Software Evolution and Maintenance

د.محمد الأحمد

Software Evolution and Maintenance

Debugging process :

إجرائية التصحيح, تابعة لمرحلة الاختبار ،عند وجود خطأ يجب مكان الخطأ ونوعه وأولوياته وشدته.

عند وجود خطأ لا نقوم بمعالجته مباشرة وإنما ندرس شدته وخطورته فيمكن أن نؤجل معالجته حسب الأولوية الخاصة به وشدته.

هناك فرق بين الأولوية والشدة:

الشدة هي مقدار تأثير الخطأ على النظام بشكل عام فيمكن أن يكون موجود في موقع حساس يؤثر على الكثير من التوابع والأنظمة الجزئية أما الأولوية هي نسبة تأثيره على المستخدمين وعددهم.

عندما نقوم بمعالجة الخطأ تنتهي العملية ب defect close وكل defect يحتاج لاختبار العودية للتأكد من عدم إضافة أخطاء جديدة أثناء معالجة الأخطاء القديمة.

Evolution Versus Maintenance

الصيانة لها علاقة بالإصلاح (يوجد أخطاء أو قد تقع أخطاء) أما التطور يكون على التغيير في المتطلبات مثل إضافة خدمات جديدة.

تحسين الأداء يعتبر صيانة وقائية.

الصيانة مجموعة أنشطة مخططة وهي إجرائية تكرارية تزايدية غالباً تتم بشكل حلزوني أما التطوير يمكن أن يأخذ فترة زمنية ويحدث في أي وقت.

قوانين Lehman:

لها علاقة بالتغيير وهي ملاحظات واقعية وهي :

1. التغيير مستمر: التغيير هو الثابت الوحيد في الحياة ويجب تقبله.
2. زيادة التعقيد: كلما زادت الخدمات في النظام كلما أصبح أكثر تعقيداً.

ملاحظة: التعقيد هو أحد واصفات البرمجية وهي من المتطلبات غير الوظيفية وتعبر عن مدى سهولة فهم البرمجية وإعادة شرحها (فهم + شرح) يتم قياسها من خلال Software matrix وعبر cc وهي عدد المسارات المغلقة ويتناسب طرداً مع التعقيد.

3. Self -regulation : عملية تطوير أي نظام لها علاقة بمكان نشره أي المؤسسة التي تم نشره فيها وبعضها لا يتطور أما بعض المؤسسات نتيجة المنافسة بحاجة للتطوير وزيادة الخدمات.
4. Conservation of organizational : لا يوجد نظام Stability مستقر بشكل مطلق.

النظم البرمجية نظم مفتوحة.

ملاحظة: النظم المغلقة نادرة الوجود وهي التي لا تتبادل مع الوسط الخارجي أما النظم المفتوحة تتبادل مع الوسط الخارجي والنظم البرمجية هي نظم مفتوحة، مثال على النظم المغلقة الساعات الذاتية.

نظم المعلومات نظم مفتوحة تتبادل مع الوسط لا يوجد ثبات بشكل مطلق دائماً يوجد تغيير لكن درجة التغيير ثابتة ،يوجد استقرار في عملية إضافة الخدمات.

5. Conversation of familiarity : الحفاظ على التشابك مقدار المحتوى في كل إصدار جديد إما يزداد بشكل ثابت أو ينقص بشكل ثابت.
6. Continuing growth : النمو بشكل متزايد.
7. Declining Quality : يجب الحفاظ على الجودة عن طريق الصيانة بشكل دائم.
8. Feed-back System : معرفة رأي المستخدمين حول النظام لتحسين الأداء وهذا يجعل النظام يستمر وينمو.

Software Maintenance

عملية إصلاح الأخطاء ويمكن أن يكون وقائي.

يوجد 3 أنواع أساسية :

1. Corrective : إصلاحي عندما نجد أخطاء نصلحها وهذا يتطلب Regressing testing للتأكد من عدم إضافة أخطاء جديدة أثناء إصلاح الأخطاء القديمة.
2. Adaptive : عندما نريد نقل النظام من بيئة لأخرى.
3. Perfective : نجعل النظام مثالي عن طريق الصيانة الوقائية كل فترة وخصوصاً عبر refactoring.

Activities to make corrections

عندما يصلنا طلب بوجود عطل أو النظام بطيء.. يدرس هذا الطلب وتحدد المشكلة بالضبط وهذا يعود لقابلية الصيانة (مدى سهولة إيجاد الأخطاء وإصلاحها) ويعود ذلك للتصميم (low cohesion high coupling).

Evidence-based Classification of 12 different types of software maintenance

- **Training** : عندما يكون نفس فريق التطوير هو المسؤول عن الصيانة لا تأخذ وقت أو جهد أما عند وجود فريق مختلف نحتاج وقت وجهد.

- Evaluate
- Consultative
- Reformative
- Updated
- Groomative
- Performance
- Preventive
- Reductive
- Adaptive
- Corrective
- Enhancive

COTS

Components of the Shapes

نظم فرعية بديلة في النظام عندما نقوم بالصيانة ويتوافر بدائل للمكونات هذا يساعد في الصيانة.

Software Evolution Models and Processes

هي عملية تكرارية كل خدمة مضافة يجب أن تمر بعدة مراحل (تحليل، تصميم، تكامل مع أجزاء النظام، تكويد، اختبار) وهكذا.

عملية التطوير بحد ذاتها هي **process Model** لها دورة حياة كاملة وهي عملية تكرارية (iterative) لأننا نعيد النظر في كل أجزاء النظام القديمة قبل إضافة أجزاء جديدة.

RUP : (Rational Unified Process)

هي إجرائية تكرارية تزايدية تستخدم في تطوير النظم البرمجية **Plan-driven** وأحدث إجرائية برمجية تستخدم في الشركات تحوي نوعين من التكرارات تكرارات صغيرة على مستوى المراحل (البدء، الإعداء، البناء، النقل) ويتميز بالجودة العالمية والسرعة في الإنجاز، السرعة بسبب وجود فرق تعمل على التوازي والجودة بسبب وجود التكرارين.

Software Maintenance Standards

يوجد مجموعة فرضيات تحددها IEEE وISO نبدأ بـproblem identification أي تحديد المشكلة والأجزاء التي نريد تطويرها ثم

- Analysis
- deign
- Implementation
- System test
- Acceptance test and Delivery: عملية المكاملة مع النظام الحالي

أي Model كامل يتم خلال التطوير.

Software Configuration management

هو أحد Umbrella Activities فريق التطوير هو جزء من فريق العمل تهتم بشكل أساسي ب :

- Change management : إدارة التغييرات.
- Release and version management : إدارة الإصدارات والنسخ

ما يسلم للزبون إصدارات (release)

ما يبقى ضمن الشركة من تطويرات (version) تحصل على أجزاء النظام.

Refactoring

تقنية أساسية في مرحلة التحليل والتصميم والتكويد والصيانة تعتبر أحد أهم تقنيات reengineering لا تغير في الوظائف فقط تحسين البيئة الداخلية وهذا ليوفر صيانة وقائية.

Reengineering

هي إجرائية (process) تشمل :

Reverse engine : العودة إلى الخلف من الكود حتى Model لدراسة المشاكل.

Forward engineering : التقدم إلى الأمام مما يوفر رؤية حول أسباب حدوث المشاكل

$$Reenginerring = Reverse\ engineering + \Delta + Forward\ engineering$$

Legacy System

نظام قديم غير قابل للاستبدال أو التغيير نريد التطوير عليه طالما يمكن أن نقوم بصيانته لكن بعد فترة يصبح غير قابل للتطوير ويجب استبداله ولكن هذا يأخذ فترة زمنية طويلة.

يوجد مجموعة قرارات يمكن اتخاذها :

1. التجميد : توقف العمل بهذا النظام .
2. Out source : معظم هذه الأنظمة تم تطويرها داخل الشركة بما يتناسب مع متطلباتها عندها نركز على الخدمات الأساسية وجميع الخدمات الإضافية بمكن الحصول عليها من مصدر خارجي.
3. متابعة الصيانة.

4. **Discard and redevelop** : التخلص منه وتطوير نظام جديد.

5. **Wrap** : تقنية مهمة (التغليف) إضافة طبقة برمجية توفر **API** للتواصل بين نظام جزئي قديم ونظام جديد.

ملاحظة : **Application Programming Interface : API**

6. **Migrate** : الترحيل أو التهجير يوجد لدينا داتا مهمة نريد الحفاظ على بنيتها الأساسية بما يتناسب مع بيئة النظام الجديد.

Impact Analysis

تحليل التأثير : أي طلب في التغيير لا نأخذه بعين الاعتبار بل ندرسه وندرس تأثيره على النظام ونعطيه أولوياته ويمكن أن ينهار النظام عند إضافة بعض الخدمات ثم نأخذ القرار بتنفيذه أو لا حسب استقلاليته وتأثيره على بقية أجزاء النظام.

Software Reuse

يوجد عدة مستويات لإعادة الاستخدام أعلاها اعادة استخدام المعرفة يأتي بعدها إعادة استخدام الكود (system – component – object)

أهميتها : السرعة في عملية التطوير ،جودة عالية ،كلفة أقل لكن مع مراعاة وجود قيود في التطوير من حيث استقلالية لنكون ومرونته لكي يمكننا إعادة تخصيصه.

انتهت المحاضرة