بررسی انواع تست و ابزارهاي تست خودکار نرم افزار

گردآوري و تنظيم : مارال سليقه

پیشگفتار

پیچیدگی و اندازه ي افزارهاي کامپیوتري همواره رو به افزایش است. هر محصول نرم افزاري مخاطبان خاصی د ارد. براي مثال یک نرم افزاربانکی مخاطبانی کاملاً متفاوت از مخاطبان یک نرم افزار بازي کامپیوتري دارد . بنابراین، زمانیکه سازمانی یک محصول نرم افزاري را می‌نویسد یا خریداري می کند، منطقاً باید اطمینان حاصل کند که آ یا محصول نرم افزاري براي کاربران او، مخاطبان او، خریداران، و سایر ذینفعان قابل پذیرش هست یا نه . تست نرم افزار فرایند تلاش براي این گونه ارزیابی هاست. به عبارت دیگر تست نرم افزار فرا یند امتحان یک برنامه کاربردي براي کشف خطاها و تضمین اینکه نیازمند هایي موجود را برآورده می‌کند و با سخت‌افزار مشتري سازگار است.

امروزه سازمان هاي نرم افزاري زمان و منابع زیادي را در تحل یل و تست نرم افزار صرف می کنند. از نظر مهندسان نرم افزار نوشتن کدهاي تست، به خودي خود، مثل توسعه خود محصول وقت گیر و گران است. بنابراین راه حلی براي خودکار کردن تست نرم افزار غیرقابل اجتناب است. خودکارسازي تست‌ها نسبت به تست دستی برتر هاي زیادي دارد . در جامعه امروزي پروژه‌هاي نرم افزاري پیچیده هستند و برا ي حل مسائل پیچیده طراح ی می‌شوند. خودکارسازي تست‌ها باعث می شود که توسعه‌دهندگان زمان بیشتري براي تمرکز بررو ي د یگر جنبه ها داشته باشند و بتوانند خطاهاي نرم افزار را به صورت مؤثرتري رفع نمایند. علاوه براین، ازآنجایی که تست‌ها را می‌توان در هر زمان و به هر تعداد دفعاتی اجرا کرد، توان از تست هاي قبلی استفاده ي مجدد نمود. و به این ترتیب کارایی تست را افزایش و زمان تست را کاهش داد. خودکارسازي تست نرم افزار زحمت و پیچیدگی انجام تست را کاهش می‌دهد.

در این متن پس از ارائه ي مقدماتی در مورد تست نرم افزار و اهمیت خودکارسازي تست نرم افزار، به معرفی و بررسی ویژگی‌هاي ابزارهاي متداول تست نرم افزار پرداخته شده است.

فهرست مطالب

فهرست شکل‌ها

1- مقدمه‌اي بر فرايند تست نرم افزار

تست نرم افزار فرايندي است كه كيفيت نرم افزار كامپيوتري را مشخص مـی‌كنـد. تـست نرم افزار شـامل فرايند اجراي يک برنامه با هدف يافتن باگ هاي افزاري است، اما محدود به آن نيـست.كيفيـت مطلـق نيست، بلكه براي افراد مختلف نسبی است . با اين تصور، تست كردن هرگز نمی‌تواند صحت نرم افزارهـاي كامپيوتري دلخواه را به طور كامل اثبات كند. يک نكته مهم اينست كه تست افزار، بايد از نقطه نظرات مختلفی از تضمين كيفيت نرم افزار لحاظ شود، كه با همه حوزه هاي فرايند تجاري همراه باشـد نـه فقـط حوزه هاي تست.

تست نرم افزار ممكن است به عنوان يک قسمت مهم از فرايند تضمين كيفيـت نرم افزار تلقـی شـود . در تضمين كيفيت نرم افزار متخصصان فرايند نرم افزار و ديدگاه وسيع‌تري روي نرم افزار و توسـعه آن دارنـد. آنها فرايند مهندسی نرم افزار را بررسی می كنند و آن را براي كاهش ميزان خطاهاي منجـر بـه شكـست ،

تغيير می‌دهند. عنصر تعيين كننده ميزان شكست قابل قبول، ماهيت افزار اسـت. بـازي ويـدئويی كـه براي شبيه سازي پرواز يک هواپيما طراحی شده،منطقاً بايد نسبت به‌افزاري كه براي كنترل يک خط پرواز واقعی به كار می‌رود ، تحمل شكست بيشتري داشته باشد.

شكست نرم افزار از طريق فرايندهاي زير رخ می‌دهد. يک برنامه نويس يک خطايی را انجام مـی‌دهـد كـه منجر به يک شكست در كد منبع نرم افزار می شود. اگر اين خطا كامپايل و اجرا شـود، در موقعيـت هـاي خاصـی سيستم نتايج نادرستی توليد می كند كه منجر به شكست می‌شودلزوماً. همه خطـا هـا منجـر بـه شكست نمی‌شوند. براي مثال خطاها در كدهايی كه برنامه هرگز به اجراي آنها نمی‌رسد[1]. هرگز منجر به شكست نخواهد شد . يک خطا زمانی منجر به شكست می شود، كه محيط تغيير مـی‌كنـد. مثـالی از ايـن

---

1 dead code

تغییرات در محیط شامل نرم افزارهایی هستند که در یک پلتفرم جدید سخت افزاري یـا نرم افزاري، یـا تغییرات در دادههاي منبع، یا تعاملات با افزارهاي متفاوت اجرا میشوند.[Kan01-1]

روشهاي مختلفی براي تست افزار وجود دارد . دوباره بررسی کردن و امتحانات سخت و مهم ، تستهاي استاتیک نامیده میشوند، در حالیکه اجراي واقعی برنامه با یک مجموعه تست داده شده در مرحله خاصی از فرایند توسعه، تست پویا نامیده میشود.

اهداف تست

"گلن مایزر" درباره افزار چند قاعده را بیان می کند که به خوبی به عنوان اهداف تست عمل می کند.

تست فرآیند اجراي برنامه به قصد یافتن خطاهاست.

مورد تست خوب، موردي است که احتمال یافتن خطاهاي کشف شده در آن، بالا باشد. **تست موفق،** تستی است که خطاهاي کشف نشده را کشف می کند.

اهداف بالا نشانگر یک تغییر دیدگاه زیبا هستند، و برخ لاف این دیدگاه عامیانه که تست موفق، تستي

است که در آن خطایی یافته نشود . اگر تست با موفقیت اجرا شود (مطابق اهداف ذکر شده در بالا)، خطاهاي افزار را بر ملا خواهد نمود. به عنوان مزیت دوم، تست نشان می دهد که عملکردهاي افزار ظاهرا" مطابق مشخصه کار می کنند، و خواسته هاي رفتاري و کارایی ظاهرا" برآورده شده اند. به علاوه، داده هاي جمع آوري شده به موازات انجام تست، شاخص خوبی از قابلیت اطمینان افزار و شاخصی از کلیت کیفیت نرم افزار به دست می دهند . ولی تست نمی تواند نبود خطاها و نقایص را ثابت کند . بلکه فقط می تواند نشان دهد که خطاها و نقایص وجود دارند.

اصول تست

مهندس نرم افزار پیش از اعمال روش ها در خصوص موارد تست مؤثر، باید اصول پایه اي را که تست نرم افزار را هدایت می کنند، درك کند . "دیویس" مجموعه اي از اصول پیشنهاد می کند، که در اینجا از آنها استفاده خواهیم کرد:

همه تست ها باید تا حد خواسته هاي مشتري قابل ردیابی باشند. چنانکه دیدیم، هدف تست افزار، کشف خطاها است. یعنی اکثر نقایص شدید (از دیدگاه مشتري) آنهایی هستند که باعث می شوند برنامه نتواند خواسته هاي خود را برآورده کند.

تست باید مدت ها قبل از شروع تست، طرح ریزي شود. طرح ریزي تست می تواند به محض کامل شدن مدل خواسته ها آغاز شود. تعریف مشروح موارد تست می تواند به محض منسجم شدن مدل طراحی آغاز شود. بنابراین، همه تست ها را می توان پیش از تولید هر گونه کد، برنامه ریزي و طراحی کرد.

اصل "پارتو" در تست نرم افزار صدق می کند. به عبارت ساده، اصل "پارتو" بیان می کند که 80 درصد همه خطاهاي کشف شده طی تست، احتمالا" در 20 درصد همه مؤلفه ها برنامه قابل کشف هستند. مسئله، جدا سازي مؤلفه هاي مظنون و آزمودن کامل آنهاست.

تست باید در ابعاد کوچک آغاز شود و به ابعاد بزرگتر گسترش یابد. اولین تست ها برروي هر یک از مؤلفه ها انجام می شوند. با پیشرفت تست، خطاهاي مجموعه اي از مؤلفه هاي مجتمع و سپس کل سیستم یافت می شود.

تست کامل امکان پذیر نیست. تعداد مسیرهاي ممکن براي برنامه متوسط نیز زیاد است. لذا، اجراي هر ترکیبی از مسیرها امکان پذیر نیست. ولی این امکان وجود دارد که برنامه را در حد کفایت پوشش دهیم.

براي آنکه تست بیشترین بازدهی را داشته باشد، باید توسط یک شخص ثالث بی طرف انجام شود. منظور از بیشترین بازدهی آن است که خطاها را با احتمال بیشتري بیابد. به دلایلی که قبلا" در همین فصل ذکر شد، مهندس افزاري که سیستم را ایجاد کرده است، بهترین کسی نیست که باید همه تست ها را انجام دهد.

2- انواع، روش‌ها و سطوح تست

انواع تست

تست‌هاي متنوعی را می توان بر روي سيستم ها اعمال نمود تا آنان را از جنبه هاي مختلف مورد تست و تست قرار داد. در اين بخش به اين تست ها اشاره می شود وهريک را مختصرا توضيح می دهيم.

تست عملکرد

در اين نوع تست ، نرم افزار تست ميشود تا از لحاظ درستی عملکرد بررسی شود . تستها نوشته ميشوند تا ببينند که ايا نرم افزار همان گونه که انتظار ميرود عمل ميکنداگر. چه تست عملکرد معمولاً در انتهاي فرايند توسعه انجام ميشود ولی ميتواند –و بايد – سريعتر اغاز شود . کامپوننت ها و فرايند هاي مجزا ميتوانند خيلی سريعتر انجام شوند ، حتی زودتر از اينکه بتوان تست عملکرد را روي سيستم انجام داد .

تست استرس

برنامه در مقابل بار سنگين مثل مقادير عددي پيچيده ، مقادير زيادي ورودي و مقادير زيادي پرس و جو2 امتحان ميشود . که ميزان باري که برنامه ميتواند ان را تحمل کند را بررسی ميکند . هدف طراحی محيطی است که مخرب تر از محيطی که برنامه در دنياي واقعی و در شرايط نرمال با ان روبرو ميشود .اين مسئله سخت ترين و پيچيده ترين مقوله ا ز تست است که بايد انجام شود و احتياج به تلاش توأم همه تيم هاي برنامه نويسی دارد . يک محيط تست با چندين ايستگاه تست ايجاد ميشود و در هر ايستگاه يک اسکريپت سيستم را تست ميکند .اين اسکريپتها رو به رشد هستند و ايستگاه ها نيز بتدريج افزايش يابند و همه بصورت همزمان روي سيستم فشار اورند تا سيستم متوقف شود .سيستم را

[2] query

تعمير ميكنند و تست استرس دوباره انجام ميشود تا سيستم به حدي از استرس برسد كـه بـالاتر از حـد مورد انتظار مشتري باشد . شرايط رقابت 3 و رخنه هاي حافظه 4 در تست استرس پيدا ميشوند . شرايط رقابت تعارضی بين حداقل دو ايستگاه تست است و هر تست زمانيكه به تنهايی كار كند به درسـتی كـار ميكند .اما زمانيكه دو تست به صورت موازي كار كنند ، يكی يا هر دو تـست شكـست ميخورنـد . و ايـن معمولاً به خاطر يک قفل است كه به درستی مديريت نشده است . يک رخنه حافظه زمانی رخ ميدهد كه يک تست ، حافظه تخصيص يافته را رها ميكند و به درستی حافظه را بر نميگرداند . به نظر تست درست كار ميكند ، اما بعد از مدتی اجراي تست حافظـه در دسـترس كـاهش پيـدا ميكنـد و سيـستم شكـست ميخورد .

تست بار

برنامه در مقابل بار زياد يا ورودي ها تست ميشود مثل تست وب سايتها براي يافتن نقاطی كه در ان نقاط وب سايت يا برنامه شكست ميخورد و يا نقاطی كه در انها كارايی وب سايت كا هش پيدا ميكند . تـست بار در سطح بار از پيش تعيين شده اي انجام ميشود،**Error! Reference source not found.**معمـولاً بالاترين باري كه سيستم ميتواند بپذيرد، د ر حاليكه هنوز به درستی كار ميكند . توجه كنيد كه تست بار نمی‌خواهد سيستم را با فشار اوردن از پا در اورد . اما ميخواهد سيستم را بطور پيوسته زير بار نگـه دارد .در مفهوم تست بار ، بايد مجموعه داده هاي زيادي براي تست در دسترس داشـته باشـيم . باگهـا ظهـور نميكنند مگر اينكه شما بـا موجوديتهـا ي خيلـی بـزرگ مثـل هزاران كـاربر در انبـار داده هـايی مثـل LDAP,NIS , Active directory كار كنيد و....تست كنند گان به ابزار هاي اتومـات بـراي ايجـاد ايـن مجموعه داده هاي بزرگ دارند .اما خوشبختنه با هر زبان اسكريپت نويسی ميتوان اين كار را انجام داد .

[3] Race condition
[4] memory leaks

تست اکتشافی

مشابه تست بالبداهه است و براي يادگيري و كنكاش نرم افزار صورت ميگيرد . ويک روش قوي و جالب براي تست است . در بعضی مواقع ممكنه تا حدودي قوي تر از زبان‌هاي اسكريپت نويسی براي تست باشد .

تست رگرسيون

بعد از تغيير نرم افزار ، جه براي تغيير در عملكرد يا براي تصحيح يک خطا ، يک تست رگراسيون تمام تستهايی كه قبلاً نرم افزار انها را با موفقيت انجام داده را دوباره اجرا ميكند تا اطمينان حاصل كند كه نرم افزارتصادفاً در عملكرد هاي قبلی دچار خطا نشده است .تست رگراسيون ميتواند در هيچ يا همه سطوح قبلی صورت بگيرداين. تستهاي رگراسيون اكثراً اتومات شده اند .انواع خاص تري از تستهاي رگراسيون با عنوان تست سلامت 5 شناخته شده اند و زمانيكه ما خواهيم سريعاً رفتار عجيب سيستم

را بررسی كنيم . و يا تست دود 6 زمانيكه عملكرد هاي ابتدايی چک ميشوند .

تست رگراسيون نوعی تست كه تمركز روي تست مجدد بعد از اعمال تغييرات است . در تستهاي رگراسيون قديمی، يک تست دوباره تكرار شد . اما در تستهاي رگراسيون ريسک گرا ، ما نواحی قبلی را مثل قبل تست ميكنيم اما از تستهاي متفاوتی كه به تدريج پيچيده تر ميشوند استفاده ميكنيم .تستهاي رگراسيون قديمی‌معمولاً تا حدودي اتومات بودند .

تست رگراسيون ميخواهد دو ريسک را كاهش دهد :

تغييري كه ميبايست يک خطا را ازبين ميبرد ، شكست ميخورد .

بعضی تغييرات اثر جانبی دارند ، اگر تصحيح نكنيد خطاي قبلی باقی ماند و اگر تصحيح كنيد خطاي جديد ايجاد ميشود .

---

[5] sanity testing
[6] smoke testing

تست قابلیت استفاده

این نوع تست تنها در مواردي انجام ميشود كه رابط كاربر 7 براي برنامه خیلی مهم باشد و بايد براي هر كاربر، خاص باشد . تست قابلیت استفاده ، فرایند کار کردن مستقیم یا غیر مستقیم با کاربران نهایی براي شناخت اينكه چگونه هر كاربر يک بسته نرم افزاري را احساس ميكند و چگونه با ان تعامل ميكند .این فرایند نقاط قوت و ضعف برنامه را براي كاربران كشف ميكند . بنابر این هدف این تست، شناسایی مشکلات مربوط به طرز استفاده سیستم از دید کاربران می‌باشد . در انجام این تست فاکتورهاي انسان ی كه عمدتا subjective باشند، مورد توجه قرار می‌گیرند و بهم ین علت انجام این نوع تست معمولا كار پ يچيده اي است كه در بس ياري قسمتهاي آن امكان خودكارسازي تست موجود نمی‌باشد.[Mye04] [Mye79]

تست امنیت

طراحی و تست سیستم هاي نرم افزاري براي اطمينان از ايمن و مطمئن بودن سیستم، مسألهاي اساسی است كه توسعه دهندگان نرم افزار و متخصصان تست با آن مواجه هستنداخيراً. مسألهي ايمنی و مطمئن بودن به خاطر ازدياد برنامه هاي كاربردي تجاري براي استفاده روي اينترنت اهميت مازادي يافته است . اگر كاربران اينترنت اعتقاد داشته باشند كه اطلاعات شخصی آنها ايمن نيست و براي افراد غيرمجازي كه با استفاده از اينترنت آسيب می رسانند قابل دسترسی است، آيندهي تجارت الكترونيكی به مخاطره می‌افتد. تست امنیت ویژگی هایی از سیستم که به دردسترس بودن، یکپارچگی و محرمانه بودن داده ها و خدمات سیستم مرتبط است را ارزیابی می کند. كاربران/مشتريان بايد توسط اين موضوع ترغيب شوند كه نيازهاي امنيتی آنان در زمان تعيين نيازمنديها كاملاً مشخص است، و بنابراین مسائل امنیتی توسط طراحان و آزمایندگان مورد توجه قرار می‌گیرد. [BUR03]

[7] User interface

هدف تست امنیت بررسی کارایی مکانیزم هاي دفاعی سیستم وب در مقابل دسترسی‌هاي نامطلوب کاربران بدون مجوز و حفظ منابع سیستم در مقابل کاربران ناشایست، و همچنین دادن دسترسی به کاربرانی که مجوز دارند، می باشد. آسیب پذیري هاي سیستم که بر روي امنیت سیستم تأثیر می گذارد ممکن است منشأ در کد برنامه داشته باشند یا در کامپوننت هاي سخت‌افزاري، نرم افزاري یا میان افزاري سیستم. هردوي محیط اجرا و همچنین برنامه ممکن است در نقص‌هاي امنیتی دخیل باشند. در مورد نرم افزار هاي مبتنی بر وب پیاده سازي‌ها و تکنولوژي‌هاي اجرایی ناهمگن همراه با تعداد بسیار زیاد کاربران و همچنین امکان دسترسی آنان از هر جایی، این برنامه ها را از برنامه‌هاي کاربردي معمولی آسیب‌پذیرتر تست امنیت آنها را سخت‌تر می‌سازد.

شش مفهوم اساسی امنیت که باید در تست امنیت پوشش داده شود به این شرحاند: محرمانه بودن، جامعیت، تصدیق هویت[8]، مجوز دادن[9]، در دسترس بودن و عدم انکار.

تست پوشش

تست پوشش به دو دسته کلی پوشش عبارات [10] و پوشش انشعابات [11] می توان تقسیم نمود. در تست پوشش عبارات ، کد طوري اجرا میشود که هر عبارتی از برنامه حداقل یکبار اجرا شود و این باعث میشود بفهمیم همه جملات بدون اثر جانبی اجرا میشوند.

از آنجائیکه هیچ برنامه نرم افزاري نمیتواند در مد پیوسته اي از کد اجرا شود، در بعضی از نقاط نیاز است که براي یک عملکرد خاص به نقطه اي خارج از کد انشعاب کنیم. تست پوشاندن انشعابات کمک میکند که همه انشعابات در کد را ارزیابی کنیم و اطمینان حاصل کنیم که هیچ انشعابی در کد منجر به رفتار غیر نرمال در برنامه کاربردي نمیشود.

---

[8] Authentication
[9] Authorization
[10] statement coverage
[11] Branch Coverage

روش‌هاي تست

طراحی تست‌هایی براي نرم افزار و هر محصول مهندسی دیگر، می تواند به اندازه طراحی خود محصول اولیه دشوار باشد . با این حال به دلایلی که پیش از این بحث شد، مهندسان نرم افزار غالبا" با تست به عنوان موارد تست در حال توسعه اي رفتار می کنند، که ممکن است درست به نظر آیند، ولی از کامل بودن آنها چندان اطمینانی نباشد . با بخاطر داشتن اهداف تست، باید تست هایی را طراحی کنیم که احتمال یافتن خطاها در حداقل زمان، بیشتر باشد . هر محصول مهندسی (و اکثر چیزهاي دیگر) را می توان به یکی از دو روش آزمایش کرد:

با دانستن عملکرد خاصی که نرم افزار براي انجام آن طراحی شده است، می توان تست هایی طراحی کرد که نشان می دهند هر عملکرد به طور کامل درست است، و در عین حال، در هر عملکرد به دنبال یافتن خطاها هستند.

با دانستن طرز کار داخلی محصول، می توان تست هایی ترتیب داد که اطمینان دهند همه چیز جفت و جور است . یعنی، عملیات داخلی طبق مشخصه اجرا می شوند و با همه مؤلفه هاي داخلی به طور مناسب تمرین شده است.

روش اول را تست جعبه سیاه و دومی را تست جعبه سفید می نامند. در سالهاي اخیر یک روش بینابینی هم مورد توجه قرار گر فته است که در مواقعی که دسترسی به برخی از مولفه هاي داخلی نرم افزار ها داریم مورد استفاده قرار می گیرد . یه این روس تست جعبه خاکستري می گویند. در این بخش این روشها را به تفصیل شرح می دهیم.

تست جعبه سیاه

این نوع از تست با نرم افزار به عنوان یک جعبه سیاه برخ ورد میکنند که هیچ درکی از رفتار داخلی ان وجود ندارد . هدف ان تست کردن عملکرد نرم افزار مطابق با نیازمندیهاست ، تا ببینند که تا چه حد

نیازمندیهاي ذکر شده براورده میشوند .بنابراین طراح تست داده ها را وارد میکند و خروجی را از شی؛ تست بیند.براي این سطح از اجر اي تست ، نیاز است تا طراح تست ، موارد تست طراحی شده را به تست کننده ،کسی که باید بررسی کند که ایا براي یک ورودي خاص ، مقدار خروجی یا رفتار سیستم ، منطبق بر انچه مورد انتظار است و در مورد تست ذکر شده هست یا نه .براي طراحی موارد تست یک دیدگاه خارجی از سیستم میگیرد و این نوع تستها میتوانند عملکردي 12 یا غیر عملکردي13 باشند که معمولاً عملکردي عمل میکنند . طراحان تست ورودیهاي معتبر و غیر معتبر را انتخاب میکنند و خروجی درست را انتخاب میکنند . و هیچ دانشی از ساختار داخلی شیء در دسترس نیست.[PRE05]

این روش از طراحی تست در همه مراحل از تست نرم افزار قابل استفاده است . تست واحد و تست یکپارچگی و تست عملکرد و تست سیستم و تست پذیرش .بالاترین سطح و بالطبع بزرگترین و پیچیده ترین جعبه باید بصورت جعبه سیاه صورت گیرد تا ساده تر شود .تست جعبه سیاه که تست رفتاري نیز نامیده میشود روي نیازمندیهاي عملکردي نرم افزار تمرکز میکند.تست جعبه سیاه است که مهندسان نرم افزارا قادر میکند تا مجموعه اي از شرایط ورودي را مشتق کنند که تا کاملاً همه نیازمندیهاي عملکردي برنامه را مورد بررسی قرار دهند .تست جعبه سیا ه یک جایگزین براي تست جعبه سفید نیست بلکه یک روش مکمل که یک کلاس متفاوتی از خطا ها را نسبت به روشهاي جعبه سفید پوشش میدهد.

تست جعبه سفید

زمانی ممکن است که تست کننده به ساختار داده هاي داخلی و کد و الگوریتم ها دسترسی دارد.روشهاي تست جعبه سفید شامل ایجاد تستهایی است که بعضی از معیارهاي پوشش کد14 را براورده میکنند . براي مثال طراح تست میتواند تستهایی را طراحی کند که باعث شود همه عبارات برنامه حاقل یکبار اجرا

[12] functional
[13] non functional
[14] code coverage

شوند.ساير مثالهاي تست جعبه سفيد عبارتند از تست جهش 15 و روشهاي تزريق خطا 16 . تستهاي جعبه سفيد ، همه تستهاي استاتيک را شامل ميشوند.[PRE05]

روشهاي تست جعبه سفيد ميتوانند براي ارزيابی کامل بودن يک مجموعه تست - که با روشهاي تست جعبه سياه ايجاد شده اند -نيز به کار رود .اين به تيم نرم افزاري اجازه ميدهد که قسمتهايی از سيستم را ارزيابی کنند که کمتر تست شده اند و اطمينان حاصل شود که نقاط عملکرد خيلی مهم تست شده اند.دو فرم معمول از پوشش کد عبارتند از ، پوشش تابع ، که روي توابع اجرا شده گزارش ميدهد و پوشش جملات ، که روي تعداد خطوط اجرا شده براي تکميل تست گزارش ميدهد.از اين دو معيار پوشايی حاصل می شود. تست جعبه سفيد همچنين ممکن است شامل مهندسی معکوس براي مشخص کردن مثلاً مقادير مرزي به کار رود.

تست جعبه سفيد مزايايی دارد از جمله : چون لازمه ان دانستن ساختار داخلی کد است فهميدن اينکه چه نوع از داده هاي ورودي و خروجی ب راي تست کاراتر برنامه مناسب است، آسانتر می‌شود. مزيت ديگر تست جعبه سفيد آنست که به بهينه سازي کد نيز کمک می کند. به پاك کردن خطوط اضافی کد نيز کمک می‌کند. اين خطوط اضافی ممکن است منجربه خطاهاي پنهان شوند.

معايب تست جعبه سفيد شامل موارد زير می شوند: چون اگاهی و مهارت از کد و ساختار داخلی نياز است، يک تست کننده ماهر نياز است تا اين نوع تست را انجام دهد که باعث افزايش هزينه می شود . بررسی هر قطعه از کد و بدنبال خطاهاي پنهان گشتن تقريباً غير ممکن است و همچنين ممکن است منجربه مشکلاتی شود که باعث شکست برنامه می‌شوند.

[15] mutation testing
[16] fault injection

تست جعبه خاکستري

در سال هاي اخ ير عبارت تست جعبه خاکستر ي نيز به استفاده معمول اضافه شده است. کـه بـه معنـا ي داشتن دسترسی به ساختمان داده ها ي داخل ی و الگوريتمها براي اينکه بتـوانيم مـوارد تـست را طراحـی کنيم و در سمت کاربر يا در سطح تست جعبه سياه استفاده کنيم.

تست جعبه خاکستري‌مخصوصاً در مواقع ی که م يخواهيم تست رگرسيون را بين دو ماژول از کد کـه بـه وسيله دو برنامه نويس مختلف نوشته شده – و فقط ا ينترفيس ها براي تست در دسترس هستند -به کـار ميرود .

تست سيستمهاي مبتنی بروب

ويژگيهاي خاص سيستم هاي تحت وب بطور مستقيم بر موضوع تست اين سيستم ها تـاثير مـی گذارنـد. نتيجه اين ويژگيها و پيچيدگيها آن است که روشها و ابزارها و مـدلهاي رايـج بـراي تـست نرم افزارهـاي متداول، معمولا براي تست سيستم هاي تحت وب، کافی ن می‌باشند. برخی از اين روشها نيازمند تغييـر و تطبيق با محيط وب باشند و برخی نيز بکلی قابل استفاده ن می‌باشند. همچنين بـراي تـست برخـی از موارد، نيازمند روشها و مدلهاي جديد که مخصوص سيستم هاي تحت باشند، هستيم.

با توجه به اينکه در توسعه سيستمهاي تحت وب، معمولاً از مدل توسعه سريع (RAD) استفاده شـود، فرصت کمتري براي تست سيستم در اختيار توسعه دهندگان باشد. همچنين برخی موارد نظير تـست مقياس پذيري، ممکن است بطور دقيق قابل اجرا نباشند يا هزينه اجراي آنها زياد باشد. در نتيجه به نظر رسد که براي برخی انواع تست، لازم است که سيستم ابتدا بطور کامل زيـر بـار بـرود و در تعامـل بـا کاربران واقعی که رفتارهايشان لزوما قابل پيش بينی نيست، مورد تست قرار گيرد.

با توجه به حساسيت زياد نسبت به زمان توسعه اين سيستم ها، همچنين بـا توجـه بـه اينکـه ترکيبـات متعددي از مرورگرها، سيستم عاملها، و محيطهاي اجرا وجود دارد، اگر بخواهيم يک سيـستم را در برابـر تمام اين ترکيبات مورد تست قرار دهيم، نياز به خودکارسازي روالهاي تست به شکل چشمگيري، افزايش

يابدهمچنين. با توجه به اينكه سيستم هاي تحت وب دائماً در حال تغيير و بروزرسانی می‌باشند، قابليت اجراي مجدد تست ها بطور خودكار، بسيار مورد نياز باشد. بدين ترتيب توان پس از انجام حجم مناسبی از تغييرات بر روي سيستم، دوباره تست ها را بطور خودكار بر روي سيستم انجام داد . در مورد سيستم هاي متداول، معمولاً پس از ارائه سيستم به بازار و مشتريان، تغييرات سيستم خيلی محدود اتفاق افتد و در فواصل زمانی متفاوت، نسخه هاي جديد نرم افزار يا وصله هاي امنيتی آن، توزيع

شوند د ر نتيجه نرخ تغييرات بسيار پايين است . اما در سيستم هاي تحت وب، چون سيستم بر روي سرور نصب شود و از طريق شبكه قابل دسترس است، اين امكان وجود دارد كه طراحان سيستم، براحتی و بطور مداوم سيستم را بروزرسانی نمايند و تغييرات لازم را اعمال نمايند.

يكی از مسائل مهم در تست سيستم هاي تحت وب آن است كه هنوز معيارهاي دقيق و قابل اعتماد و مورد توافق، براي تست اين سيستم ها، معرفی نشده است.

سيستم هاي تحت وب، در طول زمان بسيار تغيير كرده اند . سيستم هاي اوليه، شامل صفحات ايستاي HTML بودند . اما سيستمهاي امروزي، شامل صفحات پويا با تركيبی از تكنولوژي هاي متفاوت، نظير PHP، ASP، JSP، XML و JDBC باشند. مواردي نظير قالب‌هاي چندرسانه‌اي جديد، و يا تكنولوژي AJAX، افزايش اجزاي مختلفی كه در طراحی سيستم دخيل هستند، همه و همه به پيچيده و پوياتر شدن اين سيستم ها منجر شده اند.

هدف اصلی تست يک سيستم تحت وب، اجراي آن سيستم به منظور كشف خرابی ها و خطاهاي آن باشد. خرابی (failure)، به يک ناتوانی مشخص سيستم در اجراي يک وظيفه از پيش تعيين شده بر اساس معيارهاي تعريف شده، باشد. برخی خرابی ها مربوط به خطاهاي عامل انسانی در طراحی و پياده ساز ي سيستم باشند. اما برخی خرابی ها نيز مربوط به محيط اجرا، مثلا باگهاي نرم افزار مرورگر يا مشكلات شبكه، مربوط شوند و منشا مشكل، طراحان سيستم نمی‌باشند. در نتيجه براي تشخيص

انواع مختلف خرابی ها، انواع متفاوت تست مورد نياز باشد.

بطور اساسی، محیط اجرا، بر نیازمندیهاي غیر عملیاتی سیستم، نظیر مقیاس پذیري، ثبات، و سازگاري سیستم، تاثیر گذارد و نه بر نیازمندیهاي عملیاتی سیستم، یعنی تطبیق بین آنچه سیستم انجام دهد و آنچه باید انجام دهددر. نتیجه معمولاً سیستم مسئول ارضاي نیازهاي عملیاتی، و محیط اجرا مسئول ارضاي نیازهاي غیرعملیاتی باشد، البته تفکیک کامل بین این دو مورد وجود ندارد. اما توان گفت که تست سیستم هاي تحت وب را توان از دو دیدگاه بررسی نمود:

1. از دیدگاه اول، نیازمندیهاي عملیاتی سیستم مورد تست قرار گیرند. یعنی عملکرد سیستم در مقایسه با آنچه که باید انجام دهد، یعنی منطق سیستم و وظایف تعریف شده در مستندات، مورد تست قرار گیرد.
2. از دیدگاه دوم، نیازمندیهاي غیرعملیاتی سیستم مورد ارزیابی قرار گیرند.

نکته مهم آن است که هر دو نوع تست باید بطور مناسب اجرا شوند و نمی‌توان یکی از این دو مورد را جایگزین دیگري نمود. **Error! Reference source not found.**

سطوح مختلف تست

تست نرم افزار در فازهاي متفاوت و در سطوح مختلف انجام می پذیرد. این سطوح عبارتند از:

تست واحد

یکی از مراحل اولیه تست یک سیستم، تست واحد[17]، باشد که هر یک از واحدها یا ماژولهاي تشکیل دهنده یک برنامه را بطور مستقل، مورد تست قرار دهد. معمولا تست واحد توسط خود برنامه نویسان و به موازات توسعه سیستم انجام شود. یعنی هر برنامه نویسی که ماژولی را براي سیستم می‌نویسد، خود، وظیفه تست آن ماژول را نیز بعهده دارد و نیازي نیست تست آن ماژول به پس از تکمیل سیستم موکول شود.

[17] Unit testing

هدف از انجام تست واحد، اطمینان از درستی عملکرد واحدها یی است که پس از توسعه، در قسمتها ي مختلف سیستم مورد استفاده قرار خواهند گرفت.

تست واحد، معمولا جزء تستهاي جعبه سفید به حساب آورده شود که نیاز به دسترسی به ساختار درونی کد مورد تست دارد.

لازم به ذکر است که علیرغم اهمیت بسیار زیاد تست واحد در فرآیند تضمین کیفیت افزار، این نوع تست نمی‌تواند جایگزین دیگر انواع تست شود . بعنوان مثال، با استفاده از تست واحد نمی‌توان کیفیت رابط گرافیکی سیستم را ارزیابی نمود یا تست بار را نمی توان با استفاده از تست واحد انجام داد [Ham04].

تست یکپارچگی

هدف از تست یکپارچگی سیستم[18]، آن است که مطمئن شویم اجزاي مختلف سیستم، در کنار یکدیگر، بخوبی کار می کنند و تعاملات، ارتباطات و رد و بدل کردن داده ها در بین ماژولهاي مختلف سیستم، بدرستی انجام می‌شود و در نتیجه، کل سیستم عملکرد صحیحی دارد.

تست یکپارچگی را می توان در سطوح متفاوتی انجام داد. مثلا، می‌توان هر یک از ماژولهاي اساسی سیستم را بعنوان یک سیستم در نظر گرفت (که خودش از اجزاي کوچکتري تشکیل شده) و تست یکپارچگی را در مورد آن انجام داد . همچنین می‌توان کل سیستم را بعنوان یک سیستم واحد در نظر گرفته و آن را مورد تست قرار داد.

نکته قابل توجه آن است که نباید اینطور تصور شود که انجام تست واحد بر روي ماژولهاي سیستم، ما را از انجام تست یکپارچگی بی نیاز می‌کند. در واقع هر دو نوع تست مذکور لازم می‌باشند و هر یک توانایی خاص خود را دارند . نقطه مورد توجه تست یکپارچگی، نقاط تماس و تعامل ماژولها با یکدیگر است و

[18] Integrity testing

ماژولها را در کنار یکدیگر و در ضمن کار با هم، مورد تست قرار می‌دهد، در حالیکه تست واحد، ماژولها را بطور مستقل و جدا از بقیه اجزاي سیستم مدنظر قرار می‌دهد. [Cra02] [Chu05]

تست سیستم

کـل سیستم کاملاً یکپارچه را تست کند تا بررسی کند که ایا تمام نیازمندیها براورده شوند یـا نـه.

قبل از عرضه نسخه نهایی یک افزار ، تستهاي الفا و بتا نیز علاوه بر تستهاي فوق انجام شوند.

تست عملکرد شبیه سازي شده یا واقعی با مشتریان یا کاربران پتانسیل ، یا یک تـیم تـست مـستقل در سایت برنامه نویسان‌تست. الفا معمولاً براي افزار هاي تولید انبوه به عنوان نوعی از تست پذیرش بکـار برده شود . و قبل از مرحله تست بتا صورت پذیرد .

نسخه هایی از نرم افزار ، که نسخه هاي بتا نامیده میشوند ، به مخاطبان محدودي در خارج از تیم برنامه نویسان عرضه شود . افزار به گروهی از افراد عرضه میشود تا تستهاي بیشتري انجام شود و اطمینان حاصل کنیم که افزار خطا ها یا باگهاي کمی‌دارد. کاهی اوقات نسخه هاي بتا به عموم عرضه میشود تا میزان بازخورد ها افزایش یابد .

تست پذیرش

این نوع تست بر اساس نیازمندیهاي مستند شده کاربران سیستم انجام شود و هدف از انجام آن کسب اطمینان از تامـ ین نیازهاي کاربران توسط سیستم، باشد. به بیان دیگر در این نوع تـست مـی‌خـواه یم مطمئن شو یم که سـ یستم تولـ ید شده از دید کاربران قابل قبول است یا خیر. بهمین علت بهتر است انجام تست توسط خود کاربران یا نمایندگان آنها و در محیط و شرایط واقعی صورت گیرد.[Cra02] [Chu05]

در نهایت ، تست پذیرش توسط کاربران نهایی یا مشتریان انجام میشود تا پذیرش محصول صورت بگیـرد یا نه . تست پذیرش ممکن است به عنوان بخشی از فرایند ، زمانیکه از یک فاز توسعه به فاز دیگر میرویم نیز صورت گیرد .

3- خودکارسازي تست

امروزه سازمان هاي نرم افزاري زمان و منابع زيادي را در تحل يل و تست نرم افزار صرف مـ كنندى. از نظـر مهندسان نرم افزار نوشتن كدهاي تست، به خودي خود، مثل توسعه خود محصول وقت گير و گران است. تست نرم افزار فرايند امتحان يک برنامه كاربردي براي كشف خطاها و تضمين اينكه نيازمند هاي موجود را برآورده مـ ىكند و با سخت‌افزار مشتري سازگار است.

خودكارسازي تست‌ها نسبت به تست دستـ ى برتـر هـاي زيـ ادي دارد. در جامعـه امـروزي پـروژه‌هـاي افزاري پيچيده هس تند و برا ي حل مسائل پيچيده طراحى مـ ىشوند. سازندگان ابزارهاي تست افـزار اغلب نياز به زمان دارند تا درباره يک مساله خاص آگاه ى پيدا كنند و با تكنولوژي مربـوط بـه آن مـساله آشنا شوند . بنابراين برا ي رس يدن به ضرب الاجل تعيين شده پروژه ، تـيم تـست بايـد يـک ابـزار تـست خودكار را ايجاد نما يد كه به عنوان مكمل ى برا ي فرآيند تست موجود عمل نمايد . با وجود اينكـه ممكـن است هز ينه اول يه اجرا ي آن در شروع كار سنگ ين باشد اما در طـى فرآينـد توسـعه ، ايـن هزينـه جبـران خواهد شد . خودكارسازي تست‌ها باعث مـ ى شود كه توسعه دهندگان زمان بيشتري براي تمركـز بـررو ي ديگر جنبه ها داشته باشند و بتوانند خطاها ي نرم افزار را به صورت مؤثرتري رفـع نماينـد . عـلاوه بـراين، ازآنجايى كه تست‌ها را مـ توان‌ى در هر زمان و به هر تعداد دفعاتـ ى اجرا كرد، توسعه‌دهندگان قادر خواهند بود به سادگ ى‌خطا را مجدداً ايجاد كنند تا نقص موجود در كد نرم افزار را بيابنـد در حاليكـه در تـست

دستى اجرا ي مجدد خطا سخت مـ ى باشد ز يرا گاهى اوقات هنگام انجام تست دستى، فرد تست‌گـر تمـام عملياتى كه طى روال تست كردن انجام داده است را بخاطر ندارد.

همچنین باید اشاره کرد که بسیاري از تلاش‌ها در زمینه خودکارسازي تست‌ها به نتایج مورد انتظار دست نیافته‌اند. گاهی اوقات در زمـ ینه ایجاد و نـگهداري خودکارسازي تست سرماي گذاري عظیمی مـ ی‌شـود امـا پس از ساخت ، حتی هزینه سرماي گذاري شده قابل جبران نیست . بسیار مهم است که یـک تحلیل کامل در مورد هز ینه و منافع حاصل از خودکارساز ي تست دسـ تی مـورد نظـر ، انجـام شـود. اغلـب، موفقیـ ت هنگامی حاصل می شود که بر رو ي پیدا کردن قسمتهایی از نرم افزار که خودکارسازي آنهـا سـودمند بـه نظر می رسد متمرکز شویم و نه بر رو ي خودکارسازي کل نرم افزار. خودکارسازي تست می تواند هزینه و پیچیدگی زیادي را براي تیم تستگر به هـ مراه داشته باشد و یا در صـورتیکه توسـط افـراد مناسـب و در مواردي که انجام آن مورد تایید است انجام شود، مـ ی‌تواند کمک قابل توجهی را به این تیم ارائه دهد.

بهتراست زمانی به خودکارسازي تست بپردازیم که حداقل یکی از شرایط زیر در مورد پروژه مهیا باشد :

Test case ها و محیط هاي تست قابل استفاده مجدد باشند.

نیاز کمی به دانش محیطی در تست‌ها داشته باشیم .

سیستم‌ها استاندارد و مستقل باشند.

رمزگذاري و تدوین قوانین[19] اصلـ تری‌ین استاندارد در مدیریت دانش باشد.

باید توجه داشت که تست خودکار به این معنا ن یست که کل فرآیند تست نرم افزار بـه صـورت خودکـار انجام مـ ی‌شود. بلکه به معنا ي تست نرم افزار با کمک کامپ یوتر است. به طـور خلاصـه تـست خودکـار بـه معناي خودکارساز ي فرآیند تست دستی است که در حال حاضر استفاده مـ ی‌شود. این عمل نیاز بـه یـ ک فرآیند تست دستی ساخت یافته دارد که در حال حاضر در سازمان یا شرکت موجود می باشد. استفاده از تست خودکار پرهزینه است. به کاربردن تست خودکار به این معنی نیست که دیگر نیازي به تست دستی نداریم و یا مـ ی توان تعداد افراد تـ یم تست را کاهش داد بلکه تست خودکار مکملـی بـراي فرآینـد تـست

[19] codification

موجود مـی باشد . توسعه ، بازبـینی و مستند ساز ي یک نمونه تست[20] خودکار می تواند بین 3 تا 10 برابر بیشتر از ایجاد و اجراي یک نمونه تست دستی زمان بر باشد ، بخصوص اگـر از روش record/playback -که در اکثر ابزارهاي تست وجود دارد- به عنوان متد اصلی تست خودکار استفاده شود .

[20] Test case

4- ابزارهاي تست خودكار نرم افزار

براي خودكار سازي تست نرم افزار ابزارهاي گوناگونی موجود است. برخی از اين ابزارها تجاري و برخی ديگر متن باز هستند . هر يک از اين ابزارها براي يک يا چند نوع از تست به کار می‌رود. در اين فصل به توضيح ابزارهاي مهم تست خودکار نرم افزار می‌پردازيم و ويژگی‌هاي هر يک را بيان می‌کنيم.

xUnit

Kent Beck در سال 1997، SUnit که يک چارچوب ساده براي تست واحد برنامه هاي نوشته شده به زبان Smalltalk باشد را ارائه نمود. پس از مدتی اين مدل توسط Kent Beck و Erich Gamma، براي زبان جاوا مورد استفاده قرار گرفت و JUnit ايجاد شد که امروزه بعنوان مهمترين ابزار تست واحد برنامه‌هاي جاوا مقبوليت بسيار زيادي يافته است . ساختار مورد استفاده در اين دو ابزار، در عين سادگی، از قابليت و کارآيی بالايی برخوردار بود . بهمين علت در طول زمان، ابزارهاي مشابهی که از ايده JUnit براي زبان هاي ديگر استفاده نمايند معرفی شدند . امروزه ابزارهاي مشابهی که همه از ساختار و مدل مشابهی برخوردارند براي تست واحد برنامه هاي نوشته شده به زبانهاي مختلف، وجود دارند که بعلت شباهت زياد، همه تحت عنوان خانواده xUnit مورد ارجاع قرار گيرند. بعنوان چند عضو از اين خانواده، که همه نيز ابزارهاي متن باز باشند، توان از CppUnit براي زبان CPP، NUnit براي زبان‌هاي پلتفرم .Net، PyUnit براي Python، VBUnit براي و يژوال بيسيک، XMLUnit براي اسناد XML نام برد. **Error! Reference source not found.**

يکی از موفق ترين اعضاي خانواده xUnit، ابزار JUint است که در قسمت به اختصار مورد معرفی قرار گيرد.

JUnit

JUnit[21]، یک چارچوب متن -باز براي تست برنامه هاي نوشته شده به زبان جاوا باشد که خودش نیز به زبان جاوا و توسط Erich Gamma و Kent Beck در سال 1997 نوشته شده است. در واقع JUnit، بر اساس طرحی از Kent Beck با نام SUnit، که یک چارچوب تست براي برنامه هاي نوشته شده به زبان Smalltalk باشد، توسعه داده شده است . قابلیت و کارایی بالا در عین سادگی، موجب شده است تا JUnit، بعنوان یک الگو قرار گیرد و چارچوب‌هاي مشابه آن براي زبان‌هاي برنامه نویسی دیگر، نظیر ASP، C#، Eiffel، Delphi، PHP، Perl، Python و Visual Basic، ایجاد شود. این چارچوب‌ها در مجموع خانواده xUnit را تشکیل دهند که همه از نظر ساختار و مدل کاري مشابه یکدیگر باشند . JUnit امروزه بعنوان یک استاندارد غیر رسمی براي انجام تست واحد برنامه هاي جاوا مطرح است و از مقبولیت بسیار بالایی برخوردار است.

فایل‌هاي اجرایی و همچنین کدهاي JUnit، از طریق سایت رسمی آن قابل دسترس می‌باشد.

عنصر مرکزي JUnit، کلاس TestCase می باشد که با استفاده از آن می‌توانیم تست‌هاي خود را ایجاد کنیم. این کلاس شامل متدها یی براي ایجاد و اجراي تست ها می‌باشد. بعنوان مثال فرض کنیم کلاسی داریم با نام Circle که داراي تعدادي فیلد و متد می باشد. اگر بخواهیم با استفاده از JUnit، به انجام تست واحد این کلاس بپردازیم، باید یک کلاس تست پیاده سازي نماییم که تست‌هاي لازم را بر روي کلاس Circle انجام می‌دهد. در نسخه‌هاي قبل از JUnit 4.0، تنها یک راه براي این کار وجود دارد. براي این کار باید یک کلاس بنویسیم که از کلاس TestCase مشتق شود. سپس در این کلاس متدهایی پیاده‌سازي می‌کنیم که هر یک به تست یکی از ویژگی‌ها یا رفتارهاي کلاس Circle می‌پردازند. در نهایت براي اجراي تست، باید این کلاس را کامپایل نموده و آن را براي اجرا به JUnit بدهیم. JUnit، بطور خودکار یک شیء از این کلاس ایجاد می کند و هر یک از متدهاي این کلاس که نامش با عبارت test شروع می شود را بطور خودکار اجرا می نماید. بنابراین افزودن تست‌هاي جدید، فقط نیازمند افزودن متدهاي جدیدي می‌باشد که نامشان با عبارت test شروع می‌شود.

[21] http://www.junit.org

در JUnit 4.0، روش ديگري نيز فراهم گرديد که براساس آن ديگر نيازي نيست کلاس تست از کلاس TestCase مشتق شود . همچنين نيازي نيست نام متدهاي تست با عبارت test شروع شود. در اين روش که مبتنی بر استفاده از annotation هاي جاوا (که در J2SE 5.0 معرفی گرديد) توان متدهاي تست را با استفاده از تگ‌هاي خاصی علامت‌گذاري کرد و آنها را به JUnit شناساند. در ادامه بحث، فقط روش اول را مد نظر قرار می‌دهيم، زيرا در اکثر متون و راهنماهاي موجود در وب، همچنين در نمونه کدهاي موجود، عمدتاً از اين روش استفاده شده است.

براي اجراي تست هاي نوشته شده، از کلاسهايی که بدين منظور در JUnit طراحی شده اند استفاده نماييم. JUnit، دو کلاس براي اجراي تست ها ارائه کند، که به آنها test runner گوييم:

اجرا کننده تست مبتنی بر متن که کلاس junit.textui.TestRunner باشد.

اجراکننده تست مبتنی بر Swing که کلاس junit.swingui.TestRunner باشد.

همانطور که پيشتر گفته شد، کلاس تست بايد از کلاس TestCase مشتق شود، در نتيجه متدهايی که در کلاس TestCase تعريف شده اند به ک لاس تست به ارث رسند، مهم‌ترين اين متدها، متدهاي assert باشند که براي بررسی نتايج به کار روند. مثلاً:: متد assertTrue بررسی می‌کند که آيا مقدار آرگومان آن برابر true است يا false. اگر مقدار آن برابر true است اين به معناي موفقيت متد تست باشد، در غير اينصورت، JUnit، اين مساله را بعنوان شکست متد تست گزارش نمايد. بنابراين در عمل، در پياده سازي متدهاي تست، از متدهايی نظير assertTrue، assertFalse، assertEquals، assertNotNull، assertNull و متدهاي ديگر که بدين منظور تعريف شده اند استفاده نماييم.

در مجموع، JUnit، براي انجام تست واحد برنامه هاي جاوا، ابزار بسيار مناسب، قدرتمند و در عين حال ساده باشد. بخصوص نسخه 4.0 آن بسيار ساده تر و غنی تر از پيش باشد و امکانات جديدي ارائه کند. بعنوان مثال توان با استفاده از annotation هاي جاوا، تعيين کنيم که متدهاي تست به چه ترتيبی اجرا شوند، يا هر متد چند بار اجرا شود، يا اينکه مثلاً يک متد تحت چه شرايطی اجرا شود.

لازم به ذكر است كه JUnit، با تعدادي از محيط هاي توسعه برنامه‌هاي جاوا نظير Eclipse، JBuilder و Intellij IDEA، نيز يكپارچه شده است و امكان تبادل با JUnit از درون اين نرم افزارها بخوبی فراهم است. همچنين با توجه به مقبوليت روزافزون متدهاي توسعه چابک افزار (agile methods)، كه توجه و تمركز بيشتري بر مسأله تست دارند، اهميت و رواج JUnit، رو به افزايش است. **Error! Reference source not found.**

HTTPUnit

پروژه httpunit[22] در سال 2000 توسط Ruse Gold شروع شد و اولين پروژه متمركز در حوزه تست نرم افزار است . و يک چارچوب منبع باز تست نرم افزار، براي تست وب سايت‌ها بدون استفاده از مرورگر است. در httpunit می‌توان از ارسال فرم‌هاي http و احراز هويت دسترسی ساده http و جاوا اسكريپت و كوكی و ...استفاده كرد. httpunit به زبان جاوا نوشته شده است و به كدهاي تست جاوا اجازه پردازش صفحات بازگشتی بصورت متن يا كانتينر هايی از فرم ها و جداول لينک‌ها ويا XMLDOM را می دهد . httpunit براي استفاده با junit بسيار مناسب است و می‌توان به سادگی تست هايی نوشت كه رفتار مناسب يک وب سايت را بررسی می كنند. httpunit به اتوماتيک كردن تست برنامه‌هاي وب بسيار كمک كند و به همين دليل در تست رگراسيون كمک بسياري می‌نمايد.

API هاي موجود در اين نرم افزار تعاملات وب را در سطح پاسخ‌ها و دريافت‌هاي http مدل می‌كند. نمونه زير را ببينيد:

```
WebConversation wc = new WebConversation();
 WebResponse resp = wc.getResponse("http://www.google.com/");
 WebLink link = resp.getLinkWith("About Google");
link.click();
 WebResponse resp2 = wc.getCurrentPage();
```

[22] http://httpunit.sourceforge.net/index.html

همـان طـور کـه مـشاهده مـی‌شـود در کـد از کـلاس‌هـاي WebConversion و WebRequest و WebResponse استفاده شود.

پايه httpunit کلاس webconversion است، که مکان سايتی را که می‌خواهيم بررسی کنيم را می‌گيرد. براي استفاه از آن، بايد يک درخواست ايجاد کرده و از webconversion تقاضاي پاسخ نمود .

مثلاً:

```
  WebConversation wc = new WebConversation();
WebRequest req = new
   GetMethodWebRequest("http://www.meterware.com/testpage.html" );
  WebResponse   resp = wc.getResponse( req );
```

حالا پاسخ برگردانده شده را می‌توان با تابع gettext() به عنوان متن و يـا بـصورت يـک DOM از طريـق تابع getDOM() بررسی کرد.

متأسفانه ، اگر در صفحه اي از مقادير زيادي کد هاي جـاوا اسـکريپت استفاده شـده باشـد، نمـی‌تـوان از httpunit براي تست آن صفحه استفاده کرد.

تست اتومات براي برنامه هاي جاواي standalone ، چه روي خط فرمان اجرا شوند و چه بـه عنـوان يـک کامپوننت استفاده شوند، با استفاده از Junit اجرا می شوند. چون برقـراري ارتبـاط در برنامـه هـاي وب از طريق http است و نه از طريق فراخوانی توابع جاوا، بنابراين Junit بـراي تـست برنامـه‌هـاي وب مناسـب نيست. در اينجا httpunit به کمک ما آيد. اين ابزار به طور رايگان در دسـترس اسـت و امکـان تـست برنامه‌هاي وب را فراهم می‌کند و در واقع مکملی براي Junit است.

توانيد از httpunit جداگانه استفاده کنيد و يا اينکه توانايی هاي هر دو را ترکيب کنيد و از ويژگی هـاي reporting , regression موجود در junit نيز استفاده کنيد. httpunit شامل کدهايی براي شـبيه سـازي عملکرد يک مرورگر، و همچنين پشتيبانی جزئی از JavaScript است. در اصل httpunit يک کلاينت وب قابل برنامه‌ريزي است.

Client API هاي موجود در httpunit اين امكان را به شما می‌دهند كه تست‌هايی بنويسيد تا كا ربري را كه به يک برنامه وب از طريق يک مرورگر دسترسی پيدا می كند را، شبيه سازي كنيد. httpunit خيلی از عملكردهايی را كه شما از يک مرورگر انتظار داريد را پشتيبانی می‌كند، از جمله مديريت كوكی براي session ها و ارسال فرم‌ها از طريق متدهاي GET و POST و احراز هويت و.... توان به دنبال عنصر خاصی روي صفحه، لينک به لينک يا فرم به فرم گشت و مطمئن شد كه برنامه نتايج درستی را برگردانده است.

دو روش اساسی براي انجام تست‌هاي اتومات روي برنامه‌هاي وب عبارتند از:

از طريق ماكروهاي ضبط شده كه می‌توانند تكرار شوند

از طريق API هاي قابل برنامه‌ريزي كه پاسخ‌هاي http را بررسی می‌كنند.

روش اول احتياج به بازبينی‌هاي متناوب دارد و كاربرد آن سخت است. براي يک API قابل برنامه ريزي، بايد اجزاي فايل HTML دريافت شده از وب سرور را از هم جدا كرد و سلسله مراتب الما هاي HTML و محتويات متنی را تشكيل داد. اين روش انعطاف پذيرتري است كه Httpunit به كار می‌برد.

نقطه ضعف روش دوم در مقايسه با روش اول زمان طولانی براي ايجاد تست‌ها و نياز به آشنايی با روش‌هاي برنز نويسی است . اما زمان maintenance در روش دوم كوتاه‌تر است زيرا با تغييرات كوچک نياز به ضبط دوباره ماكرو نيست.

كوتاهی زمان maintenance اهميت بيشتري دارد كه طولانی بودن زمان ايجاد تست را كم رنگ می‌كند. علاوه بر اين عيب ديگري كه روش اول دارد اينست كه نمی‌توانيم از روش‌هاي XP (روش‌هاي تست از ابتدا) استفاده كنيم، چون بايد يک برنامه‌ي در حال كار موجود باشد.

HTMLUnit

ابزار HTMLUnit[23] یک مرورگر است که به زبان جاوا نوشته شده و دست‌کاري سطح بالاي صفحات وب مثل تکمیل فرم ها و کلیک کردن ابرمتن و دسترسی به المان هاي خاصی در صفحه را ممکن مـی سـازد . نیازي نیست تا در خواست‌هاي سطح پایین TCP/IP و HTTP را ایجاد کنید، بلکه فقط getpage(URL) را اجرا کنید و ابرمتن دلخواه را پیدا کنیـد و حـالا جاوااسـکریپت و ajax و html بـه صـورت اتوماتیـک پـردازش مـی‌شـوند. اسـتفاده معمـول‌تـر از htmlunit در خودکارسـازي تـست صـفحات وب (حتـی بـا کتابخانه‌هاي بسیار پیچیده جاوا اسکریپت) است. گاهی اوقات نیز می‌توان براي webscraping و یا دانلود محتوي وب سایت استفاده کرد . htmlunitاخیراً پشتیبانی خـود را در زمینـه javascript افـزایش داده است. API هاي htmlunit کمی از httpunit سـطح بـالاتر اسـت و تعـاملات وب را بـصورت مـستندات و اینترفیس‌هایی که کاربر با آنها تعامل می‌کند، مدل می‌کند.

```
WebClient wc = new WebClient();
HtmlPage page = (HtmlPage)
wc.getPage("http://www.google.com");
HtmlForm form = page.getFormByName("f");
HtmlSubmitInput button = (HtmlSubmitInput)
form.getInputByName("btnG");
HtmlPage page2 = (HtmlPage) button.click();
```

همان طـور کـه مـی‌بینیـد کـد از webclient و page و link و form و button و... اسـتفاده مـی‌کنـد . صـفحاتی که از javascript ساده استفاده کرده‌اند، زمانیکه با httpunit تست می‌شوند کار مـی‌کننـد ولـی صـفحاتی که از کتابخانه هاي دیگر جاوااسکریپت استفاده می کنند، هنگامیکه با httpunit تست مـی شـوند ممکن است کار کنند یا نکنند . در واقع htmlunit یـک مرورگـر بـراي برنامـه‌هـاي جاواسـت و صـفحات HTML را مدل می‌کند و API هایی را عرضه می‌کند که به شـما اجـازه فراخـوانی صـفحات و پرکـردن فرم‌ها و کلیک کردن روي لینک ها و ... رادقیقاً همان طور که یک مرورگر نرمال عمل می‌کند، مـی دهـد. پشتیبانی‌نآ از جاوا اسکریپت خوب و رو به پیشرفت است و تقریباً بـا کتابخانـه‌هـاي پیچیـده ajax نیـز

[23] http://www.sourceforge.net/projects/htmlunit

تواند کار کند و می تواند Firefox و IE را شبیه سازي کندمعمولاً. httpunit همراه با چارچوب هاي دیگري مثل junit و testNG استفاده می شود. از httpunit در ابزارهاي منبع باز مثل canoowebtest و JWebUnit و Webdriver و ...به عنوان مرورگر استفاده می‌شود.

ویژگیهايhtmlunit :

پشتیبانی از پروتکلهاي http و https پشتیبانی از کوکی ها

پشتیبانی از روشهاي GET و POST

پشتیبانی از proxy server

پشتیبانی از javascript [html]

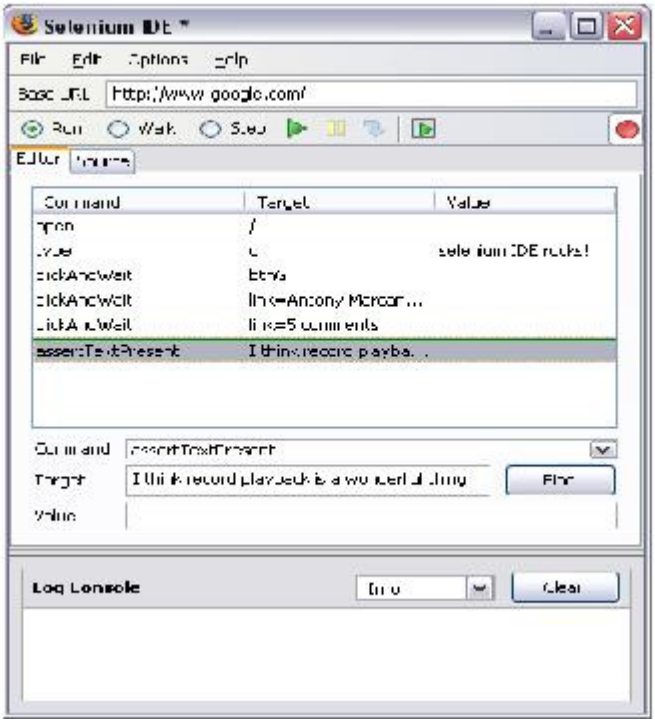
Selenium

Selenium[24] مجموعه‌اي از ابزارهاست تا در پلتفرم‌هاي مختلف بتوانید تست برنامه‌هاي وب را اتومات کنید و در مرورگرها و سیستم عامل هاي متفاوتی اجرا می شود htmlunit و httpunit رفتار یک مرورگر را تقلید می‌کنند ولی تست‌هاي seleniumدقیقاً روي یک مرورگر واقعی اجرا می‌شوند. کدهاي جاوا اسکریپت که selenium تولید می کند در حین برنامه در حال اجرا ایجاد می‌شوند و سپس دقیقاً مثل کاربر با برنامه تعامل می‌کنند.

Selenium IDE

یک add-on روي firefox که کلیک ها و تایپ ها و سایر عملیات را ضبط می‌کند و شما می‌توانید در مرورگر آ نها را دوباره اجرا کنید و یا اسکریپت ایجاد شده را ویرایش کنید. توجه کنید که در سایر مرورگرها از این نسخه نمی‌توان استفاده کرد.

[24] http://www.testinggeek.com/selenium.asp



شکل 4-1- Selenium IDE

Selenium Remote Control

انتخاب زبان و مرورگر دلخواه :

تست‌ها را روي مرورگرها و چارچوب‌هاي مختلف اجرا می کند و تست‌ها را به زبان دلخواه می‌نویسد. و بسیار پیشرفته تر از بالایی است و قادر است گستره وسیع تري از برنامه هاي وب را تست کند و مقیاس پذیرتر است و نیز براي آ غاز به ک ار کم ی به اطلاعات فنی نیاز است. در زیر نمایش ساختاري ساده شده آن را می‌بینید.

Selenium RCاز دو بخش تشکیل شده است:

یک سرور که بصورت خودکار مرورگر را اجرا و یا تمام می‌کند و براي درخواست‌هاي وب آ نها بصورت یک http proxy عمل می‌کند.

کتابخانه‌هاي کاربر براي زبان برنامه نویسی دلخواه شما

سرور seleniumمستقیماً بـه مرورگـر از طریـق Ajax(xmlhttorequest) متـصل مـی‌شـود. مـی‌توانیـد مستقیماً دستورات را به سرور با استفاده از درخواست‌هاي ساده HTTP GET/POST بفرستید. بـه ایـن معنا که از هر زبان برنامه‌نویسی که می‌تواند در خواست‌هاي HTTP ایجاد کند تا تست‌هاي selenium را روي مرورگر اتومات کند، استفاده کنید. سرور سلنیوم به عنوان یک http proxy تنظیم شده در سمت کاربر است و بین مرورگر و وب سایت شما قرار می گیرد. به این ترتیب قادر خواهید بود یـک مرورگـر بـا

قابلیت‌هاي سلنیوم، براي اجراي جاوا اسکریپت روي وب سایت‌ها داشته باشید.

با شروع اجراي یک مجموعه تست:

کلاینت یا برنامه شما به سرور سلنیوم دسترسی دارد.

سرور سلنیوم یک مرورگر را با URL ایجاد میکند .که صفحه وب هسته سلنیوم را بـار خواهد کرد.

هسته سلنیوم اولین دستور را از کلاینت یا برنامه شما میگیرد (از طریق httpproxy تعبیه شـده در سرور سلنیوم)

هسته سلنیوم دستور را اجرا می‌کند؛ مثلا بار کردن یک صفحه. [sel]

EMMA

EMMA[25]، یک ابزار متن باز براي انجام تست پوشش کد بر روي برنامه هاي نوشته شده بـه زبـان جـاوا باشد. با استفاد ه از آن توان بررسی کرد که در حین اجـراي برنامـه، کـدام دسـتورات برنامـه اجـرا شدهاند و چه مقداري از کدهاي برنامه پوشش داده شد انده . بعنوان مثال توان بررسی کـرد کـه کـدام متدها اجرا نشده اند یا کدام کلاسها اصلا مورد استفاده قرار نگرفته اند. مهمترین ویژگی EMMA سرعت بالاي آن باشد.

[25] http://emma.sourceforge.net/

روش کار بدین ترتیب است که یک برنامه اجرایی را به EMMA دهیم. EMMA بـه اجـراي برنامـه پردازد و پس از پایان برنامه، گزارشهاي مناسبی در قالب فایلهاي متنی یا فایلهاي html یا xml ایجاد کند که جاوي اطلاعات مناسبی باشند و مـشخص مـیکننـد کـه در حـین اجـراي برنامـه، کـدام قسمتهاي کد مورد اجرا قرار گرفته اند.

نکته مهم آن است که براي استفاده از این ابزار، نیازي نیست متن کد برنامه را در اختیار داشـته باشـیم . فقط کافی است فایلهاي class. برنامه که حاوي بایت کدهاي جاوا باشـند را بـه آن بـدهیم. EMMA ابتدا کدهاي خاصی را در میان کدهاي برنامه، درج کند و سپس به اجراي برنامه پردازد و سپس با استفاده از کدهایی که درج کرده به تشخیص دستوراتی که اجرا نشده اند پردازد.

EMMA، تواند میزان پوشش کلاسها، متدها، خط ها و بلاکهاي برنامه را گـزارش نمایـد. گزارشـهاي تولید شده بسیار مناسب و گویا باشند و اگر از قالـب html بعنـوان سـاختار گـزارش اسـتفاده نمـوده باشیم، امکان جابجا شدن بین صفحات گزارش با استفاده از لینکهایی که در این صفحات درج شـده انـد، بسیار راحت و کارآ باشد.

EMMA هم تواند از فایلهاي class. بعنـوان ورودي اسـتفاده نمایـد و هـم مـیتوانـد از فایلهـاي jar. استفاده نماید.

لازم به ذکر است که EMMA، بطور کامل به زبان جاوا نوشته شده است.

بعنوان مثال، برنامه زیر را که شامل دو کلاس Circle و Test باشد در نظر بگیرید.

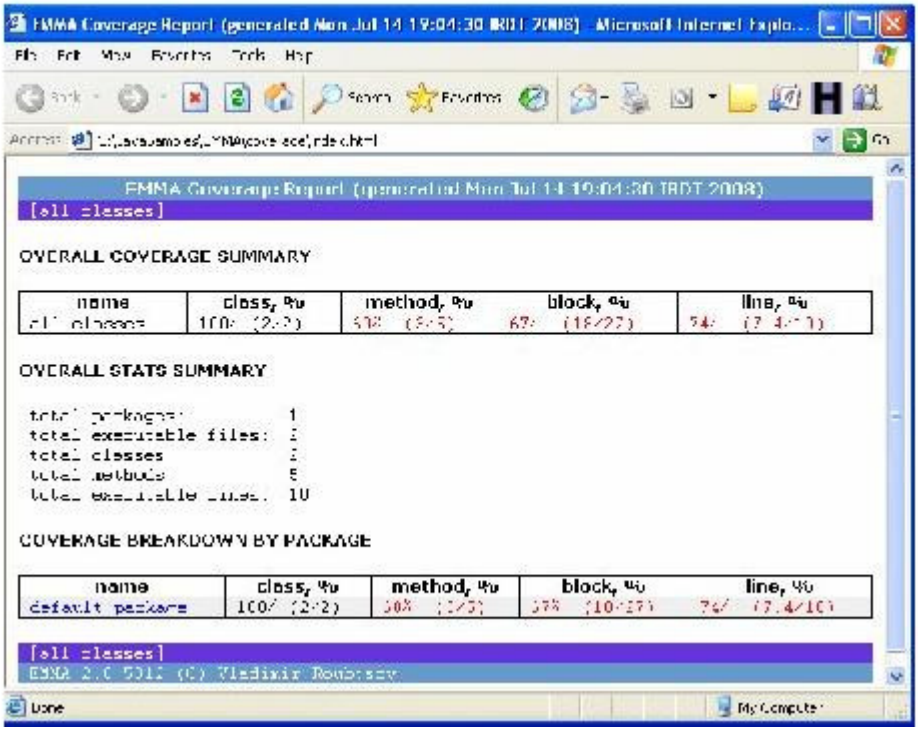
Test.java	Circle.jva
public class Test {	public class Circle { private int r;
public static void main(String[] args) {	public Circle(int r) { setR(r); }
Circle c = new Circle(10);	public void setR(int r) { if (r < 0) r = -r; this.r = r; }
}	public int getR() {
}	

```
                return r;
                }
                }
```

پس از کامپایل این دو کلاس، فایل Test.class را به EMMA دهیم تا برنامه را اجرا نماید و گزارش میزان پوشش برنامه را تولید نماید. بطور پیش فرض، گزارش توليدي در قالب یک فایل متنی ساده نوشته شود. که نمونه آن در زیر نمایش داده شده است.

```
[EMMA v2.0.5312 report, generated Mon Jul 14 18:59:40 IRDT 2008]
---------------------------------------------------------------------------
                                                                          --
                                                OVERALL COVERAGE SUMMARY:

[class, %]  [method, %]  [block, %]   [line, %]  [name]
100% (2/2)   60%  (3/5)!  67%     (18/27)!     74%  (7.4/10)!  all classes

                                                   OVERALL STATS SUMMARY:

                                                     total packages:     ١
                                                     total classes:      ٢
                                                     total methods:      ٥
                                                total executable files:  ٢
                                               total executable lines:   ١

                                              COVERAGE BREAKDOWN BY PACKAGE:

                [class, %]    [method, %] [block, %]    [line, %]  [name]
    100% (2/2)   60%  (3/5)!  67%  (18/27)!       74%    (7.4/10)! default
                                                                    package
---------------------------------------------------------------------------
                                                                          --
```

اگر گزارش را در قالب فایل html ایجاد نماییم، نتیجه شبیه آن چیزي خواهد بود که در تصویر زیر نمایش داده شده است.



شکل 4-2- **قالب** HTML **براي** EMMA

در این مثال، گزارش EMMA بیان کند که از دو کلاس موجود در برنامه، هر دو کلاس مورد استفاده قرار گرفته اند، اما از متدهاي موجود در این کلاسها، در مجموع 3 متد از بین 5 متد مورد استفاده قـ رار گرفته اند. لازم به توضیح است که برنامه با اجراي متد main، اجرا شود و سپس بر اثر ایجاد یک شیء از کلاس Circle، متد سازنده این کلاس فراخوانی شود که در درون آن نیز متد setR فراخوانی شده است. بنابراین در مجموع، 3 متد اجرا شده اند . اما علت اینکه چرا EMMA، تعداد متدهاي موجود در برنامه را 5 متد اعلام کند، آن است که در جاوا اگر براي یک کلاس متد سازنده اي ایجاد نکنیم، کامپایلر بطور خودکار، یک متد سازنده پیش فرض براي کلاس تعریف کند که کار خاصی هم انجام نمی‌دهد. در مثال فوق، در تعریف کلاس Test، هیچ سازنده اي تعریف نکرده ایم. بنابراین کامپایلر یک متد سازنده پیش فرض براي کلاس ایجاد کرده است و EMMA، در شمارش متدها، آن متد را هم به حساب آورده است.

مواردي نظیر مورد فوق، (تعداد متدهاي برنامه)، ممکن است بعنوان False Positive هاي EMMA به حساب آیند که البته با کمی دقت توان توجیه فنی آنها را نیز درك نمود.

در مجموع، EMMA، بعنوان یک ابزار براي تست پوشش برنامه هاي جاوا ابزار مناسب و قدرتمندي باشد که مزیت مهم آن، عدم نیاز به متن کد برنامه، و همچنین سرعت بالاي آن باشد. [emma]

Testing Framework based on .NET

در این چارچوب هدف توسعه یک ابزار تست افزاري کاملاً اتومات شده است. این ابزار test script هایی براي تست واحد (unit testing) ، تست یکپارچگی (integration testing) و regression testing ایجاد می‌کند.

ابزار تست باید کارهاي تست را، از تولید اسکریپت‌هاي تست، ایجاد test case ها، اعلام نتایج ، و تصحیح باگ‌ها را انجام دهد.

هدف در ابزار تستی که بر اساس .NET Framework است ،نشان دادن راهی است که بتوان یک ابزار تست را بصورت اتومات ایجاد و توسعه داد. مراحل ویرایش داده ها کاهش یابد و یک اسکریپت براي تست کل برنامه کاربردي ایجاد میشود .و شما مجبور نخواهید بود اسکریپت هاي تست را بصورت دستی ویرایش و خطا یابی کنید . محصول نهایی به سادگی یک برنامه را میپذیرد و نتایج تست را به شما باز میگرداند.

در .NET Framework.میتوان ابزار اتومات تست بسیار قوي براي تست یک محصول پیچیده افزاري با کمترین میزان دخالت انسان داشت. اگرچه زبان بحث شده در اینجا C# است ولی به معناي تمجید از محصولات مایکروسافتی و انتقاد از سایر محیط هاي برنامه نویسی و framework ها نیست و متدلوژي ارائه شده در این اینجا قابل تعمیم به سایر محیط هاي برنامه نویسی و framework ها نیز هست. javaو .NET. محیط هاي کاملا شی گرا وبراي تکمیل پروژه هاي ابزار هاي تست است.

با ابزارها ي اتومات تست افزاري فراهم شده در NETFramework نیازي به نوشتن دستی اسکریپت هاي تست و یا ضبط سناریو هاي تست نیست و این ابزار قابل استفاده مجدد است .

ابزار معرفی شده در net framework. براي اتومات کردن تست افزار قابلیت هاي زیر را دارد.

بصورت دینامیک اسمبلی[26] تحت تست را یاد میگیرد.

بصورت دینامیک کارهاي سخت و خسته کننده را انجام میدهد.

ایجاد و اجراي test script ها بصورت برنامه ریزي شده

رابط هاي اشیاي COM و سایر کامپوننتهاي افزاري را با یک مجموعه داده مشخص تست میکند.

براي تست صحت نتایج تست به یک پایگاه داده دسترسی دارد.

به رجیستري ویندوز براي اطمینان از صحت نتایج دسترسی دارد.

با ترکیب دو فضاي نام Reflectionو CodeDom قادر خواهیم بود فرایند جمع اوري اطلا عات و نوشتن اسکریپتهاي تست را با حداقل دخالت انسانی انجام دهیم.

فضاي نام Reflection میتواند یک اسمبلی خاص را بازنشانی کند.و ان را به کلاسها و توابع عضو و property ها تجزیه میکند . بنابراین بصورت اتوماتیک فرایند جمع اوري اطلاعات را انجام میدهد .و این فرایند در واقع مشابه فرایند یادگیري و شناخت مهندسان تست از برنامه است.

فضاي نام CodeDom میتواند اسکریپتهاي تست را بصورت دینامیک بر اساس اطلاعات بدست امده از فرایند فوق بنویسد.

پروژه automated test نیاز به فضایی براي ذخیره داده هاي مربوط به تست کامپوننت ها دارد.این داده ها شامل نام کلاسهاي یک اسمبلی،سازنده ها،نام سایر اعضاي کلاس ، مقادیر پارامترهاي توابع و سازنده ها ست.از xml , excel براي قابلیت ویرایش داده ها استفاده میکنیم و MS Excel API هاي قابل برنامه ریزي انعطاف پذیري زیادي براي تست کنندگان فراهم میکند تا داده هاي تست را سازمان دهی کنند و میتوان براي نوشتن اسکریپتهاي تست (که داده هاي تست را براي برنامه تست میکنند) و ذخیره نتایج به کار برد. داده هاي تست و نتایج تست با xml و excel نمایش داده میشوند.

---

[26] در دنیای .NET. برنامه های فیزیکی کامپایل شده مثل فایلهای .dll , .EXE ،با نام اسمبلی شناخته میشوند.یک اسمبلی میتواند شامل یک یا چند ماژول مثل فضاهای نام و یا کلاسها باشند.

پرونده هاي xml براي ذخيره و تبادل داده ها يک استاندارد هستند. با اضافه کردن يادداشت هاي مستند سازي xml در محيط برنامه نويسی C# ، ميتوان يک help document درست کرد.اگر در هنگام نوشتن يک تابع بتوان يک مورد تست طراحی کرد بسيار خوب است.

در کنار توابع کلاس اسمبلی براي کشف نوع در زمان اجرا ، فضاي نام system.reflection تسهيلات ديگري نيز فراهم آورد.با Late binding ميتوان يک متد را با پارامترهاي دلخواه در زمان اجرا فراخوانی کرد. با شناسايی حضور يک type، ميتوان بصورت ديناميک هر تابعی را فراخوانی کرد .

فضاي نام .NET CodeDOM

هدف اصلی تست نرم افزار نوشتن TestScriptاي است که تمامی کلاس‌ها، متدها و خصوصيت‌هاي يک اسمبلی را تست نمايد . هسته ي ابزار تست اتوماتيک ما مکانيزم خودکاري است که کارهاي زير را انجام دهد:

يافتن کلاس‌ها، متدها و خصوصيت‌ها در اسمبلی تحت تست

تعريف مورد تست‌هايی مبتنی بر ماهيت اسمبلی

نوشتن TestScriptاي که کار تست را انجام می‌دهد

اجراي TestScript

نمايش نتايج تست

استفادهي مجدد از اسمبلی TestScript، براي تست Regression و Integration

يکی از موارد استفاده ي مرسوم فضاي نام CodeDOM براي ايجاد اتوماتيک کد منبع (اصلی) است . هدف اين توليدکننده هاي کد اين است که کارهاي کدينگ را کمتر نمايد. وقتی اين برنامه اجرا می‌شود، تواند برنامه‌ي ديگري را در کسري از ثانيه توليد کند.

فضاي نام CodeDom قابليت ساخت برنامه هايی را دارد که در زمان اجرا کدهايی به زبان‌هاي مختلف توليد کند . اين فضاي نام داراي Type هايی است که ساختار برنامه را مشخص می‌کند و مستقل از زبان برنامه‌نويسی جاري است . زبان‌هاي معمولی که CodeDom توليد می‌کند، C#.NET و VB.NET است . علاوه بر اين CodeDom انواعی را براي کامپايل کد برنامه در زمان اجرا دارد که با استفاده از آنها تست افزار را پردازشی کاملاً خودکار می‌کنيم.

فضاي نام CodeDom، Source codeاي را بر اساس يک مدل واحد ايجاد می‌کند. بنابراين Source code می‌تواند به هر زبانی که مشخصات CodeDom را حمايت می کند، توليد شود. اين فضاي نام يک نمايش در مود شی که مستقل از زبان است، براي ساختار کدي که در حافظه است ارائه می‌دهد. [sybex]

Push To Test Test Maker

Push to test يک چارچوب کد باز براي خودکار سازي تست توسط برنامه نويسان افزار و مديران IT و بازرسان کنترل کيفيت است تا بتوانند سيستم هاي اطلاعاتی را تست و کنترل و هدايت کنند . برنامه نويسان براي تبديل تستهاي واحد به تست عملکرد از push to test استفاده می‌کنند.

Push to test يک محيط غنی براي ايجاد تست هاي واحد ارائه می‌دهد. از جمله :

محيط گرافيکی راحت براي کاربر ايجاد می‌کند.

زبان‌هاي اسکريپت نويسی شی گرا مثل jython , java براي ايجاد اسکريپتهاي تست،ارائه کند.

کتابخانه اي قابل گسترش از protocol handler ها . شامل پياده سازي‌هايی قابل استفاده از http , https , soap , rest , xml-rpc , smtp , pop3 , imap , وب سرويسها .

ويزارد هايی براي ايجاد يک عامل جديد، از جمله عامل recorder که ميتواند اسکريپت براي شما ايجاد کند.

کنسول براي اجراي عامل‌هاي تست در يک محيط توزيع شده تست.

کتابخانه‌اي از عامل‌هاي تست نمونه کاملاً عملی.

ويزاردها و رکوردر هايی براي ايجاد اتوماتيک تست ها بدون نوشتن کدهاي اسکريپت تست وجود دارد . براي تست‌هايی که نياز به اسکريپت نويسی دارند، در push to test امکان اسکريپت نويسی با زبان هاي مهم از جمله java, jython,Groovy, PHP,Ruby,... وجود دارد. علاوه بر اين از سرويس‌هاي SOA,webservice,ajax,... که از پروتکل‌هاي http, https, soap, xml-rpc, telnet و پروتکل‌هاي ايميل استفاده می‌کنند را، پشتيبانی می‌کند. محيط زمان اجـ راي push to test بصورت خودکار تست هاي عملکرد واحد را به تست بار و تست مقياس پذيري وتست کارايی و تست رگراسيون و تست کنترل سرويس تبديل می‌کند.

تست‌هايی که به عملکردهايی براي تست برنامه‌هاي وب و وب سرويس‌هاي SOA و سيستم‌هاي Email و برنامه‌هاي جاوا و بقيه مناسب است.

تست برنامه هاي وب: در برنامه هاي وب کاربران به عملياتی از طريق مرورگر وب دسترسی دارند و در چارچوب push to test تکنولوژي ذخيره و اجراي مجدد (capture / playback)براي تست اين برنامه ها عرضه ميشود. براي برنامه وب test agent يک کاربر مرورگر وب است.

تست وب سرويس ها و SOA(Service oriented architecture) : وب سرويس ها و SOA ارتباطات XML بين سرويس و سرويس يا برنامه با برنامه را عرضه ميکنند. در push to test ابزار soapUI زبان توصيف يک وب سرويس(WSDL) را ميخواند و تست هاي واحد و تست عملکرد ايجاد ميکند. براي وب سرويس، تستها به نظر برنامه يا سروري هستند که درخواستها را ايجاد ميکنند و منتظر پاسخها مانند.

تست سيستم هاي email: سيستم هاي email امکان دسترسی به مکانهاي ذخيره سازي پيامهاي email از طريق افزار هاي email client و پورتال هاي دسترسی به email از طريق مرورگر وب را فراهم ميکنند . در push to test براي ارتباط با يک مکان ذخيره سازي پيام از protocol handler هاي

SMTP,IMAP,POP3 استفاده میشود .و الحاقات MIME و رمز نگاري UTF و احراز هویت را پشتیبانی میکند .

تست برنامه هاي جاوا : در Push to test ، اسکریپت هاي test agent ،قادرند فراخوانی هاي صریح به توابع برنامه جاواي شما داشته باشند.

تست و تصحیح

محیط تستی که در بخش هاي قبلی توصیف شد چگونگی استفاده از چارچوب و ابزار هاي کمکی را براي ساخت تستهایی که رگراسیون و عملکرد و مقیاس پذیري و کارایی و قابلیت اعتماد را بررسی میکنند ، بیان میکرد. push to test یک راه حل یکتا براي تست و تصحیح عواملی که مانع از رسیدن به اهداف کارایی و عملکردي یک سیستم می شود را ارائه میدهد. براي مثال با اجراي یک سناریوي تست میتوان روي یک سیستم بار ایجاد کرد و پایگ اه داده و سرور برنامه را با ابزار هاي مانیتورینگ کنترل کرد و مواردي که باعث کاهش کارایی میشود را شناسایی و بهبود بخشید.

در یک محیط تست جعبه سیاه ، با درخواستهاي متوالی از سرویس و مشاهده نتایج میتوان روي آ ن سرویس بار ایجاد کرد . تست مقیاس پذیري جعبه سیاه پاسخهاي سرویس را با افزایش درخواس هاي همزمان نظاره میکند و برنامه را براي یک محیط واقعی ارزیابی میکند.

Push to test همچنین براي تستهاي جعبه خاکستري نیز مناسب است. از سرویسی که میخواهیم آ ن را تست کنیم اطلاعات داخلی را پرس و جو می کنیم تا مسائل عملکردي و مقیاس پذیري آن را کشف کنیم . سرویس تحت تست اطلاعا ت وضعیت را از عملکرد داخلی لایههاي پایگاه داده و برنامه کاربردي از

cpu , محیط سرویس ارائه میکند. براي مثال به درخواستهاي اشکالزدایی سرویس مربوط به بکارگیري

I/O , memory و وضعیتهاي تراکنشها پاسخ میدهد. این اطلاعات اضافی در سه مقوله باعث افزایش کارایی میشود.

با اندازه‌گيري توان عملياتی test agent هاي سمت كاربر ، آمادگی سيستم سرور براي مديريت جمعيت پيش‌بينی شده‌اي از كاربران را نشان می‌دهد.

تركيب عمليات عامل هاي تست در حين تست تغيير می يابند تا رخنه‌ها شناسايی شوند و مقياس پذيري حل شود.

اندازه گيري استفاده از پهناي باند CPU، I/O و حافظه معيارهاي برنامه‌ريزي براي ظرفيت را شناسايی كند.

Test network

با testnetwork شركت‌ها قادرند تست سيستم هاي خود را اتومات كنند تا عملياتی را كه به درستی انجام می شوند را، بصورت خودكار، بررسی كنندا تبار اضافی را مديريت كنند و متناوباً سيستم ها را كنترل كنند تا دچار مشكل نشوند.

Test network روي testmaker با ارائه اين ويژگيها ايجاد ميشود.

اجراي agent ها با مقياس بزرگتر روي يک ماشين . براي مثال روي يک سيستم پنتيوم 1GHz ، testmaker قادر است 1 تا 150 عامل همزمان را شبيه سازي كند و لی با testnetwork حدود 10000 يا بيشتر را ميتوان شبيه سازي كرد .

با معماري testnode / كنسول testnetwork عامل هاي تست به سرورهاي كوچكی تبديل ميشوند كه ميتوانند عمليات را مستقلا مديريت كنند.

Test Maker Monitor

يک ابزار كمكی است بـ راي بررسی آمار CPU و شبـ كه و حافظه ي سرور برنامه وب است. آمار جمع‌آوري شده از monitor يک تحليل ريشه‌اي در نمودارهاي ارائه شده در push to test با برقراري ارتباط بين امار كارايی و استفاده از منابع ارائه می‌كند .

Testmaker شامل ويزاردها و recorder براي آ سان كر دن ايجاد تست هاست. اگرچه اينها ابزار هاي قوياي در جاي خودشان هستند ولی كار هاي كمی را می توانند انجام دهند . برا ي مثال فرض كنيد ميخواهيد تستی بنويسيد كه نتايج بازگشتی از سرور را ، مطابق با منطق پيچيده اي تحليل می كند و عمل مناسب را بر اساس نتايج انجام می دهد. در موقعيتهايی مثل اين ، پيشنهاد می‌شود كه از قابليتهاي اسكريپت نويسی پوياي testmaker استفاده كنيد.

iMacros

iMacros[27]، كه مخفف عبارت internet macros باشد، محصول شركت iOpus، است كه در ابتدا يک تيم نرم افزاري آلمانی بود، و در حال حاضر در آمريكا و چند كشور اروپايی و آسيايی، متخصصانی دارد. iMacros ابزاري است كه براي خودكارسازي تعاملات كاربران با سيستمهاي تحت وب طراحی شده است. انگيزه اصلی اين بوده است كه كارهايی را كه كاربران بطور تكراري و مكرر انجام دهند، با استفاده از iMacros بطور خودكار انجام شود و در نتيجه در وقت و انرژي كاربر صرفه جويی شود. مثلا كارمندي كه هر روز بايد به چند سيستم تحت وب، وارد شود و كارهاي روتينی را انجام دهد، تواند با استفاده از iMacors، بخش زيادي از كارهاي تكراري خود را بطور خودكار انجام دهد.

اساس كار iMacros مبتنی بر مدل capture/replay است، يعنی كارهايی كه كاربر در مرورگر انجام هد را در قالب يكسري ماكرو ضبط و ذخيره نمايد و بعد قابليت اجراي آن ماكروها را در مرورگر دارد. بدين ترتيب، كاربر تواند ابتدا تعاملات خود با سيستم مورد نظر را ضبط نمايد، و سپس با استفاده از iMacros، اين تعاملات را بارها و بارها بطور خودكار تكرار نمايد. همين قابليت تعامل خودكار با صفحات وب، iMacros را به ابزاري قوي براي خودكارسازي تست سيستمهاي تحت وب تبديل كرده است.. در حال حاضر كاربران زيادي در سراسر جهان از iMacros براي انجام تست هاي مختلف بر ر وي سيستمهاي تحت وب خود استفاده كنند.

---

[27] http://www.iopus.com

ماکروها، فایلهاي متنی ساده اي هستند که مستقل از پلتفرم و مرورگر باشند. یعنی مثلا ماکرویی که در مرورگر IE ایجاد شده است را توان در مرورگر Firefox اجرا کرد.

برخی از کاربردها و ویژگی هاي iMacors بدین ترتیب است:

پرکردن فرمهاي موجود در صفحات وب، بعنوان مثال فرم ورود به یک سایت تست هاي مختلف بر روي سیستم هاي تحت وب، نظیر تست کارآیی، تست امنیت، تست بار و استرس استخراج داده ها از صفحات وب

بطور خودکار

Upload کردن اطلاعات در سیستم هاي تحت وب بطور خودکار

امکان استفاده از ماکروهاي iMacors در زبانهاي برنامه نویسی نظیر Java، #C، و PHP و Python

انجام کارهاي تکراري در سیستمهاي تحت وب، بطور خودکار و بدون نیاز به دخالت کاربر امکان ذخیره کلمات رمز بطور محرمانه و امن، براي انجام تعاملاتی که به کلمه رمز کاربر نیاز دارند. iMacros از الگوریتم رمز AES با طول کلید 256 بیت استفاده نماید.

براي اندازه گیري زمان پاسخ سیستمهاي تحت وب و ارزیابی کارایی این سیستمها

قابلیت پشتیبانی از سایتهایی که از Flash یا جاوا اسکریپت استفاده نمایند.

iMacros، داراي سه نسخه است که در سایت مربوطه امکانات آنها با هم مقایسه شده است. [imacro] [imacro2]

لازم به ذکر است که JUnit، با تعدادي از محیط هاي توسعه برنامه‌هاي جاوا نظیر Eclipse، JBuilder و Intellij IDEA، نیز یکپارچه شده است و امکان تبادل با JUnit از درون این نرم افزارها بخوبی فراهم است. همچنین با توجه به مقبولیت روزافزون متدهاي توسعه چابک نرم افزار (agile methods)، که توجه و تمرکز بیشتري بر مسأله تست دارند، اهمیت و رواج JUnit، رو به افزایش است.

5- چشم‌انداز و نتيجه‌گيري

حجم برنامه هاي كاربردي روز به روز در حال افزايش است . امروزه سازمان‌هاي نرم افزاري زمـان و منـابع زيادي را در تحل يل و تست نرم افزار صرف م ي‌كنند. از نظر مهندسان نرم افزار نوشـتن كـدهاي تـست، بـه خودي خود، مثل توسعه خود محصول وقت گير و گران است. با افزايش حجم و پيچيـدگی نـرم افزارهـا، تست كردن دستی آنها كاري بسيار سنگين و طاقت فرسا و گاه غير ممكن می‌شود. بنابراين خودكارسازي تست راه‌حلی است كه كارهاي سنگين و پرزحمت تست را ساده می‌نمايد.

خودكارسازي تست زمانی مؤثر است كه شرايط نرم افزار تحت تست مناسب باشد. بايد توجـه داشـت كـه تست خودكار به اين معنا ن يست كه كل فرآيند تست نرم افزار به صورت خودكار انجام م ي‌شود. بلكـه بـه معناي تست نرم افزار با كمک كامپ يوتر است . استفاده از تست خودكار پرهزينه است. به كـاربردن تـست خودكار به اين معنی نيست كه ديگر نيازي به تست دستی نداريم بلكه تست خودكار مكملی براي فرآيند تست موجود می باشد.

از ابزارهاي تست خودكار می توان به خانواده ي xUnit اشاره نمود. در اين متن به معرفی jUnit پرداختيم كه امروزه بعنوان مهم‌ترين ابز ار تست واحد برنامه هاي جاوا مقبوليت بسيار زيـادي يافتـه اسـت. سـاختار مورد استفاده در اين ابزار، در عين سادگی، از قابليت و كارآيی بـالايی برخـوردار اسـت. عـضو ديگـر ايـن خانواده httpUnit است كه يک چارچوب منبع باز تست افزار، براي تست وب سايت‌ها بدون استفاده از مرورگر است. در httpunit می‌توان از ارسال فرم‌هاي http و احراز هويـت دسترسـی سـاده http و جـاوا اسكريپت و كوكی و ...استفاده كرد. ابزار HTMLUnit نيز يک مرورگر است كه به زبان جاوا نوشته شده و دست‌كاري سطح بالاي صفحات وب مثل تكميل فرم ها و كليک كردن ابرمتن و دسترسی به المـان هـاي خاصی در صفحه را ممكن می‌سازد.

ابزار تست خودکار دیگر Selenium است که مجموعه‌اي است از ابزارها که تست اتومات برنامه‌هاي وب را در پلتفرم هاي مختلف امکان پذیر می کند. تست‌هـاي seleniumدقیقاً روي یـک مرورگـر واقعـی اجـرا شوند. کدهاي جاوا اسکریپت که selenium تولید می کند در حین برنامه در حال اجرا ایجاد می‌شـوند و سپس دقیقاً مثل کاربر با برنامه تعامل می‌کنند.

EMMA نیز یک ابزار متن باز براي انجام تست پوشش کد بر روي برنامه هاي نوشته شده به زبـان جـاوا باشد. با استفاده از آن می توان بررسی کرد که در حین اجـراي برنامـه، کـدام دسـتورات برنامـه اجـرا شدهاند و چه مقداري از کدهاي برنامه پوشش داده شدهاند.

چارچوبی براي تست خودکار بر مبناي .NET Framework. که با استفاده از زبان C# پیـادهسـازي شـده است، نیز در این متن بررسی شد . هدف این چارچوب توسعه یک ابزار تست نرم افزاري کاملاً اتومات شده است. این ابـزار test script هـایی بـراي تـست واحـد (unit testing) ، تـست یکپـارچگی (integration testing) و regression testing ایجاد می‌کند.

iMacros نیز ابزاري تجا ري براي خودکارسازي تعاملات کاربران با سیـستمهاي تحـت وب اسـت . انگیـزه اصلی این بوده است که کارهایی را که کـاربران بطـور تکـراري و مکـرر انجـام مـی‌دهنـد، بـا اسـتفاده از iMacros و با مدل capture/replay بطور خودکار انجام شود.

Push to test یک چارچوب کد باز براي خودکار سازي تست به منظور بتوانند تـست و کنتـرل و هـدایت سیستم هاي اطلاعاتی می‌باشد. برنامه نویسان براي تبدیل تست‌هاي واحد به تست عملکـرد از push to test استفاده می‌کنند. محیط زمان اجراي push to test بصـورت خودکار تست‌هاي عملکـرد واحـد را بـه تست بار و تست مقیاس پذیري وتست کارایی و تست رگراسیون و تست کنترل سرویس تبدیل می‌کند.

چارچوب ها و ابزارهاي تست خودکار نرم افزار بسیار متنوع هستند و هر یـک ویژگـی‌هـاي خـاص خـود را داراست و براي کاربردهاي خاصی نیز مورد استفاده قرار می گیرد. در این بین جاي یک چـارچوب تـست خودکار جامع که مجموعه‌اي از ابزارهاي خوب و کاراي موجود باشد خالی است و نیاز به چنین چـاچوبی

به وضوح قابل درك است . چارچوب پيشنهادي شامل تست محصول نهايی به روش جعبه سياه است و تأكيد اصلی آن برروي تست عملكرد نرم افزار و مقايسه آن با نيازمنديهاي سيستم است. علاوه براين در محصول نهايی، آزمونهاي جعبه سياه ديگري همچون تست استرس، تست بار و تست امنيت نيز پيش بينی شده است . از طرف ديگر، در چارچوب پيشنهادي براي توليد مجموعه موارد تست، چندين روش پيش بينی شده است كه از جمله آنها توليد موارد تست از روي مدل نرم افزار (مانند مدل UML) و يا از روي كد منبع نرم افزار است . اين كار به معناي استفاده از ابزارهاي جعبه سفيد می باشد. اما چون تستهاي اجرا شده بر روي سيستم تحت آزمون بصورت جعبه سياه عمل می كنند، در مجموع می توان گفت متدولوژي آزمون در اين چارچوب، روش جعبه خاكستري است. چراكه گرچه نيازي به داشتن اطلاعات دقيق از كد منبع نرم افزار و ارتباطات داخلی آن براي انجام آزمون نداريم، اما دسترسی به يک مدل كلی از نرم افزار (ويا دسترسی به خود كد منبع براي بدست آوردن يک مدل) به ما در تهيه موارد تست جامعتر كمک می نمايد.

٦- مراجع

[Ham04] Paul Hamill, *Unit Test Frameworks*, O'Reilly, 2004.

[Kan01-1] Kaner, Cem; James Bach, Bret Pettichord (2001). *Lessons Learned in Software Testing: A Context-Driven Approach*. Wiley, 4. ISBN 0-471-08112-4.

[Chu05] Huey–Der Chu, John E Dobson and I–Chiang Liu, *FAST: A Framework for Automating Statistics–based Testing*, 2005.

[Cra02] Rick D. Craig, Stefan P. Jaskiel, *Systematic Software Testing*, Artech House Publications, 2002.

[Ngu01] Hung Q. Nguyen, *Testing Applications on the Web: Test Planning for Internet-Based Systems*, Wiley, 2001.

[Luc06] Giuseppe A. Di Lucca, Anna Rita Fasolino, *Testing Web-based applications: The State of the Art and Future Trends*, Information and Software Technology 48 (2006) 1172-1186.

[Pre05] Pressman, R., *Software Engineering: A practitioners guide*, 6th edition, McGraw-Hill, 2005.

[Mak07] Mikko Mäkinen, *Model Based Approach to SoftwareTesting*, May 22, 2007

[Spr07] Sara E. Sprenkle, *STRATEGIES FOR AUTOMATICALLY EXPOSING FAULTS IN WEB APPLICATIONS*, summer 2007

[Mas04] Vincent Massol, ted Husted, *JUnit in Action*, Manning, 2004.

[imacro] http://www.iopus.com

[emma] http://emma.sourceforge.net

[sybex] Kanglin Li, Menqi Wu, Effective Software Test Automation: Developing an Automated Software Testing Tool, Sybex, 2004.

[sel] C. Titus. B, G. Gheorghio, J.Huggins, "An Introduction to testing web application with twill and selenium" O'Reilly , June 11,2007

[html] http://www.sourceforge.net/projects/htmlunit

[imac2] "Imacro-manual",© 2001 - 2006 iOpus SoftwareGmbH