KNO-1301-5104

ارائه الگوهای طراحی جنبه‌گرا برای بهبود تحمل‌پذیری خطای نرم‌افزار با رویکردی مبتنی بر کیفیت نرم‌افزار و معماری‌های نوین

مریم اسداله زاده کرمانشاهی m_asadolahzade@aut.ac.ir
دانشکده مهندسی کامپیوتر و فن آوری اطلاعات،دانشگاه صنعتی امیرکبیر، تهران، ایران

چکیده

تحمل‌پذیری خطا یکی از مهم‌ترین ویژگی‌های کیفی سیستم‌های نرم‌افزاری بحرانی محسوب می‌شود که نقش اساسی در افزایش قابلیت اطمینان، دسترس‌پذیری و پایداری سامانه‌ها ایفا می‌کند. پیاده‌سازی مکانیزم‌های تحمل‌پذیری خطا در روش‌های سنتی توسعه نرم‌افزار، به‌ویژه برنامه‌نویسی شیءگرا، معمولاً موجب افزایش پیچیدگی طراحی، کاهش قابلیت نگهداری و کاهش قابلیت استفاده مجدد اجزای نرم‌افزاری می شود. برنامه‌نویسی جنبه‌گرا به‌عنوان رویکردی نوین در مهندسی نرم‌افزار، امکان جدا سازی دغدغه‌های متقاطع را فراهم کرده و می‌تواند بسیاری از محدودیت‌های روش‌های سنتی را برطرف سازد.

در این پژوهش، دو تاکتیک مهم تحمل‌پذیری خطا شامل بلوک‌های بازیابی و برنامه‌نویسی چندنسخه‌ای با استفاده از رویکرد برنامه‌نویسی جنبه‌گرا مدل سازی شده‌اند. برای هر تاکتیک، یک الگوی طراحی جنبه‌گرا ارائه گردیده و میزان تأثیر آن بر شاخص‌های کیفیت نرم‌افزار مورد ارزیابی قرار گرفته است. همچنین کاربرد این الگوها در معماری‌های نوین مبتنی بر میکروسرویس‌ها و چارچوب‌های AspectJ و Spring AOP بررسی شده است.نتایج ارزیابی نشان می‌دهد که استفاده از برنامه‌نویسی جنبه‌گرا موجب بهبود جدا سازی دغدغه‌ها، افزایش قابلیت نگهداری، افزایش قابلیت استفاده مجدد و کاهش وابستگی میان اجزای سیستم می شود. همچنین استفاده از الگوهای پیشنهادی در سامانه‌های توزیع‌شده و مبتنی بر میکروسرویس‌ها می‌تواند نقش مؤثری در افزایش تاب‌آوری و قابلیت اطمینان نرم‌افزار داشته باشد.

واژگان کلیدی:تحمل‌پذیری خطا، برنامه‌نویسی جنبه‌گرا، Aspect-Oriented Programming، بلوک‌های بازیابی، برنامه‌نویسی چندنسخه‌ای، کیفیت نرم‌افزار، میکروسرویس

۱. مقدمه

با گسترش روزافزون سامانه‌های نرم‌افزاری و وابستگی سازمان‌ها به خدمات دیجیتال، قابلیت اطمینان نرم‌افزار به یکی از مهم‌ترین شاخص‌های کیفیت تبدیل شده است. در بسیاری از سامانه‌های حساس نظیر سامانه‌های بانکی، پزشکی، هوانوردی و زیرساخت‌های حیاتی، وقوع خطا می‌تواند خسارات جبران‌ناپذیری ایجاد نماید.

در سال‌های اخیر، توسعه معماری‌های ابری و میکروسرویس‌ها موجب افزایش پیچیدگی سامانه‌های نرم‌افزاری شده است. در چنین محیط‌هایی، مدیریت خطا و بازیابی از خرابی اهمیت ویژه‌ای پیدا کرده است. یکی از چالش‌های اصلی در پیاده‌سازی مکانیزم‌های تحمل‌پذیری خطا، پراکندگی کدهای مربوط به مدیریت خطا در بخش‌های مختلف سیستم است.

برنامه‌نویسی جنبه‌گرا راهکاری مناسب برای جداسازی این دغدغه‌های متقاطع ارائه می‌دهد و امکان مدیریت مستقل سیاست‌های تحمل‌پذیری خطا را فراهم می‌سازد.

۲,۱. تحمل‌پذیری خطا

تحمل‌پذیری خطا توانایی سیستم در ادامه عملکرد صحیح علی‌رغم وقوع خطا است. هدف اصلی این مفهوم حفظ تداوم سرویس و جلوگیری از شکست کامل سیستم می‌باشد.

مهم‌ترین روش‌های تحمل‌پذیری خطا عبارت‌اند از:

- افزونگی اطلاعاتی
- افزونگی زمانی
- افزونگی سخت‌افزاری
- بلوک‌های بازیابی
- برنامه‌نویسی چندنسخه‌ای

۲-۲. برنامه‌نویسی جنبه‌گرا

برنامه‌نویسی جنبه‌گرا (AOP) با هدف جداسازی دغدغه‌های متقاطع معرفی شد.

مفاهیم اصلی:

- Aspect
- Join Point
- Pointcut
- Advice
- Weaving

این مفاهیم امکان اضافه نمودن رفتارهای جدید به سیستم بدون تغییر مستقیم در منطق اصلی برنامه را فراهم می‌کنند.

۳. مرور پژوهش‌های مرتبط

پژوهش‌های کلاسیک

تحقیقات اولیه در زمینه بلوک‌های بازیابی و برنامه‌نویسی چندنسخه‌ای توسط Brian Randell و Algirdas Avizienis انجام شد. این پژوهش‌ها بنیان نظری تحمل‌پذیری خطای نرم‌افزار را شکل دادند .

پژوهش‌های نوین(2020–2026)

تحقیقات جدید بر موضوعات زیر تمرکز دارند:

- Resilience Engineering
- Cloud Fault Tolerance
- Self-Healing Systems
- Service Mesh Resilience
- Chaos Engineering
- Microservice Fault Isolation

نتایج این مطالعات نشان می‌دهد که جداسازی سیاست‌های مدیریت خطا از منطق کسب‌وکار همچنان یکی از مهم‌ترین چالش‌های سامانه‌های مدرن است.

۴. روش تحقیق

روش تحقیق از نوع کاربردی ـ توسعه‌ای است و شامل مراحل زیر می‌باشد:

۱. تحلیل تاکتیک‌های تحمل‌پذیری خطا
۲. شناسایی دغدغه‌های متقاطع
۳. طراحی الگوهای جنبه‌گرا
۴. پیاده‌سازی نمونه آزمایشی
۵. ارزیابی و مقایسه با رویکرد شیءگرا

۵. الگوی طراحی بلوک‌های بازیابی جنبه‌گرا

در این الگو سازوکار بلوک بازیابی در قالب یک Aspect مستقل پیاده‌سازی می‌شود. در مدل پیشنهادی، جنبه از طریق Pointcut به عملیات اصلی متصل شده و فرآیند اجرای نسخه‌های جایگزین را کنترل می‌کند .

مزایا:

- جداسازی کامل منطق تحمل خطا
- حذف کدهای تکراری
- افزایش قابلیت استفاده مجدد
- سهولت توسعه

۶. الگوی طراحی برنامه‌نویسی چندنسخه‌ای جنبه‌گرا

در این مدل چند پیاده‌سازی مستقل از یک سرویس وجود دارد و نتایج آن‌ها توسط مکانیزم رأی‌گیری ارزیابی می‌شود. سازوکار رأی‌گیری در قالب یک جنبه مستقل پیاده‌سازی می‌شود .

مزایا:

- کاهش وابستگی میان نسخه‌ها
- تمرکز مکانیزم رأی‌گیری در یک واحد مستقل
- افزایش انعطاف‌پذیری طراحی

۷. مطالعه موردی

برای ارزیابی مدل‌ها، برنامه مرتب‌سازی داده‌ها انتخاب شد. سه الگوریتم مختلف شامل:

- Bubble Sort
- Insertion Sort
- Selection Sort

در چهار سناریو پیاده‌سازی شدند:

۱. بلوک بازیابی شیءگرا
۲. بلوک بازیابی جنبه‌گرا
۳. برنامه‌نویسی چندنسخه‌ای شیءگرا
۴. برنامه‌نویسی چندنسخه‌ای جنبه‌گرا

۸.معیارهای ارزیابی

معیارهای استفاده‌شده عبارت‌اند از:

جداسازی دغدغه‌ها

- CDC
- CDO

اتصال

- CBC
- DIT

اندازه برنامه

- VS
- NOA

جدول ۱. مقایسه رویکرد شیءگرا و جنبه‌گرا از نظر معیارهای کیفیت

معیار	رویکرد شیءگرا	رویکرد جنبه‌گرا	میزان بهبود
CDC	مبنا	بهتر	٪15
CDO	مبنا	بهتر	٪15
CBC	مبنا	کمتر	٪45
NOA	مبنا	کمتر	٪27

جدول ۲. مقایسه ویژگی‌های دو تاکتیک تحمل‌پذیری خطا

ویژگی	برنامه‌نویسی چندنسخه‌ای	بلوک‌های بازیابی
افزونگی	زمانی	طراحی
تعداد نسخه‌ها	یک نسخه اصلی و جایگزین	چند نسخه مستقل
مکانیزم تصمیم‌گیری	آزمون پذیرش	رأی‌گیری
هزینه پیاده‌سازی	متوسط	زیاد
قابلیت اطمینان	بالا	بسیار بالا

۹. نتایج

جدول۳ – نتایج مقایسه

معیار	جنبه‌گرا	مبنا	میزان بهبود
جداسازی دغدغه‌ها	مبنا	بهتر	٪15
اتصال بین مؤلفه‌ها	مبنا	کمتر	٪45
اندازه برنامه	مبنا	کمتر	٪27

تحلیل نتایج نشان می‌دهد:

- قابلیت نگهداری افزایش یافته است .
- قابلیت استفاده مجدد افزایش یافته است .
- پیچیدگی طراحی کاهش یافته است .
- فهم سیستم بهبود یافته است .

۱۰.کاربرد در معماری میکروسرویس

در معماری‌های مدرن می‌توان الگوهای پیشنهادی را برای پیاده‌سازی:

- Retry Pattern
- Circuit Breaker
- Bulkhead Pattern
- Fallback Pattern
- Service Mesh Fault Injection

به کار برد.

در این معماری‌هاAspect ها می‌توانند سیاست‌های تحمل خطا را مستقل از سرویس‌های کسب‌وکار مدیریت نمایند.

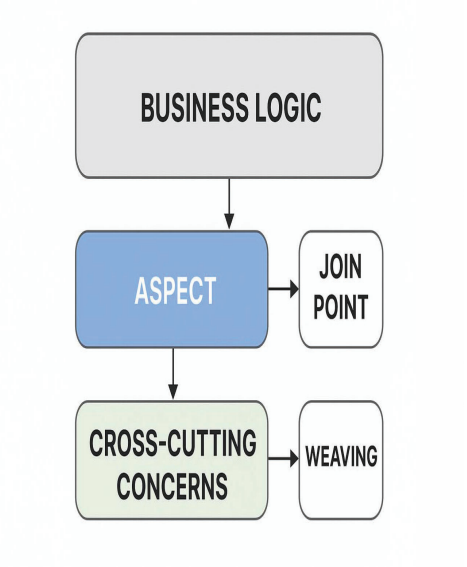
۱۱. مقایسه با فناوری‌های مدرن

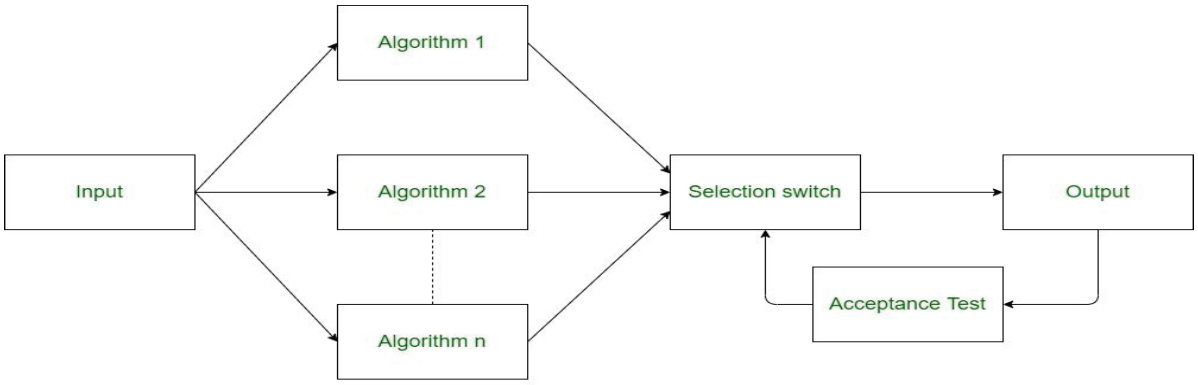
امروزه چارچوب‌های زیر مهم‌ترین ابزارهای جنبه‌گرایی محسوب می‌شوند:

- AspectJ
- Spring AOP

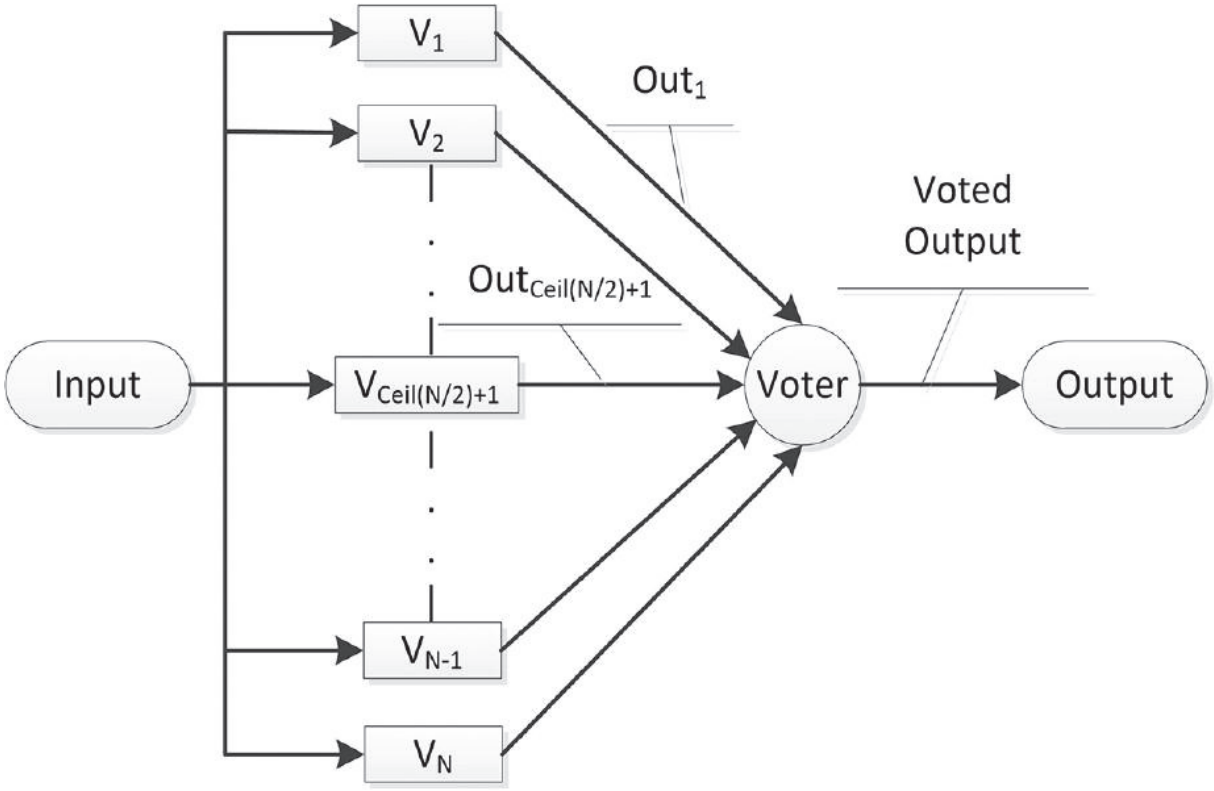
الگوهای ارائه‌شده در این پژوهش قابلیت پیاده‌سازی مستقیم در هر دو چارچوب را دارند.

معماری پیشنهادی سیستم





Recovery Blocks



شامل:

۱. لایه کسب وکار
۲. جنبه تحمل‌پذیری خطا
۳. آزمون پذیرش
۴. نسخه‌های جایگزین
۵. ماژول رأی‌گیری
۶. سرویس‌های میکروسرویسی

۱۲ .نتیجه‌گیری

در این پژوهش دو الگوی طراحی مبتنی بر برنامه‌نویسی جنبه‌گرا برای تاکتیک‌های بلوک بازیابی و برنامه‌نویسی چندنسخه‌ای ارائه و ارزیابی شد. نتایج نشان می‌دهد که مزیت اصلی رویکرد جنبه‌گرا در تمرکز مکانیزم‌های تحمل‌پذیری خطا در یک واحد مستقل است. این موضوع علاوه بر کاهش پراکندگی کد، امکان توسعه موازی تیم‌های نرم‌افزاری را فراهم می‌کند .

رویکرد پیشنهادی موجب بهبود جداسازی دغدغه‌ها، کاهش اتصال بین مؤلفه‌ها و کاهش اندازه برنامه می‌شود. همچنین قابلیت استفاده مجدد، قابلیت نگهداری و فهم‌پذیری سیستم بهبود می‌یابد. بنابراین برنامه‌نویسی جنبه‌گرا می‌تواند گزینه‌ای مناسب برای توسعه سامانه‌های قابل اطمینان در محیط‌های مدرن ابری و میکروسرویسی باشد.

منابع

[1] باس، ل.، کلمن، پ. و کازمن، ر. (۱۳۹۸). معماری نرم‌افزار در عمل. ترجمه محمد رسولی. تهران: دانشگاه صنعتی شریف.

[2] سمیعی، م. و رضایی، ع. (۱۴۰۱). «بررسی روش‌های تحمل‌پذیری خطا در سامانه‌های توزیع‌شده». فصلنامه مهندسی نرم‌افزار ایران، ۱۴(۲)، ۴۵-۵۸.

[3] AspectJ in Action, Manning Publications, 2015.

[4] Software Architecture in Practice, Addison-Wesley, 2021.

[5] Algirdas Avizienis, Laprie, J.C., Randell, B., & Landwehr, C. “Basic Concepts and Taxonomy of Dependable and Secure Computing.” IEEE Transactions on Dependable and Secure Computing, 1(1), 2004.

[6] Garcia, J., Popescu, D., & Medvidovic, N. “Fault Tolerance Patterns in Microservice Architectures.” Journal of Systems and Software, 2021.

[7] Zhang, Y., Li, H., & Wang, X. “Aspect-Oriented Approaches for Resilient Cloud Applications.” Future Generation Computer Systems, 2022.

[8] Kumar, P., Singh, R., & Sharma, V. “Software Fault Tolerance Techniques: A Survey.” ACM Computing Surveys, 2023.

[9] Silva, R., & Costa, P. “Self-Healing Mechanisms in Cloud-Native Systems.” IEEE Access, 2023.

[10] Chen, X., et al. “Resilience Engineering for Distributed Systems.” IEEE Software, 2024.

[11] Ahmed, S., & Rahman, T. “A Comparative Study of AspectJ and Spring AOP for Enterprise Applications.” Software Practice and Experience, 2022.

[12] Li, J., Wang, K., & Zhao, M. “Fault Recovery Strategies in Kubernetes-based Microservices.” Future Internet, 2024.

[13] Brown, T., & Wilson, M. “Design Diversity and N-Version Programming Revisited.” Journal of Systems Architecture, 2021.

[14] Hassan, M. et al. “Chaos Engineering and Fault Injection in Cloud Platforms.” IEEE Cloud Computing, 2025.

[15] Patel, N., & Gupta, R. “Service Mesh Based Fault Tolerance in Microservices.” Software Quality Journal, 2024.

[16] Spring Framework Documentation. Spring AOP Reference Guide, 2025.

[17] Eclipse Foundation. AspectJ Programming Guide, 2025.

[18] Richardson, C. Microservices Patterns. Manning Publications, 2021.

[19] Newman, S. Building Microservices, 2nd Edition, O’Reilly, 2021.