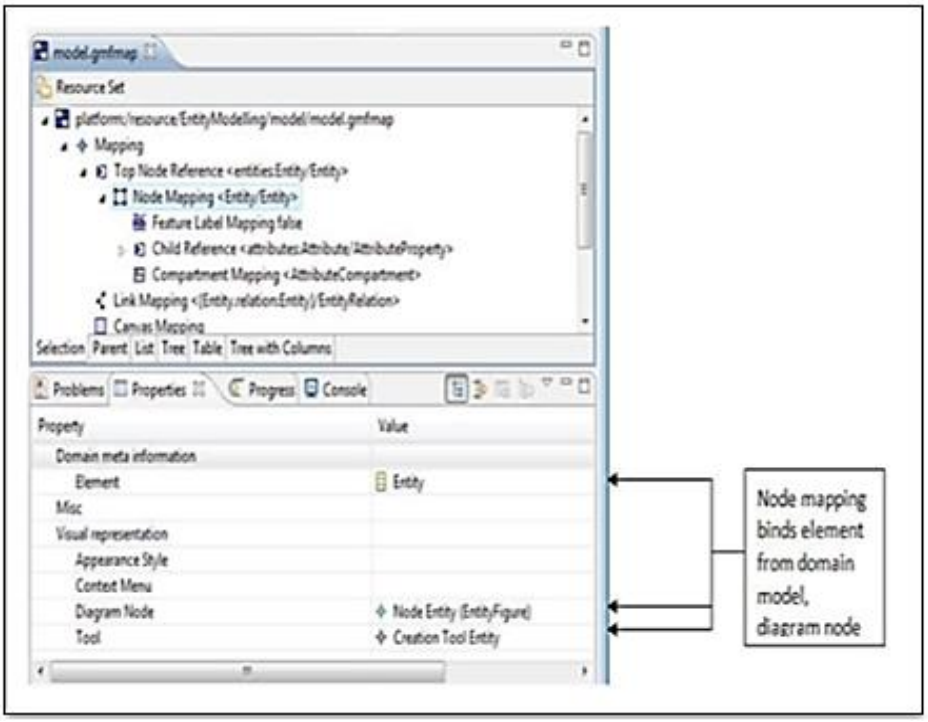
الشكل (14-11) نموذج Ecore



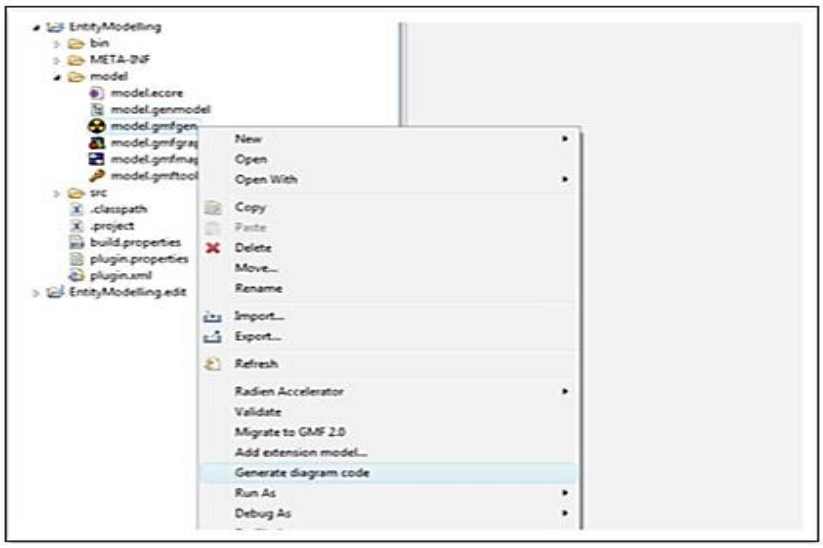
الشكل (14-12) gmfgraph



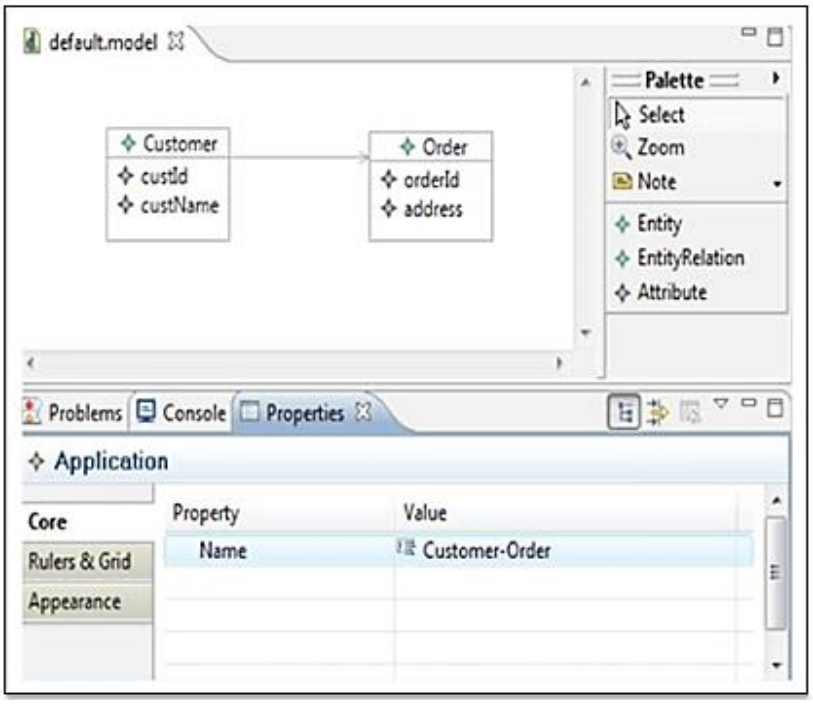
الشكل (13-14) gmftool



الشكل (14-14) gmfmap



الشكل (14-15) توليد الرماز الخاص بالمخطط



الشكل (14-16) النموذج النهائي

7.14. الخلاصة

(وهكذا نجد أنّ العمل على أداة GMF تسهل عملية بناء وتطوير المحررات الرسومية وتسلك نفس منهج EMF وهو عبارة عن نماذج مُقادة في بناء الواجهات.)

الفصل الخامس عشر
التحقيق وواجهات التنفيذ

1.15. مقدمة

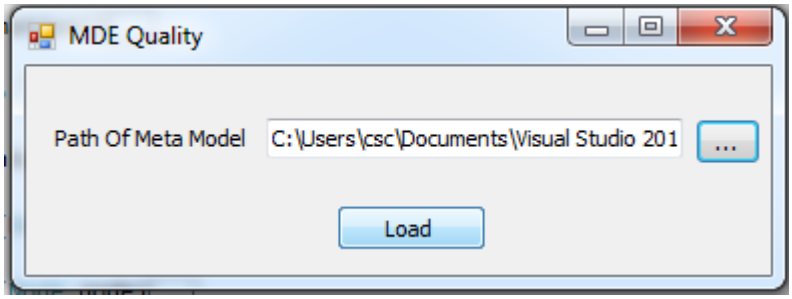
نقوم في هذا الفصل باستعراض واجهات التنفيذ وذلك باختبار النظام على مجموعة من النماذج المترفعة المعيارية العالمية ونستعرض النموذج المترفع قبل التقسيم وبعده.

2.15. النماذج المعيارية وآلية تقسيمها

1.2.15. نموذج PADL

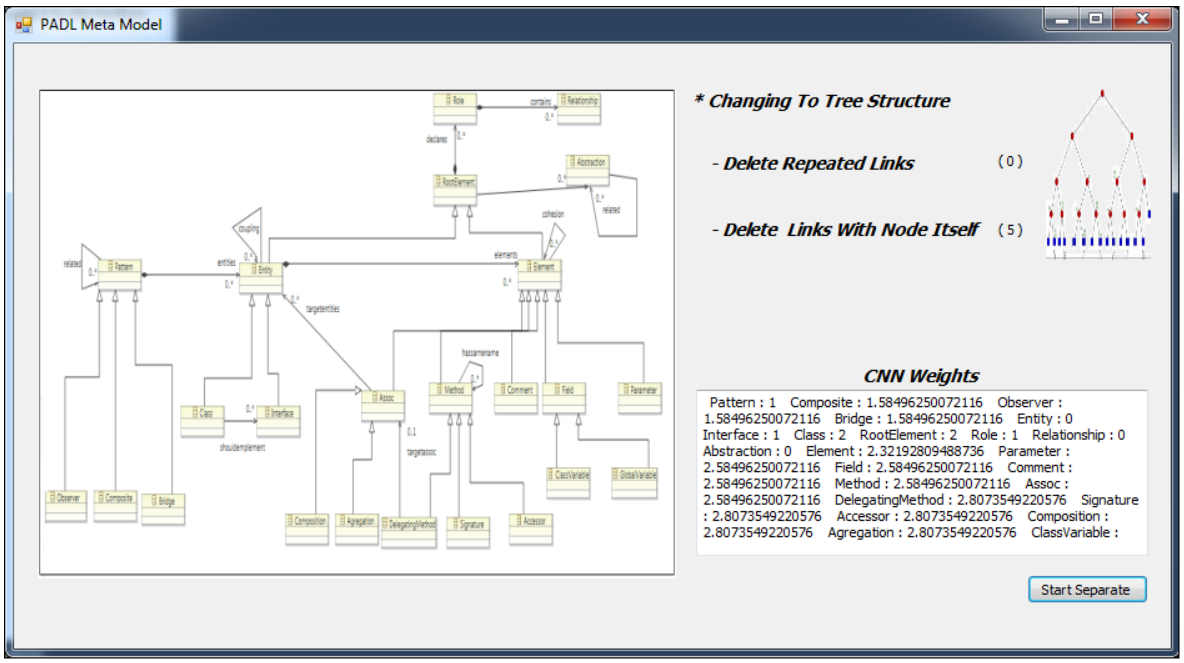
هو نموذج مترفع للغة توصيف السويات التجريديّة والنماذج وتعتمد المبادئ الغرضية التوجّه، وسلسلة الأشكال تبيّن التحميل والتنفيذ.

الشكل الأول ويبيّن اختيار النموذج المترفع المراد تقسيمه.



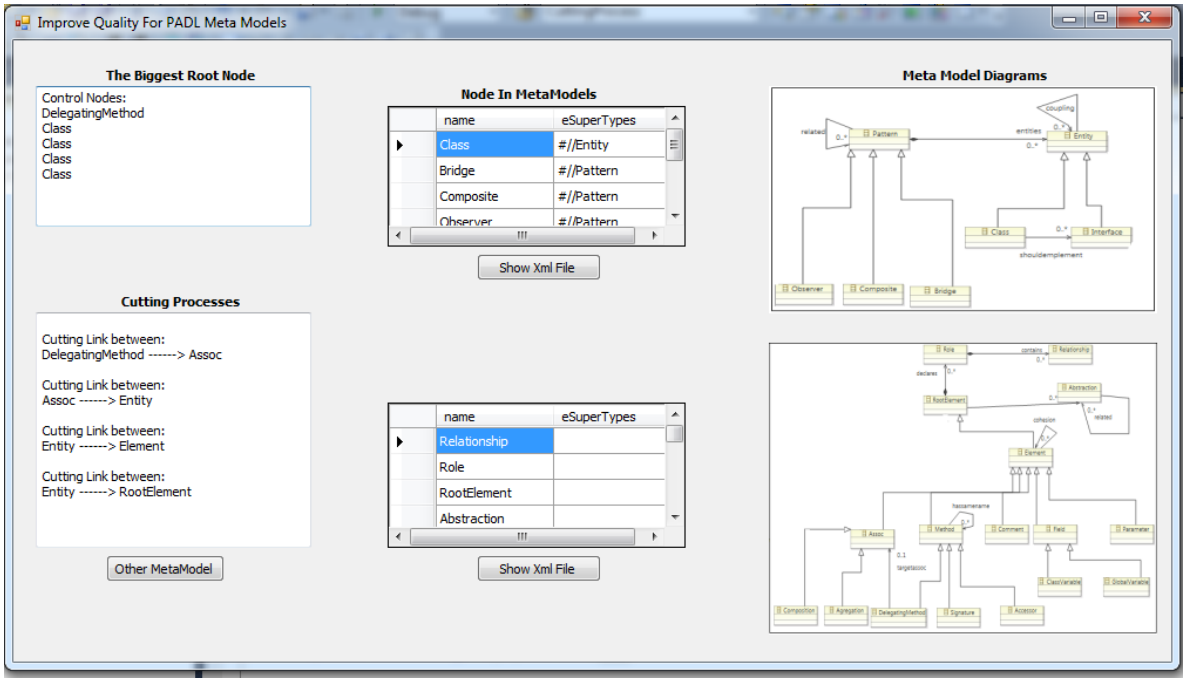
الشكل (15-1) واجهة اختيار النموذج المترفع المطلوب تقسيمه.

والشكل الثاني يبيّن المخطط الذي يمثل النموذج المترفع الذي اختاره المستخدم ويبيّن سلسلة الحسابات العددية لمفهوم الرقم المحسوب ***CNN*** لكل عقدة من العقد الموجودة في النموذج المترفع.

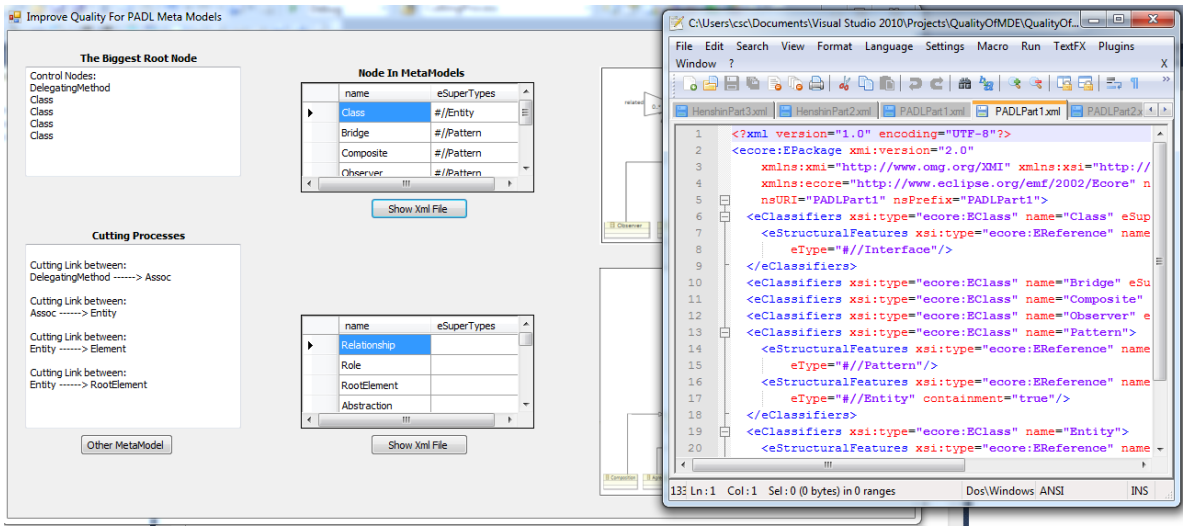


الشكل (15-2) واجهة تعرض المخطط والحسابات العددية *CNN*.

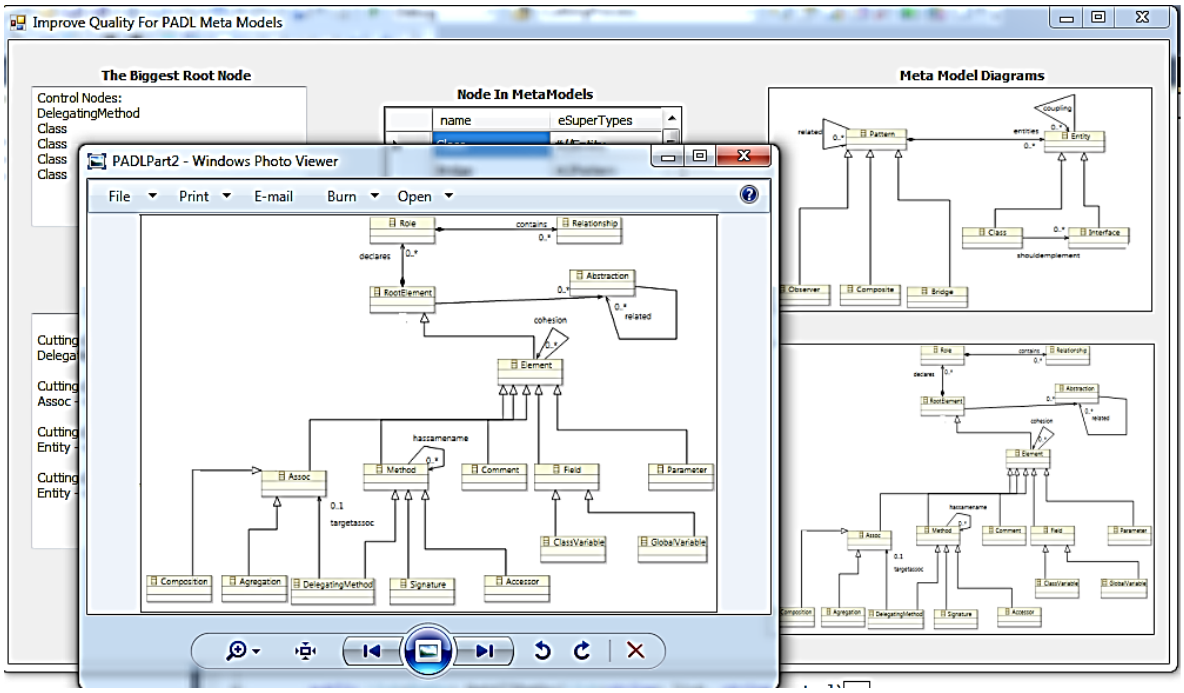
الشكل التالي يوضح العقد التحكمية التي اتُخذت للتقسيم في كل مرة، ويبيّن الروابط التي تمّ اقتطاعها بين العقد، ويبيّن النماذج المترفعة الجزئية التي آل إليها النموذج المترفع بعد التقسيم. يعرض ملفات العقد الموجودة في ملفات XML الجزئية وكذلك المخطط البياني لكل جزء على الترتيب.



الشكل (15-3) واجهة تعرض النماذج المترفعة الجزئية التي تمّ تقسيمها من النموذج المترفع الكلي.



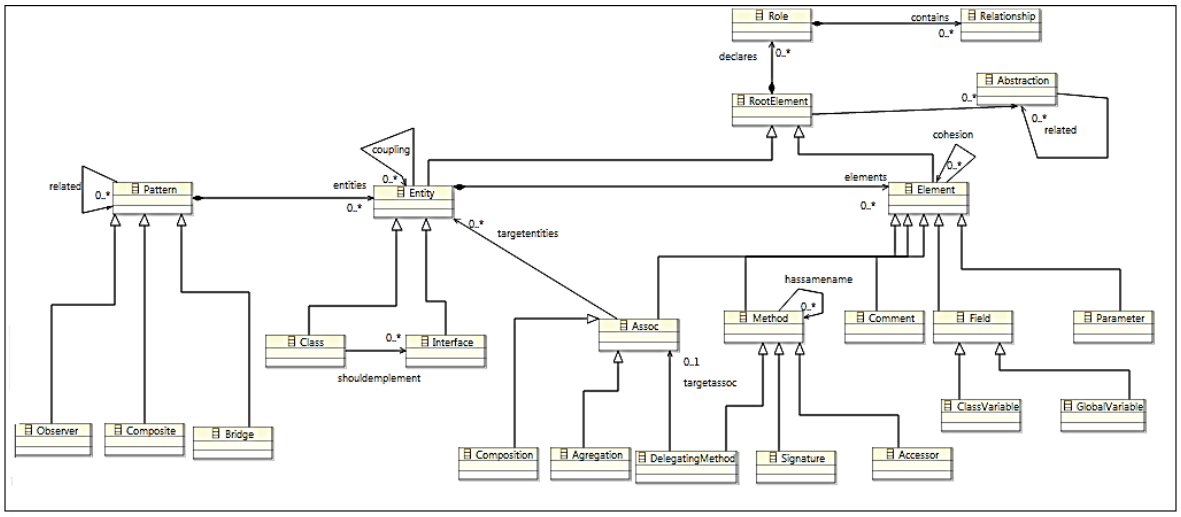
الشكل (15-4) ملف XML لأحد النماذج المترفعة الجزئية.



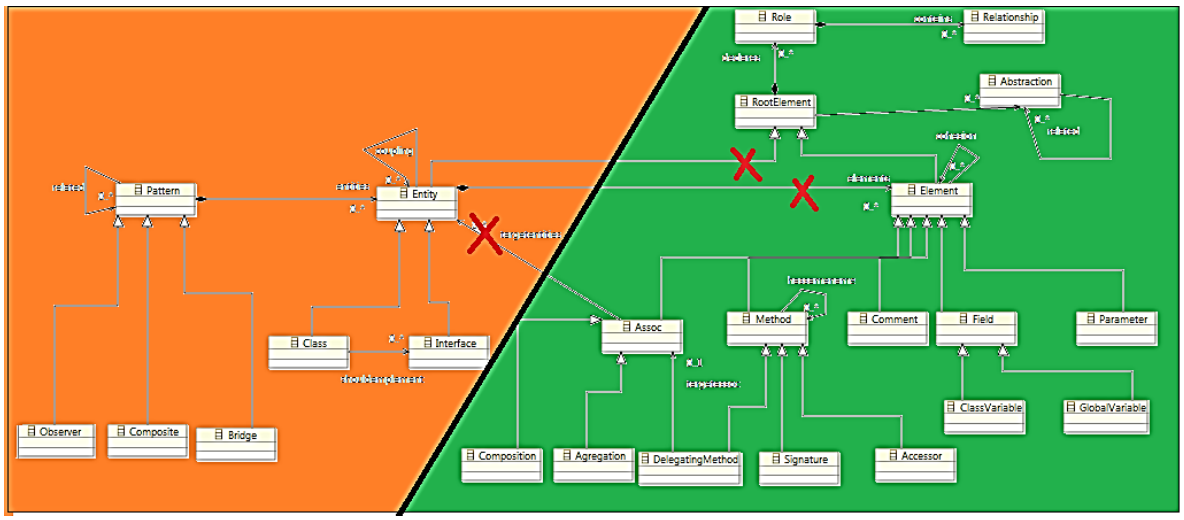
الشكل (15-5) مخطط النموذج المترفع الجزئي.

2.2.15. وجهة النظر حول تقسيم نموذج PADL المُقترح

نتج عن التقسيم نموذجان مترفعان جزئيان، الأول يختص بالعناصر الغرضية التوجّه من توابع وعناصر ومتحولات والعلاقات فيما بينها، والثاني يختص بالنماذج التصميميّة. فالنماذج الجزئية الناتجة أكثر تخصصاً وأسهل استخداماً.



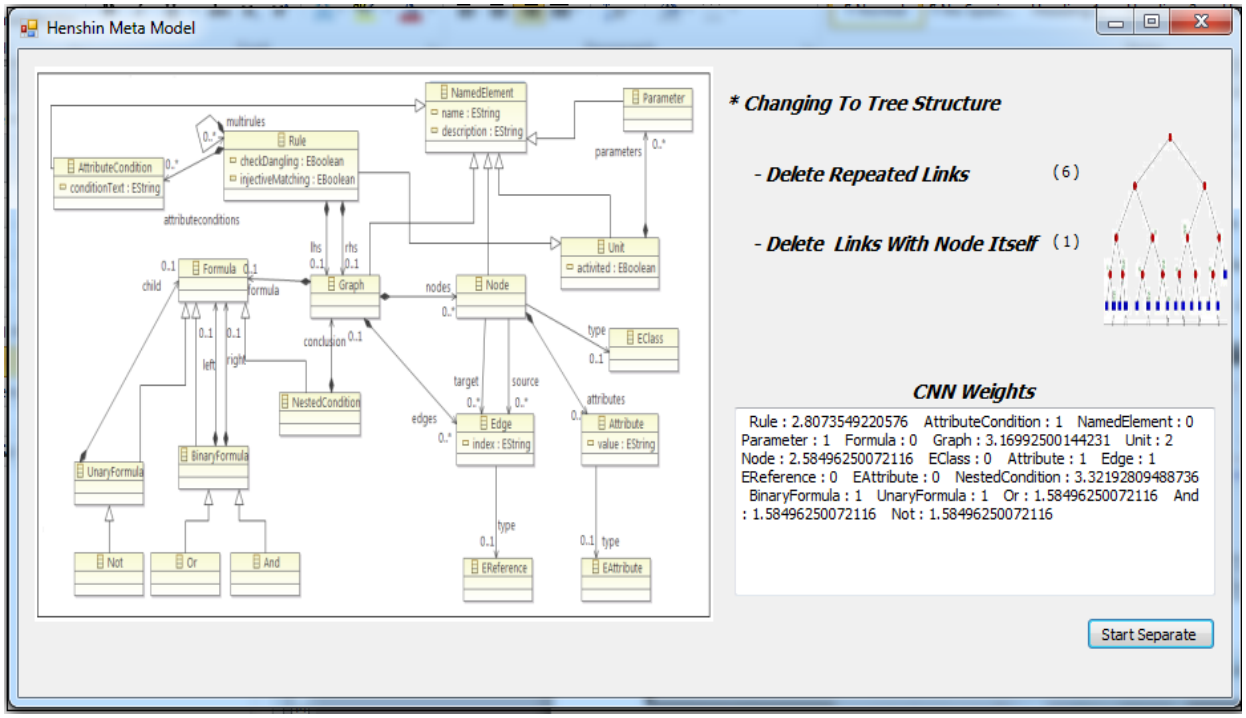
الشكل (15-6) النموذج المترفع PADL قبل التقسيم.



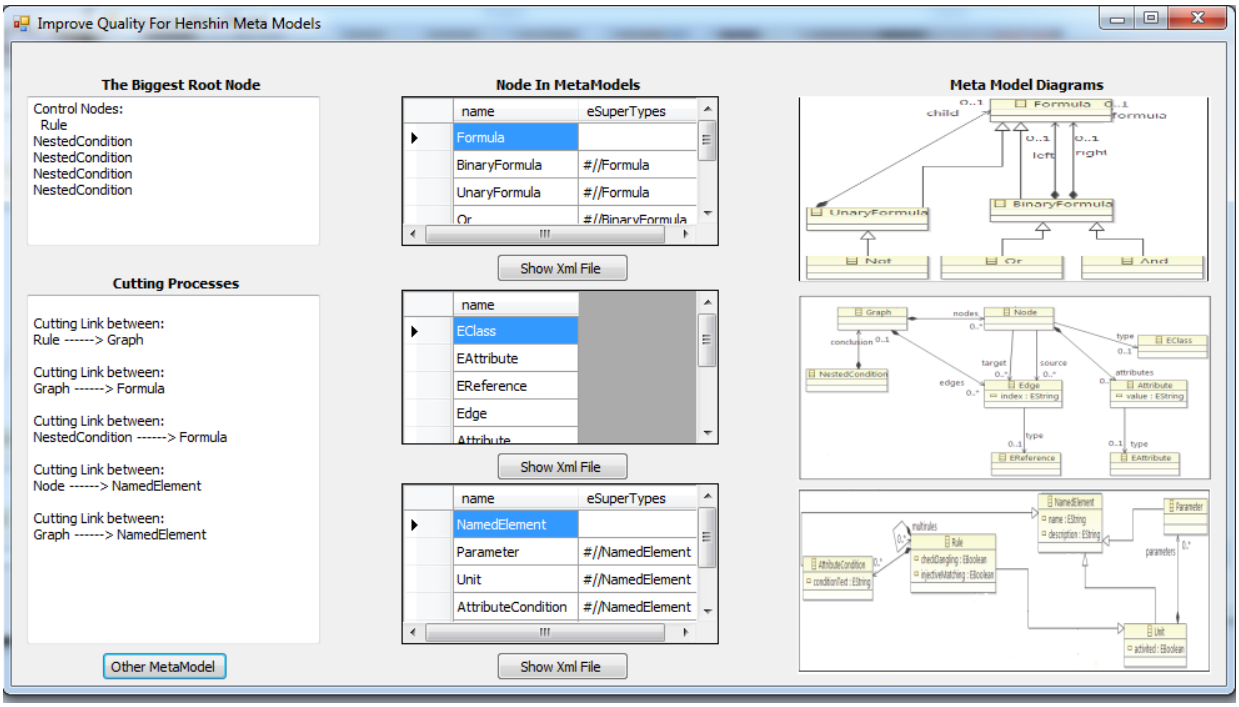
الشكل (15-7) النموذج المترفع PADL بعد التقسيم.

3.2.15. نموذج Henshin

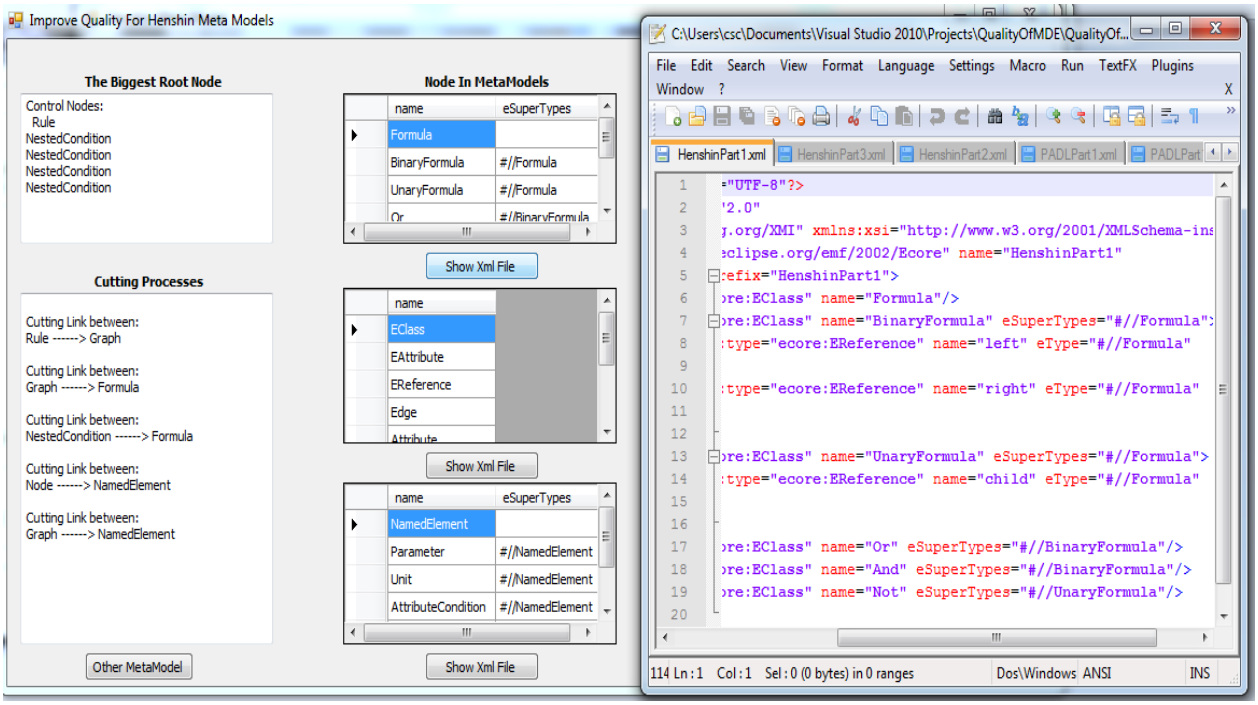
نستعمل جزء من نموذج مترفع لقواعد التحويل المنمذجة من قبل نموذج Henshin، وهو نموذج مترفع لأداة تهدف إلى النمذجة وتحقق انتقالات لغة النمذجة وإعادة تشكيل البيان. وفيما يلي مجموعة لصور الواجهات الخاصة باستعراض النموذج المترفع وكذلك نتائج تقسيمه.



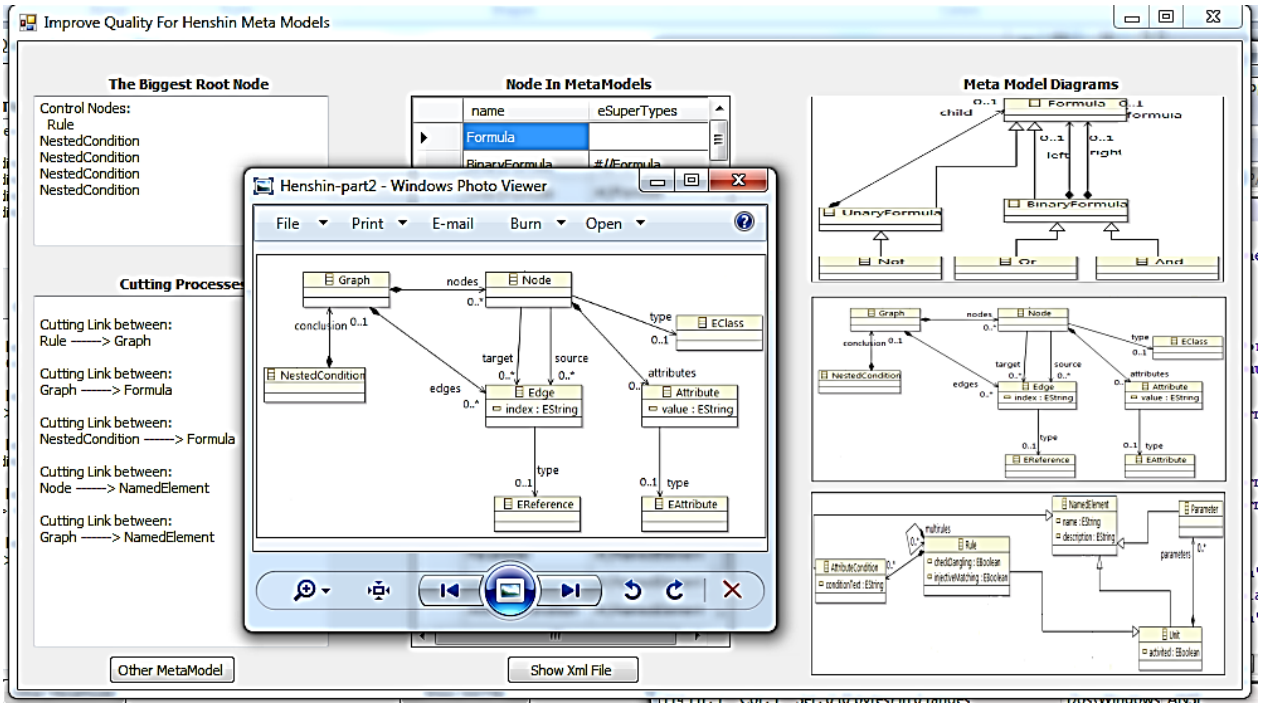
الشكل (15-8) واجهة تعرض المخطط والحسابات العددية *CNN*.



الشكل (15–9) واجهة تعرض النماذج المترفعة الجزئية التي تمّ تقسيمها من النموذج المترفع الكلي.

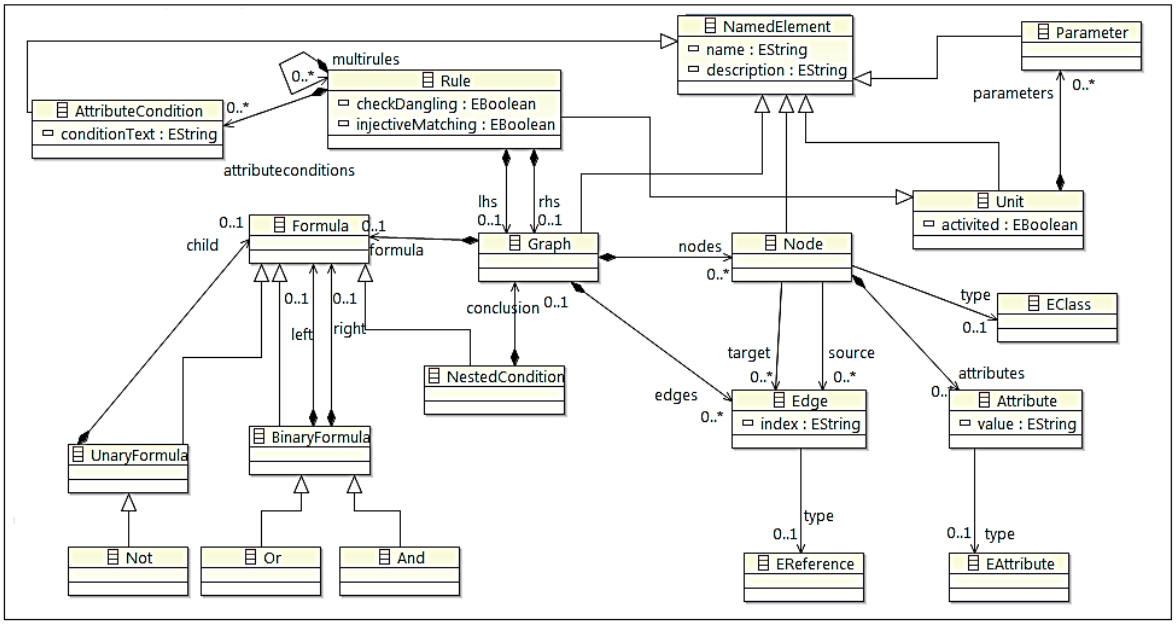


الشكل (15–10) ملف XML لأحد النماذج المترفعة الجزئية.

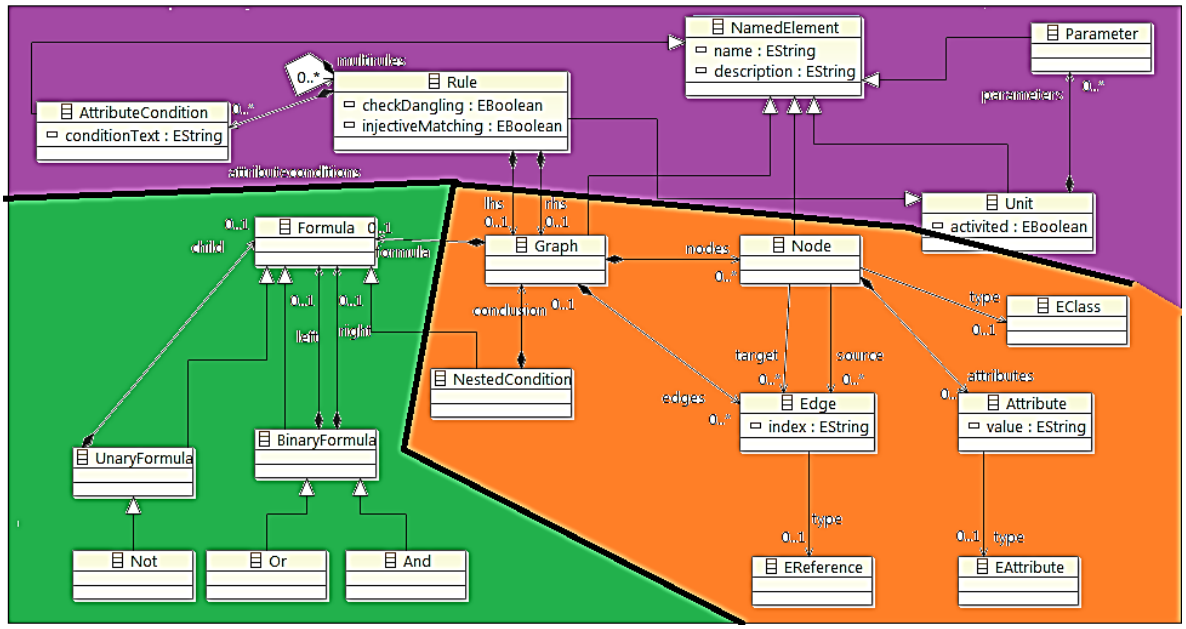


الشكل (15-11) مخطط النموذج المترفع الجزئي.

4.2.15. وجهة النظر حول تقسيم نموذج Henshin المُقترح



الشكل (15-12) مخطط النموذج المترفع قبل التقسم.

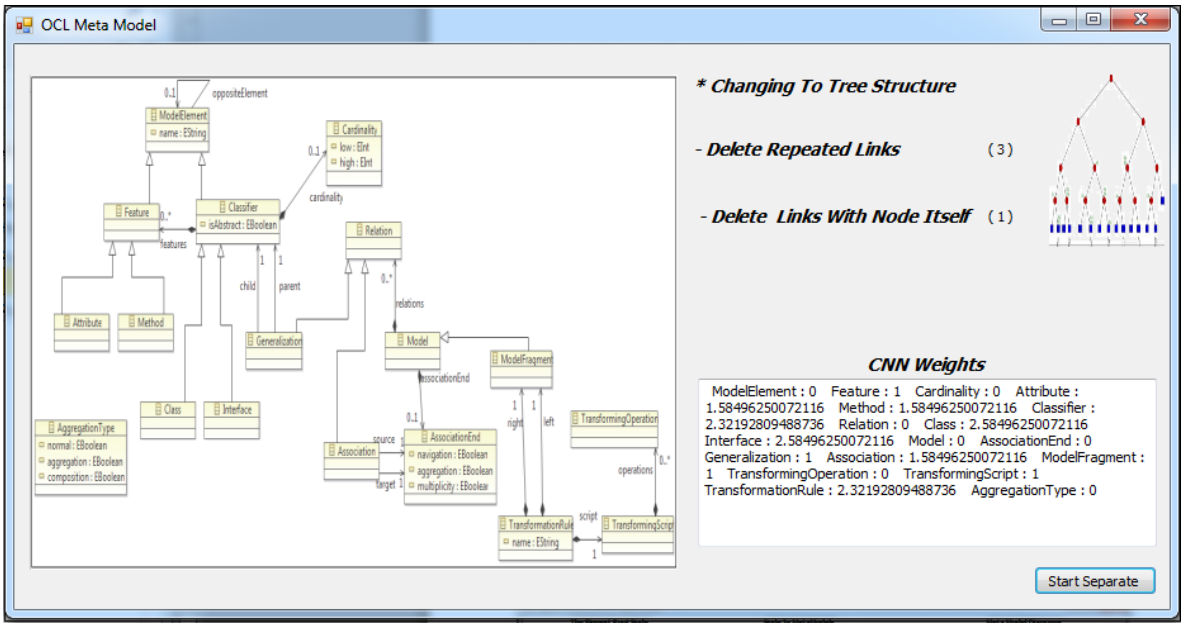


الشكل (15-13) مخطط النموذج المترفع بعد التقسيم.

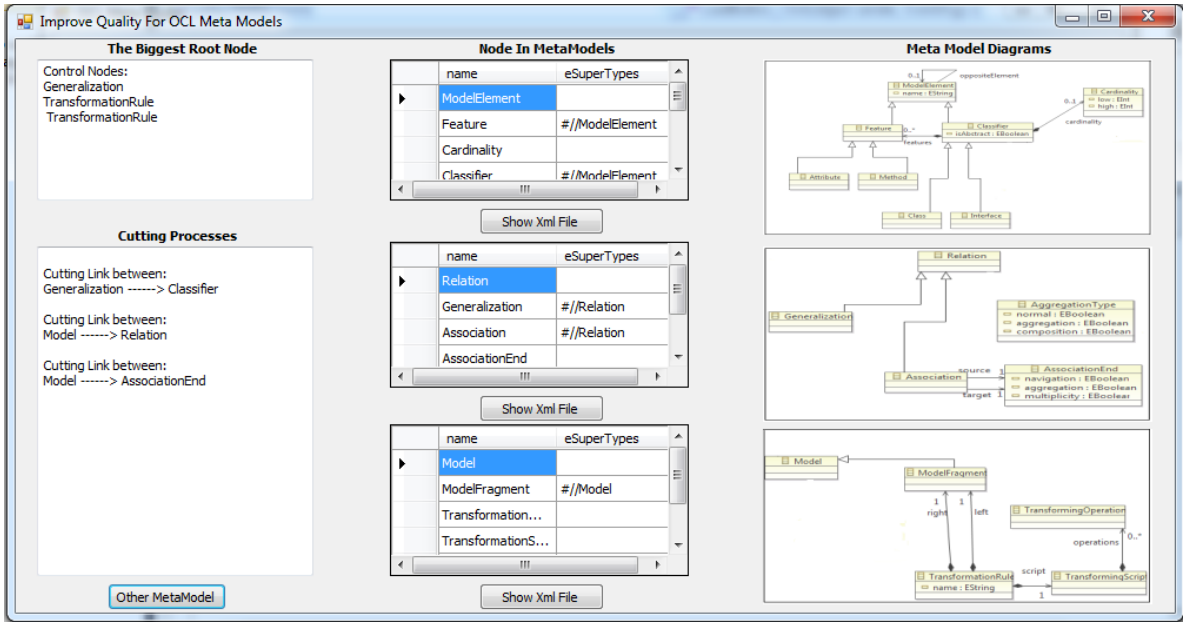
نتج عن التقسيم 3 نماذج مترفعة جزئيّة، الأول يختص بعناصر البيان من عقد ووصلات وأنواع لهذه العقد، الثاني يختص بالصيغ المنطقيّة الأحادية والثنائيّة، والثالث اختص بالقواعد والوحدات والمتحولات. فنلاحظ أنّ عملية التقسيم حققت منطقيّة عالية وأداءً جيّداً في جعل المفاهيم المتشابهة والمترابطة ضمن نموذج مترفع جزئي واحد، مما يضمن سهولة ومرونة في استعمال أدوات النمذجة الناتجة عنها، ووضوح في إطار العمل المعتمد على هذه الأدوات.

5.2.15. نموذج OCL

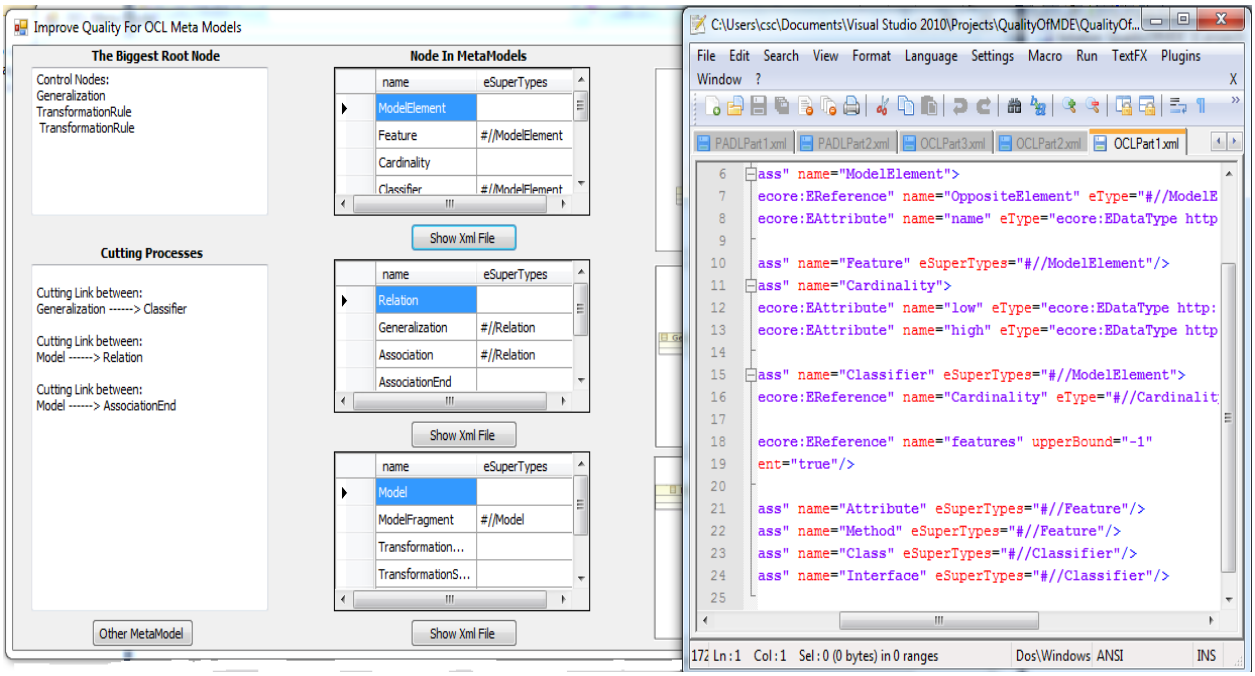
وهو نموذج مترفع للتمثيل البياني للغة موجّهة بالقيود. وفيما يلي سلسلة صور لواجهات التنفيذ المسؤولة عن التحميل والتقسيم.



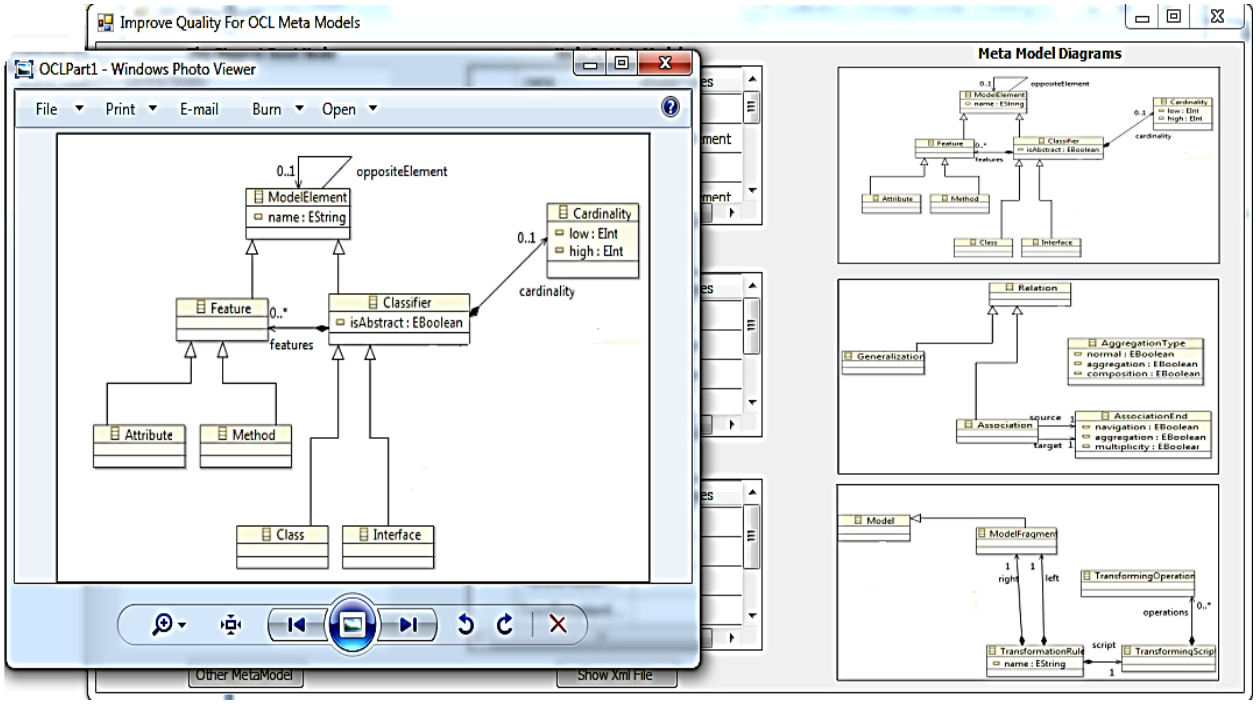
الشكل (15-14) واجهة تعرض المخطط والحسابات العددية *CNN*.



الشكل (15-15) واجهة تعرض النماذج المترفعة الجزئية التي تمّ تقسيمها من النموذج المترفع الكلي.

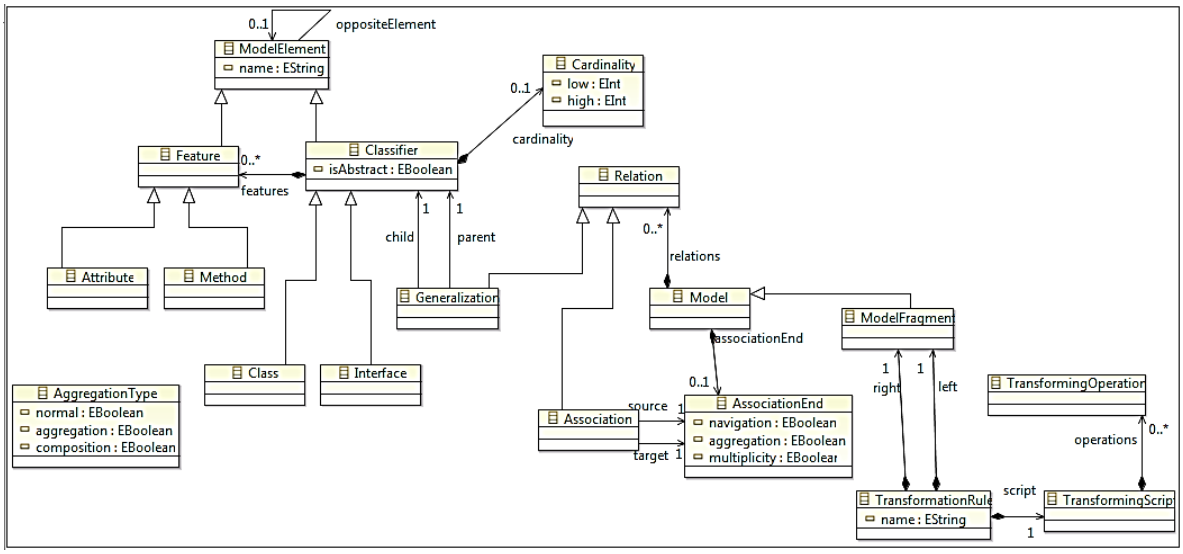


الشكل (15-16) ملف XML لأحد النماذج المترفعة الجزئية.

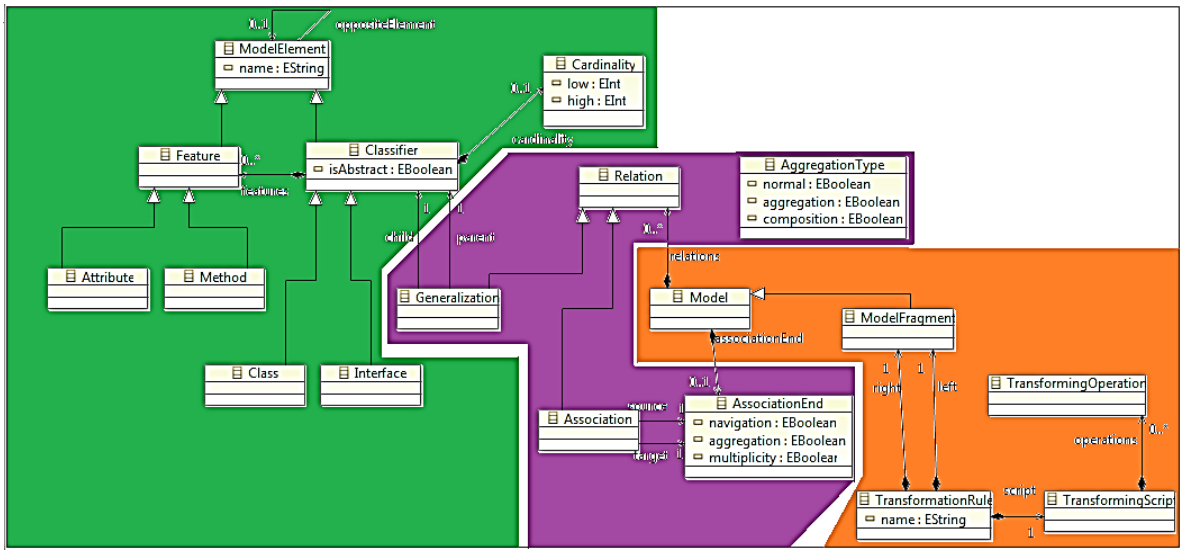


الشكل (15-17) مخطط النموذج المترفع الجزئي.

6.2.15. وجهة النظر حول تقسيم نموذج OCL المُقترح



الشكل (15-18) مخطط النموذج المترفع قبل التقسيم.

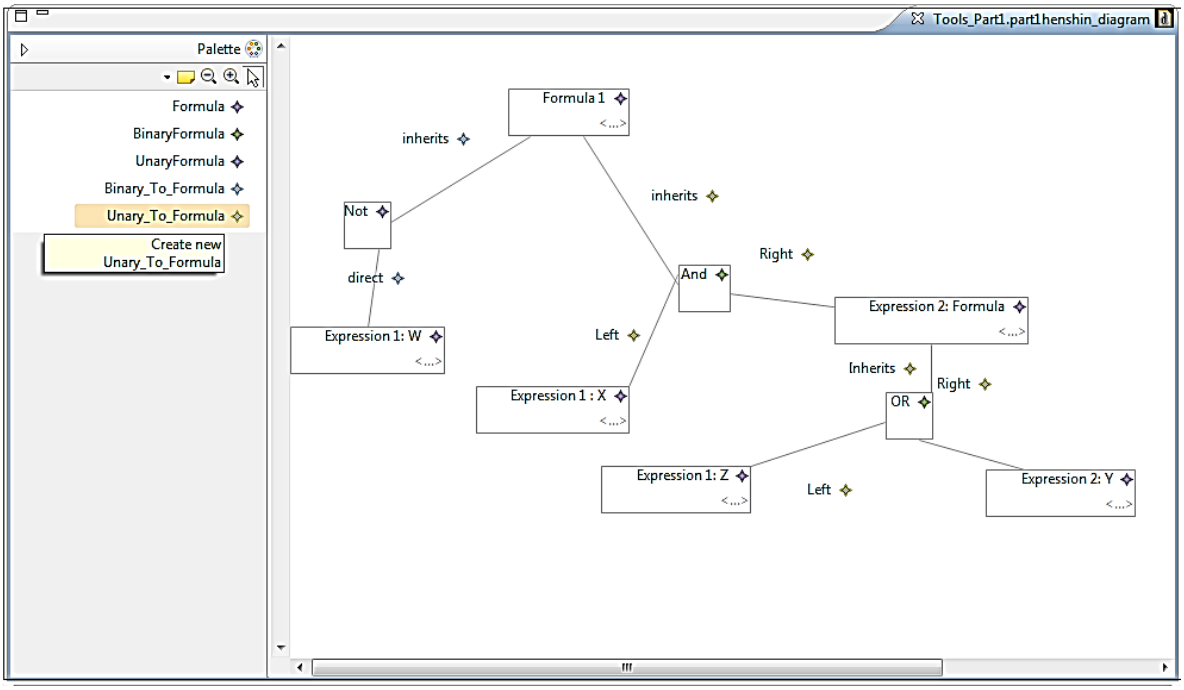


الشكل (15-19) مخطط النموذج المترفع بعد التقسيم.

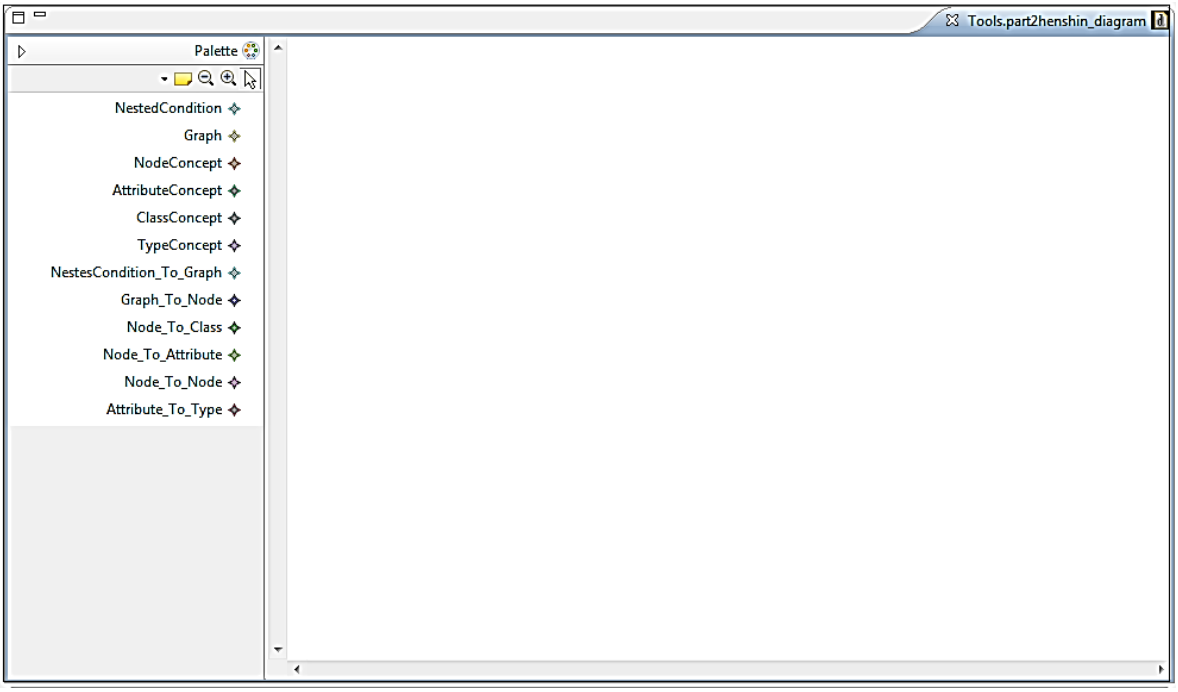
نتج عن التقسيم 3 نماذج جزئية مترفعة، الأول يهتم بالنماذج وتحويلاتها. والثاني يهتم بالعلاقات وأنواعها، والثالث يهتم بعناصر البرمجة الغرضية التوجه.

3.15. الأدوات المتولّدة من النماذج المترفّعة

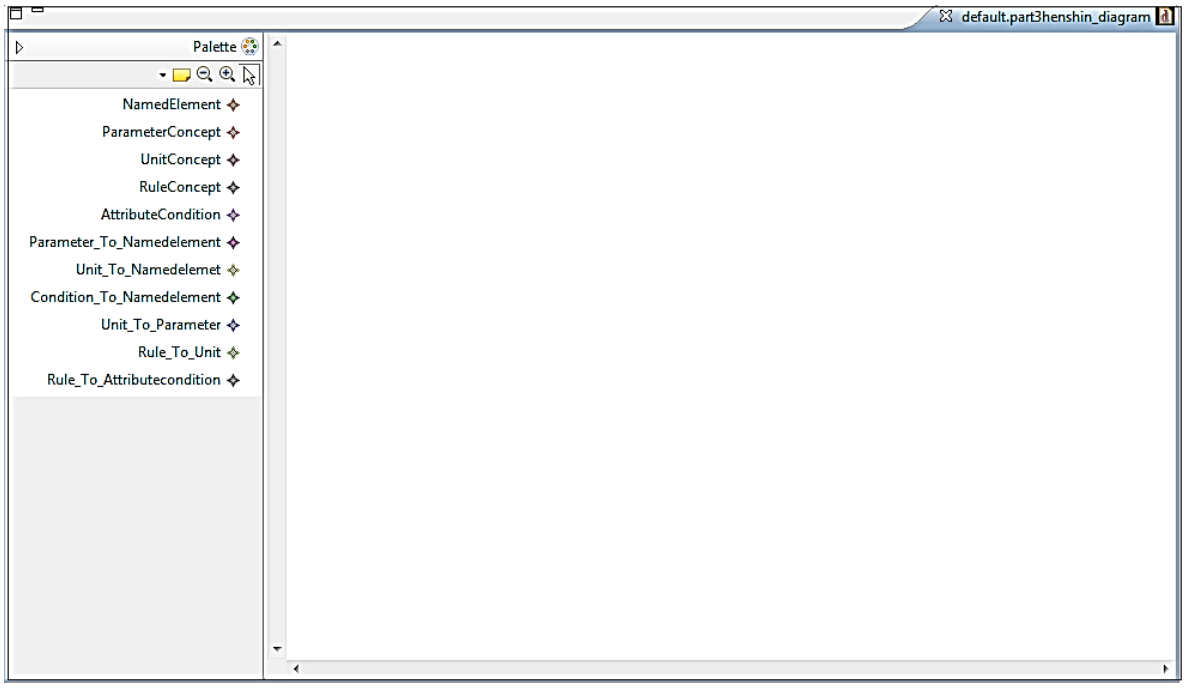
نستعرض مجموعة الأدوات المتولّدة عن النموذج المترفّع Henshin وقد تمّ تقسيمه إلى 3 أقسام وفقاً للخوارزميّة المُقترحة، ونستعرض النموذج الكلي الذي يربط بين الأجزاء التي تغدو أشبه برزم برمجيّة.



الشكل (15-20) الأدوات المتولدة عن النموذج المترفع الجزئي الأول Henshin.

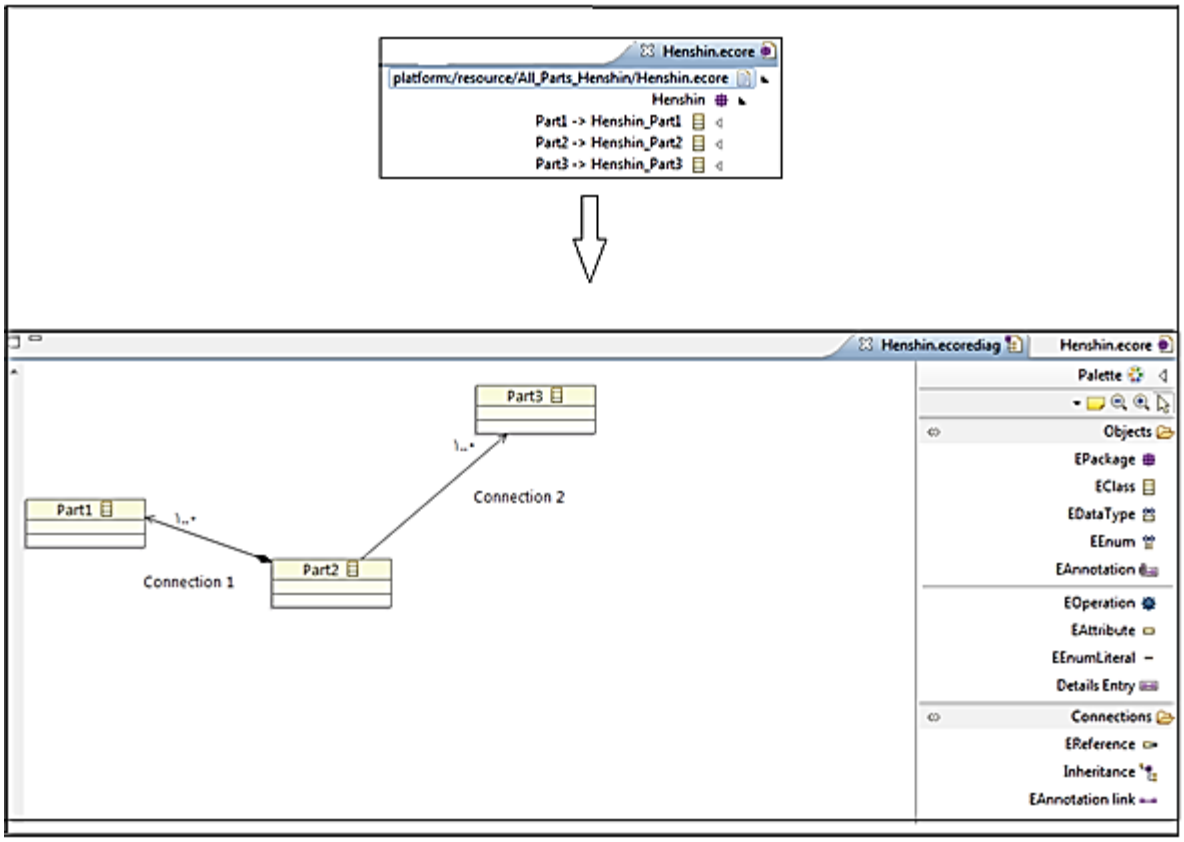


الشكل (15-21) الأدوات المتولدة عن النموذج المترفع الجزئي الثاني Henshin.



الشكل (15-22) الأدوات المتولدة عن النموذج المترفع الجزئي الثالث Henshin.

الشكل (15-23) الأدوات المتولدة عن النماذج المترفعة الجزئية الثلاثة لـ Henshin.



الشكل (15-24) النموذج النهائي الرابط بين النماذج الجزئية الثلاثةالناتجة عن تقسيم النموذج المترفع Henshin.

4.15. الخلاصة

(رأينا في هذا الفصل أهم الواجهات البيانية الخاصة بمشروع التقسيم الذي يعمل على تقسيم النماذج المترفّعة إلى عدد من النماذج المترفعة الجزئية ورأينا أهم الأدوات المتولّدة عن هذه النماذج.)

الخلاصة والآفاق المستقبليّة

في نهاية هذه الدراسة يمكن أن نوجز أهم النتائج التي توصلنا إليها على الشكل التالي:

- يُعتبر موضوع جودة البرمجيات من المواضيع الهامة بسبب تعقيده وتغلغله بالأنظمة البرمجيّة، علاوةً على ذلك فالاتجاه الحالي من التطوير والصيانة يتطلب قياس الجودة مع الكثير من التفاصيل. إنّ الجودة العالية للبرمجيّة تسمح بالتخفيف من كلفة التصحيح، والاختبار، وإعادة استخدام المنتج، وتتأثر الجودة بشكل عكسي مع وجود العيوب البرمجية؛
- قمنا بدراسة العيوب في السويات المنخفضة (سوية الرماز والتصميم) وارتقينا بها نحو السويات العليا (سوية النماذج والنماذج المُترفعة) من سويات الهندسة المُقادة بالنماذج؛
- اقترحنا منهجيّة لتقسيم النماذج المُترفعة الكبيرة إلى عدد من النماذج المُترفعة الجزئية لإزالة العيوب المتمركزة حول الحجم الكبير وضعف الترابط في السويات العُليا، وذلك في محاولة لزيادة إمكانية الاستخدام والفهم والصيانة والمراجعة.. .، وذلك بالاعتماد على النتائج التجريبيّة للممارسة العمليّة التي تمّ طرحها كجزء من مادة هندسة البرمجيات /3/ لطلاب السنة الخامسة في كلية الهندسة المعلوماتية من جامعة دمشق؛
- قمنا بتمديد منهجيّة التقسيم لتشمل المنحى المنطقي أيضاً، وبذلك نكون قد صالحنا ما بين المنهج المنطقي والمنهج الهيكلي؛
- قمنا باختبار المنهجيّة على نطاق واسع شمل طلاب السنة الخامسة للتأكد من مطابقة المنهجيّة المُقترحة لمعايير الجودة العالمية؛
- استطعنا نقل طريقة التقسيم ومناقشته الموضوعيّة من المنحى اليدوي إلى المنحى التلقائي الآلي؛
- قمنا بتقديم البحث للنشر في مجلة Computer Science Journal وهي مجلة علمية محكمة ماليزيّة الأصل ونحن بصدد انتظار أن تقر بالبحث والمنهجيّة المُقترحة.

وأخيراً وليس آخراً نرى أنّ العمل المُنجز في هذه الرسالة يؤسس لمنحى مهم في مجال النمذجة في الهندسة المُقادة بالنماذج، إذ تعمل على التوسع الشاقولي من خلال زيادة جودة إجراءاتها من خلال

تلافي العيوب الموجودة في السوية العليا من نماذجها، وذلك في محاولة لضبط باقي السويات الأدنى منها.على حين أنّ أغلب الأبحاث كانت تعمل على التوسع الأفقي للهندسة المُقادة بالنماذج من خلال استخدامها فقط دون العمل على زيادة جودة إجراءاتها.

وكأعمال مستقبليّة نرى أنّه من الممكن العمل على تغطية الجوانب التي لم تشملها دراستنا، فالعمل في هذا المجال واسع النطاق لذا كان لابدّ من وضع حدود للعمل وتأطير للبحث، وحصر التركيز في نطاق محدود لنتمكن من الإنجاز. فنحن نرى أنّ مناقشة العلاقة العوديّة وإزالتها من ضمن المخططات الشجريّة أمر غاية في الأهمية كوننا نحتاج لدراسة أهمية المفاهيم وأهمية العلاقات لنحدد أيهما ستُحذف. ونرى في أمر دراسة علاقات التجميع (Association) الغير الموجّهة أمر غاية في الأهمية، والأمر نفسه مع عدد الروابط ما بين المفهومين، وضرورة أخذ عناصر المفاهيم وكذلك التوابع الخاصة بهم بعين الاعتبار عند عملية التقسيم والتوزين. فكل هذه الحالات أغفلناها لنتمكن من رسم إطار عمل يكون كخطوة أولى ويرسم معالم أساسيّة في هذا المجال.

وأخيراً وليس آخراً كل ما وصلنا إليه من تمديد عيوب الصف الكبير God Class من المستوى الأدنى (مستوى الرماز والتصميم) إلى المستوى الأعلى (مستوى النماذج المترفعة)، ساهم في رسم منحىً شاقولياً للعمل في الهندسة المُقادة بالنماذج (الاهتمام بجودة السويات المجرّدة في محاولة لتلافي العيوب في السويات الدنيا) بعد أن كان الشائع هو العمل على المنحى الأفقي له (استخدام تقنياته للنمذجة مع إغفال منحى الجودة)، ونرى أنّ العمل على تمديد باقي عيوب الرماز إلى السوية الأعلى يرسم آفاقاً مستقبليّةً حافلةً في هذا الاتجاه.

جدول المصطلحات

المصطلح الأجنبي	الاختصار	الترجمة العربي
Model Driven Engineering	MDE	الهندسة المُقادة بالنماذج
Meta Model	M2	النماذج المترفّعة
Model Smells	------	عيوب التصميم أو عيوب النموذج
Platform Specific Model	PSM	النموذج المعتمد على إطار العمل
Platform Independent Model	PIM	النموذج المستقل عن إطار العمل
Computation Independent Model	CIM	النموذج المستقل عن الحساب
Meta Modeling	-----	عملية النمذجة المترفّعة
Domain Specific Language	DSL	اللغة المحددة النطاق
Refactoring	------	إعادة البناء
Simple Web Model	SWM	نموذج الويب البسيط
Pattern and Description Language	PADL	لغة النماذج والتوصيف المترفعة
Objects and Constraints Language	OCL	لغة تعريف الأغراض والقيود
God Class/Blob Class	GC	الصف العظيم / الكبير
Brain Class	BC	صف التشابك الكبير
Metrics	------	المقاييس
Refactoring	------	إعادة البناء / التصحيح

Foreign Class	FC	الصفوف الخارجيّة
Data Class	DC	صفوف البيانات / صفوف هامشيّة
Dynamic Time Warping	DTW	زمن الدورة الديناميكيّة
Antipatterns	------	نماذج سيئة التصميم
Bayes Network	BBN	شبكة Bayes الاحتمالية
Inheritance Anomaly	-----	شذوذ الوراثة
Concurrent Systems	-----	النظم المتسايرة
Resource Description Framework	RDF	إطار توصيف الموارد
Software Quality Assurance	SQA	ضمان جودة البرمجيات
Total Quality Management	TQM	الإدارة الشاملة للجودة

الملاحق

الملحق /أ/ – الاستبيان الأول

Damascus University.

Faculty of Information Technology, Department of Software Engineering.

Fifth year, Software Engineering / 3 / course.

Year 2014-2015, First semester.

The Student's Name : [redacted]

1. Smells in Meta-Models are more dangerous from smells in other levels in MDE.

a. Strong agree. ***b. Agree.*** ***c. Disagree.***

2. Large size and less cohesion between the concepts in meta-models are the main reasons to smells.

a. Strong agree. ***b. Agree.*** ***c. Disagree.***

3. What are the main drawbacks which smells cause in Meta-models?

Smells in Meta-Models are negative points, because they make produced tools harder to use, need more time to training and understanding.

4. To the best using, suggest a partitioning to the following meta-models to make them easy to use in your opinion.

- *The first case study is about a meta-model for a graphical notation of OCL language [7] which was published in IEEE conference in 2010.*
- *The second case study is about a meta-model for CMMI [8] [9], It was suggested by some students for graduated project in Damascus University. It included three main functions (collecting, analyzing and management of requirements).*

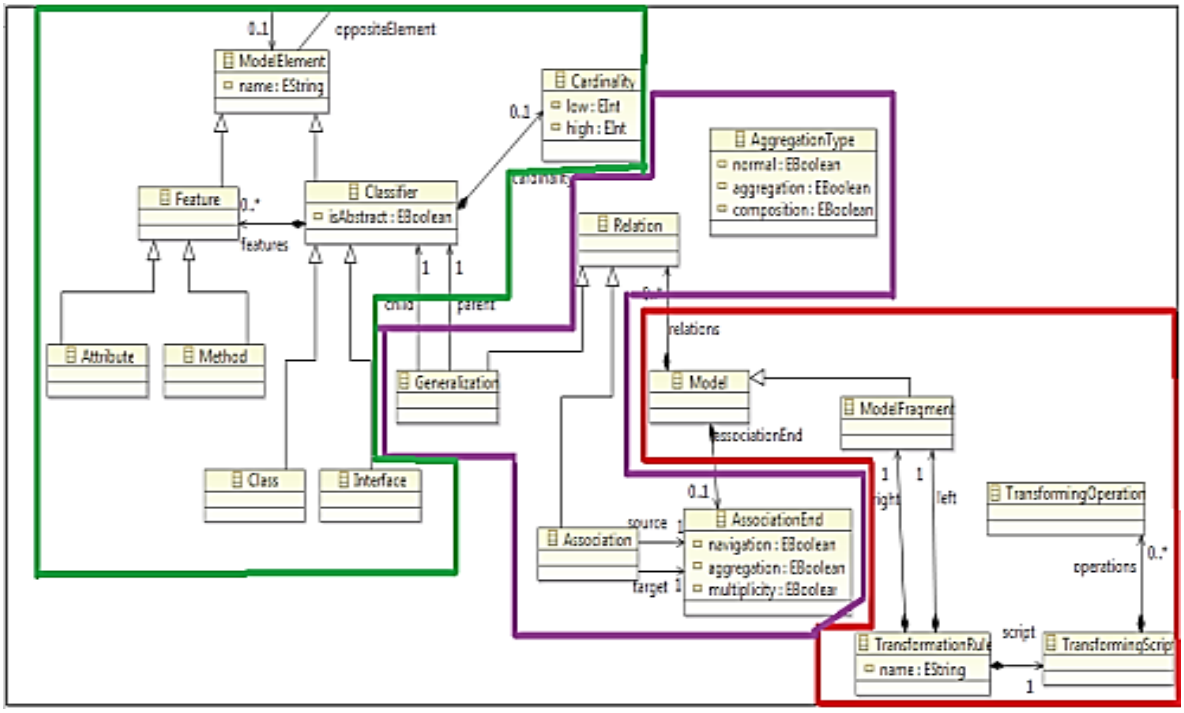
[7] *- Stolc, M., Polasek, I. (2010). A Visual Based Framework for the Model Refactoring Techniques, 8th IEEE International Symposium on Applied Machine Intelligence and Informatics, Slovakia.*

[8] *- Kulpa, P., Margaret,K. (2008). Interpreting The CMMI A Process Improvement Approach 2nd Edition.*

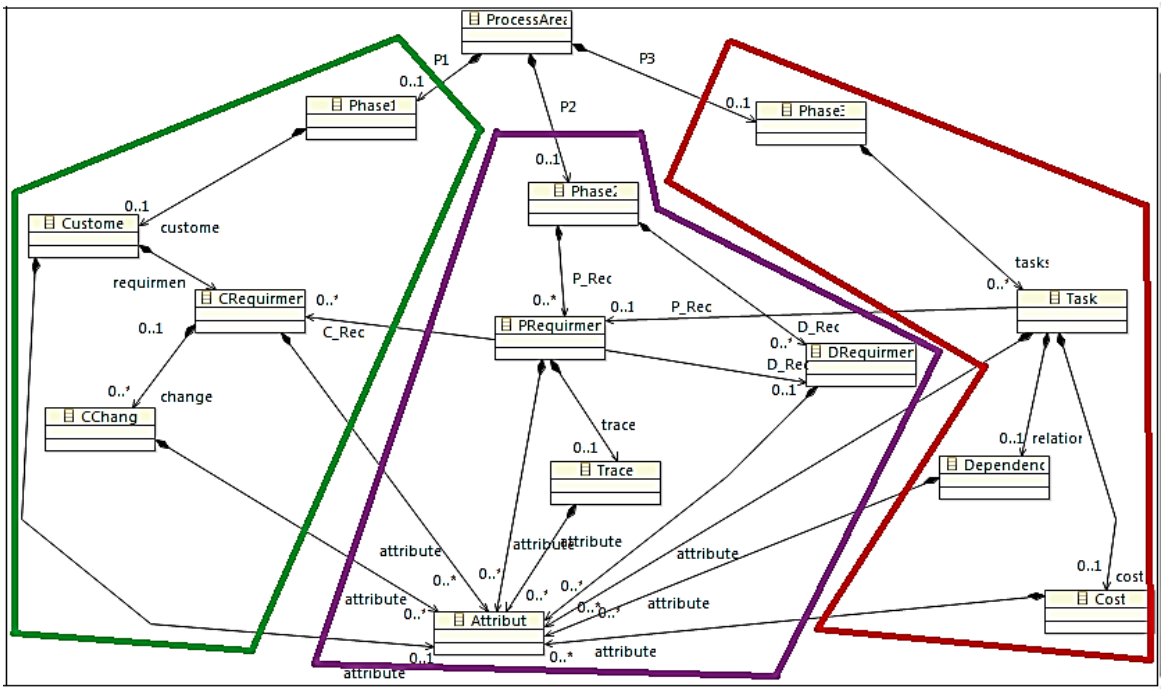
[9] *- Konrad, M., Shrum, S., Chrissis, M.-B. (2011). CMMI® for Development, Third Edition, Addison-Wesley Professional.*

- *The third case study is about CMMI Meta-models,too. It was suggested by some students for graduated project in Damascus University.*

- *The fourth case study is about Web ML Meta-Model[10] . It was suggested by some students for graduated project in Damascus University.*

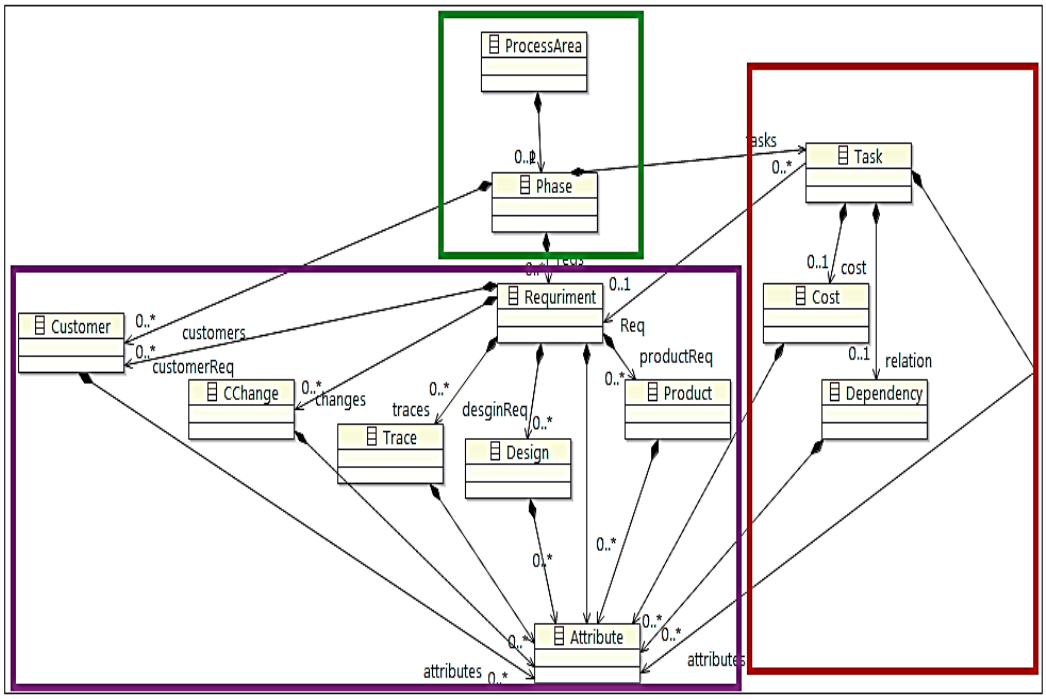


The First Case Study, OCL Meta-Model

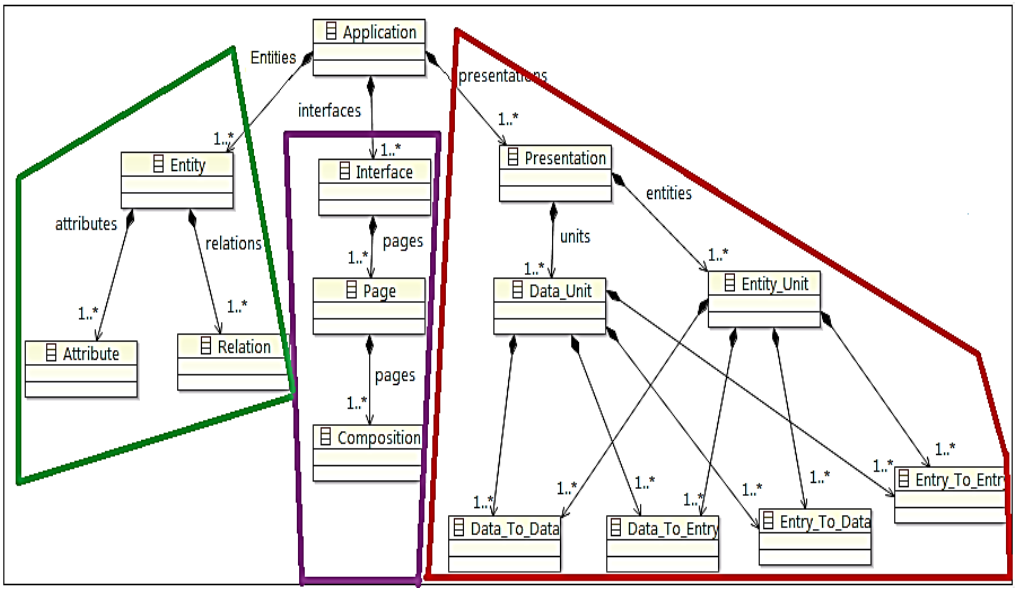


The Second Case Study, CMMI Meta-Model

[10] - *Hennicker, R., Koch, N. (2002). Modeling the User Interface of Web Applications with UML, Germany.*



The Second Case Study, CMMI Meta-Model



The Fourth Case Study, Web ML Meta-Model

Explain Your Ideas about partitioning here. You can attach more papers if you need.

I tried to separate the large Meta-Model to many partial Meta-models that have smaller size and more correlation links and has logical relations.

I try to partitioning meta-model in way to ensure usability, flexibility and understanding to the produced toos. ***End of the First Survey*** .

الملحق /ب/- الاستبيان الثاني

Damascus University.

Faculty of Information Technology, Department of Software Engineering.

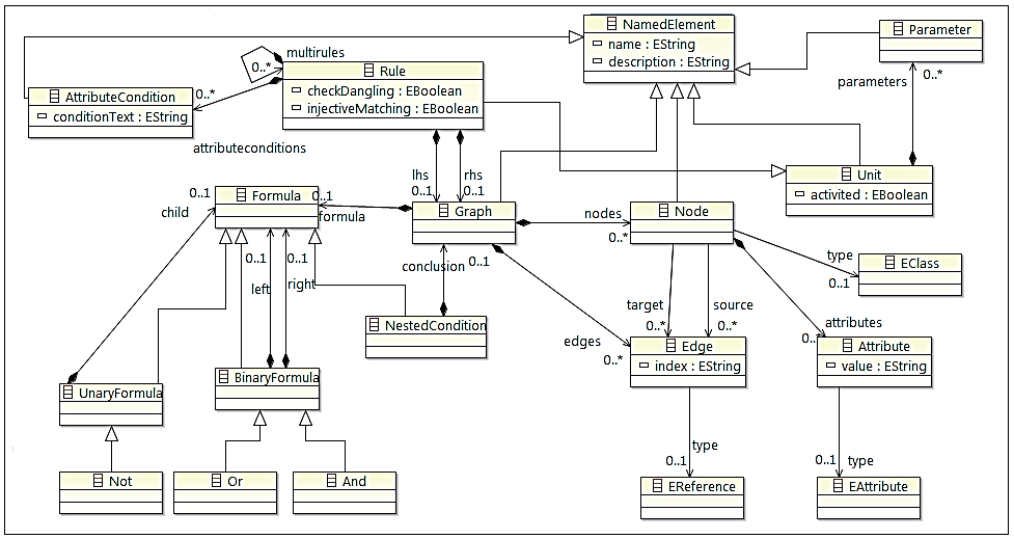
Fifth year, Software Engineering / 3 / course.

Year 2014-2015, First semester.

The Student's Name: [redacted]

**** Check the following meta-models and answer -in average and approximately values- the question.***

The first meta-model is a part of Henshin meta-model[11], which is a high-level graph rewriting and model transformation language and tool targeting models defined in the Eclipse Modeling Framework (EMF)[12]



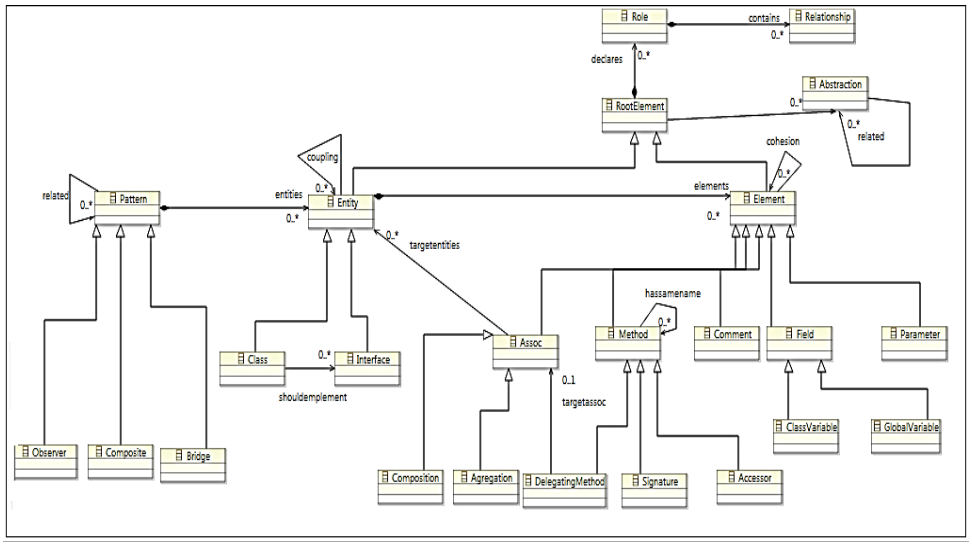
The First Meta Model –Part of Henshin Meta Model

The second meta-model is about Pattern and Abstract level Description Language (PADL)[13].

[11] *- Tichy, M., Krause, C., Liebel, G. (2011). Detecting performance bad smells for Henshin model transformations, University of Gothenburg, Sweden.*

[12] *- Steinberg, D., Budinsky, F., Paternostro, M., Merks. E. (2009). EMF: Eclipse Modeling Framework. Addison-Wesley, 2. edition..*

[13] *- Moha, N., Bouden, S., Gu'eh'eneuc, Y. G. (2007). Correction of High-Level Design Defects with Refactorings, Department of Informatics and Operations Research University of Montreal, Quebec, Canada.*



The Second Meta Model – PADL Meta Model

1. How much time would you want to make your models from these meta-models?

a. Less than two hours. *b. Less than four hours.*

c. Less than six hours. *d. More than six hours.*

2. How many button clicks do you want to make a concept in model from these meta-models?

a. Less than six clicks. *b. Less than eight clicks.*

c. Less than ten clicks. *d. More than ten clicks.*

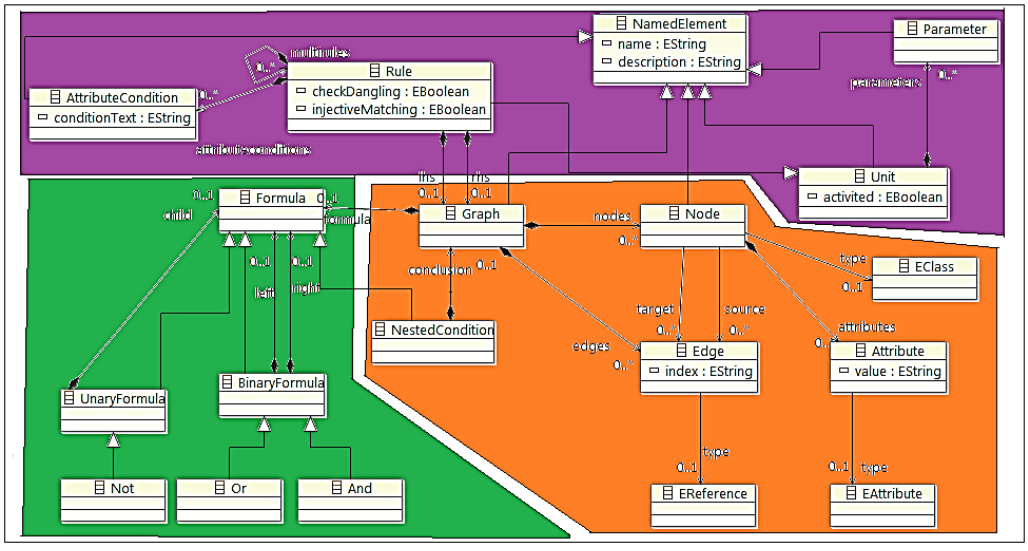
3. Do you use any help files or any documentations to understand the designs?

a. Yes. ***b. No.***

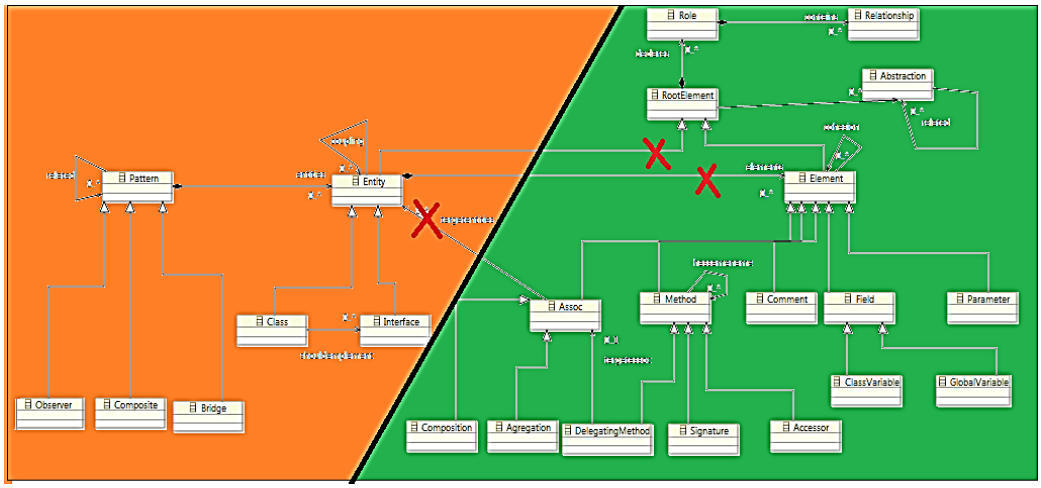
==========

**** Check the following meta-models after partitioning and answer in average and approximately values- the question.***

Each color refers to the independent partial of meta-models.



The Partial Meta Models of Henshin



The Partial Meta Models of PADL

4. How much time would you want to make your models from these meta-models?

b. *Less than two hours.* b. *Less than four hours.*

c. *Less than six hours.* d. *More than six hours.*

5. How many button clicks do you want to make a concept in model from these meta-models?

b. *Less than six clicks.* b. *Less than eight clicks.*

c. *Less than ten clicks.* d. *More than ten clicks.*

6. ***Do you use any help files or any documentations to understand the designs?***

a. Yes *b. No.*

7. Do you think that model after refactoring is:

i.	***More Easy to use?***	***a. Yes***	***b. No***
ii.	***More Usability?***	***a. Yes***	***b. No***
iii.	***More Understanding?***	***a. Yes***	***b. No***
iv.	***More Efficiency?***	***a. Yes***	***b. No***
v.	***Need Less Training?***	***a. Yes***	***b. No***

8. Do you think that we enhance the quality after partitioning?

a. Yes, of course. *b. Yes.*

c. No.

End of the Second Survey .

الملحق /ج/- الورقة البحثيّة المنشورة

Bassem Kosayba
Lubna Mansour

IMPROVEMENT OF QUALITY FOR MODEL DRIVEN ENGINEERING PROCESSES

Abstract *The quality of software development processes importance increases which makes maintenance and update of these procedures are more easily, because the change must be done.*

Model Driven Engineering (MDE) is an approach to build software development processes by proving a large success through wide expansion in all information technology fields.

There are few studies to measure the quality of MDE software processes. So, we focus in this paper on how to measure the quality of these processes and we present algorithms to enhance their quality.

Keywords *Model smells, Refactoring, Meta-Models, Partitioning, Empirical studies, Large size, Less cohesion*

1. Introduction

Software quality is important because of the complexity and pervasiveness of software systems. Moreover, the current trend in outsourcing development and maintenance requires means to measure quality with great details. High quality of software allows reducing the cost of refactoring, testing and reuse the production. Object-oriented quality is adversely impacted by software smells [1] .
Smells are indicators or symptoms of the possible presence of defect in the code or design [2]. They can cause a decline in the quality and performance. The code or design will need a longer time for adjustment, extension and understanding [3, 4]. Smells early detection and correction would benefit the development and maintenance processes.
Meta Models (M2), which drives The Model Driven Engineering (MDE) approach, is the basics for the production of modeling tools. Modeling tools allow us to define Models (M1) in one step in the process. Software quality highly relates with development process. Development process is affected by tools quality that supports its steps. Whenever the highest quality tools increases the quality of development process that use these tools [5].

MDE extended to include all fields [5, 6, 7, 8]. It is an approach focuses on models and operations of models. But researches that interest in measuring and improvement of this approach procedures is almost non-existent.

In this study, we focus on measurement quality of model driven processes and give ways to enhance their quality. We see that the quality of a model driven process is directly affected with the quality of used tools in each process's stage.
We inspire our ideas from code and design smells studies to enhance quality. We will transfer these studies from code level to meta-model level that drives model driven processes. We will modify software design enhance algorithms to be able to use them in meta-model design enhancing, and make them devoid from smells.
In his paper, we explain the results of our empirical studies with the fifth year students at Information Technology Faculty in Damascus University. We attempted to enhance the quality by discovering the most important smells in meta-model levels and refactoring them. Most of these smells were about the large size and less cohesion between concepts of meta-models.
We benefit of the students' empirical studies results to suggest an algorithm to divide meta-models, which are the basic of generating modeling tools, to many partial meta-models.
This partitioning algorithm depends on mathematical criterions that resulted from empirical studies. We targeted to avoid large and cohesion smells. This allowed generating more usable modeling tools and more specializing. It made a clear and understood framework, which we can easily modify.
The rest of this paper is organized as follows: Section 2 presents The Importance of Model Driven Engineering. Section 3 displays The Extension of Code and Design Smells.
Section 4 displays the Analyzing of the Empirical Studies. Section 5 discusses The Proposed Algorithm. Section 6 describes The Implementation of The Approach on Standard Meta-Models. Section 7 presents The Assessment Criteria. Section 8 describes Conclusion and Perspective. In the appendixes, there is a sample of students' surveys.

1.1. The Importance of Model Driven Engineering

MDE depends on using models and their transformations to organize system development process. It allows describing an approach to identify a problem and solution ways. It separates software development process to many abstract levels [9], then each level is separately modeled by meta-models [10]. It achieves the separation between system interests and increases models reusability. So, it generates automatically tools to support modeling in each interest [11]. Models are different in each level in professional and technical accuracy. The transferring between these levels is achieved by model transformations [12].
Meta-Modeling consists of many levels that shape a hierarchal diagram. Each level of models generates the following level. It begins with system level (M0) which is generated by model level (M1). Model level is generated by meta-model level (M2). The final level is a meta-meta-model (M3) which is generated by itself. (Figure 1) clarifies the hierarchy structure.

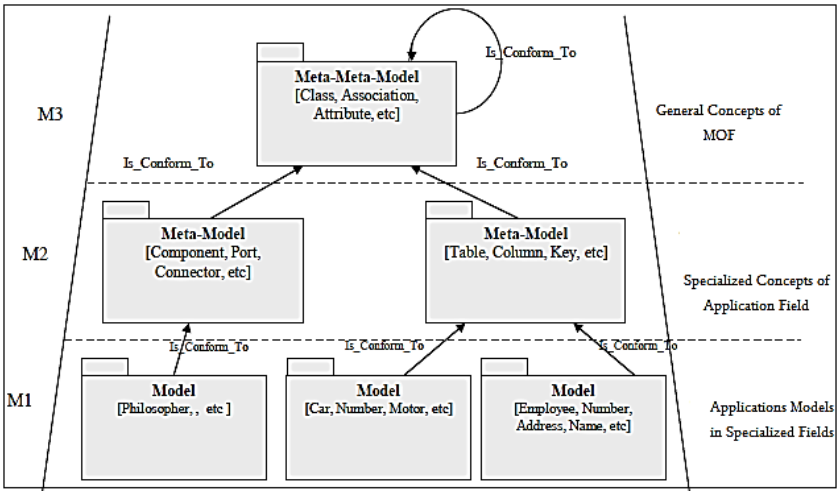


Figure 1. The Hierarchy of Meta-Modeling.

Meta-models, which drive MDE process, could be used to generate modeling tools in each interest, such as in [9]. Consequently, smells in Meta-Models cause smells in generated modeling tools which cause retiring in quality of development process that uses these tools. We conclude that the higher quality of meta-models is, the produced tools and the development process based on these tools become more high quality, easier, simpler to use.

This conclusion is based on fact that meta-models are the basic unit that MDE depends on it to build and partition the theoretical frame of specific development process. Modeling tools that are generated starting of meta-models also directly support the different stages of practical frame of development process.

Example:

We can design a process to develop web application in two main steps:

In the first step we design the content of the application's pages.

In the second one we specify the order of surfing the pages and the transferring between them.

This process to be achieved by using MDE, we define a Meta-Model to design the application's pages and we define another Meta-Model to describe the order of surfing these pages.

The first Meta-Model is used to generate a tool which allows to design application's pages and the second meta-model is used to generate a tool which allows to load page's models and specify the way of surfing and transferring between them.

After that, we can generate total application by an code generator and towards many technics (Web application, Smart phone, ...).

The features of this process are:

- The application's pages are independent from each other, so they can be reused in many other applications.
- The control definition of the application (how to surf the pages) is achieved through a central place instead of the random dissipated control in the different pages.
- No need to users' training in technical way to develop the application. Users only have to decide their requirements from the application (number of pages, the order of navigation pages).
- Capitalizing the logic of application work and keeping it as a model in order to generate the application in many current technics or future technics.

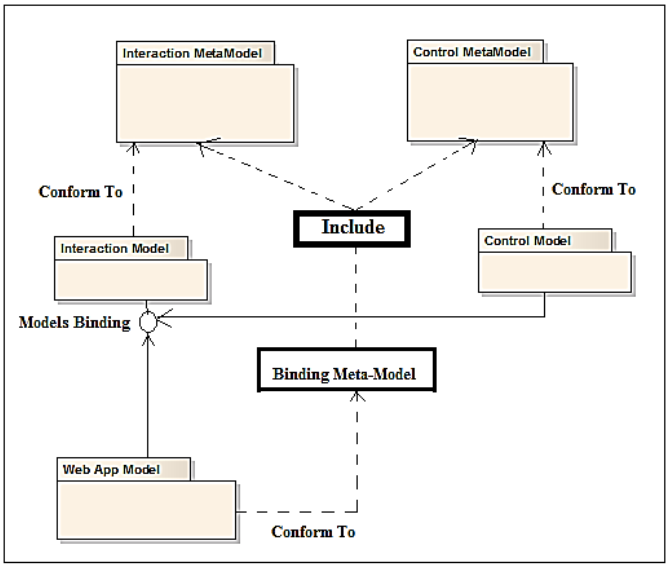


Figure 2. Model Driven Procedure of Web Application Development.

Our approach can be summarized as quality assurance of model driven development processes through removing smells from meta-models which drive these processes. So the following section introduces code smells and refactoring ways to use them in our algorithm for enhancing quality of meta-models.

2. The Extension of Code and Design Smells

The presence of code and design smells can have a severe impact on the quality of the program. Consequently, their detection and correction have drawn the attention of both researchers and practitioners who have proposed various approaches to detect code and design smells in programs.

Initially, Fowler et al. defined a set of 22 code smells that have different characteristics and may influence software systems in different ways [13]. Depending on the particular code smell, the system component that is measured can either be a class, method or subsystem. Other researchers later modified and extended the initial set of smells [2, 14, 15, 16, 17, 18]. Webster wrote the first book on "anti-patterns" in object-oriented development [19]; his contribution covers conceptual problematic, coding, and quality-assurance problems. Riel defined 61 heuristics characterizing good object-oriented programming to assess software quality manually and improve design and implementation [20].
The main smells that previous works have contributed significantly are: Duplicated Code, Long Parameter Lists, Data Classes, Lazy Classes, God Classes and Brain Classes.
Most of these smells are about high complexity problem that resulted from a large size, and decomposition problem which is resulted from less cohesion of inner classes. There are some approaches to detect smells. Most of them are manual [21], and some of them rely on certain rules [17, 22, 23, 24] or on metrics that relates to size and cohesion in the class level. Some of these metrics are in the following [25].

• TCC (Tight Class Cohesion)	• NMD (Number of Methods Declared)
• LCOM (Lack Of Cohesion Methods)	• NAD (Number of Attributes Declared)
• LOC (Lines Of method Code)	• NoDC (Number of Data Classes)

Refactoring these smells are varied between extracting attribute from one class to other classes and partitioning the class to many classes or extracting new classes [25]. But refactoring in these ways isn't always the right choice. In some cases, they cause system decomposition. So we have to specify thresholds, if a meta-model traverses them, we have to refactor it. That's because of the conformity of the other quality criterions aren't broken through.

The early detection of smells in high levels in MDE procedures helps in saving effort and time later on. We are going to draw the attention for meta-models level for being steps in the basic units of this procedure. We

use them to generate tools that support different steps in the procedure. The good meta-models have their restrictions on terms of size and number of contained concepts. We have to model them in more related and cohesion way. Consequently the tools which are generated by them are more easily to use and learn [25, 26] and the procedure will be more easily to understand and maintenance. This principle complies with the principles of the software engineering methodology which follows the saying "Divide and Conquer". The work space is divided into a set of packages. Each package collects all similar elements. This allows organizing the work and raising its quality.

We can summarize our work in "Finding the appropriate way to partition the procedure to many steps". We benefited from the related works in code and in designing smells and extend them to cover the meta-models which have the main role in drive the software procedures.

Example:

The large size of a Meta-Model causes a generated tool that has many concepts related to many fields. These concepts are hard to be controlled and learnt.

Consequently, we must partition meta-models that have large size and less cohesion between their concepts to many partial meta-models which has smaller size, high cohesion and more specialized, as we found in a procedure of web application development example.

We will propose an algorithm to divide meta-models to many partial meta-models depending on empirical data. Most similar studies basically depend on empirical data [5, 6, 7, 8] to generalize their conclusion.

3. Models for Analyzing of the Empirical Studies

The empirical study was achieved in the Faculty of Information Technology Engineering at Damascus University. The group of the research was about 100 students from the department of Software Engineering and Information Systems. The empirical study lasted for six months. We had planned it as the following points:

- The students had to part many Meta-Models (CMMI Meta-Models, WebML Meta-Model, OCL Meta-Models ...) to guarantee the appropriate collection of the most similar concepts and the most related ones in partial groups. This guarantees easy use, learning and understanding of the development procedure and its produced tools. The figures in Appendix /A/ clarify the partitioning ways that most students had done.

- The students had to generate the tools from the Meta-Models by using GMF [28] framework before partition and after it.

- The students had to compare between tools that are generated from the Meta-Models before and after partition according to the ease of understanding the tools, the required time to define a model by using the produced tools and the clarity of the general developed procedure.

- We credited the partition results and proposed an algorithm relying on the empirical data to mechanize the manual way. We will explain that in the next section.

We test our proposed algorithm on standard global models. There are PADL [27], which is a Meta-Model about Pattern and Abstract level Description Language, and Henshin [4] which is a high-level graph rewriting and model transformation language. The students compared between the development procedure before and after the partitioning according to the required time to model creation, Numbers of using the documentation and helping files that explain the tools and the number of button click to create a model. The comparing between the resulted values of these factors is sure the quality improvement after partition approach applying. Appendix /B/ clarifies a sample about students answers.

4. The Proposed Algorithm

Figures in the Appendix /A/ are a sample about the partitioning ways that most students decided. To mechanize the partition way, we need a unified structure to represent the model to be an accepted input to our algorithm. The tree structure is suitable one to simplify working and numeral calculations. We can map between Meta-Model and Tree structure by representing the concepts of Meta-Models by nodes of tree, and the relation between concepts of Meta-Models by the links between nodes of tree.

4.1. Conversion Algorithm to Tree Structure

We dealt during the previous meta-models with 3 types of links:

- **Composition Relation**: We transfer it to directed arrow link from a whole concept toward a partial one. (Figure 3) illustrates the meaning.

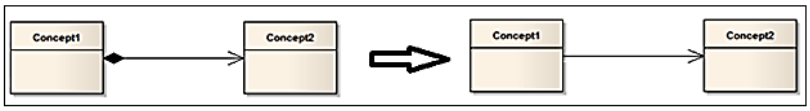


Figure 3. Turning the Composition Relation.

- **Inheritance Relation**: We transfer it to one directed arrow link from a sub class to a super class. (Figure 4) illustrates the meaning.

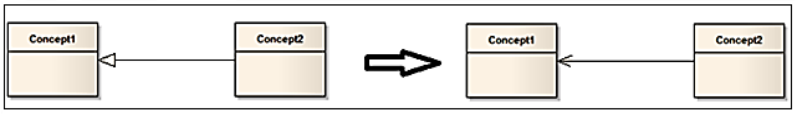


Figure 4. Turning the Inheritance Relation.

- **Association Relation**: We kept it on its current status. (Figure 5) illustrates the meaning.

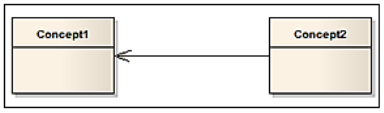


Figure 5. The Association Relation.

The study did not expand to include duplicate inverse relationships or even numbers on both sides of relationships that determines the number of related relationships of concepts (one -to- one) or (one to many). The target is to know the direction of the relationship only. So we disregarded them, and restricted the focus on the three last relations. (Figure 6) illustrates the resulted general tree diagram after application of the conversion algorithm.

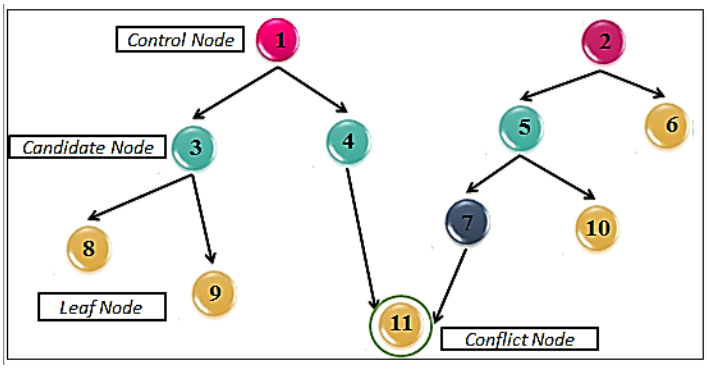


Figure 6. The tree structure.

4.2. The Main Nodes in the Tree Structure

We recognize between 4 kinds of node:

- **Leaf Node**: This node has no output link. It has only input links. The nodes (6, 8, 9, 10and 11) are 'Leaf' nodes in (Figure 6).

- **Control Node**: This is the highest node which refers to other nodes. It might be more than one control node in the same diagram. The nodes (1 and 2) are 'Control' nodes in (Figure 6).

- **Candidate Node**: This node has a direct link with a control node. In the case of partition, it could be a control node in the partial meta-model. The nodes (3, 4, 5 and 6) are 'Candidate' nodes in (Figure 6).

- **Conflict Node**: This node is shared between two candidate nodes or more. It has direct or indirect links with them. In the case of separating the meta-models, this node will be conflicted node between the partial meta-models. The node (11) is a 'Conflict' node in (Figure 6).

In the following section, we present a proposal algorithm for a partition depending on general concepts.

4.3. The General Proposed Concepts

In this section, we propose concepts to partition the Meta-Model relying on analyzing of the previous empirical studies.

- **The Calculated Number of Node (*CNN*) Concept**

  We propose ***CNN*** concept to indicate the level of the nodes and their output links until reaching to the leaves levels. We propose using a logarithmic function as a damping factor. The calculation of ***CNN*** is like the following:

$$CNN = Log\ 2\ (Number\ of\ Output\ Links\ to\ low\ levels + 1)$$

  We add the number one to the number of output links to avoid infinity values.
  Leaf node has no low level, so the calculated number is zero. It is calculated as following.

$$CNN\ (Leaf\ Node) = Log\ 2\ (0 + 1) = log\ 2\ (1) = 0$$

  (Figure 7) illustrates an example about calculation about ***CNN*** for nodes.

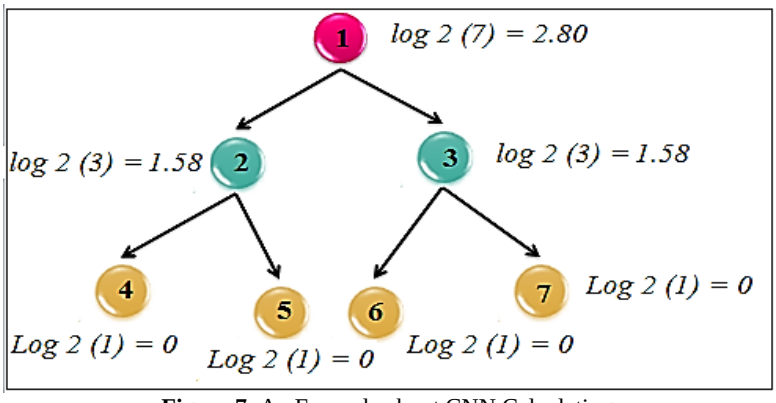


Figure 7. An Example about CNN Calculating.

- **Weight of Node (WN) Concept**

  We suggest this concept to express the size and cohesion between a specific node and a control node in a partial meta-model. The smaller the ***WN*** is, the node becomes more close and hard to separate from a partial meta-model.

We need many concepts in this stage:

- **Cohesion** between the node and control node of partial meta-models. When we have a case of conflicting between two control nodes, we have to determine which one has a weaker correlation with the conflicted node. Less cohesion is one of the main factors that cause smells. We have to remove it.

- **Size and importance** of control node of partial meta-models. We have to avoid a large size of partial meta-models. The large size is one of the main factors that cause smells. We have to remove it.

- **Dumping factor** helps making smaller values and develops a specific threshold for the partition.

The proposed algorithm consists of many factors which depend on the previous concepts:

- Number of the links between the node and the control node in the partial meta-model → This represents the correlation.

- Calculated number for the control node (***CNN***)→ This represents the size and the importance.

- Log 2 → This represents the dumping factor.

The final formula of approach is as follows.

$$Weight\ of\ Node\ (WN) = Log\ 2\ (Number\ of\ Links\ between\ this\ Node\ and\ Control\ Node) + CNN\ (Control\ Node)$$

Weight of Candidate Node (which is separated from the control node by one link) is as following.

$$Weight\ CAndidate\ Node\ (WCAN) = Log\ 2\ (1) + CNN\ (Control\ Node) = CNN\ (Control\ Node)$$

Each conflicted node has more than one WN, because there are many controlling nodes which are conflicting upon this node. So we have to calculate WN between the current node and each one of the control nodes. The smaller WN is, the node become more close and hard to separate from partial meta-models. We give an example about WN in (Figure 8).

When we want to separate the meta-model into many partial meta-models, the control nodes will be conflict on the sharing nodes. Such as the node (7) in (Figure 8). We have to calculate two WNs to this node with the two control nodes that conflicts upon this node.

Larger WN means a greater distance, greater size but less coherent. So the larger WN link needs truncating.

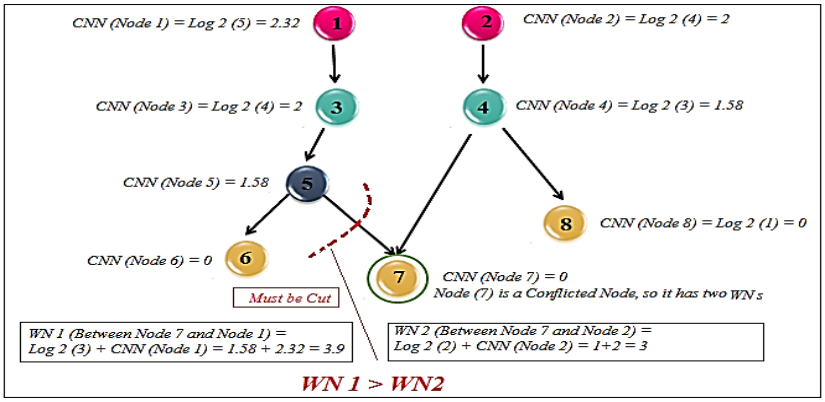


Figure 8. WN Calculating for the Conflicted Node.

4.4. The Mechanism of Applying the Proposed Algorithm

- Threshold for separating the meta-models is 4. If ***CNN*** (Control Node) >= 4 → There are 16 concepts or more in the same meta-model level. We find this number large; we have to reduce it by partition.

- The priority in beginning of calculation (***WN***) is to the node which has larger ***CNN***; because it refers to a larger size and it may be closer to separate in order to reduce size.

- In the case of two candidate nodes or more are linked by each other, we will have the following options:

  - If these nodes have large ***CNN***, links between them will be removed and each node will be a separate stand-alone control node for partial meta-models.

  - Or Else links between them will be maintained. Therefore, it will be one control node have the other nodes under it.

- We test node after node:

  - We allow with existence of a single free control node, it will be the container form and the link between partial models. The rest of the control nodes must be mindful of their belonging to one of the partial models when we split the meta-model in order to prevent the decomposition the system.

  - If a conflicted upon node has two equal ***WN*** with two control nodes, the partitioning priority refers to the user.

  - ***CNN*** and ***WN*** are re-calculated directly after each partitioning.

- If we have more than one control node, we have to do the following steps:

  - If the node has only one link, and upper nodes have only one link, we won't cut that link. We don't want to decompose the meta-model.

  - We can start separating if:
    CNN (Node) >= Log 2 (Number of model's concept / 3).

5. The Implementation of the Approach on Standard Meta-Models

We have implemented the proposed partitioning approach on many international standard meta-models. We clarify the steps of proposed algorithm by the following case studies.

Case Study 1: The meta-model is a part of Henshin meta-model [4], which it is a high-level graph rewriting and model transformation language and tool targeting models defined in the Eclipse Modeling Framework (EMF) [28]. (Figure 9) displays it.
We have many steps of the partitioning algorithm. They are described as the following:

1. Transferring to the tree structure, we transfer the relations according to the rules which were explained previously.

2. Calculating ***CNN*** for all nodes. (Figure 10) displays tree structure and ***CNN*** for nodes

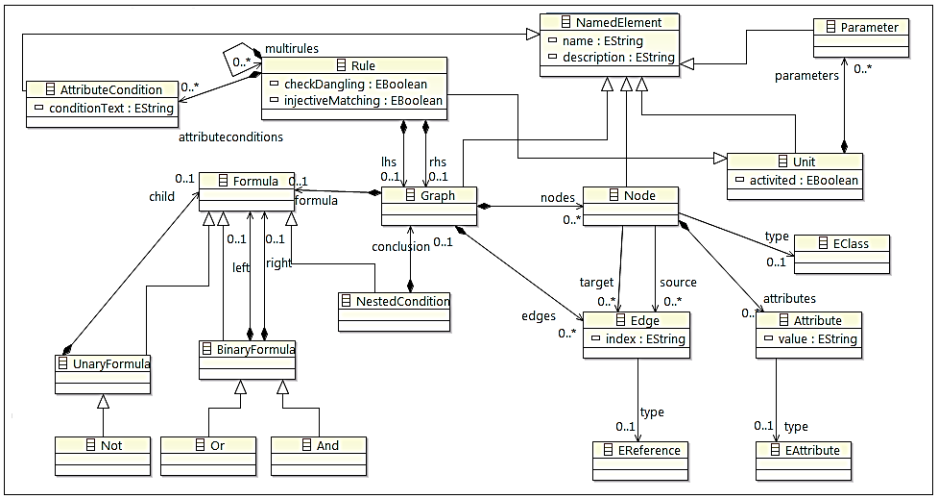


Figure 9. Part of Henshin Meta-Model Rules.

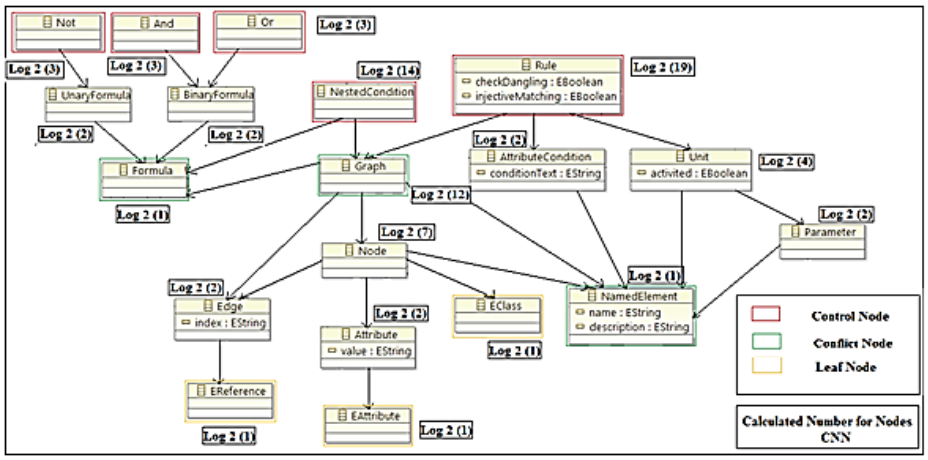
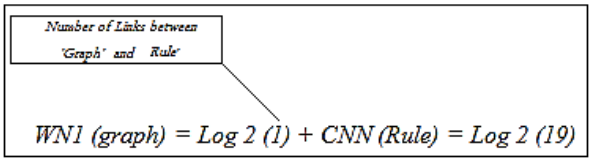
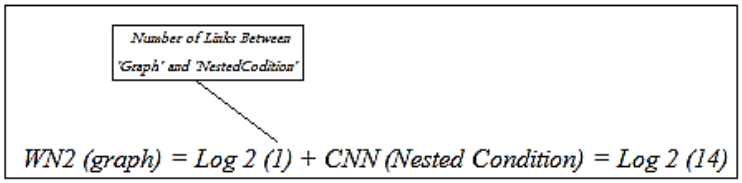


Figure 10. The Tree Structure of Henshin Meta-Model and the Calculated Numbers ***CNN***.

3. We start from the biggest CNN node, which is 'Rule' node, we search for the conflicted node, we find 'Graph' node, which is also a candidate node. The two control nodes, 'Rule' and 'Nested Condition' are conflicting upon it. We calculate WNs' numbers between it and the two control nodes.





4. We compare between the values (***WN*** 1 >***WN*** 2). We cut the link from the largest value. We cut (Graph - Rule) link and recalculate ***CNN*** for nodes. The values are modified. Then, we start searching for conflicted upon nodes. Then, we cut the links that return to the upper ***WN*** value. We repeat these steps until we reach to the threshold limit. When we reach to the limits, we will stop testing. (Figure 11) displays the final partitioning of tree Henshin Meta-Model. It consists of 3 partial meta-models.

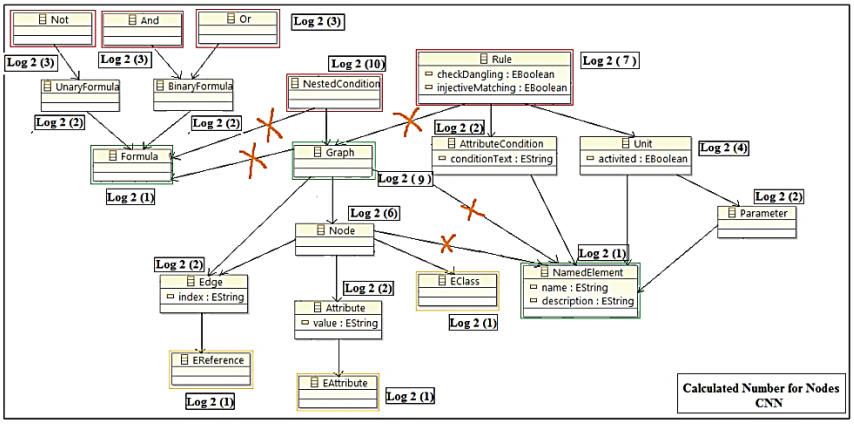


Figure 11. Henshin Tree Structure After Partitioning.

There are many control nodes (Not, And, Or) that are in conflicting each other upon other node some other nodes.
We couldn't cut these nodes being having only one link. We don't want to have free nodes and decomposing the meta-model.
In this case, the Meta-Model is partitioned into 3 partial Meta-Model:

- **The First One:** specializes in graph elements (Nodes, Edges and Kind of nodes).
- **The Second One:** specializes in Formula (Unary and Binary) ones.
- **The Third One:** specialized in Rules, Units and Variables.

(Figure 12) displays the main parts of Henshin Meta-Model before transforming it to the tree structure.

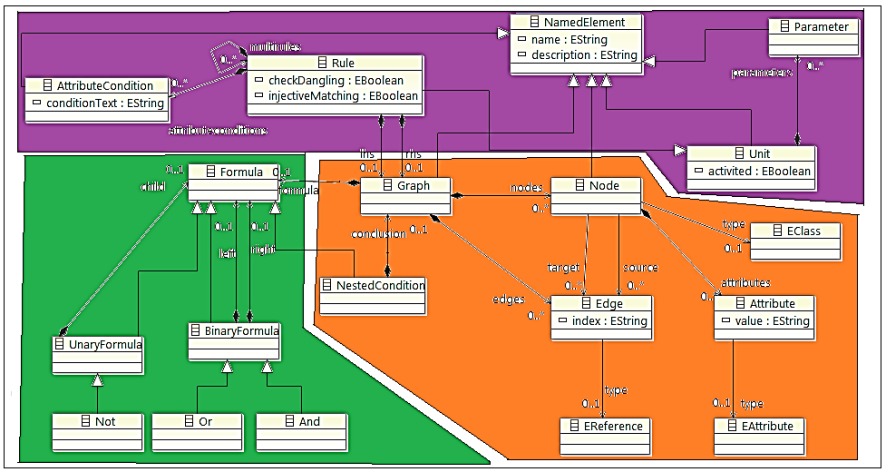


Figure 12. Henshin Meta-Models After Partitioning.

The partitioning operation achieves:

- High logic in the collection of the similar and linked concepts in one partial Meta-Model.
- The Facility and Flexibility in using the generated tools.
- Clarity of the framework which depends on generated tools.
  - Independent tools from each other more specialized and customized tools. Consequently, they can be reused in many other applications.

These results are proved by the empirical studies at Faculty of Information Technology Engineering in Damascus University through use the generated tools from Henshin Meta-Models before and after partitioning.
The students' assessment were positive, we got these assessments by surveys. In the Assessment Section, we will explain the statistics and diagrams. These surveys are a strong proof about improving the quality. Appendix /B/ explains the survey.
(Figure 13) displays the interface of our application which partitions Henshin Meta-Models to many partial Meta-Models.

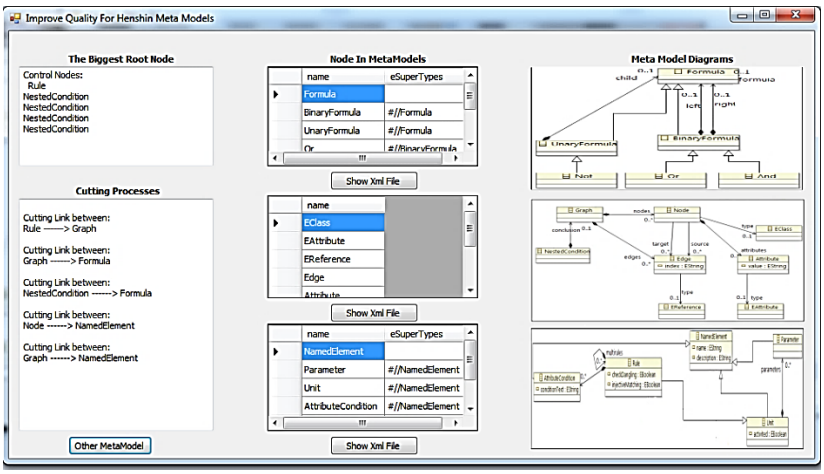


Figure 13. The Interface of our Application.

We generated the independent tools from the partial Meta-Models by GMF framework which is presented by IBM. It automatically generates tools for modeling from Meta-Models.(Figure 14) displays the generated tools.

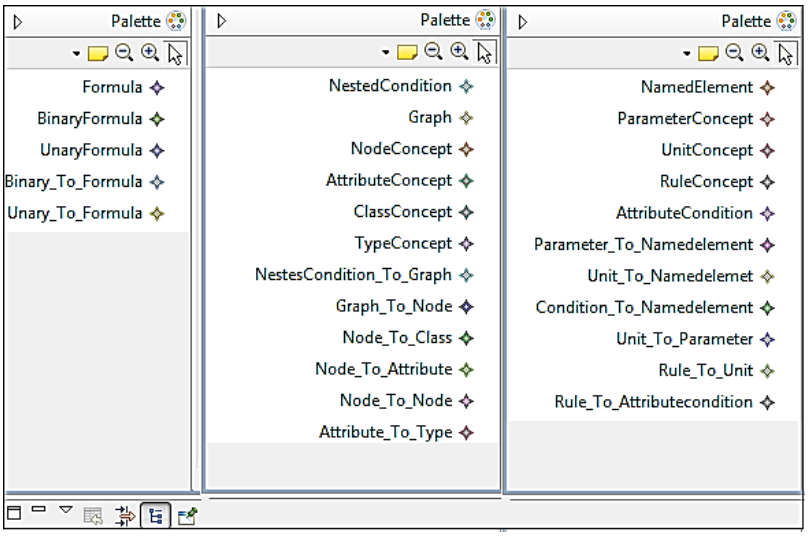


Figure 14. Generated Tools by Partial Meta-Models of Henshin

2.The Second Case Study: is about Pattern and Abstract level Description Language (PADL) [27]. (Figure 15) displays the meta-model.

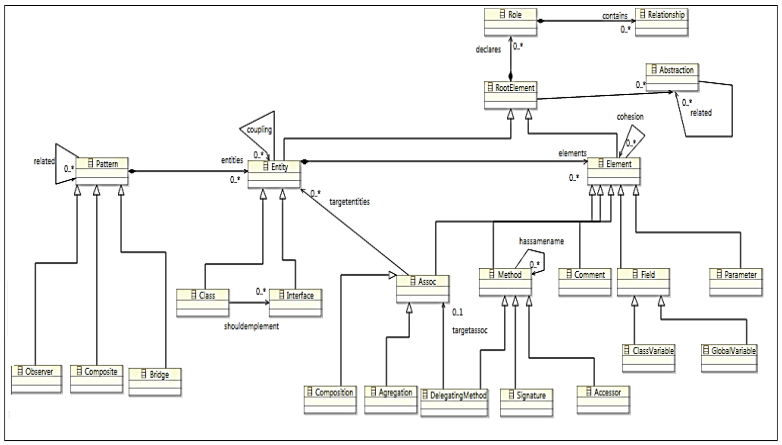


Figure 15. PADL Meta-Model.

Two partial Meta-Models are resulted from PADL partitioning:

- **The First One:** specializes in Object Oriented Elements (Functions, Elements, Variables and Relations between them).
- **The second one:** specializes in Design Patterns (Bridge, Observer …).

(Figure 16) displays the PADL tree structure after partitioning by our approach.
We noticed that the resulted partial Meta-Models are:

- More customization and easier to use.
- Its ability to generate the general structure of the system through the part of Design Patterns, after that the generating of the content of these design (Classes and Functions) are able.

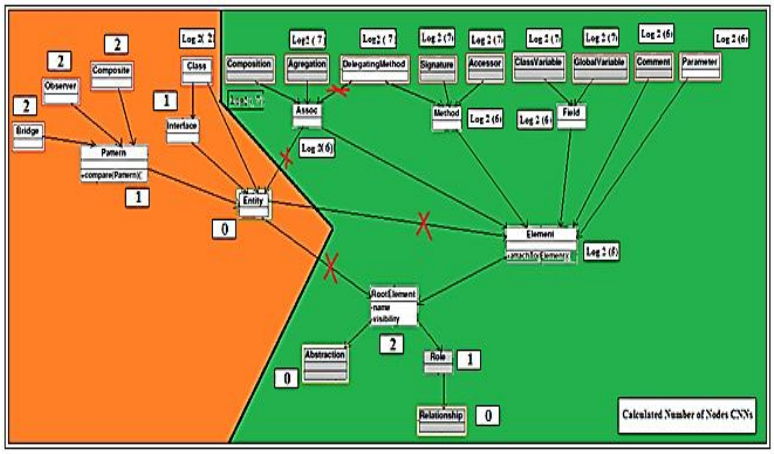


Figure 16. PADL Tree Structure After Partitioning.

(Figure 17) displays PADL Meta-Model after partitioning.

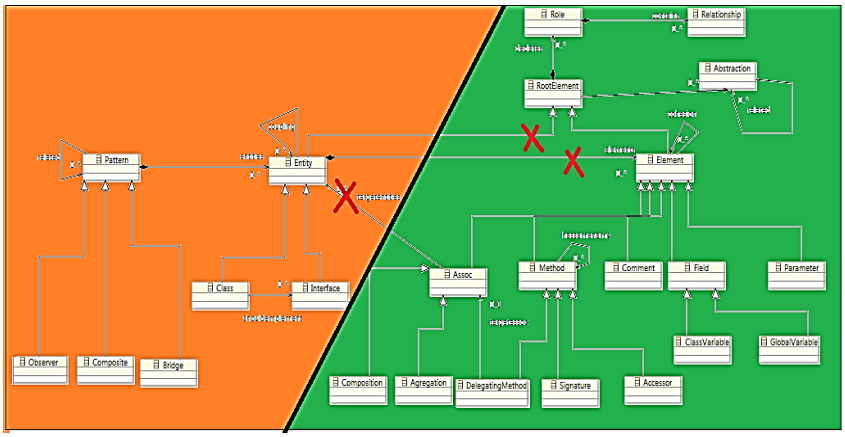


Figure 17. PADL Meta-Model After Partitioning.

The results of partitioning PADL Meta-Models are proved by the empirical studies like Henshin Meta-Model. The students' assessments were positive. They are displayed in Assessment Section.
(Figure 18) displays the interface of our application which separates PADL Meta-Models to Partial Meta-Models.

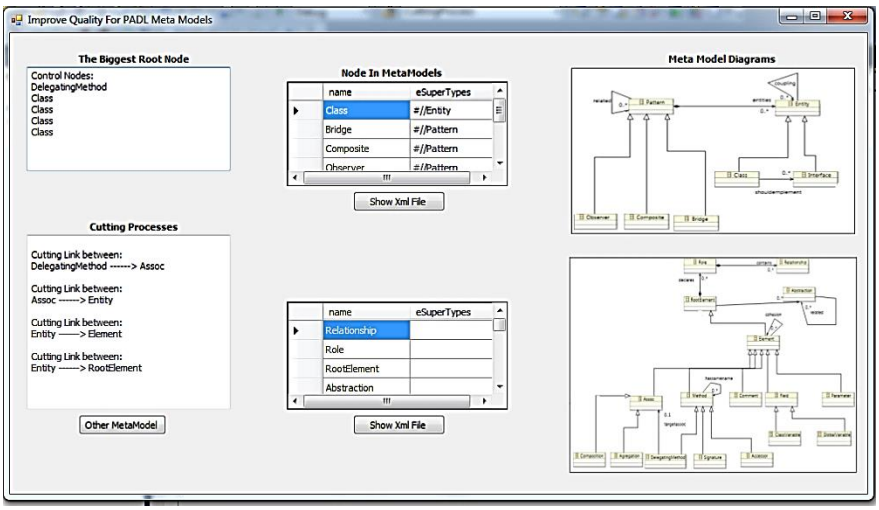


Figure 18. Interface of our Application

(Figure 19) displays the independent tools which are generated from partial Meta-Models by GMF framework.

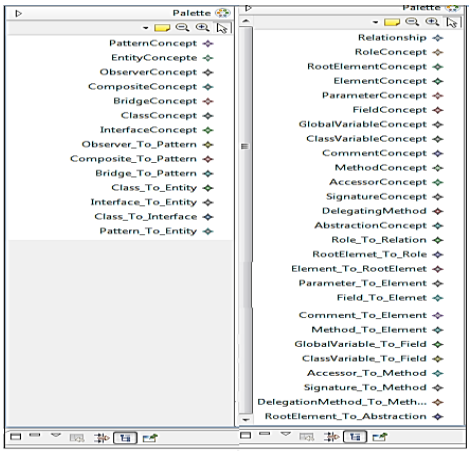


Figure 19. Generated Tools by Partial Meta-Models of PADL.

6. The Assessment Criteria

The essence of our proposed algorithm targets to partition meta-models, which are large size and less cohesion, to many smaller linked partial meta-models. So we can control them more easily. The developed process will be more comprehended and easier. The produced tools also will be more easy to use.

To be able to test our algorithm in improvement quality for software engineering processes, which is depending on MDE, we need to compare between the produced tools by meta-models before using partitioning algorithm and after it.

We depend on many Criterions for comparing, there are:

1- **Usability:** It is measured by the average Number of Button Clicks, which is required to create a model by using the produced tools.
2- **The clarity and the comprehension:** they are measured by the required time to understand and use all produced tools and by number of requested times to the teaching files and documentations which explain the use of tools.
3- **Robust:** It is measured by number of fatal errors during the use of these tools .
4- **Reuse:** The partitioning achieved extra specializing and customizing in generated tools, which make us able to reuse these specialized tools.

These criteria apply to the international quality software criteria ISO [29, 30].

We had given many meta-models to 100 students under graduated. Students generated tools from these meta-models. Then we gave them many partial meta-models which were resulted by partitioning algorithm. The students also generated the convenient tools. Students gave their assessments and wrote notes about tools performance in the two situations before and after the partition. We drew diagrammatic charts to explain the results.

Figure [20] illustrates the reducing in times needed for the use.

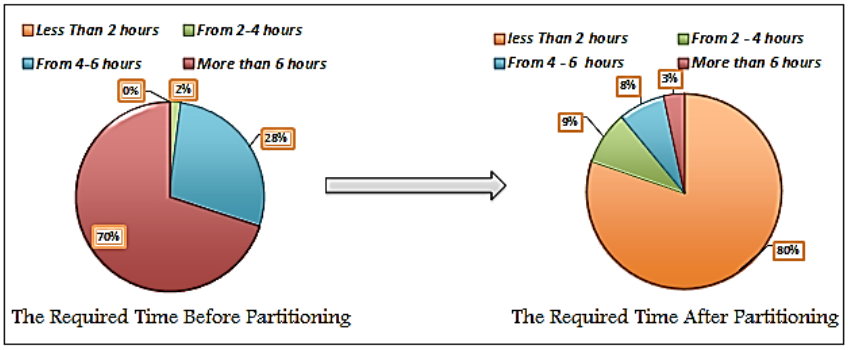


Figure 20. The Required Time to create a model.

As we have seen, there is a **clear improvement of the needed time**. Table [1] displays values.

Table [1]. The Ratio of Students According to the Required Time.

Before Partitioning				
Number of hours	*Less than 2*	*2-4*	*4-6*	*More than 6*
Ratio of Students	0%	2%	28%	70%
After Partitioning				
Ratio of Students	80%	9%	8%	3%

The chart [1] explains the Time Enhancing.

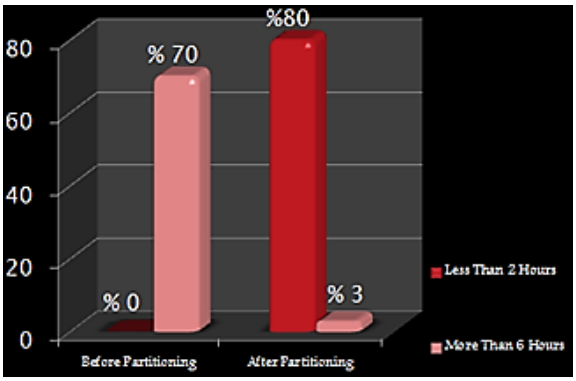


Chart [1]. The Improvement of Time.

Figure [21] illustrates the reducing of number of clicks, which is required to create a specific concept on average. It refers to the increase of the easiness of use.

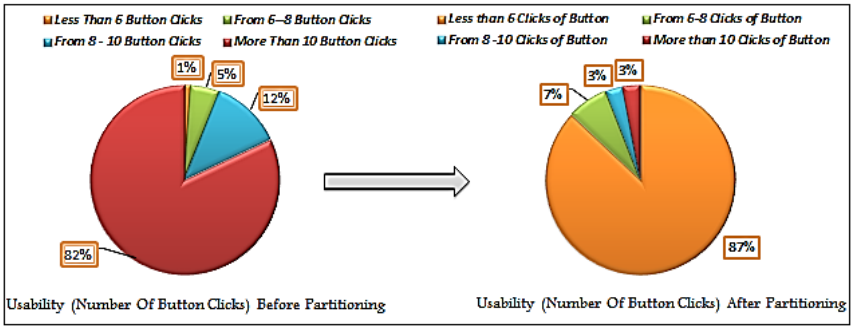


Figure 21. Number of Clicks on average to create an element.

As we have seen, there is a clear **improvement of the Number of clicks** which refers to the usability. Table [2] displays values.

Table [2]. The Ratio of Students According to the Number of Button Clicks.

Before Partitioning				
Number of Clicks	*Less than 6*	*6-8*	*8-10*	*More than 10*
Ratio of Students	1%	5%	12%	82%
After Partitioning				
Ratio of Students	87%	7%	3%	3%

The chart [2] explains the Usability Enhancing.

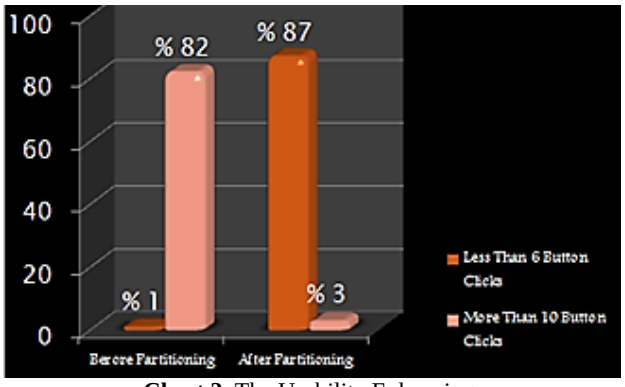


Chart 2. The Usability Enhancing.

(Figure22) illustrates the reducing of using documentations and teaching files after the partitioning. Tools understanding and clarifying are increased.

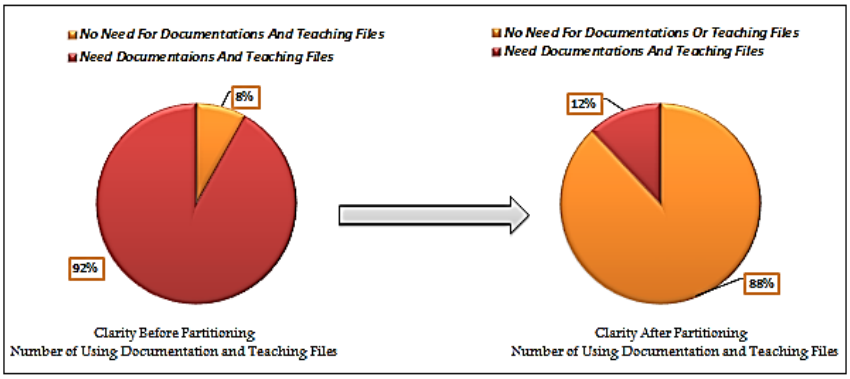


Figure 22. The Number Times of Using Documentations and Teaching Files.

As we have seen, there is a clear **improvement of the Number Times of Using Documentations and Teaching Files** which refers to the Clarity. Table [3] displays values.

Table [3]. The Ratio of Students According to the Number Times of Using Documentations.

Before Partitioning		
	Using Documentation	**No Using of Documentations**
Ratio of Students	92%	8%
After Partitioning		
Ratio of Students	12%	88%

The chart [3] explains the enhancing in using documentation and teaching files, which refers to the increase of clarity.

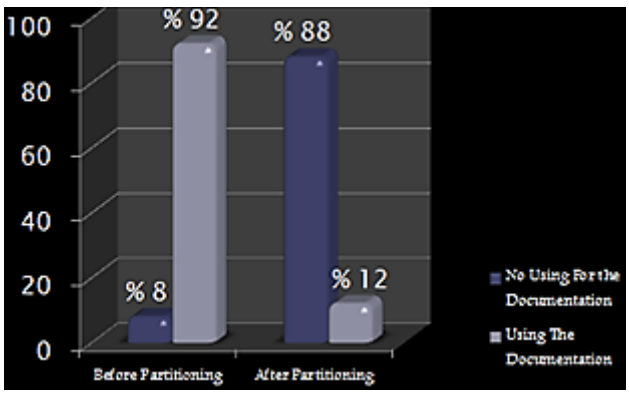


Chart 3. The Clarity Enhancing.

It clearly appears that the proposed partitioning algorithm achieved the quality criteria deservedly. The table [4] summarizes that.

Table [4]. The Total Quality Criteria

Quality Criterions	Explanation
Usability	Number of button clicks in average for creating an element was reduced to less than 6 clicks after partitioning.
Understandability and Clarity	The time to understand and use the meta-model was reduced to less than 2 hours after partitioning. Using of documentation and teaching files ratio was reduced 80% after partitioning.
Robustness	No fatal errors when we use it. Meta-models were tested by more than 100 times after partitioning. There are no fatal errors.
Reusability	Reusability was achieved because we partition the big Meta-Models to the smaller, more specialized and more customized Meta-Models. So we can use these partial Meta-Models in other development procedures.

7. Conclusion and Perspective

In this paper, we presented an algorithm to improve quality of MDE software procedures by separation and partitioning the big size and less cohesion meta-models to more small and less cohesion partial meta-models.
We inspired our views from Code Smells and God Class smells. We extended these smells from the low level to the high level (Meta-Models levels) in MDE methodology.
We have collected the results of empirical studies that had lasted for six months by 100 students in the Faculty. We put rules and limits in an algorithm to part Large Meta-Models to many partial Meta-Models according to the increase of the quality. We proved of the increase of quality by students' assessments.
This act is considered as a pioneer one, because most studies and researches in MDE field focus on the (wide expansion) without improving the procedures quality and the criteria of the quality. Our work is considered as a vertical expansion that improves quality.
As a perspective, we can expand to cover other kinds of links such as undirected association relation and recursive relations. We can add the numbers to independent the sides of a relation (one to one) or (one to many) as a weighed value.
We can make the partition not only on the structural approach. We can expand it to cover the semantic approach through lexical analysis by using ontologies.
Consequently, we will get more accurate partitioning algorithm.
Finally, we suggest simulate to other smells in the code and design level (Low Level) and expand them to the Meta-Model level (High Levels).
As also, we can open horizons for simulate other smells and developing other criterions for measuring the quality and ascertaining of its increase.

References

[1] Brown, W. J., Malveau, R. C., Brown, W. H. , McCormick III, H. W., Mowbray, T. J.: Anti-Patterns: Refactoring Software, Architectures, and Projects in Crisis .*John Wiley and Sons*, 1st edition, 1998.

[2] Moha, N., Gu´eh´eneuc, Y. G., Duchien, L., Le Meur, A. F.: DECOR: A Method for the Specification and Detection of Code and Design Smells, *IEEE Computer Society,* Montreal, 2010.

[3] Rahman, F., Bird, C., Devanbu, P.: What is that Smell?,*MSR (72-81). Cape Town*, South Africa, 2010.

[4] Tichy, M., Krause, C., Liebel, G.: Detecting performance bad smells for Henshin model transformations. Sweden: University of Gothenburg, 2011.

[5] Baker, P., Loh, Sh., Weil, F.: Model-Driven Engineering in a Large Industrial Context — *Motorola Case Study. Berlin, Heidelberg, Springer-Verlag*,2005.

[6] France, R., Rumpe, B.: Model driven development of complex software: A Research Roadmap. *Future of Software Engineering, IEEE The Computer Society,* 2007.

[7] Mohagheghi, P., Dehlen, V.: Where is the Proof? – A Review of Experiences from Applying MDE in Industry. *Proc. 4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA'08)*, 2008.

[8] Hutchinson, J., Rouncefield, M.,Whittle,J.: Model Driven Engineering Practices In Industry. *UK: School of Computing and Communications in Lancaster University*, 2009.

[9] KOSAYBA, B.: A framework for Model Driven Production of Graphic ModelingTools. Damascus, Syria, *IEEE ICCTA*, 2006.

[10] Saraiva, J., Silva, A.: CMS-based Web-Application Development Using Model-Driven Languages. Lisboa, Portugal, *ACM*, 2010.

[11] Misbhauddin, M., Alshayeb, M.: Towards a Multi-view Approach to Model-driven Refactoring. Dhahran, Saudi Arabia: King Fahd University of Petroleum and Minerals, African Conference for Software Engineering and Applied Computing, 2012.

[12] KOSAYBA, B.: Towards Standard Control Protocol through Internet Using MDE Approach. Damascus, Syria, *IEEE*, 2008.

[13] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.

[14] Marinescu, R.: Detecting Design Flaws via Metrics in Object Oriented Systems. TOOLS (39) (173-182). *IEEE Computer Society*, 2001.

[15] Marinescu, R.: Measurement and Quality in Object-Oriented Design (PhD Thesis). Timisora: "Politehenica": University of Timisora, 2002.

[16] Emden, E. V., Moonen, L.: Java Quality Assurance by Detecting Code Smells.*WCRE, 97-108*, 2002.

[17] Marinescu. R.: Detection strategies: Metrics-based rules for detecting design flaws. *Proceedings of the 20th International Conference on Software Maintenance, pages 350{359}. IEEE Computer Society Press,* 2004.

[18] Lanza, M., Marinescu, R.: Object-Oriented Metrics in Practice, *Springer*, 2006.

[19] Webster, B. F.: Pitfalls of Object Oriented Development, M & T Books, 1st edition, 1995.

[20] Riel, A. J.: Object-Oriented Design Heuristics. *Addison- Wesley*, 1996.

[21] Travassos, G., Shull, F., Fredericks, M., Basili, V. R.: Detecting defects in object-oriented designs: using reading techniques to increase software quality. *Proceedings of the 14th Conference on Object-Oriented Programming, Systems, Languages, and Applications, pages 47{56.ACM Press*, 1999.

[22] Munro. M. J.: Product metrics for automatic identification of bad smell design problems in java source-code. *Proceedings of the 11th International Software Metrics Symposium. IEEE Computer Society Press,* 2005.

[23] Alikacem, E. H., Sahraoui. H.: *Generic metric extraction framework.Proceedings of the 16th International Workshop on Software Measurement and MetrikKongress (IWSM/MetriKon*), pages 383{390}, 2006.

[24] Moha, N., Gu'eh'eneuc, Y.-G., Meur, A.-F. L., Duchien, L., Tiberghien, A.: From a domain analysis to the specification and detection of code and design smells.Formal Aspects of Computing (FAC), 2009.

[25] Vaucher, S., Khomh, F., Moha, N., Gu´eh´eneuc, Y. G.: Tracking Design Smells: Lessons from a Study of God Classes, *Ptidej Team Dept. de G´enieInformatiqueE´colePolytechnique de Montre´al, Canada.IEEE*, 2010.

[26] Arendt, Th., Kranz, S., Mantz, F., Regnat, N., Taentzer, G.: Towards Syntactical Model Quality Assurance in Industrial Software Development: Process Definition and Tool Support, Philipps-Universit¨atMarburg, Germany, 2011.

[27] Moha, N., Bouden, S., Gu'eh'eneuc, Y. G.: Correction of High-Level Design Defects with Refactorings, Department of Informatics and Operations Research University of Montreal, Quebec, Canada, 2007.

[28] Steinberg, D., Budinsky, F., Paternostro, M., Merks. E.: EMF: Eclipse Modeling Framework. Addison-Wesley, 2.edition, 2009.

[29] Stelzer, D., Mellis,W., Herzwurm,G.: A critical look at ISO 9000 for software quality management. *Software Quality Journal*, Germany, 1997.

[30] Pressman, R., Graw Hill, Mc.: Software Engineering — A Practitioner's Approach, 2001.

Affiliations

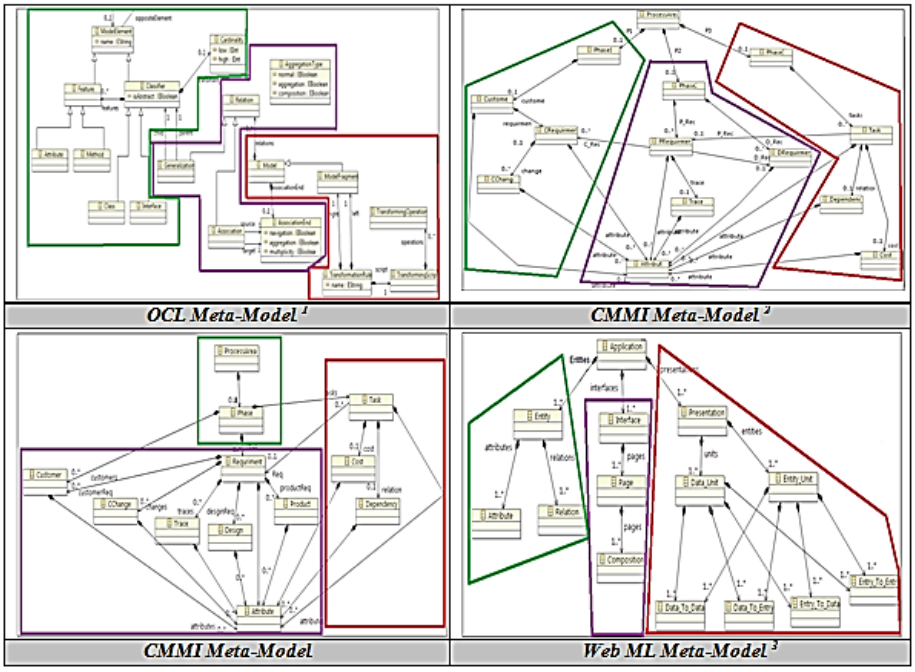
Bassem Kosayba
PhD, is an associated professor at Information Technology Engineering Faculty, Damascus University, Syria. E-mail: script.java@gmail.com.

Lubna Mansour
Master student at Information Technology Engineering Faculty, Damascus University, Syria. Email: lubna.mansour.88@gmail.com.

Appendixes

Appendix /A/

Partition the Meta-Models [14] [15] [16] ***logically, try to make the generated tools as usability and easy to understand as possible.***



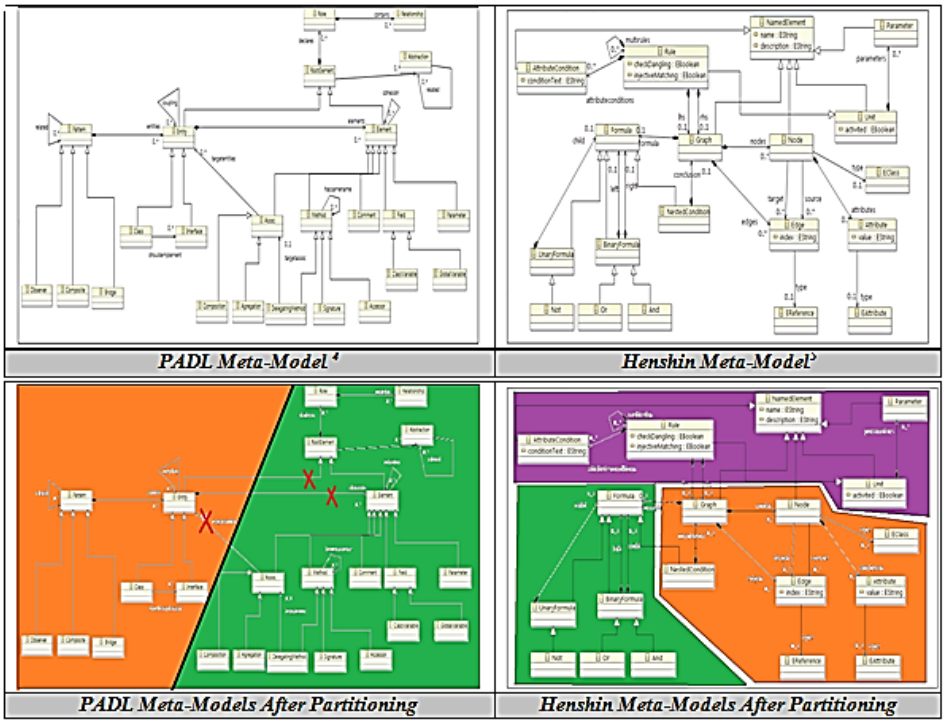
Appendix /B/

Assess the following Meta-Models [17] [18] ***before and after partitioning according to the empirical using to the generated tools.***

Regard the following assessment criteria.

- ***Number of hours to create your own model.***
- ***Number of button clicks on average to create an element correctly.***
- ***Number of using to the documentation and teaching files.***

[14] -Stolc, M., Polasek, I. (2010). A Visual Based Framework for the Model Refactoring Techniques, 8th IEEE International Symposium on Applied Machine Intelligence and Informatics, Slovakia.
[15] -Kulpa, P., Margaret, K. (2008). Interpreting The CMMI A Process Improvement Approach 2nd Edition.
[16]- Hennicker, R., Koch, N. (2002). Modeling the User Interface of Web Applications with UML, Germany.
[17]- Moha, N., Bouden, S., Gu'eh'eneuc, Y. G. (2007). Correction of High-Level Design Defects with Refactorings, Department of Informatics and Operations Research University of Montreal, Quebec, Canada.
[18]- Tichy, M., Krause, C., Liebel, G. (2011). Detecting performance bad smells for Henshin model transformations, University of Gothenburg, Sweden.



Assessment Criteria	*Before Partitioning*	*After Partitioning*
Number of hours to create your own model.	*More than six hours*	*Less than two hours*
Number of button clicks in average to create an element correctly.	*More than 10 button clicks*	*Less than 6 button clicks*
Number of using to the documentation and teaching files.	*Using documentation and Teaching Files*	*Using documentation and Teaching Files*

Do you feel that the quality is enhanced by partition the Meta-Models to many partial Meta-Models?

- *Yes, of course. Meta-Model is easier to understand, maintenance, use, clear, extend and flex.*

ورقة قبول النشر

مراجع البحث

[1] Hutchinson,J., Whittle,J., Rouncefield,M. (2009). *Empirical Assessment of MDE in Industry*, School of Computing and Communications in Lancaster University, UK.

[2] Baker, P., Loh, P.S., Weil, F. (2005). *Model-Driven Engineering in a Large Industrial Context - Motorola Case Study*, ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2005), Germany.

[3] Mohagheghi,P., Dehlen,v. (2008). *A Metamodel for Specifying Quality Models in Model-Driven Engineering*, Norway.

[4] Ganssle, J. (2008). *A Trillion Lines of Code?*, in Embedded.com.

[5] France, R and Rumpe, B. (2007). *Model driven development of complex software*, IEEE The Computer Society.

[6] Frankel, D. (2003). *Model Driven Architecture Applying MDA to Enterprise Computing*.

[7] Afonso, M., Vogel, R., and Teixeira, J. (2006). *From code-centric to model-centric software engineering: practical case study of MDD infusion in a systems integration company*, in Workshop on MBD/MOMPES.

[8] Anda, B., Hansen, K., Gullesen, I., and Thorsen, H. (2006). *Experiences from Introducing UML-based Development in a Large Safety-Critical Project Emprical Software Engineering.*

[9] Arisholm, E., Briand, L., Hove, S. E., and Labiche, Y. (2006). *The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation*, IEEE Transacions on Software Engineering.

[10] Briand, L., Labiche, Y., Di Penta, M., and Yan-Bondoc, H. (2005). *An Experimental Investigation of Formality in UML-based Development*, IEEE Transactions on Software Engineering.

[11] Lange, C. F. J., Chaudron, M.R.V. (2006). *Effects of Defects in UML Models: An Experimental Investigation, International Conference on Software Engineering.*

[12] Nugroho, A., Flaton, B.,.Chaudron, M.R.V. (2008). *An Empirical Analysis of the Relation between Level of Detail in UML Models and Defect Density*, in Model Driven Engineering Languages and Systems (MODELS).

[13] Dobing, B. and Parsons, J. (2006). *How UML is Used*, in Communications of the ACM.

[14] Forward, A. and Lethbridge, T. (2008). *Problems and opportunities for model-centric versus code-centric softwaredevelopment*, in Workshop on Models in Software Engineering (at ICSE).

[15] MediaDev. (2005*). Wide gap amongst developers' perception of the importance of UML tools.*

[16] The Middleware Company. (2003). *Model Driven Development for J2EE Utilizing a Model Driven Architecture Approach: Productivity Analysis (White Paper)*.

[17] The Middleware Company. (2004). *Model Driven Development for J2EE* Utilizing a Model Driven Architecture Approach: Maintainability Analysis (White Paper).

[18] Cruz-Lemus, J., Genero, M., Manso, M., and Piattini, M. (2005). *Evaluating the Effect of Composite States on the Understandability of UML Statechart Diagrams in Model Driven Engineering Languages and Systems (MODELS).*

[19] Razali, R., Snook, C. F., Poppleton, M. R., Garratt, P. W., and Walters, R. J. (2007). *Experimental Comparison of the Comprehensibility of a UML-based Formal Specification versus a Textual One*, in Conference on Evaluation and Assessment in Software Engineering (EASE).

[20] France, R and Rumpe, B. (2007). *Model driven development of complex software: A Research Roadmap. Future of Software Engineering*, IEEE The Computer Society.

[21] Genero, M., Piattini, M., and Manso, E. (2004). *Finding Early Indicators of UML Class Diagrams Understandability and Modifiability*, in International Symposium on Empirical Software Engineering (ISESE).

[22] Grudin, J. (1988). *Why CSCW applications fail: problems in the design and evaluation of organizational interfaces.* Proc. CSCW, New York. ACM.

[23] Reznicek, M., Lüdeling, A., Krummes, C., Schwantuschke, F., Walter, M., Schmidt, K., Hirschmann, H., Andreas, T. Das Falko-Handbuch. (2012). *Korpusaufbau und Annotationen Version 2.01*.

[24] Tichy, M., Krause, C., Liebel, G. (2011). *Detecting performance bad smells for Henshin model transformations*, University of Gothenburg, Sweden.

[25] Rahman, F., Bird, C., Devanbu, P. (2010). *What is that Smell?*, MSR (72-81), Cape Town, South Africa.

[26] Moha, N., Hacene, A., Gu´eh´eneuc,Y. G., Valtchev, P. (2006). *Refactorings of Design Defects using Relational Concept Analysis*, University of Montr´eal, Canada.

[27] Moha, N., Gu´eh´eneuc, G., Duchien, L., Franc¸oise, A. (2010). *DECOR: A Method for the Specification and Detection of Code and Design Smells*, Montreal.

[28] William F. Opdyke. (1992). *Refactoring Object-Oriented Frameworks.* Ph.D. thesis, Department of Computer Science, University of Illinois at UrbanaChampaign.

[29] Fowler, M. (June 1999). *Refactoring Improving the Design of Existing Code.* Addison-Wesley, 1st edition. isbn: 201-48567-2.

[30] Ducasse, S., Lanza, M.,, Tichelaar. S. (June 2000). *Moose: an extensible language independent environment for reengineering object-oriented systems*. In CoSET '00:Proceedings of the 2nd International Symposium on Constructing Software Engineering Tools.

[31] Tichelaar,S., Ducasse, S., Demeyer, S., and Nierstrasz, O. (2000). *A meta-model For language independent refactoring.* International Conference on Software Evolution, IEEE Computer Society Press.

[32] The Refactory Inc. (October 1999). *Refactoring browser.*

[33] XRef. XRefactory. (2000). *Previously Named XRef - Speller.*

[34] Amiot, H., Cointe, P., Gueheneuc, Y. (2002). *Un meta-modμele pour coupler application et detection des design Patterns*, Hermμes Science Publications, janvier.

[35] Moha, N., Bouden, S., Gueheneuc, Y. (2007). *Correction of High-Level Design Defects with Refactorings*, Department of Informatics and Operations Research University of Montreal, Quebec, Canada.

[36] Genero, M., Piattini, M., Calero, C. (2005). *A Survey of Metrics for UML Class Diagrams*, Journal of Object Technology.

[37] Lange, C.F. (2007). *Assessing and Improving the Quality of Modeling: A series of Empirical Studies about the UML*, Ph.D. thesis, Department of Mathematics and Computing Science, Technical University Eindhoven.

[38] Sunye, G., Pollet, D., Le Traon, Y., Jezequel, J. (2001). *Refactoring UML models*, in: Proc. UML 2001, Vol. 2185 of LNCS, Springer-Verlag.

[39] Porres, I. (2003). *Model Refactorings as Rule-Based Update Transformations*, in: Stevens, G. B. P., Whittle, J., Ed.), Proc. UML 2003: 6th Intern, Conference on the Unified Modeling Language, LNCS, Springer.

[40] Arendta, T., Mantzb, F., Taentzera, G. (2012). *EMF Refactor: Specification and Application of Model Refactorings within the Eclipse Modeling Framework.* ,aPhilipps-Universit¨at Marburg Germany.

[41] Brambilla, M., Cabot, J., Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*, Synthesis Lectures on Software Engineering, Morgan & Claypool.

[42] ABmann, U., Bartho, A., Burger, C., Cech, S., Demuth, B., Heidenreich, F., Johannes, J., Karol, S., Polowinski, J., Reimann, J., Schroeter, J., Seifert, M., Thiele, M., Wende, C., Wilke, C. (2012). *DropsBox: the Dresden Open Software Toolbox. Software & Systems Modeling*.

[43] Arendt, T., Taentzer, G., Weber, A. (2013). *Quality Assurance of Textual Models within Eclipse using OCL and Model Transformations* , Philipps-Universit□at Marburg, Germany.

[44] Olbrich,s., Cruzes,D. (2010). *Are all Code Smells Harmful? A Study of God Classes and Brain Classes in the Evolution of three Open Source Systems*, 26th IEEE International Conference Maintenance, Romania.

[45] Li, W., Shatnawi, R. (2007). *An empirical study of the bad smells and class error probability in the post release object-oriented system evolution*, Journal of Systems and Software.

[46] Moha, N., Khomh,F., Vaucher,S. (2010). *Tracking Design Smells:Lessons from a Study of God Classes*, Montreal, Canada.

[47] Khomh,F., Vaucher,S. (2009). *A Bayesian Approach for the Detection of Code and Design Smells*, 9th International Conference on Quality Software,Montreal, Canada.

[48] Dhambri, K., Sahraoui, H., Poulin, P., "Visual detection of design anomalies", IEEE Computer Society, Finland,2008.

[49] Simon,F., Steinbrauckner, F., Lewerentz, C. (2001). *Metrics based refactoring*, IEEE Computer Society, Washington, USA.

[50]Mantyla, M., Lassenius, C. (2006). *Subjective evaluation of software evolvability using code smells: An empirical study. Empirical Software Engineering*.

[51] Mantyla, M., Vanhanen, J., Lassenius, C. (2004). *Bad Smells Humans as Code Critics*, IEEE Computer Society.

[52] Marinescu, R. (2001). *Detecting Design Flaws via Metrics in Object Oriented Systems*, IEEE Computer Society.

[53] Emden, E. V., Moonen, L. (2002). *Java Quality Assurance by Detecting Code Smells*.

[54] Lozano, A., Wermelinger, M., Nuseibeh, B. (2007). *Evaluating the harmfulness of cloning: a change based experiment*, MSR, p18.

[55] Marinescu, R. (2002). *Measurement and Quality in Object Oriented Design (PhD Thesis)*, University of Timisora.

[56] Munro, M. (2005). *Product Metrics for Automatic Identification of Bad Smell Design Problems in Java Source-Code*. METRICS 05, p15.

[57] Szolovitz, P. (1995). *Uncertainty and decisions in medical informatics In Methods of Information in Medicine*.

[58] Cowell, R., Verrall, R., Yoon,Y. (2007). *Modeling operational risk with bayesian networks*, Journal of Risk and Insurance.

[59] Fenton, N., Neil, M. (2007). *Managing Risk in the Modern World : Applications of Bayesian Networks*, London Mathematical Society.

[60] Bauskar,B., Mikolajczak,B. (2004). *Modeling Inheritance Anomaly in Concurrent Systems using Colored Petri Nets*, IEEE International Conference on Systems, Massachusetts, Dartmouth.

[61] Cmogorac, L., Ramamohanarao, K.. (October 28,1995). *Inheritance Anomaly*.

[62] Matsuoka, S., Yonezawa, A. (1993). *Analysis of Inheritance Anomaly in Object Oriented Concurrent Programming Languages*, in: Research Directions in Concurrent Object Oriented Programing, MIT Press, pp 107-150.

[63] Kuno, Y. (October 1997). *Solving Inheritance Anomaly Problems By State Abstraction Based Synchronization.*

[64] Mnrata, T. (1989). *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, pp 541-580.

[65] Bauskar, B. (2003). *Integration of Object Oriented Design and Colored Petri Nets with Abstract Node Approach*, Masters Thesis, University of Massachusetts Dartmouth.

[66] Al-Ahmad, W., Steegmans, E. (1999). *Modeling and Reuse Perspectives of Inheritance Can be Reconciled*, American University of Sharjah, Department of Computer Science, IEEE International Conference on Systems.

[67] Madsen, O. (1993). *Object-Oriented Programming in the Beta Language*, Wokingham, England.

[68] Goldberg, A., Robson, D. (1989). *The Language and its Implementation*, Reading MA.

[69] Meyer, B. (1992). *Eiffel: The Language, Prentice-Hall, Englewood Cliffs (NI).*

[70] Arnold, J., Gosling, G. (1996). *The Java Programming Language*.

[71] Stroustmp, B. (1994). *The C++ Programming Language*, Second Edition, Reading MA.

[72] Snyder, A. (1987). *Inheritance and the Development of Encapsulation Software Components*, Research Directions in Object-Oriented Programming.

[73] Zhou, L., Chen, H., Zhang, Y., Zhou, C. (2008). *A Semantic Mapping System for Bridging the Gap between Relational Database and Semantic Web*, American Association for Artificial Intelligence (www.aaai.org).

[74] Moon, C., Baik, D., Wie, Y., Park, J. (2010). *The RDFS mapping for recursive relationship of relational data model*, Service-Oriented Computing and Applications (SOCA), IEEE International Conference.

[75] Choi, M., Moon, C., Baik, D. (2010). *Interoperability between a Relational Data Model and an RDF Data Model*, NCM2010: 6th International Conference on Networked Computing and Advanced Information Management.

[76] Jenkins,N. (2010). *A Project Managment Primer: Basic Principles Scope Triangle*, Australia.

[77] Zuse, H. (2003). *Resolving the Mysteries of the Halstead Measures*, Germany.

[78] Roger S. Pressman, (2000). *Software Engineering: A Practitioner's Approach* (European Adaptation), Fifth Edition.

[79] Malhotra,N., Pruthi,N. (2012). *An Efficient Software Quality Models for Safety and Resilience*, International Journal of Recent Technology and Engineering (IJRTE).

[80] Kitchenham, B., Pfleeger, S.L. (2002). *Software quality: the elusive target*, IEEE Conference, Manchester.

[81] Garvin, D. (1987). *Competing on the Eight Dimensions of Quality*.

[82] Stelzer, D., Mellis,W., Herzwurm,G. (1997). *A critical look at ISO 9000 for software quality management*, Software Quality Journal , Germany.

[83] Mohagheghi, P., Dehlen, V. (2007). *An Overview of Quality Frameworks in Model-Driven Engineering and Observations on Transformation Quality*, Norway.

[84] Faily, Sh., Fléchais, I. (2010). *A Meta-Model for Usable Secure Requirements Engineering*, UK.

[85] Wiele, T., Dale, B., Iwaarden, J. (2007). *Managing Quality*, 5th Edition.

[86] Mitra, J., Singh, C. (1999). *Pruning and Simulation for Determination of Frequency and Duration Indices of Composite Power Systems*, IEEE Transactions on Power Systems, Vol. 14, No. 3, pp. 899–905.

[87] Moha,N. (2008). *Refactoring of Design Defect using Relational Concept Analysis*.

[89] Hutchinson,J., Rouncefield, M.,Whittle,J., (2009). *Model Driven Engineering Practices In Industry*, School of Computing and Communications in Lancaster University, UK.

[90] Mohagheghi, P., Dehlen, V. (2008). *Where is the Proof? – A Review of Experiences from Applying MDE in Industry*. Proc. 4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA'08).

[91] Sanfilippo, A., Tratz, S., Gregory, M., Chappell, A., Whitney,, P., Posse, Ch., Paulson, P., Baddeley, B., Hohimer, R., White, A. (2006) . *Ontological Annotation with WordNet*, Pacific Northwest National Laboratory, 902 Battelle Blvd, Richland, PO Box 999, WA 99352, USA.

[92] World Wide Web Consortium, (2004). *Resource Description Framework 'RDF'*.

[93] World Wide Web Consortium, (2008). *SPARQL*.

[94] Banerjee, S., Pedersen, T. (2002). *Extended Gloss Overlaps as a Measure of Semantic Relatedness*, University of Minnesota, Duluth.

[95] Tichy, M., Krause, C., Liebel, G. (2011). *Detecting performance bad smells for Henshin model transformations*, University of Gothenburg, Sweden.

[96] Steinberg, D., Budinsky, F., Paternostro, M., Merks. E. (2009). *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2. edition.