٨٥

تطوير برمجيات الأجهزة الطبية
Medical Device Software Development

Richard C. Fries
Manager, Support Engineering, Datex-Ohmeda, Inc., Madison, WI

Andre E. Bloesch
Datex-Ohmeda, Inc., Madison, WI

تتحول أنواع كثيرة من الأجهزة الطبية بشكلٍ سريع لتُصبح كثيفة البرمجيات. تتحكم البرمجيات بعمل هذه الأجهزة كما تقوم بجمع وتحليل المعلومات للمساعدة في اتخاذ قرارات العلاج كما توفر وسيلة لتفاعل المستخدمين مع الجهاز الطبي. تقوم البرمجيات في هذه الأجهزة بتحويل الكمبيوتر عام الهدف إلى عنصر مُخصص الهدف في الجهاز الطبي. كما هو الحال في تصميم البيئة الصلبة في الكمبيوتر، فإن تحديد متطلبات البرمجيات ووضع تصميم برمجيات سليم وتنفيذها بشكلٍ صحيح هي تحديات فكرية صعبة. يعتمد التطوير الجيد للبرمجيات على مزيج من الإبداع والانضباط. يوفر الإبداع حلولاً للعقبات الفنية الجديدة والتحديات الجديدة في السوق واحتياجات المستخدمين. يوفر الانضباط النوعية والوثوقية للمنتج النهائي.

إن تصميم وتنفيذ البرمجيات هو عملية متعددة المراحل يتم فيها ترجمة متطلبات النظام والبرمجيات إلى برنامج وظيفي يتناول كل المتطلبات. يبدأ تصميم البرمجيات بنتاج عمل مواصفات مُتطلبات البرمجيات (Software Requirements Specification). يُعتبر التصميم في حد ذاته بنية النظام التي تتناول كل واحدة من متطلبات المواصفات وأي معايير أو ضوابط برمجية مناسبة. يبدأ التصميم في تحليل خيارات التصميم للبرامج والمفاضلة فيما بينها. ومن ثم يتم إنشاء البنية العامة للبرنامج جنباً إلى جنب مع منهجية التصميم التي سوف تُستخدم ولغة البرمجة التي سوف تُنفذ. يُجرى تحليل المخاطر ومن ثم يُصقل لضمان ألا يُسبب أي خلل في أي عنصر من البرنامج ضرراً للمريض أو المستخدم أو النظام. توضع المقاييس للتأكد من فعالية وموثوقية البرنامج. تتم مراجعة مصفوفة تتبع المتطلبات لضمان التطرق إلى جميع المتطلبات. ثم تتم مراجعة تصميم البرنامج من قِبَل الأقران للتأكد من اكتماله.

يستمر التصميم بتجزئة بنية البرامج إلى وحدات وإسناد وظائف محددة لكل وحدة وضمان التوضيح التام لمعالم واجهات الترابط الداخلية والخارجية. يتم اختيار أسلوب وتقنيات الترميز على أساس قيمتها المُبرهنة والوظيفة المقصودة من النظام وبيئة النظام. تضمن مراجعات الأقران اكتمال وفعالية التصميم. كما يضع التصميم التفصيلي الأساس لفعالية التحقق والتثبت اللاحقة. يُعتبر استخدام الأدوات الآلية خلال برنامج التطوير وسيلة فعالة لتبسيط عملية التصميم والتطوير وتساعد في تطوير الوثائق اللازمة.

يُعتبر التخطيط مفتاح النجاح في عملية التحقق والتثبت. يُغطي تخطيط هذه العملية كامل دورة حياة التطوير بدءاً من توليد المتطلبات وحتى إصدار المنتَج. يوثّق التخطيط الأولي لعملية التحقق والتثبت في خطة خطتها للبرنامج (Software Verification and Validation Plan - SVVP). يصف الـ SVVP دورة حياة عملية التحقق والتثبت كما يعطي لمحة عامة عنها ويصف فعاليات دورة حياتها ويعرِّف توثيقها ويناقش الإجراءات الإدارية لها. ويمكن الحصول على توجيهات ممتازة عن تخطيط التحقق والتثبت في IEEE Std 1012 كما يمكن الحصول على تفسير لـ Std 1012.في IEEE Std 1059.

معايير وضوابط البرمجيات

Software Standards and Regulations

هناك عدد لا يحصى من معايير البرمجيات لمساعدة المطورين في تصميم وتوثيق البرنامج. تُغطي معايير الـ IEEE عملية التوثيق خلال جميع مراحل التصميم. تصف المعايير العسكرية طريقة تصميم وتطوير البرنامج للاستخدام العسكري. كما أن هناك أيضاً معايير بخصوص جودة وموثوقية البرمجيات لمساعدة المطورين في إعداد البرنامج عالي الجودة. أنتج المجتمع الدولي معايير تتناول في المقام الأول سلامة البرمجيات. يُعتبر المعيار في كل حالة من هذه الحالات بمثابة وثيقة طوعية وضعت لتوفير مبادئ توجيهية لتصميم وتطوير واختبار وتوثيق البرمجيات.

تُعتبر الـ FDA في الولايات المتحدة مسئولة عن ضمان أن يكون الجهاز الذي يستخدم البرمجيات (أو البرنامج بحد ذاته كجهاز) آمن وفعال لتحقيق الغرض المقصود منه. أنتجت الـ FDA عدة مشاريع لمبادئ توجيهية للمراجعين ولمبادئ توجيهية للفاحصين ولسياسات البرمجيات ولضوابط الممارسة الجيدة في التصنيع (GMP) تُعالج برمجيات الأجهزة والعمليات. بالإضافة إلى ذلك، فقد تم إعداد مبادئ توجيهية لمُراجعي الـ FDA وبرامج تدريبية للمفتشين والمراجعين. يتطرق الإصدار الجديد من تنظيم الـ GMP إلى البرمجيات كجزء من مرحلة التصميم.

تتصدر الولايات المتحدة غيرها من البلدان في وضع مبادئ توجيهية لتطوير البرمجيات الطبية. ولكن، هناك حركة في العديد من المنظمات الدولية لوضع لوائح ومبادئ توجيهية للبرمجيات والأجهزة التي يتم التحكم بها عن طريق البرمجيات. على سبيل المثال، يتناول ISO 9000-3 على وجه التحديد تطوير البرمجيات بالإضافة إلى ما هو وارد

في ISO 9001. تُعالج الـ CSA قضايا البرمجيات في أربعة معايير تغطي البرامج الجديدة والبرامج التي سبق وضعها في التطبيقات الهامة وغير الهامة. إن لدى الـ IEC وثيقة تطوير برمجيات يتم تطويرها حالياً، إضافة إلى وثيقة لتحليل المخاطر (IEC 601-1-4).

بدائل التصميم والمفاضلة فيما بينها
Design Alternatives and Tradeoffs

إن العزم على تصميم وتخصيص المُتطلبات هو عملية تكرارية للغاية. يُفترض أن تتمكن التصاميم البديلة من تلبية المتطلبات (أو أن تكون مُرشحة لتلبيتها). إن البت في هذه التصاميم هو نشاط إبداعي بشكلٍ أساسي وهي عملية "شذب وتجريب (cut and try)" لتحديد التصميم الذي يمكن أن يقوم بالعمل. تتعدد التقنيات المحددة والمُستخدمة وتتطلب مجالاً واسعاً من المهارات. تشتمل المهارات على نظرية التحكم وعملية التحسين واهتمام بارتباط الإنسان مع الآلة واستخدام التجهيزات الحديثة لاختبارات التحكم ونظرية الاصطفاف (queuing theory) وهندسة الاتصالات والكمبيوتر والإحصاء وغيرها من التخصصات. يتم تطبيق هذه التقنيات على عوامل مثل الأداء والوثوقية والجدول الزمني والتكلفة وقابلية الصيانة واستهلاك الطاقة والوزن وتوقع العمر.

سوف يتم التخلص من بعض التصميمات البديلة بشكلٍ سريع، في حين تتطلب التصميمات البديلة الأخرى تحليلاً أكثر حذراً. يتم تقييم قدرات ونوعية كل تصميم بديل باستخدام مجموعة من عوامل التصميم الخاصة بكل تطبيق وبأساليب تمثل تصميم النظام.

سوف تكون بعض التصاميم البديلة متفوقة في بعض النواحي، في حين تكون تصاميم بديلة أخرى متفوقة في نواح أخرى. تتم المفاضلة (trade off) بين هذه البدائل على أساس العوامل التي تُعتبر مهمة بالنسبة للنظام الذي يتم تصميمه. ينشأ التصميم من سلسلة من القرارات التكنولوجية التي يتم توثيقها بمخططات بُنية تجمع بين جوانب البيانات وانسياب التحكم. باعتبار الوظيفية عنصراً تكرارياً في اتخاذ القرارات التكنولوجية وأنها تتجسد بانسياب البيانات ومخططات انسياب التحكم من تحليل مُتطلبات النظام، فإنه يتم تخصيصها إلى مكونات مختلفة في النظام. على الرغم من أن أساليب اختيار عناصر معينة في التكنولوجيا لا تُعتبر جزءاً من المنهجية، فإنه يتم توثيق الآثار المترتبة على القرارات في مخططات مُتطلبات وتوقيت الأداء الداخلي.

وأخيراً، يتم أخذ جميع العوامل بعين الاعتبار بما في ذلك رغبات العملاء والمواضيع السياسية لإنشاء التصميم الكامل للنظام. يُسمى نتاج تصميم نظام بـ "نموذج البنية". تشمل البنية مكونات النظام وتخصيص المُتطلبات ومواضيع مثل الصيانة والموثوقية والتكرار والاختبار ذاتي.

بُنية البرمجيات
Software Architecture

إن بُنية البرمجيات هي جزء رفيع المستوى من تصميم البرمجيات وهي الإطار الذي يحتوي على أجزاء التصميم الأكثر تفصيلاً. توضَّح البُنية عادة في وثيقة واحدة تسمى "مواصفات البُنية". يجب أن تكون البُنية شرطاً مسبقاً للتصميم المفصل وذلك لأن جودة البنية تُحدد سلامة مفاهيم النظام. وهذا بدوره يحدد الجودة النهائية للنظام.

تحتاج بنية النظام أولاً إلى نظرة عامة تصف النظام في خطوطه العريضة. كما ينبغي أن تتضمن أدلة على أنه قد تم النظر في البدائل للتنظيم النهائي والأسباب التي دعت إلى اختيار التنظيم المستخدم من بين هذه البدائل. ينبغي أن تتضمن البُنية أيضاً على التالي:

- تعريف بالوحدات الرئيسية في البرنامج. يجب توضيح العمل الذي تقوم به كل وحدة بشكل جيد فضلاً عن واجهة ترابط كل وحدة.
- وصف للملفات الرئيسية والجداول وهياكل البيانات التي سوف تُستخدم. يجب أن تصف البدائل التي تم النظر فيها وينبغي تبرير الاختيارات التي تمت.
- وصف لخوارزميات معينة أو الإشارة إليها.
- وصف للخوارزميات البديلة التي تم النظر فيها وتوضيح أسباب اختيار خوارزميات معينة.
- مواصفات الأهداف الرئيسية المطلوب تحقيقها في حال النظام الموجه لأهداف معينة. ويجب أن تُحدِّد البُنية مسؤوليات كل هدف من الأهداف الرئيسية والطرق التي يمكن بموجبها للهدف أن يتفاعل مع الأهداف الأخرى. يجب أن تتضمن وصفاً لهرميات الفئات وتحولات الحالة واستمرار الهدف. كما ينبغي أن تصف الأهداف الأخرى التي تم النظر فيها ويجب أن تعطي أسباب تفضيل التنظيم الذي تم اختياره.
- وصف لإستراتيجية التعامل مع التغيرات بوضوح. ينبغي أن تبين أنه قد تم النظر في التحسينات الممكنة وأن التحسينات المُرجحة هي أيضاً الأسهل للتنفيذ.
- تقدير لمقدار الذاكرة المستخدمة في الحالات الاسمية والقصوى.

تهتم بُنية البرمجيات بائنين من الخصائص الهامة لبرنامج الكمبيوتر: الهيكل الهرمي للمكونات الإجرائية (الوحدات) وهيكل البيانات. يتم استخلاص بُنية البرمجيات من خلال عملية التقسيم التي تربط عناصر الحل البرمجي إلى أجزاء من المشكلة الحقيقية التي تم تعريفها ضمناً خلال تحليل المُتطلبات. يبدأ تطور هيكل البرنامج مع تعريف المشكلة. يظهر الحل عندما يتم حل كل جزء من المشكلة من قِبَل عنصر واحد أو أكثر من عناصر البرنامج.

اختيار المنهجية
Choosing a Methodology

يبدو أن عدد منهجيات التصميم يساوي تقريباً عدد المهندسين الذين ينفذونها. عادة ما يستلزم اختيار المنهجية وصفاً لتحليل المُتطلبات وإجراءات التصميم. إن لكل طريقة من الطرق العديدة المعروفة ميزاتها الخاصة وذلك اعتماداً على الاستخدام الذي تُطبق فيه هذه الطرق. يجب أن تسير عملية تعيين الأدوات وعملية اختيار المنهجية جنباً إلى جنب. وينبغي اقتناء الأدوات لدعم منهجية التصميم المعروفة أو المؤقتة ولدعم تنفيذ الخطط. يتم في بعض الحالات شراء الأدوات لدعم المنهجية المستخدمة بالفعل. في حالات أخرى، تُملي مجموعة الأدوات المتاحة تعيين المنهجية. من الناحية المثالية، يتم اختيار الاثنين في نفس الوقت بعد إجراء تقييم شامل للحاجة.

يجب ألا يعتمد الاختيار الصحيح لمجموعة الأدوات ومنهجية التصميم على الإعلانات المبهرجة أو على اقتراح من قِبَل مُرشد منهجية ذا سلطة. من المهم فهم البيئة التي سيتم توظيف هذه المنهجية فيها والمنتج الذي سوف تُطبق فيه. يجب أن يستند القرار إلى معايير أخرى من بينها: حجم المشروع (أي عدد المُتطلبات) ونوع المُتطلبات (أي، زمن حقيقي برمجي أو زمن حقيقي في البيئة الصلبة "hard or soft real-time") ومدى تعقيد المنتج النهائي وعدد المهندسين ومستوى خبرتهم ومهارتهم والجداول الزمنية للمشروع وميزانيته ومتطلبات الموثوقية والتحسينات المستقبلية على المنتج (أي، مخاوف الصيانة). ينبغي تطبيق عوامل التثقيل على معايير التقييم. بطريقة أو بأخرى وسواء كان التقييم يجري بطريقة رسمية أو غير رسمية وسواء كان ينطوي على شخص واحد أو أكثر، ينبغي أن يتم التقييم لضمان الملائمة المناسبة للمُنظمة والمنتَج.

إن أهم عامل يجب اعتباره من أجل التنفيذ الناجح لمنهجية التصميم (بغض النظر عن الطريقة المستخدمة) هو اقتناع فريق تطوير البرمجيات. يجب أن تتوفر لدى فريق تطوير البرمجيات الثقة في أن هذا النهج هو مناسب للتطبيق ويجب أن يكون هذا الفريق مستعد و"متحمس" للخوض في المشروع. يتطلب تنفيذ منهجية التصميم انضباطاً لا هوادة فيه. هناك العديد من المشاريع التي باءت بالفشل نتيجة لعدم الالتزام والوفاء.

إن الطريقتين الرسميتين الأكثر معرفةً والمُطبقتين على تصميم المنتجات الطبية هما طريقة التحليل/التصميم الموجه لأهداف معينة (Object Oriented Analysis/Design) وطريقة التحليل/التصميم المُركّب (Structured Analysis/Design) "من الأعلى إلى الأسفل" التقليدية. هناك مزايا وعيوب لكل طريقة من هاتين الطريقتين. إذا تم العمل بأي طريقة من هاتين الطريقتين بأسلوب منظم ومنهجي جنباً إلى جنب مع تصميم النظام الكهربائي، فإن ذلك يمكن أن يوفر منتجاً آمناً وفعالاً.

اختيار اللغة

Choosing a Language

إن لغات البرمجة هي آليات التدوين التي يتم استخدامها لتنفيذ منتجات البرمجيات. تُمارس الميزات المتوفرة في لغة التنفيذ تأثيراً قوياً على الهيكل البنيوي وتفاصيل الخوارزمية للبرنامج. كما ثُبت أن اختيار اللغة يؤثر أيضاً على إنتاجية المبرمج. أظهرت بيانات الصناعة أن فاعلية المبرمجين عند استخدام اللغة المألوفة تكون أكثر من فعاليتهم عند استخدام اللغة غير مألوفة. يُحقق المبرمجين الذين يعملون مع اللغات عالية المستوى إنتاجية أفضل مما يفعله أولئك الذين يعملون مع اللغات منخفضة المستوى. يميل المطورون الذين يعملون مع اللغات المُفسَّرة (interpreted languages) إلى أن يكونوا أكثر إنتاجيةً من أولئك الذين يعملون مع اللغات المُجمَّعة (compiled languages). يمكن في اللغات التي تتوفر فيها كل من الصيغة المُفسرة والصيغة المُجمعة أن يتم تطوير البرامج بفعالية في الصيغة المُفسرة ومن ثم إطلاق هذا البرنامج بالصيغة المُجمعة الأفضل أداءً.

توفر لغات البرمجة الحديثة مجموعة متنوعة من الميزات لدعم تطوير وصيانة منتجات البرمجيات. تشمل هذه الميزات ما يلي:

- التحقق القوي من النوع.
- تجميع منفصل.
- أنواع بيانات مُحددة من قِبَل المُستخدم (User-defined data types).
- تغليف البيانات.
- تجريد البيانات.

إن القضية الرئيسية في التحقق من النوع هي المرونة مقابل الأمن. توفِّر اللغات قوية النوع الحد الأقصى من الأمن في حين توفر اللغات ذات النوع التحويلي الضمني الآلي (automatic type coercion) أقصى قدر من المرونة. إن الاتجاه الحديث هو نحو زيادة التحقق القوي من النوع بالميزات التي تزيد المرونة مع الحفاظ على أمن التحقق القوي من النوع.

يتيح التجميع المنفصل إمكانية الإبقاء على وحدات البرنامج في المكتبة. يتم ربط الوحدات إلى نظام البرنامج حسب الاقتضاء عن طريق مُحمِّل الربط. إن الفرق بين التجميع المستقل والتصنيف المنفصل هو أنه يتم إجراء التحقق من النوع عبر واجهات ربط وحدات التجميع بواسطة خدمة تجميع منفصلة وليس من قبل خدمة تجميع مستقلة. تسمح أنواع البيانات المُحدَّدة من قِبَل المُستخدم (بالاقتران مع التحقق القوي من النوع) للمُبرمج بنمذجة الكيانات وعزلها عن المجال الصعب باستخدام نوع بيانات مختلف لكل نوع من أنواع الكيانات الصعبة. يُعرِّف تغليف البيانات كيانات بيانات مُركبة من حيث العمليات التي يمكن أن تُجرى عليها، كما يتم تخميد تفاصيل تمثيل

البيانات ومعالجة البيانات بواسطة الآليات. يختلف تغليف البيانات عن أنواع البيانات المُجردة (abstract data) في أن التغليف يوفر حالة واحدة فقط للكيان.

يوفر تجريد البيانات (Data abstraction) آلية قوية لكتابة البرامج جيدة التنظيم وسهلة التعديل. يمكن تغيير التفاصيل الداخلية لتمثيل البيانات ومعالجة البيانات حسب الرغبة شريطة عدم تغيير واجهات ترابط إجراءات التعديل. سوف لن تتأثر المكونات الأخرى للبرنامج بهذا التغيير ربما باستثناء التغييرات في خصائص الأداء وحدود المقدرة. يمكن باستخدام خدمة تجريد البيانات تعريف كيانات البيانات بدلالة الأنواع المحددة مسبقاً والأنواع المحددة من قِبَل المُستخدم وتجريدات أخرى للبيانات مما يتيح التطوير المنهجي للتجريد الهرمي.

تحليل مخاطر البرمجيات
Software Risk Analysis

تُحدِّد تقنيات تحليل مخاطر البرمجيات أخطار البرمجيات ومتواليات الفشل المفردة والمتعددة الحرجة بالنسبة إلى السلامة، كما أنها تُحدِّد متطلبات السلامة للبرمجيات بما في ذلك متطلبات التوقيت كما تقوم بتحليل وقياس البرمجيات من أجل السلامة. في حين تُركز المتطلبات الوظيفية في كثير من الأحيان على ما يجب أن يقوم به النظام، فيجب على مُتطلبات الخطر أن تشمل أيضاً ما لا يجوز للنظام القيام به، بما في ذلك وسائل القضاء مع أخطار النظام وضبطها والحد من الأضرار في حالة وقوع حادث مؤسف. يُعتبر تحديد الطرق التي يمكن بموجبها للبرامج والنظام أن يفشلا بأمان وتحديد المدى المقبول للفشل من الأجزاء المهمة لمتطلبات المخاطر.

وقد تم اقتراح واستخدام العديد من التقنيات للقيام بتحليل المخاطر، وتشمل:

- تحليل أخطار البرمجيات (Software hazard analysis).
- تحليل شجرة الفشل للبرمجيات (Software fault tree analysis).
- منطق الوقت الحقيقي (Real time logic).

إن تحليل أخطار البرمجيات (مثل تحليل أخطار البيئة الصلبة) هي العملية التي يتم فيها تحديد الأخطار وتصنيفها فيما يتعلق بالحرجية والاحتمال. تشتمل المخاطر المحتملة التي يجب النظر فيها على أنماط التشغيل العادية وأنماط الصيانة وفشل النظام أو الحوادث غير العادية في البيئة والأخطاء في الأداء البشري. ما إن تُحدَّد الأخطار حتى يتم تعيين الشدة والاحتمال لكل واحدة منها. تنطوي الشدة على القياس النوعي لأسوأ حادث معقول يمكن أن ينجم عن الأخطار. يشير الاحتمال إلى تكرار حدوث الخطر. ما إن يتم تحديد الاحتمال والشدة لخطر ما، حتى يتم إنشاء نمط الضبط (أي وسائل الحد من احتمال و/أو شدة الخطر المحتمل المترافق). وأخيراً، يتم اختيار أسلوب أو أساليب الضبط لتحقيق نمط الضبط المرتبط بهذا الخطر.

إن منطق الوقت الحقيقي هي العملية التي يُحدِّد بواسطتها مصمم النظام منذ البدء نموذجاً للنظام من حيث الأحداث والإجراءات. يصف نموذج الأحداث/الإجراءات تبعية البيانات والترتيب الزمني للإجراءات الحسابية التي يجب اتخاذها رداً على الأحداث في تطبيق الزمن الحقيقي. يمكن أن يُترجم هذا النموذج إلى صيغ منطق الزمن الحقيقي. يتم تحويل الصيغ إلى مسندات Presburger الحسابية مع وظائف صحيحة غير مترجمة. ثم يتم استخدام إجراءات القرار لتحديد ما إذا كان تأكيد خطر مُعيَّن هو نظرية قابلة للاشتقاق من مواصفات النظام. إذا كان الأمر كذلك، فإن النظام آمن فيما يتعلق بسلوك التوقيت المُشار إليه بهذا التأكيد ما دام التنفيذ يستوفي مواصفات المتطلبات. إذا كان تأكيد الخطر غير قابل للاستيفاء فيما يتعلق بالمواصفات، عندئذ فإن النظام هو بطبيعته غير آمن لأن التنفيذ الناجح للمتطلبات سيؤدي إلى انتهاك تأكيد الخطر. أخيراً، إذا كان إنكار تأكيد الخطر قابل للاستيفاء في ظل ظروف معينة، عندئذ يجب فرض قيود إضافية على النظام لضمان سلامته.

مقاييس البرمجيات
Software Metrics

يجب أن تخضع البرمجيات للقياس من أجل تحقيق مؤشر حقيقي للجودة والموثوقية. يجب أن تكون سمات الجودة على صلة بمتطلبات محدَّدة للمُنتَج ويجب أن تكون قابلة للقياس الكمي. يتم إنجاز هذه الأهداف من خلال استخدام المقاييس. تُعرف مقاييس جودة البرمجيات بأنها المقاييس الكمية للسمة التي تصف نوعية المُنتَج أو العملية البرمجية. يبدأ استخدام المقاييس لتحسين نوعية وأداء وإنتاجية البرامج بعملية موثقة لتطوير البرامج التي يتم تحسينها بشكل تدريجي. تؤسَّس الأهداف بالنسبة إلى مدى تحسينات الجودة والإنتاجية المطلوبة خلال الفترة الزمنية المحددة. تُستمد هذه الأهداف من الأهداف الإستراتيجية للمؤسسة التجارية كما ينبغي أن تتسق معها أيضاً.

يجب اختيار المقاييس المفيدة في تحقيق الأهداف المحددة للبرنامج (المُستمدة من متطلبات البرنامج) التي تدعم تقييم البرامج بما يتفق مع المتطلبات المحدَّدة. يجب من أجل تطوير التقييمات الدقيقة تأسيس خط أساس تاريخي يتألف من البيانات التي تم جمعها من مشاريع البرمجيات السابقة. يجب أن تكون البيانات المجمعة دقيقة بالقدر المعقول وأن تكون مُجمعة من أكبر عدد ممكن من المشاريع وأن تكون متسقة وممثِلة لتطبيقات مشابهة للعمل الذي يتم تقييمه. ما إن يتم تجميع البيانات حتى تُصبح عملية الحساب القياسي ممكنة.

بعدئذ يتم تعريف المقاييس التي يمكن استخدامها لقياس التقدم الدوري في تحقيق أهداف التحسين. يمكن استخدام بيانات المقاييس المُجمعة كمؤشرات على المجالات الصعبة لعملية التطوير وإجراءات التحسين المُحددة. يمكن مقارنة وتحليل هذه الإجراءات فيما يتعلق بأفضل عوائد الاستثمار التجاري. توفر بيانات القياس معلومات عن الاستثمار الحكيم في الأدوات من أجل تحسين الجودة والإنتاجية.

لا بد من تنفيذ آلية تغذية راجعة بحيث يتسنى لبيانات المقاييس أن توفر توجيهاً لتحديد الإجراءات اللازمة لتحسين عملية تطوير البرامج. تؤدي التحسينات المستمرة في عملية تطوير البرمجيات إلى منتجات عالية الجودة كما تؤدي إلى زيادة إنتاجية فريق التطوير. يجب إدارة وضبط إجراءات عملية التحسين من أجل تحقيق عملية تحسين ديناميكية مع مرور الوقت.

المقدرة على تتبع المُتطلبات
Requirements Traceability

أصبح من الواضح وبشكلٍ متزايد مدى أهمية التتبع الشامل للمتطلبات خلال مراحل التصميم والتطوير للمنتَج البرمجي وخاصة في المشاريع الكبيرة ذات المتطلبات التي قد يصل عددها إلى الآلاف أو عشرات الآلاف. بغض النظر عن منهجية التصميم والتنفيذ فمن المهم ضمان تلبية التصميم للاحتياجات خلال كافة مراحل التصميم. ينبغي من أجل ضمان تصميم وتطوير المنتَج وفقاً لمتطلبات المُنتَج في جميع مراحل التطوير أن يتم تخصيص المتطلبات الفردية إلى عناصر تصميم. يجب أن يكون كل متطلب برمجي (كما قد يظهر على سبيل المثال في مواصفات مُتطلبات البرنامج) قابلاً للتمييز بشكلٍ فريد. ينبغي أن تكون المتطلبات التي تنتج عن قرارات التصميم (أي متطلبات التنفيذ) قابلة للتحديد بشكلٍ فريد كما ينبغي تتبعها جنباً إلى جنب مع المُتطلبات الوظيفية للمنتَج. لا تضمن هذه العملية فقط بناء جميع المزايا الوظيفية ومزايا السلامة في المنتج كما هو محدد، ولكنها أيضاً تقلل بشكلٍ كبير من احتمال عدم الانتباه لبعض المتطلبات خلال الإجراءات. يمكن للمزايا التي تم إغفالها أن تكون ذات تكلفة باهظة عندما تصبح عبارة عن تعديلات على التصميم في نهاية عملية التطوير.

إن مصفوفة تتبع المتطلبات (RTM) هي عموماً بُنية جدولية تُشكّل فيها مُحددات المُتطلبات الصفوف بينما تُشكل كيانات التصميم عناوين الأعمدة. تُعلّم خلايا المصفوفة الفردية بأسماء ملفات أو بمُحددات لنموذج التصميم للدلالة على أن المُتطلب قد تم استيفاؤه داخل كيان التصميم. تضمن الـ RTM الاكتمال والاتساق مع مواصفات البرنامج الذي يمكن أن يتحقق من خلال تشكيل جدول يسرد مُتطلبات المواصفات مقابل طريق تحقيق كل مُتطلب في كل مرحلة من مراحل عملية تطوير البرنامج.

مراجعات البرنامج
Software Reviews

تُعتبر المراجعات المناسبة زمنياً والمحدَّدة بشكلٍ جيد جزءاً لا يتجزأ من جميع عمليات التصميم. ينبغي أن يُنتج كل مستوى من التصميم على مُنجزات يخضع تصميمها للمراجعة. ينبغي أن تشمل خطط تطوير مشروع البرنامج على قائمة لمراحل التصميم والمُنجزات المتوقعة في كل مرحلة وتعريف سليم للمُنجزات التي سوف تُدقق في كل مراجعة.

إن لمراجعة جميع مواد التصميم فوائد عدة. أولاً وقبل كل شيء، يضطر المُصممون إلى رفع جودة عملهم عندما يعلمون أن عملهم هو قيد المراجعة. ثانياً، غالباً ما تكشف المراجعات النقاط غير الواضحة في التصميم والطرق البديلة لها. وأخيراً، تُستخدم الوثائق التي يتم إنشاؤها من خلال المراجعات للحصول على موافقات الوكالات على العملية والمنتَج.

يمكن أن تتخذ مراجعات البرامج عدة أشكال مختلفة:

- المعاينة والتصميم والكود.
- شرح حذر لتفاصيل الكود (Code walk-throughs).
- قراءة الكود.
- مراجعات لتوضيح المنتَج البرمجي إلى العميل "Dog-and-pony shows".

تُعتبر المعاينة نوعاً معيناً من أنواع المراجعة وقد ثبت أنها فعالة للغاية في الكشف عن العيوب وأنها اقتصادية نسبياً بالمقارنة مع التجريب. تختلف المعاينات عن المراجعات المعتادة في عدة نواحٍ:

- تُركز قوائم التحقق اهتمام المُراجع على المجالات التي كانت ذات صعوبات في الماضي.
- ينصب التركيز على اكتشاف الخلل وليس التصحيح.
- يستعد المراجعون لاجتماع المعاينة بشكلٍ مُسبق حيث يقومون بإعداد قائمة بالمشاكل التي قد اكتشفوها.
- يتم جمع البيانات في كل عملية معاينة ويتم تغذيتها في عمليات المعاينة المستقبلية لتحسينها.

بينت التجربة العامة مع المعاينات أن الجمع بين مُعاينات تصميم الكود عادة ما يزيل من ٦٠٪ إلى ٩٠٪ من العيوب في المنتَج. تُعرف المعاينات الروتينيات المعرضة للخطأ في وقتٍ مبكر حيث تُشير التقارير إلى أن المعاينات تؤدي إلى عدد أقل من العيوب بنسبة ٣٠٪ لكل ١٠٠٠ سطر من الكود مقارنةً مع ما يقوم به الشرح الحذر لتفاصيل الكود. تُعتبر عملية المعاينة منهجية بسبب قوائم التحقق القياسية التي تتبعها والمهام القياسية. كما أنها ذاتية التحسين لأنها تستخدم التغذية الراجعة الرسمية لتحسين قوائم التحقق ولمراقبة معدلات الإعداد والمعاينة.

عادة ما ينطوي الشرح الحذر "walkthrough" على شخصين أو أكثر يقومون بمناقشة التصميم أو الكود. كما يمكن أن يكون غير رسمي كجلسة كبيرة مرتجلة حول اللوح الأبيض، وقد يكون رسمياً كما هو الأمر في الاجتماع المُخطط له مع شفافيات وتقرير رسمي يرسل إلى الإدارة.

فيما يلي بعض الخصائص المميزة لـ walkthrough:

- عادة ما تتم استضافة وإدارة الـ walkthrough من قِبَل مُنفذ التصميم أو الكود قيد المراجعة.
- إن الغرض من الـ walkthrough هو تحسين الجودة التقنية للبرنامج بدلاً من تقييمها.

- يستعد جميع المشاركون لـ walkthrough من خلال قراءة وثائق التصميم أو الكود والبحث عن مجالات الاهتمام.
- ينصب التركيز على اكتشاف الخطأ وليس التصحيح.
- يتسم مفهوم الـ walkthrough بالمرونة ويمكن تكييفه للاحتياجات المحددة للمنظمة التي تستخدمه.

عندما يُستخدم الـ walkthrough بذكاء فإنه يمكن أن يسفر عن نتائج مماثلة لنتائج المعاينة، أي أنه يمكن أن يؤدي إلى العثور على ما بين ٣٠٪ و ٧٠٪ من الأخطاء في البرنامج. لقد ثبت أن الـ walkthrough هو أقل فعاليةً من المعاينات بشكلٍ هامشي، ولكن يمكن أن يكون في بعض الظروف مُفضلاً.

تُعتبر قراءة الكود البديل لعمليات المعاينة وwalkthroughs. يتم في قراءة الكود قراءة الكود الأصل من أجل الأخطاء. كما يُعلّق القراء على الجوانب النوعية للكود مثل التصميم والأسلوب وسهولة القراءة وقابلية الصيانة والكفاءة. تنطوي قراءة الكود عادة على شخصين أو أكثر يقرؤون الكود بشكل مستقل ومن ثم يجتمعون مع كاتب الكود لمناقشته. يُقدم كاتب الكود القوائم الأصل إلى قرّاء الكود من أجل التحضير للاجتماع. يقرأ شخصان أو أكثر الكود بشكل مستقل. يستضيف كاتب الكود عند انتهاء المراجعين من قراءة الكود اجتماع قراءة الكود الذي يركز على المشاكل التي اكتشفها المراجعون. أخيراً، يقوم كاتب الكود بإصلاح المشاكل التي حددها المراجعون.

أما "dog-and-pony shows" فهي المُراجعات التي يتم فيها توضيح المنتَج البرمجي إلى العميل. إن الغرض من هذه المراجعات هو التوضيح للزبون أن المشروع قابل للتطبيق وبذلك فهي مُراجعات إدارية بدلاً من أن تكون مُراجعات تقنية. ينبغي ألا يتم الاعتماد على هذه المراجعات لتحسين الجودة التقنية للبرنامج. يأتي تحسين التقنية من المعاينات والـ walkthroughs وقراءة الكود.

القدرة على التنبؤ بالأداء ومحاكاة التصميم
Performance Predictability and Design Simulation

إن محاولة التنبؤ بأداء النظام في الزمن الحقيقي هو نشاط رئيسي في التصميم وهو ما يغفل عنه غالباً بعض مطوري البرامج. غالباً ما يُمضي مُصممو البرامج خلال مرحلة التكامل ساعات لا تحصى في محاولة لضبط النظام الذي يحتوي في تصميمه على بعض المعوقات. توفر النقاط التالية في مقدمة عملية التصميم مدخلاً أساسياً في مواصفات تصميم البرمجيات: (١) تقديرات التنفيذ لواجهات ربط النظام، (٢) أزمنة الاستجابة للأجهزة الخارجية، (٣) عدد مرات تنفيذ الخوارزمية، (٤) زمن تبديل سياق نظام التشغيل، (٥) عدد مرات الوصول إلى أجهزة الإدخال/الإخراج.

ينبغي من أجل التصاميم وحيدة المعالج تطبيق تقنيات النمذجة الرياضية مثل "تحليل المعدل مفرد الأسلوب" (Rate Monotonic Analysis, "RMA") لضمان إمكانية تنفيذ جميع العمليات المطلوبة من وحدة المعالجة ضمن الأطر الزمنية المتوقعة. كثيراً ما يقع مصممو الأنظمة في فخ اختيار المعالجات قبل الأخذ بعين الاعتبار تصميم البرمجيات حيث يتعرضون لخيبة أمل كبيرة "وتوجيه أصابع الاتهام" عند الإفراج عن المنتج. لا بد لاختيار المعالج في المشروع الناجح إلا أن يتم بعد اكتمال دراسة تحميل المعالج.

يكتسب إجراء تحليل أداء النظام في البداية في التطبيق متعدد المعالجات نفس القدر من الأهمية. يمكن أن يكون من الصعب تشخيص وإيجاد حلول لشذوذات النظام متعدد المعالجات مع توقع الاتصالات الثقيلة بين المعالجات إضافة إلى توقعات وظيفية ثقيلة أخرى. غالباً ما يكون قصور الأداء الذي يبدو أنه خطأ من أحد المعالجات نتيجةً لانهيار مُتحرك لقصورات أصغر من قِبَل أحد المعالجات أو النظم الفرعية الأخرى. يمكن التخلص من سنوات مرحلة تكامل حلول العيوب من خلال بدء العمل بتحليل تصميم النظام و/أو محاكاة التصميم. تتوفر الأدوات التجارية وبسهولة للمساعدة في أداء تحليل اتصالات الشبكة والمعالجات ومحاكاة التنفيذ. بالأخذ بعين الاعتبار هرم الجهد المطلوب في تصميم البرمجيات، فإن تصحيح الخلل في طليعة التصميم يؤدي في نهاية المطاف إلى وفورات هائلة في التكاليف وزيادة الموثوقية.

كتابة الكود (الترميز)
Coding

كان المُصطلح "تطوير البرمجيات" لسنوات عديدة مرادفاً لكتابة الكود. أما اليوم فإن كتابة الكود بالنسبة للعديد من مجموعات تطوير البرمجيات هو أحد أقصر مراحل تطوير البرمجيات. وفي الواقع، تتم كتابة الكود في بعض الحالات (وعلى الرغم من أن ذلك نادراً جداً في عالم تطوير البرمجيات) في الزمن الحقيقي بشكلٍ آلي من وثائق تصميم عالية المستوى (mspecs) عن طريق أدوات مؤتمتة تدعى "مولدات الكود".

تتوقف فعالية مرحلة كتابة الكود (سواء كانت مع أو بدون مولدات كود آلية) على نوعية واكتمال وثائق التصميم المولدة في مرحلة تطوير البرمجيات التي تسبق مباشرة مرحلة كتابة الكود. ينبغي أن تكون عملية كتابة الكود عملية تحول بسيطة من المواصفات النمطية وبالأخص الكود غير الحقيقي (pseudocode). لا تترك الـ mspecs الكاملة ولا الـ pseudocode المطور بشكل مناسب سوى القليل من التفسير في مرحلة كتابة الكود مما يقلل فرصة الخطأ.

إن أهمية نمط كتابة الكود (شكل الكود) ليست بضخامة أهمية القواعد التي تسهل استيعاب التدفق المنطقي (كيفية ترابط الكود). وبنفس السياق، يجب على التوثيق (التعليقات) الذي يتم مع الكود أن يُعالج في معظم الأحيان "لماذا" تم تطبيق الوظيفة بدلا من "كيف" تم تطبيقها. يُساعد ذلك القارئ على فهم السياق الذي تم فيه

استخدام قطاع معين من الكود. مع بعض الاستثناءات القليلة الثمينة (مثل سواقات الأجهزة عالية الأداء)، ينبغي تقدير قيمة نوعية كود المصدر من خلال قابلية قراءته وليس من حجمه الخام (أي عدد الأسطر) أو من خلال قدرته على الاستفادة من ميزات المعالج.

أدوات دعم التصميم
Design Support Tools

يتطلب تطوير البرمجيات عملاً مكثفاً جداً، ومن ثم فإنه عرضةً للخطأ البشري. أصبحت الحزم التجارية التي تدعم تطوير البرمجيات في السنوات الأخيرة وعلى نحو متزايد أكثر قوةً وأقل كلفةً وأكثر توفراً للتقليل من الوقت الذي يُقضى للقيام بأشياء يمكن لأجهزة الكمبيوتر القيام بها. على الرغم من أن اختيار الأدوات المناسبة يمكن أن يعني التفاني المُقدَّم لبعض الموارد الأكثر موهبة في فريق التطوير، إلا أنه يمكن أن يجلب زيادة كبيرة وطويلة الأمد في إنتاجية المجموعة.

لقد استفادت مؤسسات تطوير البرمجيات جيداً من أدوات CASE التي تقلل من الوقت الذي يُقضى في توليد وثائق تصميم واضحة وشاملة. تتمتع الحزم الآلية لتصميم البرمجيات بمزايا كثيرة. يمكن أن يستخدم التوثيق الرسمي كدليل على مطابقة إجراء تطوير المنتجات من أجل موافقات الوكالات. تُيسر الوثاق الواضحة والمُحدَّثة تحسين الاتصالات بين المهندسين مما يؤدي إلى تصاميم أكثر فعاليةً وموثوقيةً. تُخفض الأشكال الموحدة للوثائق منحنيات التعلم المقترنة مع أوصاف التصميم الفريد بين مصممي البرامج مما يؤدي إلى صياغة تصميم أفضل وبالوقت المناسب. كما ينخفض مجموع تكاليف دورة حياة البرامج (وخصوصاً أثناء الصيانة) وذلك بسبب زمن الـ ramp-up المُخفض والتعديلات الأكثر كفاءةً وموثوقيةً. أخيراً، يمكن للوثائق الإلكترونية أن تُخزن وأن يُحفظ بنسخ احتياطية منها بسهولة خارج الموقع مما يؤدي إلى تفادي حصول أي أزمة في حال وقوع كارثة بيئية. بالمختصر، يترافق التكييف مع أدوات CASE بتكلفة في بداية المشروع إلا أنه يتم استردادها من خلال تحسينات كبيرة في نوعية البرامج والقدرة على التنبؤ بالزمن اللازم للتطوير.

التصميم كأساس للتحقق والتثبت
Design as the Basis for Verification and Validation

تضمن عملية التحقق تلبية جميع منتجات مرحلة تطوير مُحدَّدة لتوقعات معينة. قبل الشروع في المستوى التالي الأدنى أو مرحلة التالية الأدنى في التصميم، ينبغي التحقق من مُنتَج (أو نواتج) المرحلة الراهنة مقابل مدخلات المرحلة السابقة لها. لا يمكن أن تكون عملية التصميم "جيدة" من دون عملية التحقق المتأصلة حيث تسيران بطبيعة الحال جنباً إلى جنب.

ينبغي أن تُحدد خطط إدارة مشروع البرامج (خطط ضمان جودة البرامج) كل مراجعات التصميم. يولِّد كل مستوى تصميم وثائق للمراجعة أو مُنجزات للتحقق منها مقابل مطالب المرحلة السابقة. ينبغي أن تصف خطط إدارة البرامج ولكل نوع من المراجعات ما يلي: (١) الغرض، (٢) المواد المطلوبة، (٣) قواعد الجدولة، (٤) نطاق المراجعة، (٥) توقعات الحضور، (٦) مسؤوليات المراجعة، (٧) الشكل الذي يجب أن تبدو عليه مذكرة المراجعة، (٨) فعاليات المتابعة، (٩) أية مُتطلبات أخرى تتعلق بتوقعات الشركة.

على مستوى الكود، ينبغي أن تضمن مُراجعات الكود تلبية الوظيفية المُطبقة ضمن روتينٍ ما جميع التوقعات الموثقة في "mspecs". كما يجب معاينة الكود للتأكد من تلبية جميع قواعد كتابة الكود.

يشتمل خرج التصاميم الجيدة للبرمجيات على متطلبات تنفيذ. تشتمل متطلبات التنفيذ كحدٍ أدنى على القواعد والتوقعات الموضوعة على المصممين لضمان تماثل التصميم فضلاً عن القيود والضوابط والتوقعات الموضوعة على التصاميم لضمان تحقيق المستوى الأعلى من المُتطلبات. قد تشتمل الأمثلة العامة لمتطلبات التنفيذ على: (١) قواعد للوصول إلى منافذ الإدخال/الإخراج، (٢) متطلبات التوقيت للوصول إلى الذاكرة، (٣) تحكيم الإشارات، (٤) مخططات الاتصالات بين المهام، (٥) مهام عنونة الذاكرات، (٦) قواعد التحكم بالحساس أو الجهاز. يجب أن تُعالج عملية التحقق والتثبت متطلبات التنفيذ وكذلك مُتطلبات المستوى الأعلى للبرامج لضمان عمل المنتج كما صُمم له أن يعمل.

دورة حياة التحقق والتثبت
Verification and Validation Life Cycle

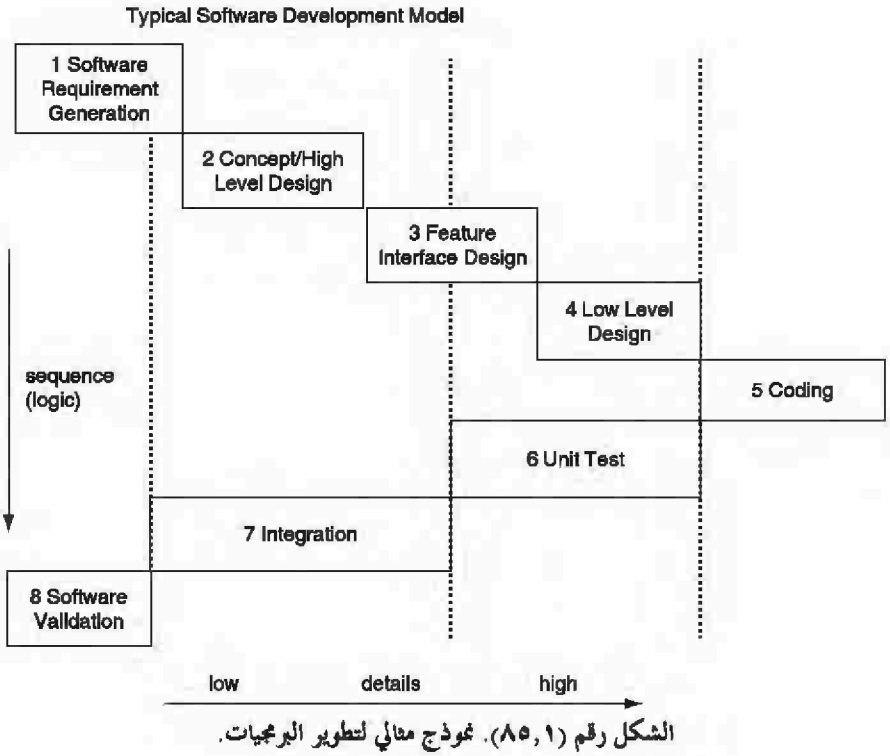
يحدد الـ SVVP الإطار الذي تنطلق منه فعاليات التحقق والتثبت. يُعتبر معيار الـ IEEE 1012-1986 مفصّلاً جداً عن دورة حياة التطوير ومُنتجات التحقق والتثبت التي يتم إنشاؤها في كل مرحلة. يستدعي المعيار على وجه التحديد المراحل التالية:

- مرحلة المفهوم.
- مرحلة المتطلبات.
- مرحلة التصميم.
- مرحلة التنفيذ.
- مرحلة الاختبار.
- مرحلة التثبيت.
- مرحلة التشغيل والصيانة.

الأكثر اعتياداً في قطاع الصناعة اتباع النموذج الأبسط للتحقق والتثبت من البرامج كما هو مبين في الشكل رقم (٨٥,١).

يُعتبر هذا النموذج بسيطاً من حيث إنه يركز فعاليات التحقق والتثبت على البرامج التي تم إنشاؤها بدلاً من التركيز على إنتاج كل مرحلة من مراحل التطوير. كما تُستخدم وثيقة أخرى، تُدعى خطة ضمان جودة البرمجيات (Software Quality Assurance Plan، "SQAP")، لتغطية جوانب التحقق والتثبت الأخرى لتطوير المنتجات مثل مراجعات مُتطلبات البرامج وتحليل المخاطر والأخطار ومراجعات التصميم.

يُغطي اختبار التثبت من البرامج، باستخدام النموذج المُبين في الشكل رقم (٨٥,١)، التثبت من مُنتَج البرمجيات من خلال الاختبار مقابل المتطلبات المولّدة في مرحلة توليد المتطلبات. تُشابه عملية التثبت من البرنامج اختبار الصندوق الأسود. يُغطي اختبار التكامل التحقق من الميزات المحددة في مرحلة توليد المتطلبات ومرحلة المفهوم/التصميم رفيع المستوى ومرحلة تصميم ترابط الميزات. يفحص اختبار التكامل جمع وحدات البرمجيات حيث يتم ربط وحدات البرمجيات معاً لتوفير ميزة المُنتَج أو مستوى رفيع من الأداء الوظيفي. في النهاية، يُغطي اختبار الوحدات التحقق عند المستوى الأدنى الذي يتضمن مرحلة تصميم ترابط الميزات ومرحلة تصميم منخفض المستوى. عادة ما يغطي اختبار الوحدات التحقق من مجموعات الوحدات البرمجية الصغيرة.



الشكل رقم (٨٥,١). نموذج مثالي لتطوير البرمجيات.

نظرة عامة على التحقق والتثبت
Verification and Validation Overview

تصف النظرة العامة على التحقق والتثبت العديد من التفاصيل الإدارية للمشروع التي يجب معالجتها من أجل أداء مهام التحقق والتثبت، حيث تشمل تنظيم التحقق والتثبت من حيث الجدول الزمني والموارد والمسؤوليات وأية أدوات خاصة وتقنيات ومنهجيات.

يصف التنظيم الشخص المسئول عن تنفيذ جهود التحقق والتثبت المختلفة. يُستخدَم هذا القسم في الممارسة العملية لوصف الشخص المسئول عن الحفاظ على المختبرات والشخص المسئول عن إدارة اختبار التثبت والشخص المسئول عن إدارة اختبار التحقق والشخص المسئول عن حل الحالات الشاذة.

تُعتبر جدولة التحقق والتثبت مهمة جداً لتخطيط الفعاليات من أجل دعم الجدول الإجمالي للمُنتَج. غالباً ما يتم إدخال جدول عالي المستوى في خطة التحقق والتثبت من البرامج بينما يتم الحفاظ على جدول مفصل بعيداً عن خطة التحقق والتثبت من البرامج. يتم ذلك بشكلٍ عام بسبب اختلاف تطبيقات برامج توليد الجدول وتطبيقات برامج توليد الوثائق.

يصف قسم الموارد كل التجهيزات والبرمجيات والبيئة الصلبة اللازمة لتنفيذ فعاليات التحقق والتثبت. كما يجب إدراج قسم فرعي للتثبت من استخدام أي أدوات برمجية حيث يتم استخدام الأداة البرمجية كجزء من عملية التحقق والتثبت. يجب وفقاً للوائح الـ FDA (FDA، لائحة نظام الجودة، 21 CFR Part 820، ١ يونيو ١٩٩٧) التثبت من أي جهاز يُستخدم في نظام جودة أي شركة. ومن الواضح أنه يندرج أي تثبيت اختبار (test fixture) أو أي تطبيق برمجي مخصص تحت هذا التعريف. ومن ثم فمن الضروري التثبت من أداة البرمجيات.

فعاليات دورة حياة التحقق والتثبت Verification and Validation Life Cycle Activities

عادة ما يتضمن هذا القسم في الـ SVVP على المعايير والمدخلات/المخرجات والمراجعات ومنهجية الاختبار والتدريب لكل مرحلة من مراحل التحقق والتثبت. تصف المعايير الهدف الذي يجب أن تحققه كل مرحلة. تُحدد مرحلة المدخلات/المخرجات الأشياء المطلوبة كمدخلات ومخرجات لكل مرحلة.

ومن المفيد أيضاً أن تضاف فقرة فرعية على الوظيفة العامة التي سيتم اختبارها. لا يستدعي معيار IEEE ذلك إلا أن إضافة هذه الفقرة الفرعية يُعتبر مفيداً لمراجعي خطة التحقق والتثبت من البرنامج. إنها تقدم معلومات عن الوظيفية المُجدولة للاختبار.

وثائق التحقق والتثبت Verification and Validation Documentation

يرتبط مع كل مرحلة تحقق وتثبت فعاليات ووثائق خاصة بها. عادة ما يكون هناك خطط اختبار وإجراءات اختبار ونتائج وتقارير اختبار لكل مرحلة من مراحل التثبت من الوحدة والتثبت من التكامل والتثبت من البرنامج.

توجد عادة بعض الوثائق الإضافية من أجل التحقق من البرمجيات. عادة ما يكون لمجموعة التثبت من البرمجيات دليل "المُتطلبات مقابل الاختبار". يضمن الدليل تغطية جميع المُتطلبات المحددة في الـ SRS في اختبارٍ ما. ومن المفيد أيضاً أن تشتمل وثيقة الدليل على دليل "الاختبار مقابل المُتطلب". يساعد هذا المرجع العكسي في ضمان أن يكون للمُتطلب تغطية كافية. كما تولّد مجموعة التثبت من البرنامج وثيقة استجابة للإجراءات التي تخدم كوسيلة لإغلاق أي من القضايا أو التعليقات التي وقعت في أثناء أي عملية تثبت رسمية. وأخيراً، تكون مجموعة التثبت من البرنامج مسئولة عن كتابة تقرير نهائي يُدعى تقرير التحقق والتثبت من البرمجيات (SVVR) الذي يلخص جميع الفعاليات من مراحل التحقق والتثبت السابقة.

الإجراءات الإدارية للتحقق والتثبت Verification and Validation Administrative Procedures

يوفر قسم الإجراءات الإدارية للتحقق والتثبت إرشادات إضافية بشأن سبل إجراء التحقق والتثبت. يُعرِّف المعيار IEEE Std 1012-1986 هذا القسم بأنه يشتمل على إعداد التقرير/الحل للحالة الشاذة وسياسة تكرار المهمة وسياسة الانحراف وإجراءات الضبط والمعايير/الممارسات/المؤتمرات (conventions).

ومن المفيد أيضاً وجود قسم للمقاييس. يشمل المعيار IEEE Std 730.1-1989 وهو معيار IEEE لخطط ضمان جودة البرمجيات (Software Quality Assurance Plans, "SQAP") على مقاييس كجزء من الـ SQAP. تصف الأقسام الفرعية التالية بمزيد من التفصيل إعداد التقارير عن الحالات الشاذة وجمع المقاييس وإعداد التقارير عنها كما أنها توفر بعض المواضيع الإضافية التي تعتبر هامة في التحقق والتثبت.

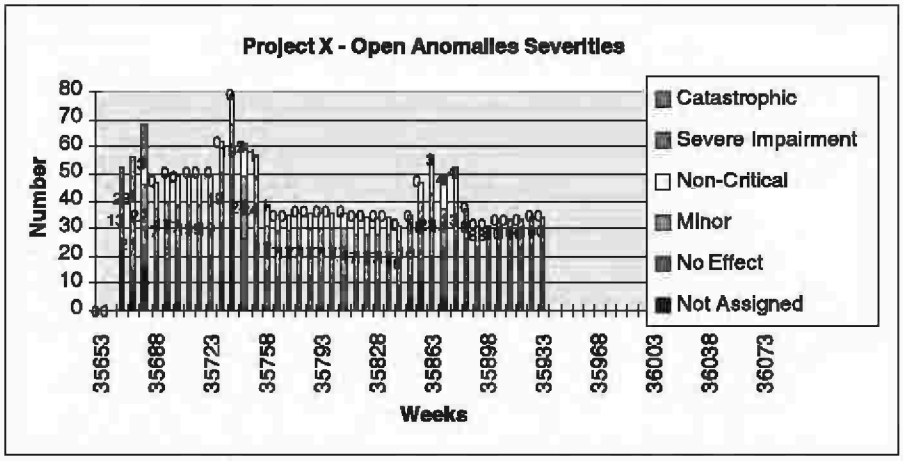
إعداد التقارير عن الشذوذ ووضع الحلول لها Anomaly Reporting and Resolution

من إحدى نتائج التحقق والتثبت توليد الحالات الشاذة التي وجدت أثناء الاختبار. يجب أن تكون عملية إعداد التقارير عن الحالات الشاذة جاهزة من أجل تسجيل الحالات الشاذة. استخدمت بعض المؤسسات كمبيوتراً مخبرياً لتوثيق الحالات الشاذة. غالباً ما يتم استخدام برنامج قاعدة بيانات يعتمد على جهاز الكمبيوتر الشخصي لتسجيل وتخزين الشذوذ إلكترونياً.

بالإضافة إلى ذلك، يجب أن يكون إجراء إيجاد الحلول للحالات الشاذة التي يتم اكتشافها ضد منتجات البرمجيات جاهزاً. يمكن أن يكون ذلك عبارة عن لجنة من الأشخاص المسئولين عن المنتج أو مجرد مُنسق اختبار البرنامج الذي يعمل بشكلٍ أساسي مع مطور البرنامج. تُحدد الأطراف المسئولة مخاطر كل حالة شذوذ.

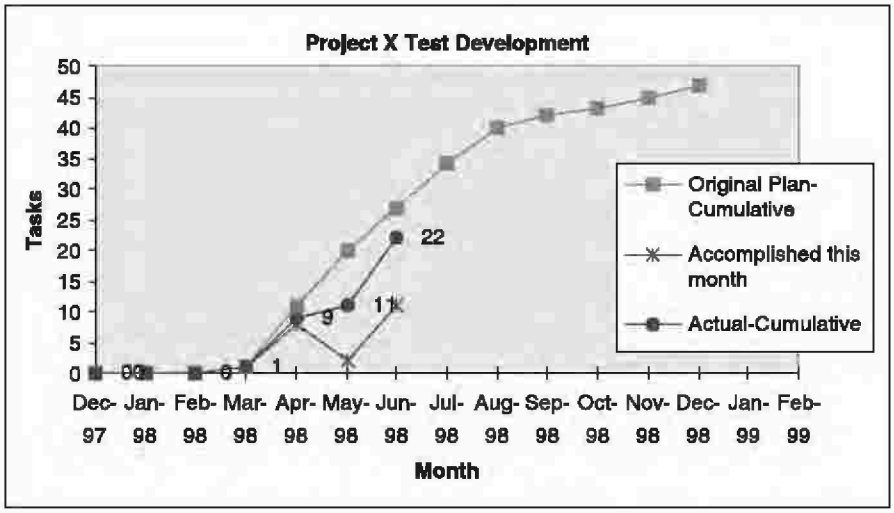
مقاييس التحقق Validation Metrics

هناك جوانب عديدة للتحقق والتثبت حيث يكون من المفيد من أجلها وضع بعض المقاييس. الأولى والأكثر وضوحاً هي مقاييس الشذوذ. يبين الشكل رقم (٨٥.٢) مثالاً على تعقب مقياس الشذوذ. يبين هذا الرسم البياني تتبع الشذوذات ذات الخطورة العامة كتابع للزمن.



الشكل رقم (٨٥,٢). مخطط قياسي يُبين الشذوذات المفتوحة مُخزنةً وفقاً لخطورتها.

الجانب الثاني من مقاييس الاختبار هي تلك التي تنطوي على تطوير الاختبار. تُستخدم هذه المقاييس لمراقبة عملية تطوير الاختبار ولمراقبة بارامترات الاختبار الشاملة لجميع المشاريع. تضمن مراقبة عملية تطوير الاختبار سير هذه العملية وفقاً للجدول المحدد. يبين الشكل رقم (٨٥,٣) مثالاً على ذلك.



الشكل رقم (٨٥,٣). مُراقبة عملية تطوير الاختبار.

إدارة التهيئة Configuration Management

مثلما تُعتبر المحافظة على تهيئة كود تطوير المُنتجات أمراً مهماً، فإن لممارسة إدارة التهيئة على بروتوكولات الاختبار نفس الأهمية. تشمل تهيئة مواد الاختبار التالي:

- مكتبات النصوص.

- نصوص الاختبارات.
- البروتوكول اليدوي.
- نتائج الاختبارات.
- خطط وإجراءات الاختبارات.
- تصميم نظام الاختبارات.

قوالب البروتوكول Protocol Templates

من المستحسن وجود قالب لبروتوكول الاختبار من أجل جعل تطوير الاختبارات أكثر تماثلاً وفعاليةً. إذا كانت نصوص الاختبار مكتوبة عندئذ يُقترح وجود مجموعة من المبادئ التوجيهية لعملية التطوير.

تنفيذ الاختبار Test Execution

يناقش هذا القسم المناهج المُستخدمة عند تنفيذ البروتوكول اليدوي والنصوص الآلية.

تحسين العملية Process Improvement

ليس هناك أي عملية اختبار كاملة. هناك دائماً وسائل لتحسين عملية تطوير الاختبارات لكي تقوم بالأمور بكفاءة أكبر. ومن ثم يجب تخصيص بعض الوقت لمراجعة العمليات الحالية التي هي قيد العمل. إذا كان فريق الاختبار كبيراً بما فيه الكفاية فيمكن إنشاء فريق عمليات لمعالجة العمليات وتغييرات العمليات ضمن فريق الاختبار. يجب أن تحدث مناقشات بين أعضاء فريق الاختبار لتحديد العمليات التي عملت بصورة جيدة والعمليات التي لم تعمل بشكل جيد.

تطوير الاختبار
Test Development

تحليل المُتطلبات ومتطلبات التخصيص Requirements Analysis and Allocating Requirements

يُفترض في يومنا هذا أن تكون مواصفات مُتطلبات البرمجيات(SRS) مكتوبةً. يصف الـ SRS السلوك ووظائف البرمجيات في الجهاز الطبي. يجب أن تحصل الـ SRS على مدخلاتها من المصادر التالية :

- اشتقاق مواصفات المنتج.
- إجراءات الضبط من تحليلات المخاطر والأخطار.
- المتطلبات التنظيمية مثل الحوسبة ذات الصلة بالسلامة.

يُركز جزء كبير من جهد التحقق والتثبت على إثبات أن مُتطلبات البرامج قد تم استيفاؤها. لذا، يجب تعداد مُتطلبات الـ SRS للسماح بالتخصيص السليم لمراحل الاختبار المختلفة وللاختبارات المُحددة.

يشمل التحليل الأولي للـ SRS تحديد مكان اختبار مُتطلب معين. غالباً ما يستند مكان اختبار المُتطلب إلى قدرات تثبيت الاختبار (test fixture) المستخدم في التثبت. يمكن اختبار معظم المُتطلبات عادةً في مستوى التثبت. ثم يتم اختبار توازن المتطلبات خلال اختبار التكامل أو اختبار الوحدة. تتم المُتطلبات التي لديها حافز خارجي (إلى وحدة المعالجة المركزية "CPU") وقدرة على مُراقبتها خارجياً بشكلٍ عام في مستوى التثبت. أما الاستثناءات لذلك فهي الحالات التي قد تتطلب توقيتاً محدداً جداً حيث يتم حساب التوقيت داخلياً في وحدة المعالجة المركزية.

أما المتطلبات التي يتم اختبارها خلال اختبار التكامل أو اختبار الوحدة فهي التي لا يمكن عادة اختبارها بشكلٍ كافٍ أثناء اختبار التثبت. إن اختبار مُتطلب "خطأ مُراقب البرنامج software watchdog error" هو مثال جيد على المُتطلب الذي يتم اختباره خلال اختبار التكامل أو اختبار الوحدة. يقوم البرنامج في هذه الحالة بمراقبة نفسه ولا يكون هناك أي وسيلة خارجية تتسبب بالأخطاء.

أما الخطوة الثانية في مرحلة تحليل المُتطلبات فهي تحديد أفضل وسيلة لتجميع المتطلبات في اختبارات. إن أحد المنهجيات البسيطة نسبياً لتحقيق هذه الغاية هي إنشاء اختبارات على أساس أقسام الـ SRS. عادة ما يَصف أحد أقسام الـ SRS إحدى الميزات بشكلٍ كامل تقريباً. يتم من خلال اتباع شكل الـ SRS تطوير الاختبارات وفقاً للميزة وهو الأمر الذي يُعتبر نهجاً منطقياً. بطبيعة الحال، سوف يكون هناك مُتطلبات في الـ SRS لا يمكن فيها تطبيق هذا النهج. يجب في مثل هذه الحالات إجراء المزيد من التحليل لتقييم ما إذا كان يجب إعداد اختبارات إضافية أو ما إذا كان بالإمكان تخصيص أحد هذه المُتطلبات الخاصة لتتبع الـ SRS.

كما ذُكر في بداية هذا المقطع، يجب كتابة إجراءات الضبط من تحليل المخاطر والأخطار في الـ SRS. إذا تم ذلك، عندئذ يمكن وببساطة تحليل مُتطلبات المخاطر والأخطار تلك وتخصيصها إلى اختبارات كما هو الحال بالنسبة إلى جميع المتطلبات الأخرى.

إن تخزين مصفوفة التخصيص هو جانب نهائي لتخصيص متطلبات الاختبار. إنه بسيط بما يكفي لإبقاء المصفوفة في وثيقة جدول بيانات أو وثيقة إلكترونية. ومع ذلك فإنه يمكن أيضاً تخزينها في قاعدة بيانات. إن ميزة تخزينها في قاعدة البيانات هي تمكُّن المُستخدم من تعقب تطور الاختبار وتوليد العديد من التقارير.

مراحل ومناهج الاختبار Testing Phases and Approaches

اختبار الوحدة Unit Testing

يُجرى اختبار الوحدة على أصغر كمية من الكود. ما الذي يُشكل "وحدة" هو موضوع غالباً ما يكون عُرضة للبحث والمناقشات الطويلة. بغض النظر عن تعريف الوحدة، فإن القصد من اختبار الوحدة هو ضمان اختبار وحدات البرنامج ذات أدنى مستوى.

هناك العديد من الطرق لاختبار الوحدة مثل تحليل مسار الفروع أو اختبار ارتباط الوحدات. يستخدم المُطوِّر في تحليل مسار الفروع أداة تطوير ليخطو خلال كل مسار داخل الوحدة. أما في اختبار ارتباط الوحدات، فيقوم المُطور باختبار الوحدة من منظور "الصندوق الأسود". ومن ثم تُختبر الوحدة فقط من خلال تغيير المدخلات إلى وحدة البرمجيات.

يعتمد نوع اختبار الوحدة الذي سيكون مطلوباً على عوامل كثيرة. يُجرى في الأجهزة الطبية عادةً تحليل مُفصل لمسار الفروع على وحدات البرامج التي تعتبر حرجة بالنسبة لسلامة المريض أو المستخدم. أما بالنسبة إلى وحدات البرامج الأخرى الأقل خطورة فيمكن اختبارها بشكلٍ مُبرر باستخدام اختبار ارتباط الوحدات الأقل شموليةً.

اختبار التكامل Integration Testing

هناك عدة تعريفات لاختبار التكامل. يُغطي التعريف التكامل بين اثنين أو أكثر من الوحدات البرمجية. أما التعريف الأوسع نطاقاً فيغطي التكامل بين النظم الفرعية المادية (كل منها مع برنامجه المبني فيه) للتأكد من أنها تعمل معاً.

يجلب كل نوع من اختبارات التكامل الكيانات المُطوَّرة المنفصلة معاً لضمان عملها مع بعضها. يشتمل نوع الاختبار الذي يُجرى عادةً في هذه المستويات على كتابة البروتوكولات من أجل تجريب الارتباط.

اختبار التثبت Validation Testing

إن اختبار التثبت هو عملية إثبات أن المنتج يُحقق مواصفات المنتج. كما أنه يمكن أن يعني الذهاب خطوة إضافية ومحاولة التأكد من أن المنتج لا يقوم بما ليس من المفترض القيام به. ومن ثم وظفّت العديد من طرق الاختبار لتجريب المنتج البرمجي. فيما يلي شرح لهذه الطرق.

الاختبار القائم على المتطلبات Requirements-Based Testing

إن الاختبار القائم على المتطلبات هو النهج الأساسي المُستخدم للتثبت من البرنامج. أساساً، يتم تحليل المُتطلبات من الـ SRS وتخصيصها لاختبارات محددة. يُستخدم في تطوير هذه الاختبارات أساليب اختبار مختلفة مثل اختبار العتبة أو اختبار الحدود. إن خطوات الاختبار هي الإجراءات المتتابعة التي يجب اتخاذها لإثبات تحقيق المُتطلب.

اختبار العتبة Threshold Testing

إن اختبار العتبة هو عملية إثبات أن حدثاً ما سوف يحدث عندما يتجاوز بارامتر محدد قيمة معينة. يُعتبر الإنذار (مثل إنذار انخفاض مستوى البطارية) مثالاً جيداً على مثل هذا البارامتر في الجهاز الطبي. إذا كان الجهاز

يعمل على البطارية وانخفض مستوى الجهد في البطارية إلى ما دون مستوى العتبة عندئذ يتم إعلان الإنذار. يتم في هذه الأنواع من الاختبارات وضع نطاق تسامح حول مستوى العتبة. يُحدَّد التسامح من خلال المتطلبات في الـ SRS كما يمكن أن تتأثر بدقة نظام الاختبار. يُصمم الاختبار عندئذ ليقوم بتغيير جهد البطارية حتى يعبر خلال العتبة بحيث يشهد "اعتراضاً للإنذار" ضمن نطاق التسامح. بعد أن يتم اعتراض الإنذار، يأتي الجزء التالي من اختبار العتبة وهو تغيير البارامتر (وهو جهد البطارية في هذه الحالة) في الاتجاه المعاكس للتأكد من أن الإنذار لم يعد مُعلناً.

اختبار الحدود Boundary Testing

إن اختبار الحدود هو ممارسة اختبار بارامترٍ ما عند حدوده ومحاولة تجاوز هذه الحدود. تُعتبر قيمة الأكسجين المقاسة والمعرَّفة بأنها لا تزيد عن ١٠٠٪ إحدى الأمثلة الجيدة على ذلك. (يمكن لقراءة الأكسجين أن تتجاوز ١٠٠٪ إذا كان حساس الأكسجين خارجاً عن المعايرة أو إذا كانت معايرته غير صحيحة). عندئذ يُصمم الاختبار ليُثبت أنه يمكن إظهار قراءة ١١٠٪ من الأكسجين. بالإضافة إلى ذلك، يراقب الاختبار رد فعل الجهاز إذا تجاوزت قيمة الأكسجين المقاسة ١١٠٪.

اختبار الإجهاد Stress Testing

إن اختبار الإجهاد هو عملية تعريض الوحدة لوابل من المدخلات العشوائية (كثيراً ما تكون ضغطات على لوحة المفاتيح أو إدارة عشوائية للأزرار) في محاولة لإحداث فشل برمجي. يمكن القيام بذلك يدوياً أو باستخدام نظام اختبار آلي.

إن لكل من اختبار الإجهاد اليدوي واختبار الإجهاد الآلي مزاياه الخاصة. إن ميزة اختبار الإجهاد اليدوي هو أن المُختبر يمكنه اختبار بعض جوانب الجهاز ويمكنه ملاحظة السلوك الشاذ في نواحي أخرى من الجهاز. أما ميزة اختبار الإجهاد الآلي فتكمن في إمكانية إدخال نظام الاختبار لمدخلات الجهاز بسرعة أكبر بكثير من سرعة المُختبر البشري.

اختبار التشغيل الطويل Volume Testing

إن اختبار التشغيل الطويل هو عملية تجريب الوحدة لفترة ممتدة من الزمن. وجدنا في تطبيقاتنا الطبية أن تنفيذ اختبار الحجم لمدة ٧٢ ساعة يمكنه أن يكشف عن مشكلات لم يتم العثور عليها بأي طريقة آخرى.

يتم اختبار التشغيل الطويل في أغلب الأحيان بنظام اختبار آلي. لا يعني اختبار التشغيل الطويل بالضرورة استخدام الضغط العشوائي السريع على المفاتيح مثل اختبار الإجهاد. يستخدم اختبار التشغيل الطويل نهجاً منطقياً لاختبار كافة المسارات في البرامج. تُستخدم الخوارزميات الاحتمالية، مثل تابع كثافة الاحتمال (PDF)، عندما يكون الجهاز في وضعيةٍ معينة وذلك من أجل تحديد الوضعية التالية.

اختبار السيناريو Scenario Testing

إن اختبار السيناريو هو عملية كتابة اختبار يحاكي الاستخدام الفعلي للبرنامج من وجهة نظر المستخدم. على سبيل المثال، يجب أن يغطي الاختبار في آلات التخدير إحدى الحالة الإكلينيكية. يفيد اختبار السيناريو في كشف مشاكل النظام كما أنه يمكن أن يكشف عن عيوب في الاستخدام الكلي للمنتج.

اختبار النظام System Testing

يتم اختبار النظام للتثبت من الجهاز الطبي بأكمله وليس فقط البرمجيات المبنية في الجهاز. يمكن أن يكون هناك تداخل كبير بين اختبار البرمجيات واختبار النظام ويعتمد ذلك على طريقة تجهيز الاختبار. على سبيل المثال، عادة ما تتطلب البرمجيات المدمجة في جهاز التنفس الصناعي أنظمة فرعية ميكانيكية وكهربائية أخرى من أجل أداء عملها. من الصعب كتابة محاكاة متطورة لخداع برامج جهاز التنفس الصناعي للتفكير في أن كل الأنظمة الفرعية لجهاز التنفس موجودة. لذا، فقد يشابه نظام اختبار البرنامج نظاماً كاملاً. يعود للأطراف المسئولة ضمان معرفة درجة التداخل. الأهم من ذلك ينبغي اختبار كل شيء في كل مرحلة من مراحل تطوير المنتجات.

تنفيذ الاختبار وإعداد التقارير
Test Execution and Reporting

خطة الاختبار Test Plan

يجب من أجل التثبت من البرمجيات كتابة خطة الاختبار قبل تنفيذ الدخول في الاختبار. تتبع البُنية تلك البُنية المعرَّفة في ANSI/IEEE Std 1012. تتكون الجوانب الرئيسية لخطة الاختبار من وصف للتغييرات في البرمجيات وكذلك الاختبارات التي سيتم تنفيذها لإثبات التطبيق الصحيح لتلك التغييرات في البرمجيات. تُكتب خطة الاختبار كلما تم تنفيذ اختبار تثبت كامل أو اختبار ارتداد (regression test).

نموذج تهيئة الاختبار Test-Configuration Form

باعتبار أنه من المهم جداً أن يكون هناك قدرة على تكرار الاختبار فمن الضروري توثيق تهيئة الأجهزة التي استخدِمت لتنفيذ الاختبار. تُكتب هذه الوثائق على نموذج تهيئة الاختبار. يمكن لمحتويات هذه النماذج أن تختلف. إلا أن الحقول الرئيسية هي التاريخ والبند والرقم التسلسلي للبند كما أن هناك مكاناً لمهندس الاختبار لتوقيع اسمه. يُصبح نموذج تهيئة الاختبار جزءاً من مجموعة وثائق الاختبار.

تنفيذ البروتوكول اليدوي Executing Manual Protocol

يشتمل تنفيذ البروتوكول اليدوي للاختبار الرسمي على طباعة إجراءات الاختبار والمرور على خطوات الإجراءات. باعتبار أن هذه المطبوعات هي جزء من الوثائق الرسمية فإنه يجب على مهندس الاختبار توقيع وتأريخ هذه الوثائق. إذا وجد الفاحص مشاكل أثناء الاختبار فإنه يقوم بتوثيق الموضوع في إجراءات الاختبار.

من المهم التطرق إلى المذكرات المكتوبة بخط اليد على ورقة الإجراءات. يمكن أن تُمثل هذه المذكرات مشاكل تم العثور عليها أو انحرافات في الإجراءات. يجب أن يكون هناك خاتمة لجميع هذه العلامات. توثّق هذه الخاتمة رسمياً في وثيقة استجابة الإجراءات. سوف تتم تغطية وثيقة استجابة الإجراءات لاحقاً في هذا الفصل.

تنفيذ البروتوكول الآلي Executing Automated Protocol

عادة ما يصل تنفيذ البروتوكول الآلي إلى إعداد عمل مُجمَّع (batch job) الذي يُقدِّم سلسلة من الاختبارات المُجدولة للتنفيذ. يقوم مهندس الاختبار بتعبئة نموذج تهيئة اختبار ليكون كورقة لبدء الاختبار. يتم عندئذ تنفيذ العمل المُجمَّع هذا ليلاً وتُحلل النتائج في اليوم التالي.

أظهرت التجربة أنه يمكن أحياناً لاختبار العمل المُجمَّع أن يفشل. عادة ما يُعزى هذا الفشل إما إلى مواضيع التوقيت أو إلى تغير ميزة الجهاز الهدف من دون تغيير الاختبار. وعندما يحدث ذلك فمن المعتاد من الناحية العملية وببساطة تصحيح نص البرنامج وإعادة تنفيذ الاختبار. إلا أنه في الحالات التي يستهلك فيها تصحيح نص الاختبار كمية كبيرة من الزمن فإنه أكثر كفاءةً في بعض الأحيان من تنفيذ هذا الاختبار يدوياً لاستكمال الاختبار.

نتائج الاختبار Test Results

يجب تحليل نتائج الاختبار بعد توليدها. تتم مراجعة النتائج من الاختبارات الآلية عن طريق تنفيذ عمليات البحث عن الكلمات الرئيسية التي تشير إلى ما إذا كان للاختبار أية مشاكل. يُجرى تحليل إضافي من خلال مراجعة عينات من نتائج الاختبار الآلي. يضمن ذلك تنفيذ الاختبارات كما هو متوقع. قد يكون هناك حاجة لتنفيذ تحليل آخر مثلما هو الحال عندما يتم توليد كمية كبيرة من البيانات خلال الاختبار حيث يجب بعد ذلك تخفيض البيانات وتحليلها لتقييم صحتها.

يجب تدبر نتائج الاختبار جنباً إلى جنب مع غيرها من وثائق الاختبار. يجب ضم النتائج من إجراءات الاختبار اليدوي مع غيرها من الوثائق الورقية وتخزينها في موقع رسمي. كما يجب وضع ملفات النتائج الإلكترونية تحت تحكم التهيئة (configuration control). يتم ذلك باستخدام أداة إدارة التهيئة (configuration-management tool) أو بعملية النسخ الاحتياطي للملفات إلى وسيط مضبوط.

تقارير الاختبارات
Test Reports

تقرير اختبار التثبت من البرمجيات Software Validation Test Report

تشمل البنود الرئيسية في هذا التقرير خلاصة لما تم اختباره وكيف تم الاختبار والمشاكل التي تم مواجهتها وتوصية بإطلاق البرمجيات.

متطلبات لاختبار الدليل Requirements to Test Cross Reference

تغطي هذه الوثيقة السبل التي تم بموجبها تخصيص المتطلبات إلى الاختبارات.

وثيقة استجابة الإجراءات Procedures Response Document

كما ذكر آنفا، يجب مراجعة جميع العلامات المكتوبة بخط اليد على إجراءات الاختبار اليدوي وأن تنتهي بخاتمة. توثق نتائج هذه الخاتمة في وثيقة استجابة الإجراءات. نموذجياً فإن كل علامة في إجراء الاختبار تكون إما مشكلة برمجية أو خطأ في الإجراءات أو انحرافاً في الإجراءات أو تعليقاً.

معلومات إضافية

Additional Information

Bass L, Clements P, Kazman R. Software Architecture in Practice. Reading, MA, Addison Wesley Longman, 1998.

Bloesch A. Overview of Verification and Validation for Embedded Software in Medical Devices. Handbook of Medical Device Design. New York, Marcel Dekker, 2001.

Boehm BW. A Spiral Model of Software Development and Enhancement. Computer 5, 1988.

Boehm BW. Software Engineering Economics. Englewood Cliffs, NJ, Prentice Hall, 1981.

Booch G. Object-Oriented Analysis and Design with Applications, 2nd Edition. Redwood City, CA, Benjamin Cummings, 1994.

Booch G, Jacobson I, Rumbaugh J. The Unified Modeling Language User Guide. Reading, MA, Addison Wesley Longman, 1998.

Deutsch MS, Willis RR. Software Quality Engineering—A Total Technical and Management Approach. Englewood Cliffs, NJ, Prentice Hall, 1988.

Dyer M. The Cleanroom Approach to Quality Software Development. New York, Wiley, 1992.

Fairley RE. Software Engineering Concepts. New York, McGraw-Hill, 1985.

Food and Drug Administration, 21 CFR Part 820. Washington, DC, U.S. Government Printing Office, 1997.

Fries RC. Reliable Design of Medical Devices. New York, Marcel Dekker, 1997.

Fries RC, Pienkowski P, Jorgens J. Safe, Effective, and Reliable Software Design and Development for Medical Devices. Med Instrum 30(2), 1996.

Hatley DJ, Pirbhai IA. Strategies for Real-Time System Specification. New York, Dorset House, 1987.

Humphrey WS. Managing the Software Process. Reading, MA, Addison-Wesley, 1989.

Institute of Electrical and Electronics Engineers. IEEE Standard 730–IEEE Standard for Quality Assurance Plans. New York, Institute of Electrical and Electronics Engineers, 1989.

Institute of Electrical and Electronics Engineers. IEEE Standard 829–IEEE Recommended Practice for Software Requirements Specifications. New York, Institute of Electrical and Electronics Engineers, 1998.

Institute of Electrical and Electronics Engineers. IEEE Standard 830–IEEE Standard for Quality Assurance Plans. New York, Institute of Electrical and Electronics Engineers, 1989.

Institute of Electrical and Electronics Engineers. IEEE Standard 1012–IEEE Standard for Software Verification and Validation. New York, Institute of Electrical and Electronics Engineers, 1986.

Institute of Electrical and Electronics Engineers. IEEE Standard 1016–IEEE Recommended Practice for Software Design Descriptions. New York, Institute of Electrical and Electronics Engineers, 1986.

Institute of Electrical and Electronics Engineers. IEEE Standard 1059–IEEE Guide for Software Verification and Validation Plans. New York, Institute of Electrical and Electronics Engineers, 1993.

Kan SH. Metrics and Models in Software Quality Engineering. Reading, MA, Addison–Wesley, 1995.

Leveson NG. Safeware. Reading, MA, Addison-Wesley, 1995.

McConnell S. Code Complete. Redmond, WA, Microsoft Press, 1993.

McConnell S. Rapid Development. Redmond, WA, Microsoft Press, 1996.

Page-Jones M. The Practical Guide to Structured Systems Design—2nd Edition. Englewood Cliffs, NJ, Prentice Hall, 1988.
Pressman R. Software Engineering. New York, McGraw-Hill, 1987.
Putnam LH, Myers W. Measures for Excellence. Englewood Cliffs, NJ, Prentice Hall, 1992.
Rakos JJ. Software Project Management for Small to Medium Sized Projects. Englewood Cliffs, NJ, Prentice Hall, 1990.
Rumbaugh J et al. Object-Oriented Modeling and Design. Englewood Cliffs, NJ, Prentice Hall, 1991.
Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Manual. Reading, MA, Addison-Wesley, 1998.
Sommerville I, Sawyer P. Requirements Engineering. Chichester, England, Wiley, 1997.
Storey N. Safety-Critical Computer Systems. Harlow, England, Addison-Wesley, 1996.
Thayer RH, Dorfman M. Software Requirements Engineering—2nd Edition. Los Alamitos, California, IEEE Computer Society Press, 1997.
Yourdon E. Modern Structured Analysis. Englewood Cliffs, New Jersey, Yourden Press, 1989.