پردیس علوم
دانشکده ریاضی، آمار و علوم کامپیوتر

کاربرد هاي داده کاوي در مهندسي نرم افزار

نگارنده

زهرا امیدی

استاد راهنما

مهندس ابراهیم نقیب زاده مشایخ

پایان‌نامه برای دریافت درجه کارشناسی
در رشته علوم کامپیوتر

تیر ماه ۱۳۹۷

چکیده

دسترسی بیشتر به داده های ایجاد شده به عنوان بخشی از فرآیند تولید نرم‌افزار به ما این امکان را می دهد تا تکنیک های جدید تجزیه و تحلیل را براي این داده ها به کار گیریم و از نتایج آن ها برای هدایت بهینه سازی فرایند استفاده کنیم. در این نوشتار، منابع اطلاعاتی مختلفی را معرفی می کنیم و اصول و تکنیک های داده کاوی را در مورد داده های مهندسی نرم افزار مورد بحث قرار می دهیم. داده هایی که می توانند استخراج شوند، توسط بیشتر بخش های فرآیند تولید ایجاد مي شوند : استخراج نیازها، تجزیه و تحلیل تولید، آزمایش، اشکال زدایی و نگهداری. بر اساس این رده بندی، ما روش های کاوشي را که مورد استفاده قرار گرفته اند، مورد بررسی قرار می دهیم و آن ها را با توجه به بخش های متناظر فرایند تولید و وظایفي که دارند، رده بندی می کنیم. بنابراین این مطالعه، محققان را با یک مرور کلی از تکنیک های داده کاوی که به داده های مهندسی نرم افزار مربوط می شوند، آشنا می کند و به محققان در انتخاب تکنیک های داده کاوی مناسب برای کار خود کمک مي کند.

پیش گفتار

پیشرفت های اخیر تکنولوژی مدیریت داده ها به همراه ابزارها و روش های جمع آوری کارا، ذخیره سازی و نمایه سازی داده ها را فراهم می کنند. حجم زیادی از داده ها در تولید نرم افزار تولید می شوند که سازمان های نرم افزاری آن ها را به امید استخراج اطلاعات مفید و به دست آوردن درک بهتري از فرآیندها و محصولاتشان جمع آوري مي كنند. همچنین مقدار قابل توجهی از فراداده هاي پروژه هاي نرم افزاري متن باز (OSS) در مخازن نرم افزار جمع آوري و نگهداري مي شوند. به عنوان مثال، پروژه $FLOSSmole$[1] داده‌هاي چند پروژه ي نرم افزاري متن باز را باهم تلفيق مي كند و آزادانه آنها را در قالب های مختلف ارائه می دهد. پس یک فراوانی افزایشي در داده هاي ذخیره شده در مخزن نرم افزار وجود دارد که می تواند نقش مهمی در بهبود بهره وری و کیفیت نرم افزار داشته باشد. به ویژه، داده ها در تولید نرم افزار می توانند به نسخه های برنامه، ردیاب های اجرا، گزارش های خطا / اشکال و پکیج های متن باز اشاره کنند. لیست های پستی، انجمن های گفتگو و خبرنامه ها نیز اطلاعات مفیدي در مورد یک قسمت از نرم افزار ارائه می دهند. این رشد انفجاری توانایی مهندسان نرم افزار برای جمع آوری و ذخیره اطلاعات مهندسي نرم افزار، نیاز به ابزاري جدید، مقیاس پذیر و کارآمد برای تجزیه و تحلیل داده ها را ایجاد کرده است.

داده کاوی تکنیک هایی را برای تجزیه و تحلیل و استخراج الگوهای جدید و جالب از داده ها فراهم می کند.به طور رسمی، داده کاوي به عنوان فرآیند استنتاج اطلاعات ناشناخته و مفید بالقوه از مجموعه داده ها، تعریف شده است. بنابراین استخراج اطلاعات مهندسی نرم افزار به تازگی توجه محققان

[1] $http://flossmole.org/$

را به عنوان یک وسیله امیدوار کننده برای رسیدن به هدف بهبود نرم افزار جلب کرده است. الگوهای استخراج شده دانش می توانند مهندسین نرم افزار را در پیش بینی، برنامه ریزی و درک جنبه های مختلف پروژه یاري کنند تا بتوانند به طور مؤثرتری از فعالیت های توسعه و مدیریت پروژه حمایت کنند. چالش های متعددی در کاوش مخازن نرم افزاري وجود دارد:

- داده های پیچیده : مهندسان نرم افزار معمولا از داده های با انواع منحصر بفردی برای انجام کار مهندسی نرم افزار استفاده می کنند. با این حال، حوزه مهندسی نرم افزار می تواند پیچیده باشد و در نتیجه، وظایف مهندسی نرم افزار به طور فزاینده ای نیاز به استخراج انواع داده های مرتبط چندگانه برای دستیابی به نتایج موثرتری دارد. علاوه بر این، مواردی وجود دارند که اطلاعات مهم، نه تنها به فقره های منحصر بفرد داده، بلکه به ارتباط بین آن ها وابسته است. بنابراین نیاز به تکنیک هایی که تجزیه و تحلیل انواع داده های پیچیده و همچنین ارتباط بین داده های مهندسي نرم افزار را در نظر می گیرند، در مهندسی نرم افزار، قوی تر از همیشه است.
- داده های بزرگ : حجم بسیار زیادی از اطلاعات در مخازن مهندسی نرم افزار جمع آوری و ذخیره می شود. در اغلب موارد مهندسان نرم افزار تنها چند مخزن محلي را برای انجام وظایف خود تجزیه و تحلیل می کنند. در حالي که، ممکن است مقدار بسیار کمي از داده هاي مربوطه در مخازن محلی مهندسی نرم افزار برای کمک به استخراج الگوهای مطلوب (به عنوان مثال استخراج الگوهای روش هاي API) وجود داشته باشد. اگر ما بتوانیم مخازن نرم افزاري در سطح اینترنت را کاوش کنیم، می توان این مشکل را حل کرد. همچنین تکنیک های کاوش را می توان براي بسیاری از مخازن نرم افزاری در یک سازمان یا در سراسر سازمان و یا حتی در کل جهان متن باز به کار برد. بنابراین نیاز به تکنیک هایي که قادر به کاوش کارآمد داده هاي با مقیاس هاي بزرگ هستند، بوجود مي آید.

در این نوشتار، ما یک مرور کلي از رویکردهایي که هدفشان متصل کردن بخش های تحقیقاتی داده کاوی و مهندسی نرم افزار به منظور ایجاد تکنیک

های کارآمدتر برای پردازش نرم افزار است ، ارائه می دهیم. در فصل اول، مقدمه ای بر مفاهیم و رویکردهای اصلي داده کاوی ارائه می دهیم، در حالی که در فصل دوم، انواع مختلفی از اطلاعات مهندسی نرم افزار را که می توانیم کاوش کنیم، توصیف می کنیم. و در فصل ۳ به بررسی روش هایی که از تکنیک های داده کاوی در زمینه ي مهندسی نرم افزار استفاده می کنند، می پردازیم. به ویژه، این مطالعه بر اساس ویژگی های اصلي زیر است:

۱. روش داده کاوی استفاده شده (فصل ۱)

۲. داده‌های مهندسی نرم‌افزار که روش‌هاي داده‌کاوي روي آن ها اعمال می‌شود (فصل ۲)

۳. وظایف مهندسی نرم‌افزار که به کمک روش‌هاي داده‌کاوي می‌توانند انجام شوند (فصل ۳).

بنابراین، هدف این مطالعه معرفی مفاهیم و تکنیک‌های اساسی به محققان است که بتوانند برای به دست آوردن درک بهتر از فرآیندهای مهندسی نرم‌افزار، داده‌کاوی را به کار ببرند. به طور موازی، محققان می‌توانند تکنیک‌های اخیر را برای درک بهتر اطلاعات پروژه، شناسایی محدودیت‌های فرایندهای فعلی و تعریف روش‌هایی که وظایف مهندسی نرم‌افزار را تسهیل می‌کنند، به کار ببرند. به طور خلاصه، مفاهیم کلیدی این کار عبارتند از:

- رده بندی رویکردهای استفاده شده در کاوش داده‌های مهندسی نرم‌افزار، با توجه به ناحیه‌هایي از مهندسی نرم‌افزار که آنها به کار مي‌روند.
- چارچوب تجزیه و تحلیل مبتنی بر ماتریس، پل ارتباطی مهندسی نرم‌افزار با روش‌های داده‌کاوی است.
- گردهم آوردن ناحیه هاي تحقیقاتي داده کاوي و مهندسی نرم‌افزار. تعدادی از رویکردهایی که از داده‌کاوی در وظایف مهندسی نرم‌افزار استفاده می کنند، روش های کاری جدیدی را برای پژوهشگران و متخصصان در مهندسی نرم‌افزار ارائه می‌دهند.

فهرست مطالب

فصل ۱
داده کاوی و کشف دانش

اهداف اصلی داده‌کاوی پیش‌بینی و توصیف است. پیش‌بینی برآورد کردن مقدار آینده یا پیش بيني مقدار متغیرهای هدف بر مبنای مطالعه‌ي ساير متغیرها است. توصیف به کشف الگوها به منظور کمک به ارائه‌ي داده‌ها به شیوه‌ای قابل فهم‌تر و قابل بهره برداری‌تر، تمرکز دارد. یک توصیف خوب، توضیح خوبی از رفتار داده‌ها ارائه مي‌دهد. اهمیت نسبي پیش بینی و توصیف برای کاربردهای مختلف داده‌کاوی، متفاوت است. با این حال، در مورد کشف دانش، به نظر می رسد که توصیف مهم‌تر از پیش بینی باشد، بر خلاف برنامه‌هاي تشخیص الگو و یادگیری ماشین، که پیش بینی مهم‌تر است.

داده‌کاوی، با توضیح و تجزیه و تحلیل عمیق مفاهیم و فرآیندهای نرم‌افزاري، به وظایف مهندسی نرم‌افزار کمک می کند. بر اساس تکنیک‌های داده‌کاوی، می‌توانیم روابط میان پروژه‌های نرم‌افزاری را استخراج کنیم. داده‌کاوی می‌تواند از اطلاعات استخراج شده برای ارزیابی پروژه‌های نرم‌افزاری و یا پیش‌بینی رفتار نرم‌افزار، بهره ببرد. تعدادی از روش‌های داده‌کاوی برای برآورده کردن نیازهاي برنامه های مختلف پیشنهاد شده‌اند. همه آن‌ها مجموعه ای از عملکردهای داده‌کاوی را برای شناسایی و توصیف الگوهای جالب در دانش استخراج شده از مجموعه داده‌ها فراهم مي‌کنند. در زیر ما به طور مختصر وظایف اصلی داده‌کاوی و نحوه استفاده از آن‌ها در مهندسی نرم‌افزار را شرح می‌دهیم.

- خوشه بندی - تکنیک یادگیری بی نظارت

خوشه بندی یکی از کارهاي مفید در زمینه داده کاوی است، که برای

کشف گروه‌ها و شناسایی توزیع‌ها و الگوهای جالب در داده‌های اساسی به کار گرفته مي‌شود. مساله‌ي خوشه بندي در مورد بخش بندی یک مجموعه داده‌ي داده شده، به گروه‌ها (خوشه‌ها) است به طوری که نقاط داده‌ي یک خوشه بیشتر به یکدیگر شبیه باشند تا به نقاط خوشه‌های مختلف. در فرایند خوشه بندی، هیچ رده‌ی از پیش تعریف شده و یا نمونه‌ای که نوع روابط مطلوب و معتبر بین داده‌ها را پیش بینی کند، وجود ندارد. بنابراین این مساله به عنوان یک فرآیند یادگیری بدون نظارت شناخته می‌شود. خوشه بندی را می‌توان برای ایجاد یک دید کلي از توزیع داده‌های اساسي و همچنین شناسایی خودکار داده‌هاي دورافتاده یا منزوي، به کار برد. در مهندسی نرم‌افزار، خوشه‌بندی می‌تواند برای تعریف گروه‌های مولفه‌های مشابه بر اساس تعداد تغییرات و معیار تعداد سیکلمات (تعداد مسیرهای مستقل خطی در کد منبع برنامه) استفاده شود.

- رده بندی - تکنیک یادگیری با نظارت

مسئله رده بندی به طور گسترده در زمینه‌هاي تحقیقاتي آمار، شناخت الگو و یادگیری ماشین به عنوان یک راه حل ممکن برای مسئله‌ي اکتساب دانش یا استخراج دانش مورد مطالعه قرار گرفته است. رده بندی یکی از وظایف اصلی در داده کاوی برای اختصاص دادن یک فقره داده به مجموعه‌ای از رده‌هاي از پیش تعریف شده، است. رده بندی را می‌توان به عنوان یک تابع (رده‌بند) تعریف کرد که یک فقره داده را به یکی از چند رده از پیش تعریف شده، نگاشت می‌کند. مجموعه‌ای از رده‌هاي از پیش تعریف شده و یک مجموعه‌ي آموزشی از نمونه‌های از پیش رده‌بندی شده، رده‌بندی را مشخص می‌کند. برعکس، فرایند خوشه‌بندی به نمونه ها و رده‌هاي از پیش تعریف شده وابسته نیست. هدف فرایند رده‌بندی، ایجاد یک مدل است که می‌تواند برای رده‌بندی فقره‌های آینده که رده‌بندی آنها ناشناخته است، مورد استفاده قرار گیرد. یکی از تکنیک‌های رده‌بندی که به طور گسترده استفاده مي‌شود، ساخت درخت‌های تصمیم است. آن‌ها را می‌توان برای کشف قوانین رده‌بندی برای یک ویژگی منتخب از یک مجموعه داده، با تقسیم کردن سیستماتیک اطلاعات موجود در این مجموعه داده، مورد استفاده قرار داد. درخت‌هاي تصمیم نیز یکی از ابزارهایی هستند که برای ساخت مدل‌های رده‌بندی

در زمینه مهندسی نرم‌افزار انتخاب شده اند. شکل ۱.۱ یک درخت رده‌بندی را نشان می‌دهد که برای ارائه‌ي یک سازوکار شناسایی مولفه‌های نرم‌افزاری خطرناک بر اساس ویژگی‌های مولفه و سیستم آن، ساخته شده است. بنابراین، بر اساس درخت تصمیم داده شده، می‌توانیم قانون زیر را که به تصمیم‌گیری در مورد خطاها در یک مولفه کمک می‌کند، استخراج کنیم:

IF (# of data bindings > 10) AND (it is part of a non real-time system) THEN the module is unlikely to have errors

- قوانین انجمنی و کاوش الگوهاي تکرارشونده

کاوش قوانین انجمني علاقه‌مندی بسیاري را به خود جلب کرده است؛ چرا که قوانین، یک روش مختصر برای ارائه‌ي اطلاعات مفید بالقوه است که به راحتی توسط کاربران نهایی قابل درک است.قوانین انجمني "همبستگی" اساسي بین ویژگي‌هاي یک مجموعه داده را نشان می‌دهد. این همبستگی‌ها به شکل زیر ارائه می‌شوند:

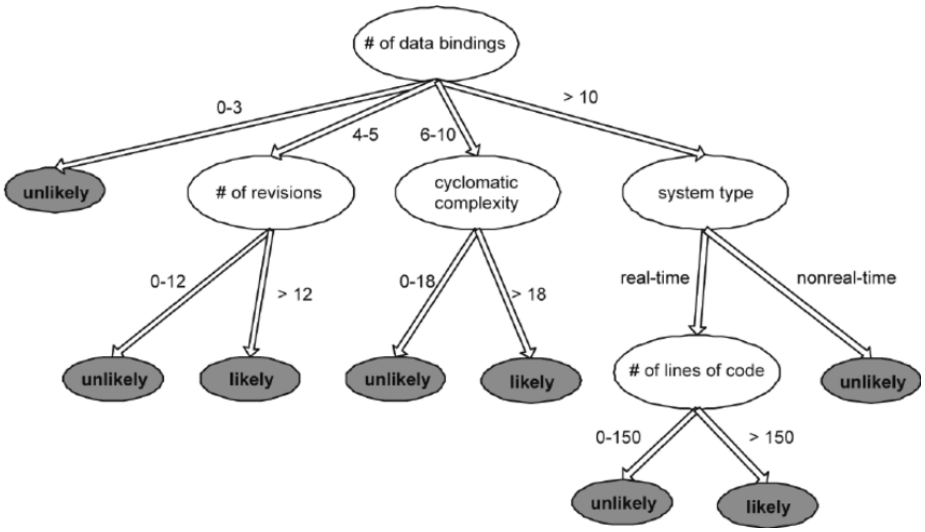
$$A \rightarrow B$$

که A و B به مجموعه‌اي از ویژگی‌ها در داده های اساسي اشاره دارند. بنابراین، آن‌ها برای استخراج اطلاعات براساس انطباق در یک مجموعه داده استفاده می‌شوند. به عنوان مثال، تجزیه و تحلیل کارنامه‌ي خطاهای سیستم که در مولفه‌های نرم‌افزاری کشف شده‌اند، می‌تواند روابط بین رویدادهای استنتاجي را بر اساس ویژگی های مولفه نرم افزاری و دسته بندي‌های خطاها استخراج کند. چنین قاعده‌ای به فرم زیر خواهد بود:

(large/small size, large/small complexity, number of revisions) → (interface error, missing or wrong functionality, algorithms or data structure error etc.)

- خلاصه سازی و تشریح داده‌ها.

تشریح داده‌ها خلاصه‌ای از ویژگی‌های داده براي یک رده خاص داده است. داده‌ها بر اساس نیازهاي مشخص شده توسط کاربران گردآوری می‌شوند. این تکنیک‌ها می‌توانند برای کشف مجموعه‌ای از الگوها از مخازن مهندسی نرم‌افزار که ویژگی‌های خاصي را برآورده مي‌کنند، استفاده شوند.



شکل ۱.۱: درخت رده بندی برای شناسایی مولفه های نرم افزار خطرناک.

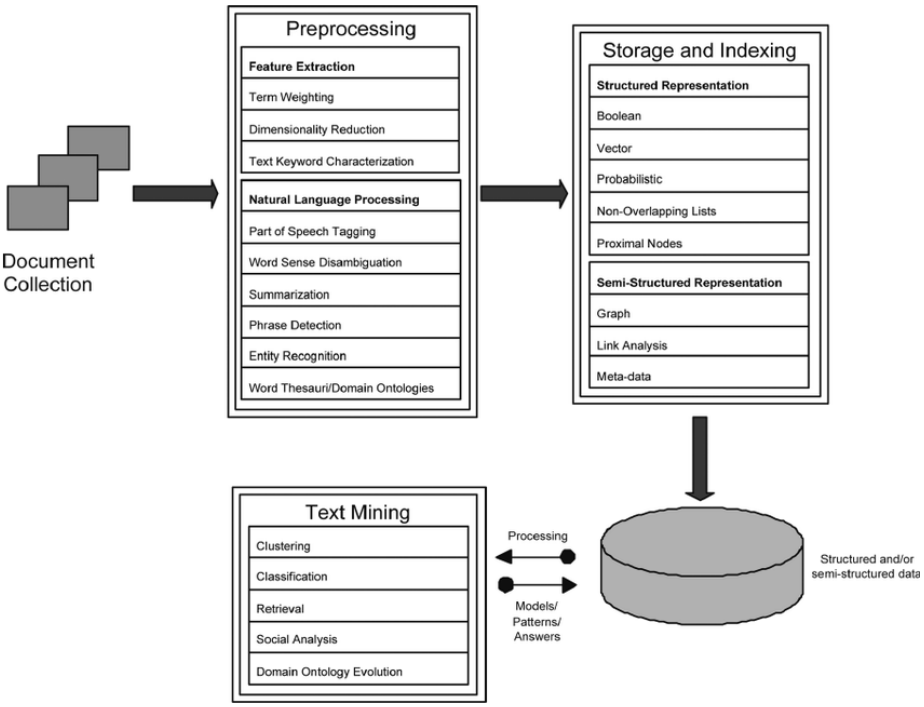
- تشخیص تغییر و انحراف.

تشخیص تغییر و انحراف، بر کشف مهم‌ترین تغییرات در داده‌ها از مقادیري که قبلا اندازه گیری شده است، تمرکز دارد. بنابراین این تکنیک‌ها می‌توانند به شناسایی تغییرات کد منبع یا شناسایی تفاوت‌هاي بین الگوهای استخراج شده از مخازن مهندسی نرم‌افزار کمک کنند.

رویکردهای مختلفي برای انجام وظایف داده‌کاوی بالا و به کار بردن انواع مختلف داده ها توسعه داده شده اند. آنها تکنیک ها را از جنبه‌های مختلف مدیریت داده‌ها و تجزیه و تحلیل داده‌ها، از جمله شناخت الگو، یادگیری ماشین، آمار، بازیابی اطلاعات، تجزیه و تحلیل متن و مفهوم به کار مي‌برند.

متن‌کاوي (کاوش متن) به عنوان یک مورد خاص از داده‌کاوی معرفی می‌شود و به فرایند استخراج اطلاعات از متن اشاره دارد. مخازن مهندسی نرم‌افزار، شامل اطلاعات متنی مانند کد منبع، لیست‌های پستی، گزارش‌های اشکال و کارنامه‌ي اجرا است.

کاوش منابع متنی برای بسیاری از فعالیت‌های مهم در مهندسی نرم‌افزار ضروري است: ردیابی نیازها؛ بازیابی اجزای مخزن؛ شناسایی و پیش‌بینی ناکارایی‌هاي نرم‌افزار؛ نگهداری نرم‌افزار ؛ آزمایش و غیره. روش‌های به کار گرفته شده در متن‌کاوي، بسته به کاربرد، معمولا نیاز به تبدیل متون به یک



شکل ۲.۱: پیش پردازش، ذخیره سازی و پردازش متن در متن کاوی.

نمایش ساختاری حد واسط دارند، که می‌تواند به عنوان مثال ذخیره‌سازی متون در یک سیستم مدیریت پایگاه داده، بر اساس یک شماي خاص باشد. اگرچه در بسیاری از رویکردها، بهتر است یک فرم حد واسط نیمه ساخت یافته‌ي متن نیز نگهداري شود، براي مثال مثل نمایش مستندات در قالب یک گراف، که در آن تجزیه و تحلیل جامعه و تکنیک‌های گراف می‌توانند مورد استفاده قرار گیرند. مستقل از اهداف کاری، متن‌کاوي نیاز به تکنیک‌های پیش پردازشی دارد که معمولا تحلیل کیفی و کمی ویژگی‌های مستندات را به همراه می‌آورد. در شکل ۲.۱، نمودار مهمترین مراحل تحلیل پیش پردازش و همچنین مهمترین تکنیک‌های متن‌کاوي نشان داده شده است.

فصل ۲

اطلاعات مهندسی نرم‌افزار

ماهیت داده‌هایی که توسط تکنیک‌های داده‌کاوی در مهندسی نرم‌افزار مورد استفاده قرار می‌گیرند، می‌تواند به عنوان روشی متمایز از روش‌های پایه‌اي عمل کند، زیرا بر پیش‌پردازش و همچنین پس‌تحلیل تاثیر می‌گذارد. در ادامه، ما منابع مختلفی از اطلاعات مهندسی نرم‌افزار را که داده‌کاوی براي آن‌ها به کار رفته است ارائه می‌دهیم. این ارائه همچنین تلاش می‌کند تا سختي تهیه‌ي داده‌ها را برای پردازش منعکس کند.

۱.۲ مستندات

داده‌های مستندات نرم‌افزاری از اهمیت زیادی برخوردار هستند اما در کنار آن از پیچیدگی بالایی نیز برای پردازش توسط تکنیک‌های داده‌کاوی برخوردارند. برنامه کاربردی، مدیریت سیستم و مستندات کد منبع، یک میانگیر بزرگ مستندات و متن رایگان برای تجزیه و تحلیل و کاوش نرم‌افزار تشکیل می‌دهند. از میان قسمت‌هاي اطلاعاتی متن، که می‌توان آن را با ارزش‌ترین مقدار برای استفاده در تکنیک‌های کاوش در نظر گرفت،مي‌توان به توضیحات نرم‌افزار، پیکربندی استفاده و راه اندازی، راهنمای کاربر، مسائل مربوط به مدیریت فایل، ورود به سیستم، مجوز و مسائل مربوط به سازگاری اشاره کرد. علاوه بر مستندات خارجي نرم‌افزار، مستندات داخلی نیز ممکن است نقش منبع داده مهم را بازی کنند. اگرچه اکثر مستندات در چند دسته‌ي مختلف از اسناد،

مانند قالب سند قابل حمل، فایل های سیستم حروف چيني و متن، اچ‌تی‌ام‌ال قرار مي‌گيرند. با توجه به تنوع انواع اسناد و داده‌های متنی مورد استفاده، لازم است که یک مولفه پیش‌پردازش برای اطلاعات اسناد، که قادر به استفاده از تجزیه‌کننده‌های تمام انواع اسناد ذکر شده دربالا است، وجود داشته باشد.
یکی دیگر از منابع مهم اطلاعاتی که در مستندات نرم‌افزار قرار دارد، داده‌های چندرسانه‌ای است. دستورالعمل‌های شکل‌ها، و همچنین دستورالعمل‌های صوتی و تصویری، همگي می‌توانند به عنوان مقدار اطلاعات اضافه شده در نظر گرفته شوند. در چنین مواردی، تکنیک‌های کاوش چندرسانه‌ای باید برای به عهده گرفتن پیش‌پردازش و پس‌پردازش، و افزایش هزینه‌های کلي پردازش صورت گیرد.

۲.۲ اطلاعات مدیریت پیکربندی نرم‌افزار

داده‌های به وجود آمده از سیستم‌های مدیریت پیکربندی نرم‌افزار[۱] ممکن است شامل کد نرم‌افزاری، مستندات، مدل‌های طراحی، حسابداری وضعیت، ردیابی نقص و همچنین داده‌های کنترل تجدید نظر (مستندات و نظرات همراه با نسخه های نرم‌افزاری در CVS تصویب شده) باشند. مستقل از سیستم کنترل نسخه پایه (متمرکز یا توزیع‌شده) مقدار داده‌های در دسترس SCM بزرگ است و بنابراین یک مطالعه دقیق و درک تمیز از دامنه نرم‌افزار مورد نیاز است، در نتیجه ارزشمندترین داده‌ها نگهداری می‌شوند. اکثر داده‌هاي SCM ها متن ساختار یافته است.
سیستم‌های مدیریت کد منبع غیر متمرکز[۲] در سال‌های گذشته رشد چشمگیری داشته است. $DSCM$ها می‌توانند به محققان مهندسی نرم‌افزار، داده‌های جدید و مفید ارائه دهند که آن‌ها را قادر می‌سازند تا فرآیندهای نرم‌افزاری را بهتر درک کنند. با این حال داده‌های $DSCM$ باید با دقت کاوش شوند. از آنجا که تفاوت هایي در معنای اصطلاحات استفاده شده در سیستم‌های مدیریت کد منبع متمرکز و غیرمتمرکز وجود دارد، تله‌هاي بالقوه‌ای ممکن است در تجزیه و تحلیل این داده‌ها وجود داشته باشند.

[۱] Software configuration management systems (SCMs)
[۲] Decentralized source code management (DSCM)

۳.۲ کد منبع

کد منبع براي داده‌کاوی در مهندسی نرم‌افزار می‌تواند منبع مهمی از داده‌ها باشد. کاربردهای متنوع داده‌کاوی در مهندسی نرم‌افزار، از کد منبع برای کمک به نگهداری نرم‌افزار، درک برنامه و تجزیه و تحلیل اجزای نرم‌افزار استفاده کرده‌اند. جزئیات این رویکردها در فصل ۳ مورد بحث قرار می‌گیرد. پیش‌پردازش اولیه براي کد منبع در دست، همیشه یک هشدار است، از آنجا که یک تجزیه‌کننده برای زبان کد منبع مربوطه باید در دسترس باشد.

پس از تجزیه، کد منبع را می‌توان به عنوان متن ساختار یافته مشاهده کرد. جنبه‌های مرکزی استفاده از تکنیک‌های داده‌کاوی براي کد منبع شامل پیش‌بینی تغییرات آینده از طریق کاوش تاریخچه‌ي تغییرات، پیش‌بینی انتشار تغییرات، نقص‌های حاصل از تاریخچه نهان و همچنین پیش‌بینی تراکم نقص در فایل‌های کد منبع است.

۴.۲ کد کامپایل شده و ردیابی اجرا

کد کامپایل‌شده در قالب کد هدف، یکی از منابع داده جایگزین برای استفاده از تجزیه و تحلیل پایا در مهندسی نرم‌افزار است. همچنین کد کامپایل‌شده به عنوان یک منبع داده براي تکنیک‌های داده‌کاوی به منظور کمک به تشخیص نرم‌افزارهای مخرب مورد استفاده قرار گرفته است. علاوه بر این، اصول کاوش وب به طورگسترده در برنامه‌هاي قابل اجراي شي‌گرا به منظور کمک به درک برنامه برای روش‌هاي مهندسی معکوس استفاده شده است. هنگامی که مولفه‌ها و اجزای نرم‌افزاری مورد آزمایش قرار می‌گیرند، زنجیره ای از رویدادها رخ می‌دهد که در ردیابی اجرا[۳] ثبت می‌شوند. استخراج الگوي اجرا نیز در ردیابی اجرا تحت چارچوب تجزیه و تحلیل پویا به منظور کمک به استخراج عملکردهای سیستم‌های نرم‌افزاری استفاده شده است.

[۳] execution traces

۵.۲ پایگاه داده‌های اشکال و ردیابی موضوع

پایگاه داده‌هاي گزارش اشکال[۴] یا ردیابي موضوع[۵]، مهمترین گزارش موضوع در سیستم‌های نرم‌افزاری هستند. داده‌های ساخت یافته (چندتایی‌های پایگاه داده) شامل توضیحات یک موضوع، جزئیات گزارش‌دهنده و تاریخ / زمان هستند که سه نوع اطلاعات استانداردي‌اند که می‌توانند در پایگاه داده‌های ردیابی موضوع یافت شوند. تکنیک‌های یادگیری ماشین در گذشته برای پیش‌بینی تخصیص صحیح برنامه‌نویسان به اشکال‌ها، پاک کردن پایگاه داده از ظهور یک خطاي مشابه، یا حتی پیش‌بینی مولفه‌های نرم‌افزاری که در همان زمان تحت اثر اشکال‌هاي گزارش شده بوده‌اند، به طور موفقیت‌آمیزي استفاده شده است.

۶.۲ لیست هاي پستی

سیستم‌های نرم‌افزاری بزرگ، و به ویژه نرم‌افزارهای متن باز، لیست‌های پستي را به عنوان پل ارتباطي بین کاربران و برنامه‌نویسان ارائه می‌دهند. لیست‌هاي پستی، داده‌ي سخت به شمار مي‌آیند، زیرا حاوی حجم زیادي از متن‌های آزاد هستند. گراف‌های پیام و نویسنده را به راحتی مي‌توان از داده‌ها بیرون کشید، اما تجزیه و تحلیل محتواي آن‌ها سخت است زیرا از آن جایي که احتمالا پیام‌ها از پاسخ‌هایي تشکیل شده‌اند، نیاز به بررسی بحث‌های اولیه و یا قبلی در لیست‌های پستی مطرح مي‌شود. کاربردهای داده‌کاوی در لیست-های پستی شامل تجزیه و تحلیل متن، خوشه‌بندي متن موضوعات مورد بحث، و تجزیه و تحلیل زبان‌شناسی پیام‌ها برای برجسته کردن پروفایل‌ها و ویژگي‌های شخصیتي برنامه‌نویسان می‌شود، اما محدود به این‌ها نیست.

---

[۴] Bug
[۵] Issue-tracking

فصل ۳

داده کاوی برای مهندسی نرم افزار

با توجه به توانایی داده‌کاوی برای مقابله با حجم زیاد داده‌ها و کارایی آن برای شناسایی الگوهای پنهان دانش، داده‌کاوی در تعدادی از کارهای تحقیقاتی به عنوان راهی برای پشتیبانی از نگهداري نرم‌افزار مقیاس صنعتي، اشکال‌زدایی، آزمایش ارائه شده است. نتایج کاوش می‌توانند به مهندسان نرم‌افزار براي پیشگیری از خرابی نرم‌افزار، استخراج و رده‌بندی اشکالات رایج، شناسایی روابط بین رده‌ها در یک کتابخانه، تجزیه و تحلیل داده‌های نقص، کشف الگوهای دوباره استفاده شده در کد منبع و به همین ترتیب خودکارسازی روند تولید کمک کنند. به طور کلی با استفاده از داده‌کاوی، متخصصین و محققان می‌توانند اطلاعات بالقوه‌ي مهندسی نرم‌افزار را بیاموزند و از نتایج کاوش براي مدیریت بهتر پروژه‌های خود و تولید سیستم‌های نرم‌افزاری با کیفیت بالاتر که در زمان و بودجه‌ي تعیین شده تحویل داده مي‌شوند، استفاده کنند. در بخش‌های بعدی، در مورد ویژگی‌های اصلی رویکردهاي کاوشی که در مهندسی نرم‌افزار مورد استفاده قرار گرفته‌اند و نحوه‌ي استفاده از آن‌ها در چرخه حیات مهندسی نرم‌افزار، بحث مي‌کنیم. ما رویکردها را با توجه به وظایف مهندسی نرم‌افزاري که آن‌ها کمک می‌کنند و تکنیک‌های کاوشي که آن‌ها استفاده می‌کنند، رده‌بندی می‌کنیم.

جدول ۱.۳: رویکرد های کاوش استفاده شده در ردیابی و استخراج نیازها

نتایج تحلیل داده ها	داده ورودی	رویکرد کاوش
نیازها	مستندات	رده بندی
نیازها	SCM و لیست‌های پستی	بازیابی اطلاعات
	کارنامه CVS	خلاصه سازی اطلاعات

۱.۳ ردیابي و استخراج نیازها

در این بخش ما دربارهي چگونگي به کار بردن تکنیک‌های تجزیه و تحلیل داده‌ها براي استخراج یا ردیابی سیستم نیازها بحث می‌کنیم. تحقیقات انجام شده در زمینهي تجزیه و تحلیل نیازها به داده‌کاوی با بالاترین سطح آن اشاره مي‌کنند، از جمله فعالیت‌ها و متدلوژی‌های خاص و مرتبط از آمار، یادگیری ماشین و بازیابی اطلاعات. جدول ۱.۳ ویژگی‌های اصلی تکنیک‌های مورد بحث در زیر را خلاصه می‌کند.

۱.۱.۳ رده بندی

یک رویکرد جدید، بر بهبود بهره‌برداری از نیازهای سطح بالا و سطح پایین با استفاده از بازیابی اطلاعات متمرکز شده است. به طور خاص، آن‌ها جهان مستندات را به عنوان اجتماع عناصر طراحی و نیازهاي فردی در نظر می‌گیرند و آن‌ها مساله ردیابی نیازها را به پیدا کردن شباهت‌هاي بین نمایش فضای برداري نیازهاي سطح بالا و سطح پایین نگاشت مي‌کنند، به این ترتیب آن را به یک وظیفهي بازیابي اطلاعات کاهش مي‌دهند. به عنوان گسترش این مطالعه، محققان بر کشف عوامل موثر بر رفتار یک تحلیلگر در هنگام کار با نتایج بدست آمده از ابزارهای داده‌کاوی در مهندسی نرم‌افزار تمرکز کرده‌اند. تمام تحقیق براساس فرضیه‌ی تایید شدهي دقت ردیابي‌هاي کاندیدای تولید شده توسط رایانه بر دقت ردیابی‌هاي تولید شده توسط تحلیلگر تاثیر می‌گذارد، است. این مطالعه نشان می‌دهد که چگونه عملکرد ابزارهایی که از طریق استفاده از بازیابی‌اطلاعات، نیازهاي سطح بالا و سطح پایین را استخراج می‌کنند، بر زمان مصرف شده توسط یک تحلیلگر برای ارائهي بازخورد و همچنین بر عملکرد آن تأثیر می‌گذارد. نتایج

نشان می‌دهند که سیستم-های داده‌کاوی فراخوان[1] کمي در بازخورد زمان مصرف شده‌ي تحلیلگر دارند. به طور موازي، فراخوان بالا نیز منجر به تعداد زیادی مثبت‌کاذب[2] می‌شود که باعث می‌شود تحلیلگر تعداد زیادی از نیازها را کاهش دهد و فراخوان را مبهم کند. به طور کلی نتایج گزارش شده نشان می‌دهند که تحلیلگر تمایل دارد که صحت[3] و فراخوان را با هم در یک سطح نگه دارد.

۲.۱.۳ خلاصه سازي اطلاعات

از دیدگاه دیگر، متن‌کاوي در مهندسی نرم‌افزار برای اعتبارسنجی داده‌هاي لیست‌های پستی،کارنامه‌ي CVS و فایل‌های تغییر ورودی نرم‌افزار متن‌باز استفاده شده است. محققان مجموعه‌ای از ابزارها، یعنی $SoftChange$[4]، را ایجاد کردند که اعتبارسنجی داده‌هاي منابع متني نرم‌افزار متن‌باز ذکر شده در بالا را پیاده‌سازی می‌کنند. ابزارهای آن‌ها این انواع داده‌ها را از پروژه‌های متن‌باز، بازبینی، خلاصه‌سازی و اعتبارسنجي می‌کنند. بخشی از تجزیه و تحلیل آن‌ها می‌تواند فعال‌ترین برنامه‌نویسان یک پروژه متن‌باز را مشخص کند. آمار و دانش جمع‌آوری شده توسط تجزیه و تحلیل $SoftChange$ به طور کامل مورد بهره‌برداری قرار نگرفته است، زیرا روش‌های پیش‌بینی دیگري می‌توانند با توجه به قطعات کدي که ممکن است در آینده تغییر کنند، یا تجزیه و تحلیل انجمني بین اهمیت تغییرات و افراد، مورد استفاده قرار گیرند (یعنی آیا همه‌ي تغییرات ثبت شده توسط فعال‌ترین برنامه‌نویس، در مقیاس و در عمل به اندازه‌ي بقیه مهم بوده است؟).

۲.۳ تجزیه و تحلیل تولید

این بخش، یک مرور کلی از رویکردهای کاوش مورد استفاده برای کمک به روند تولید را فراهم می‌کند. ویژگی‌های اصلی این رویکردها در جدول۲.۳ خلاصه شده‌است.

---

[1] Recall
[2] False negative
[3] Precision
[4] http://sourcechange.sourceforge.net/

جدول ۲.۳: رویکرد های کاوش مورد استفاده در توسعه نرم‌افزار

نتایج تحلیل داده ها	داده ورودی	رویکرد کاوش
فرایندهای نرم افزار	کد منبع	خوشه بندی
ردیابی اشکال ها	SCM ، کد منبع	رده بندي، بازیابی متن
تصحیح نقص	نقص	کاوش الگوهای تکرارشونده
قوانین تلاش	داده	قوانین انجمنی
شرط‌های نادیده‌گرفته‌شده	برنامه گراف وابستگی	قوانین انجمنی

۱.۲.۳ خوشه بندی

متن‌کاوي نیز در مهندسی نرم‌افزار برای کشف فرایندهای تولید استفاده شده است. فرایندهای نرم‌افزاری شامل رویدادهایي مانند روابط عوامل، ابزارها، منابع و فعالیت‌های سازماندهی شده توسط ساختارهای کنترل جریان است که بیان مي‌دارد، این مجموعه از رویدادها به صورت سری، موازی، تکراري اجرا می‌شوند یا یکی از مجموعه‌ها به طور انتخابی اجرا می‌شود. کشف فرآیند نرم‌افزار، منابع تولید را (به عنوان مثال کد منبع، رونوشت‌هاي ارتباطات و غیره) به عنوان ورودی مي‌گیرد و هدف آن کشف دنباله‌ای از رویدادهاي مشخص کننده وظایفی است که منجر به تولید آن‌ها شده است. محققان یک روش نوآورانه برای کشف فرایندهای نرم‌افزاری مخازن وب نرم‌افزارهای متن‌باز ارائه داده‌اند. روش آن‌ها شامل تکنیک‌های استخراج متن، تحلیل موجودیت وتجزیه و تحلیل شبکه‌های اجتماعی است و بر اساس فرایند رده‌بندی‌های موجودیت است. روش‌های خودکار اصلاح رده‌بندی با استفاده از وظایف متن‌کاوي می‌تواند مورد استفاده قرار گیرد، به طوری که این روش مستلزم وابستگی شدید به اقدامات، ابزار، منابع و عاملان رده‌بندی است. یک مثال می‌تواند استفاده از خوشه‌بندی متن روي منابع متني نرم‌افزار باز و استخراج فقره‌هاي کاندیداي جدید برای رده‌بندی ناشی از برچسب‌های خوشه‌ای باشد.

آن‌ها به عنوان متن ورودی، لیست‌هاي پستی توسعه دهنده $Apache$ را استفاده کرده‌اند. تحلیل موجودیت ضروری است، زیرا بسیاری از افراد بیش از یک نام مستعار استفاده می‌کنند. پس ازساختن گراف اجتماعی که از

ارتباطات بین فرستنده و پاسخ‌دهنده ساخته شده است، آن‌ها یک تحلیل شبکه اجتماعی انجام دادند و به یافته‌های واقعا مهمي مانند رابطه‌ي قوی بین فعالیت ایمیل و فعالیت سطح کد منبع رسیدند. علاوه بر این، تجزیه و تحلیل شبکه‌های اجتماعی در این سطح، گره‌های مهم (افراد) را در بحث‌ها نشان می‌دهد. اگرچه تجزیه و تحلیل گراف و ارتباط در این روش دخیل بود،اما استفاده از تکنیک‌های رتبه‌بندي گره‌ها، مانند $PageRank$ یا سایر تکنیک‌های پردازش گراف مانند $Spreading\ Activation$ انجام نشدند.

۲.۲.۳ رده بندی

مخازن کد منبع، اطلاعاتی غني را ذخیره می‌کنند که نه تنها برای مدیریت و ساخت کد منبع مفید است، بلکه یک گزارش دقیق نیز به دست می‌دهد که چگونه کد منبع در طول توسعه پیشرفت کرده است. اطلاعات مربوط به شواهدی از $refactoring$ کد منبع در مخزن ذخیره می‌شود. همچنین همان‌طور که اشکال‌ها رفع مي‌شوند، تغییرات انجام شده برای اصلاح مشکل ثبت مي‌شوند. همان‌طور که API های جدید به کد منبع اضافه می‌شوند، یک راه مناسب استفاده از آن‌ها به صورت ضمنی در کد منبع، توضیح داده می‌شود. سپس، یکی از چالش‌ها، توسعه‌ي ابزارها و تکنیک‌هایی است که استخراج و استفاده از این اطلاعات مفید را به طور خودکار انجام می‌دهند.

در [۵۶] یک روش پیشنهاد شده است که از داده‌هایی استفاده مي‌کند که رفع اشکال‌هاي کاوش شده از مخزن کد منبع برای بهبود تکنیک‌های تجزیه و تحلیل پایا براي پیدا کردن اشکال‌ها را توصیف می‌کند. این یک رویکرد دو مرحله‌ای است که از تاریخچه تغییر کد منبع یک پروژه نرم‌افزاری براي بهبود جستجوی اشکال‌ها استفاده می‌کند.

اولین گام در این فرآیند، شناسایی انواع اشکال‌هایي است که در نرم‌افزار رفع شده‌اند. هدف بررسي کردن داده‌هاي تاریخی ذخیره شده برای پروژه نرم‌افزاري است، تا بتوانیم بفهمیم که چه داده‌هایي وجود دارند و چگونه ممکن است در وظیفه‌ي یافتن اشکال‌ها مفید باشند. بسیاری از اشکال‌هاي موجود در تاریخچه‌ي CVS، نامزدهاي خوبی برای شناسایی توسط تجزیه و تحلیل آماری، بررسي اشاره‌گر $NULL$ و بررسی مقادیر بازگشتی تابع هستند.

گام دوم ساختن یک یابنده‌ي اشکال به وسیله‌ي این یافته‌ها است. ایده

این است که یک بررسی‌کننده‌ي مقدار بازگشتی یک تابع، بر اساس این دانش که یک نوع خاص از اشکال چند بار در گذشته رفع شده است، توسعه یابد. به طور خلاصه، این بررسی‌کننده به دنبال مواردی می‌گردد که در آن‌ها مقدار بازگشتی یک تابع قبل ازاینکه مورد آزمایش قرار بگیرد، در کد منبع استفاده شود. استفاده از یک مقدار بازگشتی می‌تواند به معنای پاس دادن آن به عنوان یک شناسه تابع به یک تابع، استفاده از آن به عنوان بخشی از یک محاسبه، ارزیابی مقدار آن اگر یک اشاره‌گر باشد یا مقداردهي دوباره‌ي آن قبل از آزمایش باشد. همچنین مواردی که مقادیر بازگشتي هیچ وقت توسط تابع فراخوانی ذخیره نمی‌شوند، بررسی می‌شوند. آزمایش کردن یک مقدار بازگشتی به معنی آن است که یک تصمیم کنترل جریان به مقدار بازگشتي اطمینان دارد.

بررسی کننده یک تحلیل جریان داده روي متغیر نگهداری مقدار بازگشتی انجام مي‌دهد، فقط به منظور تعیین این که آیا این مقدار قبل از آزمایش مورد استفاده قرار گرفته است یا خیر. آن به سادگی، متغیر اصلی را که مقدار بازگشتی در آن ذخیره می‌شود، شناسایي مي‌کند و استفاده بعدی از آن متغیر را تعیین می‌کند.

علاوه بر این، این بررسی‌کننده هشدارهایی را که در یکی از دسته‌های زیر می‌یابد، دسته‌بندی می‌کند:

- هشدارها برای مقادیر بازگشتی‌اي که به طور کامل نادیده گرفته می‌شوند، یا حالتي که مقدار بازگشتی ذخیره شده باشد اما هرگز استفاده نشده باشد.
- همچنین هشدارها برای مقادیر بازگشتی‌اي که در یک محاسبه استفاده می‌شوند قبل از این که در یک عبارت کنترل جریان مورد آزمایش قرار بگیرند.

هر مقدار بازگشتی‌اي که قبل از این که مورد آزمایش قرار گیرد، به عنوان یک شناسه تابع به یک تابع پاس داده مي‌شود، و همچنین مقدار بازگشتي هر اشاره‌گري که بدون آزمایش مورد ارجاع قرار می‌گیرد، پرچم گذاری می‌شود.

با این حال انواعي از توابع وجود دارند که باعث مي‌شوند روند تجزیه و تحلیل پایا، هشدارهای مثبت کاذب تولید کنند. مشخص کردن این که

مقدار بازگشتي كدام تابع نياز نيست بررسی شود، بدون دانش قبلی، سخت است. تکنیک‌های کاوش مخزن کد منبع می‌توانند باعث بهبود نتایج تجزیه و تحلیل پایا باشند. به طور خاص، داده‌ي بدست آمده از مخزن كد منبع و از نسخه فعلی نرم‌افزار كه ما كاوش مي‌كنيم، برای تعیین الگوی استفاده‌ي واقعی برای هر تابع استفاده می‌شود.

به طور كلی، مشاهده شده است كه اشكال‌هاي فهرست شده در پايگاه داده‌هاي اشكال و آن‌هايي كه توسط بررسی تاريخچه‌ی تغييرات كد منبع يافت شده‌اند، در نوع و سطح انتزاع متفاوت هستند. مخازن نرم‌افزار، همه‌ي اشكال‌هاي رفع شده در هر مرحله از فرآيند توليد را ثبت مي‌كنند و در نتيجه آن‌ها اطلاعات بسيار مفيدی را ارائه می‌دهند. بنابراين، ثابت شده است كه يک سيستم برای تكنيک‌های تشخيص اشكال، زمانی كه داده‌هاي بدست آمده از مخزن كد منبع را به صورت خودكار كاوش مي‌كند، موثرتر است.

۳.۲.۳ قوانين انجمني و كاوش الگوهاي تكرار شونده

يک رويكرد در [۴۴] پيشنهاد شده است كه از تكنيک‌های استخراج قوانين انجمني برای تحليل داده‌های نقص استفاده می‌كند. نقص‌های نرم‌افزاری شامل اشكال‌ها، مشخصات و تغييرات طراحی است. داده‌ي نقص جمع‌آوری شده تحت تجزيه و تحليل، متغيرهای مقياس اسمی هستند مانند شرحي از نقص، اولويت برای رفع نقص و وضعيت آن و همچنين متغير مقياس فاصله و نسبت، با توجه به تلاش و طول مدت تصحيح نقص. يک روش كاوش قوانين انجمني گسترش يافته برای استخراج اطلاعات مفيد و آشكار كردن قوانين مرتبط با تلاش اصلاح نقص، به كار گرفته شده است.

محققان پيشنهاد استفاده از تكنيک‌های كاوش گراف برای كشف قوانين شرطي ضمنی در يک پايه كد و براي كشف قوانين نقضي كه نشان دهنده‌ي شرط ناديده گرفته شده است،را دادند. آن‌ها برنامه‌ها و قوانين برنامه نويسي شرطي را در قالب گراف‌هاي وابستگی نمايش دادند. سپس آن‌ها از تكنيک‌هاي كاوش زيرگراف‌های تكرارشونده و كاوش مجموعه فقره‌های تكرارشونده براي كشف قوانين شرطی شامل پيش‌شرط‌ها و پس‌شرط هاي فراخوانی‌های تابع و همچنين كشف نقض‌هاي آن قوانين استفاده مي‌كنند.

جدول ۳.۳: روش های کاوش استفاده شده در تست نرم افزار

نتایج تحلیل داده ها	داده ورودی	رویکرد کاوش
شبکه آزمایش‌کننده تابع	متغیر های I/O سیستم نرم افزار	رده بندی
خوشه‌های پروفایل‌های اجرا	پروفایل‌های اجرا	خوشه بندی
رده‌بندهای رفتار نرم‌افزار	اجراهای برنامه	خوشه بندی، رده بندی

۳.۳ آزمایش کردن

ارزیابی نرم‌افزار بر اساس آزمایش‌هایی است که توسط آزمایش‌کنندگان نرم‌افزار طراحی شده است. بنابراین ارزیابی خروجی‌های آزمایش با تلاش‌های قابل توجه آزمایش‌کنندگان انسانی مرتبط است، که اغلب دانش ناقصي از مشخصات نیازها دارند.

رویکردهای داده‌کاوی را می‌توان برای استخراج اطلاعات مفید از نرم‌افزار مورد آزمایش، به کار برد که می‌تواند به تست نرم‌افزار کمک کند. به طور خاص، مدل‌های داده‌کاوی ایجاد شده از نرم‌افزار آزمایش شده، می‌تواند برای بازیابی مقادیر گم شده و مشخصات ناقص، طراحی مجموعه‌ای از آزمون‌های رگرسیون و ارزیابی درست بودن خروجی‌های نرم‌افزار در هنگام منتشر کردن نسخه‌های جدید سیستم استفاده شود. یک کتابخانه تست رگرسیون باید شامل یک حداقل تعداد تستی باشد که تمام جنبه‌های ممکن عملکرد سیستم را، پوشش می‌دهد. برای اطمینان از طراحی موثر تست‌هاي رگرسیون جدید، باید نیازهاي واقعی یک سیستم موجود را پوشش دهد.

بنابراین، یک آزمایش‌کننده باید مشخصات سیستم را تجزیه و تحلیل کند، تحلیل ساختاری کد منبع سیستم را انجام دهد و نتایج اجرای سیستم را برای تعریف روابط ورودی خروجی در نرم‌افزار مورد آزمایش، مشاهده کند. جدول ۳.۳ تکنیک‌های اصلی داده‌کاوی را که در زمینه تست نرم‌افزار مورد استفاده قرار می‌گیرند خلاصه می‌کند.

۱.۳.۳ خوشه بندی

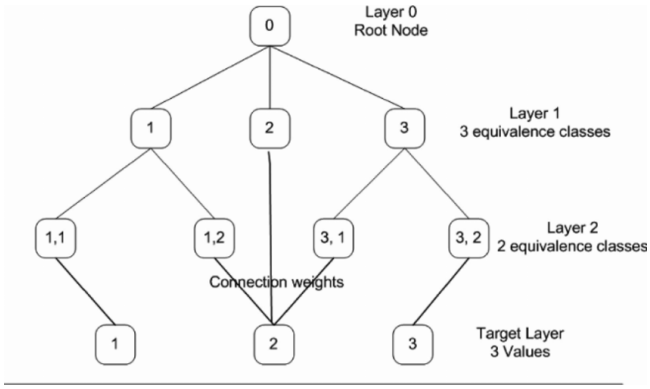
یک رویکرد دیگر، که روش‌های تحلیل خوشه را برای انتخاب مجموعه‌ای از اجراها که برای تبعیت از نیازها مورد ارزیابی قرار می‌گیرند، استخراج مي‌کند. رویکرد پیشنهادي از یک مجموعه‌ي پروفایل‌هاي اجرايي[۵] استفاده مي‌کند که اینگونه تعریف مي‌شوند، اجراي نسخه‌ي تحت آزمایش نرم‌افزار روي یک مجموعه‌ي داده شده از ورودي‌هاي برنامه. الگوریتم خوشه‌بندی برای فیلتر کردن پروفایل‌ها بر اساس ویژگی‌های مشابهشان استفاده می‌شود. سپس پروفایل‌های اجرایی از نتایج خوشه‌ها انتخاب می‌شوند.

رویکردی که هدف آن تجزیه و تحلیل مجموعه‌ای از اجراهاي برنامه‌ها و تعریف رده‌بندهاي رفتار نرم‌افزار است، در [۸] پیشنهاد شده است. با توجه به این مطالعه، مدل مارکف برای کدگذاری پروفایل‌های پروژه‌ها مورد استفاده قرار می‌گیرد. سپس مدل‌های مارکف اجراهای برنامه‌های فردی، با استفاده از الگوریتم خوشه‌بندی زنجیره‌ای، خوشه‌بندی شده است. روند خوشه‌بندی قصد دارد اجراهای مشابه برنامه را جمع کند و سپس رده‌بندهای موثر رفتار برنامه را تعریف کند. همچنین خودراه‌اندازی[۶] به عنوان یک تکنیک یادگیری فعال استفاده می‌شود، در نتیجه رده‌بندهاي یادگیری به طور مداوم تحت آموزش قرار مي‌گیرند. به طور خاص، این روش با شناسایی نمونه‌های آموزشی جدید به رده‌بندها کمک مي‌کند و سپس رده‌بندها را می‌توان با استفاده از مجموعه‌ي گسترده‌ای از نمونه‌هاي آموزشي، بازآموزی کرد.

۲.۳.۳ رده بندی

یک رویکرد که هدف آن خودکارسازي تجزیه و تحلیل ورودی خروجی داده‌های اجرا براساس یک متدولوژي داده‌کاوی است، که در [۳۵] پیشنهاد شده است. این متدولوژي مبتنی بر شبکه اطلاعات فازی[۷] است که دارای ساختار درخت‌مانند بی نظیری است. اجزای شبکه شامل گره‌ي ریشه، یک تعداد قابل تغییري از لایه‌های پنهان (یک لایه برای هر ورودی انتخاب شده) و

[۵] execution profiles
[۶] Bootstrapping
[۷] info-fuzzy network (IFN)



شکل ۱.۳: یک مثال از ساختار شبکه اطلاعات فازی

لایه هدف (خروجی) که نشان‌دهنده‌ي مقادیر خروجی ممکن است. یک ویژگی ورودي مشابه در تمام گره‌های یک لایه (سطح) مورد استفاده قرار می‌گیرد در حالی که هر گره‌ي هدف با یک مقدار (رده) در دامنه‌ي ویژگی هدف مرتبط است. اگر مدل شبکه اطلاعات فازی به منظور پیش‌بینی مقادیر یک ویژگی هدف پیوسته استفاده شده باشد، گره‌های هدف، بازه‌هاي از هم جدا (بدون اشتراک) در محدوده‌ي ویژگي را نشان می‌دهند.

یک لایه پنهان l شامل گره‌هایی است که نشان‌دهنده‌ي ترکیبی از مقادیر l ویژگي ورودی اول هستند که مشابه تعریف یک گره‌ي داخلی در یک درخت تصمیم استاندارد است. گره‌های نهایی (پایانه) شبکه نشان‌دهنده‌ي ارتباطات غیرافزوده از مقادیر ورودی هستند که خروجی‌های متمایزي تولید می‌کنند. با توجه به اینکه شبکه از داده‌های اجرایی یک سیستم نرم‌افزاری استنباط می‌شود، هر اتصال دروني بین گره‌ی پایانه و هدف، یک خروجی احتمالی یک نمونه آزمایشی را نشان می‌دهد. شکل ۱.۳ ساختار شبکه اطلاعات فازی را نشان می‌دهد.

یک شبکه اطلاعات فازی جداگانه برای نشان دادن هر متغیر خروجی ساخته شده است.

مولفه‌های اصلی محیط مبتنی بر شبکه اطلاعات فازی ارائه شده در [۳۵] عبارتند از:

- سیستم میراث[۸]. این مولفه یک برنامه، یک جزء یا یک سیستم مورد

[۸] Legacy (LS)system

آزمایش را در نسخه‌های بعدی نرم‌افزار نشان می‌دهد.

- مشخصات ورودی‌ها و خروجی‌های برنامه[9].داده‌های پایه برای هر متغیر ورودی و خروجی در سیستم میراث.
- مولد تست تصادفی[10] . این مولفه، ترکیباتی تصادفی از مقادیر، در محدوده هر متغیر ورودی تولید می‌کند.
- بستر تست[11]. این مولفه موارد آموزشی ایجاد شده توسط مولفه مولد تست تصادفی را به سیستم میراث می‌فرستد.

الگوریتم شبکه اطلاعات فازی بر روی ورودی‌های ارائه شده توسط مولد تست تصادفی و خروجی‌های حاصل از یک سیستم میراث با استفاده از مولفه بستر تست مورد آموزش قرار مي‌گیرد. یک مولفه شبکه اطلاعات فازی جداگانه برای هر متغیر خروجی ساخته شده است. الگوریتم شبکه اطلاعات فازی، نمونه‌هاي آموزشی را که به طور تصادفی از طریق مولفه مولد تست تصادفی تولید می‌شوند، به عنوان ورودی در نظر مي‌گیرد و خروجی‌ها توسط سیستم میراث برای هر نمونه آزمایشي تولید مي‌شوند. الگوریتمشبکه اطلاعات فازی بطور مکرر اجرا می شود تا زیرمجموعه‌ي متغیرهای ورودی مربوط به هر خروجی و مجموعه‌ي متناظر نمونه‌هاي آزمایشي غیرافزوده را پیدا کند. نمونه‌هاي آزمایشي واقعی با استفاده از یک سیاست آزمایشي موجود، از رده‌های هم ارز به طور خودکار تشخیص داده شده، تولید می‌شوند.

۴.۳ اشکال زدایی

برآورد اشکال در نرم افزار فعالیتی بسیار اساسی برای برنامه ریزی موفق و صحیح پروژه است. تمام داده‌های مربوط به اشکال نرم‌افزار در مخزن اشکال نگهداری می‌شوند. مخزن اشکال نرم‌افزار در بر گیرنده اطلاعات جالبی در زمینه تولید یک پروژه است و به کار گیری داده‌کاوی در این مخزن‌ها می‌تواند الگوهای جالبی را نشان دهد. برای مثال به کار گیری یک روش داده‌کاوی

[9] Specification of Application Inputs and Outputs (SAIO)
[10] Random test generator (RTG)
[11] Test Bed (TB)

جدول ۴.۳: روش های کاوش استفاده شده در اشکال زدایی

نتایج تحلیل داده ها	داده ورودی	رویکرد کاوش
تشخیص خطاهای منطقی	تست های ورودی‌خروجی	رده بندی احتمالاتی
اشکال‌های منطقی	اجراهای برنامه	رده بند SVM
درخت تصمیم شکست‌ها	پروفایل‌های اجرایی و نتیجه	رده بندی،درخت تصمیم
الگوهای استفاده فراخوانی	کد منبع	قوانین انجمنی

پیش بینی، می‌تواند برای تشخیص اشکال نرم‌افزار در مخزن اشکال به کار رود. در ابتدای کار خلاصه و توضیحات اشکالی که نیاز به برآورد دارد با خلاصه و توضیحات اشکال‌های موجود در مخزن اشکال مقایسه و برای این کار از مدل تشابه وزنی استفاده می‌شود. گام بعدی محاسبه زمان مورد نیاز برای اصلاح تمام اشکال‌های مشابه و گرفتن میانگین آن جهت برآورد پیش بینی شده یک اشکال است[۶۰].

خطاهاي منطقی برنامه به ندرت موجب تناقض‌هاي دسترسی به حافظه می‌شوند اما خروجی‌های نادرست تولید می‌کنند. تعدادی از تکنیک‌هاي کاوش برای شناسایی خطاهای منطقی و کمک به اشکال‌زدایی نرم‌افزار، مورد استفاده قرار گرفته‌اند (جدول۴.۳ را ببینید).

۱.۴.۳ رده بندی شکست‌هاي نرم افزار

یک استراتژی نیمه خودکار برای رده‌بندی شکست‌های نرم‌افزاری در [۴۶] ارائه شده است. این رویکرد مبتنی بر این ایده است که اگر m شکست در طی یک دوره زمانی که نرم‌افزار اجرا می‌شود، مشاهده شود، احتمال دارد که این خرابی‌ها به دلیل تعداد نقص‌های متمایز کمتري باشد. فرض کنید $F = \{f_1, f_2, ..., f_m\}$ مجموعه‌ای از شکست‌های گزارش شده است و هر شکست تنها به دلیل یک نقص است. سپس F را می توان به $m > k$ زیرمجموعه‌ی $F_1, F_2, ..., F_k$ تقسیم کرد به طوری که تمام شکست‌هاي در F_i توسط یک نقص d_i مشابه ایجاد شده اند که$k \geq i \geq ۱$. این بخش‌بندی، رده‌بندی شکست واقعی نامیده می‌شود. در ادامه، ما فازهای اصلی استراتژی را برای تقریب زدن رده‌بندی شکست واقعی توصیف می‌کنیم:

۱. نرم‌افزار برای جمع آوری و انتقال به توسعه‌ي پروفایل‌های اجرایی یا

اجراهای گرفته شده پیاده‌سازي شده است، و سپس به کار گرفته می‌شود.

۲. پروفایل‌های اجرایی متناظر با شکست گزارش‌شده، با یک نمونه تصادفي از پروفایل‌های اجراهای عملیاتی‌اي که برای آن‌ها هیچ شکستي گزارش نشده است، ترکیب شده‌اند. این مجموعه‌ی پروفایل‌ها برای انتخاب یک زیر مجموعه از تمام ویژگی[۱۲] های پروفایل برای استفاده در گروه‌بندي شکست‌های مرتبط، مورد تجزیه و تحلیل قرار گرفته است. یک ویژگی از یک پروفایل اجرایی، مربوط به یک خاصیت[۱۳] یا عنصر از آن است. به عنوان مثال، یک پروفایل فراخواني تابع شامل یک شمارش اجرا برای هر تابع در یک برنامه است و هر شمارش یک ویژگی از پروفایل است. پس استراتژی انتخاب ویژگی به شرح زیر است:

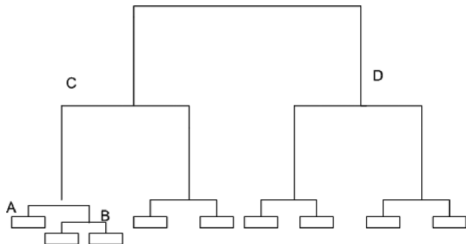
- مجموعه ویژگی‌های کاندید را تولید کنید و از هر یک برای ایجاد وآموزش یک رده‌بند الگو برای تشخیص تمایز بین شکست‌ها از اجراهای موفق، استفاده کنید.
- ویژگی‌های رده‌بندی که بهترین نتایج را به دست می‌دهد،را انتخاب کنید.

۳. پروفایل‌های شکست‌هاي گزارش شده با استفاده از تجزیه و تحلیل خوشه‌ای، تجزیه و تحلیل می‌شوند تا شکست‌هایي که پروفایل‌هایشان مشابه است با توجه به ویژگی‌های انتخاب شده در فاز ۲ ، گروه شوند.

۴. رده‌بندی شکست حاصل در گروه‌ها به منظور تأیید یا اصلاح آن، مورد بررسی قرار می‌گیرد.

استراتژی توصیف شده فوق، رده‌بندی اولیه‌ای از شکست‌های نرم‌افزاری را ارائه می‌دهد. بسته به کاربرد و نیازهاي کاربر، این رده‌های اولیه می‌توانند ادغام شوند یا تقسیم شوند تا شکست‌های نرم‌افزاری، به طور مناسبي شناسایی شوند.
در [۱۹]، دو تکنیک مبتني بر درخت، برای اصلاح رده‌بندی اولیه شکست‌ها پیشنهاد شده است. در زیر ما ایده اصلی این رویکردها را ارائه می‌دهیم.

---
۱۲ feature
۱۳ attribute



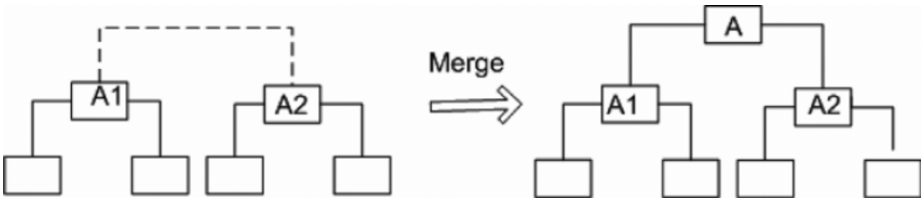
شکل ۲.۳: سلسله مراتب خوشه ها

۲.۴.۳ اصلاح رده‌بندی شکست‌ها با استفاده از دندروگرام‌ها

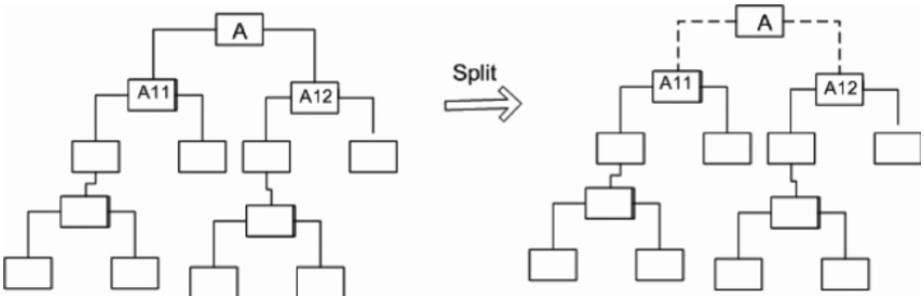
یکی از استراتژی‌هایی که برای اصلاح دسته‌بندی اولیه شکست پیشنهاد شده است، براساس نموداري درخت‌مانند (شناخته شده به عنوان دندروگرام[۱۴]) است. به طور خاص، از آن‌ها استفاده می‌کنند تا تصمیم بگیرند که چگونه خوشه‌های غیر یکپارچه باید به دو یا چند زیر مجموعه تقسیم شوند و تصمیم بگیرند که کدام خوشه‌ها برای ادغام باید در نظر گرفته شوند. یک خوشه در یک دندروگرام به یک زیردرخت نظیر می‌شود که نشان‌دهنده‌ي ارتباط‌هاي بین زیرخوشه‌های آن است. هر چه جد مشترک دو خوشه، در فاصله‌ي دورتري از ریشه‌ي دندروگرام قرار داشته باشد، آن دو خوشه بهم شبیه ترند. به عنوان مثال، بر اساس دندروگرام ارائه شده در شکل ۲.۳ می توان دید که خوشه‌های A و B بیشتر از خوشه‌های C و D مشابه هستند. بزرگترین زیردرخت همگن یک خوشه، بزرگترین زیردرختي است که شامل شکست‌هایی با یک علت مشابه است. اگر خوشه‌بندی بیش از حد درشت باشد، ممکن است برخی از خوشه‌ها دارای دو یا چند زیر درخت همگن بزرگ باشند که شامل شکست‌هایی با علل مختلف هستند. چنین خوشه‌ای باید در سطحي که در آن زیردرخت‌های همگن بزرگ آن، متصل هستند، تقسیم شود، به طوری که این زیردرخت ها به دو خواهر تبدیل می‌شوند همانطور که شکل ۳.۳ نشان می‌دهد. اگر خیلی خوب باشد، ممکن است دو خواهر خوشه‌هایی با شکست‌هایي با علل مشابه باشند. چنین خواهراني (خوشه‌ها) باید در سطح والدینشان ادغام شوند (شکل ۴.۳).

بر اساس این تعاریف، استراتژی پیشنهاد شده برای اصلاح رده‌بندی اولیه

[۱۴] dendrogram



شکل ۳.۳: ادغام دو خوشه. خوشه جدید A حاوی خوشه هایی است که توسط دو زیردرخت همگن $A1$ و $A2$ معرفی شده اند.



شکل ۴.۳: تقسیم یک خوشه: دو خوشه جدید (زیر درخت های با ریشه های $A11$ و $A12$) به زیردرخت های بزرگ همگنی در خوشه قدیمی مربوط می شوند.

شکست‌ها با استفاده از دندروگرام‌ها دارای سه مرحله است:

۱. تعداد خوشه‌هایی که دندروگرام باید به آن تقسیم شود، را انتخاب کنید.

۲. خوشه‌های فردی را برای همگن بودن، با انتخاب دو اجرايي که در خوشه داراي پروفايل‌هايي با بيشترين بي شباهتي هستند، بررسی کنید. اگر اجراهای انتخاب شده دارای علل مشابه یا مرتبط هستند، احتمال دارد که تمام شکست‌های دیگر در خوشه نیزچنین باشند. اگر اجراهای انتخاب شده علتي مشابه يا مرتبط ندارند، خوشه همگن نیست و باید تقسیم شود.

۳. اگر نه خوشه و نه خواهر آن براساس مرحله ۲ تقسیم نشدند و شکست‌های مورد بررسی یک علت داشتند، آنها را ادغام می‌کنیم.

خوشه‌هایی که توسط عملیات ادغام یا تقسیم تولید شده‌اند، باید به همان شیوه تجزیه و تحلیل شوند، که امکان تقسیم یا ادغام بازگشتي را بدهند.

۳.۴.۳ پالایش با استفاده از درختان رده بندی

تکنیک دوم پیشنهاد شده توسط فرانسیس و همکاران، متکی بر ساخت یک درخت رده‌بندی برای تشخیص اجراهای ناموفق است. یک درخت رده‌بندی، نوعی رده‌بند الگویی است که به شکل یک درخت تصمیم‌گیری دودویی است. هر گره داخلی در درخت با یک عبارت رابطه‌ای برچسب‌گذاری می‌شود که یک ویژگی عددی از شی‌اي که رده‌بندي شده است، را با یک مقدار تقسیم ثابت، مقایسه مي‌کند. از سوی دیگر، هر برگ از درخت با مقدار پیش‌بینی شده‌ای برچسب گذاری می‌شود که رده‌ای است که برگ نشان‌دهنده آن است.

اگر درخت رده‌بندی داده شده باشد، ما باید درخت را از ریشه به برگ پیمایش کنیم، تا یک شی را رده‌بندی کنیم. در هر گام از پیمایش قبل از رسیدن به برگ، مقدار عبارت گره فعلی را ارزیابي می‌کنیم. هنگامی که شي به یک برگ می‌رسد، مقدار پیش‌بینی شده‌ي آن برگ به عنوان رده پیش‌بینی شده برای این شی، در نظر گرفته مي‌شود.

در مورد مساله‌ي رده‌بندی شکست نرم‌افزار، ما دو رده موفقیت و شکست

را در نظر می‌گیریم. الگوریتم‌های رده‌بندي و درخت رگرسیون[۱۵] برای ساخت درخت رده‌بندی مربوط به شکست‌های نرم‌افزار، مورد استفاده قرار گرفت. فرض کنید یک مجموعه آموزشي از پروفایل‌های اجرا به شما داده شده است

$$L = \{(x_1, j_1), ..., (x_N, j_N)\}$$

که هر x_i یک پروفایل اجرا را نشان می‌دهد و j_i نتیجه (موفقیت / شکست) مربوط به آن است. مراحل ساخت درخت رده‌بندی بر اساس L عبارتند از:

- انحراف گره $t \in L$ به صورت زیر تعریف می‌شود

$$d(t) = \frac{1}{N_t} \sum (j_i - \overline{J}(t))^2 \qquad (۱.۳)$$

که N_t اندازه t است و $\overline{J}(t)$ مقدار متوسط j در t است.

- هر گره t به دو فرزند t_R و t_L تقسیم می‌شود. تقسیم‌بندی‌ای انتخاب مي‌شود که دارای بیشترین کاهش انحراف باشد. یعنی، از مجموعه‌ی تقسیم‌هاي ممکن S، تقسیم بهینه توسط فرمول زیر به دست مي‌آید:

$$s^* = argmin_s(d(t) - \frac{N_{t_L}}{N_t} d(t_R) - \frac{N_{t_L}}{N_t} d(t_L)) \qquad (۲.۳)$$

- اگر $\alpha \geq d(t)$ برای یک آستانه‌ی α باشد، یک گره برگ اعلام می‌شود.

- مقدار پیش بینی شده برای یک برگ، میانگین مقدار j در بین اجراهاي آن برگ است.

۴.۴.۳ قوانین انجمني و کاوش الگوهای تکرارشونده

دو رویکرد برای کاوش الگوهای استفاده از فراخواني از کد منبع وجود دارد. اولین رویکرد، بر مبنای ایده‌ي کاوش مجموعه فقره‌ها است. این روش، زیر مجموعه‌های فقره‌های تکرارشونده را شناسایي مي‌کند که حداقل، یک کمترین مقدار پشتیبانی تعریف شده توسط کاربر را برآورده می‌کند. نتایج استفاده از این رویکرد براي کد منبع، الگوهایی نامرتب و مرتبط با فراخواني‌های تابع است. از سوی دیگر، رویکرد استخراج الگوی سری،

[۱۵] Classification And Regression Tree (CART)

مجموعه‌ای از الگوهای مرتب با حداقل میزان پشتیبانی برآورده شده، تولید می‌کند. به طور کلی، این رویکردها می‌توانند به الگوهای کاوش استفاده از فراخواني و در نتیجه شناسایی بالقوه اشکال‌ها در یک سیستم نرم‌افزاری کمک کنند.

۵.۳ نگه داری

یک مشکل که در مهندسی نرم‌افزار با آن مواجه می‌شویم نگه‌داری صحیح نرم‌افزار است. به دست آوردن و شناسایی نقص‌های سیستم قبل از آن که منجر به خطا شوند، برای ما مطلوب است. به نظر می‌رسد خیلی از خطاها به گروه‌هایی تقسیم می‌شوند که هر گروه از نقص خاصی در سیستم ناشی شده‌اند.

تحقیقات اخیر، بر روش‌های داده‌کاوی به منظور آسان کردن دسته‌بندی خطاها بر اساس علتشان تمرکز دارد. به طور خاص، این رویکردها نیاز به سه دسته اطلاعات زیر دارند:

۱. مشخصات اجرایی که نشان دهنده علل خطای به وجود آمده هستند.

۲. اطلاعات حسابرسی که وجود خطا را تایید می‌کنند.

۳. اطلاعات تشخیصی که در تعیین علت بروز خطا استفاده می‌شوند.

در ادامه روش‌های داده‌کاوی که نگه‌داری نرم‌افزار را راحت تر می‌کنند، آمده است.

۱.۵.۳ خوشه بندی

به طور خاص، روش‌های خوشه‌بندی به مهندسان کمک می‌کنند تا درک بهتری از ساختار کد منبع و ارزیابی قابلیت نگه‌داری آن داشته باشند. رویکرد ارائه شده به زیر مجموعه‌هایی از کد منبع شامل موارد زیر اعمال شده است:

۱. ماهیت‌هایی که متعلق به دامنه‌های ساختاری و یا رفتاری هستند.

۲. ویژگی‌هایی که ماهیت‌های گفته شده را توصیف می‌کنند(مانند نام رده، تابع و ....)

۳. معیارهای استفاده شده به عنوان ویژگی اضافه که به برنامه‌نویسان نرم‌افزار در درک سیستم تحت نگه‌داری کمک می‌کنند.

بخش دیگر چارچوب فرایند استخراج ویژگی‌های کد منبع است سپس اطلاعات استخراج شده در یک پایگاه‌داده ذخیره می‌شوند تا روش‌های داده‌کاوی روی آن‌ها اعمال شوند. در این رویکرد خاص، روش‌های خوشه‌بندی برای تحلیل اطلاعات ورودی و فراهم کردن درک نسبی از سیستم نرم‌افزاری استفاده می‌شوند. خوشه‌بندی با ایجاد دسته‌های مستقل از هم با توجه به شباهت‌هایشان، درکی کلی از سیستم نرم‌افزاری ارائه می‌دهد.علاوه بر این، خوشه‌بندی می‌تواند به کشف الگوهای برنامه‌نویسی و داده‌های پرت و موارد غیرمعمول کمک کند.

۲.۵.۳ رده بندی

در [۴۹] آن‌ها از داده به دست آمده از یک میلیون پروژه نرم‌افزاری متن باز موجود در پرتال $SourceForge$ به منظور ساختن مدل پیش‌بینی‌کننده برای نگه‌داری نرم‌افزار با استفاده از روش‌های داده‌کاوی و متن‌کاوی استفاده کردند.
با استفاده از$SAS\ Enterprise\ Miner$ و $SAS\ Text\ Miner$، آن‌ها روی جمع‌آوری مقادیر برای متغیرها با توجه به هزینه‌های نگه‌داری و میانگین زمان بازیابی خطا تمرکز کردند. این کار همچنین حذف پروژه‌های توسعه نیافته و یا پروژه‌های بدون گزارش خطا را به دنبال داشت و تنها پروژه‌های قابل استفاده و شامل مقادیر مورد نیاز باقی ماندند.

به علاوه، آن‌ها پروژه‌های باقی‌مانده را بر اساس توصیفشان خوشه‌بندی کردند تا بتوانند رده‌های مهم در پروژه‌های متن باز موجود در پایگاه داده $SourceForge$ را کشف کنند. سپس مقادیر میانگین زمان‌های بازیابی خطا با استفاده از توزیع مقادیرشان تبدیل به شکل دودویی (بالا یا پایین) شدند. و در نهایت از$SAS\ Enterprise\ Miner$برای رده‌بندی میانگین زمان‌های بازیابی خطا استفاده کردند.

نتایج گزارش شده، همبستگی جالبی را بین ویژگی‌هایی مانند تعداد دانلود، قدمت پروژه و استفاده از ایمیل با متغیر رده نشان می‌داد. به عنوان مثال پروژه‌های با قدمت بالا، میانگین زمان‌های بازیابی خطای بالاتری نسبت به پروژه‌های جدید داشتند.

رویکردی برای بهره‌برداری از ایده فیلتر کردن اسپم‌ها[۱۶] برای تشخیص نرم‌افزارهای آسیب‌پذیر وچو دارد. این روش بر اساس این حقیقت است که واحدهای نرم‌افزاری معیوب، الگوی کلمات و جمله‌های مشابهی دارند. میزونو و همکارانش پیاده‌سازی ابزاری برای جدا کردن واحدهای معیوب (FP) و سالم (NFP) ارائه دادند. سپس این واحدها در مرحله یادگیری دسته بند استفاده می‌شدند که بتوانند واحدهای معیوب یا سالم جدید را تشخیص بدهند.

همچنین رویکردی دیگر برای رده‌بندی داده‌های بزرگ برای فهم اساس و پایه آن‌ها در [۲۵] ارائه شد. هر چند این داده‌ها معمولا به عنوان داده پرت در نظر گرفته می‌شوند، ممکن است اطلاعات مفیدی راجع به پروژه و بهبود آن داشته باشند.

هیندل و همکارانش تصمیم به استفاده از روش‌های رده‌بندی برای مشخص کردن رده‌های مختلف داده‌های بزرگ و در نتیجه تشخیص تغییرات مختلف نرم‌افزار گرفتند. این مطالعه نشان داد که در بیشتر موارد، داده‌های بزرگ نشان‌دهنده تغییرات در معماری سیستم هستند.

روشی برای ارزیابی میزان پایداری یک مدل پیش‌بینی‌کننده، ارائه دادند. آن‌ها ۴ پروژه متن باز را بررسی کرده و ویژگی‌هایشان را استخراج کردند. سپس یک مدل پیش‌بینی نقص با استفاده از درخت تصمیم نرم‌افزار $Weka$ ساخته شد و محققان کیفیت پیش بینی را در طول زمان ارزیابی کردند. این مطالعه نشان داده که در طی زمان، تغییرات چشم گیری وجود داشته و در نتیجه باید محتاطانه استفاده شود.

۳.۵.۳ کاوش الگوهای تکرار شونده و قوانین انجمنی

محققان در مطالعه‌هایشان بهره برداری از روش‌های استخراج قوانین انجمنی به منظور تشخیص تغییرات همزمان در سیستم نرم‌افزاری را نشان می‌دهند. به عنوان مثال هدف ما کشف رابطه بین تغییر ماهیت‌های نرم‌افزار است. می‌خواهیم به این سوال پاسخ بدهیم که زمانی که یک ماهیت از کد منبع (مانند تابع A) تغییر می‌کند، چه ماهیت‌های دیگری ممکن است تغییر کنند (مانند توابع B و C)؟

[۱۶] Spam-Filtering

به طور خاص، ابزاری برای تجزیه کد منبع ارائه شده که شماره خطوط را به سطوح فیزیکی ماهیت‌ها مرتبط می‌کند. این ماهیت‌ها به صورت سه‌تایی (نام فایل، نوع، شماره) نمایش داده می‌شوند. تغییرات بعدی در مخزن به عنوان تراکنش گروه‌بندی شده‌اند. یک روش کاوش قوانین انجمنی سپس برای تشخیص قوانین به شکل

$$B,C \rightarrow A$$

استفاده می‌شود.

استفاده از روش‌های خوشه‌بندی و قوانین انجمنی برای بازیابی طراحی معماری سیستم‌های نرم‌افزاری قدیمی، با توجه به اهداف تعیین شده کاربر پیشنهاد شده‌است. کد منبع سیستم قدیمی تحلیل شده و مجموعه‌ای از بخش‌های پر تکرار آن استخراج می‌شود. با استفاده از روش‌های خوشه‌بندی و تطبیق الگو، الگوریتم ارائه شده بخش‌های مختلف سیستم قدیمی را تعریف می‌کند. بر اساس پرس و جوی کاربر، بهترین بخش مطابق با خواسته او انتخاب می‌شود. همچنین برای هر جواب ممکن می‌توان امتیازی در نظر گرفت که در نهایت لیستی رتبه‌بندی شده را برای ارزیابی به کاربر ارائه می‌دهد.

رویکردی برای شناسایی الگوهای استفاده مجدد کتابخانه‌ها نیز، یک روش پیشنهادی دیگر است. این رویکرد پیشنهادی از قوانین انجمنی برای شناسایی روابط بین رده‌ها در یک کتابخانه بهره می‌برد. با تعمیم قوانین انجمنی به قوانین عمومی، وراثت رده ها نیز در نظر گرفته می شود. بنابراین یک روش خودکار به منظور کشف الگوهای استفاده مجدد کتابخانه ها و شناسایی مشخصه استفاده آن ها توسعه می یابد.

یک رویکرد برای تحلیل کد نمونه برای پیدا کردن تغییرات استفاده در چارچوب نیز ارائه شده است. فرایند استخراج دو نسخه از کد نمونه را به عنوان ورودی می‌گیرد و هدف از بهره برداری از روش‌های کاوش الگوهای تکرار شونده توصیف تغییرات استفاده از چارچوب است. در مرحله اول، اطلاعاتی راجع به این که کد نمونه چگونه از چارچوب استفاده می کند (رده های استفاده شده و یا توابع فراخوانی شده) استخراج می شود. سپس تراکنش ها به وسیله ترکیب اطلاعات نحوه استفاده به دست آمده از دو نسخه کد نمونه ساخته می شوند. در نهایت یک الگوریتم استخراج قوانین انجمنی بر تراکنش ها اعمال شده و تمام قوانین تغییر ممکن استخراج می شود.

۶.۳ استفاده مجدد از نرم‌افزار

یکی از مهم ترین جنبه های افزایش بهره وری و کیفیت نرم‌افزار، استفاده مجدد از آن است.[۴۰،۴۵] اگر چه استفاده مجدد از نرم‌افزار می تواند شکل های مختلفی داشته باشد و موتور های جستجو برای بخش های نرم‌افزار در زبان های برنامه نویسی گوناگون حد فاصل تحقیقات در زمینه استفاده مجدد از نرم‌افزار هستند، اخیرا تلاش هایی برای استفاده از روش های داده کاوی برای تشخیص جنبه های تاثیر گذار بر موفقیت استفاده مجدد نرم‌افزار صورت گرفته است. انگیزه ی این رویکرد ها از این حقیقت ریشه می گیرد که تحقیقات قبلی نشان دهنده احتمال بالای شکست پروژه به دلیل کمبود فرایند استفاده مجدد و همچنین عدم وجود روشی برای تغییر فرایند های غیر قابل استفاده بود[۴۵].

در این راستا، موریسیو و همکارانش[۴۵] تلاش کردند تا عوامل مهم موثر بر موفقیت استفاده مجدد از نرم‌افزار را در پروژه انجام شده پیدا کنند. به طور خاص، از طریق مصاحبه اطلاعاتی از ۲۴ پروژه اروپایی بین سال‌های ۱۹۹۴ تا ۱۹۹۷ جمع‌آوری کردند و در تحلیلشان، ۲۷ متغیر که برای توصیف پروژه ها استفاده می‌شدند تعریف کردند که خلاصه‌ای از آن‌ها در [۴۰] آمده است. در حال حاضر مجموعه داده فوق با وجود تعداد کم نمونه ها، بزرگترین مجموعه داده تجربی برای استفاده مجدد نرم‌افزار را تشکیل می دهد. روی این داده ها الگوریتم های داده کاوی زیادی اعمال شده اند تا الگوهای مربوط به عوامل موثر در موفقیت استفاده مجدد نرم‌افزار را تشخیص دهند.

۱.۶.۳ رده بندی

در همان مطالعه [۴۰]، نویسندگان همچنین تلاش کردند از درخت تصمیم و به طور خاص پیاده سازی درخت $J.48$ در نرم‌افزار$Weka$ که پیاده‌سازی درخت تصمیم $C4.5$ است برای تحلیل همان داده‌ها استفاده کنند. کاربرد درخت تصمیم $C4.5$ به این صورت بود که مهم‌ترین ویژگی‌ها را از بین ۲۷ ویژگی با استفاده از آزمایش حذف ویژگی‌ها تشخیص دهد. به طور خاص آن‌ها ریشه درخت تصمیم را در هر مورد در زمانی که مهم‌ترین ویژگی حذف شود(مثلا ریشه درخت)، به دست آوردند و درخت مجددا بدون آن ویژگی ساخته می

شد.
این روش به آن ها اجازه می داد که ویژگی های ضعیف، که در طی مراحل حذف از درخت در ریشه ظاهر نمی شوند و یا ویژگی هایی که حذف آن ها تغییری در دقت رده بندی ایجاد نمی کند را شناسایی کنند.
علاوه بر تجزیه و تحلیل فوق، آن ها از یک الگوریتم یادگیری به اسم $treatment\ learning$ استفاده کرده و به طور خاص، الگوریتم $TAR2$ را اعمال کردند. ایده ساده $treatment\ learning$ این است که این الگوریتم، زیر مجموعه ای مانند A را از مجموعه داده های آموزشی D انتخاب می کند که بیشتر شامل رده های مطلوب بوده و تعداد کمی از رده های نامطلوب را داشته باشد. معیاری که بر اساس آن زیر مجموعه انتخاب می شود بر اساس R_x انتخاب شده است. $TAR2$ از کاربر می خواهد که عددی را به عنوان امتیاز به هر رده اختصاص دهد تا رده ها بر اساس مطلوبیت مشخص شوند. در تشخیص قابلیت استفاده مجدد، پروژه های موفق از این نظر باید امتیاز بالا تری نسبت به پروژه هایی با قابلیت استفاده مجدد پایین بگیرند. پس از تحلیل های انجام شده، آن ها می توانند ویژگی ها و مقادیر مربوط به آن ها که در پروژه های موفق استفاده شده‌اند، مشخص کنند.
بخش جالب کاربرد این الگوریتم داده کاوی این حقیقت است که این الگوریتم می تواند ویژگی ها و مقادیری را پیدا کند که مطالعه تجربی در [۴۵] نتوانسته بود آن ها را بیابد. این موضوع نشان دهنده اهمیت داده کاوی در مهندسی نرمافزار است.
از سوی دیگر، البته با استفاده از همان مجموعه داده برای استفاده مجدد نرمافزار، Jiang و همکارانش [۳۰] یک رویکرد یادگیری گروهی (به عنوان مثال روشی که تصمیم روش های رده بندی مختلف را ترکیب کرده و بهترین را ارائه بدهد) اعمال کردند. آن ها از الگوریتم گروهی $RF2TREE$ که جنگل تصادفی یا مجموعه ای از درخت های تصمیم را به یک درخت تصمیم تبدیل می کند استفاده کردند. تضمین شده است که این الگوریتم به شرط برقراری موارد زیر به خوبی کار می کند:

۱. مجموعه داده آموزشی اصلی همانند مجموعه موجود در [۴۵] بسیار کوچک باشد.

۲. در صورتی که جنگل تصادفی و یک درخت تصمیم هر دو توسط مجموعه داده آموزشی اصلی آموزش دیده باشند، دقت درخت تصادفی

از درخت تصمیم بالا تر باشد.

الگوریتم ابتدا از روی مجموعه داده اصلی جنگل تصادفی ایجاد می کند و سپس این جنگل تصادفی برای ساخت نمونه های مجازی که برای آموزش درخت های تصمیم استفاده می شوند، به کار گرفته می شود.

بر اساس آزمایش های انجام شده، آن ها دریافتند که مهم ترین ویژگی هایی که بر موفقیت استفاده مجدد نرم‌افزار اثر می گذارند عبارت اند از: عوامل انسانی، فرایند استفاده مجدد معرفی شده، نوع نرم‌افزار تولید شده، تعهد مدیر ارشد، و تغییرات فرایند های غیر قابل استفاده مجدد که با تحلیل های تجربی و تحلیل های داده کاوی تفاوت دارند.

۲.۶.۳ کاوش الگوهای تکرار شونده و قوانین انجمنی

Menzies و Stefano Di [۴۰] بر روی داده ذکر شده کار کردند تا بتوانند نتایج بعدی عوامل موثر روی استفاده مجدد نرمافزار را بررسی کرده و الگوهای حاصل از اعمال روش های خودکار داده کاوی را با روش های تجربی گفته شده در [۴۵] مقایسه کنند. از بین روش های مختلف، کاوش قوانین انجمنی به کار گرفته شد تا ارتباط معنا دار بین ۲۷ ویژگی را استخراج کند. استخراج قوانین انجمنی با استفاده از الگوریتم *Apriori*[۱] پیاده سازی شده توسط نرم‌افزار *Weka* انجام شد. ۱۰ قانون برتر با اطمینان ۹۰% در [۴۰] آمده است. با استفاده از قوانین انجمنی به دست آمده، نتایج جالبی به دست آمد. به عنوان مثال این حقیقت که زمانی که یک محصول نرم‌افزاری در محصولی استفاده شده بود، استفاده از سیاست پاداش برای استفاده مجدد از نرمافزار فعال نشده بود.

$$\text{SoftwareandProduct} = \text{product} => \text{RewardsPolicy} = \text{no}$$

یکی دیگر از جنبه های مهم داده کاوی در مهندسی نرم‌افزار استخراج الگوهای طراحی از نرم‌افزار به منظور استفاده مجدد آن ها است. به طور خاص، فرایند داده کاوی با هدف استخراج تصمیم های طراحی که معمولا در کد منبع پنهان شده اند انجام می شود. به طور معمول در طی فرایند طراحی سیستم نرم‌افزاری، قسمت های مختلف سیستم با طراحی اعمال شده برچسب گذاری نمی شوند و در نتیجه تصمیم های طراحی با سیستم موجود سازگار نیستند. این مشکل باعث عدم درک جزئیات سیستم می شود.

در این راستا، روش ها و ابزار هایی با هدف استخراج الگوهای طراحی از یک سیستم نرم‌افزاری ارائه شده اند.

فصل ۴

نتیجه گیری

با توجه به تولید داده‌های زیاد در مهندسی نرم‌افزار، کاربرد صحیح این داده ها برای حل مشکلات چرخه تولید نرم‌افزار اهمیت بسزایی دارد. برخی مشکلات جدی همانند ایجاد اشکال، افزایش هزینه نگهداری برنامه یا نیازمندی های نامعلوم می تواند خلاقیت و کیفیت نرم‌افزار را کاهش دهند. این نوشتار به برخی روش‌های داده‌کاوی که توانایی به کارگیری همراه داده‌های مهندسی نرم‌افزار را در چالش‌های تولید، مدیریت، اشکال‌زدایی و نگهداری دارند، اشاره کرده است. ارزش نتایج داده‌کاوی است که آن را توجیه‌پذیر می کند و به همین دلیل کیفیت و کمیت دادههای در دسترس و هزینه عملیات محاسباتی آن‌ها موفقیت این روش را در فرایند تولید نرم‌افزاری تعیین می‌کند. مهندسان و داده کاوان باید روش‌های داده‌کاوی را طوری انتخاب کنند که هزینه ابزار کاهش یابد و از داده‌های در دسترس بهترین استفاده به عمل آید. پیش بینی می‌شود در آینده تحقیقات این رشته در زمینه افزایش اتوماسیون و ساده‌تر شدن فرایندها متمرکز شود.

کتاب‌نامه

[۱] M. Halkidia، D. Spinellisb، G. Tsatsaronisc and M. Vazirgiannis ، Data mining in software engineering

[۲] J.Dong،Y. Zhao and T. Peng،Areviewof design patternmining techniques، International Journal of Software Engineering and Knowledge Engineering (۶)۱۹ ،(۲۰۰۹) .۸۵۵—۸۲۳

[۳] P. Francis، D. Leon، M. Minch and A. Podguraki، Treebased method for classifying software failures. In Proceedings of the ۱۵th International Symposium on Software Reliability Engineering، .۲۰۰۴ software modifications. In Proceedings of ۲۰th IEEE International Conference on Software Maintenance (ICSM'۰۴)، .۲۰۰۴

[۴] A. Hindle، D. German and R. Holt،What do large commits tell us؟ a taxonomical study of large commits. In Proceedings of the IEEE Working Conference on Mining Software Repositories، .۲۰۰۸

[۵] Y. Jiang، M. Li and Z.H. Zhou، Mining extremely small data sets with application to software reuse، Softw، Pract Exper (۴)۳۹ ،۲۰۰۹ .۴۴۰—۴۲۳

[۶] M. Last،M. Friedman and A. Kandel، The Data Dimining Approach to Automated Software Testing، In Proceeding of the SIGKDD Conference، .۲۰۰۵

[۷] T. Menzies and J.S. Di Stefano، More success and failure factors in software reuse، IEEE Trans Software Eng (۵)۲۹ ،(۲۰۰۳) .۴۷۷—۴۷۴

[۸] S.Morisaki،A.Monden andT.Matsumura، Defect data analysis based on extended association rulemining. InProceedings of International Workshop on Mining Software

Repositories (MSR)، .۲۰۰۷
[۹] M.Morisio، M. Ezran and C. Tully، Success and failure factors in software reuse، IEEE Trans Software Eng (۴)۲۸ ،(۲۰۰۲) .۳۵۷–۳۴۰
[۱۰] A. Podgurski، W. Masri، Y. McCleese، M. Minch، J. Sun، B. Wang and W. Masri، Automated support for classifying software failure reports. In Proceedings of the ۲۵th International Conference on Software Engineering، .۲۰۰۳
[۱۱] U. Raza and M.J. Tretter، Predicting software outcomes using data mining and text mining. In SAS Global Forum، .۲۰۰۷
[۱۲] C.C.Williams and J.K.Hollingsworth،Automatingmining of source code repositories to improve bug finding techniques، IEEE Transactions on Software Engineering (۶)۳۱ ،(۲۰۰۵) .۴۸۰–۴۶۶
[۱۳] Lovedeep، Varinder Kaur Atri، Applications of Data Mining Techniques in Software Engineering، ISSN (Online): ،۲۳۴۷-۲۸۲۰ Volume ،۲- Issue-۵،۶، ۲۰۱۴

Abstract

The increased availability of data created as part of the software development process allows us to apply novel analysis techniques on the data and use the results to guide the process's optimization. In this paper we describe various data sources and discuss the principles and techniques of data mining as applied on software engineering data. Data that can be mined is generated by most parts of the development process: requirements elicitation، development analysis، testing، debugging، and maintenance. Based on this classification we survey the mining approaches that have been used and categorize them according to the corresponding parts of the development process and the task they assist. Thus the survey provides researchers with a concise overview of data mining techniques applied to software engineering data، and aids practitioners on the selection of appropriate data mining techniques for their work.

Faculty of Science
School of mathematics، statistics and computer science

Application of Data mining in Software engineering

By

Zahra Omidi

Supervisor

Ebrahim Naghibzade Mashayekh

Project for recieving bachelor degree
Computer Science

July 2018