



فصل یازدهم

بازرسی و کنترل درستی انجام کار

هدف های رفتاری

- ۱- روش های Trace و Debug کردن برنامه را انجام دهد.
- ۲- ورود داده های فرضی برای بررسی درستی انجام کار را انجام دهد.
- ۳- روش های بررسی سندهای تجزیه و تحلیل را توضیح دهد.
- ۴- روش های بررسی مستندسازی سیستم اطلاعاتی را توضیح دهد.

۱-۱ مقدمه

چرا در برخی موارد سیستم‌های اطلاعاتی نیاز متقاضیان را برطرف نمی‌کند؟ چرا در برخی موارد سیستم‌های اطلاعاتی قابل اصلاح یا تغییر یا ارتقاء نمی‌باشند؟ راه‌حل چیست؟ ما از متخصصین سیستم‌های اطلاعاتی سخنانی مانند این را زیاد شنیده‌ایم که: «ما نمی‌توانیم برنامه را تغییر دهیم زیرا که نمی‌دانیم چگونه آن را انجام دادیم یا ما نمی‌توانیم برنامه پایگاه اطلاعاتی را تعیین کنیم زیرا روشی که آن را طراحی کرده برای ما مشخص نیست یا» این شرایط وقتی رخ می‌دهد که سیستمی در حال عمل است و نیاز به تغییرات یا ارتقاء دارد ولی به دلیل ناآشنایی با روش‌های طراحی و برنامه‌نویسی آنلاین کارها ممکن نمی‌باشد. به اصطلاح در اکثر موارد اصلاح یک سیستم به مراتب سخت‌تر از ایجاد یک سیستم جدید است.

اما دلیل رخ دارد این مشکلات چیست؟ یک پاسخ عدم وجود مستندسازی برای سیستم اطلاعاتی و یا نقص در مستندسازی سیستم اطلاعاتی می‌باشد. بنابراین عدم مستندسازی سیستم اطلاعاتی می‌تواند به‌عنوان یک عامل در شکست پروژه در نظر گرفته شود. مستندسازی اغلب به‌عنوان نوع جدیدی از اطلاعات اضافی است که در دسترس سازمان و نیز تیم طراح سیستم قرار دارد.

۱-۲

تحلیل سیستم‌های اطلاعاتی

تحلیل سیستم‌های اطلاعاتی یکی از مراحل پیش‌نیاز برای توسعه سیستم است. در این مرحله انتظارات و نیازمندی‌های ذی‌نفعان سیستم شناسایی شده و گردش کار سیستم در قالب مدل‌ها و نشانه‌های استاندارد مستند می‌گردد تا از این رهگذر توسعه‌دهندگان سیستم بتوانند به یک شناخت جامع نسبت به سیستم دست یافته و آنرا منطبق بر نیازهای کاربران و ذی‌نفعان نهایی، طراحی و ایجاد نمایند. به منظور استاندارد نمودن ابزارها و روش‌های مدلسازی و مستندسازی سیستم‌های اطلاعاتی، مجموعه‌ای از مدل‌ها و متدولوژی‌ها ارائه شده‌اند که این مدل‌ها و متدولوژی‌ها به‌عنوان زبان مشترک تحلیل‌گران سیستم و برنامه‌نویسان به‌شمار می‌آیند. در قالب این مدل‌ها و متدولوژی‌ها مجموعه استانداردی از مستندات تهیه شده تا برنامه‌نویسان بتوانند با بهره‌گیری از این مستندات نیازهای کاربران را در قالب نرم‌افزارهای کاربردی پاسخگو باشند. متدولوژی RUP یکی از متدولوژی‌های تحلیل و مدلسازی سیستم‌های اطلاعاتی است که با رویکرد شیء‌گرا ارائه شده است. این متدولوژی از نشانه‌ها، علائم و مدل‌های زبان مدلسازی UML برای مدلسازی سیستم‌های اطلاعاتی استفاده می‌نماید. این متدولوژی در

مراحل چندگانه خود مستندات و خروجی های مختلفی را ارائه می نماید که می تواند توسط برنامه نویسان و توسعه دهندگان سیستم مورد استفاده قرار گیرد. در قالب این خدمت، قبل از اقدام برای توسعه یک سیستم اطلاعاتی و یا یک برنامه کاربردی، نیازمندی های کارکردی و غیرکارکردی سیستم با بهره گیری از متدولوژی RUP تعیین شده و الزامات توسعه آن در قالب مستند RFP ارائه می شود. در این حالت سازمان می تواند با به مناقصه گذاردن موضوع پروژه براساس مستندات RFQ و RFP نسبت به انتخاب پیمانکار مناسب برای تولید و استقرار سیستم اقدام نماید.

مستندسازی سیستم اطلاعاتی: مستندسازی یک پروژه ابزاری است برای ثبت و اصلاح و توسعه تصمیمات و فعالیت های مختلف که در چرخه حیات یک پروژه نقش دارند. مستندسازی می تواند در هدایت فعالیت ها و رسیدن به اهداف از پیش تعیین شده و نیز ارتقاء مسئولیت پاسخگوئی به مشتریان کمک نماید. مستندسازی در هر مرحله از اجرای پروژه باید به صورت جداگانه تهیه شود به صورتی که بتواند نقطه شروعی برای مرحله بعدی خود قرار گیرد.

طراح پروژه: یکی از مهمترین مراحل در ایجاد یک سیستم اطلاعاتی طرح پروژه می باشد که سندی رسمی است که برای اداره و کنترل و اجرای پروژه مورد استفاده قرار می گیرد و رئوس اهداف سیستم را نشان می دهد طرح پروژه ممکن است در طول زمان طراحی و ایجاد پروژه تعدیل گردد.

مرحله اجرای پروژه: در مرحله اجرای پروژه چندین گزارش تولید خواهد شد که بر چگونگی روش انجام فعالیت ها و چگونگی استانداردهای کیفیت مورد نیاز و چگونگی برنامه فعالیت و بودجه پروژه تأکید می نماید.

مرحله کنترل و تغییر: در این مرحله طرح پروژه به روز یا تعدیل شده و آینده پروژه پیش بینی می گردد.

زبان استفاده کنندگان: مستندسازی سیستم های اطلاعاتی می بایست برحسب نیاز و درک هر گروه از استفاده کنندگان به صورت جداگانه تهیه شود. انواع استفاده کنندگان می تواند شامل: کاربران، پشتیبان سیستم، طراحان سیستم و مشتریان بالقوه و تیم توسعه فنی و ... باشد.

نقش مستندسازی سیستم های اطلاعاتی

*مستندسازی یک سیستم اطلاعاتی به عنوان یکی از فعالیت های پر هزینه و زمانبر در اجرای پروژه در نظر گرفته می شود و لذا با وجود عقاید مختلف متخصصان در مورد نقش مستندسازی سیستم های اطلاعاتی، در اینجا لیست مزایای این جزء از ساختار سیستم را بیان می کنیم:

— وسیله روشن کردن جنبه های خاص سیستم در حال ارتقاء و یافتن پاسخ هایی برای مسائل مختلف که ممکن است در طول پروژه اتفاق بیفتد.

* نوعی قرارداد بین اعضای تیم پروژه و ذی‌نفعان سیستم و یا بین تیم پروژه و مدیریت سازمان در مورد نوع احتیاجاتی که سیستم باید برآورده کند یا می‌تواند برآورده کند.

_ وسیله ارتباطی بین اعضای تیم پروژه که اجزائی ضروری برای انجام فعالیت‌های مختلف خاص هر مرحله از چرخه توسعه سیستم هستند. در این خصوص می‌توان گفت از طریق یک مستندسازی.

_ اعضای هر مرحله مشکلات و نیازهای اعضای مراحل قبلی و بعدی را مطلع می‌گردند.

_ حمایت در زمان تصمیم‌گیری در گزینش بین راه‌حل‌های متفاوت که از طریق تحلیل‌هایی که مستندسازی سیستم‌های اطلاعاتی در اختیار تیم پروژه و مدیریت قرار می‌دهد. انجام می‌گیرد.

_ تجزیه تحلیل و طراحی مستندسازی سیستم یک نقطه شروع برای کاربری سیستم و تغییر و کاربردی کردن آن محسوب می‌شود. مستندسازی نهائی، عنصر اصلی برای اطمینان از درستی کار و حفظ سیستم می‌باشد.

_ مستندسازی سیستم‌های اطلاعاتی باید هر زمانی که اجزای سیستم اولیه تغییر می‌کند، تعدیل شود.

تمام جنبه‌هایی که در بالا ذکر شد به چرخه‌های ارتقاء سیستم برمی‌گردد. اما هر فرد باید در نظر داشته باشد که مستندسازی سیستم، مدیریت کافی برای توسعه پروژه را فراهم می‌کند. و براساس جزئیات و محتویات آن پروژه کنترل و نظارت می‌شود و نیز نیاز استفاده‌کنندگان که از طریق ملاقات‌های اسپانسرها با کاربران مشخص گردیده است در آن درج می‌شود.

اثرات نواقص مستندسازی سیستم‌های اطلاعاتی

نقص در جدول زمانی اولیه پیشرفت پروژه: به این صورت که ما نمی‌توانیم زمان مورد نیاز برای طراحی سیستم را در نظر بگیریم و نیز با در دست داشتن کل زمان موجود به تخصیص زمان برای مراحل مهم‌تر بپردازیم.

نقص در مشاهده بودجه عملیاتی طراحی سیستم: این امر باعث می‌گردد که هزینه‌های طراحی و آزمون و بازسازی و..... به طور صحیح قابل برآورد و تخصیص بین مراحل مختلف نباشد. ایجاد یک مستندسازی تکی برای تمام افرادی که نیاز به استفاده از آن را دارند، به مفهوم استفاده از زبانی است که تمامی استفاده‌کنندگان قابلیت فهم آن را ندارند.

نبود سندی روشن و قطعی از اجزای سیستم اطلاعاتی

* فقدان سیاست‌ها و استانداردها و فرآیندهای مورد نیاز که می‌بایست در طول طراحی و اجرای سیستم مورد نظر قرار گیرند.

* عدم امکان پاسخگویی پشتیبان‌های سیستم اطلاعاتی در مواقع مورد نیاز به اصلاح و تغییرات.

_ امکان وقوع اشتباه و خطا در بازسازی یا اصلاح یا تغییر سیستم اطلاعاتی.

— دوره زمانی طولانی تر برای تحلیل سیستم موجود: با توجه به عدم وجود شرحی از قواعد سیستم بین کاربران و پشتیبان برنامه نیاز به وقت بیشتر دارد.

— از قلم افتادن برخی نیازها و تقاضاهای کاربر.

مشکلات در آموزش سیستم

ارتقاء سیستم اطلاعاتی یک موضوع پیچیده است که می‌بایست هم از بعد اجزائی که باید شامل باشد و هم از بعد مسئولیت پاسخگوئی در مقابل استفاده‌کنندگان سیستم باید به آن توجه شود:

تیم پروژه باید مقداری از زمان خود را صرف مستندسازی سیستم بنمایند، زیرا از بین عوامل کیفیت سیستم، عامل انسانی مهمترین عامل در طراحی و اجرای یک سیستم است. زیرا پشتیبان برنامه باید از انگیزه‌های متقاضی و طراحان سیستم آگاه باشد تا بتواند برنامه را اصلاح کند.

مستندسازی سیستم اطلاعاتی باید برای هر گروه از متقاضیان به صورت جداگانه تهیه و نوشته شود.

مدیریت هر پروژه ای مسئول گردهم‌آوری و ترکیب مستندهای سیستم باشد.

مستندسازی سیستم همواره باید در هر مرحله از تغییرات در سیستم، تعدیل شود.

۱۱-۳

آشنایی با روش‌های debug و trace کردن برنامه

Debugger

Debugger برنامه‌ای است که به توسعه‌دهنده (Developer) اجازه می‌دهد برنامه را در حال اجرا مشاهده نماید. دو ویژگی مهم آنها قرار دادن Breakpoint و همچنین Trace کردن برنامه‌ها می‌باشد. این ویژگی‌ها به توسعه‌دهنده اجازه می‌دهد خطاهای برنامه را یافته و در جهت اصلاح آنها اقدام کند. Debugger یکی از مهمترین ابزارهای مهندسی معکوس بوده که از یک Disassembler برای برگرداندن کدها به زبان اسمبلی استفاده می‌نماید.

هیچ برنامه‌نویسی نمی‌تواند ادعا کند کدی که می‌نویسد، در اولین بار اجرا، بی‌شک درست کار خواهد کرد. حتی ماهرترین برنامه‌نویسان هم کد بی‌نقص نمی‌نویسند و همواره نیاز است که کدهای خود را رفع ایراد کنند تا از مشکلات آن باخبر شود، اما این اشکال‌زدایی یعنی چه و به چه صورت می‌توان آن را رفع کرد؟ شما یک برنامه‌نویس هستید

که خروجی آن مطابق با نتیجه دلخواه شما نیست و کدهای شما هم از نظر منطقی درست به نظر می‌آیند، ولی خروجی آن درست نیست. در این زمان باید خط به خط برنامه را گشت و مشکل را یافت. در غیر این صورت با افزایش حجم و تعداد خطوط برنامه، تصحیح هر کد به دشواری انجام خواهد شد. یکی از راه‌های یافتن اشکال این است که در هر مرحله مهم از برنامه، متغیرهای کد خود را مشاهده کنید و مطمئن شوید که درست در کدام خط و کجای برنامه داده‌های شما اشتباه تغییر می‌کنند. بعد از گذراندن این مرحله، باید به بررسی قطعه کدی که داده را خراب می‌کند پردازید. یکی دیگر از روش‌های رفع خطا، استفاده از آزمایش واحد است. در این روش شما توابعی که نوشته‌اید را می‌آزمایید تا از درستی خروجی آن‌ها مطمئن شوید. برای این کار بسته‌های کاربردی مختلفی تولید شده است که کار را برای تست هر تابع شما راحت تر می‌کند. روش دیگر، استفاده از ابزارهای اشکال‌زدایی (debugger) است. این ابزارها تمامی این امور را خودشان انجام می‌دهند و در هر لحظه خروجی نرم‌افزار را به شما نشان می‌دهند. به این ترتیب متوجه خواهید شد که دقیقاً در کجای برنامه خود داده‌های خراب تولید می‌کنید. برخی از این ابزارها به همراه محیط توسعه سیستم (ide) عرضه می‌شوند و برخی به صورت افزونه قابل دریافت هستند. چون امروز بیشتر برنامه‌نویسان قدرت محیط و ابزار استودیو را به خوبی درک کرده‌اند، سراغ یکی از ابزارهای دیباگ و ابزار استودیو می‌رویم.

visual studio debugger این برنامه به همراه تمامی نسخه‌های و ابزار استودیو منتشر شده است و امکانات زیادی دارد که می‌توان از میان آنها به موارد زیر اشاره کرد: ۱- یکسان کردن سورس و سمبل کدها به طور کامل ۲- اضافه شدن به پردازش‌های در حال اجرای روی سیستم برای اشکال‌زدایی (از این روش به منظور اشکال‌زدایی سرویس‌های ویندوزی نوشته شده در و ابزار استودیو استفاده می‌شود). ۳- امکان اشکال‌زدایی برنامه‌های نوشته شده در دات نت و برنامه‌های محلی نوشته شده در ++C ۴- امکان اشکال‌زدایی به صورت از راه دور ۵- قابلیت‌های ویژه و حرفه‌ای برای گذاشتن نقطه توقف (breakpoint) ۶- نمایش داده‌ها و وضعیت آنها. حال که با برخی از ویژگی‌های دیباگر و ابزار استودیو آشنا شدیم، نحوه استفاده از آن را در محیط و ابزار استودیو با هم مرور خواهیم کرد: در بخش منوها، با انتخاب گزینه debug، می‌توانید برنامه خود را در مود اشکال‌زدا یا بدون اشکال‌زدا اجرا کنید. اما تفاوت این دو حالت در چیست؟ در حالت اشکال‌زدا یک فایل شامل سمبل‌های برنامه در کنار فایل کامپایل شده ایجاد می‌شود. با استفاده از این فایل می‌توان دوباره برنامه را اشکال‌زدایی کرد و حتی امکان سوءاستفاده از آن را به هکرها خواهید داد. ولی در حالت بدون اشکال‌زدا یا عرضه (release) فایل شامل سمبل‌ها فعال نخواهد شد و گزینه‌های بهینه‌سازی کامپایلر فعال می‌شوند و از نظر حجم فایل ایجاد شده کوچک‌تر از فایل اصلی خواهد بود و سرعت اجرا شدن در این دو حالت در بعضی از الگوریتم‌ها تفاوت زیادی خواهند داشت.

جلوگیری از Debug کردن نرم افزار

CRC چیست؟

معمول ترین آن CRC^{۳۲} بوده که یک عدد ۳۲ بیتی است و برای هر داده ای قابل محاسبه می باشد از فایل گرفته تا یک مقدار رشته ای یا حتی یک قسمت از حافظه. همانطور که می دانید داده ها به صورت رشته ای از بایت ها قابل نمایش بوده که مقدار CRC هر بایت قابل محاسبه می باشد. الگوریتم های مختلفی برای محاسبه CRC وجود دارد ولی نکته قابل توجه این است که تمام آنها برای یک داده ثابت مقدار یکسانی را تولید می کنند. توجه داشته باشید که اگر برای یک داده مقدار CRC چند بار محاسبه شود مقادیر به دست آمده یکسان هستند و هر داده ای مقدار CRC مختص به خود را دارد به عبارت دیگر مقدار CRC هیچ دو داده ای با هم یکسان نیست. به طور معمول برای استفاده از قفل های سخت افزاری یا نرم افزاری برای ارتباط با قفل باید از DLL یا ActiveX استفاده نمود. به دلیل ماهیت خاص این نوع Object ها، آنها قابل جایگزین شدن بوده و با این کار می توان اداره نرم افزار را در دست گرفت. پیشنهاد می شود در صورت امکان با استفاده از روش های مختلف محاسبه CRC قبل از استفاده از متدهای یک DLL یا ActiveX مشخص نمایید که آیا این Object، Object اصلی است یا خیر؟

```
// Before using object
```

```
If (CalculateCRC object) = (MainCRC)
```

```
}
```

```
// you can use methods of object
```

```
{
```

استفاده از کدهای نامعلوم و نامشخص: Debug نرم افزار را مشکل ساخته و زمان بیشتری برای Crack کردن نرم افزار باید صرف شود. در واقع این کدها در نرم افزار عمل خاصی را انجام نداده و فقط در بخش هایی از نرم افزار که قرار است کلمه عبور یا شماره سریال وارد شود قرار گرفته و کار Debug را مشکل تر می کند. برای مثال دستوراتی به زبان اسمبلی وجود دارد (Macro) که با قرار دادن آنها در بین دستورات برنامه باعث می شود بعضی از Debuggerها را با مشکل مواجه شوند.

غیر فعال نکردن منوها و دکمه ها در نرم افزارهای Trial: اگر قرار است یک نرم افزار را به صورت Trial منتشر نمایید، منوها و دکمه ها را غیر فعال نکرده و کد اصلی را به طور مستقیم در نرم افزار قرار ندهید بعضی از ابزارهای نرم افزارسازی امکاناتی را در اختیار قرار داده تا بتوانید فقط کدهای مورد نظر را Compile نمایید :

```
{DEFINE Trial$}
```

```
{IFDEF Trial$}
```

```
// No Action
```

```
{ELSE$}
```

```
// No Compile Operation
```

```
{ENDIF$}
```

استفاده نکردن از رویدادها به طور مستقیم در **Borland Delphi**: به علت ماهیت خاص VCL (Visual Component Library) در Delphi یافتن Eventها از طریق Decompile کردن فایل EXE کار ساده ای می باشد بنابراین پیشنهاد می شود که برنامه نویسان دلفی از رویدادهای آن به طور مستقیم استفاده نکرده و به روش زیر عمل کنند :

```
Method1:=Button1.OnClick
```

```
Code Checksum - CRC Calculate
```

محاسبه CRC برای بخشی از کد یا تمام فایل در زمان اجرا، راه مناسبی برای جلوگیری از Debug می باشد. زیرا Debuggerها به منظور قرار دادن Breakpoint در برنامه باید در کد تغییر ایجاد کنند. در این لحظه با محاسبه مجدد CRC برای متدهای درون برنامه می توان مشخص نمود که کد تغییر کرده است یا خیر. این روش نه تنها در مقابل Debuggerها مؤثر است بلکه می توان در مقابل Code Patching نیز از آن استفاده نمود.

```
// Calculate CRC in memory
```

```
if (Func\CRC) != MainCRC
```

```
{
```

```
// Do an action to stop your progra
```

```
}
```

Trace کردن برنامه: تست نرم افزار عموماً در چهار سطح مختلف صورت می گیرد که این چهار مرحله به صورت

ترتیبی انجام می پذیرند و عبارتند از :

تست واحد (Unit testing)

تست مجتمع سازی (Integration Testing)

تست سیستم (System Testing)

تست پذیرش (Acceptance Testing)

تست واحد (Unit testing): یک واحد کوچک‌ترین قسمت قابل تست یک نرم‌افزار می‌باشد. که این واحد در برنامه‌نویسی شیء‌گرا می‌تواند یک متد باشد و در برنامه نویسی رویه‌ای می‌تواند کل برنامه (در زبانی مانند کوبول) یا یک تابع و ... باشد. هدف در این سطح از تست این است که آیا واحد مورد نظر به تنهایی کاری را که باید انجام بدهد می‌دهد یا نه.

تست مجتمع‌سازی (Integration Testing): تست واحد را برای هر کدام از واحدها به صورت جداگانه انجام دادید و از صحت عملکرد آنها مطمئن شدید. همه واحدها به صورت منفرد به طور صحیح وظایف خود را انجام می‌دهند، آیا نیازی به تست اینکه وقتی واحدها کنار هم قرار گرفتند و ارتباط برقرار کردند وظایفشان را به شکل صحیح انجام می‌دهند هست یا نیست. فرض کنید ۲ نفر مشغول کاری هستند هنگامی که موارد مورد نیاز برای انجام کار به طور کامل مهیا باشد هر کدام از آن ۲ فرد می‌توانند کارشان را به شکل کامل انجام بدهند. اما اگر موارد مورد نیاز برای یکی از آنها توسط دیگری تأمین شود ممکن است موارد تهیه شده دقیقاً چیزی نباشد که فرد دوم نیاز دارد، یا زمانی زیاد برای تحویل آن موارد مورد نیاز باشد که عملکرد فرد دوم را با مشکل روبرو کند. پس ما نیاز داریم تا مطمئن شویم که آیا واحدها در کنار هم کار می‌کنند، به درستی فراخوانی می‌شوند، و داده‌های درستی را در زمان مناسبی از طریق واسط‌های آنها عبور می‌دهند. (تست مجتمع‌سازی یکی از مهمترین و شاید مهمترین سطح از تست باشد، به خصوص زمانی که سیستم تغییرات زیادی دارد بعد از انجام تغییرات هرگز این مرحله را فراموش نکنید. پس روش‌های مختلف تست مجتمع‌سازی را بررسی و مطالعه کنید).

تست سیستم (System Testing): تست مجتمع‌سازی را برای نرم‌افزار مورد نظر انجام دادید و از این مطمئن شدید که تمام قطعات در کنار هم می‌توانند قرار بگیرند و بدون هیچ مشکلی وظایفشان را انجام دهند. قطعات در کنار هم مجتمع شده‌اند و پیکره اصلی نرم‌افزار تشکیل شد، ولی نرم‌افزار خود جزئی از یک سیستم است و نیاز است که با عناصر دیگر این سیستم مانند سخت افزارها ارتباط برقرار کند و با آنها مجتمع شود. پس نیاز داریم تا مطمئن شویم که سیستم به عنوان یک واحد به طور کامل عمل خواهد کرد و نیازمندی‌های سیستم را برآورده می‌کند. این سطح از تست آخرین سطحی است که توسط توسعه‌دهندگان صورت می‌گیرد تا قبل از تحویل نرم‌افزار به کاربر نهایی برای تست از عملکرد آن مطمئن شویم. برای نمونه موارد زیر در تست سیستم مورد بررسی قرار می‌گیرد:

تست امنیت (Security Testing): فرض کنید سیستم باید اطلاعات حساس و حیاتی را پردازش و مدیریت کند و افرادی هستند که به دنبال دسترسی غیرمجاز به این اطلاعات و سوءاستفاده از آن هستند. برای اطمینان از عملکرد سیستم در برابر نفوذگران ما باید مکانیزم امنیتی ایجاد شده در سیستم را بررسی کنیم تا مطمئن شویم که سیستم

می تواند نفوذهای غیر قانونی را تشخیص دهد و در برابر آنها عکس العمل نشان دهد.

تست بازیابی (Recovery Testing): در این نوع آزمایش باعث ایجاد مشکل و از کار افتادن سیستم به روش های مختلف می شویم و بررسی می کنیم که آیا سیستم می تواند خود را به طور خودکار بازیابی کند و به فعالیت خود ادامه دهد.

تست پذیرش (Acceptance Testing): نرم افزار به طور کامل توسط توسعه دهندگان در تمام سطوح تست، با موفقیت تست شد، اما آیا نرم افزار واقعاً به طور کامل (آن گونه که کاربر نهایی می خواهد) کار می کند. آیا تمام نیازهای فعلی کاربر نهایی را برآورده می کند. پس ما به آزمایشی نیاز داریم که توسط کاربران نهایی، مشتریان و نه توسعه دهندگان صورت می گیرد و هدف آن است که کاربر مشخص کند عملیاتی که برنامه انجام می دهد نیازمندی های آنها را برآورده می کند یا نه. تست پذیرش دارای انواع مختلفی است که می توان به موارد زیر اشاره کرد:

تست آلفا: تست آلفا در سایت توسعه دهنده نرم افزار و در اغلب موارد توسط کارمندان داخلی و در بعضی از موارد توسط مشتری انجام می گیرد.

تست بتا: تست بتا در سایت مشتریان و توسط مشتریان که از سیستم استفاده خواهند کرد صورت می گیرد و مشکلات مشاهده شده را به توسعه دهندگان گزارش می کند.

خلاصه فصل

تحلیل سیستم‌های اطلاعاتی یکی از مراحل پیش‌نیاز برای توسعه سیستم است. در این مرحله انتظارات و نیازمندی‌های ذی‌نفعان سیستم شناسایی شده و گردش کار سیستم در قالب مدل‌ها و نشانه‌های استاندارد مستند می‌گردد تا از این رهگذر توسعه‌دهندگان سیستم بتوانند به یک شناخت جامع نسبت به سیستم دست یافته و آن را منطبق بر نیازهای کاربران و ذی‌نفعان نهایی، طراحی و ایجاد نمایند.

مستندسازی یک پروژه ابزاری است برای ثبت و اصلاح و توسعه تصمیمات و فعالیت‌های مختلف که در چرخه حیاتی یک پروژه نقش دارند. مستندسازی می‌تواند در هدایت فعالیت‌ها و رسیدن به اهداف از پیش تعیین شده و نیز ارتقا مسئولیت پاسخگویی به مشتریان کمک نماید.

Debugger برنامه‌ای است که به توسعه‌دهنده (Developer) اجازه می‌دهد برنامه را در حال اجرا مشاهده نماید. دو ویژگی مهم آنها قرار دادن Breakpoint و همچنین Trace کردن برنامه‌ها می‌باشد. این ویژگی‌ها به توسعه‌دهنده اجازه می‌دهد خطاهای برنامه را یافته و در جهت اصلاح آنها اقدام کند. Debugger یکی از مهم‌ترین ابزارهای مهندسی معکوس بوده که از یک Disassembler برای برگرداندن کدها به زبان اسمبلی استفاده می‌نماید.

تست نرم‌افزار عموماً در چهار سطح مختلف صورت می‌گیرد که این چهار مرحله به صورت ترتیبی انجام می‌پذیرند و عبارتند از:

تست واحد (Unit testing)

تست مجتمع‌سازی (Integration Testing)

تست سیستم (System Testing)

تست پذیرش (Acceptance Testing)

تست واحد (Unit testing): یک واحد کوچک‌ترین قسمت قابل تست یک نرم‌افزار می‌باشد که این واحد در برنامه نویسی شیء‌گرا می‌تواند یک متد باشد و در برنامه‌نویسی رویه‌ای می‌تواند کل برنامه (در زبانی مانند کوبول) یا یک تابع و ... باشد. هدف در این سطح از تست این است که آیا واحد مورد نظر به تنهایی کاری را که باید انجام بدهد می‌دهد یا نه.

در تست مجتمع‌سازی (Integration Testing) ما نیاز داریم تا مطمئن شویم که آیا واحدها در کنار هم کار می‌کنند، به درستی فراخوانی می‌شوند، و داده‌های درستی را در زمان مناسبی از طریق واسط‌های آنها عبور می‌دهند.

نرم‌افزار خود جزئی از یک سیستم است و نیاز است که با عناصر دیگر این سیستم مانند سخت‌افزارها ارتباط برقرار کند و با آنها مجتمع شود. پس نیاز داریم تا مطمئن شویم که سیستم به‌عنوان یک واحد به‌طور کامل عمل خواهد کرد

و نیازمندی‌های سیستم را برآورده می‌کند. در این سطح تست سیستم انجام می‌شود. پس ما به آزمایشی نیاز داریم که توسط کاربران نهایی، مشتریان و نه توسعه‌دهندگان صورت می‌گیرد و هدف آن است که کاربر مشخص کند عملیاتی که برنامه انجام می‌دهد نیازمندی‌های آنها را برآورده می‌کند یا نه. این تست، تست پذیرش نام دارد.

خودآزمایی

- ۱- با استفاده از نرم‌افزار Visio پروژه سیستم آموزشی هنرستان (خودآزمایی فصل قبل) را مدل‌سازی نمایید.
- ۲- با استفاده از امکانات زبان ویژوال بیسیک، پروژه تمرین بالا را Debug و Trace نمایید.
- ۳- پروژه تمرین یک را از طریق تست‌های چهارگانه ارزیابی کنید.