

پایان نامه

مقطع کارشناسی ارشد

رشته : مهندسی فناوری اطلاعات

گرایش: شبکه

عنوان: اولویت بندی کارآمد موارد تست نرم افزار به کمک شبکه های بیزی

چکیده

رگرسیون این اطمینان را حاصل می کند که تغییرات بر روی رفتار کنونی نرم افزار اثر نامطلوبی نگذاشته است. یکی از تکنیک های تست رگرسیون تکنیک اولویت بندی است که باعث افزایش کارایی تست می شود. تکنیک های اولویت بندی موارد تست در بهبود تشخیص نرخ خطای تست رگرسیون موثراند. با این حال، بیشتر تکنیک های پیشنهاد شده قبلی برای خطاهای تشخیص داده شده در طول تست، شدت برابری را در نظر می گیرند که در عمل اینگونه نیست. علاوه بر این، بیشتر تکنیک های موجود بر روی اطلاعات قبلی بدست آمده از اجرای موارد تست قبلی یا تغییر در کد برنامه تکیه می کنند و تعداد کمی از آنها می توانند به طور مستقیم برای تست غیر رگرسیون استفاده شوند. در این رساله، با هدف بهبود نرخ تشخیص خطا برای تست رگرسیون و همچنین تست غیر رگرسیون، پیشنهاد کردیم یک رویکرد جدید اولویت بندی موارد تست را با استفاده از شبکه های بیزی با تکیه بر تجزیه تحلیل ساختار برنامه تحت تست. برای پیاده سازی رویکرد پیشنهادی، ما از ابزار Netica و همچنین روش کد نویسی در نرم افزار MATLAB استفاده می کنیم. و همچنین از متریک APFD (متوسط درصد شناسایی خطا) جهت ارزیابی نرخ تشخیص خطا در این رساله تکیه

می‌کنیم. و روش مبتنی بر شبکه‌های بیزی بر اساس تجزیه و تحلیل ساختار برنامه را ارائه خواهیم نمود. در نهایت با کمک یک بررسی موردی، کاربرد روش خود را نشان می‌دهیم. نتایج مقایسه تکنیک پیشنهادی با سایر تکنیک‌ها نشان می‌دهد که روش ارائه شده نسبت به سایر روش‌ها نتیجه دقیق‌تری را ارائه می‌نماید. ضمن اینکه روش پیشنهادی قادر به تصمیم‌گیری در وضعیت عدم قطعیت هم می‌باشد و این به عنوان ویژگی محسوب می‌شود. همچنین علاوه بر در نظر گرفتن معیار احتمال ابتلا به خطا و اهمیت خطا، معیارهایی نظیر اثر گذاری خطای مازول بر دیگر مازول‌ها و شدت خطای هر مازول را نیز مورد توجه قرار می‌دهد و به همین خاطر روشی کامل‌تر از روش مشابه می‌باشد و این توانایی را دارد که علاوه بر تست رگرسیون در تست توسعه نرم افزار هم مورد استفاده قرار بگیرد.

کلمات کلیدی: تست رگرسیون، شبکه‌های بیزی، اولویت بندی موارد تست، نرخ تشخیص خطا

فهرست رئوس مطالب

فصل اول : مقدمه و کلیات تحقیق

3	 مقدمه
5	 1-1- یک شیوه استراتژیک برای آزمایش نرم افزار
6	 1-2- اصول آزمایش نرم افزار
6	 1-3- برخی از انواع سطوح تست نرم افزار
9	 1-4- آشنایی با شبکه‌های بیزی
9	 1-4-1- مقدمه ای بر شبکه‌های بیزی
10	 1-5- اندازه گیری و متریک
16	 1-6- بیان مسئله

- ۷-۱- چالش موجود در تست رگرسیون 17
- ۸-۱- راه حل برای چالش موجود در تست رگرسیون 17
- ۹-۱- توجیه ضرورت انجام طرح 18
- ۱۰-۱- هدف از اجراء 19
- ۱۱-۱- نوآوری تحقیق 19

فصل دوم: ادبیات و پیشینه تحقیق

- ۱-۲- پیشینه تحقیق 21
- ۲-۱-۱- کارهای مرتبط 21
- ۲-۱-۲- بررسی مشکلات موجود در روش‌های مطالعه شده قبلی 22
- ۲-۲- تست نرم افزار 24
- ۳-۲- صحت و اعتبار سنجی 24
- ۴-۲- اهداف آزمایش 25
- ۵-۲- اصول آزمایش 26
- ۶-۲- قانون Pareto در فرآیند تست نرم افزار 26
- ۷-۲- چند نمونه از انواع تست 27
- ۸-۲- مراحل انجام تست 27
- ۹-۲- ویژگی‌های یک نرم‌افزار تست پذیر 28
- ۱۰-۲- ویژگی‌های یک تست خوب 29
- ۱۱-۲- طراحی نمونه‌های آزمایش 30
- ۱۱-۲-۱- تست جعبه سیاه 31
- ۱۱-۲-۲- تست جعبه سفید 31

- 31 ۳-11-۲ آزمایش ساختار کنترل
- 31 ۴-11-۲ آزمایش واحد
- 32 ۵-۱۱-۲ خطاهای متداول محاسبه که اغلب مشاهده می‌شوند
- 32 ۱۲-۲-۱۲-۲ آزمایش یکپارچه سازی
- 33 ۱۳-۲-۱۳-۲ آزمایش رگرسیون
- 35 ۱۴-۲-۱۴-۲ متدولوژی های مربوط به تست رگرسیون
- 35 ۱-۱۴-۲-۲ اجرای مجدد همه ی تست ها
- 36 ۲-۱۴-۲-۲ انتخاب تست رگرسیون
- 36 ۳-۱۴-۲-۲ کاهش مجموعه تست
- 36 ۴-۱۴-۲-۲ اولویت بندی موارد تست
- 37 ۱۵-۲-۱۵-۲ اولویت بندی
- 37 ۱-۱۵-۲-۲ مقدمه ای بر اولویت بندی
- 38 2-15-2 معیارهای اولویت دهی
- 39 ۳-۱۵-۲-۲ اولویت بندی موارد تست
- 39 ۲-۱۶-۲-۱۶-۲ متریک
- 39 ۱-۱۶-۲-۲ مقدمه ای برای متریک
- 40 ۲-۱۷-۱۷-۲ متریک های تست نرم افزار
- 40 ۲-۱۷-۱-۱۷-۲ خواص متریک ها در شرایط ایده آل
- 43 ۲-۱۸-۱۸-۲ معیار و متریک در تست نرم افزار
- 43 ۱-۱۸-۲-۱۸-۲ مراحل انجام کار در فرایند اندازه گیری
- 43 ۲-۱۹-۱۹-۲ متریک های آزمون
- 44 ۲-۲۰-۲۰-۲ مزایای استفاده از متریک ها

- ۲۱-۲- شبکه های بیزی 45
- ۲۱-۲-۱- استنتاج با استفاده از توزیع توام کامل 45
- ۲۱-۲-۲- مشکلات استنتاج با توزیع توام کامل و راه حل آن ها 47
- ۲۱-۲-۳- مثالی از شبکه های بیزی 48
- ۲۲-۲- مفاهیم شبکه های بیزی 50
- ۲۲-۲-۱- نمایش توزیع توام کامل 50
- ۲۲-۲-۲- رابطه های استقلال شرطی در شبکه های بیزی 52
- ۲۲-۲-۳- نمایش کارآمد توزیع های شرطی 53
- ۲۳-۲- یادگیری شبکه های بیزی 54
- ۲۴-۲- استنتاج دقیق در شبکه های بیزی 55
- ۲۵-۲- استنتاج بوسیله محاسبه تک تک عناصر احتمالی 55
- ۲۶-۲- الگوریتم حذف متغیر 57
- ۲۷-۲- استنتاج تقریبی در شبکه های بیزی 58
- ۲۸-۲- روش های نمونه گیری مستقیم 58
- ۲۸-۲-۱- نمونه گیری با رد کردن 59
- ۲۸-۲-۲- نمونه گیری وزن دار 60
- ۲۸-۲-۳- نمونه گیری زنجیره مارکوفی 61
- ۲۸-۲-۴- جمع بندی شبکه های بیزی 62
- ۲۹-۲- تحولات انجام شده تا کنون 63

فصل سوم : روش تحقیق

- ۳- انگیزه و هدف ما از ارائه این رویکرد 66

- ۳-۱- رویکرد پیشنهادی 68
- ۳-۱-۱- روند کلی در رویکرد پیشنهادی 68
- ۳-۲- محاسبه و استخراج شاخص‌ها برای ماژول‌ها 70
- ۳-۳- معیارهای رویکرد پیشنهادی 70
- ۳-۳-۱- اهمیت هر ماژول 71
- ۳-۳-۲- احتمال ابتلا به خطای ماژول 72
- ۳-۳-۳- اثرگذاری خطای ماژول بر دیگر ماژول‌ها 72
- ۳-۳-۴- شدت خطای هر ماژول 73
- ۳-۴- شاخص‌های وزن دهی به ماژول‌ها 74
- ۳-۵- ساخت شبکه بیزی 74
- ۳-۶- ایجاد ساختار گراف 75
- ۳-۷- محاسبه جداول احتمال شرطی 76
- ۳-۸- تبدیل اندازه‌ی کیفی صفت‌ها به مقادیر کمی 79
- ۳-۹- روش اول برای صفت‌های سه حالت 79
- ۳-۹-۱- مثالی از روش اول برای صفت‌های سه حالت 80
- ۳-۱۰- روش دوم برای صفت‌های سه حالت 81
- ۳-۱۰-۱- مثالی از روش دوم برای صفت‌های سه حالت 81
- ۳-۱۱- تبدیل اندازه‌ی کیفی صفت‌های غیرهم وزن به مقادیر کمی 82
- ۳-۱۱-۱- مثالی از روش تبدیل اندازه‌ی کیفی صفت‌های غیرهم وزن به مقادیر کمی 83
- ۳-۱۲- پیاده سازی مدل تست کارآمد نرم افزار با استفاده از نرم افزار Netica 85
- ۳-۱۳- پرکردن جدول احتمال شرطی با استفاده از کد نویسی 87
- ۳-۱۴- نمونه هایی از جداول احتمال شرطی فاکتورهای تست و کارآمدی اولویت بندی 90

۹۴ ۱۵-۳ پیاده سازی رویکرد پیشنهادی در مثال واقعی

فصل چهارم : محاسبات و یافته های تحقیق

۹۷ ۴-۱ ارزیابی مدل پیشنهادی

۹۷ ۴-۲ متریک (APFD)

۹۸ ۴-۳ اولویت بندی با کمک تکنیک شبکه های بییزی

۱۰۱ ۴-۴ اولویت بندی با تکنیک اصلی

۱۰۲ ۴-۵ اولویت بندی با تکنیک تصادفی

۱۰۳ ۴-۶ مقایسه روش های اولویت بندی با روش پیشنهادی

فصل پنجم : نتیجه گیری و پیشنهادات

۱۰۸ ۵-۱ نتیجه گیری

۱۱۰ ۵-۲ پیشنهادات آینده

۱۱۱ پیوست الف: واژه نامه ی فارسی به انگلیسی

۱۱۴ پیوست ب: واژه نامه ی انگلیسی به فارسی

۱۱۷ منابع و مآخذ

فهرست جداول

۴۵ جدول ۱-۲: توزیع یک قلمرو ساده

۶۳ جدول ۲-۲: سیر تحولات در مورد اولویت بندی موارد تست

۶۷ جدول ۳-۱: سیستمی ساده شامل ۱۰ ماژول و ۱۰ مورد تست

- جدول ۳-۲: سطح شدت خطای هر ماژول ۷۴
- جدول ۳-۳: تعداد سطرهای جدول احتمال شرطی فاکتورهای تست ۷۸
- جدول ۳-۴: انتساب مقدار عددی به اندازه‌های کیفی در صفت‌های سه حالت ۷۹
- جدول ۳-۵: بازه‌های تبدیل میانگین به اندازه‌های کیفی در صفت‌های سه حالت ۸۰
- جدول ۳-۶: نمونه سطری از جدول احتمال شرطی فاکتوراحتمال ابتلا به خطای ماژول، سه حالت ۸۱
- جدول ۳-۷: سطری از جدول احتمال شرطی احتمال ابتلا به خطای ماژول با سه ویژگی فرعی ۸۲
- جدول ۳-۸: نمونه سطری از جدول احتمال شرطی فاکتوراحتمال ابتلا به خطای ماژول، سه حالت ۸۵
- جدول ۳-۹: اندازه درصد احتمال حالت‌ها در فاکتورها بدون مشاهده ویژگی‌های فرعی، سه حالت ۸۷
- جدول ۳-۱۰: اندازه ویژگی‌های فرعی در یک پروژه نرم افزاری نمونه ۹۴
- جدول ۴-۱: تعداد خطای شناسایی شده توسط موارد تست با توجه به زمان کل هر مورد تست ۹۸
- جدول ۴-۲: نتایج ارزیابی از نسخه‌های مختلف برنامه تحت تست ۱۰۶
- جدول ۴-۳: تکنیک‌های اولویت بندی استفاده شده در این ارزیابی ۱۰۶

فهرست تصاویر و نمودار

- شکل ۱-۲: گراف متریک‌های کلی نرم افزار ۴۱
- شکل ۲-۲: شبکه بیزی قلمرو دستگاه آژیر ۴۸

- شکل ۳-۲: ارائه مفاهیم ساختاری به دو صورت معمول ۵۳
- شکل ۴-۲: جواب به درخواست با محاسبه عبارت بهینه تر ۵۶
- شکل ۵-۲: الگوریتم حذف متغیر ۵۷
- شکل ۶-۲: الگوریتم نمونه‌گیری با رد کردن ۵۹
- شکل ۷-۲: الگوریتم نمونه‌گیری وزن‌دار ۶۱
- شکل ۸-۲: الگوریتم نمونه‌گیری وزن‌دار ۶۲
- شکل ۳-۱: مازول‌های یک سیستم ساده نرم افزاری ۶۶
- شکل ۳-۲: یک پیاده‌سازی عمومی برای چارچوب مبتنی بر شبکه‌های بیزی ۶۹
- شکل ۳-۳: گراف شبکه بیزی تست کارآمد نرم افزار ۷۵
- شکل ۳-۴: پیاده‌سازی مدل تست کارآمد نرم افزار با استفاده از نرم افزار Netica ۸۶
- شکل ۳-۵: شبه کد پر کردن جدول احتمال شرطی برای اهمیت مازول با دو ویژگی فرعی ۸۸
- شکل ۳-۶: شبه کد پر کردن جدول احتمال شرطی برای تعداد خطاها با سه ویژگی فرعی ۸۹
- شکل ۳-۷: جدول احتمال شرطی تعداد موارد تست ۹۰
- شکل ۳-۸: جدول احتمال شرطی تعداد خطاها ۹۱
- شکل ۳-۹: جدول احتمال شرطی اهمیت مازول ۹۲
- شکل ۱۰-۳: جدول احتمال شرطی کارآمدی نهایی تست نرم افزار ۹۳
- شکل ۳-۱۱: پیاده‌سازی مدل تست کارآمد نرم افزار بروی سیستم مدیریت پرونده‌های قضایی .. ۹۵
- شکل ۴-۱: نمودار نرخ شناسایی خطا با روش BN برای اولویت بندی موارد تست ۱۰۰
- شکل ۴-۲: نمودار نرخ شناسایی خطا با روش Orginal برای اولویت بندی موارد تست ۱۰۲
- شکل ۴-۳: نمودار نرخ شناسایی خطا با روش Random برای اولویت بندی موارد تست ... ۱۰۳
- شکل ۴-۴: نمودار مقایسه نتایج روش پیشنهادی با دو روش دیگر در نرخ شناسایی خطا ۱۰۴

فصل اول

مقدمه و کلیات تحقیق

سیستم های نرم افزاری امروزه با فراگیر شدن در علوم مختلف نقش بسیار مهمی را در برطرف نمودن نیازها و خواسته های مشتریان ایفا می کنند و همچنین به عنوان یک جزء اصلی و لاینفک در امور روزمره به حساب می آیند. با گسترش روز افزون تولید سیستم های نرم افزاری همچنان تقاضا برای تولید سیستم های نرم افزاری جدید وجود دارد. بحث مهم بعد از تولید نرم افزارها نگهداری و ارتقاء آنها می باشد. وجود خطا و اشتباه در نرم افزارها می تواند منجر به خسارات زیادی از قبیل هزینه های مالی، زمانی، فیزیکی و حتی در برخی کاربردهای حساس و بحرانی مانند کاربردهای پزشکی، کنترل کننده موشک و کنترل کننده های ترافیک هوایی خسارت جانی را نیز به بار آورد. از این رو برای اینکه قابلیت اطمینان را در استفاده از سیستم های نرم افزاری افزایش دهیم باید نرم افزار را مورد تست قرار دهیم. تست نرم افزار در توسعه سیستم های نرم افزاری از جایگاه مهم و با ارزشی برخوردار است. به خصوص در سیستم های نرم افزاری مقیاس بزرگ و پیچیده امروزی. زیرا فعالیت های تست هم زمان بر و هم هزینه بر هستند. نرم افزارها برای اینکه ارتقاء یابند می بایست توسعه داده شوند و نسبت به نسخه های اولیه رشد و تکامل یابند. یکی از فعالیت های مهم و هزینه بر در جهت ارتقاء نرم افزار تست نرم افزار است که انواع متفاوتی از تست برای بخش های مختلف و در زمانهای مختلف طراحی و ایجاد شده اند. تست فرآیندی است مخرب که محصول نرم افزاری را مورد حمله قرار می دهد تا اینکه خطا بروز کند. تست نرم افزار شامل تحقیق و بررسی بر روی نرم افزار تولید شده است که این تحقیق و بررسی برای پیدا کردن خطاها انجام می شود. به طور کلی یک سری از سوال و جواب هایی هستند که نرم افزار را با آن امتحان می کنیم در حالی که از برنامه انتظار داریم با توجه به ورودی هایی که با استفاده از سوالات وارد می کنیم، جواب های صحیحی را به عنوان خروجی به دست دهد. آزمایش نرم افزار حیطه وسیعی از فعالیت های مربوط به تولید برنامه های رایانه ای را دربرمی گیرد که از آزمایش کردن کد برنامه توسط برنامه نویس گرفته تا نشان دادن عملکرد درست یک سیستم اطلاعاتی بزرگ به مشتری. سازمانها یا شرکت هایی که نرم افزارها را توسعه می دهند، محصولی به نام نرم افزار تولید می کنند. ولی چه عامل یا عواملی باعث می شوند که یک نرم افزار از نرم افزار

مشابه دیگر متمایز و برجسته شود؟ عوامل متعددی را می‌توان نام برد که باعث این برتری و تمایز شود اما یکی از این عوامل می‌تواند کیفیت محصول نهایی باشد که به بازار عرضه خواهد شد. اما برای رسیدن به این نقطه برتری، باید چگونه عمل کرد و اندیشید؟ یکی از پاسخ‌ها به این سوال بدون شک تست نرم افزار و نحوه انجام آن می‌تواند باشد.

اهمیت آزمایش نرم افزار و اثرات آن بر کیفیت نرم افزار نیاز به تأکید بیشتر ندارد. Deutch در این باره اینگونه بیان می‌نماید: توسعه سیستمهای نرم افزاری شامل یکسری فعالیت‌های تولید می‌باشد که امکان اشتباهات انسانی در آن زیاد است. خطاها در ابتدای یک فرآیند و مراحل توسعه بعدی آن ظهور می‌نمایند. به دلیل عدم توانایی انجام کارها و برقراری ارتباط به صورت کامل، توسعه نرم افزار همواره با فعالیت تضمین کیفیت همراه است. آزمایش نرم افزار عنصری حیاتی از تضمین کیفیت نرم افزار می‌باشد و مرور تقریبی مشخصه، طراحی، و تولید کد را نشان می‌دهد.

[1]

1-1- یک شیوه استراتژیک برای آزمایش نرم افزار

آزمایش، مجموعه فعالیت‌هایی است که می‌تواند از قبل به صورت سیستماتیک برنامه‌ریزی و هدایت شوند. به این دلیل، الگویی برای آزمایش نرم افزار باید برای فرآیند نرم افزار تعریف شود. این الگو شامل مجموعه مراحل است که می‌توان تکنیک‌های خاص طراحی نمونه‌های آزمایش و روش‌های آزمایش را در آن قرار داد.

چند استراتژی آزمایش نرم افزار در این رابطه پیشنهاد شده است، همه آنها برای توسعه دهنده نرم افزار،

الگویی را به منظور آزمایش فراهم می‌کنند و همگی دارای خصوصیات زیر هستند:

✓ آزمایش از سطح مؤلفه شروع می‌شود به سمت خارج در جهت مجتمع سازی کل سیستم

کامپیوتری پیش می‌رود.

- ✓ تکنیک های متفاوت آزمایش، در نقاط زمانی مختلف مناسب می باشند.
- ✓ آزمایش توسط توسعه دهنده نرم افزار و برای پروژه های بزرگ توسط گروه مستقل آزمایش، هدایت می شود.
- ✓ آزمایش و اشکال زدایی فعالیت های متفاوتی هستند، اما اشکال زدایی باید با هر استراتژی آزمایش همراه باشد.

یک استراتژی برای آزمایش نرم افزار باید آزمایش های سطح پایینی را هدایت کند که برای بازبینی صحت پیاده سازی یک قطعه کد کوچک لازم می باشند. همچنین این استراتژی باید آزمایش های سطح بالایی را سازمان دهی کند که اکثر توابع سیستم را در رابطه با نیازهای مشتری اعتبار سنجی می نمایند. یک استراتژی باید راهنمایی هایی را برای مجری و مجموعه ای از علایم نشان دهنده را برای مدیر فراهم نماید. چون این مراحل استراتژی آزمایش زمانی انجام می شوند که فشار مربوط به پایان مهلت، شروع به افزایش می نماید، پیشرفت باید قابل اندازه گیری باشد و مشکلات باید تا حد امکان به سادگی برطرف شوند.

۱-۲- اصول آزمایش نرم افزار

آزمایش، موارد غیر معمول جالب توجه ای را برای مهندس نرم افزار آشکار می نماید. در ضمن فعالیت های اولیه مهندسی نرم افزار، مهندس، سعی در ایجاد نرم افزار با استفاده از مفهومی مجرد و بدست آوردن محصولی واضح و کامل دارد و اینک آزمایش باید انجام شود. مهندس تست یکسری نمونه های آزمایش ایجاد می کند که باید نرم افزار ایجاد شده را با شکست روبرو نماید. در واقع، آزمایش، یک مرحله در فرآیند نرم افزار است که می تواند به عنوان فرآیندی مخرب به جای سازنده در نظر گرفته شود (حداقل از نظر روانشناسی) به هر حال هدف از آزمایش چیزی متفاوت از آنچه انتظار می رود می باشد. در بسیاری از شرکت ها حدود ۳۰ تا ۵۰ درصد از هزینه نرم افزار را صرف تست آن می کنند، با این وجود برخی هنوز هم بر این عقیده اند که نرم افزارها قبل از انتشار و تحویل به مشتری به درستی تست نمی شوند. چند دلیل باعث این باور می شوند: اول اینکه انجام تست نرم

افزار امری است بسیار مشکل. دوم اینکه معمولاً به خاطر زمانبر بودن و دیگر عوامل از قبیل : هزینه ی بالای تست نرم افزار ، فشارهای وارده از سوی بازار ، مشتری و رقبا منجر به این شده‌اند تا مدت زمان اختصاص داده شده برای انجام تست بسیار کمتر از دیگر بخش‌های فرایند تولید نرم افزار گردد. دلیل دیگر نداشتن برنامه (استفاده از متدولوژی و ابزاری مشخص) برای تست کردن می‌باشد. با این وجود تمامی شرکت‌های تولید کننده نرم افزار با هدف تولید نرم افزاری با کیفیت و افزایش رضایتمندی مشتری، محصول خود را مورد تست قرار می‌دهند. کیفیت چیزی است که ما در تمامی چرخه تولید نرم افزار از قبیل تولید محصولات نرم افزاری ، فرایندها و خدمات ارائه شده به مشتری یا کاربران به دنبال آن هستیم.^[2]

۳-۱- برخی از انواع سطوح تست نرم افزار

تست عملکرد: در این نوع تست، نرم افزار تست می‌شود تا از لحاظ درستی عملکرد بررسی شود تست ها اجرا می‌شوند تا ببینند که آیا نرم افزار همان گونه که انتظار می‌رود عمل می‌کند. معمولاً این روش در انتهای کار انجام می‌شود ولی می‌توان از همان ابتدای کار با تست کردن قسمت‌های کوچک مثل جزء ها نتیجه نهایی را ساده کرد.

تست فشار : برنامه در مقابل بار سنگین مثل مقادیر عددی پیچیده ، مقادیر زیادی ورودی و مقادیر زیادی پرس وجو امتحان می‌شود . که میزان باری که برنامه می‌تواند آن را تحمل کند را بررسی می‌کند . هدف ، طراحی محیطی است که مخرب تر از محیطی که برنامه در دنیای واقعی و در شرایط نرمال با آن روبرو می‌شود.

تست بار : برنامه در مقابل بار زیاد ورودی‌ها تست می‌شود مثل تست وب سایت ها برای یافتن نقاطی که در آن نقاط وب سایت یا برنامه شکست می‌خورد و یا نقاطی که در آنها کارایی وب سایت کاهش پیدا می‌کند. تست بار در سطح بار از پیش تعیین شده ای انجام می‌شود. بالاترین باری که سیستم می‌تواند بپذیرد، در حالیکه هنوز به درستی کار می‌کند. اما می‌خواهد سیستم را به طور پیوسته زیر بار نگه دارد و در صورت ایجاد مشکل با پیغام زیر روبرو می‌شود.

Error! Reference source not found

تست امنیت : با پیشرفت تجارت الکترونیک در فضای وب طراحی و تست سیستم های نرم افزاری برای اطمینان از ایمن و مطمئن بودن سیستم، مسأله ای اساسی است که توسعه دهندگان نرم افزار و متخصصان تست با آن مواجه هستند.

تست رگرسیون : بعد از تغییر نرم افزار ، حتی برای تغییر در عملکرد یا برای تصحیح یک خطا ، یک تست رگرسیون روی تمام تست هایی که قبلاً نرم افزار آنها را با موفقیت انجام داده اجرا می کند تا اطمینان حاصل کند که نرم افزار تصادفاً در عملکردهای قبلی دچار خطا نشده است. تست رگرسیون می تواند در هیچ یا همه سطوح قبلی صورت بگیرد.

از تست رگرسیون به عنوان یکی از گران قیمت ترین وظایف در میان فعالیت های نگهداری نرم افزار یاد می شود [2]، تست رگرسیون، به هر نوع از تست نرم افزار که به دنبال کشف خطاهای جدید نرم افزار باشد اطلاق می شود. رگرسیون ها در محیط های عملکردی و غیر عملکردی از یک سیستم بعد از اعمال تغییرات، از جمله تکامل، قطعه کدها یا تغییرات پیکربندی آنها ساخته شده است. همچنین از تست رگرسیون برای انجام اعتبار سنجی نرم افزارهای تغییر یافته استفاده می شود.

وجود خطا در نرم افزار و اجرای آن تاثیر منفی بر روی عملکرد و کارایی سیستم می گذارد. برای اینکه اثرات این خطرات را کاهش دهند مهندسان تست بر بکارگیری و اجرای تست رگرسیون تکیه می کنند. مهندسان تست در تست رگرسیون، موارد تست موجود در مجموعه تست را که قبلاً هم بروی نرم افزار اجرا شده اند مجدداً اجرا می کنند، همچنین موارد تست جدیدی را طراحی و ایجاد می کنند تا اینکه اطمینان حاصل شود که تمامی اثرات تغییرات در نظر گرفته شده و هیچ عوارض جانبی باقی نمانده است. تست رگرسیون تقریباً به طور عمومی توسط نرم افزار سازمان بکار گرفته شده است [2].

تست رگرسیون را نه تنها برای آزمایش صحت از یک برنامه ، بلکه اغلب برای ردیابی با کیفیت بودن خروجی هم می توان استفاده کرد. برای مثال، در طراحی یک کامپایلر، تست رگرسیون می تواند اندازه

کد، زمان شبیه سازی و زمان تلفیقی از موارد مجموعه تست را اندازه گیری کند. تست های رگرسیون را می توان به طور عمده به عنوان آزمون عملکردی یا آزمون واحد طبقه بندی کرد. آزمون عملکردی، عمل کردن برنامه کامل با ورودی های مختلف است. تست واحد، عمل کردن انفرادی زیر روال ها یا متدهای شیء. ابزارهای هر دو نوع تست عملکردی و تست واحد تمایل دارند به وجود بخش سوم محصولات که بخشی از مجموعه کامپایلر نیست، و هر دو تمایل دارند به خودکار سازی. یک تست عملکردی ممکن است یک سری اسکریپت از ورودی های برنامه، و حتی شامل یک مکانیزم خودکار برای کنترل حرکات ماوس و کلیک ها باشد. یک تست واحد ممکن است مجموعه ای از عملکردهای جداگانه در خود کد باشد. بعد از تغییر نرم افزار، حتی برای تغییر در عملکرد یا برای تصحیح یک خطا یک تست رگرسیون روی تمام تست هایی که قبلاً نرم افزار آنها را با موفقیت انجام داده اجرا می کنند تا اینکه اطمینان حاصل کنند نرم افزار تصادفاً^۱ یا به طور اتفاقی در عملکردهای قبلی دچار خطا نشده است. تست رگرسیون را می توان در همه ی سطوح تست قبل یا هیچکدام از آنها مانند: تست بار، تست فشار، تست عملکرد بکار برد. از اینرو بکار گیری و اجرای تست رگرسیون برای کیفیت نرم افزار مهم است، اما قابل ذکر است که می تواند گران و هزینه بر باشد. برای کم کردن هزینه های تست رگرسیون تکنیک هایی همچون کاهش مجموعه تست و اولویت بندی موارد تست نیز پیشنهاد شده اند که هر کدام را به طور مجزا شرح می دهیم.

۴-۱- آشنایی با شبکه های بیزی

امروزه بسیاری از مشکلات انسان ها، با کمک هوش مصنوعی حل می شود. یکی از مهم ترین خصوصیات این مشکلات وجود عدم قطعیت^۱ در آنها است. روش های زیادی در هوش مصنوعی برای کنترل عدم قطعیت پیشنهاد شده اند که اکثر آنها بر پایه نظریه احتمالات و نظریه فازی بنا نهاده

^۱uncertainty

شده‌اند. در این مطالعه می‌خواهیم یک روش برای کنترل عدم قطعیت در مسائل بر پایه نظریه احتمالات به نام شبکه‌های بیزی^{۱۳} را بررسی کنیم.

۱-۴-۱- مقدمه ای بر شبکه‌های بیزی

میدانیم که روش توزیع توام کامل می‌تواند هر درخواستی را جواب دهد، ولی می‌تواند به طور غیرقابل کنترلی بزرگ شود. همچنین می‌دانیم استقلال و یا استقلال شرطی بین متغیرها می‌تواند تعداد احتمالاتی که برای محاسبه توزیع کامل لازم است، را به طور قابل توجهی کاهش دهد. در این قسمت به معرفی یک ساختمان داده به نام شبکه بیزی می‌پردازیم که نشان دهنده وابستگی‌های فرض شده بین متغیرها است و توزیع کامل با توجه به این فرض‌های وابستگی را به طور دقیق تعیین می‌کند. شبکه بیزی یک گراف جهت‌دار است که رئوس آن شامل اطلاعات مقادیر احتمالات شرطی هستند.

به طور دقیق‌تر این شبکه شامل اجزا و خصوصیات زیر است:

۱- یک مجموعه از متغیرهای تصادفی، مجموعه رئوس گراف را تشکیل می‌دهند که این متغیرها می‌توانند گسسته یا پیوسته باشند.

۲- یک مجموعه از یال‌های جهت‌دار که اگر یک یال از راس X به راس Y باشد، X را والد Y می‌نامیم.

۳- هر گره X_i ، یک توزیع احتمال شرطی $P(X_i | \text{Parents}(X_i))$ دارد که تاثیر گره‌های والد بر روی این گره را به صورت عددی نشان می‌دهند.

۴- گراف هیچ دور جهت‌داری ندارد و در واقع، یک گراف بدون دور جهت‌دار است.

^{۱۳}Bayesian Networks

ساختار شبکه نشان دهنده وابستگی‌های شرطی در قلمرو است. به صورت شهودی، معنی یک یال از X به Y وجود تاثیر مستقیم X بر Y و یا وابستگی مستقیم Y به X است. باید توجه داشت که تعیین این وابستگی‌های مستقیم برای یک فرد خبره ساختار شبکه آنچنان سخت نمی‌باشد. پس از تعیین ساختار، تعیین توزیع شرطی مربوط به گره‌ها، ساختمان داده شبکه بیزی را کامل می‌کند و با استفاده از آن می‌توان توزیع توام کامل را بدست آورد که در ادامه توضیح داده خواهد شد. [3]

۵-۱- اندازه گیری و متریک

زمانی شما می‌توانید اندازه گیری کنید چیزی را که در موردش صحبت می‌کنید و برای آن اعداد و ارقام بیان کنید که درخصوص آن دانش و اطلاعاتی داشته باشید، اما زمانیکه شما نمی‌توانید اندازه گیری کنید چیزی را و برای آن اعداد و ارقام بیان کنید، دانش شما از نوع ناچیز و غیر رضایتبخش است. دلیل اینکه ما به متریک ها نیاز داریم این است که شما نمی‌توانید بهبود دهید چیزی را که نمی‌توانید اندازه گیری کنید و همینطور شما نمی‌توانید کنترل کنید چیزی را که نتوانید اندازه گیری نمائید.

علاوه بر این متریک های تست کمک می‌کنند به شما در موارد زیر: [4]

- ✓ تصمیم گیری برای مرحله بعدی فعالیت‌ها
- ✓ بر اساس مدارک و شواهد ادعا و یا حتی پیش بینی کنید
- ✓ تصمیم گیری در فرآیندها و یا تغییر فناوری

معیارهای زیادی در خصوص کیفیت و بهره وری تست وجود دارد. متریک های تست بیشتر برای متریک های برنامه و پروژه‌ها در نظر گرفته می‌شوند. نواقص در مراحل نیازمندی‌ها، طراحی و فاز توسعه از پروژه نرم افزاری تزریق می‌شوند. تست تلاش می‌کند با توجه به افزایش درجه تزریق نقص کاهش دهد هزینه تعمیر و یا اصلاح نقص را و همچنین در مراحل بعدی از چرخه زندگی

توسعه نرم افزار است. در تست باید به انجام برنامه ریزی برای به دام انداختن نقص پیش از آنکه محصول انتشار داده شود بپردازیم. 35 درصد از پروژه های فناوری اطلاعات با موفقیت اجرا می شوند و به اتمام می رسند. برآورد تست انجام شده توسط دو نفر به احتمال زیاد با همدیگر متفاوتند. معیارهایی که در رویکردهای قبلی بکار گرفته شده اند عمدتاً بر اساس داده های جمع آوری شده در طول تست های قبلی از نرم افزار و یا تفاوت بین نسخه های مختلف نرم افزار بوده است. این گونه عملکرد آنان باعث این شده که نتوان اینگونه رویکردها را در تست غیر رگرسیون بکار برد. زیرا در تست غیر رگرسیون، داده ها و اطلاعات اجراهای موارد تست های قبلی و اطلاعاتی از تغییر کد برنامه در دسترس نیست. در کنار روش های فوق الذکر رویکردهایی هم پیشنهاد شده اند که برای ترکیب اطلاعات چندگانه از جمله دانش کاربر و نیازمندی ها برای اولویت بندی موارد تست استفاده می کنند که این امر آنها را قادر می سازد به در نظر گرفتن شدت خطا از نقطه نظر ساختار برنامه. اما همچنان ممکن است خطاهای با تهدید جدی برای سیستم زود تشخیص داده نشوند. در این پایان نامه ما برای اولویت بندی شاخصی را ارائه می کنیم که در آن بر اساس چند معیار فرایند اولویت بندی صورت می گیرد. عواملی همچون احتمال ابتلا به خطا، اهمیت خطا، اثرگذاری خطا. از سیستم های پیشرفته سرگرمی های خانگی گرفته تا سیستم های پیچیده مالی و تجاری که معاملات چند میلیاردی را در سراسر جهان سازماندهی می کنند، همگی با نرم افزار سر و کار دارند، به عبارت دیگر نرم افزار بخش جذابی ناپذیری از فعالیت های انسان مدرن است. این میزان وابستگی انسان ها به محصولات نرم افزاری و استفاده این دستاورد بزرگ در دنیای الکترونیک تاثیر نارسایی های نرم افزاری را افزایش می دهد. امروزه تولید کنندگان و طراحان نرم افزار با طیف گسترده ای از مخاطبان روبرو هستند که از آنها به عنوان کاربر یاد می شود. این کاربرها نمی توانند وجود نارسایی در محصولات نرم افزاری را تحمل کنند. در حالی که شخصی که دارای دانش تخصصی است و با ویژگی های نرم افزار آشنا است، نواقص و کاستی ها را پدیده ای عادی تلقی می کند. با وجود اینکه ریسک از دست دادن اعتماد مشتریان در نتیجه نارسایی های موجود در محصولات نرم افزاری در فضای کنونی فناوری افزایش یافته است، شرکت هایی که در حوزه طراحی نرم افزار فعالیت می کنند اساساً همان راه سنتی خود را می روند و به فکر تغییر در روش خود

نیستند. فرایند کنونی حاکم در طراحی نرم افزار به گونه ای نیست که با تضمین کیفیت محصول در راستای کاهش نارسایی ها و بهبود بخشیدن تجربه کاربر از آن عمل کند. از سوی دیگر؛ فشار فزاینده بازاریابی و تلاش شرکت ها برای عرضه سریع تر محصول، مهندسان را مجبور می کند زمان کمتری را صرف طراحی محصول یا ارتقای نسخه های جدید آن کنند که این امر خود شرایط را بدتر می کند. عوامل مذکور باعث می شود مجموعه ای از نارسایی ها در طراحی نرم افزار به وجود بیاید؛ هزینه های پشتیبانی پس از فروش افزایش یابد و سابقه بدی که کیفیت پایین نرم افزار بر جا می گذارد تشدید شود. پژوهش های دانشگاهی نشان می دهد که تنها در سال ۲۰۰۲، صنعت فناوری رقم سرسام آوری معادل ۶۰ میلیارد دلار را صرف تشخیص و ترمیم نارسایی های موجود در محصولات نرم افزاری کرده است. شمار زیادی از شرکت ها در گزارش های خود گفته اند حدود ۵۰ درصد هزینه های طراحی نرم افزار را به تشخیص و ترمیم نارسایی های موجود در محصولات خود اختصاص می دهند⁴¹. بدیهی است که وقت آن رسیده تغییراتی بنیادین در نحوه طراحی نرم افزارها ایجاد شود تا منابع با ارزش شرکت ها صرف کار مجدد بر روی محصولات می توانست از اول با دقت بیشتری طراحی شود نگردد. آزمایش؛ مرحله ای که در طراحی نرم افزار نادیده گرفته شده است. و یا اینکه آن گونه که باید به آن توجه نشده است و این عدم توجه کافی به آن سبب بروز مشکلات فوق گردیده است. نتایج یک نظرسنجی از مدیران ارشد فناوری شرکت های فروشنده نرم افزار و مدیران فناوری اطلاعات شرکت های IT نشان می دهد بیشتر مدیران در این مورد اجماع دارند که صنعت فناوری مساله تضمین کیفیت نرم افزار را نادیده گرفته و سرمایه گذاری در این زمینه کمتر از میزان بایسته است.

این پدیده غالباً به صورت های زیر نمود می کند:

- شرکت ها در برنامه خود یک مرحله آزمایش رسمی را در پایان چرخه طراحی نرم افزار می گنجانند. بیشتر آزمایش هایی که قبل از این مرحله انجام می شود غیر رسمی و موقتی است و در آنها بسیار به جنبه های مهندسی توجه می شود تا شرایط و نیازهای کاربر.

- بعضی از آزمایش‌ها تا اواخر چرخه مهندسی نرم‌افزار کاملاً نادیده گرفته می‌شوند، آزمایش‌هایی که غالباً نادیده گرفته می‌شوند از نوع جعبه سفید^۳ (آزمایش انطباق عملکرد بازدهی نرم‌افزار با توجه به ساختار درونی و کد دهی آن) و تست عملکرد هستند. این نادیده گرفتن و اهمال در آزمایش‌های لازم باعث بروز نارسایی در نرم‌افزار و مشکلات عملکردی در مواقعی می‌شود که به تصور مهندسان در مرحله طراحی توجه لازم به آن شده است.

- نبود برنامه‌ریزی رسمی برای آزمایش نرم‌افزار، مشخص نبودن و ناهماهنگی روش‌ها و ابزار آزمایش نظیر نرم‌افزارهای موجود، خدمات حرفه‌ای کارشناسان تست نرم‌افزار و چهارچوب‌های آزمایش.

- نبود معیارهای رسمی سنجش کیفیت که باعث کاهش کارایی تصمیم‌گیری‌های مدیریتی می‌شود.

تصور کنید شرکت از یک طرف باید نرم‌افزار را به موقع در اختیار مشتری مهم خود قرار دهد و از طرف دیگر نیازمند سرمایه‌گذاری بیشتر بر روی آزمایش محصول خود است، حال آن که اهداف کیفی مشخصی برای محصول خود ندارد. چنین شرکتی تلاش می‌کند اعتبار خود را نزد مشتری از دست ندهد؛ اما عدم وجود معیارهای کیفی باعث می‌شود مدیران این شرکت در تصمیم‌گیری‌های خود دچار ناتوانی و سرگردانی شوند. مهم‌ترین پیامد اجرای یک پروژه طراحی نرم‌افزار که از کاستی‌های فوق رنج می‌برد این است که تیم مهندسی اهداف کیفی را نمی‌شناسد و در راستای آنها عمل نمی‌کند. این امر باعث می‌شود میزان سرمایه‌گذاری بر روی آزمایش محصول با اهداف کاربردی نرم‌افزار سازگار نباشد و ریسک توان نرم‌افزار در تامین نیازهای کاربر افزایش یابد. سازمان‌هایی که به کمک متخصصان تست نرم‌افزار فعالیت‌های خود را به بهترین وجه ممکن انجام می‌دهند برنامه‌های عمده طراحی نرم‌افزار را با صرف زمان برابر برای تعیین اهداف کیفی و کاربرد محصول خود آغاز می‌کنند. در این نوع برنامه‌ریزی در درجه اول مجموعه‌ای از معیارها مشخص می‌شود که

³ WHITE – BOX

بخش‌های مهندسی، مدیریت محصول و مدیریت اجرایی می‌توانند از آنها برای نظارت بر پیشرفت کیفی نرم‌افزار در دست طراحی کمک بگیرند. این معیارها امکان کنترل نارسایی، اندازه‌گیری ریسک نارسایی‌ها، بسامد نارسایی‌ها، عملکرد و هزینه آزمایش را فراهم می‌کند. سپس، به عنوان بخشی از برنامه‌ریزی اولیه پروژه‌های تیم مهندسی توجه خود را روی جنبه‌های اساسی هماهنگ کردن مدیریت آزمایش و کیفیت نظیر برنامه تست واحد، برنامه تست عملکرد، قابلیت اتوماسیون تست، برنامه تست پس‌رفت، فرایندهای نظارت بر آزمایش و گزارش دهی و فرایندهای مهندسی متمرکز می‌کند تا نارسایی‌های شناسایی شده را برطرف کند. این برنامه‌ریزی و سازمان‌دهی فرایند آزمایش به مدیران و مهندسان این توان را می‌دهد که در تعیین میزان سرمایه‌گذاری بر روی منابع آزمایش، نرم‌افزارهای لازم، خدمات حرفه‌ای کارشناسان آزمایش نرم‌افزار و زیرساخت با دانش و دقت بیشتری عمل کنند. یکی از جنبه‌های کیفیت نرم‌افزار که غالباً نادیده گرفته می‌شود امکان ردیابی و پی‌گیری نرم‌افزار بعد از رسیدن به دست مشتری است. داشتن ارتباط مستقیم و پیوسته با مشتری می‌تواند به شرکت در تشخیص و رفع سریع نواقص عملکردی محصول آن کمک کند. تیم‌های مهندسی که بر روی کیفیت سرمایه‌گذاری می‌کنند باید بر روی سیستم‌های ردیابی محصول نیز سرمایه‌گذاری کافی انجام دهند. تا بتوانند با تحلیل و بررسی نواقص، عوامل اصلی تضعیف‌کننده و تقویت‌کننده کارایی نرم‌افزار را شناسایی کنند. ردیابی کارآمد به مهندسان اجازه می‌دهد با تشخیص به موقع از عوامل اصلی عملکرد خوب یا بد نرم‌افزار درس بگیرند. تیم مهندسی باید در تعیین سیستم مدیریت کیفیت چارچوب انعطاف‌پذیری را ایجاد کند تا با پیشرفت پروژه متناسب با شرایط در روش‌ها و معیارها تغییراتی ایجاد کند. سازمان‌هایی که غالباً در اجرای پروژه‌های بزرگ نرم‌افزاری با کیفیت بالا موفق بوده‌اند آنهایی هستند که برای درسی که از نتایج پروژه می‌گیرند اهمیت خاصی داده‌اند. در پایان هر پروژه حقایقی آشکار می‌شود که می‌تواند در افزایش کارایی پروژه‌های آینده شرکت تاثیر به سزایی داشته باشد. تیم مدیریت اجرایی اصولاً تلاش می‌کند با مرور نتایج پروژه قوانینی را برای فعالیت موفق‌آمیزتر در پروژه‌های آینده استخراج کند.

۶-۱- بیان مسئله

از مشکلات اساسی در مورد تست نرم افزار و اولویت بندی موارد تست به موارد زیر می توان اشاره کرد :

- ✓ ما نمی توانیم همه چیز را تست کنیم
- ✓ سیستم نمی تواند به طور کامل تست شود
- ✓ زیاد بودن تعداد موارد تست برای اجرا بر روی برنامه تحت تست
- ✓ محدود بودن زمان و هزینه برای انجام تست

۲-۱- چالش موجود در تست رگرسیون

یکی از چالش های موجود در تست رگرسیون، زیاد بودن تعداد موارد تست است. زیرا تست رگرسیون از موارد تست موجود در مجموعه تست نیز به منظور اجرای مجدد آنها بروی نرم افزار استفاده می کند تا اینکه مطمئن شود خطاهای قبلی مجدداً ظهور نمی کنند و یا اینکه نرم افزار به طور اتفاقی تست های انجام شده قبلی را با موفقیت پشت سر نگذاشته است. بیشتر تکنیک های اولویت بندی بر روی افزایش نرخ تشخیص خطا تمرکز کرده اند و در بهبود دادن آن تلاش و روش هایی را ارائه داده اند. که به صورت تجربی هم مفید بودن این تکنیک ها در افزایش تشخیص نرخ خطا در تست رگرسیون ثابت گردیده است. اما با این حال همه ی خطاها با شدت برابر در نظر گرفته می شوند بدلیل اینکه اکثر تکنیک های موجود بر روی اطلاعات بدست آمده از اجرای موارد تست قبلی یا تغییر کد منبع تکیه می کنند، تعداد کمی از تکنیک ها قادرند به طور مستقیم برای تست غیر رگرسیون استفاده شوند. برای اینکه بتوان از تکنیک پیشنهادی هم در تست رگرسیون و هم در تست غیر رگرسیون استفاده کنیم تکنیک اولویت بندی موارد تست بر اساس تجزیه و تحلیل ساختار برنامه پیشنهاد شده است. اما با این حال بایستی از تخمین زدن و پیش بینی کردن هم به همراه این تکنیک استفاده کرد تا اینکه زودتر از دیگر روش ها نتایج اولویت بندی را به مهندسان تست نشان دهد و عدم قطعیت را هم در تصمیمات لحاظ نماید، تا اینکه از اجرای موارد تست با اولویت پایین خودداری گردد و این امر سبب کاهش زمان مورد نیاز جهت انجام تست رگرسیون و صرفه جویی در

منابع و زمان خواهد شد. برای رسیدن به این اهداف، برای اولویت دهی به موارد تست از شبکه‌های بیزی می‌توان استفاده کرد و چالش ذکر شده را رفع کرد.

۸-۱- راه حل برای چالش موجود در تست رگرسیون

برای رفع چالش ذکر شده ما پیشنهاد کر دیم مکانیزم اولویت بندی موارد تست را در تست رگرسیون با کمک شبکه‌های بیزی که در این مکانیزم که بر اساس تجزیه تحلیل ساختار برنامه انجام می‌گیرد معیارهایی نظیر شدت احتمال ابتلا به خطا و میزان اثر بخش بودن خطا در نظر گرفته می‌شود. موارد تست موجود در مجموعه تست که قبلاً هم بر روی نرم افزار اجرا شده‌اند بایستی با توجه به چند معیار در اولویت بالاتری نسبت به بقیه قرار بگیرند. این معیارها عبارتند از محل‌هایی تغییر کد برنامه، محل‌هایی که در تست های قبلی خطا در آنها وجود داشته، در جاهایی که مقداری به تابع فراخوان برگردانده می‌شود، در حلقه‌های تکرار، در دستورات شرطی می باشند. با استفاده از روش شبکه‌های بیزی اطلاعات موجود اولیه و دانش موجود در مستند سازی‌ها و احتمالات اولیه که از افراد خبره دریافت می‌گردد، تا ساختار و روابط علی بین متغیر ها استخراج گردد، تا در نهایت کارایی و کارآمدی اولویت بندی به حد مطلوب خود برسد و بتواند زمان انجام تست رگرسیون و هزینه‌های لازمه را از طریق محاسبات و معیارهای مدنظر به طور چشمگیری کاهش بدهد.

این تحقیق شامل پنج فصل است. فصل اول شامل تعاریف اولیه از تست نرم افزار و اهمیت موارد تست، ضرورت و اهداف تحقیق می‌باشد و در فصل دوم به شرح و بسط مفاهیم تست رگرسیون، شبکه‌های بیزی، اولویت دهی موارد تست و سیر تحولات رخ داده تا به امروز را بررسی می‌کنیم و یک پیشینه ای از آن ارائه می‌کنیم و سپس در فصل سوم رویکرد پیشنهادی خود را شرح می‌دهیم و بعد از آن در فصل چهارم نتایج و یافته‌های حاصل از محاسبات خود را ارائه می‌کنیم و در نهایت در فصل پنجم نتایج حاصل از محاسبات انجام شده را نشان می‌دهیم و سپس پیشنهاداتی را برای کارهای آینده بیان می‌کنیم.

۹-۱- توجیه ضرورت انجام طرح

ما در این پروژه با استفاده از شبکه‌های بیزی به انجام محاسبات متریک‌های اولویت‌بندی با در نظر گرفتن عدم قطعیت در آنها می‌پردازیم. شبکه‌های بیزی روابط علی بین متغیرها را به طور رسمی و فرمال نمایش می‌دهند که این خود مزیتی برای مدل پیشنهادی ما به شمار می‌آید. همچنین متخصصان حوزه تست با به کارگیری این رویکرد و با مد نظر قرار دادن معیارهای دیگر و افزایش کیفیت فعالیت‌های تست در طول مرحله تست زمان بیشتری را برای رفع خطاهای شناسایی شده خواهند داشت.

۱۰-۱- هدف از اجراء

ارائه مدلی کارآمد برای اولویت‌بندی موارد تست که بتوان آنرا هم در تست رگرسیون و هم در تست غیر رگرسیون بکار گرفت. همچنین زمان و هزینه لازم برای تست را از طریق اولویت‌بندی کارآمد موارد تست با کمک مدل پیشنهادی مبتنی بر شبکه‌های بیزی کاهش داد. زیرا چالش موجود در تست رگرسیون زیاد بودن تعداد موارد تست است و اجرای تمامی آنها از بهره‌وری یا کارآمدی تست رگرسیون می‌کاهد. از اینرو ما قصد داریم تا با ارائه رویکردی که بتواند از داده‌های در دسترس و ساختار برنامه موجود بر اساس معیارهای مشخص شده در رویکرد پیشنهادی نتایج بهتری را در خصوص شناسایی تعداد خطای بیشتر و کاهش تعداد خطای باقیمانده ارائه کند.

۱۱-۱- نوآوری تحقیق

نوآوری موجود در رویکرد پیشنهادی ما عبارتند از: ۱- در نظر گرفتن ساختار برنامه در سطح ماژول که در روش‌های پیشنهادی قبل از این کار در سطح کلاس بوده است. ۲- در نظر گرفتن

متریک های اثرگذاری خطای مازول بر دیگر مازول ها به همراه داده های در دسترس دیگر (نیازمندی های کاربر، اجراهای قبلی موارد تست ها، داده های تاریخی، اطلاعات تغییرات برنامه و نظرات افراد خبره در حوزه تست نرم افزار) می باشد. ۳- استفاده از شبکه بیزی به کمک نظرات افراد خبره و داده های موجود برای انجام محاسبات می باشد.

منابع پارس پژوه

فصل دوم

ادبیات و پیشینه تحقیق

۲-۱- پیشینه تحقیق

۲-۱-۱- کارهای مرتبط

در این فصل، به زمینه‌های تحقیقاتی مربوط به موضوع این پایان نامه پرداخته شده است. برای بهبود فعالیت تست نرم افزار، محققان پیشنهاد کرده‌اند تکنیک های زیادی را جهت اولویت بندی موارد تست در سالهای اخیر. Wong و همکاران یک راه برای اولویت بندی موارد تست با توجه به معیار (افزایش هزینه بر پوشش اضافی) [5]. Rothermel و همکاران یک تعریف رسمی از مشکل اولویت بندی مورد تست و ارائه متریک هایی برای اندازه گیری نرخ تشخیص خطا از مجموعه تست ارائه کرده‌اند. نویسندگان APFD را به عنوان معیاری برای نرخ تشخیص خطا و ارزیابی تجربی در کاتالوگ خودشان معرفی می کنند [6]. در مطالعات موردی بزرگ تر تکنیک ها با بیش از یک معیار ارزیابی می شوند. Kim و همکاران اولویت بندی موارد تست را تدوین و رابطه کردند بر اساس تئوری احتمال [7]. تکنیک های زیادی برای تست پیشنهاد شده‌اند مانند کاهش مجموعه تست [8] و

اولویت بندی موارد تست [9]. مطالعات تجربی انجام شده نشان می‌دهند که، بیشتر تکنیک‌های اولویت بندی موارد تست در بهبود نرخ تشخیص خطا در تست رگرسیون مفید و موثر هستند. استفاده از روش بهینه سازی اجتماع مورچگان یک راه جدید برای حل مشکل محدودیت زمان اولویت بندی است. در این رهکار ارائه شده، هدف اولویت بندی در تست رگرسیون برای دوباره مرتب کردن مجموعه تست در محیط محدود است. ممکن است تکنیک‌های آگاه به زمان برای اولویت بندی موارد تست، نرخ تشخیص خطای بهتری نسبت به روش‌های سنتی اولویت بندی موارد تست دست یابد. رویکرد جدیدی را برای آگاه به زمان اولویت بندی موارد تست با استفاده از برنامه نویسی خطی عدد صحیح را پیشنهاد می‌کند. اولویت بندی موارد تست به چالشی در سیستم‌های نرم افزاری مبتنی بر مولفه شده است (CBSS) که تسهیل می‌کند توسعه سیستم‌های پیچیده را با یکپارچه سازی اجزای قابل استفاده مجدد. CBSS به عنوان یک رویکرد ظهور کرده که توسعه سریع سیستم را با استفاده از منابع و تلاش کمتر ارائه می‌کند. ایده اصلی استفاده مجدد و کاهش هزینه‌های توسعه را می‌توان به دست آورد از اجزای ارائه خدمات قابل اعتماد. بنابراین، ادغام اجزاء و آزمایش تبدیل شدن به یک مرحله مهم در CBSS. در هدف تجزیه و تحلیل روش‌های اولویت بندی موارد تست موجود مبتنی بر کد، بر اساس نیازمندی و تکنیک‌های اولویت بندی مبتنی بر مدل و در نهایت پیاده سازی آنها در CBSS است [10].

در [27] اولویت بندی موارد تست نرم افزار براساس شبکه‌های بیزی برای تست رگرسیون ارائه گردیده است. در رویکرد پیشنهاد شده، اطلاعات تغییرات در نرم افزار، کد برنامه و مجموعه تست رگرسیون ورودی‌های مدل پیشنهاد شده را تشکیل می‌دهند. در ضمن این رویکرد برای تست رگرسیون ارائه شده و برای استفاده در تست غیر رگرسیون نمی‌توان از این رویکرد بهره برد. در این روش پس از ارزیابی و محاسبه متریک احتمال ابتلا به خطا، تجزیه و تحلیل تغییرات کد برنامه و بررسی پوشش موارد تست، شواهد لازم برای ساخت شبکه بیزی استخراج می‌شوند و بر اساس اطلاعات بدست آمده ساختار گراف ایجاد می‌گردد. پس از ایجاد گراف در شبکه بیزی، استنتاج احتمالات صورت می‌گیرد و موارد تست بر اساس آن اولویت بندی می‌شوند. برای مدل پیشنهادی که مورد مطالعه قرار گرفته است می‌توان این نکته را بیان کرد، اینکه مدل پیشنهادی با توجه به استفاده از شبکه بیزی دارای یک نقص می‌باشد و آن هم اینکه فقط برای تست رگرسیون می‌توان از آن استفاده کرد و برای تست غیر رگرسیون کاربرد ندارد.

در [31] اولویت بندی بر اساس تجزیه تحلیل ساختار برنامه برای تست رگرسیون و هم تست غیر رگرسیون ارائه شده است. مدل ارائه شده مشکل موجود در رویکرد ارائه شده در [27] را حل کرده است. اینگونه که مدل خود را در سطح مازول ها ارائه کرده است و بر اساس ساختار برنامه به اولویت بندی کردن موارد تست می پردازد. در این روش با بکار گیری روش های فراخوانی گراف روابط بین مازول ها را مشخص می کند و بعد از این مرحله معیارها و متریک هایی را برای وزن دهی به هر یک از متریک ها در نظر می گیرد. ورودی این مدل را ساختار برنامه و مجموعه تست تشکیل می دهند. در مدل پیشنهاد شده، از مجموعه تست برای سنجش اینکه موارد تست چه میزان از برنامه تحت تست را پوشش می دهند استفاده می شود. در نهایت اطلاعات حاصل از مرحله تجزیه و تحلیل پوشش موارد تست نرم افزار و محاسبه متریک ها، اطلاعات مورد نیاز جهت اولویت بندی موارد تست در خروجی بدست می آیند. از موارد مهمی که در این مدل بایستی بهبود داده شوند این موارد قابل ذکر می باشند. اول اینکه عدم قطعیت در این مدل نادیده گرفته شده است و بایستی در مدل حتماً گنجانده شود. متریک های در نظر گرفته شده می تواند متعدد باشد و همچنین شدت خطاهای شناسایی شده باهم یکسان نمی باشند.

در [37] اثر محدودیت های زمانی در مورد اولویت بندی موارد تست مورد بررسی قرار گرفته است. ورودی های این مدل عبارتند از : داده های پوشش تست، اطلاعات تغییر کد ، داده های متریک کیفیت و مجموعه های تست. در این مدل از سه گروه از تکنیک های تست استفاده شده است. گروه های استفاده شده عبارتند از : تکنیک های کنترل ، تکنیک های بدون بازخورد و تکنیک های با مکانیزم بازخورد. از بین تمامی مجموعه های تست موجود نرخ شناسایی همگی محاسبه می گردد و مجموعه تست ای که نرخ شناسایی خطای در آن از بقیه بیشتر است انتخاب می گردد و سپس اطلاعات تشخیص خطای حاصل از آن به عنوان ورودی به مدل های هزینه - فایده داده می شود تا اینکه تعیین گردد در مواقعی که در محدودیت زمانی اجرا می شوند نرخ شناسایی خطای قابل قبولی را ارائه می کنند. نتایج حاصله از آزمایشات انجام گرفته در این تحقیق نشان دهنده این موضوع می باشد که استفاده از شبکه های بیزی در اولویت بندی موارد تست نتایج مطلوبی را بدست می دهند. در [39] ساخت مدل های شبکه بیزی از عدم قطعیت های موجود در تست و نگهداری نرم افزار

صورت گرفته است. عدم قطعیت در مهندسی نرم افزار که بطور گسترده در تائید، ارزیابی و پیش بینی دیده می شود. عدم قطعیت در مهندسی نرم افزار ذاتی و اجتناب ناپذیر است. عدم قطعیت در خصوص تجزیه تحلیل نیازمندی های کاربر هم یکی از بحث های مهم در حوزه مهندسی نرم افزار می باشد. در این روش از شبکه های بیزی باور الهام گرفته شده است و برای مدل کردن عدم قطعیت موجود در هر یک از فازهای مهندسی نرم افزار استفاده کرده است.

۲-۱-۲- بررسی مشکلات موجود در روش های مطالعه شده قبلی

کارهای مرتبط ذکر شده در بالا دارای نقاط ضعف و کمبودهایی هستند که در ادامه اشاره ای مختصر به آنها خواهیم داشت. اول اینکه روش های بالا همگی در بکارگیری اولویت بندی موارد تست در تست رگرسیون مشترک هستند. تمرکز بر افزایش نرخ تشخیص خطا یکی دیگر از مشترکات کارهای مرتبط می باشد. روش های مورد مطالعه برای ارائه مدلی که بتواند در تست غیر رگرسیون هم بکار گرفته شود راهکاری ندارند و برای مدل های خود در شروع فرآیند تست نیاز به یک سری موارد تست اجرا شده قبلی، اطلاعات از مستندات نوشته شده از مراحل قبل، تست های اجرا شده قبلی بر روی نرم افزار و تغییرات کد منبع دارند. از دیگر مشکلاتی که می توان ذکر کرد، در نظر گرفتن شدت مساوی برای خطاهای یافت شده است که این یک نقص محسوب می شود. با وجود موارد ضعف ذکر شده برای رویکردهای قبلی مورد مطالعه در حوزه تست نرم افزار، استفاده از یک مدل ریاضی و احتمالی می تواند بسیاری از ضعف ها را پوشش دهد. راهکار پیشنهادی ما، استفاده از شبکه های بیزی می باشد که با تکیه بر متریک هایی که بر اساس تجزیه تحلیل ساختار برنامه انتخاب شده اند در اولویت بندی موارد تست بکار گرفته شوند. امروزه شبکه های بیزی به یک ابزار متداول برای مدلسازی مسئله های مختلف آماری مبدل شده است. یکی از مهم ترین اهداف شبکه های بیزی اجتماع عدم قطعیت با سیستم های خبره است [7]. با مدل سازی معیارهای تست نرم افزار به کمک شبکه های بیزی می توان عدم قطعیت در برآورد متریک ها و فاکتورهای تست را پوشش داد. ایده اصلی در مدل پیشنهادی جدید به کارگیری شبکه های بیزی در مدلسازی معیارها و متریک های

مهم برای اولویت بندی موارد تست است. به عبارتی در دیگر رویکردها معیارهای تست به صورت جداگانه مورد بررسی و مطالعه قرار گرفته‌اند. در این مدل به صورت یکجا و هرکدام با اولویت خاص خود که توسط متخصصان این حوزه به هر یک از متریک‌ها اختصاص می‌دهند مدل سازی شده‌اند. هدف اصلی استفاده از شبکه‌های بیزی در مدل پیشنهادی این است که بتوانیم عدم قطعیت را در برآورد متریک‌های تست نرم‌افزار مدیریت نماییم و همچنین در تصمیم‌گیری و اولویت بندی لحاظ کنیم. برخی از تکنیک‌های اولویت بندی که تا کنون پیشنهاد شده فقط بروی تست رگرسیون قابل اجرا هستند و برخی قابلیت اجرا بر روی هر دو تست رگرسیون و غیر رگرسیون را دارند. ما در این تحقیق بر روی تکنیک‌های اولویت بندی موارد تست بر اساس تجزیه و تحلیل ساختار برنامه با کمک شبکه‌های بیزی تمرکز کرده‌ایم. به منظور بهبود مقرون به صرفه بودن فعالیت‌های تست، تکنیک‌های تست همچون کاهش مجموعه تست و اولویت بندی موارد تست پیشنهاد شده‌اند. تکنیک کاهش مجموعه تست به منظور پیدا کردن موارد تست زائد و حذف آنها از مجموعه تست ارائه شده است. به طوری که مهندس تست می‌تواند با صرف کمترین زمان و منابع اجرای مجموعه تست را به حداقل برساند. اما هدف اصلی در اولویت بندی به غیر از حذف موارد تست زائد، دادن رتبه به اجرای موارد تست‌ها به جهت تشخیص خطاهای اولیه که ممکن می‌گردد است. تکنیک‌های اولویت بندی مزایایی را به دنبال دارند که عبارتند از: اول اینکه، خطاهای بیشتری را تحت شرایط محدودیت منابع و زمان پیدا می‌کنند و در نتیجه قابلیت اطمینان سیستم را بهبود می‌بخشند. دوم اینکه، چون خطاها در ابتدا به نمایش در می‌آیند و تشخیص داده می‌شوند، مهندسان تست زمان بیشتری را خواهند داشت جهت رفع این خطاها و زمانبندی پروژه نرم‌افزاری خود جهت ارائه به بازار و این در نهایت به افزایش رضایت مشتری و کیفیت محصول منجر می‌شود.

2-2- تست نرم افزار

در طول فرآیند تولید نرم‌افزار، مهندس نرم‌افزار سعی می‌کند از یک سری مفاهیم مجرد و تحلیل نیازمندی‌ها به یک پیاده سازی مشخص برسد. بعد از گام پیاده سازی، مرحله تست و آزمایش نرم‌افزار یک سری نمونه‌های تست ساخته می‌شود که هدف از اعمال آنها روی نرم‌افزار ایجاد

شرایطی است که تحت این شرایط نرم افزار درست کار نکند. در واقع فرآیند تست را می توان یک فرآیند مخرب دانست. تست نرم افزار نیاز دارد که درستی نرم افزار تولید شده را کاملاً نادیده گرفته، شرایطی را ایجاد کند تا خطاهای کشف نشده، پیدا شوند.

۲-۳- صحت و اعتبار سنجی

آزمایش نرم افزار یک عنصر از عنوان گسترده تری است که اغلب با صحت و اعتبار سنجی V&V^۴ شناخته می شود. صحت اشاره به مجموعه فعالیت هایی دارد که مطمئن می سازند نرم افزار به درستی یک تابع خاص را پیاده سازی می نماید. اعتبارسنجی اشاره به مجموعه ای متفاوت دارد که مطمئن می سازند نرم افزاری که ایجاد شده منطبق بر نیازهای مشتری است. Boehm این مطلب را اینگونه بیان می کند:

✓ صحت: آیا محصول را درست ایجاد می کنیم؟

✓ اعتبارسنجی: آیا محصول درستی را ایجاد می کنیم؟

تعریف V&V شامل بسیاری از فعالیت هایی است که تضمین کیفیت نرم افزار^۵ نامیده می شوند. صحت و اعتبارسنجی شامل گروه وسیعی از فعالیت های تضمین کیفیت نرم افزار می باشد شامل: مرورهای فنی رسمی، بررسی کیفیت و پیکر بندی، نظارت بر کارایی، شبیه سازی، امکان سنجی، مرور مستندات، مرور بانک اطلاعاتی، تحلیل الگوریتم، آزمایش توسعه، آزمایش کیفی و آزمایش نصب. اگرچه آزمایش نقش بسیار مهمی را در V&V دارد، بسیاری از فعالیت های دیگر نیز لازم می باشند [11].

^۴ verification & Validation

^۵ SQA

۲-۴- اهداف آزمایش

درمورد آزمایش نرم افزار، Myers چند قانون زیر را بیان می کند که اهداف مناسبی برای آزمایش هستند:

۱. آزمایش فرآیندی است شامل اجرای برنامه با هدف یافتن خطا.
 ۲. یک نمونه آزمایش خوب، نمونه ای است که با احتمال بالایی خطاها را بیابد.
 ۳. آزمایش موفق، آزمایشی است که خطاهای یافت نشده تاکنون را بیابد.
- این اهداف تغییری دراماتیک را در دیدگاه ایجاد می نمایند. این اهداف باعث تغییر در دیدگاه متداولی می شوند که آزمایش موفق را آن نوع آزمایش می داند که در آن خطایی یافت نشود هدف، طراحی آزمایش هایی است که به طور سیستماتیک رده های متفاوتی از خطاها را آشکار نمایند، و این عمل را با حداقل مقدار زمان و فعالیت انجام دهند.

۲-۵- اصول آزمایش

قبل از بکار گیری روش های طراحی نمونه های مؤثر آزمایش، مهندس نرم افزار باید اصول اولیه ای را که آزمایش نرم افزار را هدایت می کنند بفهمد.

Davis مجموعه ای از اصول آزمایش را پیشنهاد می کند که عبارتند از :

- تمام آزمایش ها باید بر اساس نیازهای مشتری قابل پیگیری باشند.
- آزمایش ها باید مدتی طولانی قبل از شروع آزمایش برنامه ریزی شوند.
- اصل Pareto برای آزمایش نرم افزار بکار گرفته شود.
- آزمایش باید با "توجه به اجزاء" شروع شود و به سمت آزمایش "کلی" پیش رود.
- آزمایش کامل و جامع امکان پذیر نیست.
- به منظور داشتن بیشترین تأثیر، آزمایش باید توسط تیم مستقلی هدایت شود [12].

۶-۲- قانون Pareto در فرآیند تست نرم افزار

✓ ۸۰ درصد از همه خطاهای کشف شده در طول فرآیند تست، احتمالاً در ۲۰ درصد از روالهای برنامه وجود دارند. بهتر است که این روالها را شناخته و آنها را کاملاً تست کنیم.

✓ تست باید از قطعات کوچک آغاز شود و به سمت تست قطعات بزرگتر پیش رود.

تست های اولیه روی روال های منفرد برنامه اعمال می شوند. در حالیکه فرآیند تست به پیش می رود، تمرکز روی پیدا کردن خطاها در مجموعه های یکپارچه از پیمانه ها بوده، سپس به کل سیستم منتقل می شود. [13]

۷-۲- چند نمونه از انواع تست

✓ Unit Test: برای تست از پائین ترین نقطه شروع می کنیم.

(کوچک ترین واحدهای قابل تست Unit نامیده می شوند.)

✓ Integration Test: واحدها را در کنار هم تست می کنیم.

لا System Test: تستی که روی سیستم کلی (سخت افزار، نرم افزار، DB، شبکه و ...) انجام می شود.

✓ Requirement Test: آزمون نیازهای کاربر و مشتری

۸-۲- مراحل انجام تست

✓ مرحله ی تست آلفا: تست را در محیط آزمایشگاهی انجام می دهیم.

✓ مرحله ی تست بتا: نرم افزار در محیط استفاده کننده قرار می گیرد و تست می شود.

✓ تست احراز اعتبار: تستی که نرم افزار به صورت رسمی منتشر می شود و آزمون نهایی روی آن انجام می شود.

تست جامع و کامل ممکن نیست. به خاطر زیاد بودن مسیرهای موجود در یک برنامه، اجرای همه آنها و هر الحاقی از آنها با هم در طول فرآیند تست غیرممکن است. اما امکان انجام تستی که منطق برنامه را به اندازه کافی پوشش دهد و تمام شرایط طراحی رویه ای را در بر بگیرد، وجود دارد. برای بالا بردن کارایی، عمل تست باید توسط یک گروه مستقل انجام شود. تجربه نشان داده است که مهندس نرم افزاری که سیستم را ساخته است، فرد مناسبی برای پیشبرد تست نمی باشد.

۹-۲- ویژگی های یک نرم افزار تست پذیر

1- Operability: نرم افزار هر چه بهتر کار کند، کاراتر تست می شود.

- ✓ سیستم خطاهای کمتری دارد.
- ✓ هیچ خطایی، اجرای تست را متوقف نمی کند.
- ✓ محصول در گام های عملی مشخصی به تدریج تکمیل می شود.
- 2- Observability: همان چیزی را تست کنید که می بینید.

- ✓ برای هر ورودی یک خروجی معینی تولید می شود.
- ✓ حالت های سیستم در طول اجرای آن قابل رویت هستند.
- ✓ حالت های قبلی سیستم، در طول اجرای آن قابل رویت هستند.
- ✓ همه عواملی که در خروجی تاثیر می گذارند، قابل رویت هستند.
- ✓ خروجی های نادرست به آسانی قابل تشخیص هستند.

3- Controllability: هرچه بهتر بتوان نرم افزار را کنترل کرد، خودکارسازی و بهینه سازی تست بهتر انجام می شود.

- ✓ همه خروجی های ممکن، توسط یک مجموعه معین از ورودی ها قابل تولید باشند.
- ✓ همه کد، از طریق مجموعه معینی از ورودی ها قابل اجرا باشد.
- ✓ حالات سخت افزار و نرم افزار، مستقیماً توسط مهندس تست قابل کنترل باشد.
- ✓ شکل ورودی ها و خروجی ها ساختماندار و سازگار باشد.

Decomposability -4

- ✓ سیستم نرم افزاری، از یک سری مولفه یا پیمانه های مستقل ساخته شده باشد.
- ✓ مولفه های نرم افزار، مستقلاً قابل تست باشند.

5- Simplicity: هرچه اندازه نرم افزاری که تست می شود، کوچک تر و ساده تر باشد، فرآیند تست می تواند سریع تر باشد.

6- Stability: هرچه تغییرات کمتر باشد، کار تست راحت تر انجام می شود.

- ✓ تغییرات در نرم افزار بسیار کم اتفاق بیفتد.
- ✓ تغییرات در نرم افزار کنترل شده باشد.
- ✓ تغییرات در نرم افزار، تست های تاکنون انجام شده را نامعتبر نکند. (Regression)

7- Understandability: هر چقدر اطلاعات بیشتری داشته باشیم، بهتر و دقیق تر می توانیم کار تست را انجام دهیم.

- ✓ طراحی نرم افزار، خوب درک شده باشد.
- ✓ وابستگی بین اجزای داخلی، خارجی و مشترک نرم افزار به خوبی درک شده باشد.
- ✓ مستندات تکنیکی در دسترس باشند.
- ✓ مستندات تکنیکی سازماندهی خوبی داشته باشند.

✓ مستندات تکنیکی دقیق باشند.

✓ مستندات تکنیکی، کاملاً با جزئیات همراه باشند.

۲-۱۰- ویژگی های یک تست خوب

✓ در یک تست خوب، احتمال کشف خطا بالاست.

✓ تست خوب افزونگی (در هزینه و زمان) ندارد.

✓ تست خوب نباید خیلی ساده یا خیلی پیچیده باشد. [14]

۲-۱۱- طراحی نمونه های آزمایش

طراحی آزمایش هایی برای نرم افزار و محصولات مهندسی دیگر می تواند به اندازه طراحی اولیه خود محصول متغیر باشد. با این وجود، مهندسين نرم افزار اغلب با آزمایش به عنوان فعالیتی نهایی برخورد می نمایند، و نمونه های آزمایشی طراحی می کنند که ظاهراً درست هستند اما اطمینان کمی از کامل بودن آنها وجود دارد. اهداف آزمایش را به خاطر آورید، بر اساس آنها آزمایش هایی باید طراحی شوند که احتمال بالایی برای یافتن اکثر خطاها، با حداقل مقدار زمان و فعالیت داشته باشند. مجموعه ای غنی از روش های طراحی نمونه های آزمایش برای نرم افزار تکامل یافته اند. این روش ها برای توسعه دهنده، روشی سیستماتیک را برای آزمایش فراهم می کنند. مهم تر اینکه، این روش ها مکانیزمی را فراهم می کنند که به اطمینان از کامل بودن آزمایش ها کمک می کند و احتمال بالایی برای کشف خطاهای نرم افزاری را نیز تضمین می نمایند.

هر محصول مهندسی و اکثر چیزهای دیگر می تواند به یکی از این دو روش زیر آزمایش شود:

۱. با دانستن تابع خاصی که یک محصول برای انجام آن طراحی شده، آزمایش‌هایی طراحی می‌شوند که مشخص کنند هر تابع کاملاً عملیاتی است در حالیکه در عین حال در هر تابع برای یافتن خطاها جستجو نیز انجام می‌گیرد (آزمایش جعبه سیاه).
۲. با دانستن عملکرد داخلی محصول، آزمایش‌ها به گونه‌ای طراحی می‌شوند که تعیین نماید اعمال داخلی مطابق با مشخصه‌ها انجام می‌شوند و تمام مؤلفه‌های داخلی به طور مناسبی آزمایش می‌گردند (آزمایش جعبه سفید). [15]

۲-۱۱-۱ تست جعبه سیاه^۷

تست با هدف نشان دادن این که هر وظیفه‌مندی به درستی انجام می‌شود و نیز یافتن خطاها در زمینه وظیفه‌مندی را تست جعبه سیاه می‌گویند.

۲-۱۱-۲ تست جعبه سفید^۸

تست کردن برای اطمینان از این که اعمال داخلی نرم‌افزار مطابق با مشخصات موردنظر کار می‌کنند و تمام مؤلفه‌های داخلی مورد امتحان قرار گرفته‌اند، تست جعبه سفید نامیده می‌شود. در واقع این نوع تست، امتحان دقیق جزئیات روالی برنامه می‌باشد که در آن مسیرهای منطقی با ایجاد موارد تستی که مجموعه‌ای از شرایط و حلقه‌ها را بررسی می‌کنند، تست می‌شود.

۲-۱۱-۳ آزمایش ساختار کنترل

تکنیک آزمایش مسیر پایه توصیف شده، یکی از چند تکنیک آزمایش ساختار کنترلی است. اگرچه آزمایش مسیر پایه ساده و بسیار مفید است، ولی به تنهایی کافی نیست. این حالت‌ها پوشش آزمایش را گسترش داده و کیفیت آزمایش جعبه سفید را افزایش می‌دهند.

⁷ Black Box

⁸ White Box

۲-۱۱-۴ آزمایش واحد

آزمایش واحد، بر فعالیت بازبینی کوچک ترین واحد طراحی نرم افزار، که مؤلفه نرم افزار یا پیمانه نامیده می شود تمرکز دارد. با استفاده از توصیف طراحی در سطح مؤلفه به عنوان راهنما، مسیرهای کنترلی مهم آزمایش می شوند تا خطاهای موجود در مرز پیمانه پیدا شوند. پیچیدگی نسبی آزمایش ها و خطاهای آشکار شده، بامحدوده ایجاد شده برای آزمایش واحد، محدود می شود. آزمایش واحد، گرایش به جعبه سفید دارد، و این مرحله می تواند به موازات برای چند مؤلفه هدایت شود.

۲-۱۱-۵ خطاهای متداول محاسبه که اغلب مشاهده می شوند

۱. اولویت محاسباتی نادرست
 ۲. اعمال ترکیبی
 ۳. آماده سازی غلط
 ۴. عدم دقت کافی در محاسبه
 ۵. نمایش غلط یک نماد محاسباتی
- مقایسه و کنترل جریان تا حد زیادی به یکدیگر نزدیک هستند (یعنی تغییر جریان، معمولاً بعد از مقایسه انجام می شود).
- نمونه های آزمایش باید خطاهایی را از این قبیل آشکار نمایند:
۱. مقایسه انواع داده متفاوت
 ۲. عملگرهای منطقی یا اولویت نادرست
 ۳. داشتن انتظار تساوی در زمانی که خطا در دقت محاسبه باعث عدم تساوی می شود
 ۴. مقایسه غلط متغیرها
 ۵. خاتمه حلقه نامناسب یا عدم وجود خاتمه حلقه
 ۶. شکست در خروج، زمانی که حلقه نامتناهی تشخیص داده می شود

۲-۱۲- آزمایش یکپارچه سازی^۹

یک فرد مبتدی در دنیای نرم افزار ممکن است سؤالی به ظاهر ساده را بعد از اینکه آزمایش واحد بر روی تمام پیمانه‌ها صورت گرفت پرسد: “اگر همه آنها به تنهایی کار می کنند، چرا مشکوک هستید که آنها و قتی کنار هم قرار گیرند کار می کنند یا خیر؟” این مشکل، کنار هم قرار دادن مؤلفه ها، یعنی ارتباط بین آنها است. داده ممکن است در یک ارتباط ناپدید شود. یک پیمانه ممکن است اثر نامطلوب و ناخواسته ای را بر دیگری داشته باشد. زیر توابع، زمانی که با هم ترکیب می شوند، ممکن است تابع بزرگ تر مورد نظر را تولید نکنند. مقادیر نا دقیقی که در هر یک به تنهایی پذیرفته شده اند، ممکن است بزرگ شوند و به سطوح غیر قابل قبول برسند. ساختمان داده‌های سراسری ممکن است مشکل ساز شود، این لیست همچنان به طور نگران کننده ای ادامه دارد. آزمایش یکپارچه سازی، روشی سیستماتیک برای ایجاد ساختار برنامه است درحالیکه، آزمایش‌ها نیز انجام می شوند تا خطاهای مربوط به واسطه‌ها آشکار شوند. هدف، دریافت مؤلفه‌های آزمایش واحد و ایجاد ساختار برنامه ای است که توسط طراح دیکته شده است.

۲-۱۳- آزمایش رگرسیون (Regression Testing)

تجربه نشان داده است که به عنوان نرم افزار ثابت یا تکمیل شده، ظهور جدید یا ظهور مجدد خطاهای قدیمی کاملاً رایج است. گاهی اوقات ظهور مجدد رخ می دهد به دلیل اینکه رفع آنها از طریق شیوه‌های کنترل ضعیف یا خطاهای ساده انسانی در کنترل روی می دهند. تست رگرسیون به طور قابل توجهی از سوی دانشگاهیان و افراد شاغل در زمینه تست نرم افزار در طول بیست سال گذشته کانون توجه قرار گرفته است. حتی استفاده از تکنیک های تست رگرسیون اغلب منجر به

^۹ Integration Testing

محصول نرم افزاری با کیفیت بالا شده است، اما اجراهای مکرر موارد تست می تواند بسیار پر هزینه باشد که نیمی از هزینه نگهداری سیستم نرم افزاری را به خود اختصاص می دهد. هردفعه که پیمانه جدیدی به عنوان بخشی از آزمایش یکپارچه سازی افزوده می شود، نرم افزار تغییر می کند. مسیرهای جریان داده ای ایجاد می شوند، I/O جدیدی ایجاد می گردد، و منطق کنترل جدیدی فراخوانی می شود. این تغییرات ممکن است باعث بروز مشکلاتی با توابعی شوند که قبلاً بدون خطا کار می کردند. در رابطه با استراتژی آزمایش یکپارچه سازی، آزمایش رگرسیون، اجرای مجدد زیر مجموعه ای از آزمایش هایی است که قبلاً انجام شده اند تا اطمینان حاصل شود که تغییرات، باعث انتشار اثرات جانبی ناخواسته نشده اند. در محدوده وسیع تر، آزمایش های موفقیت آمیز (ازهرنوع) باعث کشف خطاها می شوند، و خطاها باید اصلاح شوند. هر زمانیکه نرم افزار اصلاح می شود، جنبه ای از پیکر بندی نرم افزار (برنامه، مستندات، یا داده هایی که آنها را حمایت می کنند) تغییر می نماید. آزمایش رگرسیون می تواند به صورت دستی هدایت شود. این عمل با اجرای مجدد زیر مجموعه ای از تمام نمونه های آزمایش، با استفاده از ابزارهای خودکار Capture/Playback انجام شود. ابزارهای Capture/Playback باعث می شوند مهندس نرم افزار نمونه های آزمایش را دریافت کند و با حرکت به عقب، مقایسه هایی را انجام دهد (16).

مجموعه آزمایش رگرسیون (زیرمجموعه ای از آزمایش هایی که باید انجام شوند) شامل سه رده متفاوت از نمونه های آزمایش می باشد :

- ✓ نمونه هایی از آزمایش هایی که تمام عملکرد نرم افزار را بررسی می نمایند.
- ✓ آزمایش های اضافی که بر عملکردهایی از نرم افزار تأکید دارند که احتمالاً با این تغییرات تحت تأثیر قرار می گیرند.
- ✓ آزمایش هایی که بر مؤلفه های تغییر یافته در نرم افزار تأکید دارند.

هدف از انجام تست رگرسیون به همراه اولویت بندی افزایش میزان نرخ تشخیص خطا می باشد. برای برآورده شدن این هدف چندین عامل دیگر را نیز بایستی مد نظر قرار دهیم. یکی افزایش

میزان نرخ تشخیص خطا باید به گونه ای باشد که در یک زمان معقول (مقرون به صرفه) انجام گیرد و دیگر نیازی به استفاده از مدل های اقتصادی هزینه - فایده برای بررسی مقرون به صرفه بودن نداشته باشیم. زمان فعالیت های تست را کاهش دهیم تا اینکه بر میزان رضایت مندی مشتری و کیفیت رویکرد ما افزوده شود. از طرفی با کاهش دادن زمان مورد نیاز برای انجام فعالیت های مربوط به تست رگرسیون و ارائه نتایج میزان قابل توجهی از هزینه و منابع مورد نیاز جهت استفاده کاهش می یابد. حال برای اینکه در فرایند تست رگرسیون نسبت به دیگر تکنیک ها بتوانیم نرخ تشخیص خطا را در کمترین زمان ممکن انجام دهیم ، افزایش رضایتمندی مشتری ، کاهش هزینه و منابع لازم جهت انجام تست و افزایش کیفیت محصول را به دنبال خواهد داشت .

در این پایان نامه ما به دنبال این هستیم که با بکارگیری معیارهای مهم تست و استفاده از شبکه های بیزی بتوانیم رویکردی کارآمدتر از دیگر روش های پیشنهاد شده قبلی جهت اولویت بندی موارد تست ارائه کنیم.

۱۴-۲- متدولوژی های مربوط به تست رگرسیون

در این بخش چهار روشی را که به تست رگرسیون مربوط اند بررسی می کنیم و هر یک را جداگانه شرح می دهیم. این روش ها عبارتند از :

۱- Retest all اجرای مجدد همه ی تست ها

۲- Regression Test Selection انتخاب تست رگرسیون

۳- Test Suite Reduction کاهش مجموعه تست

۴- Test Case Prioritization اولویت بندی موارد تست

۱-۱۴-۲- اجرای مجدد همه‌ی تست‌ها (Retest all)

در این روش آندسته از موارد تستی که دیگر برای نسخه اصلاح شده برنامه کاربرد ندارند دور انداخته می‌شوند، و تمام مجموعه‌ی موارد تست باقیمانده برای تست برنامه‌ی تغییر یافته استفاده می‌شوند.

۲-۱۴-۲- انتخاب تست رگرسیون (Regression Test Selection)

روش اجرای همه‌ی موارد تست هم زمان زیادی طول می‌کشد و هم تلاش زیادی باید صرف انجام آن شود برای تست دوباره‌ی برنامه، بدین صورت ممکن است خیلی گران شود هزینه‌ی این روش. این روش بهتر است به همراه اطلاعات در مورد برنامه، تغییر برنامه و موارد تست استفاده بشود برای انتخاب زیر مجموعه‌ای از موارد تست جهت تست کردن.

۳-۱۴-۲- کاهش مجموعه تست (Test Suite Reduction)

این روش با استفاده از اطلاعات مربوط به برنامه و مجموعه تست برای حذف موارد تست به عنوان قابلیت جدید اضافه شد و با گذشت زمان کنار گذاشته شد. این روش با روش انتخاب تست رگرسیون تفاوت دارد، در روش قبلی به طور دائم موارد تست را حذف می‌کردید، اما انتخاب شده‌ها مورد نیازند. مزیت این روش این است که هزینه‌ی اعتبار سنجی، اجرا، مدیریت مجموعه‌های تست روی نسخه‌های جدید نرم افزار را در آینده کاهش می‌دهد، این حرکت نزولی است که ممکن است تشخیص خطا را کاهش دهد با قابلیت کاهش اندازه مجموعه تست.

۴-۱۴-۲- اولویت بندی موارد تست (Test Case Prioritization)

در این تکنیک به هر کدام از موارد تست یک اولویت اختصاص داده می‌شود. برای مثال معیار برای اولویت بندی می‌تواند پوشش کد سریع موارد تست باشد که بالاترین اولویت را داراست. مزیت این تکنیک نسبت به تکنیک های قبلی این است که موارد تست را دور نمی اندازید و یا دائم موارد تست را از مجموعه ی تست حذف نمی کنید. یکی دیگر از معیارهای ممکن برای اختصاص اولویت به موارد تست می‌تواند نرخ تشخیص خطا باشد.

۴-۱۵-۲- اولویت بندی

۴-۱۵-۱- مقدمه ای بر اولویت بندی

کلمه "اولویت بندی" به معنی قرار دادن وظایف به بهترین نحو به طوریکه انجام بهتر و کامل وظیفه و به موقع آن پیش از نیاز واقعی میسر باشد. یکی از جنبه‌های مهم تست رگرسیون اولویت بندی موارد آزمون است که نیاز دارد به اجرای دستورات بر اساس معیارهای خاص. اولویت بندی موارد آزمون با اهداف گوناگون در فرایند تست در مطالعات بکار رفته است. برخی از این اهداف عبارتند از: افزایش میزان یا نرخ تشخیص خطا، تست رگرسیون مقرون به صرفه از طریق چینش موارد تست به طوری که آنهایی که از اولویت بالاتری برخوردارند در فرآیند تست زودتر انجام شوند و کم کردن زمان و هزینه لازم برای انجام فرایند مهم تست نرم افزار به منظور شناسایی و کشف خطا در کمترین زمان و صرف هزینه کمتر برای رفع آن. مطالعات انجام شده و نتایج بدست آمده از آنها نشان می‌دهند که با بهره گیری از تکنیک های اولویت بندی می‌توانیم تست رگرسیون را بهبود دهیم. در این راستا هم تکنیک های اولویت بندی پیشنهاد شده‌اند و شواهد و نتایج حاصل نشان می‌دهند که بکار گیری این تکنیک ها در تست رگرسیون مفید است.

اما نکته مهم اینکه بیشتر تکنیک های اولویت بندی که تا کنون پیشنهاد شده اند برای خطاهای کشف شده شدت یکسان و برابری را در عمل در نظر می گیرند. به دلیل اینکه بیشتر تکنیک های اولویت بندی تکیه دارند بر روی اطلاعات جمع آوری شده از اجرای موارد تست های قبلی نرم افزار یا تغییر کد منبع. به ندرت تکنیک هایی پیشنهاد شده اند که بتوانند استفاده شوند برای تست غیر رگرسیون. این تکنیک ها کمک می کنند به مهندسان با نشان دادن خطاها در اوایل تست کردن، همچنین این تکنیک ها به آنها اجازه میدهد که شروع کنند به اشکال زدایی زودتر از زمان ممکن. در این مورد، کل مجموعه آزمون هنوز هم ممکن است اجرا شود، اجتناب از اشکالات بالقوه در ارتباط با حذف موارد آزمون، و صرفه جویی در هزینه از رسیدن به موازی بیشتر از اشکال زدایی و فعالیت های آزمایش. تست رگرسیون یک فرآیند گران قیمت است که برای اعتبار سنجی نرم افزار اصلاح شده یا تغییر یافته استفاده می شود. تکنیک های اولویت بندی موارد تست، بهبود مقرون به صرفه ای از تست رگرسیون با مرتب سازی موارد آزمون به طوری که آنها یکی از اهمیت بیشتری برخوردار هستند پیش از دیگر موارد تست با اهمیت کمتر در فرایند تست اجرا شوند. با این حال، این محدودیت های زمانی هستند که می توانند بر تست رگرسیون تحمیل شوند و توسط فرآیندهای توسعه نرم افزار به شدت بر رفتار تکنیک های اولویت بندی تاثیر بگذارند. اگر این درست باشد درک بهتر از اثرات محدودیت های زمانی میتواند منجر به بهبود تکنیک های اولویت بندی، و تعمیر و نگهداری و بهبود فرآیندهای تست نرم افزار شود. تکنیک های اولویت بندی موارد تست نشان داده اند که برای بهبود فعالیت تست رگرسیون مفید هستند. با اولویت بندی موارد تست، نرخ تشخیص خطا بهبود یافته است، در نتیجه اجازه می دهد به آزمایش کنندگان برای شناسایی خطاها پیش از انجام مرحله تست سیستم. بسیاری از تکنیک های اولویت بندی قدیمی بر پوشش کد مبتنی اند، و لذا ممکن است این تکنیک ها با تمام خطاها به طور مساوی و یکسان رفتار کنند ولی در واقع نباید بدین صورت باشد.

2-15-2- معیارهای اولویت دهی

معیارهای اولویت دهی در رویکرد ما بر اساس ساختار برنامه است و باعث این می شود که مانند برخی دیگر از روش های پیشنهاد شده برای خطاهای شناسایی شده شدت یکسانی در نظر نگیریم و همچنین تکیه بر اطلاعات قبلی از اجراهای موارد تست نداشته باشیم، که اینها از برتری های رویکرد ما در مقایسه با دیگر رویکردهایی است که فقط برای تست رگرسیون بکار می روند به حساب می آید و ما می توانیم آن را هم برای تست توسعه نرم افزار و هم تست رگرسیون بکار ببریم.

۲-۱۵-۳- اولویت بندی موارد تست

تکنیک های اولویت بندی موارد تست شامل برنامه ریزی بروی موارد تست با یک نظم است که باعث بهبود عملکرد تست رگرسیون می گردد. و زمانی تست رگرسیون نا کارآمد خواهد بود که اجرای هر مورد تست برای هر تابع از برنامه در صورت رخ دادن هر تغییری بخواهد اجرا گردد. در این مواقع است که ضرورت استفاده از اولویت بندی موارد تست احساس می شود. تکنیک های اولویت بندی موارد تست، سازماندهی موارد تست درون یک مجموعه تست با مرتب سازی است به گونه ای که با اولین اجرا نرخ تشخیص خطا را به طور اثربخشی بهبود دهد.

۲-۱۶- متریک

۲-۱۶-۱- مقدمه ای برای متریک

تست نرم افزار فرآیندی است که طی آن خطاهای یک برنامه ی کاربردی کشف و رفع می شود. اما روند تست نرم افزار تنها محدود به یافتن خطاهای برنامه نمی شود، تست برای تعیین میزان کیفیت برنامه و همچنین میزان سازگاری و پاسخگویی آن به نیازهای کاربران و مشتریان است. متریک های تست نرم افزار نا محدودند بنابر این تست کننده باید با ویژگی ها و کاربرد هر یک به درستی آشنا باشد تا بتواند متریک های مناسب با کاربرد خود را انتخاب نماید .

خودکارسازی تست ها نسبت به تست دستی برتری های زیادی دارد. خودکارسازی تست ها باعث می شود که توسعه دهندگان زمان بیشتری برای تمرکز بر روی دیگر جنبه ها داشته باشند و بتوانند خطاهای نرم افزار را به صورت مؤثرتری رفع نمایند. علاوه بر این، از آنجاییکه تست ها را می توان در هر زمان و به هر تعداد دفعاتی اجرا کرد، می توان از تست های قبلی استفاده ی مجدد نمود. و به این ترتیب کارایی تست را افزایش و زمان تست را کاهش داد. خودکارسازی تست نرم افزار زحمت و پیچیدگی انجام تست را کاهش می دهد. برای بررسی کیفیت و چگونگی روند انجام تست معیارها یا متریک هایی وجود دارد؛ که بر اساس آنها می توان تصمیم گیری هایی در مورد وضعیت نرم افزار تحت تست، زمان مناسب برای انتشار نرم افزار، میزان ریسک در انتشار نرم افزار انجام داد. متریک های تست بسیار مختلف و متنوع هستند و تعداد آنها بسیار زیاد و نا محدود است. برای انتخاب متریک های مناسب با کاربرد خاص مورد نظر ما باید اطلاعاتی در مورد متریک های مختلف و موارد استفاده ی آنها داشته باشیم.

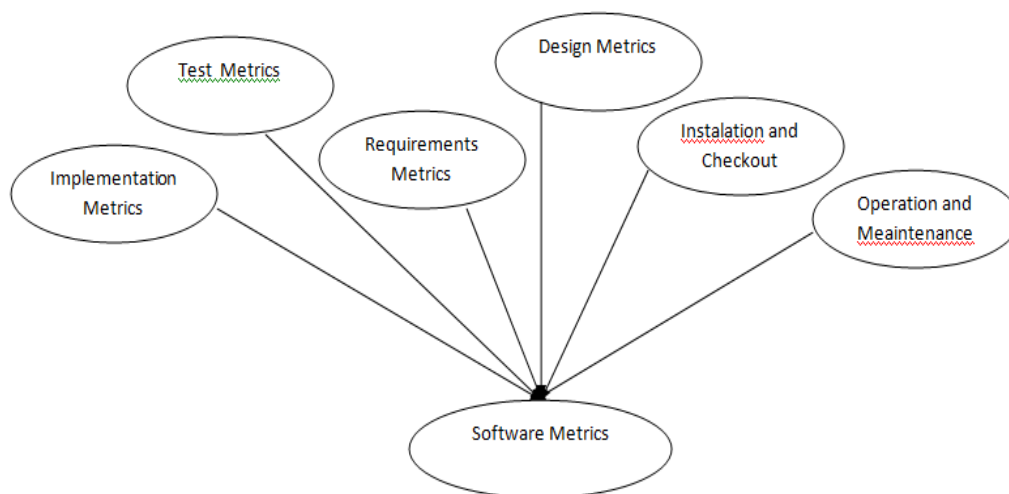
۲-۱۷- متریک های تست نرم افزار

متریک ها با عنوان استانداردهایی برای اندازه گیری تعریف می شوند. و مدتی طولانی است که در صنعت IT برای نشان دادن روشی برای اندازه گیری میزان سودمندی و کارایی یک فعالیت مشخص در یک پروژه، استفاده می شوند. متریک ها راه هایی برای پاسخ به سؤالات هستند. بدون دانستن سؤالاتی که باید پاسخ داده شوند، تلاش برای پاسخ دادن وقت تلف کردن است. برای مشخص کردن متریک هایی که برای هر پروژه نیاز دارید، از روش بالا به پایین عمل نمایید. از آنجاییکه تعداد نا محدودی از متریک های ممکن وجود دارد، برنامه نویسان باید معیارهایی برای انتخاب متریک هایی که باید روی یک پروژه مشخص اعمال شوند، داشته باشند. به طور ایده آل یک متریک باید خواص زیر را داشته باشد.

۱۷-۱-۲- خواص متریک ها در شرایط ایده آل

- ساده و قابل محاسبه باشند.
- از نظر تجربی و شهودی ترغیب کننده باشند.
- سازگاری و دست یافتنی
- سازگاری در به کارگیری واحدها و ابعاد
- برنامه ریزی زبان مستقل
- یک مکانیسم مؤثر برای بازخورد با کیفیت

در شکل (۱-۲) یک دسته بندی کلی از متریک ها را در گروه های مختلف موجود در حوزه نرم افزار را به تصویر کشیده ایم، هر کدام از این گروه های مشخص شده در این شکل از متریک ها و معیارهای خاص خود تشکیل شده اند. برای انجام تست بر روی هر کدام از موارد ذکر شده بایستی متریک های مرتبط به آن را بشناسیم و استخراج کنیم تا بتوانیم یک تست با کیفیت و مطلوبی را انجام دهیم.



شکل ۱-۲: گراف متریک های کلی نرم افزار

هدف ما از متریک ها این است که بتوانیم اولویت بندی کارآمدتری را با در نظر گرفتن معیارهای مختلف و استفاده از شبکه های بیزی ارائه نماییم. محصولات نرم افزاری برای کشف و رفع باگ ها و

همچنین تعیین کیفیت نیاز به تست دارند. روند تست نرم افزار گاهی می تواند به اندازه ی توسعه ی نرم افزار انرژی برده و با اهمیت باشد. تست نرم افزار شامل فرایند اجراهای متعدد برنامه با هدف یافتن باگ های نرم افزاری است اما محدود به آن نمی باشد، تضمین اینکه تا چه حد نیازمندی های موجود را برآورده می کند و آیا با انتظارات مشتری سازگار است یا خیر نیز توسط فرایند تست مشخص می شود. معیارهای تستی وجود دارد که به وسیله ی آنها می توان میزان کیفیت برنامه، میزان سازگاری و پاسخگویی آن به نیازهای کاربران و مشتریان را تعیین نمود. این متریک ها به دو دسته تقسیم می شوند:

✓ متریک های محصول نرم افزاری

✓ متریک های پروسه ی تست

متریک های محصول نرم افزاری شامل متریک هایی می شود که مرتبط با نتایج آزمون یا کیفیت نرم افزاری است که در حال تست شدن می باشد. از جمله ی این متریک ها به متریک دقت، امنیت، قابلیت تست و تخمین باگ ها در سیستم می توان اشاره نمود.

متریک های پروسه تست شامل متریک هایی می شود که برای ارزیابی سودمندی پروسه ی آزمون استفاده می گردند. از متریک هایی که در این دسته قرار می گیرند به توصیف متریک های مورد تست، متریک های خرابی و متریک های قابلیت استفاده نیز می توان اشاره کرد. از نظر مهندسان نرم افزار نوشتن کدهای تست، به خودی خود، مثل توسعه خود محصول وقتگیر و گران است. با افزایش حجم و پیچیدگی نرم افزارها، تست کردن دستی آنها کاری بسیار سنگین و طاقت فرسا و گاه غیرممکن می باشد. بنابراین خودکارسازی تست راه حلی است که کارهای سنگین و پر زحمت تست را ساده می نماید. خودکارسازی تست زمانی مؤثر است که شرایط نرم افزار تحت تست مناسب باشد. باید توجه داشت که تست خودکار به این معنا نیست که کل فرایند تست نرم افزار به صورت خودکار انجام می شود، بلکه به معنای تست نرم افزار با کمک کامپیوتر است. استفاده از تست

خودکار پر هزینه است. به کاربردن تست خودکار به این معنی نیست که دیگر نیازی به تست دستی نداریم بلکه تست خودکار مکملی برای فرآیند تست موجود می باشد.

۲-۱۸- معیار و متریک در تست نرم افزار

اصطلاح استفاده شده برای توصیف اندازه گیری یک ویژگی خاص از یک پروژه نرم افزاری، متریک نرم افزار نام دارد. مهندسین از اعداد برای طراحی استفاده نموده و به وسیله آن محصول را ارزیابی می کنند. اما مهندسین نرم افزار از سیستم متریک برای ایجاد نرم افزار با کیفیت استفاده می کنند. سیستم متریک فنی، پایه ای را برای تحلیل، طراحی، برنامه نویسی و آزمون ارائه می دهد و سبب می شود که آزمون به طور ملموس تری انجام شده و نتیجه به صورت کمیتی ارزیابی گردد. متریک های نرم افزاری یک روش مقداری برای ارزیابی کیفیت خصلت های درونی ارائه می کنند و مهندسین را قادر می سازد تا ارزیابی کیفیت را قبل از تولید محصول انجام دهند. پس با استخراج متریک های مناسب و مهم می توانیم گامی موثر در رسیدن به کیفیت برداریم.

۲-۱۸-۱- مراحل انجام کار در فرایند اندازه گیری

۱. کسب مقیاس های نرم افزار
۲. رابطه کردن داده ها
۳. تحلیل متریک مناسب
۴. تحلیل نتایج به دست آمده
۵. حصول شناخت از کیفیت محصول نرم افزاری [17]

۱۹-۲- متریک های آزمون

- ✓ اکثر متریک ها بر روی فرآیند آزمون متمرکز شده و به خصوصیات خود آزمون توجه نکرده اند .
- ✓ آزمون کننده ها باید بر روی تحلیل، طراحی و متریک های برنامه جهت طراحی و انجام موارد آزمون اتکا داشته باشند .
- ✓ متریک بنگ تعداد موارد آزمون مورد نیاز را از طریق آزمون مقیاس های اولیه ارائه می کند .
- ✓ متریک های طراحی معماری، اطلاعاتی را در مورد راحتی یا سختی مرتبط با آزمون جامعیت و تمامیت ارائه کرده است .

۲۰-۲- مزایای استفاده از متریک ها

- ✓ متریک ها شناخت لازم برای ایجاد مدل های موثر تحلیل و طراحی و آزمون های جامع فراهم می کنند .
 - ✓ متریک ها برای اینکه مفید واقع شوند باید ساده و قابل محاسبه باشند .
 - ✓ متریک ها باید ملموس و فهمیدنی باشند .
 - ✓ مستقل از زبان برنامه نویسی بوده و بازخورد موثری را فراهم نماید .
 - ✓ متریک های مدل تحلیل روی کارکرد، داده ها و رفتار متمرکز هستند . [18]
- یک چیز مهم و قابل تحویل از سوی تیم تست، اطلاعات است. که این اطلاعات وسیله انتقال پیشرفت کار تست و وضعیت فعالیت های تست در طول پروژه را نشان می دهد. این اطلاعات می تواند شامل چیزهای زیادی باشد به عنوان مثال : متریک ها، آمارها و خلاصه های گرفته شده از گراف ها، نمودارها و گزارش های نوشته شده در طول انجام پروژه نرم افزاری. افراد تیم های مختلف در پروژه های مهندسی نرم افزار به این اطلاعات مهم و ارزشمند نیاز دارند تا اینکه از فعالیت ها، محصولات خود حمایت کنند. با بکار گرفتن این اطلاعات در تصمیم گیری های خود می توانند به نتایج مطلوب تری دست یابند. نکته مهم در این خصوص این است که این اطلاعات ارائه شده از

سوی تیم تست برای تمام افراد تیم توسعه نرم افزاری مفید است و باید به موقع و به طور مداوم منتشر گردند.

۲-۲۱- شبکه‌های بیزی

۲-۲۱-۱- استنتاج با استفاده از توزیع توام کامل

ساده‌ترین روش استنتاج از توزیع توام کامل^۱ استفاده می‌کند که با یک مثال ساده آن را توضیح می‌دهیم. یک قلمرو را در نظر بگیرید که تنها شامل سه متغیر بولی Cavity، Toothache و Catch می‌باشد. Toothache نشان دهنده دندان درد، Cavity نشان دهنده سوراخ بودن دندان و Catch نشان دهنده گیر کردن ابزار دندان پزشک در دندان می‌باشد. توزیع توام کامل یک جدول $2 \times 2 \times 2$ است که در جدول (۲-۱) دیده می‌شود.

جدول ۲-۱: توزیع یک قلمرو ساده.

Toothache		$\neg toothache$	
catch	$\neg catch$	catch	$\neg catch$

^۱ full joint distribution

Cavity	0.108	0.012	0.072	0.008
$\neg cavity$	0.016	0.064	0.144	0.576

با استفاده از این جدول می‌توان احتمال هر رویداد دلخواه را محاسبه کرد. مثلاً احتمال رویداد

$cavity \vee toothache$ را توسط رابطه (۲-۱) محاسبه می‌کنیم:

(۱-۲)

$$P(cavity \vee toothache) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$$

همانطور که قبلاً گفته شد، در بسیاری موارد محاسبه احتمال شرطی یک رویداد در حالی که بعضی از مشاهدات داده شده است، مطلوب می‌باشد. احتمال شرطی رویدادها با استفاده از رابطه احتمال شرطی و جمع عناصری از جدول توزیع آنها قابل محاسبه است. به عنوان مثال، محاسبه احتمال یک سوراخ در دندان در صورت وجود دندان درد، به وسیله رابطه (۲-۲) محاسبه می‌شود:

$$P(cavity|toothache) = \frac{P(cavity \wedge toothache)}{P(toothache)} = \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6 \quad (۲-۲)$$

به عنوان مثال دیگر می‌توان احتمال سوراخ نبودن دندان با وجود دندان درد را نیز توسط رابطه (۲-۳) محاسبه کرد:

(۳-۲)

$$P(\neg cavity|toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)} = \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

توجه کنید که در این محاسبات $1/P(toothache)$ بدون توجه به اینکه چه مقداری برای Cavity محاسبه کنیم، ثابت باقی می‌ماند. در واقع، می‌توان آن را یک عامل برای نرمال‌سازی فرض کرد تا توزیع $P(cavity|toothache)$ مجموعی برابر ۱ داشته باشد. در طول این بررسی از α برای نشان

دادن چنین ثابت‌هایی استفاده می‌کنیم. با این نماد، دو عبارت قبلی را در رابطه (۴-۲) به صورت زیر خلاصه کرد:

(۴-۲)

$$P(\text{Cavity} | \text{toothache}) = \alpha P(\text{Cavity}, \text{toothache}) = \alpha < 0.12, 0.08 > = < 0.6, 0.4 >$$

با تعمیم روش بکار رفته در این مثال به راحتی می‌توان به یک رویه استنتاج کلی دست یافت. در اینجا، حالتی که تنها یک متغیر در درخواست وجود دارد را بررسی می‌کنیم (حالت‌های دیگر به سادگی از تعمیم این حالت بدست می‌آیند). فرض کنید X متغیر درخواست (Cavity در مثال) باشد، E مجموعه متغیرهای مربوط به شواهد را نشان دهد (Toothache در مثال)، e مقادیر مشاهده شده برای E را نشان دهد و Y مجموعه بقیه متغیرهای مشاهده نشده باشد (Catch در مثال). با توجه به این فرضیات، درخواست‌ها در حالت کلی به صورت $P(X | e)$ هستند و می‌توان به صورت رابطه (۵-۲) محاسبه کرد:

$$P(X | e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y) \quad (5-2)$$

که مجموع بر روی تمامی مقادیر y ممکن است (یعنی تمامی حالات مقادیر متغیرهای Y). توجه کنید که X و E و Y با هم تمامی متغیرهای قلمرو را تشکیل می‌دهند و $P(X, e, y)$ یک زیرمجموعه از عناصر جدول توزیع می‌باشد و بنابر این، مجموع به راحتی قابل محاسبه است. [19]

۲-۲۱-۲- مشکلات استنتاج با توزیع توام کامل و راه حل آنها

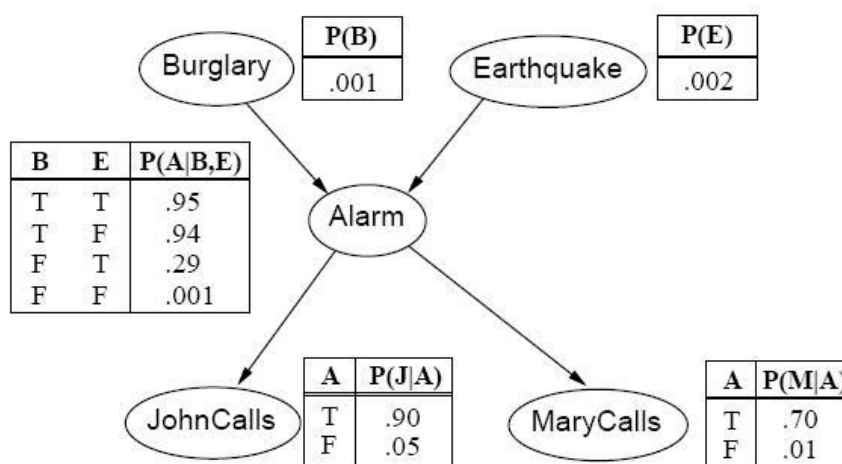
روش استنتاج با توزیع توام کامل سه مشکل اساسی دارد. اولاً، برای متغیرهای پیوسته اگر آنها یک توزیع خاص و مشخصی نداشته باشند، نیاز به یک جدول نامتناهی است. دوماً، حتی برای n متغیر بولی هم جدول توزیع 2^n خانه دارد و جواب به یک درخواست زمان $O(2^n)$ نیاز دارد. سوماً، احتمالات مربوط به توزیع را باید از داده‌های آماری تخمین زد که برای تعداد نمایی خانه در جدول نیاز به داده‌های بسیار زیادی است که در عمل موجود نمی‌باشد. برای حل این مشکلات سعی می‌شود که شکل ساده‌تری برای توزیع فرض شود تا بدست آوردن آن عملی باشد. یکی از فرض‌هایی که بسیار استفاده شده فرض مستقل بودن متغیرهای قلمرو بوده است که در عمل

نتیجه‌های خوبی نیز از آن گرفته شده است. این فرض بسیار محدود کننده است. به همین دلیل سعی شد این محدودیت تا حدی برداشته شود و وابستگی متغیرها تا حدی در محاسبه توزیع در نظر گرفته شود. براین اساس فرض وابستگی درختی و وابستگی به صورت گراف‌های جهت‌دار بدون دور^{۱۱} مورد بررسی قرار گرفت. با فرض وابستگی به صورت گراف‌های جهت‌دار بودن دور به مدلی از توزیع می‌رسیم که به شبکه‌های بیزی معروف است.

۳-۲۱-۲- مثالی از شبکه‌های بیزی

حال به توضیح مطالب ذکر شده با یک مثال می‌پردازیم. فرض کنید شما به تازگی یک آژیر دزدی خریده‌اید که در صورت وقوع دزدی، امکان زیادی دارد که به صدا در آید. علاوه بر این، در صورت وقوع زلزله‌های ضعیف هم ممکن است که آژیر به صدا درآید. همچنین، شما دو همسایه به نام‌های ماری و جان دارید که قول داده‌اند در صورت شنیدن صدای آژیر با شما در محل کارتان تماس بگیرند. جان همیشه در صورت به صدا درآمدن آژیر به شما تلفن می‌زند. ولی ممکن است که صدای تلفن شما را گاهی با آژیر اشتباه بگیرد و دراین صورت هم تلفن بزند. ماری موسیقی را با صدای بلند دوست دارد و ممکن است گاهی صدای آژیر را نشنود. حالا با توجه به مشاهده اینکه چه کسی تلفن زده است یا نزده است، می‌خواهیم احتمال دزدی را محاسبه کنیم. شبکه بیزی این قلمرو در شکل (۲-۲) آمده است. [20]

¹ Directed Acyclic Graph



شکل ۲-۲: شبکه بیزی قلمرو دستگاه آژیر [20]

در ابتدا، ساختار این شبکه را توضیح می‌دهیم و بعد در مورد جدول‌های احتمالات شرطی توضیحاتی می‌دهیم. با توجه به شبکه می‌توان دید که دزدی و زلزله به طور مستقیم بر روی به صدا درآمدن آژیر تاثیر می‌گذارند، ولی بر روی تلفن زدن جان یا ماری تاثیر ندارد، زیرا آنها تنها با شنیدن آژیر تلفن می‌زنند که این موضوع در دو یال خارج شده از گره مربوط به آژیر در شبکه معلوم است. بنابراین شبکه نشان می‌دهد که آنها از وقوع زلزله خفیف آگاه نمی‌شوند.

باید توجه داشت که گره‌ای برای نشان دادن اینکه ماری به موسیقی گوش می‌دهد و یا تلفن زنگ می‌زند در شبکه نیست. این موارد در عدم قطعیت (احتمالی بودن) مربوط به یال‌های خروجی از گره به صدا درآمدن آژیر تاثیر داده شده است. به طور کلی روشی برای اینکه به چه عواملی باید در شبکه گره اختصاص داد، وجود ندارد. در واقع، احتمالات شرطی، تمامی عواملی که در شبکه صریحاً نیامده‌اند، را به طور خلاصه در خود دارند. به این طریق، یک عامل^۲ ساده می‌تواند به طور تقریبی با یک دنیای پیچیده را مدل کند. میزان این تقریب را می‌توان با اضافه کردن اطلاعات مرتبط به شبکه افزایش داد. توزیع‌های شرطی گره‌ها را با جدول‌های احتمالات شرطی نشان می‌دهند (البته اگر توزیع‌ها پیوسته باشد از روش‌هایی دیگری استفاده می‌شود که بعداً به آنها اشاره خواهد شد). هر

^۱ agent

سطر در این جدول‌ها نشان دهنده مقدار احتمال مقادیر متغیر راس برای یک حالت شرطی خاصی می‌باشد. هر حالت شرطی یکی از مقدار دهی‌های ممکن به والدین راس را نشان می‌دهد. در جدول آخرین مقدار ممکن برای متغیر راس نمایش داده نمی‌شود، زیرا برابر یک منهای جمع احتمال بقیه مقادیر است. به طور کلی، یک جدول برای یک متغیر بولی با k والد بولی، $O(2^k)$ مقدار باید داشته باشد. یک راس که هیچ والدی ندارد تنها شامل یک سطر است که احتمالات اولیه مقادیر متغیر راس را نشان می‌دهند.

۲-۲۲- مفاهیم شبکه‌های بیزی

در بخش مقدمه نیز با شبکه بیزی آشنا شدید و در اینجا می‌خواهیم مفهوم یک شبکه بیزی را بررسی کنیم. برای یک شبکه بیزی دو مفهوم می‌توان در نظر گرفت. از دید مفهومی اول، می‌توان شبکه را تقریبی از توزیع توام کامل قلمرو دید. از دید دوم می‌توان شبکه را به صورت ساختاری که وابستگی و استقلال متغیرها را نشان می‌دهد، دید. هر دو دید معادل هم می‌باشند و دید اول برای طراحی شبکه و دید دوم برای طراحی روال استنتاج مناسب می‌باشد.

۲-۲۲-۱- نمایش توزیع توام کامل

یک شبکه یک توصیف کامل از قلمرو را ارائه می‌دهد. هر عنصر توزیع احتمال توام کامل (که از این به بعد کامل را برای خلاصه‌سازی حذف می‌کنیم)، با استفاده از اطلاعات درون شبکه قابل محاسبه است. یک عنصر در توزیع را می‌توان به صورت عطف مقداردهی متغیر مانند $P(X_1 = x_1, \dots, X_n = x_n)$ در نظر گرفت. با توجه به اطلاعات شبکه مقدار یک عنصر را به وسیله رابطه (۲-۶) می‌توان محاسبه نمود: (20)

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i)) \quad (2-6)$$

محاسبه می‌شود که $\text{parents}(X_i)$ مقادیر متغیرهای درون $\text{Parents}(X_i)$ را نشان می‌دهد. بنابراین، هر عنصر توزیع توام به صورت ضرب تعدادی از عناصر جدول‌های احتمالات شرطی محاسبه می‌شود

و این جدول‌ها به نوعی تجزیه توزیع توام را انجام می‌دهد. برای نشان دادن این موضوع، احتمال به صدا در آمدن آژیر در حالی که نه زلزله آمده است و نه دزدی شده است در حالیکه جان و ماری هر دو تماس گرفته‌اند را با رابطه (۷-۲) حساب می‌کنیم. از حرف اول متغیرها برای نشان دادن آنها استفاده می‌کنیم:

$$\begin{aligned} P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) &= P(j|a)P(m|a)P(a|\neg b \wedge \neg e)P(\neg b)P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062 \end{aligned} \quad (7-2)$$

با استفاده از رابطه (۶-۲) می‌توان ایده‌ای برای ساختن ساختار شبکه بیزی بدست آورد. این رابطه مشخص می‌کند که یک شبکه بیزی چه معنی‌ای دارد، ولی به طور صریح روش ساختن شبکه را طوری که یک نمایش مناسب از قلمرو باشد، معلوم نمی‌سازد. باید توجه داشت در یک شبکه فرض بر نوع خاصی از وابستگی بین متغیرها است که لزوماً با واقعیت تطابق ندارد. بنابراین، ما تنها دنبال یک شبکه خوب می‌گردیم نه شبکه‌ای که دقیقاً توزیع توام را مدل کند (چون ممکن است ممکن نباشد). حال نشان می‌دهیم که رابطه (۶-۲) منجر به فرضی در مورد استقلال‌های شرطی می‌شود که به فرایند ساخت شبکه کمک می‌کند.

با توجه به قانون زنجیره‌ای (chain rule) در احتمالات، می‌توانیم احتمال یک عنصر توزیع توام را به صورت:

$$P(x_1, x_2, \dots, x_n) = P(x_n | x_{n-1}, \dots, x_1) P(x_{n-1} | x_{n-2}, \dots, x_1) \dots P(x_2 | x_1) P(x_1)$$

بنویسیم. با مقایسه این رابطه با رابطه (۶-۲)، پی می‌بریم که شبکه بیزی وقتی دقیقاً برابر توزیع توام است که برای هر متغیر X_i در شبکه مطابق رابطه (۸-۲) را داشته باشیم:

$$P(X_i | X_{i-1}, \dots, X_1) = P(X_i | \text{Parents}(X_i)) \quad (8-2)$$

با این فرض که $\text{Parents}(X_i) \subseteq \{X_1, \dots, X_{i-1}\}$ باشد. رابطه (۸-۲) مشخص می‌کند که شبکه بیزی تنها وقتی دقیقاً برابر توزیع توام است که هر متغیر به طور شرطی مستقل از تمامی متغیرهای رئوس بعدی در ترتیب اندیس رئوس باشد. بنابراین، در صورتی که بخواهیم یک شبکه بیزی خوب بسازیم که تا حدی خوبی به توزیع توام نزدیک باشد، نیاز داریم تا والد‌های هر راس را طوری انتخاب کنیم که این خاصیت حفظ شود. یعنی باید والدین راس منتسب به متغیر X_i را، از رئوس منتسب به

$X_1, \dots, X_{i-1}$ که بر روی X_i تاثیر مستقیم دارند انتخاب کنیم. برای مثال، فرض کنید، قسمتی از شبکه شکل (۲-۳) را ساخته‌ایم و تنها تعیین والدین MaryCalls باقیمانده است. والدین راس این متغیر را باید از متغیرهای قبل از این متغیر انتخاب کنیم که چون تنها Alarm به صورت مستقیم بر آن تاثیر می‌گذارد، به عنوان والد آن انتخاب می‌شود. شبکه‌های بیزی با در نظر گرفتن استقلال متغیرها می‌توانند در همان حال که توزیع توام را به خوبی نشان می‌دهند، از نظر محاسباتی هم قابل قبول باشند. اما در قلمروهایی که استقلال بین متغیرها کم است، این شبکه‌ها نیز نمی‌توانند کمکی کنند. در بعضی موارد، وابستگی بین متغیرهایی که آنچنان قوی نیست در نظر گرفته نمی‌شود، تا در مقابل از دست رفتن مقداری از دقت، شبکه از نظر محاسباتی قابل قبول شود. علاوه بر این، ترتیب اضافه کردن متغیرها برای ساختار شبکه هم بسیار موثر و می‌تواند برای قلمروای که می‌تواند شبکه مناسبی داشته باشد یک شبکه با یال‌های زیاد و غیرقابل قبول از لحاظ محاسباتی نتیجه دهد. این شبکه‌ها نه تنها یال‌های بیشتری دارند، بلکه ما را با سختی حساب احتمالاتی روبرو می‌کنند که بسیار پیچیده می‌تواند باشد (مثلاً احتمال وقوع زلزله در صورت به صدا درآمدن آژیر و صورت گرفتن دزدی).

۲-۲۲-۲- رابطه‌های استقلال شرطی در شبکه‌های بیزی

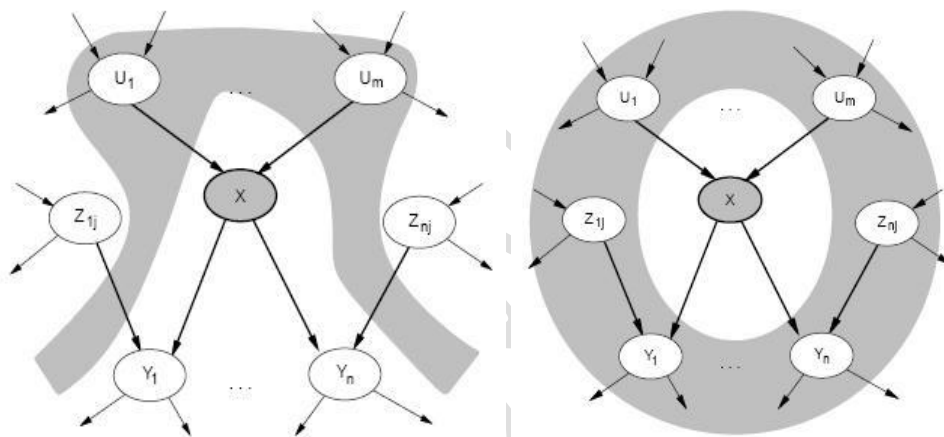
تا کنون یک مفهوم عددی برای شبکه‌های بیزی ارائه کردیم و نشان دادیم چگونه استفاده از این مفهوم می‌توان ساختار شبکه بیزی را بدست آورد. در حقیقت، ما برعکس این کار را هم می‌توانیم انجام دهیم. ما می‌توانیم از مفاهیم ساختاری (که وابستگی‌های شرطی کد شده در ساختار گرافی را نشان می‌دهد) برای بدست آوردن جدول‌های احتمالات شرطی و در نتیجه، بدست آوردن مفاهیم عددی استفاده کنیم.

مفاهیم ساختاری با یکی از دو صورت زیر که معادل هستند می‌تواند داده شود :

- ۱- یک گره به صورت شرطی مستقل از گره‌های غیربچه خود است در صورتی که والد‌های آن را داده باشند و برای مثال در شکل (۲-۲)، JohnCalls از Burglary و Earthquake در صورت داده شده بودن مقدار Alarm مستقل است.

۲- یک گره به صورت شرطی مستقل از تمامی گره‌های دیگر شبکه در صورت داده شده بودن والدهای آن، فرزندان و والدهای فرزندان آن (یا به عبارت دیگر در صورت داده شده بودن پوشش مارکوفی (MarkovBlanket) آن است. مثلاً Burglary مستقل از JohnCalls و MaryCalls در صورتی که مقادیر Alarm و Earthquake داده شده باشند، است.

توصیفی از این دو نوع نمایش در شکل (۲-۳) آمده است. از این نوع بیان استقلال‌ها و جداول احتمالات شرطی می‌توان توزیع توام را بدست آورد. بنابراین مفاهیم عددی و مفاهیم ساختاری معادل هستند.



شکل ۲-۳: ارائه مفاهیم ساختاری به دو صورت معمول [20]

۲-۲۲-۳- نمایش کارآمد توزیع‌های شرطی

در مورد تعیین جدول احتمالات شرطی باید توجه داشت که حتی اگر تعداد والدهای یک گره یک عدد کوچک k باشد، باید $O(2^k)$ مقدار برای آن گره حساب شود که به دانش زیادی برای تعیین این تعداد مقدار نیاز است. در حقیقت، این بدترین حالات است که فرض کنیم رابطه فرزندان و والدین دلخواه است و بسیاری مواقع یک رابطه مشخص و استاندارد بین مقادیر والدها و بچه برقرار است که کار تعیین توزیع شرطی را ساده می‌کند. مثلاً یک گره ممکن است لزوماً فصل مقادیر بولی والدین خود باشد. علاوه بر این، بسیاری از مسائل دنیای واقعی متغیرهایی پیوسته دارند. نمایش این

احتمالات شرطی این متغیرها به صورت جدول ممکن نمی‌باشد. در اینگونه موارد دو روش کلی استفاده می‌شود. یک روش گسسته کردن مقادیر متغیرهای پیوسته است و ایجاد جدول‌های احتمالات شرطی برای مقادیر گسسته آنها است. مشکل این روش این است که کارایی روش و دقت آن به شدت افت می‌کند و علاوه بر آن، اندازه جداول نیز بسیار بزرگ می‌شود. روش دیگر استفاده از توزیع‌های پیوسته استاندارد که با مجموعه‌ای متناهی از پارامترها تعریف می‌شوند، است. مثلاً یک توزیع نرمال شامل دو پارامتر σ, μ است و بسیار بکار می‌رود. [20]

۲۳-۲- یادگیری شبکه‌های بیزی

یکی از مشکلات استفاده از شبکه‌های بیزی این است که ایجاد کامل شبکه حتی برای یک خبره هم می‌تواند مشکل باشد. بنابراین، تلاش‌های زیادی برای یادگیری شبکه‌های بیزی صورت گرفته است. در هر شبکه بیزی ساختار آن و جداول احتمالات شرطی تعیین کننده آن هستند. بنابراین، باید بتوان با فرایند یادگیری این دو مورد را تعیین کرد. برای یادگیری خودکار ساختار شبکه یکی از روش‌های اصلی بر پایه تعیین وابستگی بین متغیرها بنا نهاده شده است. برای تعیین این وابستگی‌ها معیارهای زیادی مانند معیار آنتروپی طراحی شده‌اند. با استفاده از این معیارها مانند روشی که قبلاً توضیح داده شد، وابستگی هر متغیر X_i را نسبت به متغیرهای $X_1, \dots, X_{i-1}$ می‌سنجیم و آنها را که معیار وابستگی‌شان از یک آستانه‌ای بیشتر بود، به عنوان والد‌های X_i انتخاب می‌کنیم. پس از تعیین ساختار، اگر مقدار همه متغیرها به طور کامل قابل مشاهده باشند، از تخمین احتمال معمولی (گرفتن تعدادی نمونه و شمردن تعداد اتفاقات یک رویداد در این مجموعه نمونه) استفاده می‌کنیم. اگر بعضی از متغیرها قابل مشاهده نباشند، با استفاده از آموزش یک شبکه نرونی می‌توان مقادیر جداول احتمالات شرطی را یاد گرفت. تاکنون توضیحات مختصری در مورد شبکه‌های بیزی، مفاهیم آنها و موارد مربوط به طراحی آنها ارائه شد. در نهایت پس از طراحی شبکه تنها نکته‌ای که باقی می‌ماند، استنتاج با این شبکه‌ها است که هدف اصلی ما در بکار بردن آنها می‌باشد. [21]

۲-۲۴ - استنتاج دقیق در شبکه های بیزی

همانطور که قبلاً گفته شد، یک عمل پایه برای سیستم های احتمالی، جواب به درخواست هایی است که احتمال وقوع مقادیری خاص برای یک مجموعه از متغیرها را با فرض داده شده بودن تعدادی مشاهده (یک مقداردهی متغیرهای مشاهده) می خواهند بدانند. در اینجا از نمادگذاری ای مانند آنچه در قسمت ۱ معرفی شده، استفاده می کنیم. X نشان دهنده متغیر درخواست، E نشان دهنده مجموعه متغیرهای مشاهده $E_1, E_2, \dots, E_m$ ، e نشان دهنده مقادیر مشاهده شده برای متغیرهای E و Y نشان دهنده متغیرهایی که مشاهده نشده اند (که به آنها متغیرهای مخفی نیز می گویند) $Y_1, Y_2, \dots, Y_l$ است. بنابراین مجموعه کل متغیرها X برابر $X \cup E \cup Y$ است. یک درخواست معمول به صورت $P(X|e)$ است. در اینجا برای سادگی فرض کرده ایم که X جز متغیرهای مشاهده شده نیست، متغیرها بولی هستند و درخواست تنها در مورد یک متغیر است، زیرا تعمیم توضیحات به حالت کلی به آسانی امکان پذیر است.

۲-۲۵ - استنتاج بوسیله محاسبه تک تک عناصر احتمالی

در قسمت های قبل توضیح دادیم که محاسبه یک درخواست می تواند به صورت جمع تک تک تعدادی از عناصر توزیع توام صورت بپذیرد. با عبارت دیگر، جواب به یک درخواست، با استفاده از رابطه (۱-۲) امکان پذیر است. از آنجایی که یک شبکه بیزی دقیقاً توزیع توام را نشان می دهد (در صورتی که خاصیت گراف بدون دور جهت دار بودن گراف وابستگی متغیرها صادق باشد) و به عبارت روشن تر، از آنجایی که رابطه (۲-۲) نشان می دهد که هر عبارت $P(X, e, y)$ در رابطه (۱-۲) چگونه به صورت ضرب احتمالات شرطی قابل محاسبه است، شبکه بیزی به صورت دقیق می تواند به درخواست ها جواب دهد. باید توجه داشت که استنتاج بوسیله محاسبه تک تک عناصر از $O(n2^n)$ است. زیرا محاسبه عبارت $P(X, e, y)$ نیاز به $n-1$ ضرب دارد و y می تواند در بدترین حالت 2^n حالت را شامل شود که به $(n-1)2^n$ ضرب منجر می شود. حال سعی می کنیم با ارائه یک مثال به توضیح ایده ای پردازیم که ما را به یک الگوریتم $O(2^n)$ برای استنتاج دقیق می رساند. درخواست $P(Burglary | JohnCalls = true, MaryCalls = true)$ را در نظر بگیرید. در اینجا متغیرهای مخفی $Alarm$ و $Earthquake$ هستند. با توجه به رابطه (۲-۱) با در نظر گرفتن حرف اول اسم ها برای نشان دادن متغیرها داریم:

$$P(B | j, m) = \alpha P(B, j, m) = \alpha \sum_e \sum_a P(B, e, a, j, m) \quad (9-2)$$

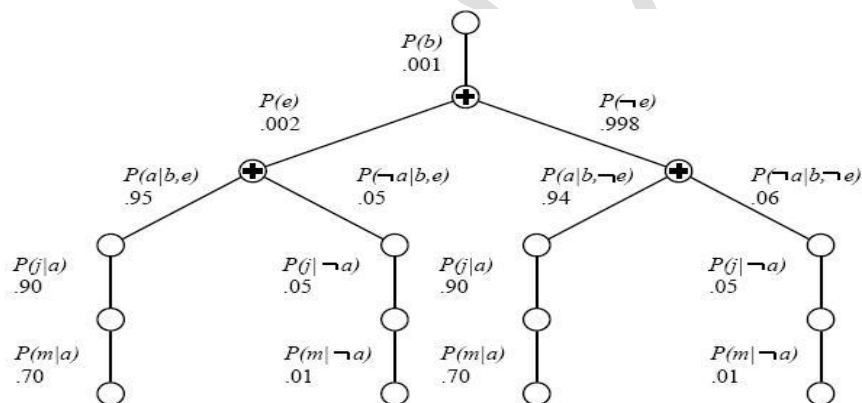
با توجه به ساختار شبکه بیزی به عبارتی به صورت رابطه (۲-۱۰) دست می‌یابیم:

$$P(j, m) = \alpha \sum_e \sum_a P(B)P(e)P(a|b,e)P(j|a)P(m|a) \quad (10-2)$$

که در اینجا $P(b)$ معادل احتمال یک مقداردهی دلخواه متغیر B است. برای محاسبه این عبارت نیاز به جمع ۵ عبارت که هر کدام از ۴ ضرب بدست آمده‌اند می‌باشد، ولی بهبودهایی با دقت بیشتر در عبارت می‌تواند مشاهده شود. $P(b)$ مستقل از متغیرهای مجموع گیری است و می‌تواند به بیرون مجموع‌ها انتقال یابد. همچنین $P(e)$ می‌تواند به خارج مجموع گیری بر روی a انتقال یابد. با در نظر گرفتن این نکات به رابطه (۲-۱۱) می‌رسیم:

$$P(b | j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e)P(j|a)P(m|a) \quad (11-2)$$

ساختار محاسبه براساس عبارت بالا که بسیار بهتر است، در شکل (۲-۴) نشان داده شده است.



شکل ۲-۴: جواب به درخواست با محاسبه عبارت بهینه‌تر [21]

۲۶-۲- الگوریتم حذف متغیر

همانطور که در مثال بالا دیده شد، الگوریتم محاسبه تک به تک عناصر، می‌تواند با حذف یکسری محاسبات تکراری با توجه به ساختار عبارت، بهبود یابد. در واقع ایده اصلی این است که هر عبارت را تنها یک بار محاسبه می‌کنیم و دفعات بعد که به این عبارت نیاز بود از مقدار محاسبه شده استفاده کنیم (این نوعی برنامه‌نویسی پویا می‌باشد). این ایده منجر به الگوریتم حذف متغیر می‌شود که به طور خلاصه به این صورت عمل می‌کند که از راست به چپ در عبارت حرکت می‌کند و با محاسبه حالت‌های متغیرها آنها را تک‌تک حذف می‌کند. نتایج میانی برای استفاده آینده ذخیره می‌شود و جمع کردن روی هر متغیر تنها برای آن قسمتی از عبارت که به این متغیر وابسته است، صورت می‌گیرد. باید توجه داشت که پیچیدگی زمانی و مکانی الگوریتم حذف متغیر وابسته به اندازه جداول نتایج میانی است و در بدترین حالت می‌تواند از $O(2^n)$ باشد. این الگوریتم در شکل (۲-۵) دیده می‌شود.

Function Elimination-Ask(X, e, bn) returns a distribution over X
 Inputs : X , the query variable
 e , evidence specified as an event
 bn , a belief network specifying joint distribution $P(X_1, \dots, X_n)$
 $factors \leftarrow []$; $vars \leftarrow Reverse(Vars[bn])$
 for each vr in $vars$ do
 $factors \leftarrow [MAKE-FACTOR(var, e) \quad factors]$
 if var is a hidden variable then $factors \leftarrow Sum - OUT(var, factors)$
 Return $NORMALIZE(POINTWISE-PRODUCT(FACTORS))$

شکل ۲-۵: الگوریتم حذف متغیر [21]

اما برای دسته‌ای خاص و پرکاربرد از شبکه‌های بیزی این الگوریتم پیچیدگی خطی براساس تعداد عناصر جداول احتمالات شرطی دارد. این دسته از شبکه‌ها، آنهایی هستند که بین هر دو راس آنها تنها یک مسیر بدون جهت وجود دارد و به آنها شبکه‌های یک همبند و یا چند درخت^۳ می‌گویند. باید توجه داشت پیچیدگی خطی براساس مقدار عناصر جداول، اگر تعداد والد‌های تمامی گره‌ها به یک حد بالا مستقل از تعداد کل گره‌ها محدود باشد، به معنی خطی بودن کل فرایند استنتاج بر اساس تعداد گره‌های شبکه است. برای شبکه‌های بیزی که یک همبند نباشند، الگوریتم حذف متغیر می‌تواند در بدترین حالت پیچیدگی نمایی داشته باشد (حتی اگر تعداد والد‌های گره‌ها به یک حد بالای ثابت محدود باشد).

۲-۲۷- استنتاج تقریبی در شبکه‌های بیزی

همانطور که دیده شد، استنتاج دقیق در شبکه‌های بیزی از نظر پیچیدگی زمانی و حافظه‌ای در مسائل بسیاری عملی نمی‌باشد و در بدترین حالت نمایی است. این مشکل باعث ایجاد دسته جدیدی از الگوریتم‌ها شد که با استفاده از نمونه‌گیری از شبکه سعی در رسیدن به یک تقریب مناسب از احتمال مورد سوال درخواست دارند. در اینجا دو دسته روش معروف از الگوریتم‌ها که نمونه‌گیری مستقیم و نمونه‌گیری زنجیره مارکونی هستند، به طور مختصر توضیح داده می‌شود. [22]

۲-۲۸- روش‌های نمونه‌گیری مستقیم

ایده اصلی این روش‌ها نمونه‌گیری با توجه به توزیعی است که شبکه بیزی نشان می‌دهد. مهم‌ترین مشکل در اعمال این ایده ساده وجود مشاهدات می‌باشد، یعنی هر نمونه‌ای که از شبکه گرفته می‌شود معتبر نمی‌باشد و باید با مقدار متغیرهای مشاهده شده سازگار باشد. برای حل این مشکل دو روش ارائه شده است که دومی معمولاً کارایی بیشتری دارد. [22]

¹ polytree

۱-۲۸-۲- نمونه‌گیری با رد کردن

این روش از یک ایده قدیمی در آمار استفاده می‌کند. از شبکه بیزی تعداد نمونه می‌گیرد و تنها آنهایی که با مشاهدات سازگار هستند را در فرایند محاسبه احتمال تقریبی در نظر می‌گیرد. این الگوریتم در شکل (۶-۲) نشان داده شده است. باید توجه داشت، اثبات می‌شود که این روش یک تقریب سازگار از توزیع شرطی را نتیجه می‌دهد. یعنی اگر تعداد نمونه‌ها به بی نهایت میل کند،

FUNCTION PRIOR-SAMPLE(bn) returns as event sampled from bn

Inputs : bn, a belief network specifying joint distribution

$P(X_1, \dots, X_n)$

$X \leftarrow$ an event with n elements

For $I = 1$ to n do

$X_i \leftarrow$ a random sample from $P((X_i | \text{parents}(X_i)))$

Given the values of $\text{Parents}(X_i)$ in X

Return X

Function REJECTION-SAMPLING (X, e, bn, N) returns an estimate of

$P(X|e)$

Local variables : N , a vector of counts over X , initially zero

For $j = 1$ to N do

$X \leftarrow$ PRIOR-SAMPLE(bn)

If X is consistent with e then

$N[x] + 1$ where x is the value of X in x

Return NORMALIZE($N[X]$)

تقریب به مقدار واقعی توزیع میل می‌کند.

شکل ۶-۲: الگوریتم نمونه‌گیری با رد کردن [23]

۲-۲۸-۲ - نمونه‌گیری وزن‌دار

دومین دسته روش از روش‌های نمونه‌گیری مستقیم، نمونه‌گیری وزن‌دار می‌باشد که برای حل مشکل نمونه‌گیری با ردکردن طراحی شده است. یکی از مشکلات بزرگ روش نمونه‌گیری با ردکردن این است که ممکن است، احتمال وقوع مشاهدات کم باشد و در نتیجه، یک قسمت اصلی از نمونه‌ها رد می‌شود. این باعث باقیماندن نمونه‌های خیلی کمی می‌شود که برای تقریب خوب، مناسب نمی‌باشند و عملاً استفاده از این روش در قلمروهای پیچیده امکان‌پذیر نمی‌باشد. نمونه‌گیری وزن‌دار مشکل نمونه‌گیری با ردکردن را به این شکل رفع می‌کند که تنها نمونه‌هایی ایجاد می‌کند که سازگار با مشاهدات هستند. این روش مقادیر متغیرهای E را ثابت در نظر می‌گیرد و تنها از مقادیر متغیرهای X و Y نمونه‌گیری می‌کند. باید توجه داشت که این نوع نمونه‌گیری توجه یکسانی به نمونه‌ها نمی‌کند و نمی‌توان تنها تعداد نمونه‌ها را برای محاسبه احتمالات موردنظر درخواست‌ها استفاده کرد. به همین دلیل به هر نمونه وزنی متناسب با احتمال وقوع مشاهدات با توجه به مقادیر متغیرهای دیگر نسبت می‌گیرد و از جمع وزن‌های نمونه‌ها برای محاسبه احتمال موردنظر درخواست استفاده می‌شود. این الگوریتم در شکل (۷-۲) آمده است. باید توجه داشت که این روش نمونه‌گیری، تقریبی سازگار را نتیجه می‌دهد. [23]

```

FUNCTION LIKELIHOOD-WEGHTING(X,e,bn,N) returns an estimate of  $P(X|e)$ 
Local variables : W, a vector of weighted counts over X , initially Zero
For j = 1 to N do
  X , w  $\leftarrow$  Weighted-sample(bn)
  W[x]  $\leftarrow$  W[x] + w where x is the value of X in x
Return NORMALIZE(W[x])
FUNCTION WEGHTED-SAMPLE(bn,e) returns an event and a weight
X  $\leftarrow$  an event with n elements: w  $\leftarrow$  1
For i= 1 to n do
  If X , has a value  $x_i$  in e
  Then w  $\leftarrow$  w  $\times$   $P(X_i = x_i | \text{parents}(X_i))$ 
  Else  $x_i \leftarrow$  a random sample from  $P(X_i | \text{parents}(X_i))$ 
Return x,w

```

شکل ۷-۲: الگوریتم نمونه گیری وزن دار [23]

۳-۲۸-۲- نمونه گیری زنجیره مارکوفی

دو روش توضیح داده شده در نمونه گیری مستقیم، برای تولید هر نمونه تمامی مقادیر را از ابتدا نمونه گیری می کنند که ممکن است از نظر زمانی هزینه بر باشد. در مقابل، روش های مبتنی بر زنجیره مارکوفی هر نمونه را با یک تغییر تصادفی در نمونه قبلی بدست می آورند. به طور دقیق تر، هر نمونه جدید به صورت تصادفی از تغییر یکی از متغیرهای غیرمشاهده ای نمونه قبلی بدست می آید. برای این کار از آن متغیر با فرض داده شده بودن مقدار مقادیر متغیرهای پوشش مارکوفی آن در نمونه قبلی، نمونه برداری می شود. به این ترتیب نمونه گیری زنجیره مارکوفی به حرکت در فضای نمونه ها (مقادیر مختلف متغیرهای غیرمشاهده شده) می پردازد. الگوریتم ساده ای براساس این ایده در شکل ۸ دیده می شود. می توان ثابت کرد که نمونه گیری زنجیره مارکوفی نیز جز نمونه گیری های سازگار می باشد. [23]

Function MCMC-ASK(X, e, bn, N) returns an estimate of $P(X|e)$

Local variables : $N[X]$, a vector of counts over X , initially Zero

Z , the nonevidence variables in bn

X , the current state of the network , initially copied from e

Initialize x with random values for the variables in Y

For $j = 1$ to N do

 For each Z , in Z do

 Sample the value of Z_i in x from $P(Z_i|mb(Z_i))$

 Given the values $M \setminus B(Z_i)$ in x

$N[x] \leftarrow N[x] + 1$ where x is the value of X in x

Return NORMALIZE($N[X]$)

شکل ۸-۲: الگوریتم نمونه گیری وزن دار [23]

۴-۲۸-۲- جمع بندی شبکه های بیزی

شبکه های بیزی یک نمایش مناسب قلمرو برای کنترل عدم قطعیت می باشند. یک شبکه بیزی یک گراف بدون دور جهت دار است که رئوس آن متغیرهای تصادفی و هر راس یک توزیع شرطی براساس والد های خود دارد. در واقع، شبکه های بیزی یک راه مناسب برای نشان دادن استقلال های شرطی که فرایند استنتاج احتمالی را سریع می کند، می باشند. شبکه های بیزی در واقع یک توزیع توام نیز برای قلمرو تعیین می کنند که هر عنصر توزیع به صورت ضرب تعدادی عناصر از جداول احتمالات شرطی رئوس می باشد. همچنین، به طور کلی اندازه گره های یک شبکه بیزی به صورت نماییکوچک تر از تعداد عناصر درون این جداول احتمالات شرطی است. بنابراین، برای نمایش کارآمد جداول توزیع های شرطی راه های زیادی طراحی شده است که استفاده از توزیع های خاص با

پارامترهای محدود از جمله این روش‌ها است. علاوه بر این، روش‌های زیادی برای استنتاج کارآمد در شبکه‌های بیزی طراحی شده است. روش‌هایی که سعی در استنتاج دقیق دارند، در قلمروهای پیچیده عملی نیستند و منجر به روش‌های استنتاج تقریبی مانند نمونه‌گیری وزن دار و نمونه‌گیری زنجیره مارکوفی می‌شوند.

۲۹-۲- تحولات انجام شده تا کنون

طیف گسترده‌ای از تکنیک‌های اولویت بندی پیشنهاد شده و مورد مطالعه قرار گرفته است. همانطور که در فصول ابتدایی این پایان نامه مطرح شده تکنیک‌های اولیه وابسته به اطلاعات مربوط به پوشش کد برای اولویت بندی بوده‌اند که در جدول (۲-۲) سیر تحولات انجام گرفته در خصوص اولویت بندی موارد تست را نشان می‌دهیم.

جدول شماره ۲-۲: سیر تحولات در مورد اولویت بندی موارد تست

نام محقق (محققین)	سال انجام	موضوع کار تحقیقاتی
D. Jeffrey, et al	2000, 2002, 2006	تکنیک‌های وابسته به اطلاعات مربوط به پوشش کد
Leon, Podgurski	2003	تکنیک‌های اولویت بندی مبتنی بر پروفایل‌های اجرا
Gupta, Jeffrey	2006	اولویت بندی موارد تست بر اساس پوشش statements در برش‌های مربوطه
Li, et al	2007	ارائه الگوریتمی بر اساس جستجو برای اولویت بندی
Korel, et al	2005, 2008	ارائه تکنیک‌های اولویت بندی مبتنی بر پوشش مدل‌های سیستم
Yoo, et al	2009	استفاده از دانش فرد خبره برای اولویت بندی

نام محقق (محققین)	سال انجام	موضوع کار تحقیقاتی
Hou , et al	2008	مطالعه‌ی اولویت بندی موارد تست بروی نرم افزار سرویس های وب
Sampath, et al	2008	مطالعه استراتژی های اولویت بندی مجموعه‌ی تست وب اپلیکیشن ها
Sherriff,et al	2007	استفاده از تاریخچه تغییرات برای جمع آوری اطلاعات و تاثیر تغییرات بر اولویت بندی موارد تست
Malishevsky,et al	2001	تکنیک اولویت بندی با استفاده از اطلاعات پوشش همراه با داده‌های در زمان اجرای تست و برآورد شدت خطا
Park,et al	2008	ارائه تکنیک برای ارزیابی زمان ها و شدت‌ها با استفاده از اطلاعات تاریخی و قدیمی موجود
Mirarab.Tahvildari	2008	استفاده از شبکه بیزی برای اولویت بندی موارد تست در تست رگرسیون
Zengkai Ma , Jianjun Zhao	2008	اولویت بندی موارد تست بر اساس ساختار برنامه

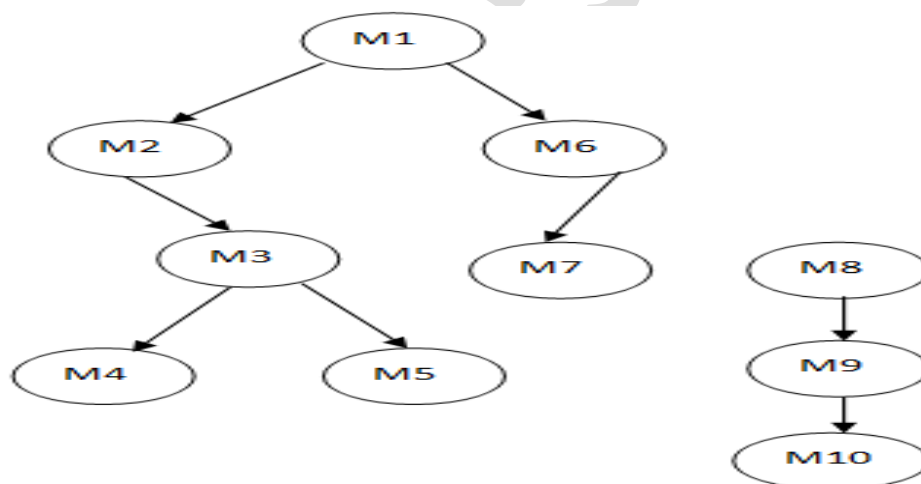
فصل سوم

روش تحقیق

3-انگیزه و هدف ما از ارائه این رویکرد

ما در این پایان نامه به دنبال این هستیم که با تمرکز بر روی ساختار برنامه، آن هم در سطح ماژول رویکرد پیشنهادی خود را اعمال کنیم. زیرا با اعتقاد بر قانون پارتو ماژول‌ها را مورد تست قرار می‌دهیم که در ساختار برنامه جزء کوچک تری از کلاس هستند. روش پیشنهادی ما در مقایسه با روش پیشنهادی مشابه متفاوت است از این حیث که بر اساس اطلاعات ساختار برنامه بر روی ماژول‌ها تکیه می‌کند. برای روشن شدن و اهمیت موضوع در شکل زیر این مسئله را به تصویر کشیده‌ایم. همانطور که در شکل (۱-۳) مشاهده می‌کنید، یک سیستم نرم افزاری به عنوان نمونه که شامل ده ماژول M_1 تا M_{10} می‌باشد و برخی ارتباطاتی که بین هرکدام از این ماژول‌ها با یکدیگر وجود دارد. و از طرفی هم یک مجموعه تست نرم افزار شامل ده مورد آزمون T_1 تا T_{10} موجود می‌باشد که ماژول‌های M_1 تا M_{10} را پوشش می‌دهد.

شکل ۱-۳: ماژول‌های یک سیستم ساده نرم افزاری



جدول شماره ۳-۱: سیستمی ساده شامل ۱۰ ماژول و ۱۰ مورد تست

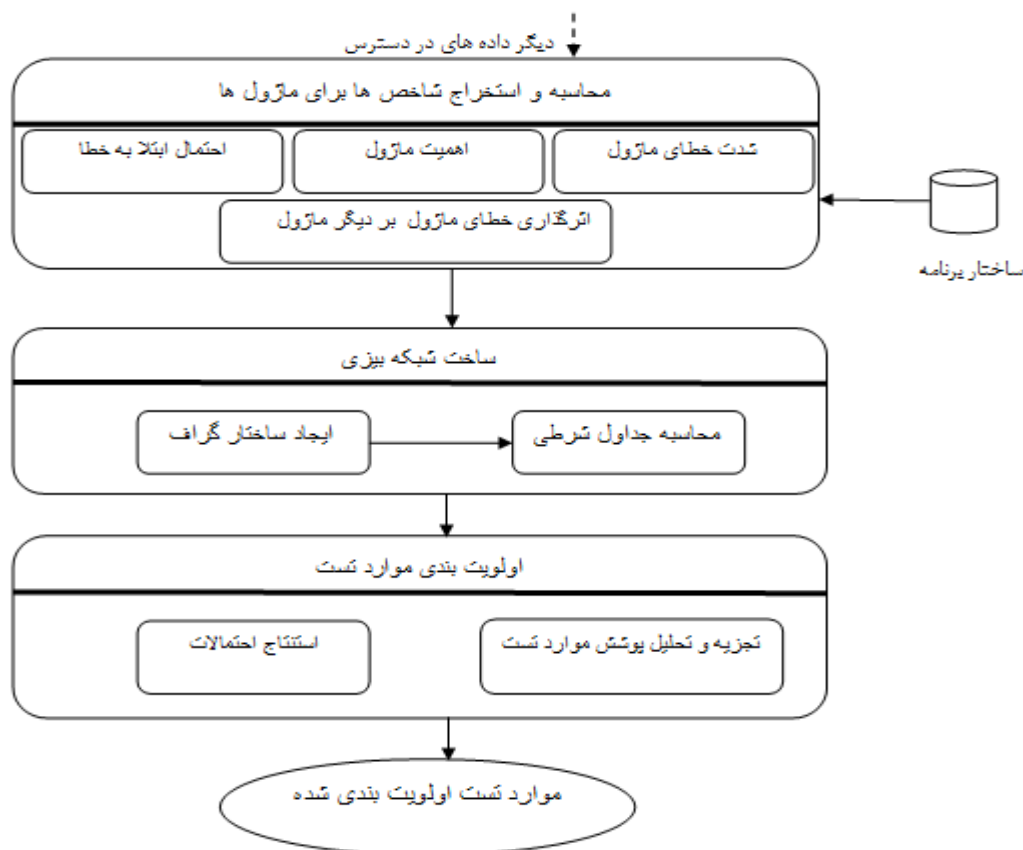
Module		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
Test Case	T1	*									
	T2		*								
	T3			*							
	T4				*						
	T5					*					
	T6						*				
	T7							*			
	T8								*		
	T9									*	
	T10										*

همانطور که در نگاه اول به شکل (۳-۱) و جدول (۳-۱) متوجه این موضوع می‌شویم که هرکدام از ماژول‌ها توسط یکی از موارد آزمون پوشش داده می‌شوند به نقاط ضعف موجود در آن هم پی می‌بریم. به جهت پیدا کردن خطاهای شدید در موعد مقرر و مناسب، باید موارد آزمون را اولویت بندی کنیم. اولویت بندی کردن موارد آزمون به عنوان مرحله ای از تست رگرسیون نیست، در اینجا، آن دسته از تکنیک های اولویت بندی که تکیه بر اطلاعات موجود در این نوع تست دارند نمی‌تواند در اینجا کاربرد داشته باشند و مفید واقع شوند. علاوه بر این، به دلیل اینکه هریک از موارد آزمون به همان تعداد از ماژول‌ها را پوشش می‌دهد و ما قادر به شناسایی اولویت موارد آزمون از لحاظ تعداد ماژول تحت پوشش هرکدام از موارد آزمون نخواهیم بود. بنابراین، برای بهبود بخشیدن به سرعت تشخیص یا شناسایی خطاهای شدید برای این مجموعه تست نرم افزار، شاخص‌های اولویت بندی دیگری را باید در اینجا مدنظر قرار دهیم که ما در ادامه معیارها و متریک هایی را که جهت مشخص کردن اولویت موارد آزمون مورد استفاده قرار داده‌ایم را به تفصیل شرح خواهیم داد.

۳-۱- رویکرد پیشنهادی

۳-۱-۱- روند کلی در رویکرد پیشنهادی

رویکرد پیشنهاد شده در این پایان نامه از ۳ بخش تشکیل شده است که در ادامه نمایش می‌دهیم. ما در رویکرد پیشنهادی خود برای حل کردن مشکل تکنیک‌های اولویت بندی ارائه شده بر تصمیم‌گیری بر اساس معیارها و فاکتورهای اولویت بندی تمرکز نموده و برای رفع آن نیز از استخراج معیارها و محاسبه هرکدام از معیارهای مورد نظر در بخش‌ها و اطلاعات مختلف بهره می‌بریم، منابع جمع‌آوری این اطلاعات عبارتند از: کد برنامه، ساختار برنامه، اطلاعات قبلی در دسترس و نظر افراد کارشناس یا متخصص در این حوزه. و همچنین پس از بدست آوردن اطلاعات مورد نیاز جهت تصمیم‌گیری و اولویت بندی از شبکه‌های بیزی استفاده می‌کنیم تا در نهایت اولویت بندی کارآمد تری را ارائه دهیم. مطالعات تجربی انجام گرفته بر روی شبکه‌های بیزی و همچنین نتایج حاصله از بکارگیری شبکه‌های بیزی ما را به دستیابی به نتایج بهتر از روش‌های دیگر اولویت بندی موارد تست و کارآمد بودن نهایی تست نرم افزار امیدوار می‌کنند. شکل (۲-۳) یک شمای کلی و سطح بالا از روش پیشنهادی ما در این پایان نامه را نشان می‌دهد. در ادامه به شرح هر یک از مراحل کار می‌پردازیم.



شکل ۳-۲: یک پیاده سازی عمومی برای چارچوب مبتنی بر شبکه های ییزی

همانطور که در شکل فوق نشان داده شده ورودی های این مدل را ساختار برنامه و داده های در دسترس تشکیل می دهند. که در مقایسه با دیگر روش های اولویت بندی که تا کنون پیشنهاد شده اند تفاوت دارد و تفاوت آن هم اینکه نیاز ما را از حیث از قبل دانستن برخی اطلاعات و اینکه شروع فرایند مستلزم وجود آنها باشد را بر طرف می کند. همچنین اولین قدم در اولویت بندی چیزی نیست جز جمع آوری تمامی اطلاعاتی که به هرچه کارآمدتر شدن فرایند اولویت بندی کمک می کند. کارآمدتر شدن اولویت بندی را با یک مثال شرح می دهیم. فرض کنید برای اینکه برای انجام کاری تصمیم گیری کنید در ابتدا اطلاعات کمی در اختیار دارید و همچنین به همراه این اطلاعات کم نیز اطلاعاتی ناقص و ناکافی هم وجود دارند، حال فرایند تصمیم گیری بر اساس این اطلاعات صورت می گیرد اما حال به نظر شما تصمیم گیری بر اساس این اطلاعات کارآمد خواهد بود؟ اما اگر

اطلاعات به خوبی و از منابع درست جمع آوری شوند و از افراد کارشناس و متخصص بهره ببریم قطعاً نتیجه‌ی بهتری از تصمیم‌گیری خواهیم گرفت.

از دیگر مزایای مدل پیشنهادی ما برای اولویت‌بندی موارد تست این است که شبکه‌های بیزی در مواردی که اطلاعات نا کافی یا اطلاعات ناقص وجود دارند هم به دلیل بهره‌بردن از احتمالات شرطی قادر به تصمیم‌گیری بر اساس اطلاعات موجود خواهد بود. ما در روش خود با در نظر گرفتن منابع اطلاعاتی همچون کد برنامه، اطلاعات قبلی موارد تست اجرا شده، اطلاعات تغییرات، نظرات کارشناسان خبره در این حوزه و ساختار برنامه بهره خواهیم برد.

۲-۳- محاسبه و استخراج شاخص‌ها برای ماژول‌ها

در این بخش از فرآیند در مدل پیشنهادی خود به ارزیابی معیارهای مورد توجه در این پایان‌نامه جهت تعیین اولویت موارد آزمون می‌پردازیم. که در ارزیابی هریک از این معیارها با توجه به داده‌های در دسترس (به عنوان مثال، نیازمندی‌های کاربر، تغییرات کد برنامه، داده‌های اجرای قبلی موارد تست ها) می‌پردازیم. قبل از اینکه وارد بحث ارزیابی متریک‌ها شویم معیارهای مورد توجه در این پایان‌نامه را معرفی می‌کنیم و سپس هر کدام را جداگانه شرح می‌دهیم. در نهایت با یکپارچه‌سازی این اطلاعات بدست آمده در مدل شبکه بیزی اولویت‌بندی را انجام می‌دهیم.

3-3- معیارهای رویکرد پیشنهادی

✓ اهمیت هر ماژول (IM)

✓ احتمال ابتلا به خطای ماژول (FP)

✓ اثر گذاری خطای ماژول (IE)

✓ شدت خطای ماژول (SE)

۱-۳-۳- اهمیت هر ماژول IM (Important of Module)

این معیار به معنی اهمیت ماژول برای تست کردن بوده و برای اهمیت ماژول وزنی هم در نظر گرفته می‌شود که با توجه به مفهوم تست تعیین می‌گردد. در مواقعی که تعیین دقیق این معیار برای یک ماژول سخت و دشوار است می‌توان مقدار آنرا کم در نظر گرفت. اما ما به دلیل استفاده از شبکه بیزی در رویکردمان و تصمیم‌گیری با وجود عدم قطعیت و نا کافی بودن اطلاعات، که به عنوان یک ویژگی برای شبکه‌های بیزی محسوب می‌شود می‌توانیم این مشکل را حل کنیم. این توانمندی در شبکه‌های بیزی را در فصول ابتدایی با ذکر مثال شرح داده‌ایم و در اینجا از تکرار مطالب خودداری می‌کنیم. مطالعات انجام گرفته و همچنین نتایج بدست آمده قبلی نشان می‌دهند که بهره بردن از شبکه‌های بیزی در تصمیم‌گیری‌های با وجود عدم قطعیت و ناکافی بودن اطلاعات می‌تواند مفید و موثر باشد. همچنین به عقیده من اهمیت ماژول بایستی از دو منظر بررسی شود، از نظر مهندس تست و از دید سیستم باید مدنظر باشد. خطای موجود در ماژولی که به ورود داده توسط کاربر مرتبط است به مراتب شدیدتر از دیگر ماژول‌هایی است که به کاربر مربوط نمی‌شوند و همچنین خطای موجود در ماژولی که به سیستم مرتبط است شدیدتر و یک تهدید جدی‌تری خواهد بود برای سیستم. حال روش ما برای محاسبه این معیار یعنی اهمیت ماژول بدین صورت است که نظرات فرد خبره و مهندس در خصوص ماژول‌ها جمع‌آوری می‌شود و در جداول قرار می‌گیرند و وزنی به هریک داده می‌شود و در نهایت با ترکیب آنها اهمیت نهایی ماژول تعیین می‌گردد. اهمیت ماژول از دید مهندس تست را هم فرد خبره هم مهندس تست می‌توانند تعیین کنند با توجه به نیازمندی‌های کاربر و مشتری محصول نرم افزاری، و در مقابل اهمیت ماژول از دید سیستم می‌تواند توسط مهندس تست برای هر ماژول پس از تعیین روابط بین ماژول‌های برنامه تحت تست تعیین گردد. بدین منظور از رابطه (۱-۳) استفاده می‌کنیم:

$$(Importans of Module) \quad IM = I_{TE} + I_S \quad (۳-۱)$$

۳-۳-۲- احتمال ابتلا به خطای ماژول FP

برخی از انواع اطلاعاتی که در دیگر پژوهش‌ها برای برآورد این معیار بکار گرفته شده‌اند عبارتند از :
پیچیدگی پیاده سازی نیازمندی‌ها و تغییرات یا اصلاح کد برنامه می‌باشند. برای وزن دادن به این معیار بایستی شاخص‌هایی را مدنظر داشت که این شاخص‌ها عبارتند از :

✓ تعداد خطاهایی که ماژول داشته یا دارد

✓ آخرین خطای رخ داده در ماژول

✓ جدیدترین تغییرات اعمال شده بروی ماژول

✓ رخ دادن خطاهای قبلی در ماژول‌ها پس از اعمال تغییرات

توجه داشته باشید که وزن این معیار با توجه به مفهوم تست در نظر گرفته می‌شود. برای مثال زمانی که با در دست داشتن کلیه اطلاعات برآورد احتمال ابتلا به خطا برای ماژول سخت و دشوار باشد می‌توانیم وزن آنرا کوچک‌تر در نظر بگیریم. همچنین برعکس این قضیه اینکه اگرچه با اطلاعات موجود احتمال ابتلا به خطای ماژول ثابت شود می‌توان وزن آنرا بزرگ‌تر در نظر گرفت. در نهایت به این نتیجه می‌رسیم که وزن اختصاص یافته برای ماژول‌ها بسیار مهم و تاثیر گذار است بر تصمیم‌گیری نهایی و محاسبات ، پس تعیین مقدار برای این معیار هرچه دقیق‌تر باشد اولویت بندی نیز دقیق‌تر خواهد بود. بدین منظور از رابطه (۲-۳) بهره خواهیم برد :

$$(Fault Pronens of modules) \quad FP= IE + SE \quad (۳-۲)$$

۳-۳-۳- اثرگذاری خطای ماژول بر دیگر ماژول‌ها IE°

برای روشن شدن اهمیت این معیار از یک مثال استفاده می‌کنیم. فرض کنید در یک سیستم نرم افزاری که از پروسیجرها و توابع مختلف تشکیل شده است بخواهیم مشخص کنیم بروز خطا در کدام یک از بخش‌های این برنامه شدیدتر از دیگر خطاهاست. آیا خطا در توابع فراخوان شدیدتر

¹ Fault Proneness 4

¹ Impact of Error 5

است یا در توابع فراخوانی شده؟ آیا خطا در متغیر هایی که توسط کاربر مقدار دهی می‌شوند شدیدتر است یا متغیر هایی که توسط تابع برگردانده می‌شود؟ این معیار بسیار مهم می‌باشد و به ما کمک می‌کند تا اینکه به خطاهای شدید تر اولویت بیشتری بدهیم. بدلیل اینکه ما در رویکردمان تمرکز را بروی تجزیه تحلیل ساختار برنامه قرار داده‌ایم و فرض بر نداشتن اطلاعات از اجراهای قبلی موارد تست ها است توجه به این متریک می‌تواند نتایج ما در اولویت بندی موارد تست را قابل اعتمادتر کند.

۴-۳-۳- شدت خطای هر ماژول SE^1

برای هر کدام از ماژول ها شدت اثر گذاری خطای رخ داده با دیگر ماژول هائی می‌تواند متفاوت باشد و در طبقه بندی ارائه شده در جایگاهی قرار گیرد. و ما این معیار را از نظر مهندس تست و هم از نظر کاربر مورد بررسی و محاسبه قرار می‌دهیم. به گونه ای که هر کدام از این دو گروه به این معیار برای هر ماژول وزنی را اختصاص می‌دهند که برآیند آن دو را ما برای تصمیم گیری لحاظ می‌کنیم. در جدول (۲-۳) سطح شدت خطاها را نشان داده ایم.

جدول ۲-۳: سطح شدت خطای هر ماژول

شدت خطای هر ماژول			
Fault Severity	شدت بالا	100	High Severity
	شدت متوسط	50	Medium Severity
	شدت کم	0	Low Severity

¹ Severity of Error

۳-۴- شاخص‌های وزن دهی به ماژول‌ها

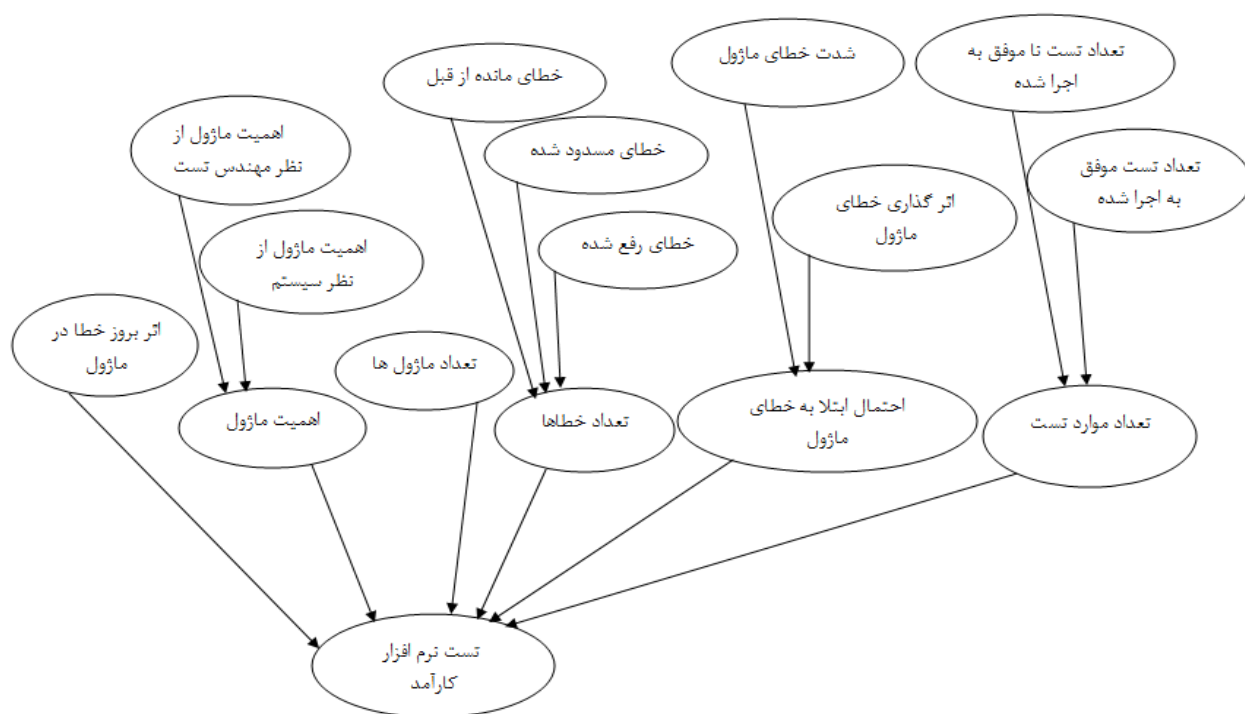
برای ماژول‌ها معیارهایی مشخص شده است که هر یک از اهمیت خاصی برخوردارند و نمی‌توان به یکی بیش از دیگری بها داد، از این حیث برای وزن دادن به هر کدام از این معیارها ما شاخص‌هایی را مدنظر قرار دادیم تا هر کدام از موارد تست بر اساس وزن مشخص شده برای معیارهایش انتخاب گردد. این شاخص‌ها عبارتند از :

- ✓ تعداد خطاهایی که ماژول داشته یا دارد
- ✓ اثر بروز خطا در این ماژول به روی دیگر ماژول‌ها
- ✓ آخرین خطای رخ داده در ماژول
- ✓ جدیدترین تغییرات اعمال شده بروی ماژول
- ✓ ماژول‌های فراخوانی شونده
- ✓ رخ دادن خطاهای قبلی در ماژول‌ها پس از اعمال تغییرات

۳-۵- ساخت شبکه بیزی

این مرحله همانطور که از نام این بخش مشخص است ما در آن به ساخت شبکه بیزی می‌پردازیم. این فرایند پس از اینکه داده‌های ورودی این مدل مورد سنجش و ارزیابی قرار گرفتند و داده‌های لازم یا شواهد جهت ساخت شبکه بیزی فراهم گردید می‌توانیم به بخش اول ساخت شبکه‌های بیزی پردازیم. بخش اول از ایجاد شبکه بیزی مرتبط است به ایجاد گراف با توجه بر داده‌ها و روابط علی معلولی استخراج شده.

۳-۶- ایجاد ساختار گراف



شکل ۳-۳: گراف شبکه بیزی تست کارآمد نرم افزار

در اینجا گراف (۳-۳) که از مرحله قبل ساخته شده است را نمایش می‌دهیم. ارتباطات و وابستگی‌هایی که در گراف فوق مشاهده می‌کنید با توجه به کارهای مشابه و نظرات افراد خبره در حوزه تست نرم افزار تعیین شده است. گراف شبکه بیزی پیشنهاد شده در شکل فوق دارای ویژگی‌های مناسبی است. یکی از مهم‌ترین این ویژگی‌ها این است که گراف پیشنهادی از استقلال شرطی بسیار خوبی برخوردار است. استقلال شرطی یک مفهوم بسیار مهم است که دانستن آن سبب ساده سازی بسیار خوب مدل می‌گردد. حتی اگر در مواردی استقلال شرطی واقعا وجود نداشته باشد، فرض وجود استقلال شرطی سبب می‌شود که بهبود بسیاری در مدل احتمالاتی مورد استفاده

صورت گیرد. ولی مدل ارائه شده در اینجا از استقلال شرطی بسیار خوبی برخوردار است. زیرا هر کدام از فاکتورها که در گراف فوق قرار گرفته‌اند با توجه به مفهوم تست به دیگر فاکتورهای در این مدل وابسته‌اند یا اینکه مستقل از آنها هستند که این امر نیز به راحتی از طریق مشاهده گراف قابل درک است. به طور خلاصه می‌توان گفت شبکه بیزی، نمایش با معنی روابط نامشخص ما بین پارامترها در یک حوزه می‌باشد. شبکه بیزی گراف جهت دار غیر حلقوی از نودها برای نمایش متغیرهای تصادفی و کمان‌ها برای نمایش روابط احتمالی ما بین متغیرها به شمار می‌رود. فقدان لبه‌ها در شبکه بیزی نشانگر جملات مستقل از هم می‌باشد.

۳-۷- محاسبه جداول احتمال شرطی

شبکه بیزی برای نمایش توزیع احتمالی ویژه و اتصال توزیع بر روی همه متغیرها به صورت نودها در گراف نمایش داده می‌شود. این توزیع با یک مجموعه از جداول احتمال شرطی^۷ مشخص می‌شود. هر نود به جدول احتمال شرطی متناسب شده و توسط اطلاعات احتمالی کمی مشخص می‌گردد. همانند جدولی احتمالات در وضعیت ممکن از نود در ترکیب ممکن از والدینش مشخص می‌گردد. برای نودهای بدون والد احتمالات بر روی نودهای دیگر بدون قید و شرط می‌باشند که این نودها احتمالات اولیه بر روی متغیرها نامیده می‌شوند. ویژگی‌های فرعی و فاکتورهای مورد نظر تست پیشنهاد شده در این پایان نامه رامی‌توان به دو روش کاملاً متفاوت در نظر گرفت. در روش اول تأثیر ویژگی‌های فرعی بر روی فاکتور تست مورد نظرشان یکسان است. در مقابل در روش دوم تأثیر ویژگی‌های فرعی بر روی فاکتور مورد نظرشان یکسان نیست و ویژگی‌ها با توجه به وزنشان روی فاکتور مورد نظر تأثیر دارند. اینکه در پروژه ای نرم افزاری ویژگی‌های فرعی هم وزن هستند یا نه، و اینکه وزن هر کدام از ویژگی‌های فرعی و تأثیر آنها روی فاکتور مربوطه چقدر است را متخصصان تست نرم افزار تعیین می‌کنند. در حالت کلی و استاندارد متخصصان حوزه تست و

^۷CPT(conditional probability tables)

مهندسان تست روش اول را که در آن همه ی ویژگی‌های فرعی یک فاکتور هم وزن بوده و تأثیر یکسان دارند را در نظر می گیرند.

هر کدام از ویژگی‌های فرعی و فاکتورهای تست و همینطور خود کارآمدی نهایی تست را می توان به اندازه های مختلف درجه بندی نمود. تقسیم بندی‌های صفات کیفی می‌تواند دو تایی ، سه تایی، چهارتایی، پنج تایی باشد. هرچه تعداد این تقسیم بندی ها بیشتر باشد جواب نهایی دقیق‌تر و با جزئیات بیشتری بوده و قابلیت اتکای بیشتری خواهد داشت. اما اگر بخواهیم تعداد صفات کیفی را زیاد کنیم تا دقت جواب نهایی را بالاتر ببریم، در مقابل هم زمان انجام کار و هم افزونگی داده‌ها افزایش می‌یابد. همانطور که با افزایش تعداد تقسیم بندی‌ها، دقت مدل بیشتر می‌شود در مقابل نیز پیاده سازی مدل سخت‌تر از حالت قبل می‌شود. در این پایان نامه تاکید روی تقسیم بندی سه تایی است. در تقسیم بندی سه تایی اندازه‌ی صفات کیفی، مقادیر در نظر گرفته شده عبارتند از: کم، متوسط و زیاد .

تعداد حالت‌های مختلف یک فاکتور یا به عبارت دیگر تعداد سطرهای جداول احتمال شرطی فاکتورها به دو مولفه بستگی دارد. یکی تعداد و ویژگی‌های فرعی آن فاکتور تست و دوم تعداد حالت‌هایی است که برای آن ویژگی‌های فرعی در نظر گرفته‌ایم . چون تعداد حالت‌های همه‌ی ویژگی‌های فرعی در این پایان نامه به صورت برابر در نظر گرفته شده (سه حالت) پس می‌توان با استفاده از رابطه n^P تعداد سطرهای جداول احتمال شرطی فاکتورها را بدست آورد که در آن P تعداد ویژگی‌های فرعی یک فاکتور تست است و n نیز تعداد حالت‌های مختلف هر کدام از ویژگی‌های فرعی است. برای مثال فاکتور تعداد خطا که دارای سه ویژگی فرعی است. اگر هر یک از ویژگی‌های فرعی را دو حالت در نظر بگیریم جدول تعداد خطا دارای ۸ سطر و اگر هر ویژگی فرعی را سه حالت در نظر بگیریم جدول تعداد خطا دارای ۲۷ سطر خواهد بود و اگر که هر ویژگی فرعی را پنج حالت در نظر بگیریم جدول تعداد خطا دارای ۱۲۵ سطر خواهد بود. آنچه مشخص است این است که تعداد حالت‌های ویژگی‌های فرعی به شدت روی تعداد سطرهای جدول فاکتورها تأثیر خواهد گذاشت. در جدول زیر برای نمایش این وضعیت‌ها و روشن شدن موضوع حالت‌های مختلف برای هر یک از فاکتورها را محاسبه کرده‌ایم و نشان می‌دهیم.

جدول شماره ۳-۳: تعداد سطرهای جدول احتمال شرطی فاکتورهای تست

تعداد سطرها			فاکتور تست
متریک دو حالت	متریک سه حالت	متریک پنج حالت	
۴	۹	۲۵	تعداد موارد تست
۴	۹	۲۵	احتمال ابتلا به خطای مازول
۸	۲۷	۱۲۵	تعداد خطاها
۲	۳	۵	تعداد مازولها
۴	۹	۲۵	اهمیت مازول
۲	۳	۵	اثر بروز خطا در مازول

در جدول (۳-۳) تعداد سطرهای جدول احتمال شرطی را به تفکیک معیار یا فاکتورهای تست بیان کردیم. جدول احتمال شرطی نهایی برای تست کارآمد نرم افزار در زمانی که فاکتور ها دو حالت باشند ۶۴ سه حالت باشند ۷۲۹ و پنج حالت باشند ۱۵۶۲۵ سطر خواهند داشت. این امر مشخص است که زمانی که فاکتورها را پنج حالت در نظر بگیریم پر کردن جداول احتمال شرطی کاری سخت و دشوار و خیلی زمان بر خواهد بود و چه بسا غیر عملی. زیرا تعداد سطرها بسیار زیاد است و همچنین امکان ورود داده‌های اشتباه در جداول هم افزایش می‌یابد. پس برای پر کردن جداول احتمال شرطی با صفت‌های پنج حالت بایستی از نرم افزارهایی مانند BNJ (پیاده سازی شبکه‌های بیزی با استفاده از JAVA) و BNT (پیاده سازی شبکه‌های بیزی با استفاده از MATLAB) استفاده

نمود. زیرا با استفاده از روش برنامه نویسی می‌توان به صورت خودکار و با استفاده از حلقه‌های تکرار جداول احتمال شرطی را پر نمود.

پس از تقسیم بندی صفت‌های کیفی می‌بایست آنها را به مقیاس‌های کمی تبدیل نماییم تا بتوانیم فاکتورهای اثر گذار بر فرایند تست نرم افزار و همچنین کارآمد بودن اولویت بندی موارد تست را محاسبه نماییم. محاسبه‌ی فاکتورها در این مدل به این صورت است که ابتدا مقادیر بدست آمده از ویژگی‌های فرعی به گونه‌ای که در ادامه شرح داده خواهد شد به مقادیر کمی تبدیل می‌گردند. از روی محاسبات انجام شده روی این مقادیر کمی، اندازه‌ی فاکتور مربوطه را محاسبه می‌نماییم. به همین طریق و با استفاده از فاکتورهای تست می‌توان کارآمدی تست را با اولویت بندی موارد تست بدین گونه مشخص نمود. با عملکردی این چنین ما قادر خواهیم بود با استفاده از شبکه‌های بیزی و احتمالات کارآمدی از نحوه‌ی اولویت بندی موارد تست را نشان دهیم.

۸-۳- تبدیل اندازه‌ی کیفی صفت‌ها به مقادیر کمی

برای تبدیل اندازه کیفی صفت‌ها به مقادیر کمی ما در رویکردمان دو روش پیشنهادی را نشان می‌دهیم و با یکدیگر مقایسه می‌کنیم. برای توضیح روش‌های پیشنهادی از حالت صفت‌های سه حالت استفاده می‌کنیم زیرا با صفت‌های کمتر دقت افت می‌کند و اگر تعداد صفات را زیاد در نظر بگیریم باعث افزایش اضافه کاری و حجم داده زیاد می‌شود.

۳-۹- روش اول برای صفت‌های سه حالت

در روش اول بدین صورت عمل می‌کنیم که، ابتدا به هر کدام از ویژگی‌های فرعی مربوط به معیارها یا فاکتورها با توجه به جدول زیر مقدار مربوط به اندازه صفت کیفی را نسبت می‌دهیم.

جدول شماره ۳-۴: انتساب مقدار عددی به اندازه‌های کیفی در صفت‌های سه حالت

اندازه کیفی	مقدار کمی
-------------	-----------

زیاد	۱۰۰
متوسط	۵۰
کم	۰

سپس میانگین صفت‌ها را محاسبه می‌نماییم. میانگین بدست آمده در یکی از بازه‌های جدول زیر قرار خواهد گرفت و با توجه به بازه مقدار کیفی صفت مربوطه بدست خواهد آمد. سطر جدول احتمال شرطی مربوطه نیز به این صورت خواهد بود که اندازه‌ی مقدار کیفی بدست آمده برابر 100 و مقدار کیفی برای دو مقدار کیفی دیگر برابر ۰ خواهد بود.

جدول شماره ۳-۵: بازه‌های تبدیل میانگین به اندازه‌های کیفی در صفت‌های سه حالت

بازه	اندازه کیفی
$66 < X < 100$	زیاد
$33 < X < 66$	متوسط
$0 < X < 33$	کم

۳-۹-۱- مثالی از روش اول برای صفت‌های سه حالت

به عنوان مثال فرض می‌کنیم در یک پروژه نرم افزاری می‌خواهیم فاکتور احتمال ابتلا به خطای ماژول را محاسبه نماییم و ویژگی‌های فرعی آن عبارتند از: اثرگذاری خطای ماژول بر دیگر ماژول‌ها با مقدار زیاد و ویژگی فرعی شدت خطای ماژول با مقدار متوسط باشد. برای اندازه گیری فاکتور احتمال ابتلا به خطای ماژول محاسبات زیر را خواهیم داشت.

ابتدا میانگین اندازه‌ی صفت‌ها را توسط رابطه (۳-۳) محاسبه می‌نماییم:

$$100 + 50 = \frac{150}{2} = 75 \rightarrow 66 < 75 < 100 \quad (3-3)$$

نتیجه بدست آمده عدد ۷۵ است، یعنی اینکه $۷۵ < ۱۰۰$ ، یعنی اینکه $۷۵ < ۱۰۰$.

پس طبق جدول در بازه ی متوسط قرار دارد یعنی میزان کیفی فاکتور احتمال ابتلا به خطای ماژول برابر زیاد است. سطرهای جدول احتمال شرطی برای فاکتور احتمال ابتلا به خطای ماژول با این ویژگی های فرعی در جدول زیر نشان داده شده است.

جدول ۳-۶: سطری از جدول احتمال شرطی احتمال ابتلا به خطای ماژول با سه ویژگی فرعی

اندازه کیفی			ویژگی های فرعی فاکتور احتمال ابتلا به خطای ماژول	
کم	متوسط	زیاد	اثرگذاری خطای ماژول	شدت خطای ماژول
۰	۰	۱۰۰	زیاد	متوسط

۳-۱۰-۱- روش دوم برای صفت های سه حالت

در این روش ابتدا به هر کدام از ویژگی های فرعی با توجه به جدول انتساب مقدار عددی به اندازه های کیفی در صفت های سه حالت، مقدار مربوط به اندازه ی صفت کیفی را نسبت می دهیم. بعد از آن میانگین صفت ها را محاسبه می نماییم و اختلاف میانگین بدست آمده از هر یک از مرزهای قبل و بعد را محاسبه می نماییم. با ضرب اختلاف هر یک از مرزها با میانگین در عدد ۲، درصد مرز دیگر بدست خواهد آمد. مرز سوم نیز عدد صفر خواهد گرفت. به این دلیل اختلاف هر یک از مرزها با میانگین در عدد ۲ ضرب می شود که صفت های سه حالت کل بازه یک صفت را به دو زیر بازه تقسیم می کنند. زیر بازه ی صفر تا ۵۰ و زیر بازه ی ۵۰ تا ۱۰۰.

۳-۱۰-۱- مثالی از روش دوم برای صفت های سه حالت

به عنوان مثال فرض می کنیم در یک پروژه نرم افزاری می خواهیم فاکتور تعداد خطاهای ماژول را محاسبه نماییم که ویژگی های فرعی آن عبارتند از: خطاهای رفع شده، خطاهای مسدود شده و خطاهای مانده از قبل که به ترتیب دارای مقادیر زیاد، کم و کم می باشند. برای اندازه گیری فاکتور تعداد خطاهای ماژول محاسبات زیر را خواهیم داشت.

ابتدا میانگین اندازه‌ی صفت‌ها را توسط رابطه (۳-۴) محاسبه می‌نماییم:

$$50 + 50 + 100 = \frac{200}{3} = 66.66 \quad (۳-۴)$$

حال مقدار میانگین بدست آمده را در بازه مربوطه قرار می‌دهیم. یعنی اینکه $۶۶,۶۶ < ۱۰۰$ ، حال بازه ما به دو بخش تقسیم می‌شود یکی از صفر تا ۵۰ و دیگری از ۵۰ تا ۱۰۰. سپس مقدار میانگین بدست آمده را از طبق رابطه‌ی ذکر شده در بالا از مرزهای بالایی و پایینی تفریق نموده تا اینکه مقدار فاصله میانگین از هریک از مرزهای بالا و پایین بدست آید. مقدار فاصله بدست آمده را در عدد ۲ ضرب می‌کنیم زیرا بازه‌ی ما به دو بازه ۵۰ تایی تقسیم شده‌اند. بدین منظور رابطه (۵-۳) استفاده می‌شود:

$$66.66 - 50 = 16.66 \gg 16.66 * 2 = 33.32 \quad (۵-۳)$$

$$100 - 66.66 = 33.34 \gg 33.34 * 2 = 66.68$$

جدول ۳-۷: سطری از جدول احتمال شرطی احتمال ابتلا به خطای ماژول با سه ویژگی فرعی

اندازه کیفی			ویژگی‌های فرعی فاکتور احتمال ابتلا به خطای ماژول	
کم	متوسط	زیاد	اثرگذاری خطای ماژول	شدت خطای ماژول
۰	۳۳,۳۲	۶۶,۶۸	زیاد	متوسط

۳-۱۱- تبدیل اندازه‌ی کیفی صفت‌های غیرهم وزن به مقادیر کمی

همانطور که گفته شد ویژگی‌های فرعی و فاکتورهای تست ارائه شده در مدل پیشنهادی را می‌توان به دو روش در نظر گرفت. در روش اول تأثیر ویژگی‌های فرعی روی فاکتور مورد نظرشان یکسان است و در روش دوم تأثیر ویژگی‌های فرعی روی فاکتور مورد نظرشان یکسان نیست و فاکتورها با توجه به وزنشان روی فاکتور مورد نظر تأثیر دارند. اینکه در پروژه نرم افزاری ویژگی‌های فرعی هم وزن هستند یا نه، و اینکه وزن هر کدام از ویژگی‌های فرعی و تأثیر آنها روی فاکتور کیفی مربوطه چقدر است را متخصصان و مهندسان تست نرم افزار تعیین می‌کنند.

تبدیل اندازه ی کیفی صفت‌های غیر هم وزن به مقادیر کمی به همان صورت صفت‌های کیفی هم وزن انجام می شود با این تفاوت که در محاسبه ی میانگین وزن صفت‌ها را نیز دخالت می دهیم. رابطه محاسبه ی میانگین برای صفت‌های غیر هم وزن به صورت رابطه (۶-۳) زیرخواهد بود.

$$M = \frac{\sum_{i=1}^n W_i \times f_i}{\sum_{i=1}^n W_i} \quad (6-3)$$

در این رابطه M میانگین، n تعداد ویژگی‌های فرعی فاکتور مورد نظر، W_i وزن ویژگی فرعی i ام و f_i اندازه فاکتور فرعی i ام است.

۳-۱۱-۱- مثالی از روش تبدیل اندازه ی کیفی صفت های غیرهم وزن به مقادیر کمی

به عنوان مثال فرض می‌کنیم در یک پروژه نرم افزاری با صفت‌های سه حالت می‌خواهیم فاکتور احتمال ابتلا به خطای ماژول را محاسبه نماییم و ویژگی‌های فرعی آن اثرگذاری خطای ماژول، زیاد و شدت خطای ماژول، متوسط باشد. وزن هر کدام از ویژگی‌های فرعی نیز برابر است با: وزن اثرگذاری خطای ماژول برابر ۲، وزن شدت خطای ماژول برابر ۱.۵ برای اندازه گیری فاکتور احتمال ابتلا به خطای ماژول محاسبات زیر را خواهیم داشت.

ابتدا میانگین اندازه ی صفت‌ها را در رابطه (۷-۳) محاسبه می‌نماییم:

$$M = \frac{(2 \times 100)(1.5 \times 50)}{2 + 1.5} = \frac{275}{3.5} = 78.5 \quad (7-3)$$

میانگین بدست آمده بین ۵۰ تا ۱۰۰ است. حال اختلاف میانگین را از هر یک از مرزهای قبلی و بعدی محاسبه می‌نماییم. سپس اختلاف هر یک از مرزها با میانگین در عدد ۲ ضرب می‌گردد تا درصد مرز

$$(78.5 - 50) \times 2 = 57 \quad \text{دیگر بدست آید.}$$

$$(100 - 78.5) \times 2 = 43$$

به عبارت ساده تر درصد احتمال مرز بالای میانگین برابر است با :

$$(M - LB) \times 2 \quad (۸-۳)$$

و همینطور درصد احتمال مرز پایین میانگین برابر است با :

$$(UB - M) \times 2 \quad (۳-۹)$$

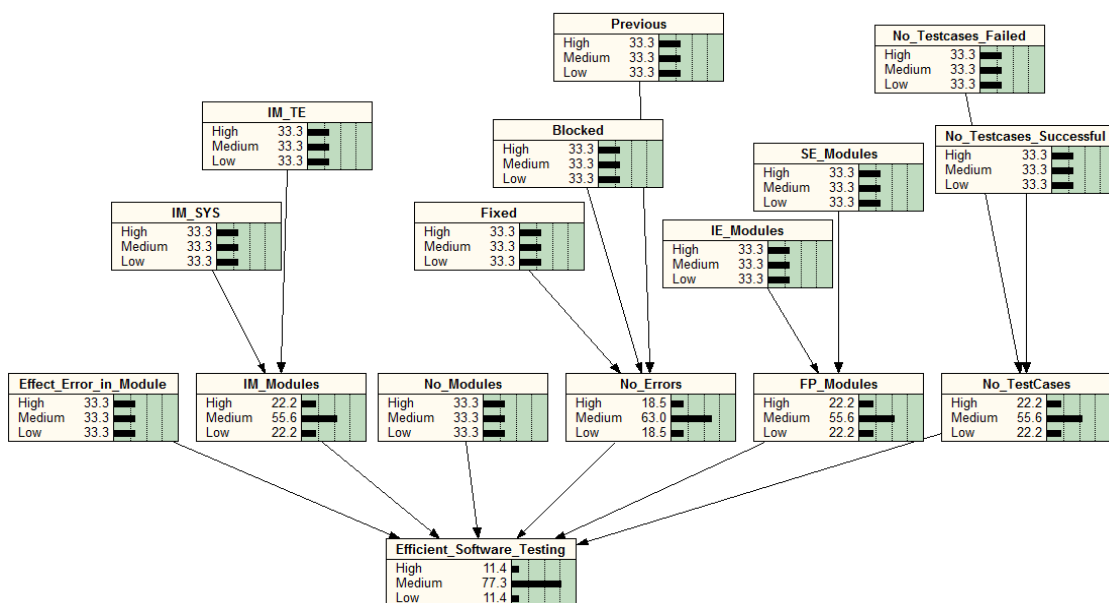
که در دو رابطه (۸-۳)(۹-۳) ارائه شده در بالا M میانگین ، LB مرز پایین میانگین و UB مرز بالای میانگین است.

جدول شماره ۳-۸ : نمونه سطری از جدول احتمال شرطی فاکتور احتمال ابتلا به خطای ماژول، سه حالت

	ویژگی های فرعی فاکتور احتمال ابتلا به خطای ماژول		اندازه کیفی		
	شدت خطای ماژول	اثرگذاری خطای ماژول	زیاد	متوسط	کم
اندازه ی صفت	متوسط	زیاد	۴۳	۵۷	۰
وزن صفت	۱,۵	۲			

۳-۱۲- پیاده سازی مدل تست کارآمد نرم افزار با استفاده از نرم افزار Netica

نرم افزار Netica یکی از نرم افزارهایی است که امروزه برای پیاده سازی شبکه های بیزی مورد استفاده قرار می گیرد. این نرم افزار یکی از ساده ترین و راحت ترین رابط های کاربری را ارائه می کند و در عین حال از جمله قدرتمندترین نرم افزارهای پیاده سازی شبکه های بیزی به حساب می آید. در شکل ارائه شده زیر مدل پیشنهادی تست کارآمد نرم افزار با صفت های سه حالتی که با استفاده از نرم افزار Netica پیاده سازی شده را نشان می دهد.



شکل ۳-۴: پیاده سازی مدل تست کارآمد نرم افزار با استفاده از نرم افزار Netica

همانطور که در شکل (۳-۴) مشخص است گراف مدل پیشنهادی ما بدون مشاهده ی هیچکدام از ویژگی های فرعی، احتمالاتی را از روی جداول احتمال شرطی برای ویژگی های فرعی، فاکتورهای کیفی و در نهایت کارآمد بودن اولویت بندی موارد تست نرم افزار در نظرخواهد گرفت. اندازه ی هر یک از حالت های ویژگی های فرعی یکسان و برابر با 33.33 است. چرا که با احتمال یکسان و سه حالت در نظر گرفته شده بودند. ولی اندازه ی حالت ها در فاکتورهای کیفی متفاوت است که آنها را در جدول (۳-۹) نمایش می دهیم.

جدول شماره ۳-۹: اندازه درصد احتمال حالت ها در فاکتورها بدون مشاهده ویژگی های فرعی، سه حالت

فاکتورهای کیفی	کم	متوسط	زیاد
تعداد موارد تست	22.2	55.6	22.2
احتمال ابتلا به خطای ماژول	22.2	55.6	22.2
تعداد خطاها	18.5	63	18.5
تعداد ماژول ها	33.3	33.3	33.3
اهمیت ماژول	22.2	55.6	22.2
اثر بروز خطا در ماژول	33.3	33.3	33.3

اندازه ی درصد احتمال حالت ها در کارآمدی تست نرم افزار بدون مشاهده ی ویژگی های فرعی برای حالت های کم و زیاد برابر ۱۱,۴ و برای حالت متوسط برابر ۷۷,۳ خواهد بود. نکته قابل توجه این است که استفاده کنندگان (مهندسان نرم افزار یا مهندسان تست) این مدل بسته به پروژه مورد نظرشان ممکن است احتمال هریک از ویژگی های فرعی را متفاوت در نظر بگیرند که این ویژگی منعطف بودن مدل پیشنهادی را نشان می دهد.

۱۳-۳- پر کردن جدول احتمال شرطی با استفاده از کد نویسی

در مواردی که تعداد سطرهای جدول احتمال شرطی شبکه‌ی بیزی طولانی است و امکان پیاده سازی گراف با نرم افزارهایی با رابط گرافیکی وجود ندارد از روش کدنویسی یا برنامه‌هایی که کد نویسی را پشتیبانی می کنند استفاده می نماییم. یکی از این برنامه‌ها BNT است که شبکه‌های بیزی را توسط MATLAB پیاده سازی می نماید. در ادامه نیز بخش‌هایی از کار انجام شده در رویکرد پیشنهادی مان را برای پر کردن جداول احتمال شرطی برای صفت‌های سه حالت با استفاده از روش کد نویسی نشان داده‌ایم. به دلیل اینکه BNT مقادیر بین صفر تا یک را برای متغیر هایش می پذیرد ما نیز در پیاده سازی خود و کدنویسی از این بازه مقادیر جداول احتمال شرطی خود را پر خواهیم کرد.

شکل ۳-۵: شبه کد پر کردن جدول احتمال شرطی برای اهمیت ماژول با دو ویژگی فرعی

```

1. Algorithm CPT with TWO metric(i,j)
2. Var : I,j, Avg
3. {
4. For (i=0 ; i<3; i++)
5. For (j=0 ; j<3 ; j++)
6. Avg=(i+j)/2
7. if (Avg=0) {
8. CPT(I,J,0)=1 ; CPT(I,J,1)=0 ; CPT(I,J,2)=0 ; }
9. Else if(Avg> 0 and Avg <1) {
10. CPT(I,J,0)=1- Avg ; CPT(I,J,1)=Avg ; CPT(I,J,2)=0 ; }
11. Else if (Avg=1){
12. CPT(I,J,0)=0 ; CPT(I,J,1)=1 ; CPT(I,J,2)=0 ; }
13. Else if(Avg> 1 and Avg <2) {
14. CPT(I,J,0)=0 ; CPT(I,J,1)=2-Avg ; CPT(I,J,2)= Avg-1 ; }
15. Else if (Avg=2) {
16. CPT(I,J,0)=0 ; CPT(I,J,1)=0 ; CPT(I,J,2)=2 ; }
}

```

```

1. Algorithm CPT with Three metric(i,j,k)
2. Var : I,j, k, Avg
3. {
4.   For (i=0 ; i<3; i++)
5.   For (j=0 ; j<3 ; j++)
6.   For(k=0; k<3; k++)
7.   Avg=(i+j+k)/3
8.   if (Avg=0) {
9.     CPT(I,J,k,0)=1 ; CPT(I,J,k,1)=0 ; CPT(I,J,k,2)=0 ; }
10.  Else if(Avg> 0 and Avg <1) {
11.    CPT(I,J,k,0)=1- Avg ; CPT(I,J,k,1)=Avg ; CPT(I,J,k,2)=0 ; }
12.  Else if (Avg=1){
13.    CPT(I,J,k,0)=0 ; CPT(I,J,k,1)=1 ; CPT(I,J,k,2)=0 ; }
14.  Else if(Avg> 1 and Avg <2) {
15.    CPT(I,J,k,0)=0 ; CPT(I,J,k,1)=2-Avg ; CPT(I,J,k,2)= Avg-1 ; }
16.  Else if (Avg=2) {
17.    CPT(I,J,k,0)=0 ; CPT(I,J,k,1)=0 ; CPT(I,J,k,2)=1 ; }
18. }

```

شکل ۶-۳: شبه کد پر کردن جدول احتمال شرطی برای تعداد خطاها با سه ویژگی فرعی

۱۴-۳- نمونه جداول احتمال شرطی فاکتورهای تست و کارآمدی اولویت بندی

در شکل های زیر نمونه هایی از جداول احتمال شرطی فاکتورها و کارآمدی اولویت بندی برای صفت سه حالت نشانه داده شده است. در ضمن این جداول احتمال شرطی در نرم افزار Netica که یکی از ابزارهای قدرتمند در پیاده سازی شبکه های بیزی است انجام گرفته است. شکل (۷-۳) جدول احتمال شرطی برای معیار تعداد موارد تست را که از دو ویژگی فرعی به نام های تعداد موارد تست موفق شده و تعداد موارد تست شکست خورده می باشد.

No_Testcases_Successful	No_Testcases_Failed	High	Medium	Low
High	High	100.00	0.000	0.000
High	Medium	50.000	50.000	0.000
High	Low	0.000	100.00	0.000
Medium	High	50.000	50.000	0.000
Medium	Medium	0.000	100.00	0.000
Medium	Low	0.000	50.000	50.000
Low	High	0.000	100.00	0.000
Low	Medium	0.000	50.000	50.000
Low	Low	0.000	0.000	100.00

شکل ۷-۳: جدول احتمال شرطی تعداد موارد تست

در شکل (۸-۳) جدول احتمال شرطی را برای معیار تعداد خطاها که خود شامل سه ویژگی فرعی به نام های کامل شده، مسدود شده و موجود از قبل می باشد را نشان می دهد.

Fixed	Blocked	Previous	High	Medium	Low
High	High	High	100.00	0.000	0.000
High	High	Medium	66.667	33.333	0.000
High	High	Low	33.333	66.667	0.000
High	Medium	High	66.667	33.333	0.000
High	Medium	Medium	33.333	66.667	0.000
High	Medium	Low	0.000	100.00	0.000
High	Low	High	33.333	66.667	0.000
High	Low	Medium	0.000	100.00	0.000
High	Low	Low	0.000	66.667	33.333
Medium	High	High	66.667	33.333	0.000
Medium	High	Medium	33.333	66.667	0.000
Medium	High	Low	0.000	100.00	0.000
Medium	Medium	High	33.333	66.667	0.000
Medium	Medium	Medium	0.000	100.00	0.000
Medium	Medium	Low	0.000	66.667	33.333
Medium	Low	High	0.000	100.00	0.000
Medium	Low	Medium	0.000	66.667	33.333
Medium	Low	Low	0.000	33.333	66.667
Low	High	High	33.333	66.667	0.000
Low	High	Medium	0.000	100.00	0.000
Low	High	Low	0.000	66.667	33.333
Low	Medium	High	0.000	100.00	0.000
Low	Medium	Medium	0.000	66.667	33.333
Low	Medium	Low	0.000	33.333	66.667
Low	Low	High	0.000	66.667	33.333
Low	Low	Medium	0.000	33.333	66.667
Low	Low	Low	0.000	0.000	100.00

شکل ۸-۳: جدول احتمال شرطی تعداد خطاها

در شکل (۹-۳) جدول احتمال شرطی را برای معیار اهمیت ماژول را که خود شامل دو ویژگی فرعی به نام های اهمیت از نظر سیستم و اهمیت از نظر مهندس تست می باشد را نشان می دهد.

IM_SYS	IM_TE	High	Medium	Low
High	High	100.00	0.000	0.000
High	Medium	50.000	50.000	0.000
High	Low	0.000	100.00	0.000
Medium	High	50.000	50.000	0.000
Medium	Medium	0.000	100.00	0.000
Medium	Low	0.000	50.000	50.000
Low	High	0.000	100.00	0.000
Low	Medium	0.000	50.000	50.000
Low	Low	0.000	0.000	100.00

شکل ۹-۳ : جدول احتمال شرطی اهمیت ماژول

در شکل (۱۰-۳) بخشی از جدول احتمال شرطی مربوط به کارآمد بودن اولویت بندی موارد تست را با در نظر گرفتن تمامی معیارهای موجود نشان می دهد.

Effect_Error_in_Module	IM_Modules	No_Modules	No_Errors	FP_Modules	No_TestCases	High	Medium	Low
High	High	High	High	High	High	100.00	0.000	0.000
High	High	High	High	High	Medium	83.333	16.667	0.000
High	High	High	High	High	Low	66.667	33.333	0.000
High	High	High	High	Medium	High	83.333	16.667	0.000
High	High	High	High	Medium	Medium	66.667	33.333	0.000
High	High	High	High	Medium	Low	50.000	50.000	0.000
High	High	High	High	Low	High	66.667	33.333	0.000
High	High	High	High	Low	Medium	50.000	50.000	0.000
High	High	High	High	Low	Low	33.333	66.667	0.000
High	High	High	Medium	High	High	83.333	16.667	0.000
High	High	High	Medium	High	Medium	66.667	33.333	0.000
High	High	High	Medium	High	Low	50.000	50.000	0.000
High	High	High	Medium	Medium	High	66.667	33.333	0.000
High	High	High	Medium	Medium	Medium	50.000	50.000	0.000
High	High	High	Medium	Medium	Low	33.333	66.667	0.000
High	High	High	Medium	Low	High	50.000	50.000	0.000
High	High	High	Medium	Low	Medium	33.333	66.667	0.000
High	High	High	Medium	Low	Low	16.667	83.333	0.000
High	High	High	Low	High	High	66.667	33.333	0.000
High	High	High	Low	High	Medium	50.000	50.000	0.000
High	High	High	Low	High	Low	33.333	66.667	0.000
High	High	High	Low	Medium	High	50.000	50.000	0.000
High	High	High	Low	Medium	Medium	33.333	66.667	0.000
High	High	High	Low	Medium	Low	16.667	83.333	0.000
High	High	High	Low	Low	High	33.333	66.667	0.000
High	High	High	Low	Low	Medium	16.667	83.333	0.000
High	High	High	Low	Low	Low	0.000	100.00	0.000

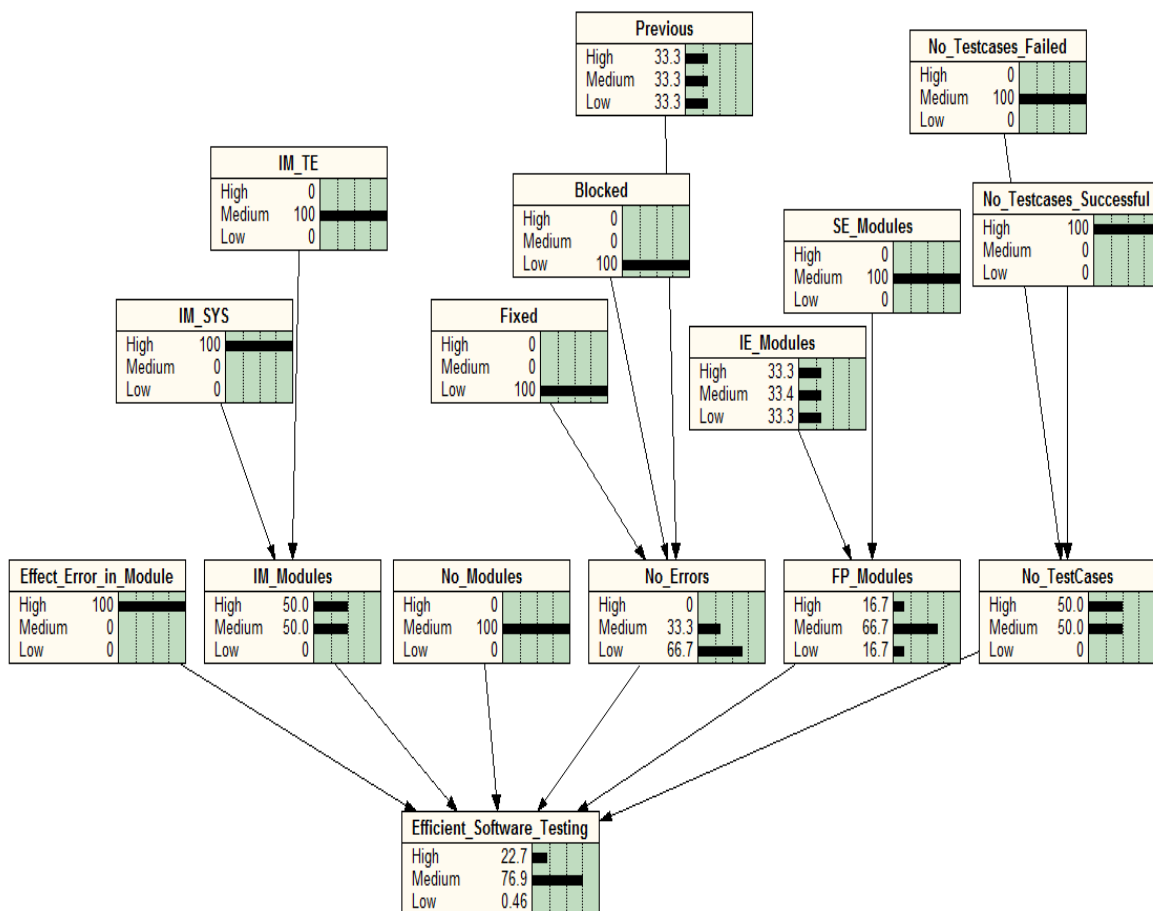
شکل ۱۰-۳ : جدول احتمال شرطی کارآمدی نهایی تست نرم افزار

۱۵-۳- پیاده سازی رویکرد پیشنهادی در مثال واقعی

در این بخش از پایان نامه رویکرد پیشنهادی برای اولویت بندی موارد تست نرم افزار را بر روی نرم افزار سیستم مدیریت پرونده های قضایی (سمپ) به طور واقعی پیاده سازی کرده ایم و بر طبق اطلاعات بدست آمده از سوی متخصصان تست و مهندسان نرم افزار شرکت نرم افزاری نوران تهران در جدول (۳-۱۰) نمایش داده شده اند که بر اساس این داده های اولیه میزان کارآمدی تست نرم افزار را با رویکرد خود محاسبه می نماییم و توسط گراف شبکه بیزی نشان می دهیم.

جدول شماره ۳-۱۰: اندازه ویژگی های فرعی در یک پروژه نرم افزاری نمونه

فاکتور های تست	ویژگی های فرعی	اندازه ویژگی های فرعی
تعداد موارد تست	تعداد تست موفق به اجرا شده	زیاد
	تعداد تست نا موفق به اجرا شده	متوسط
احتمال ابتلا به خطای ماژول	اثر گذاری خطای ماژول	متوسط
	شدت خطای ماژول	مشاهده نشده
تعداد خطاها	خطای رفع شده	کم
	خطای مسدود شده	کم
	خطای مانده از قبل	مشاهده نشده
تعداد ماژول ها		متوسط
اهمیت ماژول	اهمیت ماژول از نظر سیستم	زیاد
	اهمیت ماژول از نظر مهندس تست	متوسط



شکل ۳-۱۱: پیاده سازی مدل تست کارآمد نرم افزار بروی سیستم مدیریت پرونده های قضایی

با استفاده از نرم افزار Netica ما داده ها و اطلاعات بدست آمده را در تصمیم گیری خود لحاظ نموده ایم. همانطور که در شکل (۳-۱۱) مشاهده می کنید برخی از معیارها مقداری را ندارند و در جدول (۳-۱۰) مقدار مشاهده نشده برای آنها در نظر گرفته شده. از توانایی های رویکرد پیشنهادی این است که نتیجه نهایی نشان داده شده بر اساس داده های نا کافی بیان شده است و در زمانی که برخی از داده ها در دسترس نباشند و یا با احتمال بیان شوند ما باز هم قادر به تصمیم گیری خواهیم بود. گراف فوق پس از انجام مراحل مختلف و محاسبات کارآمدی اولویت بندی موارد تست را به صورت کمی بیان نموده است که عبارت است از: به احتمال ۰,۴۶ میزان کارآمدی اولویت بندی

موارد تست کم و به احتمال ۷۶,۹ متوسط و به احتمال ۲۲,۷ احتمال کارآمدی در اولویت بندی موارد تست زیاد می باشد.

فصل چهارم

محاسبات و یافته‌های تحقیق

4-1- ارزیابی مدل پیشنهادی

ما نیز همانند سایر روش‌های اولویت بندی برای ارزیابی روش خود از متریک APFD^{۱۸} [24] استفاده کردیم تا با این کار کارآمدی رویکرد پیشنهادی را به وضوح بتوانیم نمایش دهیم. برای نشان دادن اثر بخش بودن اولویت بندی موارد تست با روش پیشنهادی خودمان در مقابل دیگر روش‌های اولویت بندی موارد تست ما از رابطه (۴-۱) استفاده کرده‌ایم. در ادامه نتایج را به شکل نمودار نمایش می‌دهیم. جدول (۴-۱) تعداد خطاهای شناسایی شده توسط هر یک از موارد تست در مجموعه تست و کل زمانی که هریک از موارد تست بر روی نرم افزار سیستم مدیریت پرونده های قضایی می باشد. نتایجی از قبیل تعداد خطاهای شناسایی شده توسط هرکدام از موارد تست و زمان لازم که طول می کشد تا موارد تست بتوانند این خطاهای موجود را شناسایی کنند.

4-2- متریک (APFD) Average Percentage of Faults Detected

برای تعیین کمیت هدف که در اولویت بندی موارد تست، افزایش نرخ تشخیص خطاهای مجموعه‌ی تست می باشد ما نیز از این متریک استفاده می کنیم. این متریک توسط Elbaum و همکارانش توسعه یافته است [25]. این متریک متوسط نرخ شناسایی خطا را به ازاء درصد اجرای موارد تست موجود در مجموعه تست اندازه گیری می‌کند. بر اساس کارهای اولیه پیشنهاد شده، ممکن است اجرای موارد مدت زمان زیادی طول بکشد (ماه یا سال) بسته به طول اندازه مجموعه تست و یا اینکه هر یک از موارد تست در چه مقدار زمانی اجرا می شوند. به هر حال با استفاده از یک تکنیک اولویت بندی موثر و مرتب کردن موارد تست می توان نرخ خطای تشخیصی توسط همان موارد تست را افزایش داد. توجه داشته باشید که، رابطه نشان داده شده برای محاسبه متریک APFD تنها زمانی قابل استفاده است که دانش قبلی از خطاهای فعال یا در دسترس موجود باشد.

¹ Average of the Percentage of Faults Detected

این نیاز در رویکردهایی که برای تست رگرسیون استفاده می شوند قابل حل است زیرا در تست رگرسیون یک مجموعه تست اولیه در دست می باشد و با اطلاعاتی که از اجراهای قبلی موارد تست در دست است می توان این موضوع را فهمید. اما در تست توسعه نرم افزار که دیگر مجموعه تست با تعدادی موارد تست اولیه در دست نیست بایستی این مشکل را با تعدادی خطای دستی برطرف نمائیم. به گونه ای که تعدادی خطا را به صورت دستی به نرم افزار تزریق نموده و با اجرای موارد تست به کشف و شناسایی آنها بپردازیم.

جدول شماره 4-1: تعداد خطای شناسایی شده توسط موارد تست با توجه به زمان کل برای هر مورد تست

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
F1	*					*				
F2				*			*	*	*	
F3		*			*	*				*
F4							*			
F5		*						*	*	
F6				*						
F7				*	*					
F8		*	*							
F9						*				
F10	*									*
NO.fault	۲	۳	۱	۳	۲	۳	۲	۲	۲	۲
Time	۵	۷	۱۱	۴	۱۰	۱۲	۶	۱۵	۸	۹

۳-۴- اولویت بندی با کمک تکنیک شبکه های بیزی

برای انجام محاسبات در این بخش ابتدا بایستی نرخ شناسایی خطای هر کدام از موارد تست را به صورت جداگانه محاسبه کنیم. برای اینکار از رابطه (۴-۱) زیر استفاده می کنیم:

$$V_{Ti=fault/time} [25] \quad (نرخ تشخیص خطا) \quad (۴-۱)$$

با توجه به مقادیر موجود در جدول بالا محاسبات به شکل زیر با استفاده از رابطه (۴-۱) انجام می‌گیرد:

$$\begin{aligned} V_{T1} &= \frac{2}{5} = 0.4 & V_{T2} &= \frac{3}{7} = 0.42 \\ V_{T3} &= \frac{1}{11} = 0.09 & V_{T4} &= \frac{3}{4} = 0.75 \\ V_{T5} &= \frac{2}{10} = 0.2 & V_{T6} &= \frac{3}{12} = 0.25 \\ V_{T7} &= \frac{2}{6} = 0.33 & V_{T8} &= \frac{2}{15} = 0.133 \\ V_{T9} &= \frac{2}{8} = 0.25 & V_{T10} &= \frac{2}{9} = 0.22 \end{aligned}$$

با توجه به رابطه ارائه شده در بالا و محاسبات انجام گرفته اولویت بندی موارد تست بر اساس کاهش مقدار V_{Ti} خواهد بود، بدین ترتیب که V_{Ti} با مقدار بالاتر از اولویت بالاتری برخوردار است و به همین صورت تا کمترین مقدار V_{Ti} به مراتب از اولویت کمتری برخوردار خواهند بود.

از اینرو ترتیب اولویت در روش شبکه بیزی BN به شکل رابطه (۴-۲) زیر خواهد بود:

$$T_4, T_2, T_1, T_7, T_6, T_9, T_{10}, T_5, T_8, T_3 \quad (4-2)$$

بر طبق داده‌های جدول (۴-۱) $M=10$ و $N=10$ ، که M نشان دهنده تعداد خطاها بوده و N تعداد موارد تست می‌باشند.

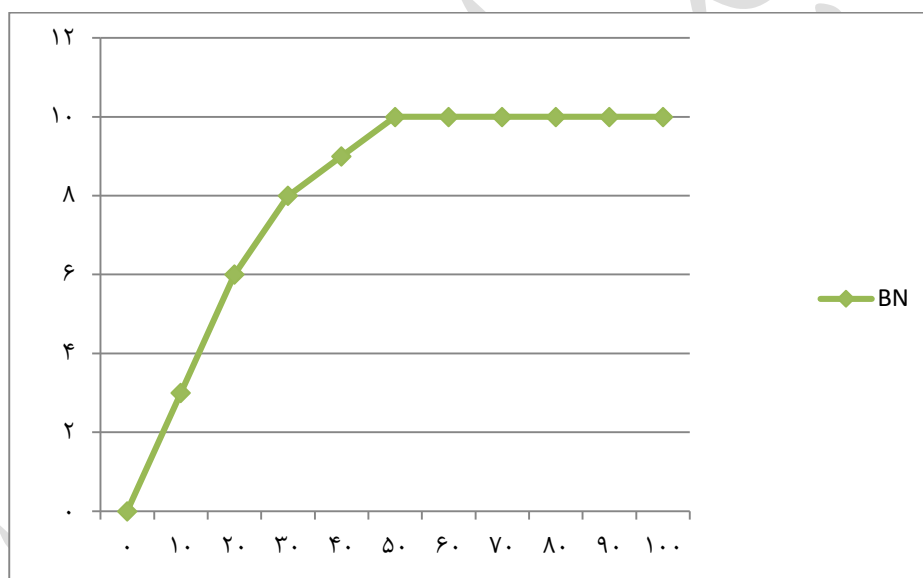
حالا تمامی مقادیر را در رابطه کامل APFD قرار می‌دهیم تا اینکه محاسبات نهایی را جهت نرخ تشخیص خطا انجام دهیم. در رابطه زیر TF_i عبارت است از موقعیت اولین تست از T است که در معرض خطای i قرار دارد.

$$APFD = 1 - \frac{TF_1 + TF_2 \dots TF_m}{nm} + \frac{1}{2n} \quad [25] \quad (4-3)$$

با توجه به مقادیر بدست آمده و مقادیر موجود در جدول در رابطه فوق جایگذاری می‌کنیم و محاسبه می‌کنیم.

$$+ \frac{1}{20} = 0.81APFD = 1 - \frac{3+1+2+4+2+1+1+2+5+3}{10 \times 10}$$

پس از اینکه مقادیر را در رابطه (4-3) جایگذاری کردیم نتیجه محاسبات بدین صورت بدست آمده اند و اینک ما برای نشان دادن نرخ شناسایی خطا در روش پیشنهادی خود از نمودار استفاده کرده ایم تا اینکه به خوبی روند کار را نمایش دهیم. لذا در شکل (4-1) نتیجه اجرای روش پیشنهادی را نشان می‌دهیم.



شکل 4-1: نمودار نرخ شناسایی خطا با روش شبکه ییزی برای اولویت بندی موارد تست

4-4- اولویت بندی با تکنیک اصلی

همانطور که در بخش قبل اشاره شد نرخ تشخیص خطا را با رابطه شماره (۴-۱) محاسبه می کنیم که برای روش پیشنهادی با توجه به مقادیر بدست آمده موارد تست را به ترتیب با بیشترین مقدار تا به کمترین مقدار V_{Ti} مرتب کردیم. برای دو روش دیگر تصادفی و اصلی نیاز به محاسبه این مقادیر نداریم زیرا اولویت بندی یا مرتب کردن در این روش ها بر اساس چینش تصادفی و مرتب سازی اولیه موارد تست انجام می گیرد و انجام این محاسبات کاری بیهوده خواهد بود.

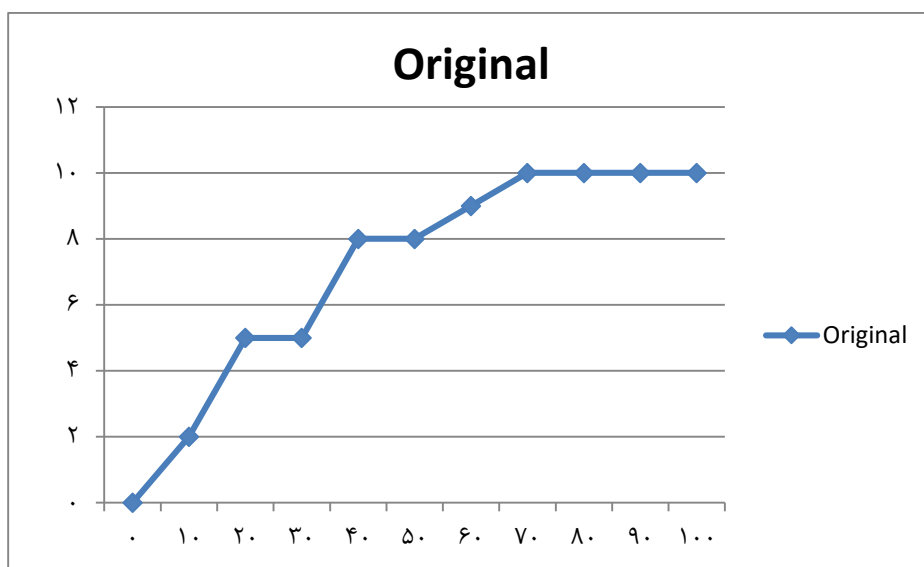
از اینرو ترتیب اولویت در روش اصلی به شکل رابطه (۴-۲) زیر خواهد بود :

$$T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10} \quad (4-2)$$

حالا تمامی مقادیر را در رابطه شماره (۴-۳) قرار می دهیم تا اینکه محاسبات نهایی را جهت نرخ تشخیص خطا انجام دهیم. بر طبق قرار گرفتن موارد تست موقعیت هر خطا را در جدول شماره ۱۳ محاسبه می کنیم و در رابطه زیر جایگذاری می کنیم.

$$+ \frac{1}{20} = 0.72APFD = 1 - \frac{1+4+2+7+2+4+4+2+6+1}{10 \times 10}$$

پس از جایگذاری مقادیر در رابطه و انجام محاسبات مقدار نرخ تشخیص خطای روش تصادفی برابر با ۰,۷۲٪ شده است. در ادامه از شکل (۴-۲) برای نشان دادن روند پیشرفت کار و تشخیص (شناسایی) خطای روش اصلی استفاده کردیم تا به وضوح نشان دهیم که هر کدام از موارد تست چه میزان از خطاها را شناسایی می کنند و در نمودار به خوبی مشخص است که برخی از خطاها با موارد تست دیگری که در اولویت بندی از اولویت بالاتری برخوردارند شناسایی شده اند و به همین دلیل در برخی از جاهای نمودار هیچ تغییری مشاهده نمی شود.



شکل ۴-۲: نمودار نرخ شناسایی خطا با روش اصلی برای اولویت بندی موارد تست

۵-۴- اولویت بندی با تکنیک تصادفی

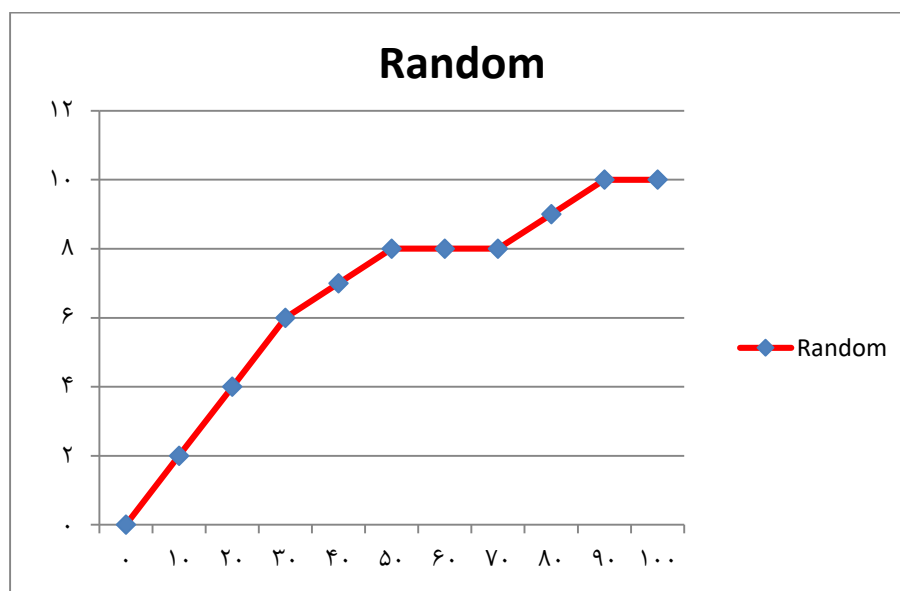
حال به بررسی میزان نرخ شناسایی خطا در روش تصادفی می پردازیم. برای این روش موارد تست از ترتیب خاصی پیروی نمی کنند و چیش موارد تست به صورت تصادفی بوده است. از اینرو ترتیب اولویت در روش تصادفی به شکل رابطه (۴-۴) زیر خواهد بود :

$$T_8, T_2, T_6, T_7, T_1, T_9, T_{10}, T_5, T_4, T_3 \quad (4-4)$$

اینک تمامی مقادیر مربوط به این ترتیب را در رابطه شماره (۴-۳) قرار می دهیم تا اینکه محاسبات نهایی را برای بدست آوردن نرخ تشخیص خطا بتوان انجام داد. یر طبق قرار گرفتن موارد تست موقعیت هر خطا را در جدول شماره (۴-۱) پیدا کرده و در رابطه زیر جایگذاری می کنیم.

$$+ \frac{1}{20} = 0.55APFD = 1 - \frac{5+9+2+4+2+9+9+2+3+5}{10 \times 10}$$

حال نتیجه محاسبات در روش تصادفی هم بدست آمده است و برای اینکه میزان پیشرفت و روند تشخیص خطاها را به تصویر بکشیم میتوانیم از نمودار بهره ببریم و ما اینکار را در ادامه انجام داده ایم. در ادامه شکل (۴-۳) نمودار نرخ شناسایی خطا را با روش تصادفی برای اولویت بندی موارد تست نمایش می دهیم.

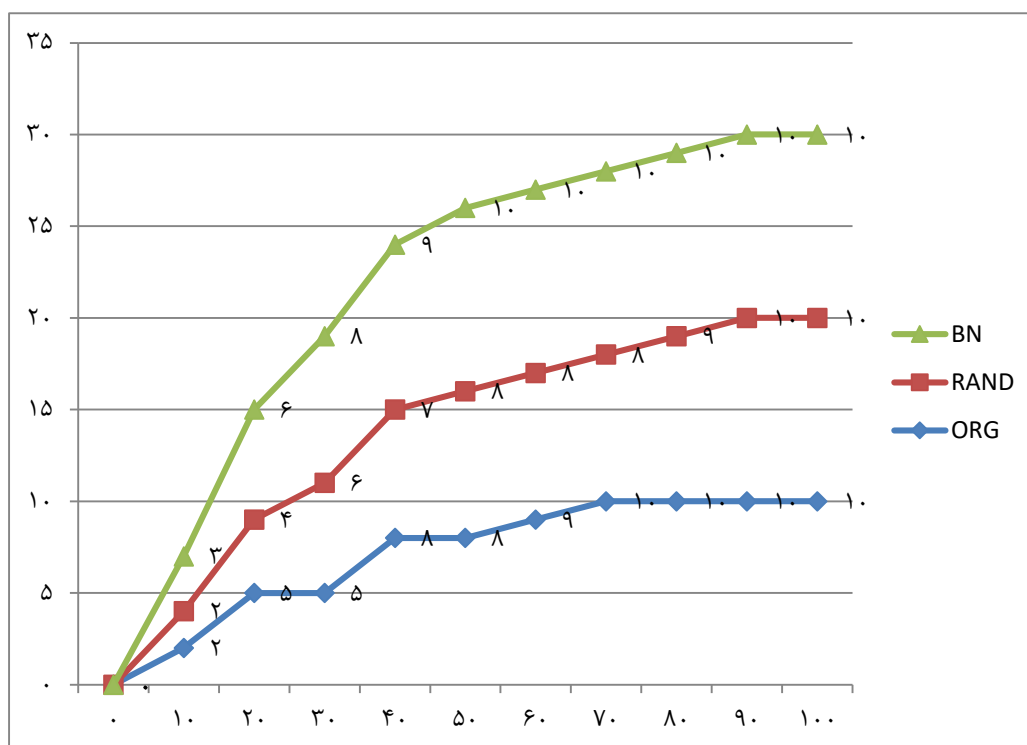


شکل ۴-۳: نمودار نرخ شناسایی خطا با روش تصادفی برای اولویت بندی موارد تست

۶-۴- مقایسه روش های اولویت بندی با روش پیشنهادی

حال برای مقایسه روش پیشنهادی خود در این پایان نامه با دیگر روش های موجود برای اولویت بندی موارد تست بایستی بر روی یک معیار مشترک در بین تکنیک های مختلف تمرکز کنیم. از این رو ما نیز بر معیار نرخ تشخیص (شناسایی) خطا که به عنوان معیار سنجش در بین تکنیک های اولویت بندی مورد استفاده قرار می گیرد، تاکید می کنیم.

برای انجام مقایسه روش پیشنهادی با دیگر روش ها در خصوص روند شناسایی خطا از شکل (۴-۴) استفاده کردیم و سه روش را باهم نشان دادیم.



شکل ۴-۴: نمودار مقایسه نتایج روش پیشنهادی با دو روش دیگر در نرخ شناسایی خطا

ما در رویکرد پیشنهاد شده در این پایان نامه را که، بر اساس ماژول های برنامه ساخته شده است ارزیابی را بدین صورت انجام می دهیم. ابتدا با مدنظر قرار دادن معیارهایی که در این پایان نامه مدنظر قرار دادیم و به صورت گراف نمایش دادیم و سپس اختصاص مقادیر کیفی به معیارهای مورد توجه مربوط به هر یک از ماژول ها توسط متخصصان تست که بر طبق معیارها برای هر کدام از ماژول های برنامه مشخص می شوند.

این مقادیر کیفی با توجه به مقدار کمی در نظر گرفته شده متناظر با هر مقدار کیفی به مقادیر کمی و در نهایت محاسبه نرخ تشخیص خطا می توانیم اولویت را به هر کدام از موارد تست موجود که برای پوشش دادن ماژول های برنامه در نظر گرفته ایم استفاده نمائیم. ما برای ارزیابی رویکرد پیشنهادی خود، نرم افزار مدیریت پرونده های قضایی (سمپ) در قوه قضاییه را انتخاب کردیم. ما برای اینکه بتوانیم یک ارزیابی درستی از روش خود انجام دهیم نیاز به داشتیم تا تعدادی خطای دستی را به نرم

افزار تزریق نمائیم، تا اینکه چگونگی شناسایی خطاها توسط موارد تست را بتوانیم رصد نمائیم. این نرم افزار همانطور که از نام آن مشخص است در مراکز دادگستری و مجتمع های قضایی برای ثبت پرونده های جدید، پیگیری روند کار پرونده های قضایی ایجاد گردیده است. لازم به ذکر است که این سیستم نرم افزاری بزرگ که هم اکنون در تمامی مجتمع های قضایی کل کشور نصب و راه اندازی شده است تاکنون به صورت دستی و توسط فرد غیر کارشناس مورد تست و ارزیابی قرار می گرفته است. و تمامی فرایند تست بکار گرفته شده برای آن در حد تست آلفا و اشکال زدایی بوده است. ضمن اینکه با بکارگیری روش پیشنهادی در این پایان نامه برای انجام تست بر روی این نرم افزار، تعداد خطای بیشتری را در زمانی کمتر از روش سنتی و دستی قبل شناسایی شده اند و این رضایت خاطر تیم برنامه نویسی را به دنبال داشته است. همانطور که در جدول (۴-۲) نشان داده شده است. تعداد خطاهای شناسایی شده در نسخه های مختلف این برنامه به این شکل است که، ما نمی توانیم به طور دقیق بگوئیم که با موارد تست خاصی می توانیم خطاهای برنامه را به طور کامل شناسایی کنیم. بکارگیری موارد تست باهمدیگر، همچنین اختصاص اولویت به هریک از موارد تست می توان نرخ تشخیص خطا را افزایش داد.

جدول ۴-۲: نتایج ارزیابی از نسخه های مختلف برنامه تحت تست

نرم افزار سمپ	تعداد موارد تست	تعداد خطاها	تعداد خط کد برنامه
نسخه ۳,۱	۰	۰	۳۸۰
نسخه ۳,۲	۱	۱	۴۴۳
نسخه ۳,۳	۴	۵	۴۷۶
نسخه ۳,۴	۸	۹	۵۰۱
نسخه ۳,۵	۱۰	۱۰	۵۳۲

در جدول (۴-۳) برخی تکنیک های اولویت بندی موارد تست مورد نظر و مقایسه در این پایان نامه را از سه جنبه با یکدیگر مقایسه می کنیم. در این مقایسه برتری تکنیک پیشنهاد شده در این پایان نامه را نشان می دهیم. اولین برتری عدم استفاده از مکانیزم بازخورد است که مدل پیشنهادی را از نظر

نداشتن بار اضافی نسبت به دیگر تکنیک های مقایسه شده برتری می بخشد. برتری بعدی تکنیک پیشنهادی سطح تست می باشد که همانگونه که در جدول نشان داده شده است، در سطح ماژول می باشد. در نهایت از حیث شواهد مورد استفاده تکنیک پیشنهادی ما از کلیه شواهد مفید (نیازمندی های کاربر، اطلاعات تغییر کد، اجراهای قبلی موارد تست، نظرات افراد خبره) استفاده می کند تا اینکه تصمیم گیری دقیق تری انجام پذیرد.

جدول ۴-۳: تکنیک های اولویت بندی استفاده شده در این ارزیابی

نام تکنیک	شواهد مورد استفاده	سطح تست	مکانیزم بازخورد
سنتی (دستی)	ندارد	کارکردی	ندارد
بهینه Optimal	پوشش	دستورالعمل ها	دارد
تصادفی Random	ندارد	کلاس	ندارد
اصلی Original	ندارد	کلاس	ندارد
شبکه های بیزی BN	پوشش، تغییر	ماژول	ندارد

فصل پنجم

نتیجه گیری و پیشنهادات

5-1 نتیجه گیری

رویکرد ارائه شده در این پایان نامه ، رویکردی جدید برای اولویت بندی موارد تست می باشد که برای بهبود بخشیدن به نرخ شناسایی خطا و همچنین شناسایی خطاهای شدید در تست رگرسیون و تست غیر رگرسیون می توان بکار برد ضمن اینکه تشخیص خطاهای شدید در مدل پیشنهادی بهبود یافته است. ما در این پایان نامه مدلی را پیشنهاد کرده ایم که بر اساس آن می توان اولویت بندی موارد تست را به گونه ای کارآمدتر از دیگر روش های پیشنهاد شده تا به حال انجام داد. زیرا روش هایی که تا کنون برای اولویت بندی موارد تست پیشنهاد شده اند و مورد مطالعه قرار گرفته اند هر یک دارای مزایا و معایبی هستند که عبارتند از : رویکرد اولویت بندی موارد تست نرم افزار بر اساس شبکه های بیزی [26,27,28,38,39,40] که مورد مطالعه قرار گرفتند، بر روی تست رگرسیون تمرکز کرده اند و منابع آنها برای جمع آوری داده های مورد نیاز جهت اولویت بندی موارد تست محدود به داده های قبلی از اجراهای موارد تست ، موارد تست موجود از تست های قبلی اجرا شده بر روی نرم افزار می باشد که این خود به عنوان یک عیب یا نقص برای آنها تلقی می گردد. زیرا اینگونه روش ها فقط برای تست رگرسیون می توانند مورد استفاده قرار گیرند و اگرچه بخواهیم برای تست رگرسیون یعنی به عنوان مثال تست توسعه نرم افزار از این روش ها استفاده کنیم قادر به چنین کاری نخواهیم بود زیرا ما در تست توسعه نرم افزار دیگر به داده های اجرای موارد تست و تعدادی موارد تست موجود دسترسی نداریم.

از دیگر مواردی که در کار مشابه با کار پیشنهادی ما وجود دارد این است که در رویکرد ارائه شده توسط [29,30] برکلاس های برنامه تمرکز شده است اما ما نیز در رویکردمان به استناد بر قانون پارتو بر مازول های برنامه تمرکز کردیم تا دقیق تر مسئله را مورد بررسی قرار دهیم و تست نرم افزار و اولویت بندی نهایی ما کارآمدی لازم را داشته باشد. همچنین ما نسبت به روش پیشنهادی که بر اساس ساختار برنامه عمل اولویت بندی موارد تست را انجام می دهد، [31] و به کار ما شباهت دارد معیارها و فاکتورهای بیشتری را در نظر گرفته ایم و همچنین رویکردمان را از حیث داده های ورودی فقط به ساختار برنامه محدود نکرده ایم و از دیگر داده های در دسترس از قبیل (نیازمندی های کاربر،

نظر افراد خبره و کارشناسان و اطلاعات قبلی اجرای موارد تست) نیز بهره می‌بریم تا اینکه بر اساس داده‌ها بتوانیم بهترین تصمیم‌گیری را انجام دهیم تا در نهایت اولویت بندی کارآمدی را در خروجی کار بدست آوریم با نرخ تشخیص خطای بیشتر برای خطاهای شدید موجود در برنامه تحت تست. مسئله دیگر اینکه در کارهای پیشنهادی تا قبل از کار پیشنهادی ما در هیچکدام از روش‌ها دیده نشده و محققان به آن توجه نکرده‌اند موضوع عدم قطعیت است. و اینکه زمان‌هایی پیش می‌آید که بایستی بر اساس اطلاعات ناکافی و احتمالی تصمیم‌گیری صورت گیرد و ما در رویکرد خویش با استفاده از شبکه‌های بیزی به این موضوع پرداخته‌ایم و آنرا در تصمیم‌گیری‌ها و در نهایت در اولویت بندی موارد تست لحاظ کرده‌ایم. در این پایان نامه وابستگی مدل پیشنهادی به داشتن اطلاعات دقیق و کامل از اجزای قبلی موارد تست کاهش یافته است و در مواقع وجود عدم قطعیت مدل پیشنهاد شده همانند روش‌های مشابه قابل استفاده و پاسخگو می‌باشد. در نهایت، مدل پیشنهاد شده قابل استفاده در تست رگرسیون و غیر رگرسیون می‌باشد.

2-5-2 پیشنهادات آینده

با توجه به قابلیت ها و امکاناتی که رویکرد پیشنهاد شده در این پایان نامه به دلیل بهره بردن از شبکه های بیزی با پشتوانه ریاضی و فرمال دارد، می توان در کارهای آینده از این رویکرد برای محاسبه پیچیدگی نرم افزار بر اساس معیارهای مختلف و در نظر گرفتن عدم قطعیت در آن میزان پیچیدگی محصول نرم افزاری را تعیین یا پیش بینی نمود. همچنین برای تست نرم افزارهای تحت وب یا انواع وب سایت ها می توان از رویکرد پیشنهادی بهره برد. در ضمن، قصد داریم در کارهای بعدی، با استفاده از آزمایشات تجربی و مدنظر قرار دادن مدل های هزینه- فایده مدل پیشنهادی خود را بهبود دهیم. ما در رویکرد پیشنهادی مکانیزم بازخورد را قرار نداده ایم و در نظر داریم برای کار بعدی که مدلی مقرون به صرفه خواهد بود از این مکانیزم هم بهره ببریم که این امر مستلزم تحقیق و پژوهش های بیشتری می باشد تا اینکه ثابت شود مدل پیشنهادی را دچار بار اضافی نمی سازد. یکی دیگر از گزینه های پیش رو برای توسعه و گسترش این کار در نظر گرفتن معیارهای ارزیابی بیشتر برای روش های اولویت بندی را می توان نام برد. در نهایت یکی دیگر از جنبه های بهبود و تکامل مدل پیشنهادی، بهره بردن از قابلیت یادگیری ماشین به همراه شبکه بیزی در کارهای آتی می باشد.

معادل انگلیسی	کلمه یا عبارت فارسی
الف	
Inference	استنتاج
Conditional Independence	استقلال شرطی
Fault Proneness	احتمال ابتلا به خطا
Importance of Module	اهمیت هر ماژول
Conditional probabilities	احتمالات شرطی
Debugging	اشکال زدایی
Software Testing	آزمایش نرم افزار
Impact of failure	اثرگذاری هر خطا
Components	اجزاء
Assessment	ارزیابی
ب	
Obtain	بدست آوردن
Improvement	بهبود
پ	
Predicted	پیش بینی
Test Coverage	پوشش تست

Tracking	پیگیری
ت	
Regression Testing	تست رگرسیون
Performance Testing	تست عملکرد
Load Testing	تست بار
Stress Testing	تست استرس
Security Testing	تست امنیت
White Box Testing	تست جعبه سفید
Black Box Testing	تست جعبه سیاه
ج	
Conditional probability table	جدول احتمال شرطی
چ	
Framework	چارچوب
Challenges	چالش ها
خ	
Severe Error	خطای شدید
Expert	خبیره
د	
Historical data	داده های تاریخی
س	
Program Structure	ساختار برنامه
ش	
Bayesian networks	شبکه بیزی
Severity of each error	شدت هر خطا
ع	

Uncertainty	عدم قطعیت
	ک
Performance	کارایی
Efficient	کارآمد
	گ
Graph	گراف
	م
Module	ماژول
System Integration	مجموع سازی سیستم
Test cases	موارد آزمون
Metrics	متریک
Limitation	محدودیت
	ن
User Requirements	نیازمندی های کاربر

معادل فارسی	کلمه یا عبارت انگلیسی
A	
ارزیابی	Assessment
B	
تست جعبه سیاه	Black Box Testing
شبکه بیزی	Bayesian networks
C	
استقلال شرطی	Conditional Independence
احتمالات شرطی	Conditional probabilities
اجزاء	Components
جدول احتمال شرطی	Conditional probability table
چالش ها	Challenges
D	
اشکال زدایی	Debugging
E	
خبیره	Expert
کارآمد	Efficient
F	
چارچوب	Framework
احتمال ابتلا به خطا	Fault Proneness
H	
داده های تاریخی	Historical data
I	
استنتاج	Inference

Importance of Module	اهمیت هر ماژول
Impact of failure	اثرگذاری هر خطا
Improvement	بهبود
L	
Limitation	محدودیت
Load Testing	تست بار
M	
Module	ماژول
Metrics	متریک
O	
Obtain	بدست آوردن
P	
Predicted	پیش بینی
Performance Testing	تست عملکرد
Program Structure	ساختار برنامه
Performance	کارایی
R	
Regression Testing	آزمون رگرسیون
S	
System Integration	یکپارچه سازی سیستم ها
Severity of each error	شدت هر خطا
Severe Error	خطا شدید
Stress Testing	تست استرس
Security Testing	تست امنیت
Software Testing	نرم افزار تست
T	

Test Coverage	پوشش آزمون
Tracking	پیگیری (ردیابی)
Test cases	موارد آزمون
U	
Uncertainty	عدم قطعیت
User Requirements	نیازمندی های کاربر
V	
Verification	تایید (صحت)
Validation	اعتبارسنجی

- [1] Pressman, R., Software Engineerig : A Practitioner's Approach, ٥th edition, Mc Graw-Hill, .٢٠٠٥
- [2] Bruegge Bernd, Dutoit, H.,Allen;Object Oriented Software Engineering, Prentice Hall, 2000
- [3] Masaharu Munetomo, Naoya Nurao, Kiyoshi Akama. Introducing Assignment Ffunctions to Bayesian Optimization Algorithms.ELSEVIER. ٦ august ٢٠٠٧
- [4] Nachiappan Nagappan. A Software Testing and Reliability Early Warning (STREW) Metric Suite. A dissertation submitted to the graduate faculty of North Carolina State University, 2005
- [5] W. Jun, Z. Yan, and J. Chen, "Test Case Prioritization Technique Based on Genetic Algorithm," in Internet Computing & Information Services (ICICIS), 2011 International Conference on, 2011, pp. 173-175.
- [6] H. Do, G. Rothermel, and A. Kinneer. Prioritizing JUnittest cases: An empirical assessment and cost-benefitsanalysis.Empirical Software Engineering: An International Journal, 11(1):33–70, 2006.
- [7] S. Kim and J. Baik, "An effective fault aware test case prioritization by incorporating a fault localization technique," in Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2010, p. 5.
- [8] W. Jun, Z. Yan, and J. Chen, "Test Case Prioritization Technique Based on Genetic Algorithm," in Internet Computing & Information Services (ICICIS), 2011 International Conference on, 2011, pp. 173-175.
- [9] B. Suri and S. Singhal, "Analyzing test case selection & prioritization using ACO," ACM SIGSOFT Software Engineering Notes, vol. 36, pp. 1-5, 2011.
- [١٠] S. Mohanty, A. A. Acharya, and D. P. Mohapatra, "A survey on model based test case prioritization," International Journal of Computer Science and Information Technologies, vol. 2, pp. 1002-1040, 2011.
- [11] C. Titus. B, G. Gheorghio, J.Huggins, "An Introduction to testing web application with twill and selenium" O'Reilly, June 1,2007
- [12] Pressman, R., Software Engineerig : A Practitioner's Approach, 5thedition, McGraw-Hill, 2005.

- [13] Bruegge Bernd, Dutoit, H., Allen; Object Oriented Software Engineering, Prentice Hall, 2000.
- [14] Kendall & Kendall; System Analysis and Design; 4th edition, Prentice Hall, 1999.
- [15] Parrington, N., Marc Roper; Understanding Software Testing; John Wiley & Sons, 1999.
- [16] Holmes, Jim; Object Oriented Computer Construction, Prentice Hall, 1995.
- [17]. IEEE (1990). IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York: IEEE. ISBN 1559370793.
- [18] Ralph Brettschneider (Motorola). Is your software ready for release? IEEE Software, July 1989.
- [19] Cowell, R. 1999, "Introduction to Inference for Bayesian Networks". In Michael I. Jordan, editor, "Learning in Graphical Models", pages 9--26.
- [20] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, second edition, Prentice Hall, 2003.
- [21] D. Heckerman, D. Geiger, and D. M. Chickering, *Learning Bayesian Networks: The Combination of Knowledge and Statistical Data*, Machine Learning, vol. 20, pp. 197-243, 1995.
- [22] Martin Neila, b., Manesh Tailor, b., David Marqueza, Norman Fentona, B., Peter Heart. Modelling Dependable Systems Using Hybrid Bayesian Networks. ELSEVIER. 2007.
- [23] Craig s. Galbraith, Alex f. Denoble, Sanford b. Ehrlich, Doug m. Kline. Can Experts Really Assess Future Technology Success? Neural Network and Bayesian Analysis of Early Stage Technology Proposals. ELSEVIER. 2007.
- [24] Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, Sebastian Elbaum, Cost- cognizant Test Case Prioritization, 2006
- [25] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, "Prioritizing Test Cases for Regression Testing," IEEE Trans. Software Eng., vol. 27, no. 10, pp. 929-948, Oct. 2001.
- [26] R. Torkar, N. M. Awan, A. K. Alvi, and W. Afzal, "Predicting software test effort in iterative development using a dynamic Bayesian network," in 21st IEEE International Symposium on Software Reliability Engineering, 2010.
- [27] S. Mirarab and L. Tahvildari, "A prioritization approach for software test cases based on bayesian networks," Fundamental Approaches to Software Engineering, pp. 276-290, 2007.

- [28] M.J. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker. Empirical studies of a prediction model for regression test selection. *IEEE Transactions on Software Engineering*, 27(3):248–263, 2001.
- [29] S. Mohanty, A. A. Acharya, and D. P. Mohapatra, "A survey on model based test case prioritization," *International Journal of Computer Science and Information Technologies*, vol. 2, pp. 1002-1040, 2011.
- [30] Y. Singh, A. Kaur, and B. Suri, "Test case prioritization using ant colony optimization," *ACM SIGSOFT Software Engineering Notes*, vol. 35, pp. 1-7, 2010.
- [31] Z. Ma and J. Zhao, "Test case prioritization based on analysis of program structure," in *Software Engineering Conference, 2008.APSEC'08. 15th Asia-Pacific*, 2008, pp. 471-478.
- [32] W.E. Wong, J.R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 264–274, November 1997.
- [33] S. Elbaum, A. Malishevsky, and G. Rothermel Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, February 2002.
- [34] Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, Sebastian Elbaum, Cost- cognizant Test Case Prioritization, 2006 .
- [35] D. Jeffrey and N. Gupta. Test case prioritization using relevant slices. In *Proceedings of the Annual International Computer Software and Applications Conference*, pages 411–420, September 2006.
- [36] J. Jones and M.J. Harrold. Test suite reduction and prioritization for modified condition/decision coverage. *IEEE Transactions on Software Engineering*, 29(3):193–209, 2003.
- [37] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *Software Engineering, IEEE Transactions on*, vol. 36, pp. 593-617, 2010.
- [38] S. Fine and A. Ziv, "Coverage directed test generation for functional verification using bayesian networks," in *Design Automation Conference, 2003. Proceedings*, 2003, pp. 286-291.
- [39] H. Ziv and D. J. Richardson, "Constructing Bayesian-network models of software testing and maintenance uncertainties," in *Software Maintenance, 1997. Proceedings., International Conference on*, 1997, pp. 100-109.

[40] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in Proceedings of the eighteenth international symposium on Software testing and analysis, 2009, pp. 201-212.

منابع پایه پرسشنامه

Abstract

Regression makes sure that the changes don't have an undesirable effect on software current behavior. One approach to regression testing is to prioritize it that consequently increases the efficiency of test. Prioritizing test cases methods, improve detection ratio of regression test fault. Though most previous proposed methods for detected fault during the test keep in mind the equality rate, it's not so in practice. Moreover, most existing methods rely on the previous information required during previous test cases or changes in program code; though few of them could be directly used for non-regression test. Throughout this article in order to improve the detection rate of fault in regression tests as well as non-regression test, we proposed a new approach for prioritizing test Cases, using bayesian networks that rely on the structure analysis of the program being tested. To implement the proposed approach, we used some tools named Netica and code programming in MATLAB, also in this article we focus on metric APFD (Average Percent of Fault Detection) to evaluate the fault detection rate. A Bayesian Networks based method based on program structure analysis will be presented further in this article. Finally, with some case study, we will present the application of our approach. Results of comparing proposed approach with some other methods imply that our approach produces a more exact outcome. Besides, this method is able to make decisions in non-certain situations which is considered to be a privilege. Moreover than giving an account to the criteria of possibility of fault occurrence and the importance of fault, the proposed approach considers some other criteria such as ratio of module fault influence upon other modules and fault severity of each module. Therefore it is a more comprehensive method comparing to similar methods. The proposed approach in addition to be used in regression test, could be useful in software development tests.

Keywords: Regression Test, Bayesian Networks, Test Cases Prioritizing, Fault Detection Rate



Mazandaran University of Science and Technology

Faculty of Engineering

Department of Computer Engineering

Thesis Title:

**Effective Software Test Cases Prioritization using
Bayesian Networks**

**A Thesis submitted to the Graduate Studies Office
in partial fulfillment of the requirements for
the degree of M.Sc. in Information Technology Engineering**

By: Mohammad Rezaqolipour

Supervisor:

Dr. Hossein Momeni

Advisor:

Dr. Babak Shirazi

Summer 2013

منابع پایه پرسشنامه