نشریه علمی- ترویجی محاسبات نرم
شمارهٔ هفتم/ بهار و تابستان ۹۴/ صفحهٔ ۵۶-۷۳

دریافت مقاله: ۱۳۹۳/۱۲/۲۳
پذیرش مقاله: ۱۳۹۴/۷/۲۲

مروری نظام‌مند بر مهندسی نرم‌افزار جنبه‌گرا: گام‌ها، روش‌ها و چالش‌ها

فرهاد علائی[۱]، عباس رسول‌زادگان[۲*]

[۱] دانشجوی کارشناسی ارشد، مرکز آموزش‌های الکترونیکی، مهندسی نرم‌افزار، دانشگاه فردوسی مشهد، مشهد، ایران
alaei.farhad@stu.um.ac.ir
[۲] استادیار گروه مهندسی کامپیوتر، دانشگاه فردوسی مشهد، مشهد، ایران
rasoolzadegan@um.ac.ir

چکیده: رویکرد مهندسی نرم‌افزار جنبه‌گرا به‌عنوان روشی نوین مبتنی بر پیمانه‌بندی در توسعه سیستم‌هاست که به‌منظور افزایش قابلیت نگهداری و استفادهٔ مجدد نرم‌افزار مطرح شده است. مهندسی نرم‌افزار جنبه‌گرا درصدد پیمانه‌بندی دغدغه‌های مداخله‌ای با بهره‌گیری از مفهوم جنبه است. امروزه این رویکرد در محیط‌های تحقیقاتی و صنعت مورد توجه زیادی قرار گرفته و تاکنون روش‌های متفاوتی پیرامون پیاده‌سازی فازهای مختلف توسعهٔ نرم‌افزار با استفاده از جنبه‌گرایی مطرح شده است که هرکدام از آن‌ها با مزیت‌ها، محدودیت‌ها و کاربردهای متنوعی همراه‌اند. با توجه به تعدد و پراکندگی این روش‌ها، ضرورت ارائهٔ مروری جامع، کامل و نظام‌مند در این زمینه احساس می‌شود. هدف از این مقاله، بررسی روش‌های مطرح در زمینهٔ توسعهٔ نرم‌افزار جنبه‌گرا، مقایسهٔ تطبیقی آن‌ها و به‌دست‌آوردن چالش‌های مهم در این حوزه است. به همین منظور، روش‌های مختلف را براساس گام‌های توسعهٔ نرم‌افزار طبقه‌بندی کرده و به تحلیل و مقایسهٔ تطبیقی نقاط قوّت، محدودیت‌ها و کاربردهای آن‌ها می‌پردازیم. نتایج مقایسه‌های انجام‌شده امکان انتخاب مناسب‌ترین روش در هر گام از توسعهٔ نرم‌افزار را برای مهندسان و محققان این حوزه فراهم می‌کند.

واژه‌های کلیدی: جنبه‌گرایی، دغدغه، جداسازی دغدغه‌ها، مهندسی نرم‌افزار جنبه‌گرا، دغدغه‌های مداخله‌ای.

۱. مقدمه

با توجه به اینکه امروزه زندگی ما به سیستم‌های کامپیوتری وابسته است، همواره تقاضا برای سیستم‌های نرم‌افزاری بیشتر و بیشتر می‌شود [۱]. با توجه به گسترش نیازها، انتظار می‌رود تا سیستم‌های بزرگ‌تر پیاده‌سازی شوند تا فعالیت‌های پیچیده را پوشش دهند. واضح است که پیاده‌سازی این سیستم‌ها بسیار پرهزینه بوده و پیچیدگی زیادی دارند [۲]. برای پاسخ گویی به این نیازها، هماهنگی بین مجموعهٔ پیچیدهٔ وظایف و تسهیل در توسعهٔ سیستم‌ها روش‌های مهندسی نرم‌افزار گسترش یافتند و به سمت پیمانه بندی کردن وظایف در سیستم پیش رفتند. تاریخچهٔ فرایند توسعهٔ نرم‌افزار، فناوری‌های زیادی را تجربه کرده است که شامل روش‌هایی مفید به‌منظور پیمانه‌بندی نرم‌افزار بودند؛ اما با بررسی تحقیقات انجام‌شده، این نتیجه حاصل می‌شود که این روش‌ها در سیستم‌های پیچیده نمی‌توانند همهٔ دغدغه‌ها را در یک واحد مجزا پیمانه‌بندی کنند. در سیستم‌های پیچیده، دغدغه‌های مداخله‌ای باعث چند تکه شدن اجزا و در نتیجه کد برنامه می‌شوند که تأثیراتی منفی بر کیفیت سیستم، به‌خصوص قابلیت استفاده مجدد و قابلیت نگهداری آن خواهد داشت [۳]. همچنین تغییر در نیازهای کاربران روند توسعهٔ سیستم را با مشکلات زیادی روبه‌رو می‌سازد.

در دو دههٔ اخیر، مفهومی با عنوان جنبه و فناوری مهندسی نرم‌افزار جنبه‌گرا مطرح شده است. این فناوری با هدف پیمانه بندی هرچه بیشتر نرم افزار، کاهش هزینهٔ توسعهٔ سیستم و کاهش پیچیدگی درونی سیستم بر روی تکنیک‌های شیءگرا گسترش یافت و همچنان نیز در حال گسترش و تکمیل است [۴ و ۵]. به‌منظور پوشش دغدغه‌های مداخله‌ای نرم‌افزار جنبه‌گرا، از واحدهای پیمانه‌بندی‌شده به نام جنبه، به‌عنوان عناصر پایهٔ تولید نرم‌افزار استفاده می‌کند [۶]. در ابتدا از جنبه‌گرایی تنها در گام پیاده‌سازی نرم‌افزار استفاده می‌شد، اما در نهایت این نتیجه حاصل شد که لازم است جنبه‌هایی که پیاده‌سازی می‌شوند، در گام‌های قبلی توسعهٔ نرم‌افزار شناسایی شوند. به همین دلیل این رویکرد توسعه یافت و همهٔ گام‌های روند توسعهٔ نرم‌افزار را در بر گرفت. تاکنون روش‌های متنوعی به‌منظور پیاده‌سازی سیستم جنبه‌گرا ارائه شده است که هرکدام از آن‌ها با مزیت‌ها، محدودیت‌ها و کاربردهای متنوعی همراه‌اند.

با توجه به نوین بودن فناوری توسعهٔ نرم‌افزار جنبه‌گرا، محققان و صنعت‌گران زیادی قصد وارد شدن به این حوزه را دارند، اما پراکندگی منابع مورد نیاز همواره هزینهٔ زیادی را برای آن ها به همراه دارد [۷]. در این مقاله سعی شده است تا ضمن معرفی رویکرد مهندسی نرم‌افزار جنبه‌گرا، گام‌های فرایند توسعهٔ نرم‌افزار جنبه‌گرا بررسی شود و براساس جنبه‌های مختلف کیفیت که توسط محققان مختلف برای گام‌های توسعه نرم‌افزار ارائه شده است، به تحلیل و مقایسه تطبیقی روش‌های مطرح در هر گام پرداخته شود. نتیجهٔ مقایسه‌های ارائه‌شده در پایان هر گام توسعه، در انتخاب روش بهینه به‌منظور تولید نرم‌افزار با کیفیت، کمک زیادی به ما خواهد کرد.

ادامهٔ مقاله به این صورت سازمان‌دهی شده است: در بخش دوم، به تعریف مفاهیم پایهٔ جنبه‌گرایی مانند جنبه و دغدغه پرداخته می‌شود. در بخش سوم ضمن معرفی مهندسی نرم‌افزار جنبه‌گرا، گام‌های این رویکرد به‌صورت نظام‌مند بیان می‌شود، روش‌های مطرح در هر گام معرفی می‌شود و براساس جنبه‌های کیفی مرتبط به گام‌های مختلف، مقایسه‌ای تطبیقی بر روی این روش‌ها صورت می‌پذیرد. در بخش چهارم به مباحثه پیرامون موضوعات چالش‌برانگیز حوزه پرداخته می‌شود و مسائل باز در رویکرد جنبه‌گرا، براساس بررسی‌های صورت گرفته بیان خواهد شد. در بخش پایانی نیز به بیان خلاصه‌ای از مباحث گفته شده و نتیجه‌گیری پرداخته می‌شود.

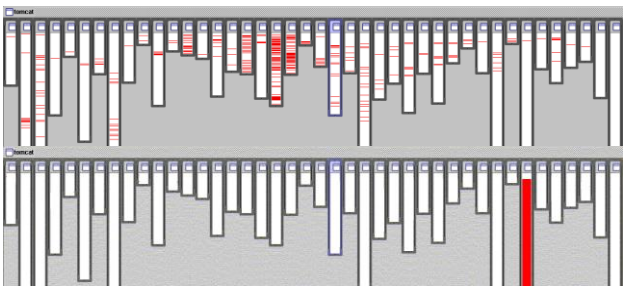
۲. مفاهیم پایه

رویکرد مهندسی نرم‌افزار جنبه‌گرا همچون دیگر مباحث در مهندسی دارای مجموعه‌ای از تعاریف و مفاهیم است که برای درک بیشتر، در ابتدای کار به معرفی مفاهیم مرتبط با حوزه تحقیق پرداخته می‌شود. یک سیستم پیچیدهٔ نرم‌افزاری را می‌توان به‌صورت پیاده‌سازی مخلوطی از چندین دغدغه تصور

کرد. دغدغه[1] یک نیازمندی و شاخص ضروری در سیستم است که به‌صورت یک مستند، سرتاسر واحدهای نرم‌افزار را می‌پوشاند. دغدغه‌ها نسبی و پویا هستند و در گذر زمان با توجه به دید و هدف توسعه‌دهنده و ذی‌نفعان سیستم تغییر می‌کند [۴ و ۸]. به‌طور کلی دغدغه‌ها را می‌توان به پنج دستهٔ کارکردی، کیفی، سیاستی، سیستمی و دغدغه‌های سازمان‌یافته تقسیم‌بندی کرد [۹]. هریک از دغدغه‌های اصلی برنامه را دغدغه هسته[2] [۱۰] می‌نامند و مجموعه دغدغه‌های هسته منطق حرفه[3] را تشکیل می‌دهند [۱۱].

در توسعه مبتنی بر مؤلفه و شیءگرا، نیازمندی‌هایی وجود دارند که نمی‌توان آن‌ها را در یک مؤلفه جای داد. به این نیازمندی‌های میان‌بَر[4]کنندهٔ مؤلفه‌ها، دغدغه‌های مداخله‌ای[5] گفته می‌شود [۴ و ۱۲]. عدم تجزیهٔ واضح از دیگر بخش‌ها ویژگی بارز این نوع دغدغه است. پراکندگی[6] و پیچش[7] در کد برنامه از علائم این نوع دغدغه است [۱۳] که باعث ضعف در ردیابی، کاهش قابلیت استفادهٔ مجدد و بهره‌وری پایین نرم‌افزار می‌شود [۱۲]. برای پوشش دغدغه‌های مداخله‌ای از مفهومی انتزاعی به نام جنبه[8] استفاده می‌شود [۸]. جنبهٔ یک واحد برنامه‌نویسی، شامل ویژگی‌های مشترک برنامه است که عملکردهای میان‌بَرکنندهٔ برنامه را در بر می‌گیرد. به بیانی دیگر، جنبه خود نوعی دغدغه است که کارکرد آن توسط دغدغه‌های دیگر و در موقعیت‌های مختلف راه‌اندازی می‌شود [۱۴]. برخلاف یک‌بعدی بودن توسعهٔ برنامه‌های شیءگرا، در توسعهٔ نرم‌افزار جنبه‌گرا پیمانه‌بندی برنامه را می‌توان از دو بعد بررسی کرد: بعد اول بعد ساختاری سیستم است و در آن کارکردهای اصلی سیستم با استفاده از مفهوم کلاس پیاده‌سازی می‌شوند. بعد دوم نیز بعد عملیاتی سیستم است و در آن کارکردهای مداخله‌ای با استفاده از مفهوم جنبه پیاده‌سازی می‌شوند. واضح است که در توسعهٔ نرم‌افزار جنبه‌گرا از هر دو مفهوم کلاس و جنبه استفاده می‌شود.

به‌منظور شناسایی موفق جنبه‌ها به سه فناوری مهم استخراج جنبه[9]، معرفی جنبه[10] و توسعهٔ جنبه[11] نیازمندیم. استخراج جنبه شامل روش‌هایی برای تشخیص صحیح دغدغه‌هاست. معرفی جنبه شامل روش‌های مورد نیاز برای تعریف جنبهٔ مناسب برای هر دغدغه است. توسعهٔ جنبه نیز شامل روش‌هایی برای ادغام جنبه در برنامهٔ اصلی است که خود شامل توسعهٔ کد اصلی و توسعهٔ کد جنبه است [۱۵]. جنبه متکی بر نقاط برش[12] و نقاط اتصال[13] است. این نقاط مشخص‌کنندهٔ محل قرارگیری جنبه در پیاده‌سازی برنامه‌اند. مجموعه دستوراتی که درون یک جنبه قرار می‌گیرند، توسط کد توصیه[14] مشخص می‌شوند. کد توصیه قابل فراخوانی مستقیم نیست و به سه نوع قبل، بعد و پیرامون طبقه‌بندی می‌شود. نوع توصیه، زمان اجرای آن را در هنگام برخورد با نقاط برش مشخص می‌کند [۱۶]. به‌منظور درک بهتر از تفاوت مفهوم جنبه در توسعهٔ نرم‌افزار جنبه‌گرا با کلاس در توسعهٔ نرم‌افزار شیءگرا در شکل (۱)، تأثیر چگونگی استفاده از جنبه بر روی کد یک برنامه در مقاسیه با کلاس نشان داده شده است.



شکل (۱): پیاده‌سازی با استفاده از جنبه در مقایسه با کلاس [۱۲]

بخش بالای شکل، برنامه‌ای را که با استفاده از کلاس پیاده‌سازی شده است، نشان می‌دهد. خطوط افقی در کد هر کلاس به‌دلیل وجود دغدغه‌های مداخله‌ای ایجاد شده‌اند. این

1. Concern
2. Core Concern
3. Business Logic
4. Cut Across
5. Crosscutting Concern
6. Scattering
7. Tangling
8. Aspect
9. Aspect Mining
10. Aspect Introduction
11. Aspect Evolution
12. Point Cuts
13. Join Point
14. Advice

دغدغه‌ها کلاس‌های برنامه را میان‌بَر می‌کنند. در بخش پایین شکل، پیاده‌سازی همان برنامه با استفاده از جنبه نشان داده شده است. دغدغه‌های مداخله‌ای درون یک جنبه قرار گرفته‌اند و از میان‌بَر شدن کلاس‌ها جلوگیری می‌شود [۱۲].

۳. جنبه‌گرایی در مهندسی نرم‌افزار

مهندسی نرم‌افزار جنبه‌گرا یک فناوری نوین در توسعهٔ نرم‌افزار است که روش‌های جدید برای پیمانه‌بندی سیستم‌های نرم‌افزاری را فراهم می‌کند. این رویکرد محدوده‌ای گسترده از تکنیک‌ها را که به پیمانه‌بندی نرم‌افزار کمک می‌کند، در بر می‌گیرد. این تکنیک‌ها شامل شیءگرایی، توسعهٔ مبتنی بر مؤلفه، طراحی الگوها و چارچوب‌های شیءگرا هستند [۴]. بنابراین توسعهٔ نرم‌افزاری جنبه‌گرا بر روی دیگر تکنیک‌های شیءگرا ساخته می‌شود [۱۷]. این رویکرد در حقیقت گسترش یافتهٔ روش‌های توسعهٔ نرم‌افزار موضوع‌گرا[1] و ویژگی‌گرا[2] می‌باشد.

جنبه‌گرایی در مهندسی نرم‌افزار باعث واحدبندی سیستم و کاهش پیچیدگی طراحی می‌شود [۱۸]. ازآنجاکه کاربردهای متفاوت در جنبه پیاده‌سازی می‌شوند و در سرتاسر برنامه پخش نیستند، برای اعمال تغییرات نیازی به جستوجو در کل برنامه نیست، بلکه کافی است جنبهٔ مورد نظر را تغییر داد، در نتیجه قابلیت نگهداری سیستم و قابلیت استفادهٔ مجدد سیستم افزایش می‌یابد [۱، ۱۶ و ۱۹]. همچنین در مدل‌های مهندسی جنبه‌گرا می‌توان با بسته‌بندی واسط‌ها و سطوح دسترسی، پنهان‌سازی اطلاعات را بسیار آسان کرد که در نتیجه باعث بالا رفتن امنیت سیستم می‌شود [۲۰]. برخلاف آنچه تصور می‌شود، جنبه‌گرایی تنها به زمان پیاده‌سازی توسط برنامه نویس سیستم محدود نیست، بلکه همچون دیگر فناوری‌های توسعهٔ نرم‌افزار از گام‌های مهندسی نیازمندی‌ها، معماری، مدل‌سازی، پیاده‌سازی و آزمایش برنامه تشکیل می‌شود. در همین راستا در ادامهٔ مقاله، به کاربرد جنبه‌گرایی در این گام‌ها اشاره می‌شود. روش‌های مطرح در هر گام به‌طور کامل معرفی و به مقایسهٔ تطبیقی آن‌ها پرداخته می‌شود.

۱.۳. جنبه‌گرایی در مهندسی نیازمندی‌ها

در مهندسی نرم‌افزار، نیازمندی عبارت است از مستندی که محصول نهایی باید ویژگی‌ها و قابلیت‌های مذکور در آن را شامل شود. این نیازمندی‌ها به دو دسته نیازهای وظیفه‌مندی[3] و نیازهای غیر وظیفه‌مندی[4] تقسیم می‌شوند [۲۱]. این تعریف در مهندسی جنبه‌گرا نیز صدق می‌کند. نیازمندی جنبه‌گرا که با نام جنبهٔ اولیه[5] شناخته می‌شود، بر روی شناسایی وابستگی‌ها و معرفی دغدغه‌های مداخله‌ای در سطح نیازمندی‌ها متمرکز است. ادغام نیازمندی‌ها یکی از اساسی‌ترین فعالیت‌ها در مهندسی نیازمندی‌های جنبه‌گراست. یکی از روش‌های مطرح به‌منظور ادغام نیازمندی‌ها استفاده از زبان توصیف نیازمندی‌هاست که در [۲۲] به آن اشاره شده است. مهندسی نیازمندی‌ها یکی از گام های مهم و اساسی در صنعت نرم‌افزار است، ازاین‌رو مهندسان نرم‌افزار نیازمند این هستند که با توجه به نوع و مقیاس پروژهٔ خود بتوانند روشی ایدئال را سرلوحهٔ کار خود قرار دهند. ازجمله ابزارهای مهمی که مهندسان نرم‌افزار در فرایند مهندسی نیازمندی‌های جنبه‌گرا استفاده می‌کنند، می‌توان به ابزار EA-Miner اشاره کرد [۲۳]. با مشخص شدن اهداف و مزایای مهندسی نیازمندی‌های جنبه‌گرا روش‌های مختلفی در این زمینه ارائه شده است. این روش‌ها مستقل از زبان برنامه‌نویسی بوده که مهم‌ترین آن‌ها بدین شرح‌اند:

- **روش ARCaDe[6]**: این روش که از روش‌های مطرح در مهندسی نیازمندی‌های جنبه‌گراست، علاوه بر تشخیص نیازمندی‌ها و تولید مستند خصوصیات نیازمندی‌ها (همچون مدل سنتی PREview)، روش‌هایی برای مشخص کردن شیوهٔ ترکیب و حل تداخل‌های مابین جنبه‌ها را نیز در بر می‌گیرد [۲۱ و ۲۳].
- **روش ARGM[7]**: این روش بر پایهٔ اهداف سیستم بنا شده

1. Subject-Oriented
2. Feature-Oriented

3. Functional Requirements (FR)
4. Non-Functional Requirements (NFR)
5. Early Aspect
6. Aspectual Requirements Composition and Decision
7. Aspects in Requirements Goal Models

است. در این روش به بررسی روابط درونی نیازمندی‌های وظیفه‌مندی و غیر وظیفه‌مندی سیستم با استفاده از گراف V پرداخته می‌شود که از تحلیل نتایج حاصل‌شده می‌توان به استخراج جنبه‌ها پرداخت [۲۴].

- **روش AOSD/UC**[1]: این روش بر پایهٔ نقاط برش و نقاط گسترش[2] بنا شده است و موارد کاربری را تا زمانی که هر مورد کاربرد بر روی چندین کلاس اثر می‌گذارند، به‌عنوان یک دغدغهٔ مداخله‌ای در نظر می‌گیرد. در این روش، موارد کاربرد به دو دسته همتا[3] و الحاقی[4] تقسیم می‌شوند [۲۵].
- **روش SMA**[5]: یک روش برای مدل‌سازی جنبه‌ها مبتنی بر زبان UML و استفاده از مدل توالی سیستم است. این روش به‌طور مداوم به تشخیص و استخراج جنبه از سناریوهای برنامه می‌پردازد. هر بخش جنبه‌ای سناریوهای برنامه به‌صورت مستقل مدل می‌شود [۲۶].
- **روش AUCDA**[6]: مشابه روش AOSD/UC عمل می‌کند؛ هرچند در این روش، صفات کیفی مانند قابلیت اطمینان و دسترس‌پذیری به‌عنوان یک مورد کاربرد به‌شمار نمی‌آیند. این روش از یک ساختار محکم برای شناسایی دغدغه‌های مداخله‌ای استفاده می‌کند [۲۷].
- **روش Cosmos**[7]: در این روش، یک طرح مدل‌سازی چندمنظوره برای دغدغه‌ها ارائه شده است. این روش دغدغه‌ها را به دو دستهٔ فیزیکی و منطقی تقسیم می‌کند [۲۸]. در این روش فضای دغدغه‌های نرم‌افزار براساس دغدغه‌ها، ارتباطات، مستندات و موضوعات مدل می‌شود [۲۹ و ۳۰].
- **روش CORE**[8]: این روش از یک فضای چندبعدی یا جداول دوبعدی برای جداسازی دغدغه‌های همپوشان و جنبه‌ها استفاده می‌کند. همچنین این روش پیشنهاد می‌کند که نیازمندی‌ها باید در یک روش یکنواخت بدون درنظرگرفتن ماهیت وظیفه‌مندی و غیر وظیفه‌مندی تجزیه شوند [۳۱ و ۳۲]. این روش علاوه بر پوشش اهداف مهندسی نیازمندی‌های جنبه‌گرا، مکانیزمی برای پشتیبانی از فاز معماری را هم شامل می‌شود [۳۳].
- **روش AOREC**[9]: این روش به برخی مسائل باز در مهندسی نیازمندی‌های سنتی و مبتنی بر مؤلفه ازجمله درجه‌بندی اجزا می‌پردازد. در این روش، جنبه یک خصیصه از سیستم است که مشخص می‌کند هر مؤلفه کدام نیازمندی را پوشش می‌دهد [۳۴]. همچنین این روش به معرفی مفهوم جنبه توده‌ای[10] می‌پردازد که در [۳۵] به آن اشاره شده است.
- **روش Them/Doc**: این روش مبتنی بر گراف، ابزارها، نمادها و مفاهیمی دارد که به شناسایی جنبه‌های پنهان در سیستم کمک می‌کند [۳۶]. همچنین این روش شیوه‌ای برای تحلیل و طراحی برنامه ارائه می‌کند [۱۰].

در بحث مهندسی نیازمندی‌ها ویژگی‌های کیفی زیادی بر انتخاب روش بهینه تاثیرگذارند. ازجمله مهم‌ترین این ویژگی‌ها می‌توان به قابلیت ردیابی[11] به‌صورت سیستماتیک اشاره کرد. این جنبه از کیفیت که یکی از مهم‌ترین نیازهای نرم‌افزار قابل درک و قابل نگهداری است، در آغاز فرایند منبع آن را مشخص می‌کند و در طول فرایند مانند یک زنجیر قابل ردیابی مراحل قبل یا بعد خود را مشخص می‌کند. ازجمله دیگر ویژگی‌های کیفی می‌توان به ترکیب‌پذیری ویژگی‌هایی که قابلیت ترکیب شدن به‌صورت یکپارچه را دارند، قابلیت توسعهٔ آسان و تغییر نیازمندی‌ها با توجه به تغییر منبع آن‌ها و یا ایجاد یا حذف یک ویژگی، مقیاس‌پذیری و داشتن هزینهٔ یکسان مستقل از اندازهٔ پروژه، قابلیت مصالحه و سبک و

1. Aspect-Oriented Software Development with Use Case
2. Extension Points
3. Peer
4. Extension
5. Scenario Modelling with Aspects
6. Aspectual Use Case Driven Approach
7. Concern-Space Modeling Schema
8. Concern-Oriented Requirement Engineering
9. Aspect-Oriented Requirement Engineering for Component-Based Software Systems
10. Aggregate Aspect
11. Traceability

سنگین کردن بین ویژگی‌های سیستم نرم‌افزاری، مسیردهی به تعیین الگوی طراحی در مراحل بعدی، همگن بودن رفتار دغدغه‌ها با هم و صحت‌سنجی و اعتبارسنجی نرم‌افزار اشاره کرد [۷]. با توجه به تنوع روش‌های موجود، در این تحقیق سعی شده تا به بررسی، تحلیل و مقایسه تطبیقی روش‌ها از نظر توجه به ویژگی‌های کیفی مطرح در بحث مهندسی نیازمندی‌ها که به آن‌ها اشاره شد، پرداخته شود. نتایج حاصل را می‌توان در جدول (۱) مشاهده کرد.

جدول (۱): مقایسهٔ تطبیقی روش‌های مهندسی نیازمندی‌های جنبه‌گرا

معیارهای مقایسه / روش‌ها	قابلیت ردیابی در آغاز فرایند	قابلیت ردیابی در طول فرایند	ترکیب‌پذیری	قابلیت توسعه	مقیاس پذیری	مصالحه و تصمیم‌گیری	مسیردهی	همگن بودن رفتار دغدغه‌ها	صحت سنجی و اعتبارسنجی
۱	✓	×	✓	✓	✓	✓	✓	✓	✓
۲	✓	✓	✓	✓	✓	✓	✓	✓	✓
۳	✓	✓	✓	✓	✓	✓	✓	✓	✓
۴	✓	×	✓	✓	✓	×	×	✓	✓
۵	✓	✓	✓	✓	✓	×	✓	✓	✓
۶	×	✓	✓	✓	✓	✓	✓	✓	✓
۷	✓	✓	✓	✓	✓	✓	✓	✓	✓
۸	×	✓	✓	✓	✓	✓	✓	×	✓
۹	×	✓	✓	✓	✓	×	✓	✓	✓

۱. روش ARCaDe ۲. روش ARGM ۳. روش AOSD/UC
۴. روش SMA ۵. روش AUCDA ۶. روش Cosmos
۷. روش CORE ۸. روش AOREC ۹. روش Them/Doc

۲.۳. جنبه‌گرایی در معماری نرم‌افزار

معماری یک سیستم در حقیقت ساختار آن سیستم است که اجزا و ویژگی‌های خارجی قابل مشاهده آن را در بر می‌گیرد [۳۷]. در معماری جنبه‌گرا، هدف اصلی مشخص کردن دغدغه‌های مداخله‌ای در طراحی‌های معماری است. در روش‌های مرسوم شیءگرا توانایی پیمانه‌بندی دغدغه‌های مداخله‌ای در این سطح وجود ندارد. در معماری جنبه‌گرا روش‌هایی تکنیکی ارائه شده است تا در یک مدل، تشخیص دغدغه‌ها از دید معماری سیستم به سادگی صورت پذیرد [۴]. این روش‌ها نشان می‌دهند که جداسازی چندبعدی دغدغه‌ها[1] چگونه به پشتیبانی از ترکیب اجزای مستقل نرم‌افزار کمک می‌کند. در ادامهٔ این بخش به روش‌های مطرح در زمینهٔ معماری برنامه‌های جنبه‌گرا اشاره خواهد شد.

- **روش PCS**[2]: این روش برای مجسم کردن دغدغه‌ها از دیدگاه‌های متفاوت در یک دید مبتنی بر معماری، شامل چندین مدل و نمودار است. این مجسم‌سازی به دو صورت سطح بالا و سطح پایین و مبتنی بر فضای UML امکان‌پذیر است [۳۸].
- **روش DAOP-ADL**[3]: این روش یک زبان توصیف معماری مبتنی بر XML برای توصیف معماری برنامه به‌صورت مجموعه‌ای از اجزا، جنبه‌ها و ارتباط درونی بین آن‌هاست. این ارتباطات بر پایهٔ دو قانون ترکیب شامل قوانین ترکیب اجزا و قوانین ارزیابی جنبه ساخته می‌شوند [۳۹].
- **روش AOGA**[4]: این روش در ابتدا با هدف پشتیبانی از پیاده‌سازی سیستم‌های چند عامله ارائه شده است، اما امروزه برای تمام سیستم‌ها کاربرد دارد. این روش با پیروی از UML سعی دارد تا حفظ جامعیت جنبه‌ها را بر عهده گیرد [۴۰ و ۴۱]. این روش علاوه بر طراحی معماری، تا حدی فازهای بررسی و توصیف دامنه و فاز پیاده‌سازی سیستم را نیز در بر می‌گیرد [۴۲].
- **روش TranSAT**: این روش یک چهارچوب برای توصیف توسعهٔ نرم‌افزار است و بر روی تسهیل فرایند توسعهٔ معماری نرم‌افزار جنبه‌گرا متمرکز است. این روش سه چالش را در زمینهٔ معماری نرم‌افزار نمایان می‌سازد. چالش اول اینکه ادغام یک دغدغهٔ جدید باید در قالب این چهارچوب صورت گیرد. چالش دوم بر روی قابلیت

1. Multi Dimension Separation of Concerns (MDSOC)
2. Perspectival Concern-Space
3. Dynamic Aspect-Oriented Platform Architecture Description Language
4. Aspect-Oriented Generative Approach

استفادهٔ مجدد جنبه‌ها تأکید دارد. چالش سوم نیز بر این نکته تأکید دارد که افزودن یک دغدغهٔ جدید نباید سازگاری سیستم را تهدید کند [۴۳].

- **روش ASAAM**[1]: این روش مبتنی بر سناریو به ارزیابی معماری پرداخته و هدف اصلی آن پشتیبانی از معماری جنبه‌گرا به‌صورت سیستماتیک است. سناریوها به سه دستهٔ مستقیم، غیرمستقیم و جنبه‌ای تقسیم می‌شوند. این فرایند مبتنی بر پنج گام داوطلب شدن برای توسعهٔ معماری، توسعهٔ سناریوها، ارزیابی انفرادی سناریوها و شناسایی جنبه‌ها، ارزیابی تعامل سناریوها و عامل‌یابی مجدد معماری است [۴۴].

این روش‌ها به سه دسته مدل کردن معماری، فرایند معماری و ارزیابی معماری سیستم طبقه‌بندی می‌شوند. در این تحقیق به مقایسهٔ تطبیقی این روش‌ها پرداخته شد که نتایج حاصل در جدول (۲) ارائه شده است.

جدول (۲): مقایسه تطبیقی روش‌های معماری نرم‌افزار جنبه‌گرا

طبقه‌بندی روش‌ها	روش‌ها / معیارهای مقایسه	قابلیت ردیابی در آغاز فرایند	قابلیت ردیابی در طول فرایند	ترکیب‌پذیری	قابلیت توسعه	مقیاس‌پذیری
مدل کردن معماری	PCS	⊗	O	●	●	O
	ADOP-ADL	⊗	O	●	●	●
فرایند معماری	AOGA	⊗	●	O	●	O
	TranSAT	⊗	●	●	O	●
ارزیابی معماری	ASAAM	⊗	O	○	●	●

بالا ● متوسط O محدود ○ پشتیبانی نمی‌شود ⊗

۳.۳. جنبه‌گرایی در طراحی نرم‌افزار

در فرایند طراحی نرم‌افزار، طراحی به معنای مدل‌سازی رفتار سیستم در وضعیت‌های متفاوت است. در مهندسی جنبه‌گرا یکی از فعالیت‌های مهم تمرکز بر روی طراحی سیستم است [45]. طراحی جنبه‌گرا بر این حقیقت متکی است که دغدغه‌هایی که در شیوه‌های طراحی و مدل‌سازی مرسوم پراکنده بودند، قابل پیمانه‌بندی باشند. روش‌های طراحی جنبه‌گرا عموماً شامل یک بخش فرایند و یک بخش زبان هستند [۴۶ و ۴۷]. این فاز از مهندسی نرم‌افزار جنبه‌گرا نیز دارای روش‌هایی است که به معرفی آن‌ها می‌پردازیم.

- **روش AODM**[2]: این روش مبتنی بر UML استاندارد است، با این تفاوت که به جای کلاس از مفهوم جنبه استفاده می‌کند. این روش به‌خوبی از جداسازی دغدغه‌های ساختاری و رفتاری پشتیبانی می‌کند [۴۸].
- **روش Theme/UML**: این روش با اطمینان بالایی از تجزیه دغدغه‌ها در طول چرخهٔ نرم‌افزار پشتیبانی می‌کند. خروجی این روش گسترشی از UML است. در این روش به جای جنبه، هر ساختار طراحی به‌عنوان یک Theme شناخته می‌شود [۴ و ۱۳].
- **روش SUP**[3]: زبان طراحی در این روش مبتنی بر UML است و از نمودار وضعیت برای نمایش فرایند نرم‌افزار استفاده می‌کند. در هر لحظه، جنبه‌های همزمان کنترل فرایند را به دست می‌گیرند.
- **روش SUP-AOD**[4]: این روش به بررسی مدل شیءگرا می‌پردازد و سعی می‌کند وضعیت مداخله‌ای را تشخیص دهد و سپس جنبه‌ها را استخراج کند. این فرایند با مجموعه‌ای گام صورت می‌پذیرد که در [۴۹] به معرفی آن پرداخته شده است.
- **روش AAM**[5]: این روش تلاش می‌کند دغدغه‌ها را از سطح میانی طراحی تا سطح بالای آن شناسایی کند و خروجی را به زبان UML مدل کند [۴]. جنبه‌های سطح بالا به‌عنوان الگوی پارامتر در نظر گرفته می‌شوند [۵۰].
- **روش CoCompose**: این روش به معرفی ساختار نهایی با استفاده از جنبه‌های سطح بالا با قابلیت استفادهٔ مجدد

1. Aspectual Software Architecture Analysis Method

2. Aspect-Oriented Design Modelling (AODM)
3. State Charts and UML Profile
4. State Charts and UML Profile Aspect-Oriented Design
5. Aspect-Oriented Architecture Modelling

می‌پردازد. زبان این روش مبتنی بر UML نیست. پشتیبانی از قابلیت استفادهٔ بالای جنبه‌ها ویژگی مهم این روش است [۵۱].

- **روش UFA**[1]: این روش گسترشی از روش AOD بوده و بر پایهٔ مدل همکاری مبتنی بر جنبه[2] عمل می‌کند. این روش از بسته‌های UML برای کپسوله‌سازی بخش‌های سیستم که در ارتباط با هم رفتاری پیچیده ارائه می‌کنند، استفاده می‌کند [۴].
- **روش UXF**[3]: در این روش جنبه‌ها بر مبنای صفات، رفتارها و ارتباط‌ها شناخته می‌شوند که این ارتباط‌ها ممکن است به شکل ارث‌بری، شرکت‌پذیری یا مستقل در نظر گرفته شوند. این روش از توسعه سیستم به‌شدت پشتیبانی می‌کند [۵۲].
- **روش AVA**[4]: در این روش، مفهوم زیرجنبه مطرح می‌شود تا دو مشکل مرسوم در روش‌های AOD، یکی تأثیر جنبه‌ها بر هم و دیگری طرح قوانین برای طراحی جنبه‌گرای قابل توسعه را پوشش دهد [۵۳].
- **روش AML**[5]: این روش با استفاده از یک سطح طراحی طراحی متوسط به پایین و با دنبال کردن روش UFA سعی دارد تا با استفاده از قابلیت استفاده مجدد جنبه‌ها از مهندسی رو به جلو پشتیبانی کند [۵۴].
- **روش ADT**[6]: این روش جزء اولین روش‌های ارائه‌شده در فاز طراحی جنبه‌گراست که در آن، ایدهٔ جداسازی دغدغه‌های همزمان به‌وجود آمد [۵۵].

روش‌های گفته‌شده را می‌توان از لحاظ درجهٔ تجرد و سطح مداخله جنبه‌ها در یکدیگر مقایسه کرد. درجهٔ تجرد شامل میزان استقلال از دیگر مدل‌های طراحی است؛ سطح مداخلهٔ جنبه‌ها نیز نوع پیاده‌سازی دغدغه‌ها در زمان برنامه‌نویسی را شامل می‌شود که شامل دو سطح متقارن و نامتقارن است. نتایج حاصل از این مقایسه در جدول (۳) نشان داده شده است.

جدول (۳): مقایسهٔ تطبیقی روش‌های طراحی جنبه‌گرا

معیارهای مقایسه / روش‌ها	درجهٔ تجرد	سطح جداسازی دغدغه‌ها
AODM	○	☒
Them/UML	○	☑
SUP	○	☑
SUP AOD	◎	☑
AAM	●	☒
CoCompose	◎	☑
UFA	●	☑
UXF	○	☒
AVA	●	☑
AML	○	☒
ADT	○	☒

بالا ● متوسط رو به بالا ◎ متوسط ◎ متوسط رو به پایین ○

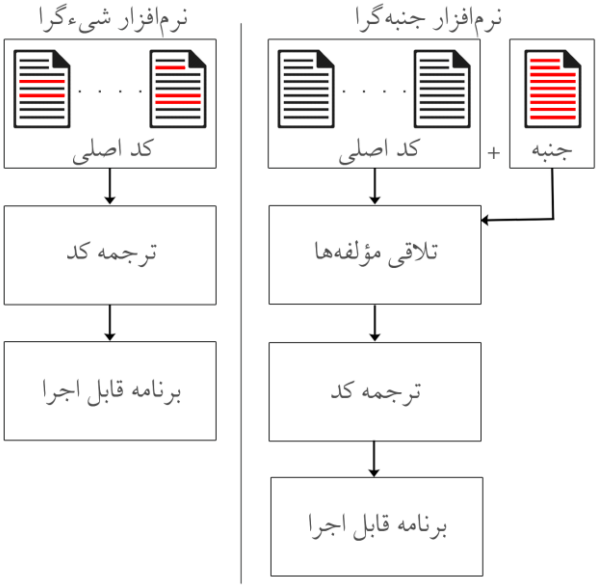
متقارن ☑ نامتقارن ☒

۴.۳. جنبه‌گرایی در پیاده‌سازی نرم‌افزار

برای پیاده‌سازی برنامه‌های جنبه‌گرا با استفاده از الگوهای طراحی موجود، از برنامه‌نویسی جنبه‌گرا[7] استفاده می‌شود [۵۶]. برنامه‌نویسی جنبه‌گرا یک پارادایم برای جداسازی دغدغه‌ها و پیمانه‌بندی‌کردن دغدغه‌های مداخله‌ای در داخل موجودیت‌های نرم‌افزاری خوش‌تعریف به نام جنبه است [57]. زبان‌های جنبه‌گرا از مکانیزمی خاص برای تلاقی[8] رفتارهای گوناگون به‌منظور ارائهٔ نرم‌افزاری کامل بهره می‌گیرند [۵۸ـ۶۰]. ترکیب جنبه‌ها و کلاس‌ها می‌تواند در زمان ترجمه، زمان بارگذاری، زمان گسترش یا زمان اجرا صورت پذیرد. یکی از تفاوت‌های توسعهٔ برنامهٔ شیءگرا و جنبه‌گرا، مسیر و چگونگی پیاده‌سازی نرم‌افزار است. همان‌گونه که در شکل (۲) مشاهده می‌شود، در پیاده‌سازی برنامه‌های شیءگرا کد اصلی که شامل کلاس‌های برنامه است، به‌طور مستقیم به برنامهٔ مترجم داده می‌شود تا خروجی مورد نظر تولید شود، اما در برنامه‌های مبتنی بر جنبه در ابتدا باید کد اصلی برنامه که از کلاس‌ها تشکیل شده است، با جنبهٔ استخراج‌شده در گام‌های

1. UML for Aspects
2. Aspectual Collaborations Model (ACM)
3. UML eXchange Format
4. Architectural View of Aspects
5. Aspect Modelling Language
6. Aspect at Design Time
7. Aspect-Oriented Programming (AOP)
8. Weaving

قبلی توسعه و با استفاده از ابزار مناسب ترکیب شود، سپس کد متلاقی برای تولید برنامه قابل اجرا به مترجم داده شود. گرچه این فرایند زمان بیشتری را صرف می‌کند، در سیستم‌های پیچیده، زمان و هزینهٔ به‌روزرسانی برنامه به‌طور قابل ملاحظه‌ای کاهش خواهد یافت.



شکل (۲): تفاوت مراحل پیاده‌سازی برنامه‌های شیءگرا و جنبه‌گرا

به‌طور کلی پیاده‌سازی برنامه‌های جنبه‌گرا از سه مرحله تشکیل می‌شود:

- تجزیهٔ جنبه‌ای: نیازمندی‌ها، به‌منظور مشخص کردن دغدغه‌های مداخله‌ای تجزیه می‌شوند.
- پیاده‌سازی دغدغه‌ها: هر دغدغه به‌صورت کاملاً جداگانه پیاده‌سازی می‌شود.
- ترکیب مجدد جنبه‌ای: با استفاده از مجتمع‌ساز جنبه، قوانین مربوط به ترکیب مجدد مشخص شده و با تلاقی جنبه‌ها سیستم نهایی تولید می‌شود.

اکثر زبان‌های مورد استفاده در زمینهٔ برنامه‌نویسی جنبه‌گرا همان زبان‌های برنامه‌نویسی شیءگرا هستند که قابلیت پشتیبانی از مفاهیمی مانند جنبه، نقاط برش، نقاط اتصال و توصیه [۱۲] به آن‌ها اضافه شده است [۳۷].

جدول (۴): مهم‌ترین ابزارهای مطرح در پیاده‌سازی نرم‌افزار جنبه‌گرا

نام ابزار	معرفی ابزار
POSTSHARP	این ابزار، یک افزونهٔ کارآمد به‌منظور ترکیب کد در زمان اجرا برای ارتقای محیط میکروسافت دات نت[1] است.
GLASSBOX	برنامه‌ای متن باز برای محیط ASPECTJ است که برای نظارت بهتر بر پیاده‌سازی برنامه، قابلیت‌هایی را به چهارچوب موجود می‌افزاید.
MOTOROLA WEAVER	این ابزار قبل از تولید کد، تلاقی ماشین‌های وضعیت UML ایجاد شده در گام‌های قبل را اجرا می‌کند که در برنامه‌های بزرگ و صنعتی بسیار مفید است.
ASPECT BENCH COMPILER (ABC)	یکی از مترجم‌های مطرح زبان ASPECTJ می‌باشد که به‌دلیل سرعت بالا و بهینه‌سازی کد اصلی، کاربران زیادی را به خود جذب کرده است.

این قابلیت‌ها از طریق افزونه‌های موجود به ابزارهای پیاده‌سازی و محیط‌های برنامه‌نویسی شیءگرا افزوده می‌شوند. ازجمله مهم‌ترین افزونه‌ها و ابزارهای مطرح در زمینهٔ پیاده‌سازی جنبه‌گرا می‌توان به مواردی که در جدول (۵) معرفی شده‌اند، اشاره کرد.

جدول (۵): مقایسهٔ تطبیقی چند نمونه از زبان‌های برنامه‌نویسی جنبه‌گرا

معیارها / زبان‌ها	نمونه‌سازی جنبه‌ها	ارث‌بری از جنبه‌ها	جنبه‌های تو در تو	اولویت دهی به جنبه‌ها	نقاط برش چندریخت	توصیه‌های چندریخت
AspectJ	×	✓	✓	✓	×	×
AspectS	✓	✓	×	×	×	×
AspectR	✓	×	×	✓	×	×
AspectC	✓	✓	✓	×	×	✓
AspectT	✓	✓	✓	×	×	✓
AspectML	دارای ساختار پشتیبان جنبه نیست.				✓	✓

از نظر مکانیزم‌های زبان‌شناسی تکنیک‌های برنامه‌نویسی جنبه‌گرا به سه دسته روش‌های مبتنی بر صافی، روش‌های مبتنی بر پیمایش و روش‌های چند بعدی طبقه‌بندی می‌شوند

1. Microsoft .NET

[۶۱]. AspectJ مهم‌ترین و مشهورترین زبانی است که امروزه از برنامه‌نویسی جنبه‌گرا پشتیبانی می‌کند. AspectJ گسترشی از زبان جاوا و اولین محیطی است که امکان برنامه‌نویسی جنبه‌گرا را فراهم کرد [۱۱]. این زبان از مکانیزم تلاقی آرمیده[1] استفاده می‌کند [۶۲]. در [۶۳] به برخی دیگر از زبان‌های رایج در زمینهٔ برنامه‌نویسی جنبه‌گرا اشاره شده است. برای مقایسهٔ زبان‌های موجود به قابلیت‌هایی که در جدول (۴) به آن اشاره شده است، توجه می‌شود [۶۴ و ۶۵].

۵.۳. جنبه‌گرایی در آزمایش نرم‌افزار

آزمایش یکی از گام‌های اساسی و پرهزینه در روند توسعه نرم‌افزار می‌باشد [۵ و ۶۶] که هدف از انجام آن یافتن خطاهای برنامه و بررسی میزان برآورده ساختن نیازها است. با توجه به اینکه سیستم‌های جنبه‌گرا ویژگی‌های جدید دارند، امکان به‌کارگیری روش‌های آزمایش مرسوم برای این برنامه‌ها مناسب نیست، زیرا جنبه‌ها از نظر محتوا با کلاس‌ها متفاوت هستند. پیمانه‌بندی برنامه به این معنی نیست که تمام دغدغه‌ها به‌صورت جداگانه رفتار می‌کنند، بلکه جنبه‌های برنامه بر یکدیگر تأثیر می‌گذارند و علاوه بر سنجش هر جنبه، آزمایش ارتباط بین آن‌ها نیز اهمیت دارد، همچنین ارتباط جنبه‌های برنامه با کلاس‌ها، آزمون برنامه را پیچیده‌تر می‌کند. آزمون‌پذیری یکی از معیارهای مهم سنجش کیفیت نرم‌افزار است که معمولاً ارزیابی آن با مشکلاتی همراه است. یکی از روش‌های مؤثر برای بررسی آزمون‌پذیری برنامه‌های جنبه‌گرا استفاده از روش MCDM[2] و با استفاده از منطق فازی و به‌دست آوردن ماتریس تصمیم است [۶۷]. در بررسی آزمون‌پذیری برنامه‌های جنبه‌گرا به‌وسیلهٔ منطق فازی، سطح جداسازی دغدغه‌ها، چسبندگی[3]، اندازهٔ برنامه و اتصال[4] معیارهای مهم تأثیرگذار در سنجش کیفیت نرم‌افزارند [۶۸].

در رابطه با آزمون برنامه‌های جنبه‌گرا، روش‌های متنوعی ارائه شده است. بسیاری از مهندسان آزمون نرم‌افزار، به‌منظور آزمون برنامه‌های جنبه‌گرا ترجیح می‌دهند آن را به‌صورت شیءگرا مورد سنجش قرار دهند و در انتها جنبه‌های شامل دغدغه‌های مداخله‌ای را با استفاده از روش‌هایی خاص بسنجند. در [۶۹] به بررسی روش‌های رسمی آزمایش مختص جنبه‌ها پرداخته شده است. در آزمودن برنامه‌های جنبه‌گرا هدف اصلی انتخاب موارد آزمون با توجه به هر جنبه است که در ادامه به معرفی آن‌ها می‌پردازیم.

- **آزمون واحد مبتنی بر جریان داده**: این روش ازجمله اولین روش‌ها در زمینهٔ آزمون برنامه‌های جنبه‌گرا است که در آن برای هر جنبه، یک آزمون سه‌سطحی شامل سطح درون‌پیمانه‌ای، برون‌پیمانه‌ای و درون‌جنبه‌ای صورت می‌پذیرد. این روش که مبتنی بر آزمون جعبهٔ سفید است [۵]، شرح می‌دهد که چگونه با ترکیب آزمون واحد و آزمون جریان داده، می‌توان جنبه‌ها را مورد سنجش قرار داد [۶۶].
- **آزمون مبتنی بر وضعیت**: این روش رفتار جنبه‌ها و ارتباط آن‌ها را در چهارچوب توالی‌های پیام می‌سنجد. در این روش با استفاده از مدل وضعیت به زبان UML رفتار جنبه‌ها به‌صورت درخت انتقال شبیه‌سازی می‌شود که هر مسیر از ریشه به برگ یک مورد آزمون را شامل می‌شود. در [۷۰] با تکیه بر این الگو، یک روش آزمون خودکار برنامه ارائه شده است. در [۷۱] با استفاده از ماشین وضعیت مور[5] مبتنی بر وضعیت برای صحت‌سنجی نرم‌افزار جنبه‌گرا ارائه شده است.
- **آزمون گراف جریان جنبه**: این الگو در حقیقت از به هم پیوستن مدل وضعیت حوزهٔ جنبه[6] و گراف جریان ایجاد می‌شود. مدل وضعیت حوزهٔ جنبه شامل نمودار انتقال وضعیت برنامه‌های جنبه‌گرا می‌باشد. گراف جریان جنبه عامل‌ها، توصیه‌ها و جریان کنترلی بین آن‌ها را با توجه به درخت انتقال وضعیت مورد بررسی قرار می‌دهد و قادر

1. Relaxed Weaving
2. Multicriteria Decision Making Approach
3. Cohesion
4. Coupling
5. Moore
6. Aspect Scope State Model

است دو دسته از اِشکال[1]ها شامل جریان کنترلی غلط و تغییرات غلط در کنترل استقلال جنبه‌ها را تشخیص دهد [۶۶].

- **آزمون برگشتی**[2]: این روش با استفاده از تحلیل کدهای بافته‌شده به دو شکل ایستا و پویا قادر است اِشکال‌های جدید را تشخیص دهد. در نوع پویا سعی می‌شود بیشترین خط کد را به‌عنوان مورد آزمون در بر گیرد و در تحلیل ایستا صحت این انتخاب سنجیده می‌شود [۵]. این روش مبتنی بر دو سناریوی مجزا می‌باشد که در [۷۲] به آن اشاره شده است. همچنین چهارچوب RETESA[3] برای این منظور پیشنهاد شده است.
- **آزمون مبتنی بر اِشکال**: در این روش پس از طبقه‌بندی اِشکال‌های احتمالی [۷۳]، با استفاده از تزریق اِشکال به سنجیدن برنامه پرداخته می‌شود. در ارائهٔ یک مدل اِشکال، به‌خصوص برنامه‌های نوشته‌شده با استفاده از AspectJ، سه عامل زیر تأثیرگذارند [۷۴]: ۱. مدل نقاط برش، دغدغه‌ها و جنبه‌ها ۲. لیست‌های اِشکال برای نقاط برش، دغدغه‌ها و جنبه‌ها ۳. مدل اِشکال جاوا. در [۵] نیز به روش آزمون تحول[4] جنبه که براساس آزمون مبتنی بر اِشکال ارائه شده، اشاره شده است.
- **آزمون واحد مبتنی بر رفتار**: در این روش رفتار اجزای مختلف مانند کد توصیه مورد آزمون قرار می‌گیرد. این روش بر پایهٔ زبان جاوا[5] و با استفاده از چهارچوب JamlUnit موارد آزمون را انتخاب می‌کند که می‌توان رفتار هرکدام را به‌طور واحد مورد سنجش قرار داد [۷۵].
- **آزمون مبتنی بر مدل**: این روش مبتنی بر مدل اِشکال و بر پایهٔ مدل زبان UML ارائه شده است. این مدل‌ها از سه نمودار کلاس، جنبه و توالی تشکیل می‌شوند. با استفاده از این مدل‌ها موارد آزمون مشخص خواهند شد.

در [۷۶] مدلی به‌منظور آزمون خودکار برنامه‌های جنبه‌گرا ارائه شده است.

علاوه بر روش‌هایی که به آن‌ها اشاره شد، روش‌های دیگری نیز در مقالات مختلف بیان شده است [۷۱ و ۷۷ــ۸۰]؛ با کمی بررسی درخواهیم یافت این روش‌ها گسترشی از روش‌های مذکورند و تفاوت چندانی با آن‌ها ندارند. روش‌های گفته‌شده را می‌توان از نظر قدرت شناسایی اِشکال‌های مختلف نرم‌افزار در سطح جنبه مورد مقایسهٔ تطبیقی قرار داد. نتایج حاصل از این مقایسه در جدول (۶) نشان داده شده است.

جدول (۶): مقایسهٔ تطبیقی روش‌های آزمون نرم‌افزار جنبه‌گرا

روش‌ها / نوع اِشکال	مبتنی بر جریان داده	مبتنی بر وضعیت	مبتنی بر جریان جنبه	آزمون برگشتی	آزمون مبتنی بر اِشکال	مبتنی بر رفتار	مبتنی بر مدل
تغییر اولویت اجرای جنبه‌ها	×	✓	✓	✓	×	×	×
پوشش اجرای توصیه‌ها	✓	✓	×	✓	✓	✓	×
پوشش اجرای جنبه‌های هم‌زمان	✓	×	×	×	×	×	✓
تشخیص انتقال غیر مجاز	×	×	×	✓	×	×	×
بررسی استقلال جنبه‌ها	×	×	✓	✓	✓	✓	✓
تغییرهای ناگهانی در وضعیت جنبه	×	×	✓	✓	×	×	✓

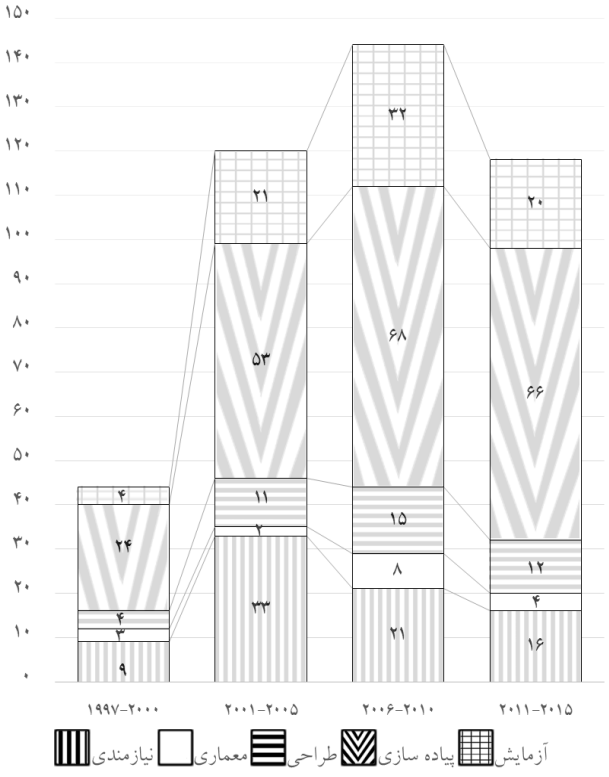
۴. مباحثه

مهندسی جنبه‌گرا همچون دیگر فناوری‌ها عاری از اِشکال نیست. اگرچه ایدهٔ مهندسی جنبه‌گرا به بهبود فرایند پیمانه‌ای ساختن برنامه کمک زیادی نکرده است، با ورود به این حوزه با مسائلی روبه‌رو خواهیم شد که به توجه بیشتری نیاز دارند. چالش اصلی این رویکرد به‌صورت جزئی، نو بودن اهداف و به‌کارگیری

1. Fault
2. Regression
3. Regression Test Selection for Aspect-Oriented Programs
4. Mutation Testing
5. Java Aspect Markup Language (JAML)

روش‌های جدید است، همان طور که دیدیم برای هر گام توسعه نرم‌افزار روش‌های زیادی ارائه شده است، اما متأسفانه هیچ سازمانی تاکنون روش‌های گفته‌شده در این مقاله را به‌صورت یک استاندارد رسمی اعلام نکرده است [۶]. همانطور که در شکل‌های (۱) و (۲) دیدیم در هر برنامه یک یا چند جنبه وجود دارد که دغدغه‌های مداخله‌ای را در بر می‌گیرد، اما در مقاله‌های حاضر به چگونگی اعمال تغییر در این مؤلفه‌ها اشاره نشده است. به‌علاوه این جنبه‌ها باعث به‌وجودآمدن تک نکتهٔ خرابی در سیستم خواهند شد که در سیستم‌های بحرانی–ایمن این مسئله می‌تواند منجر به فاجعه شود.

در بحث مهندسی جنبه‌گرا دیدیم این دیدگاه وجود دارد که در روند توسعه به‌صورت شیءگرا عمل شود و تنها توسعه‌دهندهٔ سیستم (برنامه‌نویس) وظیفهٔ پیاده‌سازی به‌صورت جنبه‌گرا را بر عهده دارد. به‌منظور تأیید این موضوع، مقالات موجود در حوزهٔ جنبه‌گرایی از سال ۱۹۹۷ تا ۲۰۱۵ بررسی شد که نتایج حاصل به تفکیک تمرکز بر گام‌های توسعهٔ نرم‌افزار در شکل (۳) نشان داده شده است.



شکل (۳): بررسی موضوعی مقالات در حوزهٔ مهندسی جنبه‌گرا

همان‌گونه که مشاهده می‌شود، بیشتر مقاله‌ها به بررسی زمینه‌های پیاده‌سازی و برنامه‌نویسی جنبه‌گرا پرداخته‌اند و توجه اندکی به دیگر گام‌ها به‌خصوص گام‌های معماری و طراحی جنبه‌گرا شده است، به‌طوری‌که تاکنون الگوی طراحی خاصی برای توسعهٔ برنامه‌های جنبه‌گرا پیشنهاد نشده است.

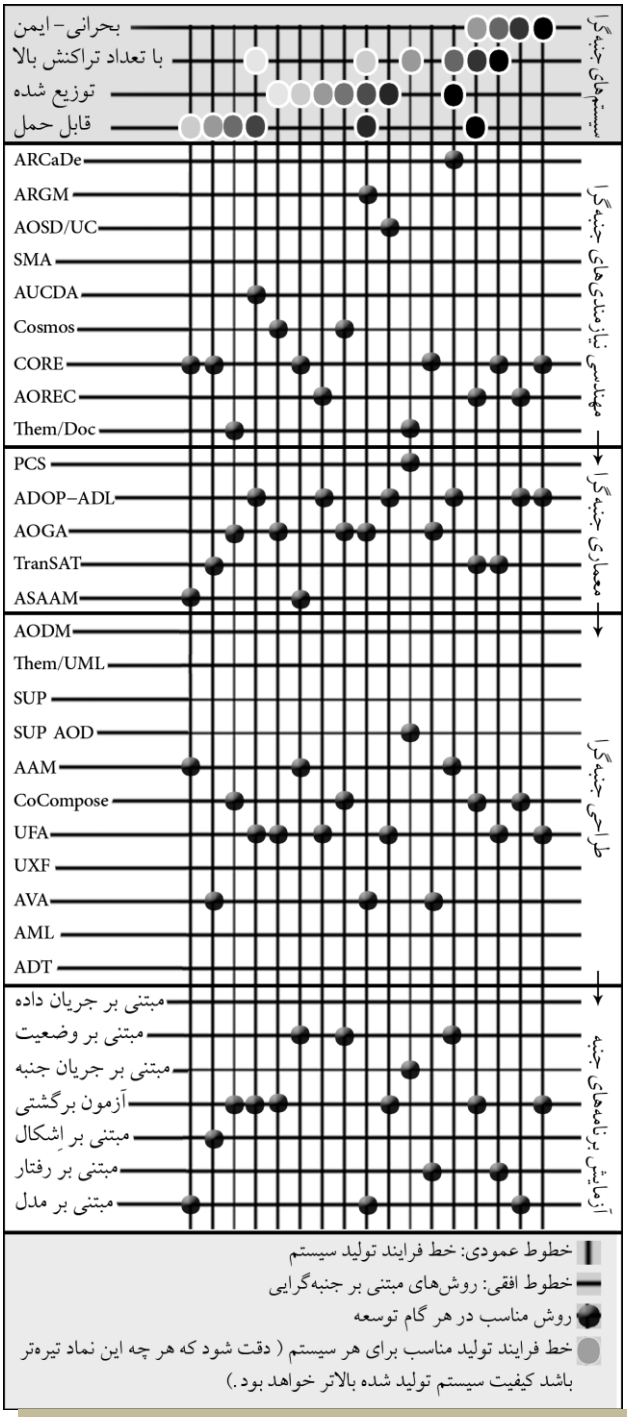
با توجه به مقایسهٔ تطبیقی انجام‌شده بر روی روش‌های مهندسی نیازمندی‌های جنبه‌گرا در جدول (۱) این نکته استخراج می‌شود که این روش‌ها تا سطح مطلوبی ویژگی‌های کیفی نرم‌افزار را پوشش می‌دهند، اما بهتر است برای جمع‌آوری این نیازمندی‌ها، الگوهایی مبتنی بر جنبه تعریف شود، زیرا یک الگوی طراحی مشخص می‌کند در گام‌های بعدی توسعه، کدام نیازمندی‌ها جزء دغدغه‌های اصلی سیستم خواهند بود. همان‌گونه که مقایسهٔ تطبیقی انجام‌شده در جدول (۲) نشان می‌دهد، در گام معماری نیز توجهی به قابلیت ردیابی دغدغه‌ها نشده است که با توجه به مهم بودن این موضوع، پوشش این قابلیت به یک مسئله باز تبدیل شده است. همچنین بهتر است به ارزیابی فرایند معماری نرم‌افزار جنبه‌گرا توجه بیشتری شود. در طراحی نرم‌افزار با توجه به جدول (۳) متوجه می‌شویم که جداسازی دغدغه‌ها در این گام به‌خوبی انجام نمی‌شود. در [۵۰] نیز به برخی مسائل باز در زمینهٔ مدل‌سازی و طراحی نرم‌افزارهای جنبه‌گرا اشاره شده است.

متأسفانه بیشتر برنامه‌نویسان به‌منظور پیاده‌سازی سیستم‌های جنبه‌گرا، از زبان‌های توسعهٔ شیءگرا استفاده می‌کنند؛ البته جدول (۵) این نکته را روشن می‌سازد که تاکنون زبان قابل قبولی به‌منظور پشتیبانی کامل از پیاده‌سازی نرم‌افزار جنبه‌گرا به بازار عرضه نشده است. همچنین به‌طور کلی زبان برنامه‌نویسی جنبه‌گرا با بازگشت به‌سمت کدهای اسپاگتی (مانند استفاده از دستورهایی نظیر Go to) فرایند اِشکال‌زدایی برنامه را با مشکل روبه‌رو می‌کند [۱۱]. تولید نرم‌افزار با استفاده از جنبه، چالش‌هایی را نیز در صحت‌سنجی و اعتبارسنجی نرم‌افزار به‌وجود آورده است [۳۷ و ۷۹]. همان‌طور که جدول (۵) مشخص می‌کند، هیچ‌کدام از روش‌های آزمون نرم‌افزار جنبه‌گرا

قابل اعتماد نبوده و از قدرت کافی برای شناسایی اِشکال برخوردار نیستند. همچنین در [۸۲] به پنج مسئلهٔ عمدهٔ فرایند تعیین جنبه‌ها، عناصر قابل استفاده در مدل‌ها، اختصاص جنبه به یک نقش خاص، سطوح استفاده شده در پیمانه‌بندی و و فرایند پیمانه‌بندی که شرکت‌های نرم‌افزاری با آن‌ها روبه‌رو شده‌اند، اشاره می‌شود. همان‌طور که مشاهده می‌شود، این مسائل بیشتر به مسیر انجام فرایند مربوط می‌شوند که گاهی در ابتدای امر مهندسان را با چالش‌هایی روبه‌رو می‌سازد.

مهندسان نرم‌افزار همواره تلاش می‌کنند تا سیستم‌هایی با انعطاف‌پذیری بالا و با حداکثر استاندارد ممکن تولید کنند. با توجه به [۸۱ـ۸۵] سیستم‌هایی که تمایل استفاده از رویکرد توسعه جنبه‌گرا در آن‌ها بیشتر است، به چهار دستهٔ اصلی سیستم‌های بحرانی-ایمن، سیستم‌های با تعداد تراکنش بالا، سیستم‌های توزیع‌شده و سیستم‌های قابل حمل طبقه‌بندی می‌شوند. در هر گام از روند تولید این سیستم‌ها، استفاده از روش‌های مشخصی برای افزایش کارایی و ارائهٔ سیستمی استاندارد توصیه می‌شود. به همین منظور با توجه به جداول ارزیابیِ روش‌های گفته‌شده در بخش‌های قبل، ارزیابی جنبه‌های کیفی نرم‌افزار جنبه‌گرا با استفاده از مدل کیفیت نرم‌افزار مک کال[1] [۸۶] و همچنین مجموعه ویژگی‌هایی که برای هرکدام از سیستم‌ها مورد نیاز است، براساس توانایی برآورده ساختن کارکردهای هر سیستم، برای هر گام، روش‌هایی پیشنهاد گردید.

روش‌های پیشنهادی برای هر سیستم در شکل (۴) مشخص شده‌اند. در گام پیاده‌سازی نیز استفاده از زبان‌های مبتنی بر جاوا پیشنهاد می‌شود. در هر سیستم اولویت توالی‌های پیشنهادی از راست به چپ و براساس شدت تیرگی نمادهای به‌کاررفته است؛ برای نمونه، توالی روش‌های CORE در گام مهندسی نیازمندی‌ها، ADOP-ADL در گام معماری، UFA در گام طراحی، پیاده‌سازی با AspectJ و آزمایش آن با آزمون بازگشتی باعث تولید سیستمی بحرانی-ایمن با اتکاپذیری نسبتاً بالا خواهد شد.



شکل (۴): راهنمای پیشنهادی برای توسعهٔ نرم‌افزار جنبه‌گرا

۵. خلاصه و نتیجه‌گیری

در این مقاله، مروری سیستماتیک بر مهندسی نرم‌افزار جنبه‌گرا با محتوای تأثیر این رویکرد در صنعت، معرفی روش‌ها به تفکیک گام‌های توسعهٔ نرم‌افزار و بررسی چالش‌های حوزه ارائه شد. مفاهیم این حوزه به‌طور صریح بیان و نشان داده شد

1. McCall's Quality Model

که مهندسی نرم‌افزار جنبه‌گرا به‌منظور کاهش پیچیدگی نرم‌افزار و پیمانه‌بندی آن به جداسازی دغدغه‌ها می‌پردازد. دیدیم فرایند توسعهٔ جنبه‌گرا، برخلاف آنچه پنداشته می‌شود، تنها به گام پیاده‌سازی نرم‌افزار محدود نیست، بلکه به‌صورت یک چتر همهٔ گام‌های تولید را از نقطهٔ شروع به جمع‌آوری نیازمندی‌ها پوشش می‌دهد؛ به همین دلیل به بررسی گام‌های این رویکرد پرداخته شد و روش‌های مطرح در هر گام معرفی و دیدی کلی از آن‌ها تشریح شد. ساخت سیستم‌های باکیفیت از اهداف اصلی همهٔ مهندسان نرم‌افزار است، اما کمبود اطلاعات کافی در حوزهٔ مهندسی نرم‌افزار جنبه‌گرا باعث سردرگمی آن‌ها و در نتیجه تولید سیستمی با کیفیت پایین و غیر استاندارد می‌شود؛ به همین سبب برای انتخاب روش بهینه به‌منظور توسعهٔ نرم‌افزار جنبه‌گرا، به مقایسهٔ تطبیقی روش‌های مختلف در هر گام از فرایند توسعه پرداخته شد. در پایان هم مباحثه‌ای دربارهٔ ارزیابی‌های انجام‌شده بر روی روش‌ها صورت گرفت و یک جدول راهنما به‌منظور انتخاب روش مناسب در هر گام برای سیستم مورد نظر ارائه شد.

مراجع

[1] Reis, *R. Q., Reis, C. A. L., Schlebbe, H., Nunes, D. J., "Towards an Aspect-Oriented Approach to Improve the Reusability of Software Process Models*", In Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, AOSD, University of Twente, Enschede, The Netherlands, 2002.

[2] Abran, A., *Software Metrics and Software Metrology*, John Wiley & Sons, 2010.

[3] Tekinerdogan, B., "*Introducing a Graduate Course on Aspect-Oriented Software Development*", International Journal of Engineering Education, vol. 21, No. 2, pp. 361-368, 2005.

[4] Chitchyan, R., Rashid, A., Sawyer, P., Garcia, A., Pinto Alarcon, M., Bakker, J., Tekinerdogan, B., Clarke, S., Jackson, A., "*Survey of Aspect-Oriented Analysis and Design Approaches*", Technical Report, AOSD-Europe-ULANC-9, Fort Collins, Colorado, USA, 2015.

[5] Amar, M., Shabbir, K., "*Systematic Review on Testing Aspect-Oriented Programs: Challenges, Techniques and Their Effectiveness*", Master Thesis, Software Engineering, School of Engineering, Blekinge Institute of Technology, Sweden, 2008.

[6] Mens, K., Tourwé, T., "*Evolutionary Problems in Aspect-Oriented Software Development*", Third International ERCIM Symposium on Software Evolution, Electronic Communications of the EASST, pp. 1-10, 2007.

[7] Khaari, M., Ramsin, R., "*Process Patterns For Aspect-Oriented Software Development*", In Engineering of Computer Based Systems (ECBS), 17th IEEE International Conference and Workshops on, IEEE, pp. 241-250, 2010.

[8] Chitchyan, R., Rashid, A., Sawyer, P., "*Comparing Requirement Engineering Approaches for Handling Crosscutting Concerns*", 8th Workshop on Requirements Engineering, pp. 1-12, 2005.

[9] Budgen, D., *Software Design*, Pearson Education, 2nd Edition, 2003.

[10] Mahoney, M., Bader, A., Elrad, T., Aldawud, O., "*Using Aspects to Abstract and Modularize Statecharts*", Workshop on Aspect-Oriented Modeling, UML, 2005.

[11] Dessì, M., spring 2.5 Aspect Oriented Programming, Packt Publishing Ltd, 2009.

[12] Boticki, I., Katic, M., Martin, S., "*Exploring the Educational Benefits of Introducing Aspect-Oriented Programming into a Programming Course*", Education, IEEE Transactions on, Vol. 56, No. 2, pp. 217-226, 2013.

[13] Baniassad, E., Clarke, S., "*Theme: an Approach for Aspect-Oriented Analysis and Design*", In Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, pp. 158-167, 2004.

[۱۴] فخرایی، س. ش.، میریان حسین آبادی، س. ح.، «**استخراج جنبه از مدل طراحی نرم‌افزار**»، دوازدهمین کنفرانس بین‌المللی انجمن کامپیوتر ایران، دانشگاه شهید بهشتی، تهران، ۱۳۸۵.

[15] Mens, T., Mens, K., Tourwé, T., "*Aspect-Oriented Software Evolution*", ERCIM News 58, pp. 36-37, 2004.

[16] Sant'Anna, C., Garcia, A., Chavez, C., Lucena, C., Von Staa, A., "*On the Reuse and Maintenance of Aspect-Oriented Software: an Assessment Framework*", In Proceedings of Brazilian

Symposium on Software Engineering, pp. 19-34, 2003.

[17] Bagherzadeh, M., Leavens, G. T., Dyer, R., "*Applying Translucid Contracts for Modular Reasoning about Aspect and Object Oriented Events*", In Proceedings of the 10th International Workshop on Foundations of Aspect-Oriented Languages, ACM, pp. 31-35, 2011.

[18] Clifton, C., Leavens, G. T., "*Observers and Assistants: A Proposal for Modular Aspect-Oriented Reasoning*", Technical Report, In FOAL Workshop, pp. 328-342, 2002.

[19] Kienzle, J., Al Abed, W., Fleurey, F., Jézéquel, J. M., Klein, J., "*Aspect-oriented Design with Reusable Aspect Models*", In Transactions on Aspect-Oriented Software Development VII, Series Lecture Notes in Computer Science, Springer Berlin Heidelberg, Vol. 6210, pp. 272-320, 2010.

[20] Al Abed, W., Kienzle, J., "*Information Hiding and Aspect-Oriented Modeling*", In 14th Aspect-Oriented Modeling Workshop, Denver, CO, USA, pp. 1-6, 2009.

[21] Chitchyan, R., Rashid, A., Rayson, P., Waters, R., "*Semantics-Based Composition for Aspect-Oriented Requirements Engineering*", In Proceedings of the 6th International Conference on Aspect-Oriented Software Development, ACM, pp. 36-48, 2007.

[22] Chitchyan, R., Khan, S. S., Rashid, A., "*Modelling and Traceability of Composition Semantics in Requirements*", In Early Aspects Workshop at AOSD, pp. 139–151, 2006.

[23] Chitchyan, R., Sampaio, A., Rashid, A., Rayson, P., "*A Tool Suite for Aspect-Oriented Requirements Engineering*", In Proceedings of International Workshop on Early Aspects at ICSE, ACM, pp. 19-26, 2006.

[24] Yu, Y., Leite, J. C., Mylopoulos, J., "*From Goals to Aspects: Discovering Aspects from Requirements Goal Models*", In Requirements Engineering Conference, Proceedings. 12th IEEE International, IEEE, pp. 38-47, 2004.

[25] Jacobson, I., "*Use Cases and Aspects–Working Seamlessly Together*", Journal of Object Technology, Published by ETH Zurich, Chair of Software Engineering, Vol. 2, No. 4, pp. 7-28, 2003.

[26] Whittle, J., Araújo, J., "*Scenario Modelling with Aspects*", In Software, IEE Proceedings, Vol. 151, No. 4, pp. 157-171, 2004.

[27] Araújo, J., Moreira, A. M., "*An Aspectual Use-Case Driven Approach*", In JISBD, pp. 463-468, 2003.

[28] Sutton Jr, S. M., Rouvellou, I., "*Applicability of Categorization Theory to Multidimensional Separation of Concerns*", In Proceedings of the Workshop on Advanced Separation of Concerns, OOPSLA, Tampa, Florida, 2001.

[29] Sutton Jr, S. M., Rouvellou, I., "*Concern Modeling for Aspect-Oriented Software Development*", Aspect-Oriented Software Development 1, pp. 479-505, 2004.

[30] Sutton Jr, S. M., "*Concerns in a Requirements Model-A Small Case Study*", In Early Aspects Workshop: Aspect-Oriented Requirements Engineering and Architecture Design (held with AOSD), Boston, USA. 2003.

[31] Tarr, P., Ossher, H., Harrison, W., Sutton Jr, S. M., "*N Degrees of Separation: Multi-Dimensional Separation of Concerns*", In Proceedings of the 21st International Conference on Software Engineering, Los Angeles, California, USA, pp. 107-119, 1999.

[32] Moreira, A., Rashid, A., Araujo, J., "*Multi-Dimensional Separation of Concerns in Requirements Engineering*", In Requirements Engineering, Proceedings, 13th IEEE International Conference on, pp. 285-296, 2005.

[33] Moreira, A., Araújo, J., Rashid, A., "*A Concern-Oriented Requirements Engineering Model*", In Advanced Information Systems Engineering, Springer Berlin Heidelberg, pp. 293-308, 2005.

[34] Grundy, J., "*Aspect-Oriented Requirements Engineering for Component-Based Software Systems*", In Requirements Engineering, IEEE International Symposium on, pp. 84-91, 1999.

[35] Grundy, J., "*Multi-Perspective Specification, Design and Implementation of Software Components Using Aspects*", International Journal of Software Engineering and Knowledge Engineering, Vol. 10, No. 06, pp. 713-734, 2000.

[36] Baniassad, E., Clarke, S., "*Finding Aspects in Requirements with Theme/Doc*", Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 2004.

[37] Brichau, J, Chitchyan, R., Rashid, A., D'Hondt, T., "*Aspect-Oriented Software Development: an Introduction*", Wiley Encyclopedia of Computer Science and Engineering, 2008.

[38] Kandé, M. M., "*A Concern-Oriented Approach to Software Architecture*", Ph.D. Thesis, Technische University of Berlin, pp. 85-133, 2003.

[39] Pinto, M., Fuentes, L., Troya, J. M., "*DAOP-ADL: An Architecture Description Language for Dynamic*

Component and Aspect-Based Development", In Generative Programming and Component Engineering, Springer Berlin Heidelberg, pp. 118-137, 2003.

[40] Garcia, A., "*From Objects to Agents: An Aspect-Oriented Approach*", Ph.D. Thesis, PUC-Rio, Rio de Janeiro, Brazil, 2004.

[41] Kulesza, U., Garcia, A., Lucena, C., "*Generating Aspect-Oriented Agent Architectures*", In Proceedings of the 3rd Workshop on Early Aspects, AOSD, Lancaster UK, 2004.

[42] Kulesza, U., Garcia, A., Lucena, C., "*Towards A Method for the Development of Aspect-Oriented Generative Approaches*", In Approaches", Workshop on Early Aspects, OOPSLA'04, 2004.

[43] Barais, O., Cariou, E., Duchien, L., Pessemier, N., Seinturier, L., "*Transat: A Framework for the Specification of Software Architecture Evolution*", Proc. of WCAT 4, pp. 31-38, 2004.

[44] Tekinerdogan, B., "*ASAAM: Aspectual Software Architecture Analysis Method*", In Software Architecture, WICSA, Proceedings. Fourth Working IEEE/IFIP Conference on, IEEE, pp. 5-14, 2004.

[45] Kienzle, J., Gray, J., Stein, D., Cazzola, W., Aldawud, O., Elrad, T., "*11th International Workshop on Aspect-Oriented Modeling*", In Models in Software Engineering, Springer Berlin Heidelberg, pp. 1-6, 2008.

[46] Loughran, N., Coulson, G., Seinturier, L., Pawlak, R., Truyen, E., Sanen, F., Hatton, N., "*Requirements and Definition of AO Middleware Reference Architecture*", Technical Report, AOSD-Europe Report, pp. 551-557, 2005.

[47] Przybylek, A., "*Separation of Crosscutting Concerns at the Design Level: An Extension to the UML Metamodel*", In Computer Science and Information Technology, IMCSIT, International Multiconference on, IEEE, pp. 551-557, 2008.

[48] Stein, D., Hanenberg, S., Unland, R., "*A UML-Based Aspect-Oriented Design Notation for AspectJ*", In Proceedings of the 1st International Conference on Aspect-Oriented Software Development, ACM, pp. 106-112, 2002.

[49] Aldawud, O., Bader, A., Elrad, T., "*Weaving with Statecharts*", In Proceedings of the Workshop on Aspect-Oriented Modeling with UML, pp. 5-8. 2002.

[50] France, R., Ray, I., Georg, Ghosh, S., "*Aspect-Oriented Approach to Early Design Modelling*", IEE Proceedings-Software 151, No. 4, pp. 173-185, 2004.

[51] Wagelaar, D., Bergmans, L, "*Using a Concept-Based Approach to Aspect-Oriented Software Design*", In Workshop on Aspect Oriented Design-Identifying, Separating and Verifying Concerns in the Design, AOSD, pp. 1-10, 2002.

[52] Suzuki, j., Yamamoto, Y, "*Extending UML with Aspects: Aspect Support in the Design Phase*", In ECOOP Workshops, Vol. 1743, pp. 299-300, 1999.

[53] Katara, M., "*Superposing UML Class Diagrams*", In AOSD, 1st Workshop on Aspect-Oriented Modeling with UML (AOM1), University of Twente, Enschede, The Netherlands, 2002.

[54] Groher, I., Baumgarth, T., "*Aspect-Orientation from Design to Code*", Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, pp. 63-69, 2004.

[55] Herrero, J. L., Sánchez, F., Lucio, F., Toro, M., "*Introducing Separation of Aspects at Design Time*", In Proc. of Aspects and Dimensions of Concerns Workshop, 2000.

[56] Shingo, K., Arai, M., Matsumoto, N., Yoshida, N., "*Aspect-Oriented Implementation of Concurrent Processing Design Patterns*", In ICCGI 2014, The Ninth International Multi-Conference on Computing in the Global Information Technology, pp. 146-150, 2014.

[57] Masuhara, H., Kiczales, G., Dutchyn, C., "*Compilation Semantics of Aspect-Oriented Programs*", In Foundations of Aspect-Oriented Languages Workshop, pp. 17-25, 2002.

[58] Kramer, M. E., Kienzle, J., "*Mapping Aspect-Oriented Models to Aspect-Oriented Code*", In Models in Software Engineering, Springer Berlin Heidelberg, pp. 125-139, 2011.

[59] Rashid, A., "*Weaving Aspects in a Persistent Environment*", ACM Sigplan Notices, Vol. 37, No. 2, pp. 36-44, 2002.

[60] Hovsepyan, A., Scandariato, R., Van Baelen, S., Berbers, Y., Joosen, W., "*From Aspect-Oriented Models to Aspect-Oriented Code?: the Maintenance Perspective*", In Proceedings of the 9th International Conference on Aspect-Oriented Software Development, ACM, pp. 85-96, 2010.

[61] Rashid, A., Loughran, N., "*Relational Database Support for Aspect-Oriented Programming*", In Objects, Components, Architectures, Services, and Applications for a Networked World, Springer Berlin Heidelberg, pp. 233-247, 2003.

[62] Aotani, T., Toyama, M., Masuhara, H., "*Supporting*

Covariant Return Types and Generics in Type Relaxed Weaving", In Proceedings of the 10th International Workshop on Foundations of Aspect-Oriented Languages, ACM, pp. 25-29, 2011.

[63] Brichau, J., Haupt, M., Leidenfrost, N., Rashid, A., Bergmans, L., Staijen, T., Ostermann, K., "*Survey of Aspect-Oriented Languages and Execution Models*", European Network of Excellence in AOSD, Chicago, USA, 2005.

[64] Alam, F. E., Evermann, J., Fiech, A., "*Modeling for Dynamic Aspect-Oriented Development*", In Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering, ACM, pp. 143-147, 2009.

[65] Chibani, M., Belattar, B., Bourouis, A., "*Towards a UML Meta Model Extension for Aspect-Oriented Modeling*", In ICSEA, the Eighth International Conference on Software Engineering Advances, pp. 591-596, 2013.

[66] Parizi, R. M., Ghani, M. M., "*A Survey on Aspect-Oriented Testing Approaches*", In Computational Science and its Applications, ICCSA, International Conference on, IEEE, pp. 78-85, 2007.

[67] Singh, P. K., Sangwan, O. P., Pratap, A., Singh, A. P., "*Testability Assessment of Aspect Oriented Software Using Multicriteria Decision Making Approaches*", World Applied Sciences Journal, Vol. 32, No. 4, pp. 718-730, 2014.

[68] Singh, P. K, Sangwan, O. P., Singh, A. P., Pratap, A., "*An Assessment of Software Testability using Fuzzy Logic Technique for Aspect-Oriented Software*", International Journal of Information Technology and Computer Science, Hong Kong, Vol. 7, No.3, pp. 18-26, 2015.

[69] Zhou, Y., Ziv, H., Richardson, D. J., "*Towards A Practical Approach to Test Aspect-Oriented Software*", SOQUA/TECOS 58, pp. 1-16, 2004.

[70] Badri, M., Badri, L., Bourque-Fortin, M., "*Automated State-Based Unit Testing for Aspect-Oriented Programs: A Supporting Framework*", Journal of Object Technology, Vol. 8, No. 3, pp. 121-126, 2009.

[71] Delamare, R., Baudry, B., Ghosh, S., Le Traon, Y., "*A Test-Driven Approach to Developing Pointcut Descriptors in AspectJ*", In Software Testing Verification and Validation, ICST'09, International Conference on, IEEE, pp. 376-385, 2009.

[72] Xu, G., "*A Regression Tests Selection Technique for Aspect-Oriented Programs*", In Proceedings of the 2nd Workshop on Testing Aspect-Oriented Programs, ACM, pp. 15-20, 2006.

[73] Alexander, R. T., Bieman, J. M., Andrews, A. A., "*Towards the Systematic Testing of Aspect-Oriented Programs*", Technical Report, Colorado State University, 2004.

[74] Wang, P., Zhao, X., "*The Research of Automated Select Test Cases for Aspect-oriented Software*", IERI Procedia, Vol. 1, pp. 2-7, 2012.

[75] Bernardi, M. L., Di Lucca, G. A., "*Testing Aspect-Oriented Programs: An Approach Based on the Coverage of the Interactions among Advices and Methods*", 6th International Conference on the Quality of Information and Communications Technology, pp. 65 – 76, 2007.

[76] Ghani, A. A. A., "*Towards Semantic Mutation Testing of Aspect-Oriented Programs*", Journal of Software Engineering and Applications, Vol. 6, No. 10, pp. 5-13, 2013.

[77] Van Deursen, A., Marin, M., Moonen, L., "*A Systematic Aspect-Oriented Refactoring and Testing Strategy, and Its Application to jhotdraw*", Technical Report, arXiv preprint cs/0503015, 2005.

[78] Parizi, R. M., Ghani, A. A. A., Abdullah, R., Atan, R., "*On the Applicability of Random Testing for Aspect-Oriented Programs*", International Journal of Software Engineering and Its Applications, Vol. 3, No. 4, pp. 1-20, 2009.

[79] Fang, H., Lu, W., Wu, F., Zhang, Y., Shang, X., Shao, J., Zhuang, Y., "*Topic Aspect-Oriented Summarization via Group Selection*", Neurocomputing, Vol. 149, pp. 1613-1619, 2014.

[80] Jack Nogueira Santos, F., Cappelli, C., Santoro, F. M., Cesar Sampaio do Prado Leite, J., Vasconcelos Batista, T., "*Aspect-Oriented Business Process Modeling: Analyzing Open Issues*", Business Process Management Journal, Vol. 18, No. 6, pp. 964-991, 2012.

[81] Putrycz, E., & Bernard, G., "*Using Aspect-Oriented Programming to Build A Portable Load Balancing Service*", In Distributed Computing Systems Workshops, Proceedings, 22nd International Conference on, IEEE, pp. 473-478, 2002.

[82] Medeiros, B., Sobral, J. L. F., "*An Aspect-Oriented Approach to Fault-Tolerance in Grid Platforms*", 5th Iberian Grid Infrastructure Conference, 2011.

[83] Afonso, F., Silva, C., Brito, N., Montenegro, S., Tavares, A., "*Aspect-Oriented Fault Tolerance for Real-Time Embedded Systems*", In Proceedings of AOSD workshop on Aspects, components, and patterns for infrastructure software, ACM, 2008.

[84] Kats, L. C., Visser, E., "*Encapsulating Software Platform Logic by Aspect-Oriented Programming: A Case Study in Using Aspects for Language Portability*", In Source Code Analysis and Manipulation (SCAM), 10th IEEE Working Conference on, IEEE, pp. 147-156, 2010.

[85] Al Abed, W., Kienzle, J., "*Aspect-Oriented Modelling for Distributed Systems*", In Model Driven Engineering Languages and Systems, Springer Berlin Heidelberg, Vol. 6981, pp. 123-137, 2011.

[86] Kumar, P., "*Aspect-Oriented Software Quality Model: The AOSQ Model*", Advanced Computing: An International Journal (ACIJ), Vol. 3, No. 2, 2012.