علوم و مهندسی کامپیوتر
نشریه علمی پژوهشی انجمن کامپیوتر ایران
مجلد ۱۲، شماره ۳ (الف)، ۱۳۹۳
صفحات ۲-۱۵
مقاله عادی

بررسی و طبقه‌بندی زبان‌های برنامه‌نویسی جنبه‌گرا و تکنیک‌های کاوش جنبه

الهه حبیبی عباس حیدرنوری

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، ایران

چکیده

تولید نرم‌افزار به شیوه جنبه‌گرا[۱]، رویکردی نوین برای مدیریت پیچیدگی، از طریق پیمانه‌ای نمودن کد برنامه در سیستم‌های نرم‌افزاری بزرگ است. زبان‌های برنامه‌نویسی جنبه‌گرا این امکان را می‌دهند تا کد مورد نیاز برای پیاده‌سازی یک وظیفه‌مندی[۲] خاص که در نقاط مختلف برنامه پراکنده شده‌اند را در قالب یک جنبه، پیمانه‌ای[۳] کنیم. زبان‌ها و ابزارهای متفاوتی برای برنامه‌نویسی جنبه‌گرا وجود دارد که هر کدام دارای قواعد دستوری و معنایی خاص خود می‌باشند. در این مقاله برآنیم تا ضمن بررسی مفاهیم پایه‌ای زبان‌های برنامه‌نویسی جنبه‌گرا، انواع رویکردها و زبان‌های موجود در این زمینه را مورد مطالعه و ارزیابی قرار دهیم. علاوه بر این، یکی از مسائل مهمی که در برنامه‌نویسی جنبه‌گرا وجود دارد، تعیین جنبه‌های سیستم (کاوش جنبه[۴]) در فازهای مختلف فرآیند ایجاد نرم‌افزار است. تاکنون رویکردهای متفاوتی در این زمینه ارائه گردیده است. در این مقاله، این رویکردها به همراه ابزارهای مربوطه را مورد بررسی و تحلیل قرار داده‌ایم.

کلمات کلیدی: جنبه، برنامه‌نویسی، جنبه‌گرا، کاوش جنبه، زبان‌های برنامه‌نویسی، پیمانه‌ای نمودن.

۱- مقدمه

در اغلب سیستم‌های نرم‌افزاری، کدی که یک وظیفه‌مندی خاص را پیاده‌سازی می‌کند، در کل کد برنامه پراکنده[۵] شده و با کدی که دیگر وظیفه‌مندی‌ها را پیاده‌سازی می‌نماید، در هم تنیده[۶] شده است. به این نوع کد، در متون مهندسی نرم‌افزار، به عنوان دغدغه‌های سراسری[۷] نام برده می‌شود و اغلب، موجب کاهش پیمانه‌ای بودن سیستم می‌گردد [۱]. برای حل این مسئله در تولید نرم‌افزار، مدت‌هاست که مهندسین نرم‌افزار از قانون جداسازی مسائل[۸] استفاده می‌کنند و اغلب زبان‌های برنامه‌نویسی نیز، به صورت صریح (اما محدود) از این قانون پشتیبانی می‌نمایند. ولیکن، مشکل دغدغه‌های سراسری همچنان در اغلب سیستم‌های نرم‌افزاری وجود دارد. بنابراین، برای بهبود در جداسازی مسائل، G. Kickzales و همکارانش، برنامه‌نویسی جنبه‌گرا را معرفی نمودند [۱]. روش برنامه‌نویسی جنبه‌گرا در کنار دیگر زبان‌های برنامه‌نویسی، موجب افزایش پیمانه‌ای بودن تصمیمات طراحی و در نتیجه، سیستم‌های نرم‌افزاری می‌شود [۲]. تکنیک جنبه‌گرا، منحصر به فاز طراحی و تصمیمات آن نیست، بلکه از همان مراحل ابتدایی تولید نرم‌افزار مانند فازهای شناسایی نیازمندی‌ها و یا تحلیل سیستم نیز می‌توان، برای جداسازی و پیمانه‌ای نمودن دامنه مسئله، از آن بهره گرفت [۳، ۴، ۵]. بدین ترتیب رویکردهای متفاوتی در زمینه برنامه‌نویسی جنبه‌گرا وجود دارد. به طور کلی برنامه‌نویسی جنبه‌گرا را می‌توان به دو شاخه مفاهیم زبان برنامه‌نویسی و کاوش جنبه‌ها تقسیم نمود. در پیاده‌سازی یک برنامه جنبه‌گرا علاوه بر اهمیت وجود زبان‌ها و ابزارهای مناسب جهت برنامه نویسی، تکامل آن برنامه نیز از جایگاه ویژه‌ای برخوردار خواهد بود.

در این مقاله برآنیم تا مروری بر زبان‌ها، چارچوب‌ها[۹]، و رویکردهای جنبه‌گرا داشته باشیم و آن‌ها را با یکدیگر مقایسه نماییم. مسائلی که در تکامل[۱۰] سیستم‌های جنبه‌گرا وجود دارند و همچنین روش‌های شناسایی و استخراج جنبه‌ها که در متون مهندسی نرم‌افزار به عنوان روش‌های کاوش جنبه نام برده می‌شوند، از جمله موارد دیگری می‌باشند که در این مقاله مورد بررسی قرار خواهیم داد.

در ادامه این مقاله، بخش ۲، به معرفی برنامه‌نویسی جنبه‌گرا و مفاهیم پایه‌ای آن می‌پردازد. سپس، بخش ۳، مروری بر زبان‌های برنامه‌نویسی جنبه‌گرا دارد. در

ادامه، بخش ۴، ابزارها و چارچوب‌هایی که برای برنامه‌نویسی جنبه‌گرا وجود دارند را معرفی می‌نماید. بخش ۵، رویکردهای جنبه‌گرا در توسعه سیستم‌های نرم‌افزاری را مورد بررسی قرار می‌دهد. سپس، بخش ۶، در زمینه تکامل سیستم‌های نرم‌افزاری جنبه‌گرا بحث می‌کند. در ادامه، بخش ۷، روش‌های کاوش جنبه را ارائه می‌نماید. در نهایت، بخش ۸، به نتیجه‌گیری می‌پردازد.

۲- معرفی برنامه‌نویسی جنبه‌گرا

در مهندسی نرم‌افزار، روش‌های متفاوتی برای پیمانه‌ای نمودن سیستم‌های نرم‌افزاری ارائه شده است. یکی از شناخته‌شده‌ترین این روش‌ها، روش برنامه‌نویسی شی‌گرا می‌باشد. هدف برنامه‌نویسی شی‌گرا، نمایش دنیای واقعی به کمک اشیا می‌باشد. اما در برخی مواقع به دلیل وجود مسائلی در طراحی و همچنین پیچیدگی دامنه، پیمانه‌ای بودن سیستم حفظ نمی‌شود [۶، ۷]. ویژگی اصلی این مسائل، که با نام دغدغه‌های سراسری شناخته می‌شوند، وجود تکراری آن‌ها در چندین واحد[11] برنامه است؛ به این معنا که در سراسر کد، پراکنده و در درون آن‌ها تنیده شده‌اند. بدین ترتیب دغدغه‌های سراسری، به هنگام تغییرات در کد برنامه، می‌توانند منبع بروز مشکلات در سیستم نرم‌افزاری شوند. بدین ترتیب، تأثیراتی منفی بر قابلیت نگهداری و تکامل نرم‌افزار خواهند داشت. برنامه‌نویسی جنبه‌گرا با فراهم آوردن یک مکان مشترک برای قرارگیری و پیاده‌سازی این مسائل، سعی در حل این مشکلات دارد. این مکان مشترک، یک واحد در کد به نام جنبه می‌باشد که ضمن افزایش میزان پیمانه‌ای بودن کد، صحت عملکرد برنامه را نیز حفظ می‌کند [۱، ۶، ۷، ۸]. لازم به ذکر است که کد جنبه، دارای قابلیت استفاده مجدد نیز می‌باشد. مسائلی مانند ثبت وقایع[12] یا مدیریت خطاها نمونه‌هایی از دغدغه‌های سراسری هستند.

تاکنون برنامه‌نویسی جنبه‌گرا را بر روی سیستم‌های بسیاری اعمال کرده‌اند. به عنوان مثال، در مقاله [۹]، برنامه‌نویسی جنبه‌گرا بر روی ابزار مدیریت کارایی مادون‌قرمز[13] اعمال شده است. نویسندگان این مقاله، مسئله مانیتور کردن کارایی برنامه‌های سازمانی را یک دغدغه سراسری می‌دانند و همین امر را دلیلی بر اعمال شیوه جنبه‌گرا بر ابزار مدیریت کارایی بیان داشته‌اند. در این مقاله ضمن اعمال روش جنبه‌گرا، مقایسه‌ای نیز میان این شیوه و روش‌های قبلی انجام گرفته است. نتیجه این مقایسه، برتری روش جنبه‌گرا، به دلیل انعطاف‌پذیری بالاتر بوده است. در مقاله [۱۰] نیز، با استفاده از برنامه‌نویسی جنبه‌گرا به بهبود کارایی برنامه‌هایی که با پایگاه داده‌ها کار می‌کنند، پرداخته است. در این مقاله به‌روز شدن تحلیل‌های آماری را یک دغدغه سراسری معرفی می‌نماید.

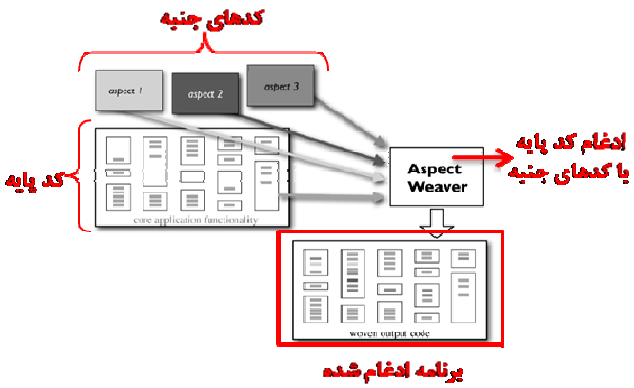
این تحلیل‌های آماری برای بهینه‌سازی پرس‌وجوها[14] در پایگاه داده مفید می‌باشند. زمان‌بر بودن محاسبات آماری و حجیم بودن داده‌های مربوط به جدول‌های مختلف از جمله مشکلات بیان شده در این مقاله‌اند، که با برنامه‌نویسی جنبه‌گرا سعی در برطرف نمودن آن‌ها دارد. با وجود مشکلاتی مانند برپایی محیط مناسب برای اعمال شیوه جنبه‌گرا، استفاده از این شیوه موجب افزایش کارایی (تا ۲۵ درصد) در مقایسه با روش‌های دیگر بوده است.

استفاده از برنامه‌نویسی جنبه‌گرا در حوزه آکادمیک در مقایسه با صنعت، پیشرفت ملموس‌تری داشته است و مثال‌های بیشتری نیز در این حوزه یافت می‌شود. بدین ترتیب، برخی افراد آکادمیک تلاش کرده‌اند تا برنامه‌نویسی جنبه‌گرا را بیشتر به حوزه صنعت وارد نمایند. در مقاله [۱۱]، به بررسی مزایای استفاده از شیوه جنبه‌گرا در صنعت و برای نرم‌افزارهای تجاری پرداخته است. برای دستیابی به این هدف، مصاحبه‌ای با ۱۱ نفر از ۵ کشور مختلف، در زمینه استفاده از برنامه‌نویسی جنبه‌گرا انجام گرفته است. جمع‌بندی این مصاحبه‌ها، ذکر مزایایی مانند خوانایی کد، کاهش پیچیدگی و کاهش خطاها برای روش برنامه‌نویسی جنبه‌گرا بوده است. اما بهره‌گیری از این شیوه، معایبی را نیز به همراه داشته است، مانند نیاز به استفاده از ابزارهای برنامه‌نویسی جنبه‌گرا، و ناسازگاری برخی از ابزارهای جنبه‌گرا با سیستم‌های موجود. در بخش ۴، مثال‌های بیشتری در زمینه استفاده از روش‌های برنامه‌نویسی جنبه‌گرا در صنعت و همچنین در حوزه آکادمیک آورده شده است.

۲-۱- مفهوم جنبه در برنامه‌نویسی جنبه‌گرا

همانطور که در قبل نیز بیان شد، برنامه‌نویسی به روش جنبه‌گرا، با ایجاد مفهومی به نام جنبه، مشکل دغدغه‌های سراسری را برطرف ساخته است. به این ترتیب که دغدغه‌های سراسری در کد اصلی برنامه، به کدهای جنبه تبدیل می‌شوند. در واقع می‌توان گفت، جنبه به عنوان یک عنصر مستقل در کد خواهد بود که به راحتی می‌تواند به کد اضافه و یا از آن حذف شود [۷، ۱۲، ۱۳]. مقاله [۴] با نگاهی متفاوت، جنبه را مفهومی می‌داند که از همان ابتدای فرآیند تولید نرم‌افزار باید به آن پرداخته شود، تا در مراحل نهایی تولید، منجر به پیچیدگی کد و پراکندگی در آن نشود.

مقاله [۱۱] جنبه را به دو نوع همگن و ناهمگن تقسیم کرده است. جنبه همگن، جنبه‌ای است که بر مکان‌های مشخصی اعمال می‌شود، در حالی که جنبه ناهمگن ممکن است بر روی چندین مکان مختلف با رفتارهای متفاوت در هر مکان، اعمال گردد. ثبت وقایع و ردیابی[15] از جنبه‌های همگن می‌باشند و تصمیم‌گیری میان دو پروتکل برای هر بار انتقال اطلاعات در شبکه جنبه‌ی ناهمگن به شمار می‌رود، زیرا در هر بار انتقال اطلاعات، ممکن است پروتکل‌های متفاوتی (در مقایسه با انتقال قبلی)، وجود داشته باشد که البته نوع اطلاعات انتقالی نیز در ناهمگن بودن جنبه بی‌تأثیر نخواهد بود.



شکل ۱- نحوه ادغام جنبه‌ها با کد پایه به وسیله Aspect Weaver [۶]

۲-۲- مفاهیم برنامه‌نویسی جنبه‌گرا

سه فاز مهم در برنامه‌نویسی جنبه‌گرا وجود دارد: کشف جنبه‌ها، استخراج جنبه‌ها، و تکامل جنبه‌ها. "آیا یک سیستم نرم‌افزاری، نیاز به برنامه‌نویسی جنبه‌گرا دارد؟ آیا دغدغه‌های سراسری در این نرم‌افزار وجود دارند؟"؛ این دو مسئله، از سؤالاتی هستند که در فاز اول (کشف جنبه) مطرح می‌شوند. در طول این فاز، کاندیداهای مناسب برای جنبه شناسایی می‌گردند. در این فاز همچنین، چگونگی پیاده‌سازی جنبه و تأثیر آن بر کیفیت نرم‌افزار مورد بررسی قرار می‌گیرد. در فاز استخراج، که به معنای جداسازی کدهای دغدغه‌های سراسری از کد اصلی است، پس از شناسایی جنبه‌ها در فاز اول، سیستم به سمت برنامه‌نویسی جنبه‌گرا حرکت می‌کند. با تبدیل کاندیداها به جنبه، رفتار سیستم بار دیگر بررسی می‌شود تا از

صحت عملکرد سیستم، اطمینان حاصل گردد. اما نرم‌افزاری کـه بـا برنامـه‌نویسـی جنبه‌گرا ایجاد شده است، مانند هر نرم‌افزار دیگری نیاز به تکامل دارد. این مسـئله در فاز سوم، مورد بررسی قرار می‌گیرد [۶].

همانطور که در بخش قبلی نیز بیان شد، هدف برنامه‌نویسی جنبه‌گرا، پیمانه‌ای نمودن سیستم، با استفاده از مفهوم انتزاعی جنبه می‌باشد. این جنبه‌ها به صـورتی مستقل از برنامه اصلی تعریف می‌شوند و به وسیله ابزاری به نام Aspect Weaver با کد پایه، ادغام می‌گردند. این ابزار، همانند یک کامپایلر عمـل کـرده و مـی‌توانـد عمل ادغام را در زمان کامپایل و یـا در زمـان اجـرا انجـام دهـد [۶، ۸]. شـکل ۱، عملکرد این ابزار را توصیف می‌نماید.

در زبان‌های برنامه‌نویسی جنبه‌گرا، سه مفهوم مهم و کلیدی وجود دارد: Join point، Pointcut، و Advice. مکانی که قرار است تا کد جنبـه بـه برنامـه اصـلی متصل گردد، Join point نامیده می‌شود. مجموعه‌ای از این مکان‌هـا (Join point ها)، Pointcut می‌باشد [۱، ۶، ۸]. لازم به ذکر است که اگر برنامه‌ای دارای تعـداد کمی از Join pointها باشد، کاندیدای مناسبی برای برنامه‌نویسی جنبه‌گرا نیسـت. Adviceها، کدهای مربوط به دغدغه‌های سراسری بوده و قـرار اسـت بـه کـدهای جنبه‌ها تبدیل شوند.

با استفاده از Aspect Weaver، این کدها به سه طریق مختلف مـی‌تواننـد در کد پایه برنامه ادغـام شـوند: اگـر کـد Advice قبـل از Join point اضـافه شـود، before advice؛ اگر کد Advice بعد از Join point اضافه شـود، after advice؛ و اگر کد Advice دقیقاً در محل Join point به کـد اصـلی برنامـه اضـافه شـود، around advice نامیده می‌شود.

به عنوان مثال، شکل ۲، قسمت‌هایی از یک برنامه جاوا را نشان مـی‌دهـد کـه برای ترسیم اشکال مختلف بر روی صفحه نمایش مورد استفاده قرار می‌گیرد. امـا، قبل از ترسیم هر شکل جدید بر روی صفحه، بایستی کـه صـفحه نمـایش پـاک و آماده شود. برای اجتناب از تکرار کد پاک کـردن صـفحه نمـایش (متـد refresh)، قبل از ترسیم هر شکل جدید، می‌توان این متد را بـه صـورت یـک جنبـه تعریـف نمود.

```
Public class Shape {...}
Public class Rectangle extends Shape {
    ...
    Void set Left Corner (corner: Point) {...}
    Void set Right Corner (corner: Point) {...}
}
Public class Circle extends Shape {
    ...
    Void set Redius (lenght: float) {...}
}
```

شکل ۲- کلاس‌های ترسیم اشکال هندسی در صفحه نمایش

```
Public aspect Update Page {
    Pointcut Display State Change () {
        Execution (void Shape+.set*(...))
    }
    After () return Display State Change () {
        Display.refresh ();
    }
}
```

شکل ۳- کلاس جنبه، دارای متد refresh

به طوری که تنها یک بار آن کد نوشته شود و سپس بـا اسـتفاده از Aspect Weaver، به صورت خودکار در مکان مورد نظر در برنامـه و در زمـان کامپایـل بـا برنامه اصلی ادغام گردد. نمونه‌ای از این کد، در شکل ۳ آورده شده اسـت. در ایـن شکل، Pointcut متدهایی را که عمـل refresh بـر روی آن‌هـا اعمـال مـی‌شـود، مشخص می‌نماید. سپس، after advice کد اصلی مربوط به جنبه (متـد refresh) را فراهم می‌کند.

۳- زبان‌های برنامه‌نویسی جنبه‌گرا

برای پشتیبانی از برنامه‌نویسی جنبه‌گرا، لازم است تا زبان‌های برنامه‌نویسی کنونی، گسترش یابند. برنامه‌نویسی جنبه‌گرا نیز مانند هر روش دیگر برنامه‌نویسـی، دارای قواعد دستوری و معنایی با نمادهـای خـاص هـر زبـان و ابزارهـای متفـاوت بـرای پیاده‌سازی آن زبان‌هاست. زبان‌های برنامه‌نویسی جنبه‌گرا را می‌توان بـه دو دسـته خاص جنبه‌گرا و مستقل از جنبه‌گرا تقسیم‌بندی نمود. در ادامه بـه بررسـی چنـد نمونه از این زبان‌ها می‌پردازیم.

۳-۱- زبان‌های برنامه‌نویسی خاص جنبه‌گرا

این گونه زبان‌های جنبه‌گرا، گسترشی[۱۶] بر یک زبان برنامه‌نویسی خاص هستند، به این معنا که با قواعد دستوری همان زبان، سازگار می‌باشـند و نمـی‌تـوان از قواعـد دستوری و معنایی این زبان جنبه‌گرا، در یک زبان دیگر استفاده نمود. در ادامه بـه دو نمونه از این زبان‌ها اشاره می‌کنیم.

۳-۱-۱- زبان برنامه‌نویسی AspectJ

زبان برنامه‌نویسی AspectJ، که معروف‌ترین زبان جنبه‌گرا می‌باشد، گسترشـی بـر زبان جاوا است [۵، ۶، ۸]. این زبان که در ابتدا در مرکز تحقیقاتی Xerox PARC ایجاد شده است، اکنون به عنوان بخشی از پـروژه مـتن بـاز[۱۷] Eclipse مـی‌باشـد [۲، ۸].

Join point در این زبان شامل فراخـوانی متـد، فراخـوانی سـازنده[۱۸]، اجـرای فراخوانی متد، اجرای فراخوانی سازنده، فیلدهای get و set، اجرای مدیریت خطاها و آماده‌سازی کلاس و شی، می‌باشد؛ اما حلقه‌ها و عبارات شرطی if، در ایـن زبـان پشتیبانی نمی‌شوند. نماد "*"، در این زبان به معنای "همه" می‌باشـد. بـه عنـوان مثـال، عبـارت call(**BankAccount:*(*))، تمـامی فراخـوانی‌هـای روی BankAccount را که دارای یک پارامتر می‌باشند، برمی‌گردانـد. نمـاد "+" بـرای نشان دادن subtype کاربرد دارد. به عنوان مثـال Exception+، throws، تمـامی خطاهایی که از نوع Exception هسـتند را throw مـی‌کنـد. Pointcut در زبـان AspectJ به صورت زیر نوشته می‌شود:

```
pointcut pointcutName ([parameters]):designator(joinpoint)
```

نام pointcut

در جلوی این عبارت، joinpointها گذاشته می شود

زبـان AspectJ، از Adviceهـای نـوع after، before، و around پشـتیبانی می‌کند. جنبه‌ها در AspectJ مانند کلاس در زبان جاوا تعریف مـی‌شـوند و دارای فیلد و متد می‌باشند [۷، ۸]. بنـابراین بـرای اسـتفاده از ایـن زبـان نیـاز اسـت تـا برنامه‌نویس یک فایل کلاس جاوا ایجاد کند و جنبه را در آن بنویسـد. سـپس کـد اصلی برنامه با کد جنبه، از طریق کامپایلر AspectJ با یکدیگر ادغام می‌شوند [۸]. مدیریت خطاها، مکانیسم سودمندی است که زبان AspectJ فراهم می‌آورد [۷].

۳-۱-۲- زبان برنامه‌نویسی AspectC++

زبان AspectC++ از نظر مفهومی مشابه زبان AspectJ می‌باشد. طراحان این زبان، سعی کرده‌اند تا قواعد دستوری و معنایی این زبان را با قواعد زبان C++ سازگار نمایند [۸، ۱۴، ۱۵]. نماد "*" در زبان AspectJ به نماد "%" در زبان AspectC++ تغییر یافته است، زیرا نماد "*" در زبان C++ برای اشاره‌گرها استفاده می‌شود. ولیکن، Join pointها در این زبان، همانند زبان AspectJ می‌باشند. Pointcut ها نیز به صورت عباراتی مشابه با عبارات منظم[۱۹] نوشته می‌شوند که می‌توانند با توجه به نوع خود، در زمان کامپایل و یا در زمان اجرا، ارزیابی گردند. زبان AspectC++، از Adviceهای نوع after، before و around پشتیبانی می‌نماید. همچنین، جنبه را می‌توان مانند کلاسی که دارای فیلد و متد می‌باشد، تعریف نمود [۸، ۱۴، ۱۵].

۳-۲- زبان‌های برنامه‌نویسی مستقل از جنبه‌گرا

این زبان‌ها، برخلاف زبان‌های خاص، به هیچ یک از زبان‌های برنامه‌نویسی وابسته نیستند. زبان .NET Weave، نمونه‌ای از زبان‌های مستقل می‌باشد. این زبان به عنوان یک مؤلفه .NET، نوشته شده است و دارای دو زیرسیستم می‌باشد: تولید کد و مدل‌سازی جنبه. زیرسیستم تولید کد، زبان اسمبلی موجود (ورودی زبان .NET Weave) را به یک اسمبلی پویا تبدیل می‌کند. زیرسیستم مدل‌سازی جنبه، فایل XML که شامل توصیف جنبه می‌باشد را ترجمه نموده و جنبه را به صورت Advice و Pointcut مدل می‌کند. همچنین در این زیرسیستم، تطابق Join point ها با جنبه‌ها مورد بررسی قرار می‌گیرد [۸، ۱۶]. می‌توان گفت که زبان .NET Weave به نوعی از زبان AspectJ ارث برده شده است، به گونه‌ای که تعاریف Advice، Pointcut و Join point مشابه زبان AspectJ می‌باشد.

۴- ابزارها و چارچوب‌های جنبه‌گرا

چارچوب‌های جنبه‌گرا به طراحان و برنامه‌نویسان جنبه‌گرا از طریق یک مجموعه از جنبه‌های از پیش‌آماده شده، کمک می‌کنند. این چارچوب‌ها می‌توانند در دو دسته چارچوب‌های زمان پیاده‌سازی (جدول ۱) و چارچوب‌های زمان جمع‌آوری نیازمندی‌ها (جدول ۲) تقسیم‌بندی شوند. دسته اول در هنگام پیاده‌سازی و یا بعد از آن در زمان تکامل نرم‌افزار کاربرد دارند. نکته قابل توجه در این زمینه این است که چارچوب‌های زمان پیاده‌سازی ارائه شده برای زبان برنامه‌نویسی جاوا، در مقایسه با زبان‌های برنامه‌نویسی دیگر، پرکاربردتر می‌باشند. دسته دوم، ابزارهایی برای شناسایی دغدغه‌های سراسری در زمان جمع‌آوری نیازمندی‌ها می‌باشند. این ابزارها، خود به دو دسته متقارن[۲۰] و نامتقارن[۲۱] دسته‌بندی شده‌اند [۴]. ابزارهای نامتقارن مربوط به مجموعه‌ای از دغدغه‌های سراسری خاص می‌باشند که از یک بُعد مورد توجه هستند (مانند ابزارهای Arcade، Theme/Doc، AO Use cases و غیره). این در حالی است که با استفاده از ابزارهای متقارن می‌توان هر دغدغه سراسری را از چندین بُعد مورد بررسی قرار داد. به عنوان نمونه، جنبه امنیت از چندین بُعد قابل بررسی است. امنیت در احراز هویت کاربر برای کارهای بانکی، یا امنیت در انتقال اطلاعات در شبکه، می‌تواند به هنگام برنامه‌نویسی جنبه‌گرا، رفتارهای متفاوتی را داشته باشد. بنابراین نمی‌توان مسئله امنیت در یک سیستم بانکداری را به یک صورت مدل نمود.

علاوه بر چارچوب‌های معرفی شده در جدول ۱، چارچوب‌های بسیاری برای برنامه‌نویسی به شیوه جنبه‌گرا (در زمان پیاده‌سازی) وجود دارد، که از آن جمله می‌توان به #AOP، ABC، #Aspect، Orthogonal Weaving Model، Aspect.NET، AspectDNG، Loom.NET، PostSharp و بسیاری دیگر، اشاره داشت. اطلاعات بیشتر در این زمینه در تحقیقات ارائه شده در مقالات [۱۷] و [۱۸] وجود دارد.

۵- رویکردهای جنبه‌گرا

موضوع مهمی که در برنامه‌نویسی جنبه‌گرا وجود دارد، تعیین جنبه‌های سیستم در فازهای مختلف فرآیند ایجاد نرم‌افزار است. برخی بر این باورند که تعیین جنبه‌های سیستم منحصر به زمان پیاده‌سازی آن نیست و در همان فازهای ابتدایی نیز می‌توان، جنبه‌های سیستم را مشخص نمود [۳]. بر این اساس، همان‌گونه که در جدول ۳ نشان داده شده است، رویکردهای متفاوتی وجود دارند که در ادامه به برخی از آن‌ها اشاره می‌شود.

۵-۱- مدل طراحی جنبه‌گرا

مدل طراحی جنبه‌گرا، رویکردی است که در مقاله [۱۹] به آن اشاره شده است. در این رویکرد، جنبه‌ها در زمان طراحی، برای زبان برنامه‌نویسی AspectJ تعریف می‌شوند و در زمان پیاده‌سازی با استفاده از این زبان، برنامه‌نویسی می‌گردند. برای شناسایی جنبه‌ها در این رویکرد، نیاز است تا مفاهیم UML و AspectJ در کنار هم قرار داده شوند. برای این منظور، مدل‌های رفتاری و ساختاری مانند نمودار کلاس[۲۲]، نمودار ترتیب[۲۳] و نمودار تعاملات برای شناسایی Join point ها و جنبه‌ها، مورد استفاده قرار می‌گیرند. اگر چه مثال‌هایی از این رویکرد مانند زمان‌بندی در سیستم خطوط تلفن وجود دارد [۱۹]، اما این رویکرد هنوز به طور عملی در دنیای واقعی مورد آزمون قرار نگرفته است.

۵-۲- رویکرد طراحی JAC

رویکرد طراحی JAC نیز همانند رویکرد طراحی جنبه‌گرا، در زمان طراحی، در پی یافتن جنبه‌ها می‌باشد [۲۰]. در این مقاله چارچوبی متن باز[۲۴] که شامل یک محیط توسعه نیز می‌باشد، برای برنامه‌نویسی جنبه‌گرا معرفی شده است. رویکرد طراحی JAC، یک رویکرد شناخته شده است که تاکنون برای یافتن جنبه‌هایی مانند احراز هویت[۲۵]، ردیابی و جلسه[۲۶] در برنامه‌های کاربر- کارگزار[۲۷] استفاده شده است. این رویکرد همچنین در طراحی وب‌سایت دروس و ابزارهای مدیریت امور تجاری مورد آزمون قرار گرفته است.

۵-۳- ایجاد نرم‌افزار جنبه‌گرا با استفاده از مدل‌سازی UML

ایجاد نرم‌افزار جنبه‌گرا با استفاده از مدل‌سازی UML، رویکرد دیگری است که در [۲۱] به آن اشاره شده است. در این رویکرد، دغدغه‌های سراسری در فاز جمع‌آوری نیازمندی‌ها به کمک موارد کاربرد[۲۸] مشخص می‌گردند. سپس، در فاز تحلیل، با کمک نمودارهای کلاس و ارتباطات[۲۹] مورد تحلیل قرار می‌گیرند. در فاز طراحی، به کمک نمودارهای ترتیب و مؤلفه طراحی می‌شوند و در فاز پیاده‌سازی، به وسیله زبان‌های AspectJ و یا HyperJ پیاده‌سازی می‌گردند. در [۲۱]، این رویکرد را برای پیاده‌سازی سیستم مدیریت هتل اعمال کرده‌اند. به غیر از این مقاله، تاکنون کاربردی از این رویکرد در صنعت گزارش نشده است. همچنین، هیچ گونه ابزاری برای پشتیبانی از این رویکرد وجود ندارد.

۵-۴- رویکرد رفتاری Aspect Weaver

رویکرد رفتاری Aspect Weaver با استفاده از نمودار ترتیب پیام[۳۰] و با کمک الگوریتم weaving (الگوریتمی برای شناسایی Join point) به شناسایی جنبه‌ها می‌پردازد [۲۲]. تمرکز کنونی این رویکرد بر بهینه ساختن الگوریتم weaving، برای سیستم‌هایی با رفتار پیچیده می‌باشد. اما این رویکرد هنوز در دنیای واقعی مورد استفاده قرار نگرفته است. ولیکن، ابزارهای مدل‌سازی برای پشتیبانی از این رویکرد رفتاری وجود دارند.

۵-۵- رویکرد پروفایل AOSD

رویکرد پروفایل AOSD، رویکردی مستقل از زبان‌های برنامه‌نویسی جنبه‌گرا است [۲۳]. از نمودار کلاس به دست آمده در فاز تحلیل، یک ماشین حالت، استخراج می‌گردد؛ رهنمودهایی در این رویکرد برای شناسایی جنبه‌ها از ماشین حالت، وجود دارد. اگرچه جنبه‌هایی مانند همزمانی، زمانبندی، نامگذاری و اداره کردن خطا در سیستم بافر، از جمله مثال‌های اعمال این رویکرد می‌باشد، اما همچنان در صنعت مورد استفاده قرار نگرفته است. مکانیسم پروفایل UML، از ابزارها و روش‌های مورد استفاده در این رویکرد می‌باشد.

جدول ۱- مقایسه چارچوب‌های جنبه‌گرا برای زمان پیاده‌سازی

چارچوب‌های جنبه‌گرا	زبان برنامه‌نویسی مربوطه	زمان ادغام با کد پایه (Weaving)	توضیحات
AspectWerkz	Java (Open Source)	- زمان اجرا - زمان بار شدن کلاس‌ها - زمان ساخت	طراحان، AspectWerkz را یک چارچوب ساده، پویا و سبک وزن می‌دانند که به راحتی با کدهای زبان جاوا یکپارچه می‌شود [۸، ۹، ۱۷، ۴۶].
JBoss	Java (Open Source)	- زمان اجرا و از طریق Dynamic Proxy	چارچوبی است که منحصراً برای جاوا تعریف شده است و دارای مجموعه‌ای از جنبه‌های از پیش آماده می‌باشد [۸، ۱۷].
HyperJ	Java (Open Source)	- زمان بار شدن کلاس‌ها	این چارچوب از دغدغه‌های سراسری چند بُعدی پشتیبانی می‌نماید [۲، ۴۷].
AspectC#	تحت .NET (Open Source)	- زمان کامپایل	چارچوبی ساده و با زمان یادگیری پایین است. از دیدگاه تعریف قواعد معنایی و دستوری مانند زبانAspectJ می‌باشد [۸، ۴۸].
Spring	Java (Open Source)	- زمان اجرا و از طریق Dynamic Proxy	تمرکز این چارچوب بر نرم‌افزارهای تجاری است و با زبان AspectJ سازگار می‌باشد [۷، ۴۹].

جدول ۲- مقایسه چارچوب‌های جنبه‌گرا برای زمان جمع‌آوری نیازمندی‌ها

	چارچوب‌های جنبه‌گرا	توضیحات
ابزارهای نامتقارن	**Probe Framework**	توصیف نیازمندی‌ها، به صورت فایل XML و یا با استفاده از نمادهای UML، ورودی این ابزار است. تولید کننده منطق و تحلیل‌گر UML، برای تجزیه ورودی‌ها می‌باشد. نمونه‌ساز و ادغام‌کننده از جمله پیمانه‌های این ابزار به شمار می‌روند [۴].
	Arcade	در این ابزار واحدهای مربوط به تجزیه نیازمندی‌های اصلی، جنبه به شمار می‌روند [۴].
	Theme/Doc	این ابزار، نمادهای گرافیکی را برای توصیف نیازمندی‌ها فراهم می‌آورد و از ابزار طراحی theme/UML برای این منظور استفاده می‌کند. در این ابزار، نیازمندی‌هایی که دارای رفتاری مشابه هستند، شناسایی شده و در یک واحد عملکردی قرار می‌گیرند. به این ترتیب حجم نیازمندی‌ها کاهش یافته و بهتر می‌توان دغدغه‌های سراسری را از میان آن‌ها انتخاب نمود [۳، ۴، ۲۴].
	Scenario Modeling	با استفاده از الگوهای تعاملات میان نیازمندی‌ها، دغدغه‌های سراسری را به شکل یک سناریو در قالب نمودار تعاملات[۳۱] UML مدل می‌کند. این نمودار برای مدل کردن رفتار خارجی سیستم ایجاد می‌شود. نمودار انتقال حالت نیز می‌تواند برای مدل کردن رفتار داخلی سیستم تولید گردد [۴].
	AO Use cases	این ابزار گسترشی بر مدل Use case با قالب AspectJ می‌باشد. برای پیمانه‌ای کردن دغدغه‌های سراسری، مانند بررسی خطاها، بهبود کارایی، مانیتور کردن و ثبت وقایع، مناسب است [۴].
ابزارهای متقارن	**Requirements Description Language (RDL)**	با تحلیل زبان طبیعی که ورودی این ابزار می‌باشد، سعی در کشف دغدغه‌های سراسری دارد. این روش مشابه با روش زبان توصیف معماری است. اما زبان طبیعی دارای مشکلاتی مانند نامفهوم و مبهم بودن برخی از جملات است. بنابراین ورودی این ابزار می‌بایست دارای قالب خاصی باشد که این قالب می‌تواند تحت زبان XML به ابزار داده شود [۴].
	Cosmos	شمایی برای نمایش ارتباط میان دغدغه‌های سراسری می‌باشد. این ابزار دغدغه‌های سراسری را به دو دسته فیزیکی (مسائل مربوط به سخت افزار، محصولات کاری، سرویس‌ها و یا منابع) و منطقی (مانند ویژگی‌ها، عنوان‌ها، حالت‌ها، توابع و به طور کلی مسائل غیرقابل لمس) تقسیم می‌کند [۴].
	Multi-Dimensional Requirement Modelling	این ابزار گسترش یافته ابزار Arcade می‌باشد. به گونه‌ای که دغدغه‌های سراسری را از چندین بُعد مورد بررسی قرار می‌دهد [۴].

۵-۶- رویکرد Theme

رویکرد Theme که در مقالات [۹، ۲۴] به آن اشاره شده است، ابزاری را برای ایجاد نرم‌افزار به صورت جنبه‌گرا، فراهم می‌آورد. Theme/Doc ابزاری است که در فاز جمع‌آوری نیازمندی‌ها، به کمک ابزار Theme/UML در فاز طراحی، به شناسایی دغدغه‌های سراسری می‌پردازد. این دو ابزار با زبان‌های برنامه‌نویسی AspectJ، AspectWerkz، و Hyper/J سازگاری دارند. رویکرد Theme یک رویکرد شناخته شده می‌باشد و تاکنون کاربردهای زیادی از این رویکرد گزارش شده است. همزمانی[۳۲] در سیستم کتابخانه دیجیتال، ثبت وقایع در سیستم مدیریت دروس، از جمله جنبه‌هایی هستند که به واسطه این رویکرد، شناسایی و نوشته شده‌اند. هیچگونه ابزار کمکی سازگار با Theme/UML برای پشتیبانی از این رویکرد وجود ندارد.

۵-۷- رویکرد مدل معماری جنبه‌گرا

رویکرد مدل معماری جنبه‌گرا، رویکردی مستقل از بستر[۳۳] نرم‌افزاری می‌باشد [۲۵]. در این رویکرد، دغدغه‌های سراسری به وسیله نمودار بسته‌های نرم‌افزاری[۳۴] و نمودار کلاس مدل می‌شوند. لازم به ذکر است که این رویکرد، به طور خاص در فازهای تولید نرم‌افزار مورد استفاده قرار گیرد. همچنین، برخلاف دیگر رویکردهای ارائه شده تاکنون که از مدل‌سازی UML برای شناسایی دغدغه‌های سراسری بهره می‌گرفتند، رویکرد مدل معماری جنبه‌گرا، از یک زبان متامدل مبتنی بر نقش[۳۵] که توسعه‌یافته‌ای از همان مدل‌سازی UML می‌باشد، استفاده می‌کند. این رویکرد در مقایسه با رویکردهایی که تاکنون معرفی نمودیم، بیشترین کاربرد را داشته است. مسئله احراز هویت در سیستم بانکداری و کنترل انتقال پول، از جمله مثال‌های این رویکرد می‌باشند، اما تاکنون در صنعت مورد استفاده قرار نگرفته است. مجموعه‌ای از ابزارها برای کمک در مدل‌سازی این رویکرد نیز وجود دارند.

۵-۸- رویکرد مبتنی بر مهندسی نیازمندی‌ها

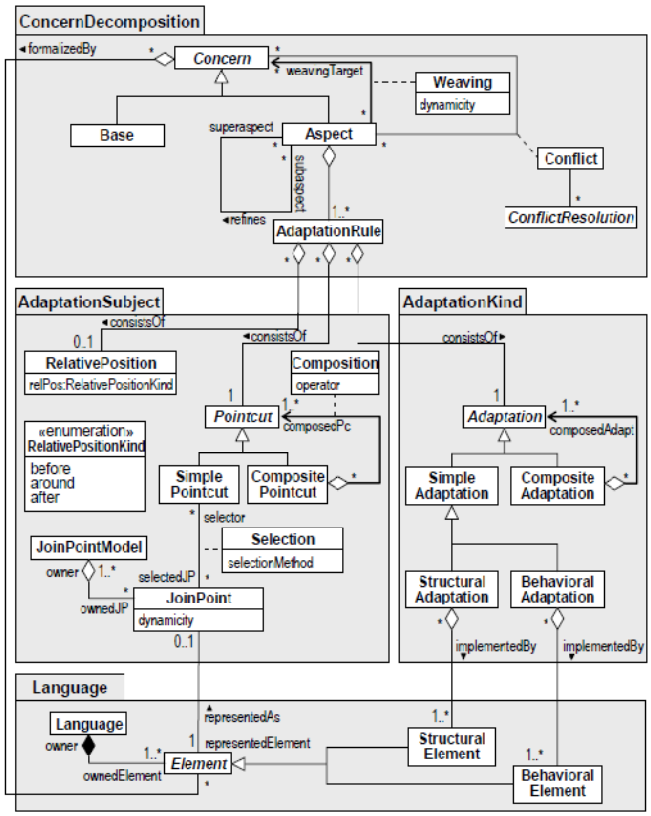
مقاله [۴]، رویکردی را مبتنی بر مهندسی نیازمندی‌ها معرفی نموده است. ایده اصلی این روش این است که دغدغه‌های سراسری از همان مراحل ابتدایی فرآیند توسعه سیستم که به جمع‌آوری نیازمندی‌ها می‌پردازد، در سیستم وجود دارند. عدم توجه به آن‌ها و همچنین به تأخیر انداختن شناسایی آن‌ها تا زمان پیاده‌سازی، می‌تواند منجر به ایجاد تناقض در میان نیازمندی‌ها شود. در این مقاله، تنها به بررسی ابزارهایی برای کمک به این ایده پرداخته و هیچ گونه مثالی را در این رابطه بیان نکرده است. نکته قابل توجهی که در این مقاله وجود دارد، دسته‌بندی ابزارها به دو دسته متقارن و نامتقارن است.

در مقاله‌ای مشابه با [۴]، ایده یافتن دغدغه‌های سراسری در فاز جمع‌آوری نیازمندی‌ها را با استفاده از روش‌های پردازش زبان طبیعی[۳۶] عملی ساخته است [۵]. در این مقاله، با استفاده از ابزار WMATRIX، از طریق روش NLP (یکی از روش‌های کاوش جنبه که در بخش ۷ معرفی خواهد شد)، به تحلیل نیازمندی‌ها در یک سیستم پردازش داده‌های اطلاعاتی یک بزرگراه پرداخته است.

۵-۹- مدل معماری مرجع

تاکنون، چندین رویکرد برای شناسایی دغدغه‌های سراسری قبل از فاز پیاده‌سازی (مثلاً، فازهای جمع‌آوری نیازمندی‌ها، تحلیل و یا طراحی) را معرفی نمودیم. در تمامی این رویکردها، تمرکز بر استفاده از مدل‌ها و نمودارهای ایجاد شده در طول این فازها بوده است. بدین ترتیب آشنایی با مدل‌ها، پیش نیاز استفاده از این رویکردهاست. برای حل این مسئله، مقاله [۲۶]، مدل معماری مرجع را برای برنامه‌نویسی جنبه‌گرا معرفی می‌کند. همانگونه که در شکل ۴ نمایش داده شده است، این مدل مستقل از هر زبان پیاده‌سازی و نمودارهای فازهای تحلیل و طراحی می‌باشد و همچنین یک دید کلی از شیوه جنبه‌گرا را در اختیار می‌گذارد. با استفاده از مدل معماری مرجع بسیاری از مفاهیم شیوه جنبه‌گرا، به طور کامل مشخص شده است. مزیت دیگر این معماری مرجع، نگاشت نمودارها و مدل‌های فازهای تحلیل و طراحی به این مدل است.

در شکل ۴، بخش Concern Decomposition در ارتباط با شکست سیستم به دغدغه‌های سراسری و ارتباط آن‌ها با یکدیگر است. عملیات ادغام جنبه‌ها با کد پایه در این بخش از معماری، می‌تواند در زمان طراحی یا در زمان اجرا باشد. بخش Adaptation Rule، در ارتباط با سازگاری جنبه‌ها با Pointcut ها است (آیا جنبه مربوطه در مکان مناسبی قرار گرفته است؟). در بخش Language نیز، زبان جنبه‌گرای مورد نظر انتخاب می‌شود، که این زبان، وابسته به اینکه در کدام فاز از فرآیند تولید به شناسایی جنبه‌ها پرداخته می‌شود، می‌تواند، یک زبان مدل‌سازی و یا یک زبان برنامه‌نویسی باشد.



شکل ۴- مدل معماری مرجع [۲۶]

۶- مسائل و مشکلات بازنویسی برنامه‌ها به صورت جنبه‌گرا

برنامه‌نویسی جنبه‌گرا مانند هر روش برنامه‌نویسی دیگری، دارای مسائل و مشکلات خاص خود می‌باشد که نیازمند تحقیقات بیشتری است. در ادامه، برخی از مسائل و تحقیقاتی که تاکنون در این زمینه انجام شده است، بررسی می‌گردد.

جدول ۳- مقایسه رویکردهای جنبه‌گرا

رویکردهای جنبه‌گرا	زبان برنامه‌نویسی پیاده‌سازی	فاز تولید برای شناسایی جنبه و نمودار مورد استفاده	زبان مدل‌سازی	مثال از اعمال رویکرد به صورت آکادمیک	مثال از کاربرد در صنعت
رویکرد مدل طراحی جنبه‌گرا	AspectJ	فاز طراحی: ID، SD، ColD، و CD	UML 1.X	زمان‌بندی در سیستم خطوط تلفن	-
رویکرد طراحی JAC	JAC	فاز طراحی: CD	UML 1.X	برنامه‌های کاربر- کارگزار	سایت دروس و مدیریت امور تجاری
رویکرد ایجاد نرم افزار جنبه‌گرا با استفاده از مدل‌سازی UML	HyperJ AspectJ	فاز جمع‌آوری نیازمندی‌ها: UCD فاز تحلیل: ComD و CD فاز طراحی: SD و CPD	UML 2	مدیریت هتل	-
رویکرد رفتاری Aspect Weaver	-	فاز تحلیل: MSC	UML 2	فرآیند ثبت وقایع	-
رویکرد پروفایل AOSD	-	فاز تحلیل: CD	UML 1.X	سیستم بافر	-
رویکرد Theme	AspectWerkz HyperJ AspectJ	فازهای جمع‌آوری نیازمندی‌ها و تحلیل: CD فاز طراحی: SD و PD	UML 1.X	همزمانی در کتابخانه دیجیتال، login در سیستم مدیریت دروس	
رویکرد مدل معماری جنبه‌گرا	-	فاز طراحی: CD و PD	UML 2, Role-Based	احراز هویت در سیستم بانکداری و کنترل انتقال پول	-
رویکردی مبنی بر مهندسی نیازمندی‌ها	-	فاز جمع‌آوری نیازمندی‌ها	سند نیازمندی‌ها (به زبان طبیعی)	خروجی ابزار جمع‌آوری اطلاعات یک بزرگراه	-
رویکرد مدل معماری مرجع	هر زبان برنامه‌نویسی	در کلیه فازهای توسعه سیستم	UML	-	-

CD: Class Diagram
ColD: Collabration Diagram
SD: Sequence Diagram
ID: Interaction Diagram
MSC: Message Sequence Chart
UCD: Use Case Diagram
ComD: Communications Diagram
CPD: Component Diagram
PD: Package Diagram

۶-۱- تأثیر دغدغه‌های سراسری بر کیفیت و تکامل نرم‌افزار

Bruntink و همکارانش [۲۷]، با استفاده از جنبه‌های مدیریت خطاها، تأثیر دغدغه‌های سراسری را بر کیفیت سیستم مورد تحلیل قرار داده‌اند. نتیجه به دست آمده از این بررسی حاکی از آن است که عدم استفاده از برنامه‌نویسی جنبه‌گرا، برنامه‌نویسی را بسیار مشکل می‌نماید و منجر به ایجاد خطاهای بسیاری می‌گردد. نویسندگان اعلام داشتند که سختی در پیاده‌سازی ممکن است از ماهیت پیچیده دغدغه‌های سراسری باشد.

Kulesza و همکارانش [۲۸]، با محاسبه مجموعه‌ای از معیارها برای دو شیوه جنبه‌گرا و شی‌گرا به مقایسه این دو روش پرداخته‌اند. نتایج به دست آمده از این مقایسه، تعداد خطوط کمتر، جداسازی بهتر مسائل، وابستگی[۳۷] ضعیف‌تر و پیچیدگی کمتر برای شیوه جنبه‌گرا می‌باشد. اما در این شیوه، تعداد مؤلفه‌ها و عملیات افزایش می‌یابد.

Gibbs و همکارانش [۲۹]، تکامل نرم‌افزار را بر روی یک سیستم، در دو حالت مختلف بررسی کرده‌اند. حالت اول، تکامل بدون استفاده از روش جنبه‌گرا بوده است و حالت دوم با روش جنبه‌گرا. نتیجه به دست آمده در تایید عملکرد بهتر حالت دوم بوده است.

Hanenberg و همکارانش [۳۰]، دو روش Aspect-Aware و Aspect-Oriented Refactoring را برای تکامل نرم‌افزارهای جنبه‌گرا معرفی نموده‌اند. در این مقاله، با استفاده از دو روش ارائه شده، به انجام دو نوع تکامل مختلف پرداخته‌اند. در نوع اول با استفاده از روش Aspect-Aware، تکامل بر روی کد پایه انجام می‌شود و به این ترتیب، مشکلی که به هنگام اصلاحات کد پایه در یک برنامه جنبه‌گرا به وجود می‌آید، بر طرف خواهد شد. در نوع دوم تکامل به جای اعمال بر کد پایه با بهره‌گیری از روش Aspect-Oriented Refactoring، بر روی کد جنبه اعمال می‌گردد. با این عمل، تمامی کدهای جنبه، پس از تغییرات، به‌هنگام خواهند شد.

Walker و همکارانش [۲]، با استفاده از اعمال برنامه‌نویسی جنبه‌گرا بر روی دو سیستم مختلف، با اهدافی مانند راحتی در اشکال‌زدایی و راحتی در تغییرات کد برنامه، به مقایسه این روش با روش‌های دیگر می‌پردازند. در این مقاله از ابزار AspectJ برای برنامه‌نویسی جنبه‌گرا استفاده نموده است. تسریع در یافتن خطاها و تصحیح آن‌ها و همچنین افزایش کارایی سیستم، در صورت استفاده از شیوه جنبه‌گرا، از نتایج به دست آمده می‌باشد.

۶-۲- پیاده‌سازی دغدغه‌های سراسری پیچیده

Bruntink و همکارانش [۳۱]، مجموعه‌ای از قالب‌های کد آماده و ساده را برای پیاده‌سازی دغدغه‌های سراسری ارائه می‌دهند. برنامه‌نویسان این امکان را نیز دارند تا این کدها را برای سیستم خود سفارشی[۳۸] کنند.

Marin و همکارانش [۳۲]، با استفاده از دسته‌بندی[۳۹]، شیوه‌ای را برای پیاده‌سازی دغدغه‌های سراسری ارائه می‌نمایند.

۶-۳- استخراج دغدغه‌های سراسری و اطمینان از صحت کد استخراج‌شده برای جنبه

Monteiro و Fernandes [۳۳] و همچنین Hanenberg [۳۰] و همکارانش، در ارتباط با انتقال کد Advice، مقالاتی را ارائه داده‌اند. اما هیچ یک از این دو روش، به صورت خودکار، قابل اجرا نمی‌باشند. در روش Hanenberg و همکارانش، قبل از انتقال کد[۴۰]، متغیرهای محلی، برای مشخص شدن ارجاعات دیگر بخش‌های کد به آن‌ها، مورد بررسی قرار می‌گیرند و در صورت عدم وجود ارجاعات، انتقال کد انجام می‌شود. اما در روش Monteiro و Fernandes با اعمال ساده‌سازی[۴۱] و روش متد جایگزین، سعی بر حل مشکلات متغیرهای محلی دارد. تعیین Join point، Pointcut و Advice مناسب نیز، با استفاده از همین شیوه می‌تواند انجام گیرد. ساده‌سازی ارائه شده می‌تواند بر تکامل نرم‌افزارهای جنبه‌گرا نیز موثر باشد.

Binkley و همکارانش [۳۴]، روشی خودکار را برای استخراج کد و تعیین Join point و Advice مناسب، ارائه کرده‌اند (مانند آنچه در روش Monteiro و Fernandes وجود داشت). اما این روش هنوز به صورت عملی مورد استفاده قرار نگرفته است. در روش ارائه شده، هفت نوع استخراج کد به همراه تولید Pointcut مناسب آن‌ها، توصیف گردیده است.

Ettinger و Verbaere [۳۵]، روشی خودکار را بر مبنای روش قطعه‌بندی[۴۲]، ارائه کرده‌اند. بر اساس این ایده، یک قطعه، شامل تمامی کد دغدغه سراسری می‌باشد. بنابراین می‌توان از روش‌های استخراج قطعه استفاده نمود.

Braem و همکارانش [۳۶]، با استفاده از روش‌های یادگیری ماشین، به کشف الگوهایی برای تولید Pointcut از مجموعه‌ای از Join pointها می‌پردازند.

Acher و همکارانش [۳۷]، استخراج دغدغه‌های سراسری را در خط تولید نرم‌افزار[۴۳] و با استفاده از مدل ویژگی‌ها[۴۴] مورد بررسی قرار می‌دهند. در واقع، این روش، دغدغه‌های سراسری را از مدل ویژگی‌ها به دست می‌آورد.

۷- کاوش جنبه

همانطور که در بخش ۲-۲ اشاره شد، سه فاز اصلی در برنامه‌نویسی جنبه‌گرا وجود دارد که اولین فاز آن کشف جنبه‌ها می‌باشد. فرآیند شناسایی دغدغه‌های سراسری و پیاده‌سازی آن‌ها به صورت کد جنبه، فرآیند کاوش جنبه نامیده می‌شود [۷]. در واقع، هدف از کاوش جنبه را می‌توان بهبود قابلیت درک سیستم و همچنین بهبود توانایی سیستم در مهاجرت به سمت برنامه‌نویسی جنبه‌گرا دانست [۳۸].

۷-۱- روش‌های کاوش جنبه

روش‌های کاوش جنبه را می‌توان در سه دسته ایستا[۴۵]، پویا[۴۶] و ترکیبی[۴۷] تقسیم‌بندی نمود. در روش‌های ایستا، فرآیند کاوش جنبه مستقیماً بر روی کد برنامه انجام می‌گیرد. این در حالیست که روش‌های پویا، از طریق اجرای برنامه‌ها و با استفاده از اطلاعات به دست آمده از زمان اجرا، به شناسایی جنبه‌ها می‌پردازند. دسته سوم، روش‌های ترکیبی هستند. همانطور که از نام این روش‌ها مشخص می‌باشد، راهکار ارائه شده در این روش‌ها، ترکیب روش‌های ایستا و پویا است. در این بخش، به بررسی روش‌های موجود در هر سه دسته می‌پردازیم. در جدول ۴، مقایسه‌ای میان روش‌های کاوش جنبه انجام گرفته است. همچنین، جدول ۵، فهرستی از ابزارهای معروف کاوش جنبه که در صنعت و یا محیط‌های آکادمیک مورد استفاده قرار گرفته‌اند را به همراه برخی از ویژگی‌های آن‌ها ارائه می‌دهد.

۷-۱-۱- روش‌های ایستا

تحلیل تعداد فراخوانی‌های ورودی[۴۸]: ایده اصلی این روش بر تعداد دفعاتی می‌باشد که یک متد فراخوانی می‌شود. به این معنا که اگر تعداد فراخوانی‌های یک متد زیاد باشد، آن متد می‌تواند به عنوان یک کاندید برای تبدیل شدن به جنبه در نظر گرفته شود [۳۸، ۳۹]. مسائل ثبت وقایع، ردیابی، بررسی پیش‌شرایط و پس‌شرایط، و مدیریت خطاها از جمله دغدغه‌های سراسری هستند که با استفاده از این روش، در کد برنامه‌ها شناسایی شده‌اند [۷]. این روش شامل سه مرحله اصلی زیر می‌باشد [۳۹]:

- برای کلیه متدهای برنامه، به صورت خودکار، تعداد دفعاتی که هر کدام از متدها در آن برنامه فراخوانی شده‌اند، محاسبه می‌گردد.
- متدهایی که تعداد فراخوانی‌های آن‌ها از یک حد آستانه[۴۹] کمتر است و همچنین، متدهای getter، setter، و نیز متدهای کمکی مانند ()toString، حذف می‌شوند.
- متدها در قالب فراخوانی کننده، فراخوانی شده، قوانین نامگذاری[۵۰] و نیز پیاده‌سازی آن متدها، به منظور شناسایی جنبه‌ها به صورت دستی، تحلیل می‌گردند.

استفاده از کلونی‌های برنامه[۵۱]: کلونی‌های یک برنامه، قطعه کدهایی از برنامه می‌باشند که بر اساس یک معیار شباهت (مثلاً، شباهت متنی، شباهت معنایی، و یا شباهت در دستورات برنامه‌نویسی)، با یکدیگر مشابه می‌باشند [۶]. بنابراین، کلونی‌های یک برنامه می‌توانند کاندیدای مناسبی برای دغدغه‌های سراسری یا جنبه‌ها باشند [۴۰]. در این روش، آن دسته از کلونی‌های برنامه که تنیدگی بیشتری در کد برنامه دارند، دارای امتیاز بیشتری برای کاندیدا شدن هستند. برای یافتن کلونی‌های برنامه می‌توان از ابزارهایی مانند CCFinder استفاده نمود [۷، ۱۲].

تحلیل شناسه[۵۲]: در برنامه‌هایی که از قوانین نامگذاری مشخص برای نوشتن کد استفاده می‌نمایند، کاندیدای جنبه می‌تواند با شناسایی گروه‌هایی با نام‌های مشابه انتخاب گردند [۴۱]. به عنوان مثال، شناسه createUndoActivity به سه بخش create، undo و activity تقسیم می‌شود. سپس با استفاده از الگوریتمی، لغاتی که ریشه مشابه دارند در یک گروه قرار می‌گیرند (مانند undoable و undo). در نهایت ، گروه‌هایی را که دارای متدها و کلاس‌های بیشتری هستند به عنوان کاندیدای جنبه انتخاب می‌شوند [۷، ۱۲، ۳۸، ۴۱].

پردازش زبان طبیعی[۵۳] (NLP): این روش، همانند روش تحلیل شناسه، از قوانین نامگذاری برای یافتن دغدغه‌های سراسری استفاده می‌نماید. یکی از روش‌های مورد استفاده در NLP، روش زنجیره لغات است که کلمات مشابه از دیدگاه معنایی را شناسایی می‌کند. این روش مشابهت، با معیار اندازه‌گیری فاصله تعیین می‌شود. الگوریتم زنجیره لغات بر روی توضیحات[۵۴]، نام متدها، فیلدها و نیز کلاس‌ها اعمال می‌گردد. خروجی این الگوریتم، لیستی از زنجیره لغات است که از دید معنایی بسیار به هم شباهت دارند. استفاده کننده این روش می‌بایست به صورت دستی از میان زنجیره لغات، کاندیداهای جنبه را انتخاب نماید [۷، ۱۲].

شناسایی متدهای یکتا[۵۵] : تعریف متد یکتا عبارت است از "متدی که مقدار بازگشتی[۵۶] نداشته، تنها یک پیام را پیاده‌سازی می‌کند، و توسط هیچ متد دیگری پیاده‌سازی نمی‌شود". در این روش، پس از شناسایی متدهای یکتا، آن‌ها بر اساس تعداد فراخوانی مرتب می‌شوند. در این مرتب‌سازی، متدهای نامربوط، مانند متدهای دسترسی، حذف می‌گردند. کسی که از این روش استفاده می‌کند، بایستی

که به صورت دستی از میان متدهای باقیمانده، جنبه‌های کاندید را انتخاب نماید [۷، ۱۲].

خوشه‌بندی[۵۷] متدهای مرتبط: روش‌های مختلفی مبتنی بر استفاده از الگوریتم‌های خوشه‌بندی، برای کاوش جنبه‌ها وجود دارد [۷، ۱۲، ۴۲]. در روش‌های خوشه‌بندی، متدهای مشابه بر اساس یک معیار شباهت، در خوشه‌های جداگانه تقسیم‌بندی می‌شوند. در روش اول [۴۲]، این خوشه‌ها بر مبنای معیار کمترین فاصله میان خوشه‌ای، با یکدیگر ادغام می‌گردند. در این روش، متدهایی که در هر بار ادغام شرکت داشته‌اند، کاندیدای تبدیل شدن به جنبه هستند. در روش دوم [۱۲]، یک متد، زمانی کاندیدای تبدیل شدن به یک جنبه خواهد بود که در هنگام خوشه‌بندی، تمایل به قرارگیری در چندین خوشه را داشته باشد. یکی دیگر از روش‌های خوشه‌بندی، روش CBFA[۵۸] می‌باشد که با ترکیب خوشه‌بندی و روش تحلیل تعداد فراخوانی‌های ورودی، به یافتن جنبه‌ها می‌پردازد [۴۳، ۴۴]. به این ترتیب که پس از یافتن تعداد فراخوانی‌های ورودی متدها، آن‌هایی را که دارای اسامی مشابه هستند، در یک خوشه قرار می‌دهد. در نهایت، خوشه‌ای که مجموع تعداد فراخوانی‌های ورودی آن‌ها بالاتر است، به عنوان کاندیدای جنبه برگزیده می‌شود [۴۳، ۴۴].

COMMIT: این روش، مبتنی بر تحلیل تاریخچه کد در سیستم‌های کنترل نسخه[۵۹] مانند CVS و یا SVN می‌باشد. به این معنا که از کد اصلی برنامه، تمامی تغییراتی را که در یک بازه زمانی خاص رخ داده، استخراج می‌گردد. سپس آن متدها و یا متغیرهایی که در آن بازه زمانی خاص، همزمان با یکدیگر تغییر کرده‌اند را به دست می‌آورد. بر این اساس، گراف وابستگی میان متدها و متغیرها، رسم می‌شود. این روش، با استفاده از یک فرمول آماری، درجه وابستگی را نیز در گراف تعیین می‌کند. آن دسته از متغیرها و متدهایی که نسبت به یکدیگر وابستگی بیشتری دارند، به عنوان کاندیداهای جنبه‌ها شناسایی می‌شوند [۴۴].

۷-۱-۲- روش‌های پویا

DynAMiT: در این روش، ردیابی‌های[۶۰] برنامه که رفتار زمان اجرای سیستم را نشان می‌دهند، تحلیل می‌شوند. برای تحلیل این ردیابی‌ها، ارتباط میان اجرای فراخوانی متدها، به دست می‌آید. چهار نوع ارتباط در این روش تعریف می‌شود. ارتباط outside-before (متد۱، قبل از متد۲، فراخوانی می‌شود)، ارتباط outside-after (متد۲، بعد از متد۱، فراخوانی می‌شود)، ارتباط inside-first (متد۱، اولین متد فراخوانی شده در متد ۲ است)، ارتباط inside-last (متد۱، آخرین متد فراخوانی شده در متد۲ است). اگر یکی از این ارتباطات، بیش از یک‌بار وجود داشته باشد (به عنوان مثال، متد۲ همواره بعد از متد۱ فراخوانی شود)، آن ارتباط، به عنوان کاندیدای جنبه خواهد بود [۱۲].

تحلیل رسمی مفاهیم[۶۱] (FCA): در این روش، سیستم بر اساس موارد کاربرد و وظیفه‌مندی‌های[۶۲] آن اجرا می‌گردد [۱۳]. خروجی این اجرا، مجموعه‌ای از ردیابی‌های اجرایی برنامه است. این ردیابی‌ها توسط الگوریتم FCA تحلیل می‌شوند. در این تحلیل، موارد کاربرد به عنوان اشیای این الگوریتم و متدهای فراخوانی شده در زمان اجرا به عنوان ویژگی‌ها در نظر گرفته می‌شوند. به این ترتیب، تمامی مفاهیمی که شامل ردیابی‌های یک مورد کاربرد مشترک هستند، به عنوان کاندیدای جنبه انتخاب خواهند شد [۷، ۱۲، ۱۳، ۳۸].

۷-۱-۳- روش‌های ترکیبی

مشکل روش‌های ایستا آن است که نتایج خود را مستقیماً از کد برنامه‌ها استخراج می‌کنند، بدون آنکه آن برنامه‌ها را اجرا کنند. ولیکن، یک سری از ویژگی‌های برنامه‌ها به صورت ایستا قابل استنتاج و تصمیم‌گیری[۶۳] نمی‌باشند و فقط در زمان اجرا خود را بروز می‌دهند. از سوی دیگر، روش‌های پویا، بسیار وابسته به ورودی‌ها و موارد کاربردی می‌باشند که برای اجرای آن برنامه مورد استفاده قرار گرفته‌اند. بنابراین، لزوما تمام ویژگی‌ها و وظیفه‌مندی‌های سیستم اجرا نگردیده‌اند. هدف از ترکیب روش‌های ایستا و پویا و ارائه روش‌های ترکیبی این است که معایب هر دو روش را پوشش دهد. اولین روش ارائه شده در تکنیک پویا، می‌تواند یک روش ترکیبی به شمار آید.

روش DynAMiT که پیش از این معرفی نمودیم، می‌تواند به عنوان یک مثال از روش‌های ترکیبی نیز در نظر گرفته شود. زیرا تحلیل ردیابی‌های به دست آمده با استفاده از اطلاعات ایستا در کد مانند فراخوانی متدها، انجام می‌شود.

در مقاله [۳۸]، به مقایسه سه تکنیک تحلیل تعداد فراخوانی‌های ورودی، تحلیل شناسه و FCA با اعمال هر سه روش بر روی یک برنامه رسم اشکال به نام JHotDraw می‌پردازد. در نتایج به دست آمده، دو روش تحلیل تعداد فراخوانی‌های ورودی و FCA، به گونه‌ای مکمل یکدیگر می‌باشند. اما روش تحلیل شناسه، تا حدی از این دو روش متمایز است. در این مقاله، از نتایج به دست آمده برای ترکیب دو روش FCA و تحلیل تعداد فراخوانی‌های ورودی استفاده کرده است. در این روش ترکیبی، می‌توان متدها با تعداد فراخوانی‌های ورودی زیاد (روش تحلیل تعداد فراخوانی‌های ورودی) که عملکردهای مشابه دارند (روش FCA) را، به عنوان کاندیدای جنبه انتخاب نمود.

در مقاله [۴۵]، یک روش برای ارزیابی روش‌های کاوش جنبه، معرفی کرده است. سپس، با استفاده از روش ارزیابی ارائه شده، به بررسی کیفیت خروجی این روش‌ها و همچنین به بررسی صحت جداسازی جنبه‌ها در روش‌های کاوش جنبه، پرداخته است. روش ارائه شده، بر اساس مدل‌های رسمی می‌باشد که دارای فرضیاتی برای معیار اندازه‌گیری ارزیابی است. طراحان این روش، آن را بر روی تمامی روش‌های کاوش جنبه معرفی شده در بالا، اعمال کرده‌اند.

۸- نتیجه‌گیری

مسائل مهمی در تصمیمات طراحی وجود دارند که پیاده‌سازی آن‌ها با زبان‌های برنامه‌نویسی موجود، منجر به پراکندگی و در هم تنیده شدن این مسائل در کد برنامه شده و در نتیجه بر کاهش پیمانه‌ای شدن سیستم اثر خواهند گذاشت. برنامه‌نویسی جنبه‌گرا با هدف برطرف ساختن این مشکل و افزایش پیمانه‌ای بودن سیستم، در کنار برنامه‌نویسی شیءگرا قرار گرفته است. علاوه بر مزیت بیان شده که منجر به کاهش پیچیدگی کد سیستم خواهد شد، اعمال آسان‌تر تغییرات به هنگام تکامل نرم‌افزار، از دیگر ویژگی‌های استفاده از این شیوه می‌باشد. اما این شیوه مانند هر روش دیگری در کنار مزایای خود، دارای معایبی نیز می‌باشد. طولانی‌تر شدن زمان کامپایل و ناسازگاری ابزارهای جنبه‌گرا با برخی از زبان‌های برنامه‌نویسی از جمله معایب اصلی این روش به شمار می‌روند. همچنین، برای مهاجرت از شیوه شیءگرا به جنبه‌گرا لازم است تا به بسیاری از برنامه‌نویسان، آموزش‌هایی در زمینه روش برنامه‌نویسی جنبه‌گرا داده شود.

ابزارها و چارچوب‌های بسیاری برای پشتیبانی از شیوه جنبه‌گرا در مراحل شناسایی نیازمندی‌ها و نیز پیاده‌سازی وجود دارند. در این مقاله تلاش نمودیم تا برخی از مهم‌ترین این ابزارها، دسته‌بندی شده و ویژگی‌های آن‌ها نیز بیان گردد. موضوع مهمی که در برنامه‌نویسی جنبه‌گرا وجود دارد، تعیین جنبه‌های سیستم در فازهای مختلف فرآیند ایجاد نرم‌افزار است، برخی بر این باورند که تعیین جنبه‌های سیستم منحصر به زمان پیاده‌سازی آن نیست و در همان فازهای ابتدایی نیز می‌توان جنبه‌های سیستم را مشخص نمود. بر این اساس، رویکردهای متفاوتی را می‌توان داشت که در این مقاله به آن‌ها اشاره شد و در جدولی با یکدیگر مورد

مقایسه قرار گرفتند. اما مرحله مهمی که در شیوه جنبه‌گرا وجود دارد، کاوش جنبه است، که در این مقاله روش‌های مطرح شده در این زمینه و همچنین ابزارهای موجود، به صورتی جداگانه مورد بررسی قرار گرفتند.

همانطور که در بخش ۶ اشاره شد، هنوز مسائل مبهم بسیاری در زمینه برنامه‌نویسی جنبه‌گرا وجود دارند که نیازمند تحقیقات بیشتر می‌باشند. حفظ رفتار سیستمی که به شیوه جنبه‌گرا تولید می‌گردد، از جمله مسائل مهمی است که با روش‌های متفاوت آزمون نرم‌افزار قابل بررسی می‌باشد. اما آزمون نرم‌افزارهای جنبه‌گرا به دلیل وجود مفهومی به نام جنبه، در مقایسه با آزمون نرم‌افزارهای غیرجنبه‌گرا دارای چالش‌هایی است. تمرکز آتی ارزیابی جنبه‌گرا، بر روی آزمون نرم‌افزارهای جنبه‌گرا برای اطمینان از صحت عملکرد آن‌ها خواهد بود.

جدول ۴- مقایسه روش‌های کاوش جنبه

روش‌های کاوش جنبه	نوع تکنیک	پیش شرایط	برنامه استفاده شده در مطالعه موردی	معایب
تحلیل گنجایش ورودی	ایستا	حداقل یکی از متدها باید دارای تعداد فراخوانی‌های ورودی بالا باشد.	TomCat 5.5 API & JHotDraw	۱- وابسته به تعیین حد آستانه برای تعداد فراخوانی‌های ورودی است. ۲- متدهای کوچک و یا به عبارتی متدهای دارای تعداد فراخوانی‌های ورودی پایین حذف می‌شوند.
کلونی‌های برنامه	ایستا	برنامه باید دارای کلونی باشد.	TomCat & ASML C-Code	۱- اگر ابزاری که برای یافتن کلونی‌های برنامه استفاده می‌شود، نتواند به خوبی عمل نماید، در شناسایی نادرست دغدغه‌های سراسری بی‌تأثیر نخواهد بود.
تحلیل شناسه	ایستا	قوانین نامگذاری در برنامه‌نویسی باید رعایت شده باشد.	JHotDraw	۱- اطلاعات جمع‌آوری شده از کدهای برنامه در برخی مواقع شامل نویز است. ۲- در برخی مواقع، مفاهیم به دست آمده تمامی جنبه‌ها را پوشش نمی‌دهند.
پردازش زبان طبیعی	ایستا	برنامه باید دارای اسامی مترادف باشد.	PetStore	۱- اگر برنامه دارای اسامی مترادف نباشد، تجزیه برنامه با استفاده از این تکنیک، با شکست مواجه خواهد شد.
شناسایی متدهای یکتا	ایستا	وجود حداقل دو متد به گونه‌ای که یکی، مسئله Cross Cut باشد و دیگری نباشد.	Smalltalk image	۱- این تکنیک وابستگی زیادی به نحوه برنامه‌نویسی دارد. به گونه‌ای که در برخی مواقع ممکن است، برنامه‌نویس جنبه‌ها را به صورت متدهای یکتا ننوشته باشد.
خوشه‌بندی متدهای مربوطه	ایستا	برنامه باید دارای متدهایی با اسامی مترادف باشد.	JHotDraw & Banking example	۱- محاسبات طولانی و نیاز به داشتن دانش در زمینه روش‌های مختلف داده‌کاوی، علاوه بر نیاز به دانش برنامه‌نویسی (که در تمام روش‌ها نیاز است)، از مشکلات این روش می‌باشد.
COMMIT	ایستا	کد منبع برنامه در دسترس باشد.	PostgreSQL & NetBSD	۱- انتخاب نامناسب بازه زمانی برای استخراج تغییرات، می‌تواند در یافتن جنبه‌ها تأثیرگذار باشد.
تحلیل رسمی مفاهیم	پویا	وجود حداقل دو مورد کاربرد به گونه‌ای که یکی، مسئله Cross Cut باشد و دیگری نباشد.	JHotDraw	۱- جنبه‌هایی که در تمامی اجراهای برنامه شرکت داشته‌اند، شناسایی نمی‌شوند. ۲- کدهایی که قابل اجرا نیستند (در چارچوب برنامه)، در نظر گرفته نمی‌شوند.
DynAMiT	پویا/ ترکیبی	ترتیب فراخوانی‌ها باید مشخص باشد.	Graffiti	۱- ترتیب فراخوانی متدهای برنامه باید مشخص باشد.
تحلیل گنجایش ورودی- FCA	ترکیبی	وجود حداقل دو مورد کاربرد به گونه‌ای که یکی، مسئله Cross Cut باشد و دیگری نباشد و حداقل یکی از متدها باید دارای تعداد فراخوانی‌های ورودی بالا باشد.	JHotDraw	۱- این روش بیشتر در حد تئوری است و هنوز در عمل، خیلی مورد استفاده قرار نگرفته است.

جدول ۵- ابزارهای کاوش جنبه

ابزارهای کاوش جنبه	ویژگی‌ها
Aspect Mining Tool (AMT)	تحلیل‌های انجام شده برای یافتن جنبه بر اساس دو نوع متن و نوع و بر روی کد برنامه انجام می‌شوند. این ابزار می‌تواند با تحلیل‌های دیگری نیز مانند مبتنی بر امضا[۶۴] کار کند. هر یک از این تحلیل‌ها به عنوان یک پرس‌وجو عمل کرده و خروجی به صورت گرافیکی نمایش داده می‌شود [۷، ۵۰، ۵۱].
Aspect Browser	یک Eclipse Plugin می‌باشد. ورودی این ابزار می‌تواند به صورت عبارات منظم باشد، که با تحلیل لغوی این ورودی، خروجی به صورت گرافیکی نمایش داده می‌شود [۷، ۵۲].
Prism	یک Eclipse Plugin می‌باشد. ورودی این ابزار می‌تواند مجموعه‌ای از الگوهای جستجو باشد. ابزار با استفاده از این الگوها به یافتن جنبه در کد برنامه می‌پردازد و خروجی را نمایش می‌دهد. در حال حاضر این ابزار جای خود را به Prism Query Language (PQL) که شباهتی با زبان SQL دارد، داده است [۵۳].
JQuery	یک Eclipse Plugin می‌باشد. ورودی این ابزار می‌تواند به صورت عبارات منظم باشد [۵۱]. این ابزار همچنین می‌تواند به صورت سلسله مراتبی، کد برنامه را پیمایش نماید[۷].
DynAMiT	ابزاری است که در روش کاوش جنبه *DynAMiT* کاربرد دارد [۱۲].
Dynamo & ToscanaJ	ابزارهایی هستند که در روش کاوش جنبه *FCA* کاربرد دارند [۱۳].

مراجع

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin, "Aspect-oriented Programming," *Proc, IEEE Int'l Conf. Object-oriented Programming*, pp. 220-242, 1997.

[2] R. J. Walker, E. Baniassad, and G. Murphy, "An Initial Assessment of Aspect-oriented Programming," *Proc, IEEE Int'l Conf. Software Engineering*, pp. 16-22, 1999.

[3] A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, M. Wimmer, and G. Kappel, *A Survey on Aspect-oriented Modeling Approaches*, Technical Report, Vienna University of Technology, Wien, Austria, 2007.

[4] S. Vijay, and A. Shetty, "A Study on Different Approaches Towards Aspect-oriented Requirements Engineering," *Indian Journal of Computer Science and Engineering*, vol. 2, no. 4, pp. 407-419, 2011.

[5] A. Sampaio, N. Loughran, A. Rashid, and P. Rayson, "Mining Aspects in Requirements," *Proc, IEEE Int'l Workshop on Aspect-oriented Requirements Engineering and Architecture Design*, pp. 62-66, 2005.

[6] T. Mens, and S. Demeyer, "Software Evolution," *Proc, IEEE Int'l Conf. Software Engineering*, pp. 116-122, 2008.

[7] M. Ceccato, "Migrating Object-oriented Code to Aspect-oriented Programming," *Proc, IEEE Int'l Conf. Software Maintenance*, pp. 497-498, 2007.

[8] N. Bhatnagar, "A Survey of Aspect-oriented Programming Languages," http://classes.soe.ucsc.edu/cmps203/Fall04/finalreports/NeerjaPaperAOP.pdf, June 2004.

[9] K. Govindraj, S. Narayanan, B. Thomas, P. Nair, and S. Peeru, "On Using AOP for Application Performance Management," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 206-212, 2006.

[10] U. Hohenstein, "Improving the Performance of Database Applications with Aspect-oriented Programming," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 12-18, 2006.

[11] A. Duck, "Implementation of AOP in Non-academic Projects," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 68-78, 2006.

[12] A. Kellens, K. Mens, and P. Tonella, "A Survey of Automated Code-level Aspect Mining Techniques," *IEEE Trans. Aspect-oriented Software Development*, vol. 4, no. 6, pp. 143-162, 2007.

[13] P. Tonella, and M. Ceccato, "Aspect Mining through the Formal Concept Analysis of Execution Traces," *Proc, IEEE Int'l Conf. Reverse Engineering*, pp. 112-121, 2004.

[14] O. Spinczyk, A. Gal, and W. Schröder-Preikschat, "Aspectc++: An Aspect-oriented Extension to the C++ Programming Language," *Proc, IEEE Int'l Conf. Tools Pacific: Objects for Internet, Mobile and Embedded Applications*, pp. 53-60, 2002.

Pace," *Proc, IEEE Int'l Conf. Object-oriented Programming*, pp. 241-261, 2005.

[30] S. Hanenberg, C. Oberschulte, and R. Unland, "Refactoring of Aspect-oriented Software," *Proc, IEEE Int'l Conf. Object-oriented and Internet-based Technologies, Concepts, and Applications for a Networked World*, pp.19-35, 2003.

[31] M. Bruntink, A. Van Deursen, and T. Tourwé, "Discovering Faults in Idiom-based Exception Handling," *Proc, IEEE Int'l Conf. Software Engineering*, pp. 242-251, 2006.

[32] M. Marin, L. Moonen, and A. Van Deursen, "A Common Framework for Aspect Mining based on Cross Cutting Cencern Sorts," *Proc, IEEE Int'l Conf. Reverse Engineering*, pp. 29-38, 2006.

[33] M. P. Monteiro, and J. M. Fernandes, "Object-to-Aspect Refactorings for Feature Extraction," *Proc, IEEE Int'l Conf. Software Engineering*, pp. 31-37, 2004.

[34] D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Automated Refactoring of Object-oriented Code into Aspects," *Proc, IEEE Int'l Conf. Software Maintenance*, pp. 27-36, 2005.

[35] R. Ettinger, and M. Verbaere, "Untangling: A Slice Extraction Refactoring," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 93-101, 2004.

[36] M. Braem, K. Gybels, A. Kellens, and W. Vanderperren, "Automated Pattern-based Pointcut Generation," *Proc, IEEE Int'l Symp. Software Composition*, pp. 42-46, 2006.

[37] M. Acher, P. Collet, P. Lahire, and R. France, "Separation of Concerns in Feature Modeling: Support and Applications," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 1-12, 2012.

[38] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourw´E, "A Qualitative Comparison of Three Aspect Mining Techniques," *Proc, IEEE Int'l Workshop on Program Comprehension*, pp. 13-22, 2005.

[39] M. Marin, A. Van Deursen, and L. Moonen, "Identifying Aspects Using Fan-in Analysis," *Proc, IEEE Int'l Conf. Reverse Engineering*, pp. 8-12, 2004.

[40] M. Bruntink, A. Van Deursen, R. Van Engelen, and T. Tourw´E, "An Evaluation of Clone Detection Techniques for Identifying Cross-cutting Concerns," *Proc, IEEE Int'l Conf. Software Maintenance*, pp. 200-209, 2004.

[41] T. Tourwe, and K. Mens, "Mining Aspectual Views Using Formal Concept Analysis," *Proc, IEEE Int'l Workshop on Source Code Analysis and Manipulation*, pp. 97-106, 2004.

[42] F. Klafke, and S. Vergilio, "A Clustering based Approach for Aspect Mining and Pointcut Identification,"

[15] M. Urban, and O. Spinczyk, "Aspect C++ Language Reference," http://www.Aspectc.Org, May 2011.

[16] D. LaErty, and V. Cahill, "Language-independent Aspect-oriented Programming," *ACM Trans. Object-oriented Programing, Systems, Languages and Applications*, vol. 38, no. 11, pp. 1-12, 2003.

[17] M. Kerste, "Aop Tools Comparison," http://Www.Ibm.Com/Developerworks/Java/Library/J-Aopwork1/, September 2005.

[18] E. Bodden, "Aspect-oriented Approaches Targeting the .Net Framework," http://Www.Bodden.De/Tools/Aop-Dot-Net, June 2009.

[19] D. Stein, S. Hanenberg, and R. Unland. "An Uml-based Aspect-oriented Design Notation," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 106-112, 2002.

[20] R. Pawlak, L. Seinturier, L. Duchien, L. Martelli, F. Legon, and G. Florin, "Aspect-oriented Software Development with Java Aspect Components," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 343-369, 2005.

[21] I. Jacobson, and P. Wei, *Aspect-oriented Software Development with Use-cases*, Addison-Wesley, 2005.

[22] J. Klein, L. H´Elou¨Et, and J. J´Ez´Equel, "Semantic-based Weaving of Scenarios," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 27-38, 2006.

[23] O. Aldawud, T. Elrad, and A. Bader, "UML Profile for Aspect-oriented Software Development," *Proc, IEEE Int'l Workshop on Aspect-oriented Modeling*, pp. 45-49, 2003.

[24] S. Clarke, and E. Banaissad, *Aspect-oriented Analysis and Design the Theme Approach*, Addison-Wesley, 2005.

[25] R. France, I. Ray, G. Georg, and S. Ghosh, "Aspect-oriented Approach to Early Design Modeling," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp.173-185, 2004.

[26] A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, and M. Wimmer, "Towards A Common Reference Architecture for Aspect-oriented Modeling," *Proc, IEEE Int'l Workshop on Aspect-oriented Programming*, pp. 23-27, 2006.

[27] M. Bruntink, A. Van Deursen, and T. Tourwé, "Discovering Faults in Idiom-based Exception Handling," *Proc, IEEE Int'l Conf. Software Engineering*, pp. 242-251, 2006.

[28] U. Kulesza, C. Santanna, A. Garcia, R. Coelho, A. Von Staa, and C. Lucena, "Quantifying the Effects of Aspect-oriented Programming: A Maintenance Study," *Proc, IEEE Int'l Conf. Software Maintenance*, pp. 223-233, 2006.

[29] C. Gibbs, C. R. Liu, and Y. Coady, "Sustainable System Infrastructure and Big Bang Evolution: Can Aspects Keep

http://Www2.Dc.Ufscar.Br/~Lawasp/2012/Artigos/03.Pdf, May 2011.

[43] D. Zhang, Y. Guo, and X. Chen, "Automated Aspect Recommendation through Clustering-based Fan-in Analysis," *Proc, IEEE/ACM Int'l Conf. Automated Software Engineering*, pp. 278-287, 2008.

[44] B. Adams, Z. Ming Jiang, and A. E. Hassan, "Identifying Cross-cutting Concerns Using Historical Code Changes," *Proc, IEEE/ACM Int'l Conf. Software Engineering*, pp. 305-314, 2010.

[45] G. Czibula, G. Cojocar, and I. Czibula, "Evaluation Measures for Partitioning-based Aspect Mining Techniques," *Journal of Computers, Communications and Control*, vol. 6, no. 1, pp. 72-80, 2011.

[46] J. Bonér, "Aspectwerkz: Plain Java AOP," *Proc, IEEE Int'l Conf. Aspect-oriented Software Development*, pp. 20-26, 2012.

[47] Multi-Dimensional Separation of Concerns for Java, http://Www.Research.Ibm.Com/Hyperspace/Hyperj/Hyperj.Htm, May 2001.

[48] H. Kim, *AspectC#: An Aosd Implementation for C#*, Master Thesis, Trinity College Dublin, Dublin, Ireland, 2002.

[49] Spring Framework, http://www.Springframework.Org, July 2012.

[50] J. Hannemann, and G. Kiczales, "Overcoming the Prevalent Decomposition of Legacy Code," *Proc, IEEE Int'l Workshop on Advanced Separation of Concerns*, pp. 167-171, 2001.

[51] A. Deursen, M. Marin, and L. Moonen, "Aspect Mining and Refactoring," *Proc, IEEE Int'l Workshop on Refactoring: Achievements, Challenges, Effects*, pp. 11-21, 2003.

[52] W. G. Griswold, Y. Kato, and J. J. Yuan, *AspectBrowser: Tool Support for Managing Dispersed Aspects*, Technical Report, University of California, San Diego, USA, 1999.

[53] C. Zhang, and H. A. Jacobsen, "Prism is Research in Aspect Mining," *Proc, ACM Int'l Conf. Object-oriented Programming, Systems, Languages and Applications*, pp. 20-21, 2004.

الهه حبیبی مدرک کارشناسی ارشد و کارشناسی خود را در رشته مهندسی نرم‌افزار به ترتیب در دانشگاه صنعتی شریف در سال ۱۳۹۲ و دانشگاه الزهرا (س) در سال ۱۳۸۹ دریافت نموده است. پس از اتمام دوره کارشناسی به مدت ۱ سال در شرکت همکاران سیستم در زمینه آزمون عملکردی نرم‌افزار به فعالیت پرداخته است. مهندس الهه حبیبی در حال حاضر در پژوهشگاه نیرو با عنوان برنامه‌نویس و آزمون‌گر مشغول به کار و پژوهش می‌باشد. زمینه‌های پژوهشی ایشان عبارتند از مهندسی نرم‌افزار، آزمون عملکردی و غیرعملکردی نرم‌افزار و پروتکل‌های کنتور هوشمند (DLMS).

آدرس پست‌الکترونیکی ایشان عبارت است از:

e.habibi1367@gmail.com

عباس حیدرنوری مدرک دکترای خود در علوم کامپیوتر را در سال ۱۳۸۸ از دانشگاه واترلو کانادا و مدارک کارشناسی و کارشناسی ارشد در مهندسی نرم‌افزار خود را به ترتیب در سال‌های ۱۳۷۸ و ۱۳۸۰ از دانشگاه صنعتی شریف دریافت نموده است. بعد از اتمام دوره دکترا، ایشان به مدت یک سال به عنوان پژوهشگر پسادکترا در دانشگاه لوگانو سوییس مشغول به انجام پژوهش بوده است. ایشان سپس به مدت یک سال به عنوان مهندس ارشد نرم‌افزار در زمینه تولید نرم‌افزارهای هوشمند همراه در شرکت در تورنتو کانادا مشغول به کار بوده است. دکتر حیدرنوری از بهار ۱۳۹۲ به عنوان عضو هیأت علمی در دانشکده مهندسی کامپیوتر دانشگاه صنعتی شریف مشغول به فعالیت می‌باشد. زمینه‌های پژوهشی ایشان عبارتند از مهندسی نرم‌افزار، مهندسی معکوس و بازمهندسی سیستم‌های نرم‌افزاری، و تولید سیستم‌های نرم‌افزاری به روش مؤلفه‌گرا.

آدرس پست‌الکترونیکی ایشان عبارت است از:

heydarnoori@sharif.edu

اطلاعات بررسی مقاله:

تاریخ ارسال: ۹۲/۹/۱۰

تاریخ اصلاح: ۹۲/۱۲/۱۱

تاریخ قبول شدن: ۹۲/۱۲/۱۸

نویسنده مرتبط: الهه حبیبی، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، ایران.

[1] Aspect-Oriented Software Development
[2] Functionality
[3] Modularity
[4] Aspect Mining
[5] Scattered
[6] Tangled
[7] CrossCutting Concern
[8] Separation of Concern
[9] Frameworks
[10] Evolution
[11] Unit
[12] Logging
[13] InfraRED
[14] Query
[15] Tracing
[16] Extension
[17] Open Source
[18] Constructor
[19] Regular Expression
[20] Symmetric
[21] Asymmetric
[22] Class Diagram
[23] Sequence Diagram
[24] Open Source Framework
[25] Authentication
[26] Session
[27] Client Server
[28] Use Cases
[29] Communications Diagram

[30] Message Sequence Chart
[31] Interaction Diagram
[32] Synchronization
[33] Platform
[34] Package Diagram
[35] Role-Based
[36] Natural Language Processing (NLP)
[37] Coupling
[38] Customize
[39] Classification
[40] Code Transformation
[41] Refactoring
[42] قطعه‌بندی یا Slicing روشی می‌باشد که در آن قطعه‌ای از کد که از متغیرها و دیگر بخش‌های کد تأثیر می‌پذیرد (backward slicing) و یا بر آن‌ها تأثیر می‌گذارد (forward slicing) را شناسایی می‌کنیم.
[43] Software Product Line (SPL)
[44] Feature Model
[45] Static
[46] Dynamic
[47] Hybrid
[48] Fan-in
[49] Threshold
[50] Naming Convention
[51] Software Clones
[52] Identifier Analysis
[53] Natural Language Processing (NLP)
[54] Comments
[55] Unique Method
[56] Return Value
[57] Clustering
[58] Clustering-Based Fan-in Analysis
[59] Version Control Systems
[60] Trace
[61] Formal Concept Analysis (FCA)
[62] Functionalities
[63] Undecidable
[64] Signature Based