هل يمكن أن نصل بصناعة البرمجيات المؤسساتية إلى مستوى الصناعات الهندسية الأخرى؟

د. عمار جوخدار1/1/2016

1 ملخص

تتحدث هذه المقالة عن تجربة الإكسير للبرمجيات الذكية في مجال صناعة البرمجيات المؤسساتية في سورية قبل وبعد الأزمة وذلك من خلال دراسة التحديات الخاصة بصناعة البرمجيات مقارنة بالصناعات الهندسية الأخرى عالمياً ومحلياً.

توضح المقالة الأسباب التي تجعل من صناعة البرمجيات المؤسساتية، التي لا تحتاج لمواد أولية، أكثر صعوبة وأقل استقراراً وأغلى ثمناً من صناعات تحتاج لمواد أولوية مرتفعة الكلفة مثل الأبنية والسيارات.

تشرح المقالة الحلول التي طورتها الإكسير منذ تأسيسها لخفض الكلفة والزمن مع رفع النوعية وكيف تفاعلت مع الأزمة.وطورت هذه الحلول لتتماشى مع ظروف الأزمة.

2 مقدمة

تشترك المشاريع الهندسية (ميكانيك، مدني، عمارة، معلوماتية، كهرباء) بثلاثة خصائص حساسة جداً وهي **الكلفة والوقت والنوعية** ويوجد خصائص أخرى ثانوية مرتبطة بها وهي قابلية الصيانة، وإدارة المخاطر وصعوبات التحقيق وغيرها.

قد يبدو أن صناعة البرمجيات أسهل من الصناعات الأخرى ومما يوحي بذلك أن عدد شركات البرمجيات أكبر بكثير من شركات صناعة السيارت والبناء وغيرها. ولكن المفارقة هي أن أسعار البرمجيات المؤسساتية أغلى بكثير من أسعار السيارت والمنازل.

من حيث **الكلفة** يمكنا أن نلاحظ أن **أسعار** السيارات تتراوح من 10 ألاف دولار إلى 100 ألف دولار، وتتراوح أسعار الشقق من 20 ألف إلى 100 ألف دولار في حين تتراوح أسعار البرمجيات المؤسساتية، مثل نظم ERP، بين 100 ألف دولار إلى مليون دولار كما هو الحال مع Oracle financial وSAP دون النظر إلى الكلف المخفية وكلف التهجير والتدريب وتصل وفقا للتخصيص المطلوب إلى عشرات الملايين من الدولارات.

من حيث **النوعية** فمن الواضح أن الصناعات الأخرى أكثر استقراراً حيث لا تحتاج السيارت والمنازل لإعادة إقلاع كل فترة ولا تتعرض لنفس الفيروسات والإختراقات. أما من ناحية المخاطر فإننا لا نجد من يطلب عقد ضمان وصيانة للمنزل في حين يطلب ضمان لمدة عام للسيارة قابل للتمديد عامين ويطلب ضمان لمدة عام مجاني للبرمجيات قابل للتمديد 5 أعوام. يبدو ذلك غريباً حيث يبدو أن احتمال تعطل سيارة أو منزل وحاجتهما لصيانة أكبر من احتمال تعطل برنامج.

من حيث **الوقت** فإن المشاريع البرمجية تحتاج لوقت أطول وتتأخر أكثر بكثير من بناء منزل أو تصنيع سيارة أو كسوة منزل ويؤثر ذلك بدوره على التقديرات المالية والنوعية ويرفع المخاطر.

3 ما هي الأسباب خلف هذه المفارقات وما هي الحلول؟

لو نظرنا لجميع هذه الصناعات (بما فيها صناعة البرمجيات) لوجدنا أنها جميعا من **الصناعات التجميعية** فهي مؤتمتة إلى حد كبير جداً من حيث المكونات الأساسية وتليها مرحلة تجميع للمكونات. فعظم شركات السيارت تشتري مكوناتها من محركات ودواليب وزجاج ومكيفات من شركات أخرى وتقوم غالباً بتصنيع الهيكل وتجميع القطع وتركيبها وأعمال الإنهاءات البسيطة والكثيرة. كما أن البناء يعتمد بشكل كبير على مكونات جاهزة مثل البلوك والاسمنت والنوافذ والأبواب والدهان بل يوجد أيضاً ما يسمى منازل مسبقة الصنع. وكذلك لدينا في البرمجيات مكونات برمجية جاهزة مثل GUI components, Enterprise components وأدوات متطورة مثل قواعد البيانات ومولدات التقارير وغيرها. وعادة فإننا عندما نتحدث عن الصناعات التجميعية فغالباً ما نتحدث عن صناعات مستقرة. **فما بال صناعة البرمجيات المؤسساتية لا تستقر على حال وما الذي يميزها عن غيرها من الصناعات؟**

لا يمكننا بأي حال من الأحوال التشكيك في المكونات، فالمكونات المعلوماتية مستقرة جداً أسوة بمكونات الصناعات الأخرى وبالتالي فإن المشكلة الرئيسية تأتي من عملية التجميع. والتي يفترض أن تكون أبسط المراحل والتي عادة ما تنقلها الشركات العالمية إلى دول العالم الثالث لرخص اليد العاملة هناك. **فما الذي يجعل من عملية تجميع المكونات في نظام برمجي مؤسساتي أصعب من تجميع المكونات في بناء منزل أو سيارة؟**

4 العوامل التي تؤثر على صعوبة التجميع

1- الحجم الفيزيائي للمكونات: كلما ازداد حجم المكون كلما ازدادت صعوبة تجميعه أو وضعه في مكانه كما يحدث في حال هياكل السيارت وحراقات التدفئة و وخزانات المياه. إلا أنه لا يوجد في المعلوماتية مكونات فيزيائية ولذلك فهذه النقطة ليست مسؤولة عن صعوبة التجميع

2- عدد المكونات: وفقاً لحجم السيارة (صغيرة أو باص أو شاحنة أو قاطرة ومقطورة) أو المنزل (شقة صغيرة أو فيلا أو قصر أو ناطحة سحاب) أو حجم البرنامج 10 حالات استعمال أو 100 أو ربما 1000. لو نظرنا لعدد مكونات سيارة أو منزل لوجدنا أنها أكثر بكثير من 100 حالة استعمال أو حتى ألف فالحجم ليس هو المسؤول عن الصعوبة.

3- الترابط بين المكونات: تتصف عملية التجميع في السيارت والأبنية بكونها **محلية** من جهة و**ستاتيكية** من جهة أخرى.

هي **محلية** لأن ربط المكونات الفيزيائية (مثل ربط مكيف بالهيكل وبالمحرك) يتم من خلال نقاط محددة وعلاقة المكون لا تكون إلا مع المكونات المجاورة له فيزيائياً وبالتالي فإن تجميع قطعة ميكانيكية مكونة من 100 مكون تحتاج لحوالي 100 عملية قد تتعداها لمئتين أو ثلاثة. أما في صناعة البرمجيات فإن برنامج مكون من 100 مكون قد يحتاج لمربع هذا العدد من عمليات التجميع أي 10000 عملية كما أنه من الممكن أن تكون العلاقة بين مجتزئين متكررة في أكثر من موضع مما يسمح برفع هذا العدد عشرات المرات. كمثال على ذلك فإن عملية شراء لأصل ثابت قد تحتاج لتكامل مع الموازنة والمحاسبة والمالية وكذلك مع المستودعات عند استلام البضاعة ومن ثم مع المحاسبة والموازنة من جديد عند صيانتها وعند شراء الأكسسوارات وكذلك مكاملة مع الأصول الثابتة عند اهتلاكها ومع إدارة النفقات عند توزيع نفقاتها على المشاريع المختلفة ومع الموارد البشرية عند تسليمها لأحدهم ومع الدوام عندما يتناوب عاملين على استعمالها والقائمة لا تنتهي.

هي **ستاتيكية** لأن عملية تكامل الأجزاء الفيزيائية والميكانيكية، إن نجحت فقد نجحت بشكل نهائي، لأن الإرتباط نهائي ولا يخضع لعامل الزمن ولا لشروط العمل أما في البرمجيات فالترابط ديناميكي. ولو عدنا لمثالنا السابق لوجدنا أن كل ما ذكرناه من ارتباط بشراء الأصول الثابتة وبقية المجتزآت لا يكفي فهنالك أيضاً العديد من الحالات الخاصة التي تتطلب مزيد من الترابط مع المالية والمستودعات والموارد البشرية مثل إلغاء عملية الشراء أو ضياع الأصل الثابت أو إتلافه.

في الواقع فإن المشكلة في الترابط بين المكونات البرمجية ليست فنية البتة وإنما وظيفية ولذلك فلا بد لمن يطور هذا المجتزأ الجديد من فهم كامل لمنطق عمل المجتزآت العمودية ليرسل المعلومات الصحيحة في اللحظة المناسبة، بل ويكون قادراً على التراجع عنها وتصحيحها، وأن يكون قادراً على مزامنتها كأن يزامن طلب الشراء مع حجز المبلغ في الموازنة، وأن يزامن استلام البضاعة من المورد مع تحقيق الدين الفعلي، وان يزامن قبول الفاتورة من المزود مع الفحص الفني للبضاعة، وأن يزامن إعطاء أمر الدفع مع نقل الاستحقاق المالي للمورد، وأن يزامن توزيع نفقات السيارة على المشاريع

مع إغلاق السنة المالية، وأن يجعل نظام الأصول الثابتة يرى هذا المنتج الجديد ذو الخصائص الجديدة على أنه أصل ثابت. في الواقع ما نحتاجه في التجميع ليس فنياً بل ليس مهندساً حتى، بل مهندس بخبرة وظيفية قادر على فهم مثل هذه الحيثيات ولذلك نرى أن أوراكل وساب تقدمان أثناء التحقيق خبراء متفرغين بأسعار غالية جداً للقيام بعملية التحقيق ولذلك نجد أن البرمجيات المؤسساتية أصبحت من الإختصاصات المتنازع عليها بين المعلوماتية وإدارة الأعمال ورأينا ظهور اختصاصات جديدة مثل MIS أو نظم إدارة المعلومات تدعي أنها تغطي الحلقة المفقودة بين المعلوماتية والإدارة.

4- تنوع المتطلبات وتغييرها: نادراً ما تتغير المتطلبات الوظيفية بعد البدء في مشروع صناعة سيارة أو بناء وإن حدث ذلك فنادراً ما يؤثر على البنيان الأساسي للمتطلبات الوظيفية بل يؤثر على متطلبات ثانوية مثل الكسوة واللون. أما في مجال البرمجيات فالموضوع مختلف تماماً لأن منطق العمل يتغير من مؤسسة لأخرى ولا ترغب المؤسسات في إعادة هندسة إجراءات العمل لديها. في الواقع فإن ثقة المستخدم بشركات السيارات كبيرة تجعله يقبل ما تقدمه له شركات السيارات دون نقاش حتى لو كانت النواقص لأسباب تصميمية كما هو الحال في علبة السرعة اليدوية التي تقبلها سائق السيارة لسنوات عن طيب خاطر مع أنه كان مضطراً لنقل السرعة يدوياً طوال الوقت من الأول إلى الثاني والثالث والرابع والخامس صعودا وهبوطاً ولو كانت برمجية وليست سيارة لما قبل المستخدم أبداً إلا بعلبة سرعة أتوماتيك منذ اليوم الأول لاختراع السيارة. في الواقع يعتقد زبون البرمجيات المؤسساتية بإنه أكثر خبرة في وضع منطق العمل من المهندس ولذلك فإنه يطلب عادة الكثير من التعديلات على الإجراءات الأفقية وبما أن خبرته في "هندسة إجراءات العمل" بسيطة فإنه سيتعثر كثيراً قبل الوصول إلى الإجراء المناسب مما سيتسبب في تأخير في الإقلاع وارتفاع في الكلف

5- صعوبات فنية: حتى من الناحية الفنية فإن عملية التعديل على البرمجيات ليست بسيطة، فمجرد إضافة حقل إلى واجهة المستخدم أو تغير نمطه سيتسبب في العودة إلى الرماز المصدر وإعادة تجميعه وترجمته واختباره وإضافته إلى قاعدة البيانات وإلى الطبقة الوسطى (المكونات المؤسساتية) وإعادة إرساء البرمجية مع الأخذ بالحسبان المحافظة على محتوى قواعد البيانات وعدم فقدان أي معلومات نتيجة التغيير في هيكلية المعطيات. يتطلب هذا العمل من 16 إلى 56 مهندس/ساعة وهي كلفة عالية جداً يصعب على الزبون تحملها مراراً وتكراراً.

6- تأمين اليد العاملة الخبيرة: عادة عندما تحتاج الصناعات الأخرى لمهندسين خبراء فإنهم يجدونهم فأي مهندس مدني خبير يمكنه متابعة مشروع بناء اعتماداً على المخططات. كما يمكن لطبيب خبير المباشرة في أي مشفى والقيام بعمله على أكمل وجه من اليوم الأول. في الواقع فإن لمخططات الأبنية معاييرها والإنسان هو الإنسان حيثما كان. أما البرمجيات فتختلف اختلافاً كبيراً فيما بينها مما يعقد عمليات استلامها وتسليمها بين المهندسين. في الواقع فإن خبرة مهندس المعلوماتية لا تسمح له باستلام مشروع من مهندس آخر بزمن أقل من شهر وغالباً ما يترك نقل مشروع أو عمل برمجي من مهندس لآخر آثراً سيئاً بشكل شبه دائم. هذا طبعاً إن كان جميع العاملين في مجال تطوير البرمجيات هم من المختصيين فهندسة البرمجيات ليست محمية بأي قوانين كما هو الحال في مجالات أخرى مثل الهندسة المدنية والميكانيكية والطب.

7- التوثيق: ربما يعتقد البعض بأن التوثيق هو حل لمشكلة تدهور خبرة الفريق مع التغييرات المتتابعة في أعضائه. ولكنه، وللأسف، في الوقت الذي يعتمد مهندسي المدني والميكانيك على هذه الوثائق والمخططات بشكل أساسي في عمليات الاستلام والتسليم فإن الوثائق في المشاريع البرمجية كثيرا ما تكون عديمة الجدوى وذلك بسبب تنوعها (متطلبات وتحليل وتصميم وخطاطات اختبار) وانفصالها سريعاً عن الواقع بسبب تكرار التغييرات في المتطلبات change requests.

8- كلفة تدوير اليد العاملة: كما ذكرنا في الفقرتين السابقتين فإن التخفيف من الأثر السلبي للتبدلات المتعاقبة في الفريق البرمجي يتطلب تقاطع زمني طويل نسبيا بين المهندس القادم والمهندس المغادر كما يتطلب توثيق دقيق ومحكم وتحديث مستمر لهذه الوثائق وهذا يتطلب زيادة في حجم الفريق وخبرات متميزة لضمان الترابط بين الوثائق من جهة وضمان آلية لاستعمالها والاستفادة منها من جهة أخرى. ولا يخفى على أحد بأن زيادة حجم الفريق سيزيد الكلفة بشكل كبير جداً لأن كلفة صناعة البرمجيات هي في 90% منها أجور يد عاملة.

9- الانتقال: إن استغناء مستخدم عن سيارة قديمة أو منزل قديم والانتقال إلى سيارة أخرى أمر يسير يتم بسهولة ومتعة ولايشبه انتقال مؤسسة من برمجيات قديمة إلى أخرى جديدة. حتى في حال وجود برمجيات مؤسساتية مختبرة ومستقرة ومجربة مثل أنظمة إدارة المشافي HMIS فإن مجرد تشغيلها في مشفى جديد يعتبر عملاً شاقاً جداً يتطلب عدة سنين ويمر بصعوبات كثيرة منها تهجير المعلومات القديمة للنظام الجديد والتدريب والتشغيل

10- ممانعة التغيير: لا نسمع بأحدهم وقد رفض أن تقوم مؤسسته باستبدال سيارته القديمة بسيارة جديدة والإنتقال من مبنى قديم إلى مبنى جديد في حين نجد العديد من الدراسات عن أسباب ممانعة التغيير resistance to change في حالة البرمجيات. فمنها ما يعود لأن الموظف اعتاد برمجيات معينة ولا يريد العودة للتعلم من جديد أو أنه يخشى منافسة الجيل الشاب له في استعمال البرمجيات الحديثة أو أنه يخشى استبداله بهذه البرمجيات وربما لفساد إداري معين.

11- كلف التشغيل والصيانة: تشكل الصيانة والتشغيل دخلاً كبيراً لصناعات السيارات والمباني وبما أن لها وجود فيزيائي فإنه من السهل أن تدفع مؤسسة أو فرد مبلغاً كبيراً على إصلاح عطل في سيارة أو في منزل لفني بسيط في حين لا تقبل دفع نفس المبلغ بنفس ساعات العمل لمهندس معلوماتية قام بتعديل أو تطوير والسبب هو غياب البعد الفيزيائي للخدمة التي قدمها.

5 ما الذي فعلته الإكسير منذ بداية تأسيسها لمعالجة هذه التحديات ؟

طورت الإكسير عام 2005 بيئة عمل لتسهيل عملية بناء وتجميع البرمجيات وتعديلها وكان العمل في هذه البيئة يتم بمرحلتين "وصف وانشر" specify and deploy وعلى الرغم من سهولة العمل على هذه البيئة إلا أن بعض التحديات مصحوبة بالأزمة السورية سارعت بعودة الإكسير للبحث والتطوير للوصول إلى بيئة جديدة تعمل بمرحلتين مختلفتين وهما "شغل ثم وصف" run then specify.

كانت دوافع الإكسير هي الوصول للحلم المشترك لمن يعمل في مجال هندسة البرمجيات وهو أن يكون الزمن اللازم لتطوير البرمجيات تابع لتعقيد المتطلبات الوظيفية فقط (محاسبة، مستودعات، تجارة إلكترونية، صيرفة، إلخ) لا للمتطلبات غير الوظيفية مثل الأمن والأداء وإدارة الموارد وتوزيع العبء والتسامح مع الأخطاء والتعافي من الأعطال وغيرها من المتطلبات غير الوظيفية التي تتطلب مهندسين مهرة بخبرة لا تقل عن خمس سنوات في نفس المجال. لنأخذ مثال عن حالة استعمال مثل "حوالة مصرفية داخلية". يمكن لطالب في السنة الاولى تنفيذ حالة الاستعمال هذه فهي لا تحتاج إلا لإنقاص رصيد الحساب الدائن ورفع قيمة الحساب المدين وتنتهي في دقائق إلا أن الخصائص غير الوظيفية المرتبطة بها تحتاج لوقت طويل مثل التنصت والتعقب والتنبيه والأمن الوظيفي وأمن المعلومات وإدارة المناقلات وربما المناقلات الموزعة وإدارة قواعد البيانات الموزعة في حال كان للمصرف عدة فروع تحتمل انقطاع الشبكة بينها وإدارة توزيع الحمل إذا كان هنالك أكثر من مخدم وطبعاً إدارة التسامح مع الأخطاء والتعافي من الأعطال إذا أردنا أن لا يتسبب خطأ ما بتوقف منظومة المصرف.

حتى بالنسبة للشركات المستعدة لدفع تكاليف عدد كبير من المهندسين المهرة فنادراً ما نستطيع (في الدول النامية) الحفاظ على المهندسين فترة طويلة وخاصة في ظل الأزمة. ولذلك فإن الوصول لبيئة عمل، تتحمل آلياً مسؤولية المتطلبات غير الوظيفية وتسمح للمهندس بالتركيز على منطق العمل فقط، سيكون من شأنه تخفيف الكلف والوقت ورفع انتاجية المهندس والتخفيف من المخاطر وهذا ما فعلته الإكسير بتطويرها لتكنولوجيا "وصف وانشر".

5.1 مرحلة وصف وانشر Specify and Deploy

وصلت بيئة عمل الإكسير، في عام 2006، في مستوى دعمها للـ MDA أو البنيان المشتق من النماذج إلى حدِ أنّ تطوير مجتزأ برمجي عالي المستوى مع إجراءات العمل المرتبطة به مثل المحاسبة والمستودعات والمهام والصرافة والصيانة والموازنة يمكن أن يقوم به شخصٌ واحد، وبمرحلتين فقط **"وصِف وانشُر"** (Specify and Deploy). حيث أنه وبمجرد توصيف المبرمج للمجتزأ من وجهة نظر وظيفية، سيقوم الإكسير بتوليد قواعد البيانات والواجهات البيانية والمكونات المؤسساتية والمتنصتات والأعمال الخلفية مع دعم كامل لأحدث نماذج التصميم مثل ORM, DAO, MVC, 3 Tiers, IoC, AOP،ودعمٍ للعنقدة clustering وتوزيع العبء load balancing وإدارة الموارد -مثل الذاكرة والمعالج- ودعمٍ لخصائصَ غير وظيفيةٍ هامةٍ، مثل scalability, auditing, tracing,.

أما بالنسبة لمكاملة إجراءات العمل مع نماذج الأغراض والمعطيات وكذلك مكاملة المجتزآت معاً فقد طورت الإكسير لغة إدارة المهام TDL وهي لغة مقروءة تسمح لأي مجتزأ بأن يطلب ما يريد من غيره حتى بدون ترجمة compilation مسبقة للمجتزئين معاً. وبالتالي فقد حققت الإكسير الأهداف التالية:

1- **سهولة في التطوير**: لا يحتاج المهندس لفهم التعقيدات الكبيرة للنظم الشبكية بل يحتاج لفهم منطق العمل
2- **سهولة في الصيانة والتبادل**: البرمجيات صغيرة الحجم ومقروءة ويمكن لأي مبرمج أن يقوم بفهم برمجيات المبرمج الآخر من خلال قراءة مخططات العمل ونماذج الصفوف والربط بينها (TDL). وبذلك فإن نقل النظام من مهندس لآخر ليس معقداً كما هو نقل برمجيات مكتوبة بجافا أو C++.
3- **سرعة كبيرة في الانتقال من التوصيف إلى البرنامج** من خلال أتمتة عمليتي التصميم والبرمجة
4- **تخفيف في الكلف**: بما أن النفقات الرئيسية لتطوير البرمجيات هي رواتب المهندسين فالسرعة تعني بالضرورة تخفيض في الكلف
5- **الاستغناء عن خبرات نادرة** مثل خبرة خمس سنوات في تصميم البرمجيات المؤسساتية المعتمدة على الوب وقواعد البيانات وتطوير واجهات الوب والمكونات المؤسساتية (EJB) وإدارة المعلومات الضخمة والأمن وتوزيع الحمل وغيرها من القضايا.
6- **سهولة التوثيق**: لا تحتاج البرمجيات المطورة باستعمال الإكسير إلى وثائق تحليل وتصميم أو ملاحظات على الرماز يكفي وجود وثيقة واحدة هي استمارة المتطلبات الصورية لأن الانتقال منها إلى البرنامج يتم بخطوة واحدة وتعديل البرنامج يتم بالتعديل على استمارة المتطلبات الوظيفية وليس على الرماز. من ناحية أخرى فإن المهندس ملزم بتحديث هذه الاستمارة لأنها هي البرنامج الفعلي الذي يقوم بتطويره ولا مهرب بالنسبة له من تحديثها مما سمح للإكسير بتجنب المشكلة القديمة في البرمجيات وهي شيخوخة الوثائق حيث يقوم المهندسين بتطوير البرمجيات دون تحديث الوثائق المرتبطة بها خاصة عندما يعملون تحت الضغط.

5.2 مرحلة شغل ثم وصف Run Then Specify

بدت الأمور مع "وصف وانشر" جيدة بل ممتازة والمكاسب-من وجهة نظر هندسة البرمجيات-كانت واضحة على الأقل في القضايا أعلاه التي تناولتها الإكسير. فالانتقال بخمس دقائق من التوصيف إلى النشر بدون تحليل أو تصميم أو برمجية يبدو رائعاً. ولكن ظهرت صعوبات من نوع آخر تفاقمت مع بداية الأزمة لأن هذه الصعوبات ترتبط بشكل مباشر أو غير مباشر مع انخفاض القدرة الشرائية لدى المستهلك من جهة وضرورة إدارة التبدل السريع للمهندسين من جهة أخرى. دفعت هذه الظروف الإكسير إلى استغلال الركود في بداية الأزمة والاستثمار من جديد بهدف الحفاظ على المهندسين الأكفاء من جهة ومعالجة هذه التحديات من جهة أخرى مما أدى إلى وصول الإكسير في نهاية المطاف إلى تكنولوجيا جديدة وهي " شغل ثم وصّف". فيما يلي قائمة بهذه الصعوبات وكيف قامت الإكسير بحلها

5.2.1 صعوبات فنية في الصيانة والتطوير وارتفاع كلف التشغيل والصيانة

على الرغم من أن المهندس يستطيع بمجرد الضغط على زر الانتقال بخمس دقائق من التوصيف الجديد إلى البرنامج النهائي بل وإعادة نشر البرنامج إلا أن هذا يبدو مجدياً في الحالات التي نطور بها حالة استعمال كبيرة نسبياً. إلا أن التعديلات الكثيرة التي قد تطرأ على النظم بعد تسليمها للمستخدم النهائي مثل إضافة حقل وحذف حقل وإعادة ترتيب الحقول وتغيير قيمة بدائية لحقل سيحتاج كل منها إلى خمس دقائق إضافة لعملية نقل النسخة للزبون أي حوالي ساعة عمل إضافية على أقل تقدير إضافة لانتظار الفرصة لإيقاف برمجيات الزبون لتحديثها. وهذا ما عبر عنه مدير عام شركة HP، عندما فشل مشروع الـERP الخاص بالشركة والتي كانت تنفذه شركة SAP، بقوله "لقد كان هنالك الكثير من القضايا كل منها بسيط لوحده إلا أنها معاً شكلت عاصفة مثالية أطاحت بالمشروع".

لحل هذه المشكلة قامت الإكسير بما يلي

1- التخلي عن تكنولوجيا "وصف وانشر" (specify and deploy) وطورت تكنولوجيا "شغل ثم وصف" (run then specify) وذلك للتخفيف من زمن وكلفة التعديلات الصغيرة وإضافة التعديلات دون إيقاف البرنامج وبدون عملية ترجمة. وبذلك استطعنا الانتقال من حوالي 16-56 مهندس/ ساعة للتعديل إلى 2-8 مهندس/ساعة للتعديل

2- أدى ذلك إلى دمج الرماز المصدر والرماز التنفيذي في مفهوم واحد هو **"التوصيف"**. في الواقع فإن برامج الإكسير أصبحت تتضمن بطريقة ما الرماز المصدر لها ووثائق المساعدة وأدوات التطوير مما يخفف من مخاطر ضياع جزء من الرماز المصدر ويغني عن عمليات إدارة الإصدارات.

5.2.2 صعوبة منطق المكاملة

إن مكاملة إجراء مثل الرواتب والأجور مع المحاسبة مثلاً يحتاج لفهم للمحاسبة وماذا يجب أن أطلب من المحاسبة لإبلاغها بالرواتب والأجور والحسميات والعمل الإضافي والتعويضات المختلفة وكيف أبلغها بحدوث خطأ ما في الحسابات السابقة أو تعديل على بعض القيم وكيف تحل مشاكل التعديل قبل وبعد الدفع الفعلي للرواتب أو حتى توطينها. مثل هذه القضايا تجعل من تطوير البرمجيات المؤسساتية صعباً جداً حتى بالنسبة لمهندس يمتلك عصاً سحرية لكتابة البرامج لأن هنالك صعوبات حقيقية في فهم ما يجب أن يقوله لهذه العصا. وهذا سيزيد من صعوبة تأهيل مهندس يضع توصيف لمثل هذه النظم. لإعطاء فكرة عن صعوبة منطق العمل فإن الشركات العالمية العاملة في مجال البرمجيات المؤسساتية تطلب مقابل تقديم خبرة رجل ليوم واحد من هذه النوع مبلغاً يتراوح بين 600 إلى 800 دولار وهو ما لا يمكن التفكير به في سورية أبداً.

لذلك قامت الإكسير بما يلي

1- **جعل الربط من مهام طرف ثالث**: تطوير آليات تجميع لا تحتاج لتدخل لا ممن كتب الإجراءات الأفقية ولا العامودية. مما يجعل كل مطور لمجتزأ يعمل وكأنه يكتب برنامجاً من الصفر ولايحتاج لمعرفة فنية ولا وظيفية بالمجتزآت الأخرى. من هذه الآليات:
   a. الدامج **Merger** والذي يستطيع القيام بدمج افتراضي لغرضين أو أكثر طورها أشخاص مختلفون وكأنهما غرض واحد ويتضمن هذا الدمج دمج افتراضي لجدوليهما في قاعدة البيانات وإن لم يكن هنالك علاقة مسبقة بين الجدولين
   b. تطبيق نموذج التصميم **IoC/AOP** على المكونات الوظيفية مثل المحاسبة والمستودعات والصرافة والموازنة وغيرها بحيث نتمكن من الاستفادة منها في الإجراءات المختلفة دون استدعائها صراحة ودون أن تستدعينا هي صراحة بل من خلال إعدادات وسيطة
   c. تطوير **ادوات تهجير وتحقيق تزايدية** يمكن إجراؤها على مراحل حتى بعد البدء بالاستعمال الفعلي للبرمجيات
   d. تطوير أدوات **لتنظيف قواعد البيانات** من المعلومات الخاطئة التي قد تدخل عليها في بدايات استعمالها وخاصة أثناء استيراد المعلومات القديمة وارتكاب أخطاء في الاستيراد واكتشاف الخطأ في مراحل متأخرة بعد استعمالها تسمح هذه الأدوات بالتراجع آلياً عن مسارات محددة من الأخطاء بدلا من العودة إلى نسخة إحتياطية قديمة من قاعدة البيانات تتسبب ليس بالتراجع عن المعلومات الخاطئة فحسب بل أيضاً عن المعلومات الصحيحة. من الأمثلة استيراد سيارة بمعلومات خاطئة من حيث تاريخ شرائها وسعرها ومن ثم قيام أحدهم بعملية استهلاك تشمل هذه السيارة ونقل معلومات المالية إلى مجتزأ المحاسبة ووقيام مجتزأ التعقب والمراقبة بتحديث سجلاته بذلك وبعد كل هذه الأعمال التي تمت خلال دقائق نكتشف بأن هنالك خطأ في معلومات هذه السيارة ونريد التراجع عنه دون العودة إلى نسخة قديمة من قواعد البيانات ودون تجنيد عدة مهندسين لتحليل قاعدة البيانات والبحث في العمليات التي تمت منذ استيراد السيارة وحذفها وضمان صحة وتكامل قاعدة البيانات بعد الحذف.
2- **نقطة مكاملة وحيدة**: جعل الإكسير من المكاملة مع المجتزآت الوظيفية عملاً سهلا ومن نقطة واحدة. وهي عبارة عن إعدادات مقروءة لا تحتاج لبرمجة بل تحتاج للإجابة على مجموعة من الأسئلة مما يسمح لمن يقوم بعملية التكامل أن يستعين بالزبون في الإجابة عنها.
3- تطوير مفهوم **نقاط المراقبة الديناميكية** والتي تسمح باستخراج تقارير دعم قرار تجميعية غنية جداً ومترابطة ومتزامنة بين المجتزآت المختلفة بصرف النظر عن هيكلية كل مجتزأ على حدة. على سبيل المثال يمكن استخراج تقارير مالية دقيقة جداً عن المبيعات وفقا للمواد والأصناف والزبائن والدول والمدن وشرائح الزبائن بصرف النظر عن بنية الشجرة المالية. هذا سيجعل التعديلات على أي مجتزأ تبقى ضمن المجتزأ ولا تتعداه لمجتزآت أخرى كما يحدث عندما نضطر لتغيير في بنية

الشجرة المالية المحاسبية لأننا نريد استخراج تقارير عن المبيعات أو المشتريات أو الموارد البشرية وما ينجم عن ذلك من صعوبات.

5.2.3 الانتقال وممانعة التغيير

قد يكون الإنتقال صعباً بسبب صعوبة تهجير المعطيات القديمة والتدرب على الأنظمة الجديدة والممانعة التي تزداد مع تأخر الانتقال إلى النظام الجديد ولذلك قامت الإكسير بما يلي:

1- طورت آليات تهجير جديدة تسمح بالبدء باستعمال النظام حتى قبل الإنتهاء من تهجير المعطيات السابقة وتهجيرها لاحقاً. سمح ذلك للمستخدمين من التمكن من المنظومة قبل تهجير معطياتهم لها والقيام بعملية التهجير بأنفسهم مما يخفض من تكاليف عملية التهجير
2- بالنسبة للتدريب فإن الإكسير اعتمدت مبدأ الأتمتة 100% أي أن المستخدم لا يقوم بإدخال إلا المعلومات الخام والموجودة على الأوراق أمامه. فلا يحتاج أي مستخدم عند إدخال فاتورة يومية اعتيادية أو عقد بيع أو شراء أو حتى مؤونة إلى فهم الشجرة المالية أو الأوامر المستودعية.
3- قامت أيضاً بتطوير online help لدعم المستخدم أثناء العمل.

5.2.4 صعوبة منطق العمل

إن إزاحة المتطلبات غير الوظيفية عن كاهل مهندس معلوماتية ليضع تركيزه على منطق العمل لا يعني أن منطق العمل سهل الفهم. فقضايا مثل المحاسبة ومراكز التكلفة والحوالات المالية والصيرفة تتضمن تفاصيل حساسة لا بد له من فهمها بدقة قبل برمجتها.

5.2.5 صعوبة تحديث التوثيق

قد يبدو أن تخفيف عدد الوثائق الداخلية إلى وثيقة هي استمارة المتطلبات سيحل المشكلة إلا أن هذه الوثيقة تتضمن حقول صورية يفهمها البرنامج وتتضمن حقول غير صورية يفهمها الإنسان وتسمح بسهولة بالتعبير عن دلالة الحقول الصورية. فمثلاً إذا ربطنا حقل الراتب بمعادلة حساب فقد يكون من الصعب فهم المعادلة من تحليلها إلى جداءات ومجاميع وربما يكون مضيعة للوقت لذلك تتضمن استمارة المتطلبات مقابل كل حقل صوري حقلاً نصياً يشرح دلالة الحقل الصوري. وبما أن بيئة العمل لا تفسر هذا الحقل فيمكن للمهندس أهماله وتجاوزه تحت ضغط العمل مما يجعل الاستمارة قابلة للفهم من قبل المهندسين فقط ويؤثر على سهولة فهمها من قبل المحللين الوظيفيين ومن قبل الزبون.

لذلك قامت الإكسير بما يلي

1- دمج وثيقة تحليل المتطلبات بالمساعدة online بحيث أن إضافة أي مساعدة online من قبل محلل النظم على النظام النهائي سينعكس آلياً على وثيقة المتطلبات ويقوم بتحديثها وبالتالي فلا يوجد حاجة للتنسيق بين محلل النظم الذي يحدث الجانب الدلالي من الوثيقة والمهندس الذي يحدث الجانب الفني منها
2- سمحت الإكسير بتطوير وثائق المتطلبات بشكل تفاعلي أثناء استخدام النظام ودون الحاجة للعودة للوثيقبة الأصلية ويمكن في أي لحظة تصدير هذه الوثيقة التي ستتضمن البعد الفني والدلالي بآن واحد
3- إيجاد **طباق صريح بين واجهات الإستعمال والتوصيف**. في الواقع فإن المهندس الذي يجيد استعمال برمجية ما من الإكسير فهو بالضرورة يجيد برمجتها وتعديلها أيضاً لأنه يستطيع معرفة العلاقة بين الرماز والبرنامج بدقة 100% والسبب هو استعمال مكونات تحليل وتصميم جاهزة أثناء بناء هذه البرمجيات باستعمال بيئة عمل الإكسير. وقد خفف هذا من مخاطر

تعاقب مهندسين مختلفين على البرمجيات أثناء فترة حياتها الطويلة نسبياً أسوة ببقية الصناعات الهندسية مثل صناعة السيارت مثلاً.

5.3 ماذا بعد شغل ثم وصف؟

1- تسعى الإكسير لجعل وثيقتها الوحيدة وهي استمارة التوصيف قابلة للقراءة من قبل المستفيد النهائي وذلك من خلال مترجمات تحول إجراءات العمل من صيغة استمارة ومراحل إلى لغة محكية. وسيسمح هذا بطريقة أو بأخرى بترجمة وثائق المتطلبات ببساطة شديدة. كما سيسمح للمهندسين الجدد بفهم البرمجيات القديمة بدون الاعتماد على المهندسين القدماء.
2- إظهار آلي لإجراءات العمل للزبون بطرق بيانية تساعده على التعلم السريع للتخفيف من زمن التدريب وخاصة للإجراءات التي تستعمل نادراً بحيث غالباً ما ينسى المستخدم ما تعلمه قبل أن يضطر لاستعماله
3- تطوير محررات تفاعلية للغة توصيف المهام TDL تسمح للزبون بتوصيف وتعديل إجراءاته بنفسه

عندما تتحقق هذه الأهداف الثلاثة فقد نستطيع بيع البرمجيات المؤسساتية في المحال التجارية، ولن يكون الزبون بحاجة إلا لاستيراد معطياته من البرمجيات القديمة والتي ستبقى لفترة أطول من أهم الصعوبات