من إظهار البرمجيات إلى إدراك البرمجيات*

المهندس قيس سمكري** الدكتور عمار جوخدار***

الملخص

أصابت صناعة البرمجيات أزمة ما بين عامي 1960 – 1980 وهذه الأزمة ما تزال مستمرة ولكنها تنمو في الخفاء. إن طبيعة المنتج البرمجي غير الوجودية هي السبب وراء عدم الانتباه إلى حقيقة وجود أزمة في صناعة البرمجيات. اعتمد الكثيرون على رؤية المنتج البرمجي من خلال آليات إظهار البرمجيات التي سمحت بالتجوال ضمن صور ومشاهد "لنماذج إظهار" metaphor أقل تعقيداً من ملفات الرماز البرمجي، كما سمحت بتركيب تخيل أو فهم لأحد أبعاد النظام البرمجي من خلال المسح اليدوي لحجم كبير من المُعطيات. إلا أن رؤية المنتج البرمجي وحدها لا تكفي وخاصة مع انتشار النظم المؤسساتية الموزعة الضخمة والمعقدة إذْ لا تغدو هذه النماذج أقل تعقيداً بكثير من الرماز البرمجي. لا بدَّ من أن تُحفز هذه الرؤية نموذجاً أو خبرة موجودة أصلاً ضمن العقل البشري وهو ما نسميه الإدراك. يتناول هذا البحث، كبديل للإظهار، آليات لإدراك البرمجيات تنقلنا من الرماز البرمجي إلى نموذج مكونات أسوة بالانتقال من نموذج الصور النقطية إلى نموذج الصور الشعاعية، واعتماده كحل حقيقي يخفف من حجم المُعطيات المراد معالجتها، ويسمح بالوصول المباشر إلى النتائج انطلاقاً من مهام محددة وعامة من مراحل هندسة البرمجيات مصنفة وفقاً لأدوار متنوعة مثل الهندسة العكسية بالنسبة إلى المحلل وضبط الأداء بالنسبة إلى المصمم وتشخيص الأخطاء بالنسبة إلى المُختبر وغيرها. أظهرت النتائج التجريبية لاستخدام آلية الإدراك من أجل مهمة إعادة هندسة نظام مؤسساتي موزع كبير نسبياً توفير 91% من الجهد والزمن اللازم لتنفيذ هذه المهمة.

الكلمات المفتاحية

إدراك البرمجيات Software Perception، إظهار البرمجيات Software Visualization، هندسة البرمجيات Software Engineering.

* أُعدَّ البحث في سياق رسالة ماجستير للمهندس قيس سمكري بإشراف الدكتور عمار جوخدار.

** قسم الذكاء الصنعي ومعالجة اللغات الطبيعية – كلية الهندسة المعلوماتية – جامعة دمشق.

*** أستاذ مساعد – قسم الذكاء الصنعي ومعالجة اللغات الطبيعية – كلية الهندسة المعلوماتية – جامعة دمشق

1 مقدمة

تتميز صناعة البرمجيات بأنها صناعة فكرية لا تقوم بتحويل أي مواد أولية. كما أن المنتج البرمجي له طبيعة غير وجودية لا تساعد على تقييم المنتج أو تقدير حجمه. ويدخل المنتج البرمجي مع مرور الزمن مرحلة الشيخوخة بسبب تغير أعضاء الفريق وأدوارهم الوظيفية والنقص التدريجي في المعرفة الخاصة بالمنتج لدى أعضاء الفريق الجُدد وتقادم التقارير والمخططات التي توصف مُتطلبات المنتج وازدياد صعوبة صيانة سطور رماز النظام البرمجي مع الزمن مما يؤدي إلى إعادة بناء البرنامج من الصفر.

أدت خصائص المنتج البرمجي، فضلاً عن الحجم والتعقيد والصعوبة التي نواجهها خلال تنفيذ جميع مراحل بناء نظم برمجية صحيحة ومفهومة من تحليل وتصميم وتنفيذ واختبار وتسليم وصيانة، إلى أزمة صناعة البرمجيات التي أصابت مشاريع تطوير البرمجيات في المدة ما بين عامي 1960 – 1980 إذْ تجاوزت المشاريع البرمجية الجداول الزمنية المحددة والموازنات المالية المرصودة لها، وتم تسليم الزبون جزءاً صغيراً مما تم الاتفاق عليه وكانت نوعيته منخفضة حيث احتوى على كثير من الأخطاء.

لم تنتهِ أزمة صناعة البرمجيات، وما تزال مستمرة. إذْ يعكس تقرير CHAOS لعام 2009 [18] والذي تصدره مجموعة Standich أن من بين كل ثلاثة مشاريع تطوير أنظمة برمجية، نجح مشروع برمجي، وأخفق المشروع الثاني، ونجح المشروع الثالث بعد أن كان مهدداً تماماً بالإخفاق حيث تجاوز المشروع الجدول الزمني المحدد له بنسبة 63%، وتجاوز الموازنة المالية المرصودة له بنسبة 45%، وتم تسليم نظام يحوي فقط 67% من الوظائف البرمجية المتفق عليها.

عمل كثير من الباحثين على إيجاد حل لهذه الأزمة، وكان من بين هذه الحلول مجال إظهار البرمجيات Software Visualization. سمح مجال إظهار البرمجيات للمستخدم بالتجوال ضمن صور ومشاهد لنماذج إظهار عكست أحد أجزاء النظام البرمجي عوضاً عن التجول ضمن ملفات الرماز البرمجي الصعبة، إلا أن المستخدم بقي في حاجة إلى مسح حجم كبير جداً من المُعطيات والبحث ضمنها من أجل تركيب يدوي لتخيل أو فهم لبُعد النظام الذي تم إظهاره. كما أن رؤية المنتج البرمجي وحدها لا تكفي ولاسيَّما مع انتشار النظم المؤسساتية الموزعة الضخمة والمعقدة. ولكي ننتقل من الرؤية للإدراك المباشر لا بد من أن تُحفز هذه الرؤية نموذجاً أو خبرة موجودة أصلاً ضمن العقل البشري، عندها فقط يستطيع العقل البشري حقيقة من إدراك ما يرى من أجزاء أو أبعاد النظام البرمجي، ومن ثمَّ تحقيق فهم حقيقي لأجزاء وأبعاد النظام.

2 البحوث المرجعية

اقترح مايرز [12] سنة 1990 تصنيف بحوث إظهار البرمجيات من حيث الهدف من عملية الإظهار، وشكل الإظهار، إظهار ساكن أو ديناميكي. في عام 1992 قدم ستاسكو وباتيرسون [13] تصنيفهم الذي احتوى تصنيف مايرز وأضافوا إليها بُعدين مهمين هما مستوى التجريد Abstractness: والذي يعكس سهولة فهم الصور والمشاهد، وسهولة إعداد

الأداة Automation: ويتدرج من التوليد الآلي لمشاهد الإظهار خلال عمل النظام، إلى عبء الإعداد اليدوي. بعد نحو السنة، بنى برايس [14] تصنيفاته وأضاف مفهوم مجال الإظهار Scope ليعبر عن:

- العمومية Generality: نوع البرمجيات التي من الممكن لأداة الإظهار أن تعمل معها.
- التصاعدية Scalability: قدرة الأداة على الصمود والعمل مع البرامج الصناعية الضخمة والكم الهائل من المُعطيات.

خلال العام نفسه قام رومان [15] بتعريف عملية إظهار البرمجيات بأنَّها مطابقة بين مكونات البرنامج وأشكال الإظهار، وقام بناء على تعريفه بتحديد اللاعبين الأساسيين في مجال إظهار البرامج وهم: المبرمج الذي قام بتطوير النظام، والمصمم الذي قام بتعريف أشكال الإظهار وبناء مطابقة بين أشكال الإظهار ومكونات البرنامج، والمستخدم النهائي وهو المستقبل لخرج الأداة. أعاد رومان صياغة خمسة محاور للتصنيف بناء على الاهتمامات والأعمال التي يقوم بها كل فاعل. في عام 2002 طرح جوناثان وفريقه [16] معايير تصنيف ركزت على بُعد المهمة من مراحل هندسة البرمجيات وشريحة المستخدمين وتُعد هذه التصنيفات من أكثر الصيغ ملاءمة لوقتنا الحالي. تلخصت المعايير في خمسة أبعاد (انظر الجدول رقم 1) هي:

1- Task لماذا نحتاج الإظهار، وتحدد نوع المهمة الموجهة لها هذه الأداة مثل مهمة فهم أداء النظام أو مهمة صيانة النظام أو غيرها،

2- Audience من هم شريحة مستخدمي الأداة، وتحدد نوعية الخبرة الوظيفية المطلوب توافرها لدى مُستخدم الأداة،

3- Target ما مصدر المُعطيات التي تم إظهارها، مُعطيات ساكنة ناتجة عن تحليل النظام، أم ملفات الرماز البرمجي، أم مُعطيات تنفيذية تم جمعها خلال عمل النظام،

4- Representation كيف كان شكل إظهار المُعطيات وما نموذج الإظهار الذي تم اعتماده،

5- Medium ما البيئة التي تم فيها الإظهار،

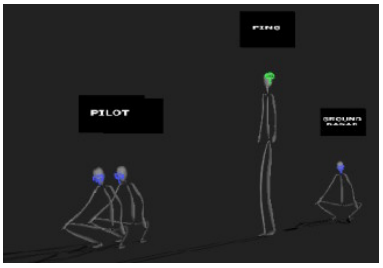
إلا أن ضخامة البرمجيات الحديثة وكبر حجم المُعطيات المرتبطة بها ودخول الإنترنت والتطبيقات الموزعة إلى قمة هرم البرمجيات الحديثة اليوم يدفعنا للتفكير بتوسيع جديد لمعايير تصنيف بحوث إظهار البرمجيات. قام هذا البحث بإضافة بُعد ملاءمة هذه الأدوات لحجم النظم المؤسساتية وبُعد ملاءمة هذه الأدوات لتكنولوجيا النظم الموزعة. كما قمنا بالتركيز على بُعد إعداد الأداة إذْ تشكل صعوبة إعداد الأداة، من حيث تهيئة الدخل وتشغيل الأداة وبناء شكل الخرج، أحد الأسباب الرئيسية وراء عدم استخدام هذه الأدوات.

جدول 1: معايير تصنيف أدوات إظهار البرمجيات

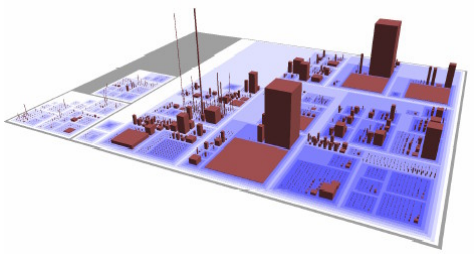
الباحث / البُعد	هذا البحث (2010)	جوناثان مالتك [16] (2002)	رومان [15] (1993)	برايس [14] (1993)	ستاسكو وباتريسون [13] (1992)	مايرز [12] (1990)
1	Task المهمة	Task المهمة	-	Effectiveness-Purpose الهدف	-	-
2	Audience شريحة المستخدمين	Audience شريحة المستخدمين	-		-	-
3	Target ماذا تم إظهاره	Target ماذا تم إظهاره	Scope المجال Abstraction مستوى التجريد	Scope المجال Content المحتوى	Aspect مصدر المُعطيات Abstractness مستوى التجريد	Abstraction مستوى التجريد
4	Representation شكل الإظهار	Representation شكل الإظهار	Interface شكل الواجهة Presentation شكل الإظهار	Form شكل الإظهار Interaction التفاعلية Effectiveness جودة نموذج الإظهار	Animation الحركة	Animation الحركة
5	Automation إعداد الأداة		Specification method طريقة التوصيف	Method طريقة الإعداد	Automation طريقة الإعداد	-
6	Medium أين تم الإظهار	Medium وسيط الإظهار	-	Form وسيط الإظهار	-	-
7	Fits enterprise systems تلاؤم حجم النظم المؤسساتية	-	-	Scope-Scalability نوعية البرامج القادرة على إظهارها	-	-
8	Fits distributed systems تلاؤم تكنولوجيا النظم الموزعة	-	-	-	-	-

بناءً على هذه المعايير قمنا بإسقاط مجموعة من التصانيف المختلفة ضمن بيئة المقارنة الجديدة وكانت النتائج كما يأتي.

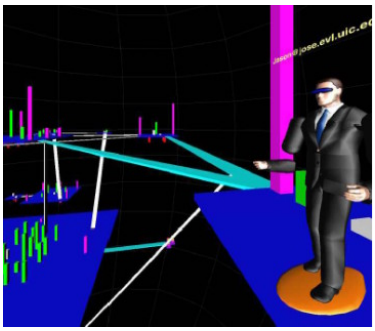
بالنسبة إلى بُعد "مصدر المُعطيات التي تم إظهارها Target" فقد اهتم جزء من البحوث بإظهار البُعد الساكن من تحليل النظام الذي يمثل الصفوف والعلاقات بين الصفوف والتوابع المعرفة ضمنها وحالات استخدام النظام. قام جوناثان مالتيك [1] بإظهارها على شكل بيئة افتراضية احتوت على مستويات متقاطعة (الشكل رقم 3). اعتمد إيكونك ماكارثي [7] شكل شجرة هرمية يمكن تصغير أجزاء منها وتكبيرها. تخيلها ريتشارد فيتل [10] على شكل ضواحٍ ومدن سكنية احتوت على أبنية بأحجام مختلفة (الشكل رقم 2)، أما بيل توملينسون [11] فقد اعتمد نموذج العلاقات الاجتماعية من أجل إظهار حالات استخدام النظام البرمجي على شكل تفاعل بين مجموعة من الوكلاء agents القادرين على التحرك ضمن المشهد من أجل تمثيل حالة الاستخدام (الشكل رقم 1).



الشكل 1: نموذج العلاقات الاجتماعية

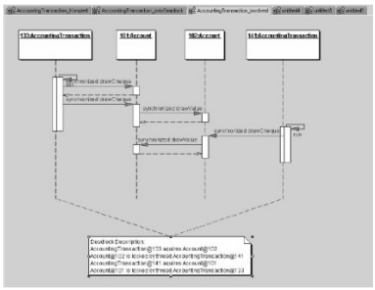


الشكل 2: نموذج المدن والضواحي السكنية



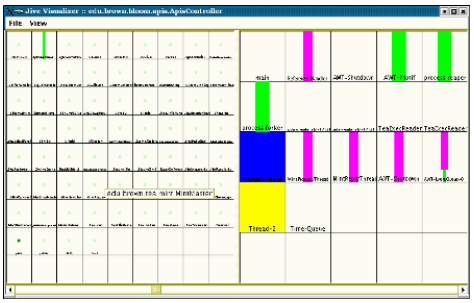
الشكل 3: بيئة افتراضية احتوت على مستويات متقاطعة

أما الجزء الآخر من البحوث فلم يتطرق لتحليل النظام كله كمصدر للمُعطيات، بل قام بإظهار معلومات ديناميكية تم جمعها خلال عمل النظام عن تنفيذ سطور رماز النظام التي مرَّ عليها الاستدعاء، حيث استعانت كاترينا ميهنر [2] بمخططات الـ UML: مخطط تدفق الأحداث عبر الزمن Sequence diagram (الشكل رقم 4) ومخطط تفاعل الأحداث Collaboration diagram، أما ستيفن رايس [4][5][8] فقد رسم أسماء الصفوف البرمجية والتوابع التي مر عليها الاستدعاء على شكل مخططات صندوقية (الشكل رقم 6). في حين أظهرت البحوث [3][9] الأغراض المنشأة في ذاكرة النظام والعلاقات فيما بينها على شكل عُقد بيان Graph وكرات ملونة في نموذج جزيئات المادة (الشكل رقم 5) على الترتيب. أما الأداة JACOT [6] فأظهرت كيفية انتقال النيسب Thread من حالة إلى أخرى خلال عمل النظام.



الشكل 4: مخططات UML لإظهار وجود Dead lock

الشكل 5: إظهار الأغراض على شكل جزيئات المادة



الشكل 6: إظهار صندوقي لعدد مرات استدعاء تعليمة من صف

أما بالنسبة إلى بُعد "المهمة Task"، فقد اهتمت الأدوات التي تم بناؤها ضمن البحوث [1][3][4] بدعم مهام التطوير، واهتمت البحوث [1][3][7] بمهام صيانة نظام برمجي. دعمت الأدوات المبنية ضمن

البحوث [2][6] مهمة اكتشاف أخطاء محددة جداً. ركزت الأدوات [5][8] على مهمة فهم أداء النظام. ومهمة فهم حالات استخدام النظام دعمها البحث [11]، أما البحث [9] فاهتم بمهمة فهم سلوك النظام والبحث [10] اهتم بمهمة فهم بنية النظام.

بالنسبة إلى بُعد "إعداد الأداة Automation"، فقد كان إعداد الأداة والمطابقة بين المُعطيات التي تم إظهارها وأشكال الإظهار ربطاً ثابتاً ومحدداً ولا يمكن تغييره ولم يغطي كامل أبعاد النظام. فما تم إظهاره كان إما ثابتاً أو له نموذج محدد [2][3][6] أو يتم تحديده يدوياً [4][5][8]، وشكل الإظهار محدد ولا يحوي دلالة أو يتم بناؤه يدوياً من قبل المبرمج كما في [4].

3 الدراسة التحليلية للبحوث المرجعية

لا تناسب هذه البحوث حجم الأنظمة المؤسساتية اليوم وتعقيد بُنيانها Architecture الموزع مثل نظم إدارة الموارد المؤسساتية التي تعمل على الإنترنت web-based ERP (Enterprise Resource Planning)، حيث يحوي بُعدها التحليلي على نحو 1000 صف و100 000 علاقة وتابع ونحو 2000 حالة استخدام، ويحوي بُعدها الديناميكي على نحو مليون غرض Object (وهو رقم بسيط إذا أخذنا حالة الضغط المُطبق على محرك البحث Google أو على تطبيق Facebook أو نظام عالي الأداء Heavy duty مثل نظام فوترة اتصالات دولة)، وعدد كبير جداً من النياسب Threads التي تعمل معاً. لم تستطع هذه الأعمال حل مشكلة الحجم الكبير من المُعطيات الذي يربك المستخدم ولم تحمل أي دلالة Semantic تسمح للمستخدم بتمييز المُعطيات الوظيفية الناتجة عن تحليل وتوصيف النظام، عن المُعطيات غير الوظيفية والناتجة عن تصميم وتحقيق النظام، عن المُعطيات التي تم الاستعانة بها من مكتبات خارجية. يعود السبب في ذلك إلى أن هذه الدلالة بالأصل لم تنعكس ضمن سطور الرماز البرمجي Source code وبقيت في ذهن المُبرمج وذلك بسبب أن لغات البرمجة التقليدية مثل لغة Java لا تسمح بهذا الشيء إلا عن طريق كتابة التعليقات Comments غير القابلة للتنفيذ حول سطور الرماز البرمجي. ومن ثمَّ بقي على عاتق المهندس التجوال والبحث اليدوي ضمن هذه الحجم الكبير من المُعطيات من أجل فرزها ومحاولة تصنيفها وتركيب فهم أو تخيل محدود عن أحد أبعاد النظام من خلال نافذة صغيرة أو صورة محدودة أو مشهد لا يستوعب أصلاً هذا الحجم الضخم من المعلومات.

جميع أنواع المهام التي من أجلها تم بناء أدوات الإظهار واضحة ومحددة وذات أهمية عالية جداً حتى في يومنا الحالي، ولكنها تهتم فقط بمهمة محددة جداً من مراحل هندسة البرمجيات، الأمر الذي اضطر المستخدم إلى إعداد واستخدام مجموعة مختلفة من الأدوات، ثم محاولة فهم الرابط غير الواقعي الذي تم بناؤه بين الصور والمُعطيات التي تم إظهارها حتى يستطيع تركيب فهم لجوانب من النظام البرمجي.

استهدفت البحوث جميعها شريحة واحدة فقط من المستخدمين اختصرت على مبرمج أو مبرمج خبير. يعود سبب ذلك أن الرؤية والإظهار كان خالياً من الدلالة لأن النوايا والأسباب والمبررات التي أدت إلى اتخاذ قرارات تحليلية وتصميمية لم تنعكس ضمن سطور الرماز البرمجي النهائي الذي

كتبه المبرمج حيث لا تسمح لغات البرمجة التقليدية بإضافة هذه الأبعاد.

أكد راينر كوشكي [17] الصعوبات التي تواجه مجال إظهار البرمجيات والناتجة عن مسح حجم ضخم من المُعطيات، والبحث اليدوي من أجل تركيب نتيجة من أشكال وصور إظهار لا تعكس واقع النظام، وعدم جدوى أداة الإظهار للمهمة الموجهة لها، وصعوبة الإعداد والتكامل مع هذه الأدوات. استنتج راينر من الدراسة التي قام بها، والتي اشتملت على 83 باحثاً في مجال صيانة النظم والهندسة العكسية وإعادة الهندسة، وجود حاجة ماسة وضرورية لتصور جديد من أجل توسعة مجال إظهار البرمجيات أطلق عليه اسم إدراك البرمجيات Software Perception.

4 موضوع البحث

يتناول هذا البحث موضوع إدراك البرمجيات وعلاقته بإظهار البرمجيات ومدى حاجتنا إلى الانتقال من الإظهار إلى الإدراك ولاسيَّما مع انتشار النظم المؤسساتية الموزعة، أسوة بالانتقال من نموذج الصور النقطية raster كبير الحجم والفقير بالدلالة، إلى مستوى نموذج الصور الشعاعية vector صغير الحجم والغني بالدلالة. نسعى بذلك لتخفيف حجم المُعطيات التي يجب معالجتها وتحقيق وصول المستخدم المباشر لنتائج مهام محددة وعامة وضرورية من مراحل هندسة البرمجيات ومصنفة وفقاً لأدوار متنوعة، مثل إنتاج وتتبع مؤشرات عن تقدم المشروع البرمجي سابقة للحظة التسليم، وفهم بنية هذه النظم وفهم سلوكها من أجل السيطرة على عمليات صيانة النظام البرمجي، وتأمين آلية فعّالة وآمنة لنقل النظام بين أعضاء الفريق وتوثيق الخبرة المتراكمة مع الزمن وحفظها، والمساعدة في اكتشاف وتصحيح الأخطاء الوظيفية ورفع الأداء وتخفيض الكلفة وتحديد مواضع التضاربات ونقاط الاختناق وضبط النظام واختباره.

5 فكرة البحث

إن رؤية المنتج البرمجي وحدها لم تكن كافية فهي إن استطاعت تخفيف حجم المسألة إلى الربع مقارنة بالرماز البرمجي Source code، فإنه ما زال كبيراً جداً بسبب ضخامة الحجم التحليلي للنظم المؤسساتية الذي يحتاج المستخدم إلى مسحه والبحث ضمنه، وضياع الدلالة المتعلقة بالبُعد الفني لتكنولوجيا النظم الموزعة المعقد وصعب البناء. نحن بحاجة إلى الانتقال من مستوى إظهار البرمجيات إلى مستوى جديد يضمن:

- الانتقال من حجم كبير جداً من المُعطيات لا يمكن مسحه من قبل أي مستخدم إلى حجم مُعطيات أصغر بكثير، وذلك من خلال التعامل مع مكونات Components بدلاً من ملفات سطور الرماز Source code أو مخططات الصفوف والعلاقات Class Diagram
- الانتقال من مستوى دلالات Semantics ونيّات Intensions تحليلية وتصميمية مفقودة وضائعة لم تنعكس ضمن ملفات الرماز البرمجي، إلى مستوى غني بالدلالة ويُحفز خبرة وظيفية أو نموذجاً في العقل البشري، وهذا ما نسميه الإدراك Perception

- الانتقال من مجرد التجوال ضمن صور لنماذج إظهار والبحث والتركيب اليدوي للنتائج، إلى وصول مباشر إلى نتائج مرتبطة بمهام محددة تسمح باتخاذ القرارات الضرورية،
- ضمان استخدام الأداة من قبل المهندسين من خلال جعل الإعداد والضبط سهلاً جداً وشبه معدوم.

فكرة البحث هي الانتقال من مجرد الرؤية والتجوال والبحث عن النتائج إلى مستوى الإدراك والوصول المباشر إلى نتائج مهام. وهي تشبه -إلى حد كبير - انتقالنا من نموذج الصور النقطية raster كبير الحجم والفقير بالدلالة إلى نموذج الصور الشعاعية vector. إذْ قمنا ببناء الصورة من أشكال هندسية لها طوبولوجيا معروفة الخصائص مثل النقطة والمستقيم ومتعدد الوجوه والأضلاع، بدلاً من الـــ pixels. أدى ذلك إلى تخفيف حجم الصورة من جهة، وتبسيط تحقيق كثير من الخوارزميات مثل تعرّف محتوى الصورة وحساب المغلف المحدب وحساب لحظة ومكان اصطدام جسمين.

وتشبه فكرة البحث أيضاً انتقالنا من النصوص الحرة التي ليس لها بنية Unstructured data أو شكل محدد إلى مُعطيات لها بنية واضحة تماماً Structured data. من أجل ذلك استعملنا استمارات وجداول Tables تتألف من مكونات محددة وواضحة الدلالة مثل الاسم والكنية، مما خفف من حجم المُعطيات وبسط جداً تحقيق خوارزميات البحث عن النتائج والترتيب والمقارنة والربط.

بالطريقة نفسها نريد الانتقال من مستوى ملفات الرماز البرمجي إلى مستوى المكونات البرمجية، من أجل إدراك البرمجيات Software Perception. من أجل تحقيق ذلك نحتاج إلى وضع نموذج مكونات component model وبناء النظام بالاستفادة من خصوصية بيئات العمل التي لديها مكونات يمكن إعادة استخدامها. هذا الأسلوب سيخفف من حجم المُعطيات التي تجب معالجتها وسيولد دلالة تبسط جداً تحقيق مهام من مراحل هندسة البرمجيات مثل مهام الهندسة العكسية Reverse engineering بالنسبة إلى المحلل وصيانة النظام Software maintenance بالنسبة إلى المطور واكتشاف الأخطاء Debugging بالنسبة إلى المُختبر. ومن ثمَّ استبدال حاجتنا إلى التدخل البشري من أجل التجوال والبحث عن النتيجة، بالوصول المباشر إلى نتيجة مهمة محددة ومعروفة.

إن تنفيذ مهمة الهندسة العكسية لنظام يعمل وتم بناؤه منذ زمن طويل هي مسألة تحتاج إلى وقت وجهد كبيرين والنتيجة لن تكون دقيقة ولاسيَّما مع حقيقة أن ملفات توصيف النظام وتحليله باتت لا تمت بأي صلة إلى واقع النظام اليوم نتيجة قيام العديد من المطورين بالكثير من التعديلات البرمجية وعدم شرح هذه التعديلات ونقلها إلى ملفات توصيف النظام. أضف إلى ذلك حقيقة انتقال أعضاء فريق التطوير إلى مشاريع جديدة أو إلى شركات أخرى ومن ثمَّ تناقص الخبرة والمعرفة بتحليل النظام وتصميمه عند أعضاء الفريق الجديد.

6 آلية تحقيق نموذج لإدراك البرمجيات

من أجل تحقيق الانتقال إلى نموذج لإدراك البرمجيات نحن بحاجة إلى:

1. إيجاد نموذج مكونات تحليلية وتصميمية من خلال استخدام بيئات العمل التي توفر مكونات لنماذج تحليلية وتصميمية يمكن إعادة

استخدامها من قبل المطور. تكون بيئات العمل موجهة لمجال تطبيقات معينة وهي في حالتنا مجال التطبيقات المؤسساتية الموزعة، مثل نظم دعم القرار DSS (Decision Support System) ونظم إدارة المعلومات MIS (Management Information Systems) ونظم إدارة الموارد ERP (Enterprise Resource Planning) ونظم إدارة إجراءات العمل BPMS (Business Process Management System) ونظم إدارة قوانين وقواعد العمل BRMS (Business Rules Management System).

2. تحديد مجموعة من العمليات المهمة والضرورية التي تعكس مهام من مراحل هندسة البرمجيات وتصنيفها وفقاً لأدوار واضحة مثل الهندسة العكسية بالنسبة إلى المحلل ومراقبة أداء النظام بالنسبة إلى المصمم وصيانة النظام بالنسبة إلى لمطور.

3. أتمتة هذه المهام بشكل يسمح بوصول المستخدم المباشر إلى نتائج مهمة محددة.

من أجل تحقيق الخطوة الأولى اخترنا بيئة العمل الإكسير Elixir framework [21] وقمنا ببناء نموذج الإكسير الإدراكي ESPM (Elixir Software Perception Module) من أجل تحقيق فكرة البحث بالاعتماد عليها. بيئة العمل الإكسير لديها نموذج مكونات تحليل ونموذج مكونات تصميم يمكن إعادة استخدامهما من قبل المطور، ولها نموذج دلالي سبق واستخدم في تطبيقات مشابهة كتوليد واجهات العمل GUI's بشكل آلي [22] والعودة بالعكس من الواجهة إلى المكون. كما استخدم في التعرف وتفسير معاني جمل مكتوبة بلغة طبيعية Natural language ومن ثم تنفيذ محتواها [23]. بيئة العمل الإكسير هي بيئة عمل ناضجة ومطورة محلياً ويمكن التعديل عليها من أجل إثبات جدوى البحث.

من أجل الخطوة الثانية لتحقيق نموذج الإدراك، قمنا بدراسة المراحل الأساسية من هندسة البرمجيات واستفدنا من خبرات مُختصين في تسليم نظم مؤسساتية موزعة كبيرة نسبياً ضمن الجداول الزمنية المحددة والموازنات المالية المرصودة ومطابقة للمتطلبات الوظيفية التي تم الاتفاق عليها [24][25][26][27][28][29]. استخلصنا من هذه الدراسة جميع المهام الأساسية والضرورية لكل مرحلة وقاطعناها مع بعض وصنفنها حسب أدوار معروفة، ثم قمنا بتحديد نوع المعلومات التحليلية والتصميمية، التي تشكل الأسباب والمبررات والنيّات التحليلية والتصميمية التي تم اتخاذها خلال مراحل تطوير المشروع البرمجي، والتي تحتاجها كل مهمة بالنسبة إلى كل دور والتي تُحفز خبرة وظيفية عن النظام البرمجي موجودة أصلاً ضمن خبرة الدور المستهدف، والكافية من أجل تنفيذ المهمة مباشرة واتخاذ قرارات بناءً عليها. كانت النتيجة تعريف 21 مهمة أساسية يحتاجها أعضاء الفريق خلال مراحل تطوير النظم المؤسساتية وصنفناها حسب 8 أدوار لديها خبرات وظيفية مختلفة. المهام مصنفة وفقاً للأدوار (انظر الجدول رقم 2):

جدول 2: المهام من مراحل تطوير النظم البرمجية مصنفة حسب الأدوار

الدور	المهمة	
مدير المشروع	إنتاج مؤشرات عن تقدم المشروع	Progress indicators
مُحلل نظم	الهندسة العكسية	Reverse engineering
	إعادة الهندسة	Re-engineering
مُصمم نظم	فهم تصميم النظام	Software design comprehension
	إعادة تصميم النظام	Software design review
	ضبط الأداء	Performance tuning
	تشخيص الأداء	Performance diagnostics
	فهم سلوك النظام غير الوظيفي (من حيث استهلاك الموارد كالذاكرة memory والمعالج processor)	Software non-functional behavior understanding
مُطور	نقل الخبرة وفهم النظام (استلام وتسليم النظام داخلياً بين المطورين)	Software understanding
	الدعم الفني	Technical support
	صيانة النظام	Software maintenance
مُختبر	اختبارات وظيفية	Functional tests
	التحقق من المتطلبات	Verification
	اكتشاف الأخطاء	Debugging
مسؤول جودة	ضمان الجودة	Quality assurance
مدير النظام	إدارة النظام وظيفياً	Business administration
	ضبط أداء النظام	Technical administration
	فهم ومراقبة سلوك المستخدمين	Monitoring
مُستخدم	استلام النظام	Software delivery
	التأكد من عمل النظام	Validation
	معرفة وجود المستخدمين على النظام	Users availability

إلا أن عدد المهام كبير وقابل للزيادة وتنفيذ كل مهمة على حدة يحتاج إلى وقت وجهد كبيرين. حاولنا البحث عن الحد الأدنى من التقارير المرنة التي تؤمن كل المعلومات الضرورية من أجل تغطية جميع هذه المهام والوصول المباشر إلى النتائج، عوضاً عن أتمتة كل مهمة على حدة الأمر الذي يؤدي إلى عدد كبير من التقارير قد يربك المستخدم فيما بعد وخاصة في حال ازدياد عدد المهام مستقبلاً. كانت النتيجة توصيف 8 تقارير فقط تحوي كل المعلومات التي تحتاجها أتمتة هذه المهام وغيرها أيضاً. هذه التقارير هي:

1. تقرير مكونات التحليل.

الهدف الرئيسي من هذا التقرير هو فهم بنية النظام والسلوك الساكن له. يسترجع هذا التقرير كامل مكونات التحليل التي بُني منها النظام مع كل خصائصها دون أي ضياع أو تعديل عليها خلال

عمل النظام. يعطي معلومات عن المكون تتعلق بالخدمة التي يتبع لها ونوع المكون: مدخل entry يظهر ضمن القائمة الرئيسية للنظام، أو مهمة task، أو تقرير report، أو قاعدة عمل business rule، أو أمر order، أو عملية operation، أو مؤقت timer، ومعلومات تصميمية عن كيفية عمله وارتباطه بالمكونات التحليلية الأخرى. كما يُظهر معلومات تنفيذية عن تفاصيل عمل كل مكون ضمن مدة زمنية يتم تحديدها.

2. تقرير العلاقات بين مكونات التحليل.

الهدف الرئيسي من هذا التقرير هو فهم العلاقات الساكنة للنظام. يُظهر هذا التقرير معلومات تتعلق بالعلاقات بين صفوف تحليل النظام تحت الإدراك والمسارات الممكنة بين هذه الصفوف. كما يسترجع هذا التقرير تفاصيل البُعد التصميمي لكل علاقة.

3. تقرير مسار استدعاء مكونات النظام.

الهدف الرئيسي من هذا التقرير هو فهم السلوك الديناميكي للنظام من أجل إدراك ما يحدث داخل النظام وهو يعمل. يظهر في هذا التقرير جميع الثنائيات من المكونات التي تربطها علاقة ديناميكية من نوع استدعاء (مكون طالب ومكون مطلوب). المكونات في هذا التقرير مصنفة حسب المكون الأول أصل الاستدعاء.

4. تقرير حالة مكونات النظام.

الهدف الرئيسي من هذا التقرير هو فهم أداء النظام ككل من الناحية الفنية ومن الناحية الوظيفية وذلك من خلال فهم العبء الديناميكي المطبق على كل مكون من مكونات النظام التحليلية والتصميمية خلال مدة عمل النظام، والذي يعرف بعدد مرات طلب هذا المكون في الثانية أو نسبة الطلب على مكون مقارنة بالطلب على باقي مكونات النظام ضمن مدة محددة.

5. تقرير استهلاك المكونات للمُعطيات.

الهدف الرئيسي من هذا التقرير هو فهم أداء النظام الديناميكي وتشخيصه من منظور طبقة المعلومات. يظهر في هذا التقرير كيفية قيام مكونات النظام التحليلية والتصميمية باسترجاع select وإضافة insert وتعديل update وحذف delete أغراض من نمط مكونات الصفوف. يُظهر هذا التقرير تفاصيل تصميمية مثل التعليمة Statement بلغة المكون التصميمي ORM controller، فضلاً عن ذلك، يُظهر هذا التقرير معلومات تنفيذية عن الاستدعاء، مثل المدة الزمنية التي استغرقها تنفيذ هذه التعليمة.

6. تقرير متابعة جلسات المستخدمين.

الهدف الرئيسي من هذا التقرير هو فهم أداء النظام الديناميكي وتشخيصه من منظور طبقة المستخدم. يظهر في هذا التقرير معلومات تتعلق بالمستخدمين خلال عملهم على النظام ممثلة على شكل جلسات sessions. تعطي كل جلسة معلومات عن أسماء الصفحات والوظائف والعمليات التي تم استدعاؤها ضمن الجلسة ومعلومات تنفيذية عن عدد مرات الوصول ضمن كل جلسة وعدد مرات الوصول إلى كل مكون خدمة أو صفحة خلال مدد يتم تحديدها من قبل المستخدم.

7. تقرير لائحة المهام.

الهدف الرئيسي من هذا التقرير هو فهم الوضع الحالي للنظام من الناحية الوظيفية من خلال تحديد

عدد المهام المنفذة والمعلقة والتي هي بانتظار التنفيذ من أجل كل نوع مهمة تم تحليلها ضمن النظام قيد الإدراك. يضيف هذا التقرير إلى كل مهمة معلومات ديناميكية عن الأشخاص المخولين بتنفيذ هذه المهام والموجودين (يعملون) حالياً على النظام. كما يُظهر من أجل كل نوع مهمة مجموعة الخصائص التصميمية المرتبطة بها.

8. تقرير إدارة المؤقتات.

الهدف الرئيسي من هذا التقرير هو فهم سلوك العمليات الآلية وأثرها في سلوك النظام وتوقعها. يظهر في هذا التقرير معلومات حول كل نوع مؤقت timer تم تحليله ضمن النظام قيد الإدراك. يضيف هذا التقرير إلى كل مؤقت معلومات تصميمية ومعلومات تنفيذية تتمثل بتاريخ بدء عمل المؤقت وتاريخ آخر استيقاظ وتاريخ الاستيقاظ القادم.

7 حالات استخدام تقارير نموذج الإدراك

توضح هذه الفقرة كيف يمكن الاستفادة من التقارير الثمانية في تنفيذ جميع مهام هندسة البرمجيات المذكورة آنفاً.

حالة استخدام 1: يحصل "المُحلل" على جميع المكونات الوظيفية من تقرير "مكونات التحليل" لتنفيذ مهمة "الهندسة العكسية".

يستطيع المحلل الوصول مباشرة إلى جميع التقارير والخدمات والوظائف المعرّفة ضمن النظام من تقرير مكونات التحليل من أجل مهمة الهندسة العكسية ودون بذل أي جهد يدوي، إذْ إنَّ المعلومات تُظهر خصائص عن كل مكون (انظر الجدول رقم 3) والهدف من وجوده ومن ثمَّ تنتقل خبرة النظام مباشرة إلى جيل جديد دون طلب التدخل البشري من قبل شخص مُختص.

جدول 3: بعض من مكونات التحليل المؤلفة للنظام

تابع للغرض	اسم المكون	نوع المكون
فاتورة مبيع	إنشاء فاتورة مبيع	مدخل (أحد بنود القائمة الرئيسية)
دفعة مالية	حركة الصندوق	تقرير
إجازة	الموافقة على إجازة	مهمة

هذا لم يكن ممكناً من ملفات الرماز البرمجي حتى ولو تم مسح كامل ملفات النظام. السبب في ذلك يعود إلى أن الدلالة المتمثلة بالمُتطلبات الوظيفية والمُبررات التصميمية التي استدعت بناء الصفوف البرمجية لم تنعكس ضمن سطور الرماز التي كتبها المبرمج وبقيت موزعة في أذهان فريق التطوير المتغير مع الزمن، مما أدى إلى عدم قدرة المستخدم على التمييز بين الصفوف البرمجية الخاصة بالنظام نفسه وبين الصفوف البرمجية التابعة إلى مكتبات خارجية تم الاستعانة بها، أو التمييز بين الصفوف التحليلية والصفوف التصميمية. ولا يوجد ما يدل على أن مجموعة من الصفوف البرمجية تشكل مكوناً واحداً أو أكثرَ من مكون أو فقط جزءاً صغيراً منه.

هذا الشيء أصبح ممكناً الآن لأن ما تم إظهاره هو المكونات الحقيقية التي بُني منها النظام، بخصائصها ودلالاتها الكاملة دون إخفاء أو ضياع أي معلومة.

حالة استخدام 2: يتحقق "المُختبر" من تغطية متطلبات الزبون من تقرير "مكونات التحليل" لتنفيذ مهمة "التحقق من المتطلبات".

يستطيع المُختبر دعم الشركة في تفادي الوقوع في مطب تسليم نظام برمجي يحتوي على وظائف أقل مما تم الاتفاق عليه مع الزبون، وذلك من خلال

جرد جميع المكونات التابعة لخدمة معينة من تقرير مكونات التحليل، والتحقق من تغطيتها للمتطلبات الوظيفية للنظام.

مثال، إذا اخترنا "فاتورة مبيع" كنمط الغرض الأساسي الذي نسميه المعاملة data object في التقرير، فنحصل على كل التقارير والمهمات ومداخل الإنشاء ضمن القائمة الرئيسية التي ترتبط بفاتورة البيع (الجدول رقم 4)، ويمكن مقابلتها مثلاً مع التقارير المطلوبة من قبل الزبون. اتباع هذا الأسلوب يسمح أيضاً بإنتاج مؤشرات عن تقدم المشروع البرمجي وتقدير الحجم المتبقي من العمل لإنجاز المشروع برمته والبقاء ضمن الجداول الزمنية المحددة.

جدول 4: جرد لمكونات التحليل المرتبطة بخدمة المبيعات عن طريق المعاملة "فاتورة مبيع"

تابع للغرض	اسم المكون	نوع المكون
فاتورة مبيع	إنشاء فاتورة مبيع	مدخل
فاتورة مبيع	عرض فواتير المبيع	تقرير
فاتورة مبيع	متابعة عملية البيع	مهمة

هذا الرابط وهذه الدلالة هي حلقة مفقودة تماماً ضمن ملفات الرماز البرمجي، فمهما بحثنا ضمن السطور ومهما كان نموذج الإظهار الذي تم اعتماده لا يمكن استرجاع الدلالات والنيّات التحليلية والتصميمية لاحقاً، حتى المهندس نفسه ينسى مع الزمن الأسباب والمبررات التي دفعته لاتخاذ قرارات تحليلية وتصميمية محددة.

أصبح هذا الشيء ممكناً الآن وبزمن مهمل لأن المهندس لم يعد بحاجة لبذل جهد جديد من أجل فهم إظهار لا يعكس واقع النظام، وإنما ما تم استرجاعه خلال عمل النظام هو ما تم بناء النظام منه.

حالة استخدام 3: يحصل "المُطور" على تفاصيل المكون من تقرير "مكونات التحليل" من أجل مهمة "نقل الخبرة وفهم النظام".

يستطيع مطور لم يشارك لا بتحليل ولا تصميم ولا تنفيذ النظام من تجميع مكونات النظام التي تتبع لخدمة معينة وتصنيفها إلى مداخل تظهر في القائمة الرئيسية ومهمات وأوامر وقواعد عمل وعدّها، مما يساعده على تخيل حقيقي لحجم وتعقيد خدمات النظام مباشرةً وذلك خلال مهمة نقل الخبرة وفهم النظام البرمجي.

مثال: معرفة أن خدمة المبيعات هي المكون الأكبر حجماً في النظام، حيث يظهر من الشكل التالي (الشكل رقم 7) وجود 3 طرائق مختلفة لإنشاء فاتورة مبيعات جديدة، وينتج عن الإنشاء نوعان مختلفان من المهام التي تحتاج إلى متابعة من قبل الموظفين، كما يوجد 6 أوامر تؤثر في موارد الشركة كأوامر حجز كميات من القطع في المستودع وتحقيق الأثر المالي المقابل لعملية البيع، وتم تعريف 5 قواعد عمل لضبط إجراء خدمة المبيعات حسب سياسة الشركة الداخلية.

الشكل 7: عدد المكونات المرتبطة بخدمة المبيعات مصنفة حسب النوع

هذا الشيء لم يكن ممكناً من قبل، لأنه لا يوجد أي دلالة ضمن سطور الرماز تربط بين مدخل في القائمة الأساسية للنظام مع مهمة تظهر في صفحة المهام، ولا يوجد أي دلالة تربط بين عملية action تظهر في واجهة متصفح إنترنت html وبين إقلاع قاعدة عمل لوقف الاستدعاء في حالات معينة فقط مثل عدم السماح بالبيع بعد الساعة العاشرة مساءً.

أصبح هذا الشيء ممكناً الآن لأن النظام تم بناؤه من مكونات تحليلية هي المدخل والمهمة وقاعدة العمل وعملية، وتوجد دلالة تربط بين المدخل وعناصر القائمة الرئيسية، كما توجد دلالة تربط بين المهمة وصفحة المهام.

حالة استخدام 4: يحصل "المُطور" على تداعيات التعديل على صف تحليلي من تقرير "العلاقات بين مكونات التحليل" لتنفيذ مهمة "صيانة النظام".

يستطيع المطور الحصول مباشرة على أسماء جميع الصفوف التحليلية التي تتأثر نتيجة تعديل مطلوب على أحد صفوف تحليل النظام خلال مهمة الصيانة وذلك قبل القيام بالتعديل من خلال طلب تقرير العلاقات بين مكونات التحليل.

إن فهم تداعيات تعديل مطلوب قبل القيام به هو أمر مهم ولاسيَّما لدى إسناد المهمة إلى مطور لم يشارك أصلاً لا في تحليل ولا تصميم ولا تنفيذ النظام البرمجي. يسمح هذا الفهم باختصار وقت تنفيذ التعديل من خلال معالجة المطور لجميع الحالات دفعة واحدة مما يضمن للزبون نوعية محددة لا تراجع عنها.

مثال: معرفة جميع الصفوف التي تشير إلى الصف التحليلي شخص / شركة (انظر الجدول رقم 5).

جدول 5: أسماء العلاقات والصفوف التي تشير على الصف "شخص / شركة"

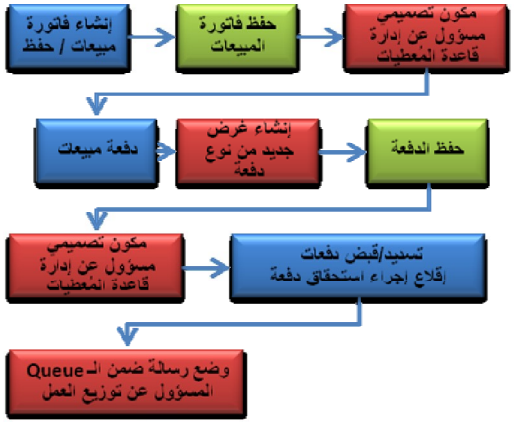
إلى الصف	من الصف	اسم العلاقة
شخص / شركة	دفعة مالية	من صالح
شخص / شركة	فاتورة مبيع	الزبون

معرفة هذه العلاقات من الرماز يحتاج إلى المسح اليدوي لكل المجتزآت البرمجية بحثاً عن العلاقات التي قد تكون موجودة ضمن صف برمجي يرث من صف آخر. قد يبدو أنه من السهل معرفة العلاقات النابعة من صف برمجي، ولكن من الصعب جداً معرفة العلاقات التي تصل إلى صف برمجي. حتى مع استخدام نماذج إظهار متطورة فإن حجم المسألة أكبر بكثير من مجرد التجوال ضمن هذه الصور والمشاهد.

حالة استخدام 5: يحصل "المُطور" على تداعيات التعديل على وظيفة من تقرير "مسار استدعاء مكونات النظام "لتنفيذ مهمة "صيانة النظام".

يحصل المطور على اكتشاف مباشر لتداعيات التعديل على إحدى خدمات النظام قبل القيام بأي تعديل خلال مهمة الصيانة من خلال طلب تقرير مسار استدعاء مكونات النظام.

مثال، إذا أردنا التعديل على وظيفة "تسديد/قبض دفعات"، فنحن بحاجة إلى معرفة كل المسارات التي تستدعي هذه الوظيفة، فنجد المسار الأول المنطلق من خدمة إنشاء فاتورة مبيعات / حفظ (كما في الشكل رقم 8)، والمسار الثاني المنطلق من تنفيذ مهمة اعتماد النسخة الأخيرة من الرواتب، إذاً لابد من التحقق من عمل هاتين العمليتين مباشرة بعد التعديل على وظيفة تسديد/قبض دفعات.

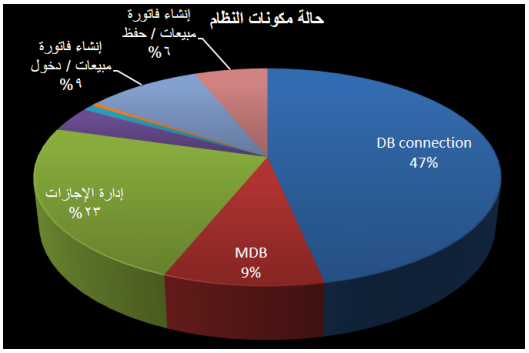


الشكل 8: أحد مسارات استدعاء المكون "تسديد / قبض دفعة" يبدأ المسار من خدمة إنشاء وحفظ لفاتورة مبيع

حالة استخدام 6: يحصل "المُصمم" على مراقبة حقيقية لأداء مكونات النظام من تقرير "حالة مكونات النظام" لتنفيذ مهمة "تشخيص الأداء".

يستطيع المصمم الحصول على معلومات ديناميكية من أجل تشخيص أداء مكونات النظام حتى ولو كان المكون الواحد مؤلفاً من عدة صفوف برمجية. يظهر في المثال (الشكل رقم 9) أن المكون DB connection هو أكثر المكونات التصميمية استهلاكاً للـ CPU مقارنة بباقي المكونات ضمن المدة التي حددها المصمم.

لا يمكن أن نحصل على هذه المعلومات نفسها من خلال جمع معلومات تنفيذية عن سطور الرماز البرمجي لأنه لا يوجد ما يدل أن مرور الاستدعاء من الصف البرمجي X أو الصف البرمجي Y أو الصف البرمجي Z له المعنى نفسه وهو الحاجة إلى الوصول إلى قاعدة المُعطيات من أجل استرجاع المُعطيات.



الشكل 9: توزع أداء النظام على المكونات المؤلفة له

سمح الاعتماد على مكونات نموذج تصميم بجمع المعلومات عن الصفوف البرمجية المؤلفة للمكون نفسهخلال عمل النظام، مما خفف كثيراً حجم المُعطيات وأظهر مباشرة توزع أداء النظام على المكونات المؤلفة له.

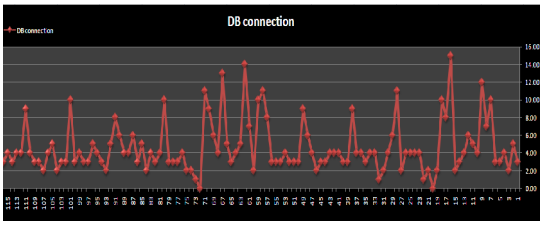
حالة استخدام 7: يحصل "المُصمم" على معلومات من أجل تنفيذ مهمة "ضبط الأداء" من تقرير "مكونات التحليل".

تبدأ مهمة ضبط أداء النظام بالاسترجاع المباشر لجميع الخصائص التصميمية لأوامر النظام ووظائفه وعملياته كما هي ودون أي فقدان أو تعديل عليها. في المثال، يمكن للمصمم مباشرة التأكد من أن الخصائص التصميمية المطبقة على أحد أوامر النظام مطابقة للممارسات الفُضلى best practice التي تعتمدها الشركة، كما يظهر في الجدول الآتي (الجدول رقم 6):

جدول 6: الخصائص التصميمية لأحد أوامر النظام

الخاصة التصميمية	القيمة / التعبير الرياضي
التنصت على المعاملة	"فاتورة مبيع"
قبل إنشاء غرض جديد	Void
بعد إنشاء غرض جديد	إنشاء دفعة مالية
في حال حدوث خطأ	التراجع عمّا تم تنفيذه

أيضاً، يمكن طلب تفاصيل عمل أي مكون ضمن مدة محددة من تقرير مكونات التحليل، فإذا اخترنا المكون الذي يؤمن الارتباط بقاعدة المُعطيات DB connection على سبيل المثال (مع العلم أنه من الممكن اختيار أي مكون آخر من مكونات النظام) فنلاحظ من الشكل التالي (الشكل رقم 10) أنه من المجدي رفع عدد الـــ db connection إلى 20 ضمن الـــ pool من أجل ضمان عدم انتظار أي طلب على مدخل قاعدة المُعطيات.



الشكل 10: عدد مرات استدعاء المكون DB connection ضمن الفترة الزمنية المحددة

لا يمكن الفصل والتمييز بين معلومات التحليل ومعلومات التصميم ضمن ملفات الرماز البرمجي لأن الدلالة بقيت لدى المحلل والمصمم ولم تنعكس في النظام، ولا يوجد ما يضمن أن سطور الرماز التي تم كتابتها يعكس ما تم الاتفاق عليه مع المبرمج.

الاعتماد على مكونات لها خصائص ودلالات معروفة وواضحة وإظهار هذه الخصائص مباشرةً عند الطلب هو ما يميز الحل المقترح.

حالة استخدام 8: يحصل "المُطور" على معلومات من أجل مهمة "تشخيص الأخطاء" من تقرير "مسار استدعاء مكونات النظام".

يستطيع المطور مباشرة اكتشاف الأخطاء debugging ضمن نظام موزع وذلك من خلال طلب تقرير مسار استدعاء مكونات النظام.

يُظهر الإجراء التالي (الشكل رقم 11) مكونات النظام التي مر عليها الاستدعاء الذي أدى إلى حدوث خطأ وصول متزامن على أحد فواتير المبيع. الخطأ ناتج عن عملية الحفظ الثانية الناتجة عن إقلاع الأمر "دفعة مبيعات مع حفظ"، إذْ إنَّ هذا الأمر قام بإعادة حفظ غرض قديم موجود في الذاكرة وهو مختلف عن الغرض الموجود في قاعدة المُعطيات نتيجة عملية الحفظ الأولى.

الشكل 11: مسار استدعاء مكونات النظام الذي أدى إلى وقوع خطأ

تماماً مدى الصعوبة في تتبع وفهم عُقد الإجراء الذي يعكس أحد مسارات استدعاء سطور الرماز البرمجي. أيضاً اكتشاف سبب الخطأ لن يكون بديهياً ويحتاج إلى جهد يدوي كبير من أجل محاولة فهم دلالة كل سطر

لم يكن من الممكن إظهار مسار الاستدعاء stack trace على شكل إجراء process واضح ومحدود الحجم من قبل بسبب الحجم الكبير من المُعطيات وكل من استخدم الأداة OptimizeIt تحت Jbuilder يُدرك

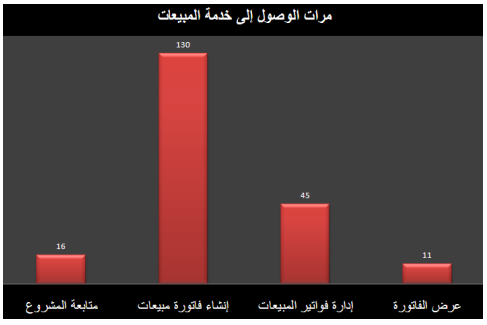
هذا الشيء لم يكن ممكناً في السابق، ويحتاج إلى تدخل يدوي ومسح حجم ضخم من المُعطيات من أجل العودة من الصفحة إلى الخدمة. السبب هو أن الصفحات التي يدخل إليها المستخدمون تم تطويرها بشكل مستقل عن التحليل والتصميم، والدلالة والنية وراء بناء هذه الصفحات بقي في ذهن فريق تصميم الصفحات.

إلا أن اليوم، اعتماد نموذج المكونات يولد دلالة تسمح بالانتقال من المكون إلى الواجهة والعودة من الواجهة إلى المكون.

حالة استخدام 10: يحصل "مدير النظام" على فهم سلوك الموظفين من أجل خدمة من تقرير "متابعة جلسات المستخدمين" لتنفيذ مهمة "إدارة النظام وظيفياً".

يستطيع مدير النظام الحصول مباشرة على فهم لسلوك الموظفين بالنسبة إلى خدمة معينة من تقرير متابعة جلسات المستخدمين من أجل إنتاج مؤشرات عن العبء الوظيفي الموجود لدى الشركة المستثمرة للنظام.

مثال، يظهر في الشكل التالي (الشكل رقم 13) أن الموظفين ينشئون عدداً كبيراً جداً من الفواتير 130 ولكن لا يوجد متابعة من قبلهم تتلاءم مع حجم الفواتير، فقط 45 دخول من أجل متابعة الفواتير و16 من أجل متابعة المشاريع الناتجة عن البيع، مما يعني وجود عبء على موظفي قسم المتابعة، ويمكن قد يكون من الصائب الاستعانة بموظف جديد من أجل مساعدتهم.



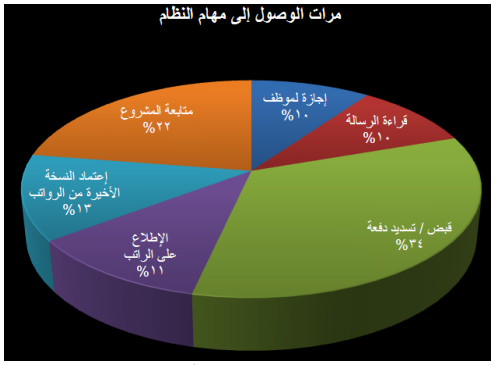
الشكل 13: سلوك الموظفين بالنسبة إلى خدمة المبيعات

برمجي مر عليها الاستدعاء. وأفضل حل كان هو الالتجاء لأحد المبرمجين الذين شاركوا بالتطوير.

الاعتماد على المكونات خفَّف جداً من حجم المُعطيات مما سمح بإظهار مسار الاستدعاء Stack trace على شكل process. كما أن تشخيص الأخطاء لم يعد يحتاج إلى فهم كل سطر برمجي من النظام، وإنما الفهم الأساسي لتحليل النظام كافٍ تماماً.

حالة استخدام 9: يحصل "مدير النظام" على فهم سلوك المستخدمين على النظام من تقرير "متابعة جلسات المستخدمين" لتنفيذ مهمة "فهم سلوك المستخدمين" ومراقبته.

يحصل مدير النظام مباشرة على مراقبة وفهم كامل لسلوك المستخدمين على النظام من تقرير متابعة جلسات المستخدمين. يعطي هذا التقرير معلومات كاملة عن سلوك المستخدم تتعلق بأوقات دخوله والصفحات التي استعرضها والعمليات التي نفذها ومحاولات الدخول والإخفاق. بمجرد معرفة الصفحات الأكثر طلباً من قبل كل مستخدم، نكون قد حصلنا على خدمات النظام التي تشكل الاهتمام الأكبر لكل مستخدم، واعتماداً على النتائج يمكن تصنيف المستخدمين إلى مجموعات والاستفادة من النتائج ضمن نظام CRM (Customer Relation Management) على سبيل المثال (انظر الشكل رقم 12).



الشكل 12: الصفحات الأكثر طلباً من قبل أحد المستخدمين

إن عدد مكونات التحليل لنظام الإكسير لإدارة الموارد البشرية هو نحو 750 مكوناً، ما بين أنماط معاملات ومداخل رئيسية وتقارير ومهام وقواعد عمل وأوامر ومؤقتات وعمليات. إعادة هندسة نظام إدارة الموارد البشرية بالطرائق التقليدية تعني أننا نحتاج إلى 750 * 60 = 45000 دقيقة، يعني 750 ساعة عمل، يعني 93 يوم عمل (على اعتبار يوم العمل مؤلفاً من 8 ساعات عمل)، يعني نحو 4 أشهر عمل (على اعتبار أن الشهر يتألف من 23 يوم عمل بين إجازات وعُطل).

باستخدام نموذج الإدراك أصبحت مهمة إعادة هندسة نظام الإكسير لإدارة الموارد البشرية يحتاج إلى 750 * 5 = 3750 دقيقة، يعني 62 ساعة عمل، يعني نحو 8 أيام عمل عوضاً عن 4 أشهر عمل. الربح في الوقت والجهد كان نحو 91%.

كما تم أيضاً استخدام هذا النموذج في أحد مهام الهندسة العكسية Reverse engineering لنظام الإكسير لإدارة موارد المؤسسة Elixir ERP (وهو أيضاً أحد المنتجات البرمجية التي تم إنشاؤها بالاعتماد على بيئة العمل الإكسير) ومن قبل مهندس لم يشارك لا في تحليل ولا في تصميم النظام، من أجل مهمة فهم جميع التداعيات التي قد تنتج عن تغيير مطلوب في أحد الصفوف التحليلية وهو "Item" فاحتاج إلى معرفة جميع العلاقات المباشرة وغير المباشرة التي تربطه أو تربط المجتزآت modules الأخرى به. كانت هذه المهمة تحتاج إلى عدة أيام عمل ومسح شامل للنظام في حين أن طلب تقرير "العلاقات بين مكونات التحليل" أظهر مباشرة وجود 25 علاقة توزعت بين المُجتزآت البرمجية:

- إدارة المستودعات Stock management
- إدارة سلاسل التزويد Sale chain management

هذا الشيء لم يكن ممكناً حتى من خلال المسح اليدوي لمُعطيات التنفيذ، لأن مدير النظام خبرته وظيفية وليست تقنية، ويحتاج إلى مبرمج خبرته فنية من أجل مساعدته على مسح المُعطيات وفرزها إلى مُعطيات مهمة وغير مهمة ثم محاولة يدوية لاستنتاج سلوك المستخدمين.

اليوم، توجد دلالة داخل النظام تسمح بتمييز مجموعة من الصفحات على أنها تشكل مكوناً تحليلياً واحداً فاستغنينا عن الجهد البشري، هذا المكون التحليلي يعكس خبرة وظيفية موجودة لدى مدير النظام فوصلنا مباشرة إلى النتائج.

8 مقارنة تجريبية بين الإظهار / الإدراك أو البحث والتجوال / الوصول المباشر

تم استخدام نموذج الإكسير الإدراكيESPM (الذي تم بناؤه من أجل هذا البحث) من أجل مهمة إعادة هندسة Re-engineering نظام الإكسير لإدارة الموارد البشرية Elixir HR (وهو أحد المنتجات البرمجية التي تم إنشاؤها بالاعتماد على بيئة العمل الإكسير) وذلك من قبل محلل نظم خبرته نحو 6 أشهر ضمن الشركة.

احتاج الوصول إلى كل مكون تحليل والإحاطة بكل المعلومات التصميمية حوله من النظام نحو 5 دقائق ناتجة فقط عن زمن طلب تقرير مكونات التحليل وزمن الاستعراض المباشر وقراءة النتائج، إذ إنَّ المعلومات كلها جاهزة ولا تحتاج إلى أي معالجة يدوية. في حين استغرق الوصول وجمع المعلومات نفسها ولكن من خلال التجول والبحث ضمن مخططات التحليل والتصميم وملفات الرماز البرمجي Source code وملفات قواعد العمل وملفات نشر النظام Deployment descriptors والبحث اليدوي عن التقاطعات بين المُجتزآت Modules وبين مخططات Diagrams المُجتزئ الواحد والربط مع ما يقابلها من صفوف برمجية من أجل الإحاطة بالنوع نفسه من المعطيات إلى نحو 60 دقيقة.

تقع عملية فلترة هذه النظم البرمجية وفهمها على عاتق المهندس نفسه، ومرد ذلك إلى ضياع المعلومات الدلالية الخاصة بمراحل التحليل والتصميم في أثناء كتابة البرنامج، إذْ لا تسمح لغات البرمجة التقليدية بإضافة البُعد الدلالي الضروري لإدراك البرمجيات. مثلاً، يتحول الصف التحليلي مثل "السيرة الذاتية" أو "CV" خلال مرحلة التحقيق Implementation والبدء بالبرمجة إلى عدة صفوف برمجية مثل Java classes، وقد تكون أسماء الصفوف البرمجية الناتجة عن التحقيق مُختلفة تماماً عن اسم الصف التحليلي الذي كان السبب في وجوده ضمن النظام، ومن ثمَّ لا توجد أي دلالة تربط بين مجموعة الصفوف البرمجية هذه والصف التحليلي "السيرة الذاتية"، مما يعني عدم قدرتنا على استرجاع هذه الدلالات خلال عمل النظام لأنها بالأصل غير موجودة. هذا يعني عدم قدرتنا مباشرةً على معرفة واكتشاف أن أحد التقارير التي تم توصيفها ضمن تحليل النظام من أجل إظهار جميع السير الذاتية له علاقة مُباشرة بمجموعة الصفوف البرمجية هذه، ومن ثمَّ فإن أي تعديل على أحد هذه الصفوف البرمجية يستدعي التحقق من أن التقرير لم يتأثر بالتعديل وأنه ما يزال يعمل. ضياع المعلومات الدلالية الخاصة بمراحل التحليل والتصميم في أثناء كتابة البرنامج يعني أننا بحاجة إلى تدخل أحد أعضاء الفريق الذي قام بالتطوير وبذل جهد يدوي وزمن ملحوظ من أجل البحث عن جميع التداعيات التي من الممكن أن تصيب النظام نتيجة هذا التعديل.

طرح هذا البحث مفهوم إدراك البرمجيات Software Perception وهو يهدف إلى إضافة والحفاظ على البُعد الدلالي للبرمجيات والمتمثل بمُعطيات تحليل النظام وتصميمه، واسترجاع هذا البُعد خلال عمل

- الصيانة وإدارة خدمات ما بعد البيع Maintenance management
- إدارة الاستهلاكات Consumption management
- إدارة الأصول الثابتة Fixed assets management.

هذه المسألة حُلت بدقائق، وكان الجهد المبذول لا يذكر، وهي مضمونة النتائج 100% مقارنة بالجهد اليدوي الذي قد يغفل بعض العلاقات.

9 الخلاصة

قد يكون إظهار البرمجيات بشكل بياني حلاً لمشكلة التجوال ضمن البرمجيات وذلك من خلال إظهار أحد أبعاد النظام البرمجي كالأغراض والتوابع والصفوف والعلاقات بينها، لكنه لا يحل مشكلة العدد الكبير من الأغراض والصفوف التي تُغرق المستخدم ولا تعطيه أي معلومات ذات صلة بالواقع لأنها بقيت في ذهن المبرمج وضمن ملفات توثيق النظام التي تتقادم مع الزمن. لم يسمح مجال إظهار البرمجيات للمُستخدم بالتمييز مباشرةً بين المكونات التحليلية الخاصة بالمحلل والناتجة عن مُتطلبات النظام ووظائفه مثل عمليات البيع والشراء وتسليم البضائع وقبض دفعات مالية وما يتعلق بها من تقارير ومهام، وبين المكونات التصميمية الخاصة بالمُصمم والتي تتعلق بتحقيق المُتطلبات غير الوظيفية للنظام مثل مكون المناقلة Transaction وتعليمات استرجاع المعطيات Query statements ومكون توزيع العمل Job، وبين غيرها من الصفوف البرمجية التي تم الاستعانة بها كمكتبات خارجية ومطورة من قبل موردين آخرين.

إدراك مؤلف من 8 تقارير أساسية سمحت بالوصول المباشر إلى نتائج تتعلق بمجوعة من التساؤلات الخاصة بكل نوع مهمة من مراحل هندسة البرمجيات.

استخدمنا نموذج الإدراك وكانت النتيجة توفير 91% من الوقت اللازم لمهمة إعادة هندسة نظام إدارة موارد بشرية، أما بالنسبة إلى الجهد الذي تم توفيره، فقد استطعنا الاستغناء عن الجهد البشري وتحقيق الوصول المباشر إلى نتائج مهمة هندسة عكسية من أجل اكتشاف تداعيات التعديل على صف تحليلي، ومن ثمَّ فإن التوفير بالجهد قد يصل إلى نسبة أعلى من 91%.

من سلبيات هذا البحث هو عدم إمكانية تطبيق نموذج الإدراك الذي تم بناؤه ضمن هذا البحث على أي نوع من الأنظمة والتطبيقات البرمجية المنتشرة اليوم. لا بد من أن تكون هذه الأنظمة والتطبيقات مبنية بالأصل وفق نموذج مكونات تحليلية ونموذج مكونات تصميمية وأن تحمل دلالات واضحة ومعروفة، فضلاً عن ضرورة توفر آلية تسمح لاحقاً باسترجاع المكونات التي بُني منها النظام مع كامل دلالاتها دون أي تعديل يطرأ عليها خلال عمل النظام.

نرى في المستقبل القريب تفرع هذا البحث باتجاه أتمتة وتحقيق المهام التي تم تعريفها اعتماداً على نموذج مكونات مختلف وتابع لبيئة عمل أخرى مثل Spring. أيضاً محاولة البرهان على إمكانية التقارير الثمانية الأساسية التي تم تعريفها من تغطية مهام جديدة من هندسة البرمجيات قد تظهر في الأعوام القادمة.

النظام والاستفادة منه عند الإظهار وعند الإجابة عن تساؤلات المُستخدم واختيار الإجابة التي تتلاءم مع الخبرة الوظيفية الموجودة لديه.

أُضيف البُعد الدلالي إلى النظام البرمجي من خلال استعمال بيئات عمل Frameworks تستبدل بناء البرمجيات بالطرائق التقليدية من كلمات مفتاحية Keywords وبُنى تحكم Control flow statements خاصة بإحدى لغات البرمجة Programming languages مثل Java و#C، ببناء البرمجيات انطلاقاً من مكونات تحليل (تقرير ومهمة ومدخل وأمر ومؤقت) ومكونات تصميم (Job event message and Listener) معروفة تؤمنها بيئات العمل ويمكن إعادة استخدامها من قبل المهندس، أسوة بطرائق بناء الصور الشعاعية من أشكال معروفة (النقطة والمستقيم ومتعدد الوجوه) بدلاً من بنائها من مجموعة من العنصورات Pixels. تسمح هذه البيئة بإضافة دلالات عن طريق تخصيص مخططات لغة التوصيف العامة UML ومن خلال تعريف مجموعة من التصنيفاتStereotypes والسمات tagged values. كما تقوم بتحميل هذه الدلالات ضمن محرك خاص يستطيع تفسير المكونات. هذا الأسلوب يسمح بتحديد طباق Mapping عكسي من مُعطيات تنفيذ تم جمعها خلال عمل النظام إلى مكونات التصميم ومنها إلى مكونات التحليل، والذي انعكس على شكل سمح للمهندس بطرح تساؤل عن أداء النظام مثلاً والحصول مباشرةً على جواب شافٍ بدلاً من التجول بين الأغراض والصفوف البرمجية بحثاً عن جواب ما.

قمنا بتحديد مجموعة من المهام التي يحتاجها كل مهندس مصنفة حسب خبرته الوظيفية مثل الهندسة العكسية وإعادة الهندسة واكتشاف الأخطاء والاختبارات الوظيفية وغيرها، وقمنا ببناء نموذج

المراجع

[1] J. Maletic, J. Leigh, and A. Marcus. (2001) Visualizing Software in an Immersive Virtual Reality Environment. In Proceedings of the ICSE'01 Workshop on Software Visualization, Toronto, pp. 49-54.

[2] K. Mehner and B. Weymann. (2001) Visualization and Debugging of Concurrent Java Programs with UML. In Online-Proceedings Workshop Software Visualization, International Conference on Software Engineering ICSE 2001.

[3] D. Heuzeroth and W. Loewe. (2002) Understanding a System's Architecture.

[4] S. P. Reiss and M. Renieris. (2003) The BLOOM Software Visualization System. In Software Visualization -- From Theory to Practice, MIT Press.

[5] S. P. Reiss. (2003) Visualizing Java in Action. In Proceedings ACM 2003 Symposium on Software Visualization.

[6] H. Leroux, C. Mingins and A. Réquilé-Romanczuk. (2003) JACOT: A UML-Based Tool for the Run-Time Inspection of Concurrent Java Programs. In Proceedings of the 1st Workshop on Advancing the State-of-the-Art in Run-Time Inspection.

[7] E. McCarthy and C. Exton. (2004) THORR: A Focus + Context Method for Visualising Large Software Systems. In the Proceedings of Program Visualization Workshop (PVW'04), Third ITiCSE (ACM).

[8] S. P. Reiss and M. Renieris. (2005) JOVE: Java as it Happens. In Proceedings of ICSE 2005.

[9] B. A. Malloy and J. F. Power. (2005) Using a Molecular Metaphor to Facilitate Comprehension of 3D Object Diagrams. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05).

[10] R. Wettel and M. Lanza. (2007) Visualizing Software Systems as Cities. In Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis).

[11] T. A. Alspaugh, B. Tomlinson, and E. Baumer. (2006) Using Social Agents to Visualize Software Scenarios. In Proceedings of the 2006 ACM symposium on Software visualization.

[12] Myers, B. A. (1990). Taxonomies of Visual Programming and Program Visualization. Journal of Visual Languages and Computing, Vol. 1, No. 1, 1990, pp. 97-123.

[13] Stasko, J. T. & Patterson, C. (1992). Understanding and Characterizing Software Visualization Systems. In Proceedings of the 1992 IEEE International Workshop on Visual Languages.

[14] Price, B. A., Baecker, R. M., and Small, I. S. (1993), "A Principled Taxonomy of Software Visualization", Journal of Visual Languages and Computing, vol. 4, no. 2, 1993, pp. 211-266.

[15] Roman, G.-C. and Cox, K. C. (1993), "A Taxonomy of Program Visualization Systems", IEEE Computer, vol. 26, no. 12, December 1993, pp. 11-24.

[16] J. I. Maletic, A. Marcus, M. L. Collard. (2002) A Task Oriented View of Software Visualization. In Proceedings of the IEEE Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002).

[17] R. Koschke (2003). Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. JOURNAL OF

SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE. J. Softw. Maint. Evol.: Res. Pract. 2003; 15:87–109 (DOI: 10.1002/smr.270).

[18] CHAOS summery (2009). www1.standishgroup.com/newsroom/chaos_2009.php.

[19] John D. Poole. (2001) Model-Driven Architecture: Vision, Standards And Emerging Technologies, ECOOP 2001. Position Paper Submitted to ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models.

[20] Model Driven Architecture, Krzysztof Czarnecki, University of Waterloo, (2003-2004) Czarnecki, Frankel, Graff, Helsen.

[21] A. Joukhadar (2008). EliXir: a framework for Building e-business applications. In Proceedings of Information and Communication Technologies: From Theory to Applications. ICTTA 2008.

[22] K. Samkari, A. Joukhadar (2008). Comparison Matrix for Web HCI. In Proceedings of Information and Communication Technologies: From Theory to Applications. ICTTA 2008.

[23] A. Joukhadar, H. Maghout (2008). Improving Agility in Business Applications using Ontology Based Multilingual Understanding of Natural Business Rules. In Proceedings of Information and Communication Technologies: From Theory to Applications. ICTTA 2008.

[24] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen & Juhani Warsta (2002). Agile Software development methods. Review and analysis. ESPOO 2002 VTT publications 478. 107 p.

[25] Walt Scacchi (2001). Process Models in Software Engineering. Final Version to appear in, J.J. Marciniak (ed.), Encyclopedia of Software Engineering, 2nd Edition, John Wiley and Sons, Inc, New York, 2001.

[26] Ricky don Preuninger (2006). The advantaged of implementing software engineering process models. Master of science in computer science and engineering. The university of Texas at Arlington.

[27] Tobias HILDENBRAND, Axel KORTHAUS. A Model-Driven Approach to Business Software Engineering.

[28] Jasmine K.S, Dr. R. Vasantha (2008). A New Process Model For Reuse Based Software Development Approach. Proceedings of the World Congress on Engineering 2008 Vol I.

[29] Jianli Dong (2009). Research Survey on Integrated Software Engineering Environment Based on Product Line. Proceedings of the Second Symposium International Computer Science and Computational Technology(ISCSCT '09) Huangshan, P. R. China, 26-28,Dec. 2009, pp. 016-019.

تاريخ ورود البحث إلى مجلة جامعة دمشق 2010/5/2.