Group-Assignment 2

Second empirical study: Effect of code bad smells on modularity

GQM Approach

The Goal-Question-Metric (GQM) approach is a systematic and structured method used for defining, measuring, and achieving goals in various domains, including software engineering, project management, and quality assurance. Based upon the assumption that a study to measure in a purposeful way it must first specify the goals, then it must trace those goals to the data that are intended to define those goals.

GQM defines a measurement model on three levels:

Conceptual level (Goal) A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment.

Operational level (Question) A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.

Quantitative level (Metric) A set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

GQM approach, software development teams can systematically evaluate, measure, and address code smells, leading to enhanced code quality, better maintainability and improved readability in the software codebase.

Goal: To investigate the impact of code bad smells on modularity in software projects.

Questions:

Q1: To what extent are design smells subject to change?

Q2: How do these code bad smells affect modularity metrics within the codebase?

Quantitative Impact Analysis:

Q3: Is there a correlation between specific types of code bad smells (e.g., high cyclomatic complexity, large class size) and decreased modularity metrics (e.g., CBO, LCOM)?

Q4: To what extent are design smells removed during the evolution of the system?

Q5: To what extent can smell removal be related to refactoring actions?

Define Metrics:

Bad Smells Metrics:

- Quantify the prevalence and types of code bad smells using tool: JDeodorant.

- Capture data on the frequency and nature of identified bad smells. (Feature Envy, God Class and Long method etc)

Modularity Metrics:

- Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM), other relevant modularity metrics using tools like C&K metrics.

- Obtain modularity metric values for classes within the same project versions analysed for code smells.

Study Design:

The proposed study reports results involving 10 Java open-source projects. These systems have been considered since these systems satisfy the following criteria.

- The programming language is Java.
- The Git repository is active and there is more than one release, there are variations in application domains, sizes, revisions, and the systems are also used by other studies.

Below are the java projects utilized to find out the code bad smells.

1. Free Universal Database Tool and SQL Client (DBeaver)

2. Generate diagrams from textual description (plantuml)

3. Graphs for everyone (neo4j)

4. OpenPDF is a free Java library for creating and editing PDF files with a LGPL and MPL open-source license.

5. Provide support to increase developer productivity in Java when using MongoDB.

6. Apollo is a reliable configuration management system suitable for microservice configuration management scenarios.

7. Selenide- Concise UI Tests with Java!

8. Sublime Packages

9. Netflix/Priam

10. swagger-api/swagger-core

Datasets

The characteristics of the selected projects that are, the number of the total commits analysed for projects and the first and last commit date.

System	Commit	First-Last Commit date

DBeaver	25,020	2015-10-21 | 2023-12-01
Plantuml	1468	2010-11-04 | 2023-11-29
neo4j	78257	2012-11-12 | 2023-10-12
OpenPDF	1389	2016-07-11 | 2023-11-30
spring-data-mongodb	3776	2011-10-13 | 2023-11-30
apollo	2853	2016-03-04 | 2023-11-29
Selenide	5044	2012-02-07 | 2023-11-30
Sublime Packages	4335	2015-06-13 | 2023-11-28
Netflix/Priam	1504	2011-07-20 | 2023-09-22
swagger-api/swagger-core	4252	2011-07-05 | 2023-12-01

Metrics for the projects (By C&K tool)

CK calculates class-level and method-level code metrics in Java projects by means of static analysis (i.e. no need for compiled code).

Project	CBO (MAX)	# of classes	# of Methods
DBeaver	196	8004	45849
Plantuml	104	3511	20679
neo4j	155	12148	80617
OpenPDF	84	904	7115
spring-data-mongodb	177	3037	17093
apollo	36	684	3846
Selenide	80	1062	5394
Sublime Packages	3	16	24
Netflix/Priam	36	476	2176
swagger- api/swagger-core	131	1314	4927

Code smell reference:

JDeodorant is an Eclipse plugin that identifies bad smells, and resolves them by applying appropriate refactoring's. JDeodorant only detects four types of bad smells. They are,

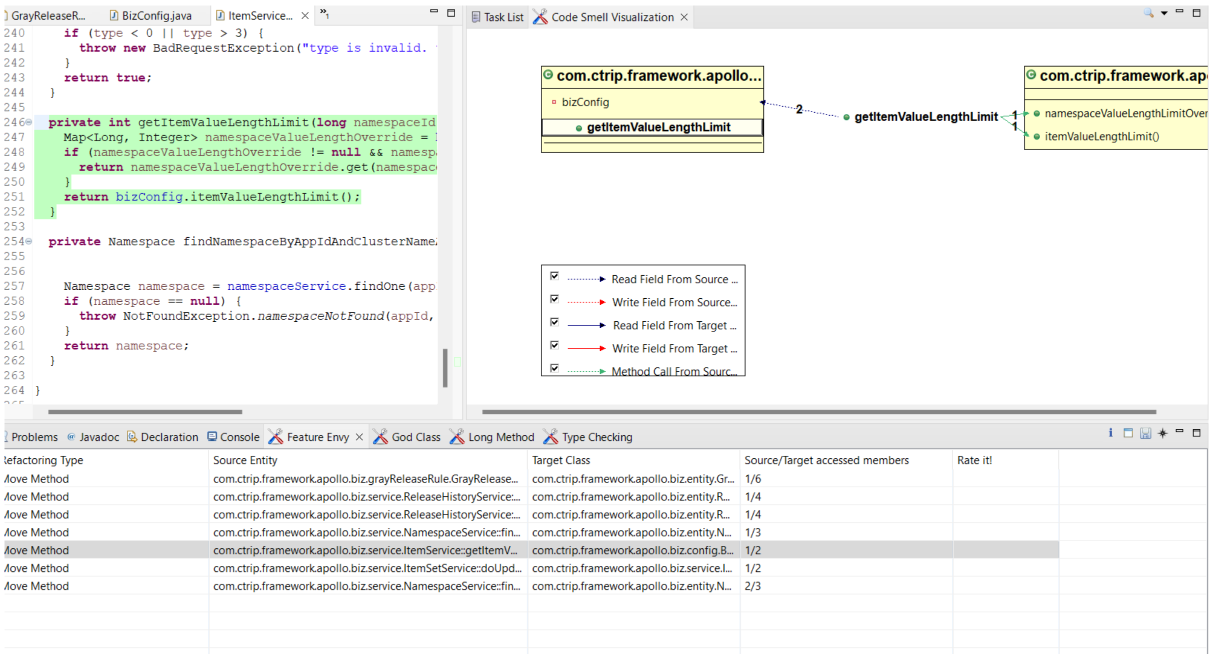
- God Class
- Long method
- Type checking
- Feature Envy

The detection techniques are based in the identification of refactoring opportunities of Extract Class for God Class, Extract Method for God Method and Move Method for Feature Envy.

Project	Feature Envy	God Class	God/Long Method	Total
DBeaver	80	140	261	481
Plantuml	5	45	93	143
OpenPDF	2	9	57	68
spring-data-mongodb	1	5	29	35
apollo	52	73	25	150
Selenide	--	--	--	
Sublime Packages	n/a	n/a	n/a	
Netflix/Priam	N/A	N/A	N/A	
swagger- api/swagger-core	4	33	2	39

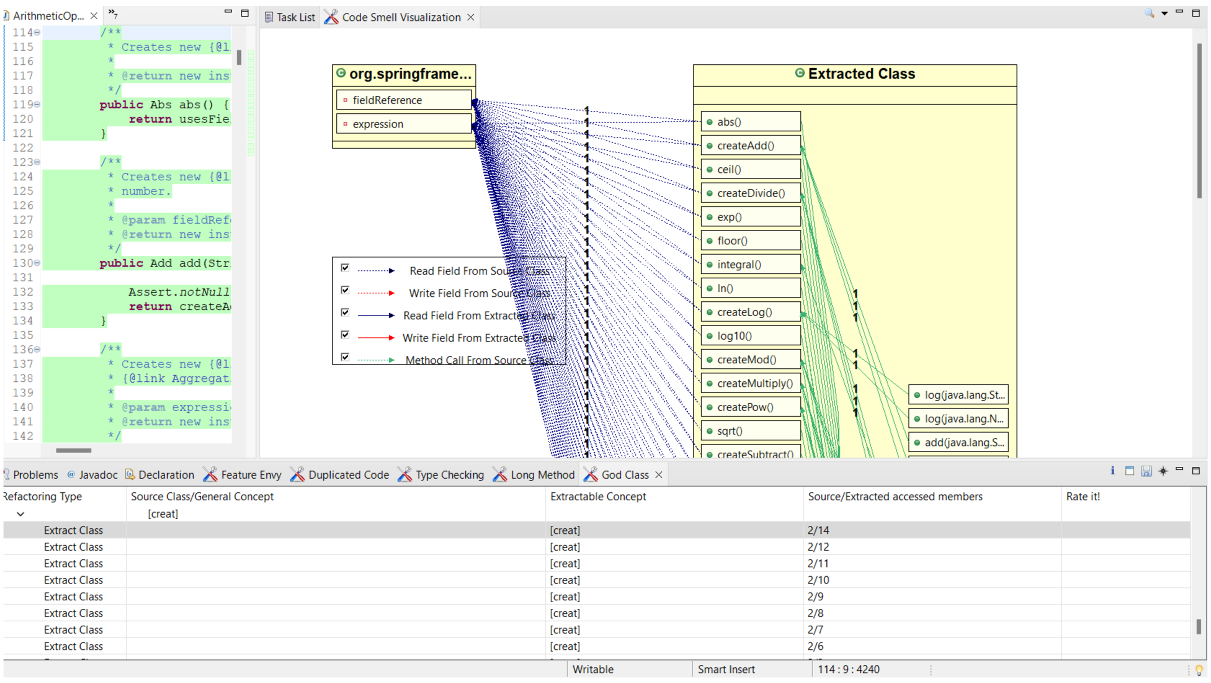
Feature Envy: JDeodorant output showing identified Feature Envy bad smells

Imported the system under study as a Java Project and opened Navigator View in Java Perspective. Then, selected the Bad Smells item in the menu bar and triggers the Feature Envy action, which in turn opens the corresponding view. After pressing the Identify Bad Smells button the Feature Envy view lists both the identified bad smells as well as the candidate refactoring's that resolve them, as shown in the below image



```
    if (type < 0 || type > 3) {
      throw new BadRequestException("type is invalid.
    }
    return true;
  }

  private int getItemValueLengthLimit(long namespaceId
    Map<Long, Integer> namespaceValueLengthOverride = 
    if (namespaceValueLengthOverride != null && namesp
      return namespaceValueLengthOverride.get(namespac
    }
    return bizConfig.itemValueLengthLimit();
  }

  private Namespace findNamespaceByAppIdAndClusterName


    Namespace namespace = namespaceService.findOne(app
    if (namespace == null) {
      throw NotFoundException.namespaceNotFound(appId,
    }
    return namespace;
  }

}
```

Refactoring Type	Source Entity	Target Class	Source/Target accessed members	Rate it!
Move Method	com.ctrip.framework.apollo.biz.grayReleaseRule.GrayRelease...	com.ctrip.framework.apollo.biz.entity.Gr...	1/6	
Move Method	com.ctrip.framework.apollo.biz.service.ReleaseHistoryService:...	com.ctrip.framework.apollo.biz.entity.R...	1/4	
Move Method	com.ctrip.framework.apollo.biz.service.ReleaseHistoryService:...	com.ctrip.framework.apollo.biz.entity.R...	1/4	
Move Method	com.ctrip.framework.apollo.biz.service.NamespaceService::fin...	com.ctrip.framework.apollo.biz.entity.N...	1/3	
Move Method	com.ctrip.framework.apollo.biz.service.ItemService::getItemV...	com.ctrip.framework.apollo.biz.config.B...	1/2	
Move Method	com.ctrip.framework.apollo.biz.service.ItemSetService::doUpd...	com.ctrip.framework.apollo.biz.service.I...	1/2	
Move Method	com.ctrip.framework.apollo.biz.service.NamespaceService::fin...	com.ctrip.framework.apollo.biz.entity.N...	2/3	

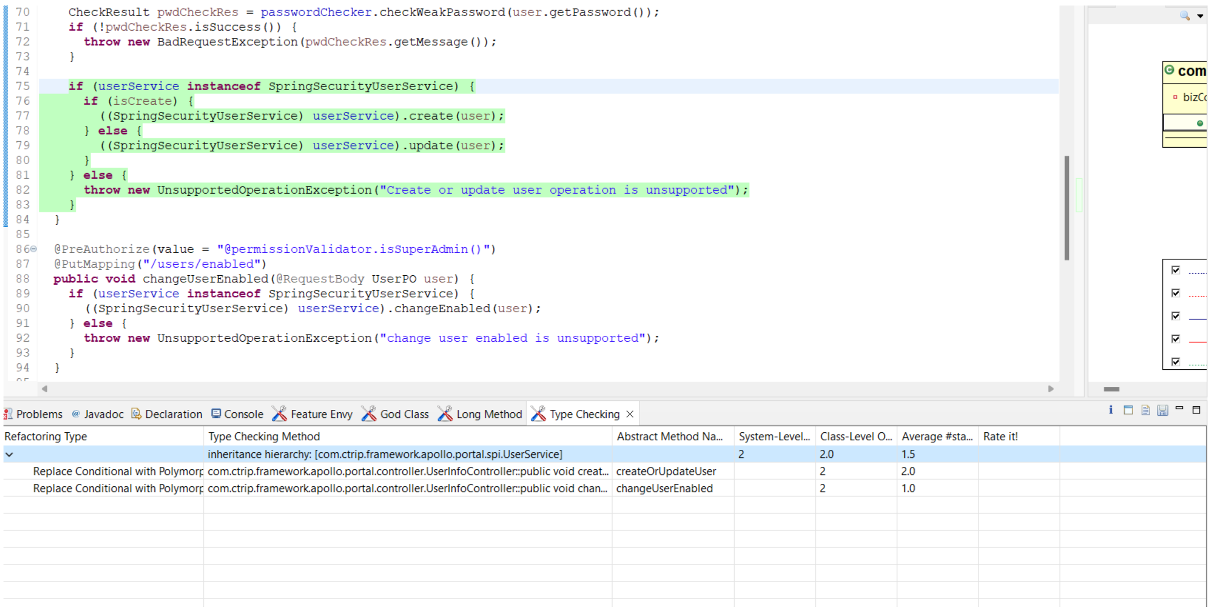


```
    /**
     * Creates new {@l
     *
     * @return new ins
     */
    public Abs abs() {
        return usesFie
    }

    /**
     * Creates new {@l
     * number.
     *
     * @param fieldRef
     * @return new ins
     */
    public Add add(Str

        Assert.notNull
        return createA
    }

    /**
     * Creates new {@l
     * {@link Aggregat
     *
     * @param expressi
     * @return new ins
     */
```

Refactoring Type	Source Class/General Concept	Extractable Concept	Source/Extracted accessed members	Rate it!
	[creat]			
Extract Class		[creat]	2/14	
Extract Class		[creat]	2/12	
Extract Class		[creat]	2/11	
Extract Class		[creat]	2/10	
Extract Class		[creat]	2/9	
Extract Class		[creat]	2/8	
Extract Class		[creat]	2/7	
Extract Class		[creat]	2/6	

Type Checking

Also known as type casting, is a way to check if an object or expression can be safely treated as an instance of a specified subtype. In other words, it checks whether an object of one type can be treated as an object of another type. Select "Type checking" from the list and JDeodorant will analyse your project and display a list of any type checking issues found.

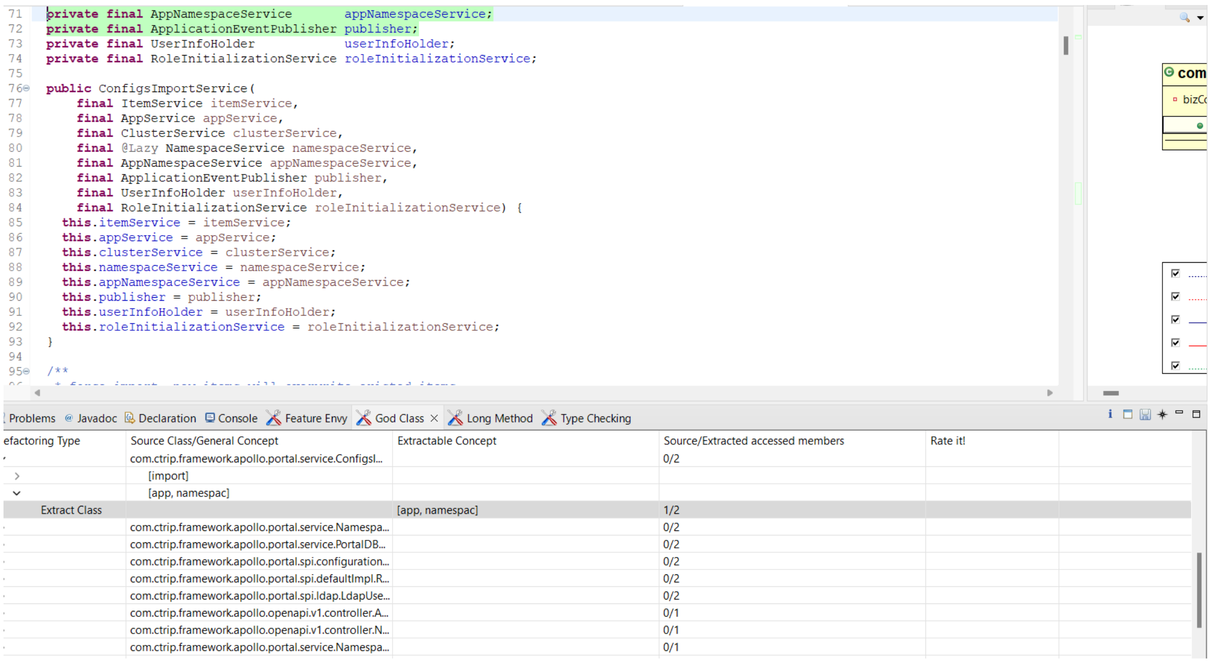


```
    CheckResult pwdCheckRes = passwordChecker.checkWeakPassword(user.getPassword());
    if (!pwdCheckRes.isSuccess()) {
      throw new BadRequestException(pwdCheckRes.getMessage());
    }

    if (userService instanceof SpringSecurityUserService) {
      if (isCreate) {
        ((SpringSecurityUserService) userService).create(user);
      } else {
        ((SpringSecurityUserService) userService).update(user);
      }
    } else {
      throw new UnsupportedOperationException("Create or update user operation is unsupported");
    }
  }

  @PreAuthorize(value = "@permissionValidator.isSuperAdmin()")
  @PutMapping("/users/enabled")
  public void changeUserEnabled(@RequestBody UserPO user) {
    if (userService instanceof SpringSecurityUserService) {
      ((SpringSecurityUserService) userService).changeEnabled(user);
    } else {
      throw new UnsupportedOperationException("change user enabled is unsupported");
    }
  }
```

Refactoring Type	Type Checking Method	Abstract Method Na...	System-Level...	Class-Level O...	Average #sta...	Rate it!
	inheritance hierarchy: [com.ctrip.framework.apollo.portal.spi.UserService]		2	2.0	1.5	
Replace Conditional with Polymorp	com.ctrip.framework.apollo.portal.controller.UserInfoController::public void creat...	createOrUpdateUser		2	2.0	
Replace Conditional with Polymorp	com.ctrip.framework.apollo.portal.controller.UserInfoController::public void chan...	changeUserEnabled		2	1.0	

God Class

Select god class from Bad smell menu item. A view will pop up along side with the console view. Select the project and then click the identification button in the god class view. God Class view table will be populated and The results will be grouped by source classes.

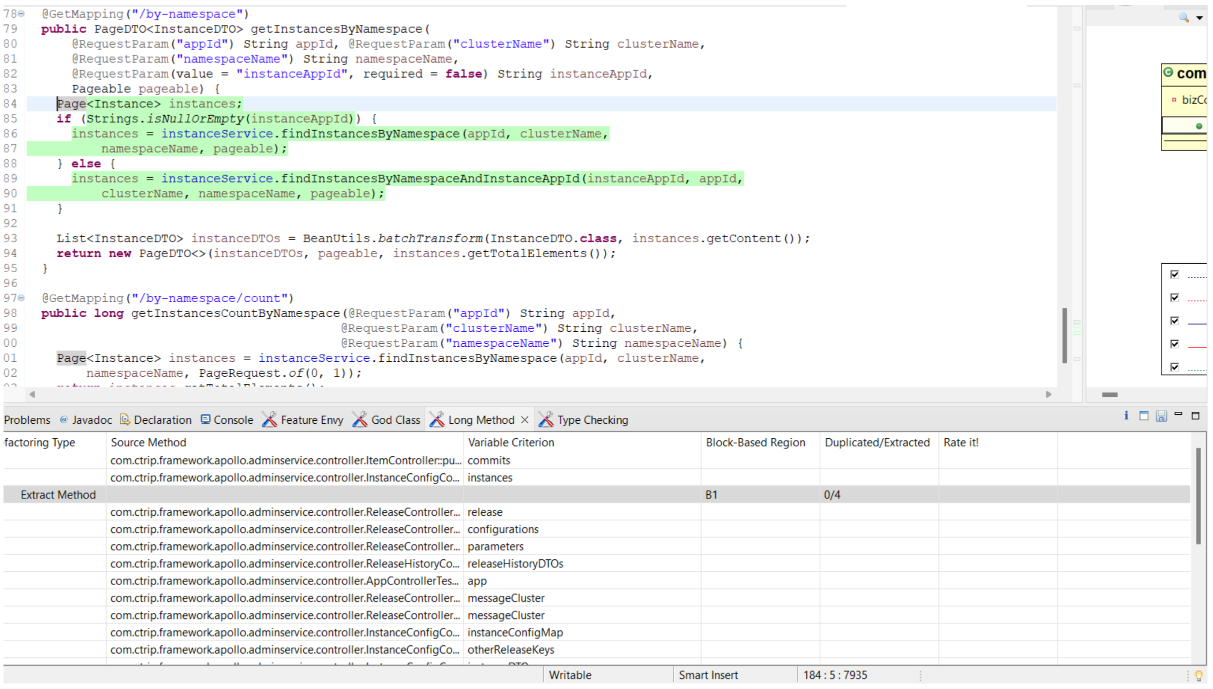


```java
private final AppNamespaceService       appNamespaceService;
private final ApplicationEventPublisher publisher;
private final UserInfoHolder            userInfoHolder;
private final RoleInitializationService roleInitializationService;

public ConfigsImportService(
    final ItemService itemService,
    final AppService appService,
    final ClusterService clusterService,
    final @Lazy NamespaceService namespaceService,
    final AppNamespaceService appNamespaceService,
    final ApplicationEventPublisher publisher,
    final UserInfoHolder userInfoHolder,
    final RoleInitializationService roleInitializationService) {
  this.itemService = itemService;
  this.appService = appService;
  this.clusterService = clusterService;
  this.namespaceService = namespaceService;
  this.appNamespaceService = appNamespaceService;
  this.publisher = publisher;
  this.userInfoHolder = userInfoHolder;
  this.roleInitializationService = roleInitializationService;
}

/**
```

efactoring Type	Source Class/General Concept	Extractable Concept	Source/Extracted accessed members	Rate it!
	com.ctrip.framework.apollo.portal.service.ConfigsI...		0/2	
	[import]			
	[app, namespac]			
Extract Class		[app, namespac]	1/2	
	com.ctrip.framework.apollo.portal.service.Namespa...		0/2	
	com.ctrip.framework.apollo.portal.service.PortalDB...		0/2	
	com.ctrip.framework.apollo.portal.spi.configuration...		0/2	
	com.ctrip.framework.apollo.portal.spi.defaultImpl.R...		0/2	
	com.ctrip.framework.apollo.portal.spi.ldap.LdapUse...		0/2	
	com.ctrip.framework.apollo.openapi.v1.controller.A...		0/1	
	com.ctrip.framework.apollo.openapi.v1.controller.N...		0/1	
	com.ctrip.framework.apollo.portal.service.Namespa...		0/1	

God/Long Method

A long method is a method that has a large number of statements or lines of code. Select "Long method" from the list of available bad smells. JDeodorant will analyse your project and display a list of any long methods found.

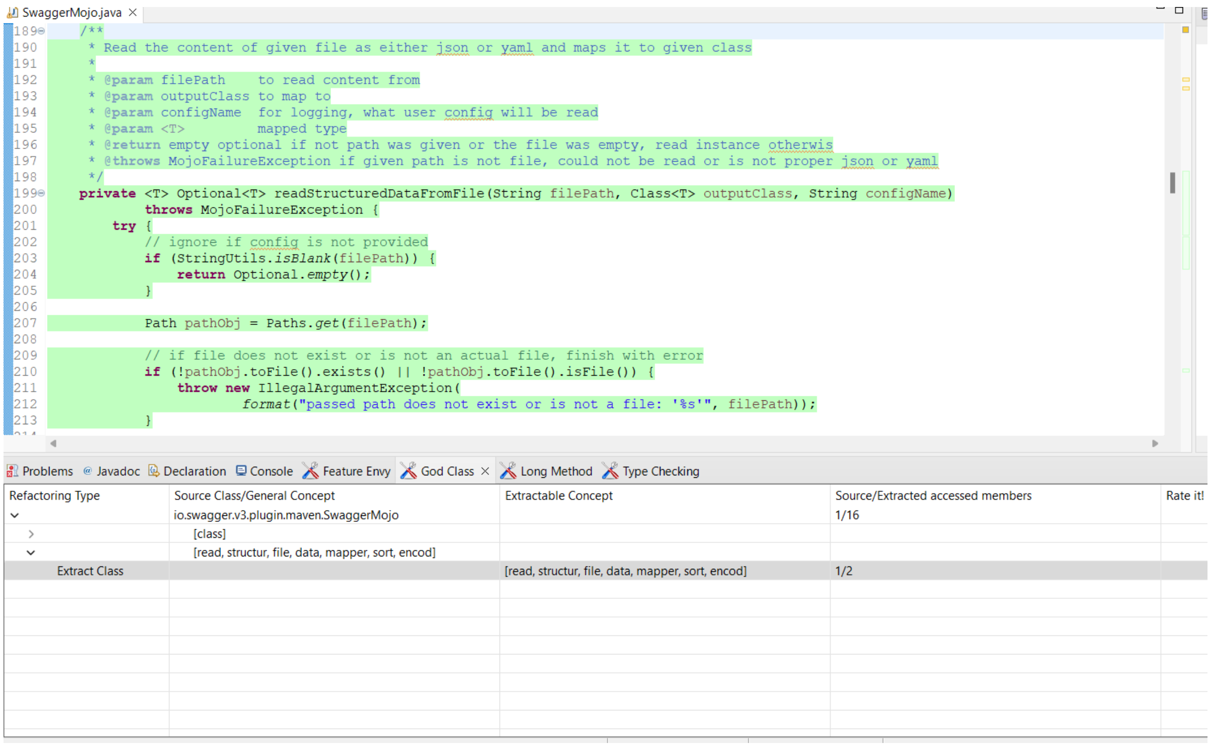


```java
@GetMapping("/by-namespace")
public PageDTO<InstanceDTO> getInstancesByNamespace(
    @RequestParam("appId") String appId, @RequestParam("clusterName") String clusterName,
    @RequestParam("namespaceName") String namespaceName,
    @RequestParam(value = "instanceAppId", required = false) String instanceAppId,
    Pageable pageable) {
  Page<Instance> instances;
  if (Strings.isNullOrEmpty(instanceAppId)) {
    instances = instanceService.findInstancesByNamespace(appId, clusterName,
        namespaceName, pageable);
  } else {
    instances = instanceService.findInstancesByNamespaceAndInstanceAppId(instanceAppId, appId,
        clusterName, namespaceName, pageable);
  }

  List<InstanceDTO> instanceDTOs = BeanUtils.batchTransform(InstanceDTO.class, instances.getContent());
  return new PageDTO<>(instanceDTOs, pageable, instances.getTotalElements());
}

@GetMapping("/by-namespace/count")
public long getInstancesCountByNamespace(@RequestParam("appId") String appId,
                                         @RequestParam("clusterName") String clusterName,
                                         @RequestParam("namespaceName") String namespaceName) {
  Page<Instance> instances = instanceService.findInstancesByNamespace(appId, clusterName,
      namespaceName, PageRequest.of(0, 1));
```

factoring Type	Source Method	Variable Criterion	Block-Based Region	Duplicated/Extracted	Rate it!
	com.ctrip.framework.apollo.adminservice.controller.ItemController::pu...	commits			
	com.ctrip.framework.apollo.adminservice.controller.InstanceConfigCo...	instances			
Extract Method			B1	0/4	
	com.ctrip.framework.apollo.adminservice.controller.ReleaseController...	release			
	com.ctrip.framework.apollo.adminservice.controller.ReleaseController...	configurations			
	com.ctrip.framework.apollo.adminservice.controller.ReleaseController...	parameters			
	com.ctrip.framework.apollo.adminservice.controller.ReleaseHistoryCo...	releaseHistoryDTOs			
	com.ctrip.framework.apollo.adminservice.controller.AppControllerTes...	app			
	com.ctrip.framework.apollo.adminservice.controller.ReleaseController...	messageCluster			
	com.ctrip.framework.apollo.adminservice.controller.ReleaseController...	messageCluster			
	com.ctrip.framework.apollo.adminservice.controller.InstanceConfigCo...	instanceConfigMap			
	com.ctrip.framework.apollo.adminservice.controller.InstanceConfigCo...	otherReleaseKeys			

Writable Smart Insert 184 : 5 : 7935

Swagger – God class



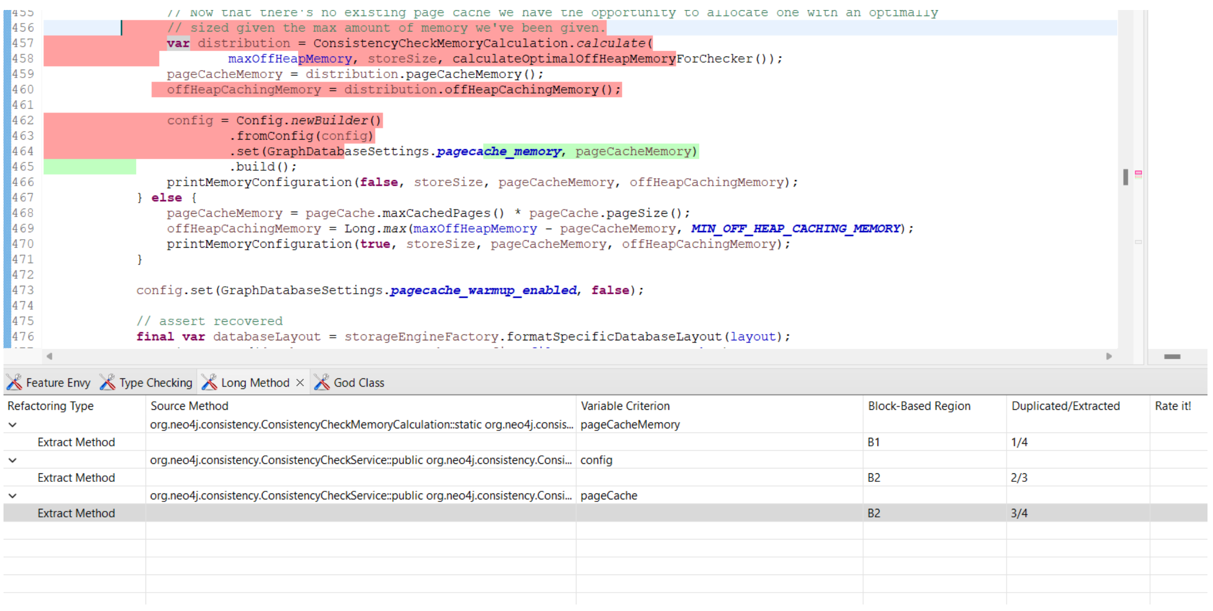
SwaggerMojo.java

```java
    /**
     * Read the content of given file as either json or yaml and maps it to given class
     *
     * @param filePath    to read content from
     * @param outputClass to map to
     * @param configName  for logging, what user config will be read
     * @param <T>         mapped type
     * @return empty optional if not path was given or the file was empty, read instance otherwis
     * @throws MojoFailureException if given path is not file, could not be read or is not proper json or yaml
     */
    private <T> Optional<T> readStructuredDataFromFile(String filePath, Class<T> outputClass, String configName)
            throws MojoFailureException {
        try {
            // ignore if config is not provided
            if (StringUtils.isBlank(filePath)) {
                return Optional.empty();
            }

            Path pathObj = Paths.get(filePath);

            // if file does not exist or is not an actual file, finish with error
            if (!pathObj.toFile().exists() || !pathObj.toFile().isFile()) {
                throw new IllegalArgumentException(
                        format("passed path does not exist or is not a file: '%s'", filePath));
            }
```

Problems | Javadoc | Declaration | Console | Feature Envy | God Class | Long Method | Type Checking

Refactoring Type	Source Class/General Concept	Extractable Concept	Source/Extracted accessed members	Rate it!
	io.swagger.v3.plugin.maven.SwaggerMojo		1/16	
	[class]			
	[read, structur, file, data, mapper, sort, encod]			
Extract Class		[read, structur, file, data, mapper, sort, encod]	1/2	

Refactor with the Jdeodorant

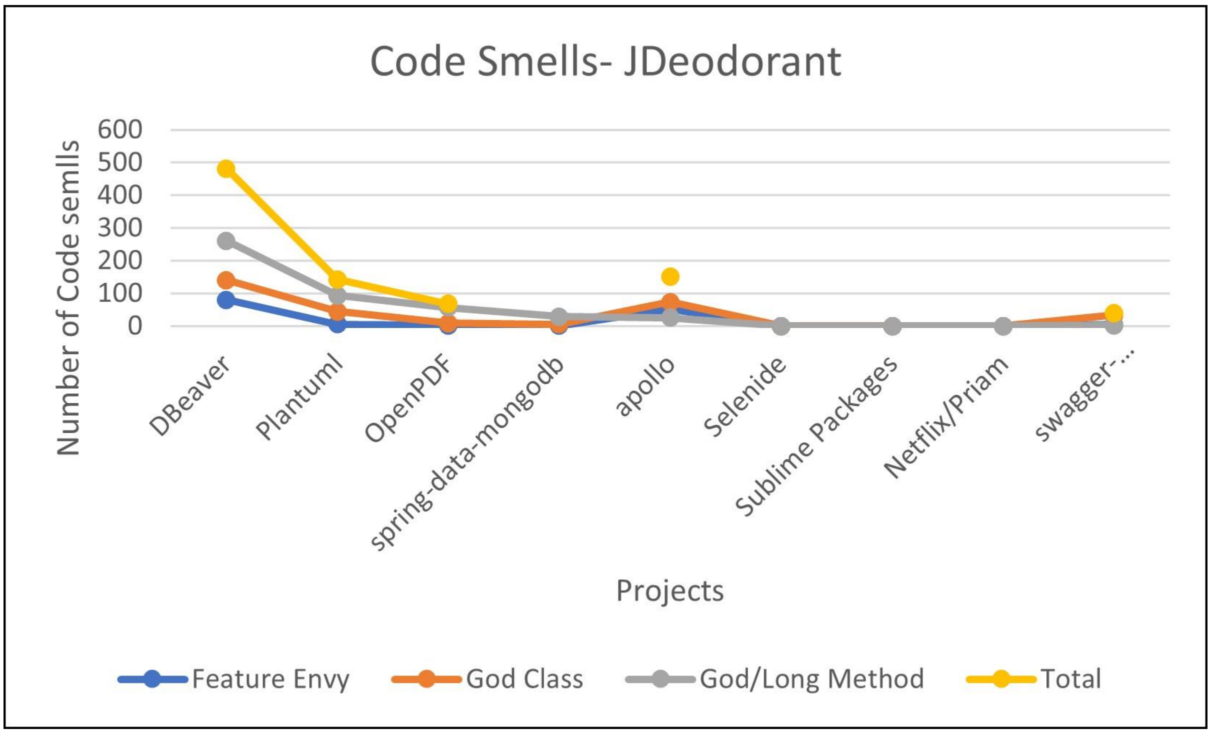


```java
            // Now that there's no existing page cache we have the opportunity to allocate one with an optimally
            // sized given the max amount of memory we've been given.
            var distribution = ConsistencyCheckMemoryCalculation.calculate(
                    maxOffHeapMemory, storeSize, calculateOptimalOffHeapMemoryForChecker());
            pageCacheMemory = distribution.pageCacheMemory();
            offHeapCachingMemory = distribution.offHeapCachingMemory();

            config = Config.newBuilder()
                    .fromConfig(config)
                    .set(GraphDatabaseSettings.pagecache_memory, pageCacheMemory)
                    .build();
            printMemoryConfiguration(false, storeSize, pageCacheMemory, offHeapCachingMemory);
        } else {
            pageCacheMemory = pageCache.maxCachedPages() * pageCache.pageSize();
            offHeapCachingMemory = Long.max(maxOffHeapMemory - pageCacheMemory, MIN_OFF_HEAP_CACHING_MEMORY);
            printMemoryConfiguration(true, storeSize, pageCacheMemory, offHeapCachingMemory);
        }

        config.set(GraphDatabaseSettings.pagecache_warmup_enabled, false);

        // assert recovered
        final var databaseLayout = storageEngineFactory.formatSpecificDatabaseLayout(layout);
```

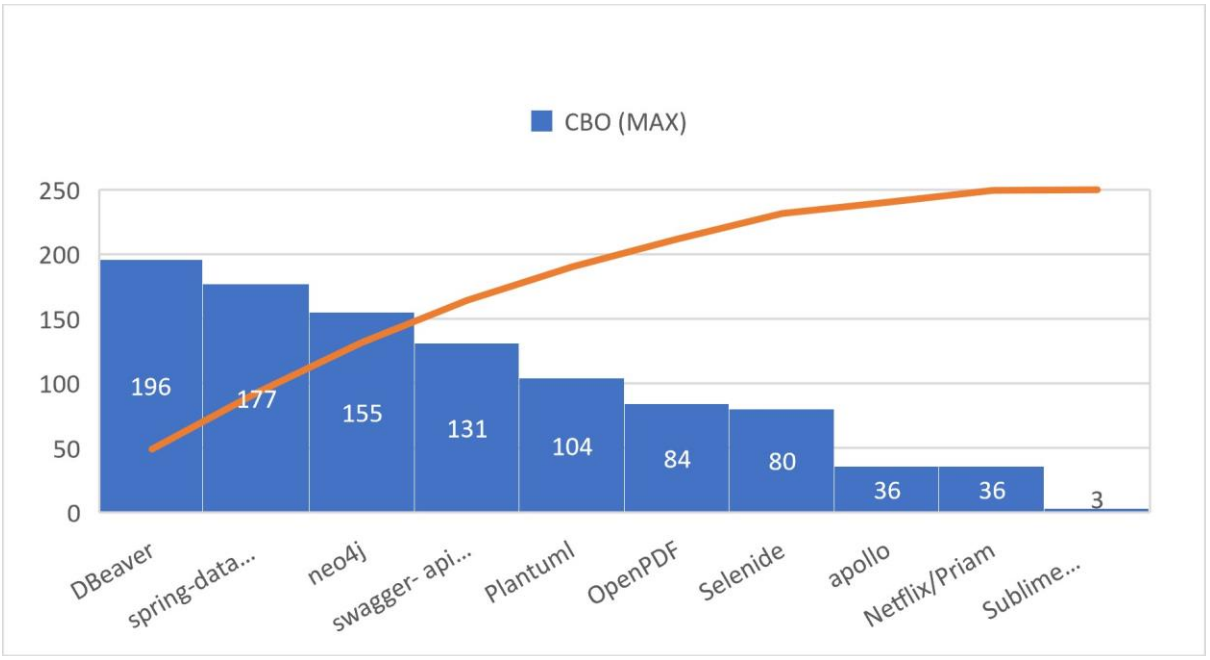
Feature Envy | Type Checking | Long Method | God Class

Refactoring Type	Source Method	Variable Criterion	Block-Based Region	Duplicated/Extracted	Rate it!
	org.neo4j.consistency.ConsistencyCheckMemoryCalculation::static org.neo4j.consis...	pageCacheMemory			
Extract Method			B1	1/4	
	org.neo4j.consistency.ConsistencyCheckService::public org.neo4j.consistency.Consi...	config			
Extract Method			B2	2/3	
	org.neo4j.consistency.ConsistencyCheckService::public org.neo4j.consistency.Consi...	pageCache			
Extract Method			B2	3/4	

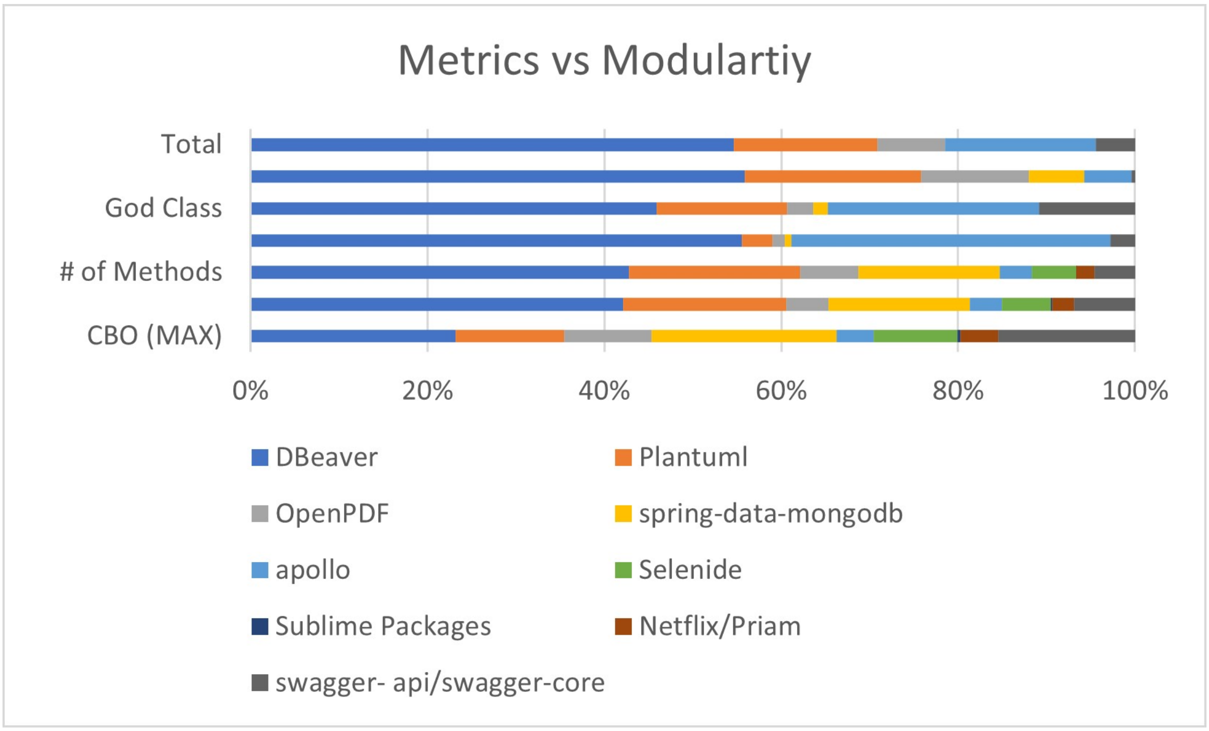
Analysis of the bad code smells with JDeodorant



C&K metric tools plot for the code smells



Design of code smell & metrics



The above evaluation results indicates that the tool accurately identified these design smells in real-world applications. Here, the **DBeaver** project contains a large number of God Classes, and the **swagger**-api/swagger-core project has several classes with high levels of Feature Envy.

It is also evident that JDeodorant detected and prevents the Long Methods, which are common signs of poorly structured code. For instance, the **OpenPDF** project has many such methods, indicating that some refactoring might be needed.

Additionally, JDeodorant's ability to detect God Classes, classes with high levels of Feature Envy, and long methods in projects such as Sublime Packages, Netflix/Priam, and Selenide suggests that the tool can be beneficial for teams working on smaller or more specific projects.

In conclusion, JDeodorant has proven to be a valuable asset in identifying and mitigating design smells in large-scale and diverse codebases. By accurately identifying and quantifying these smells, the tool helps developers maintain high-quality code and ensure the overall success of their projects.

Code Complexity and Size: Projects exhibit diversity in metrics like Coupling Between Objects (CBO), number of classes, and number of methods. Variation in these metrics suggests differences in code complexity and the scale of codebases across projects.

Feature Envy, God Class, God/Long Method: Projects showcase varying counts of specific code smells, highlighting potential areas of concern within the codebase. Higher counts of Feature Envy, God Class, and God/Long Method might indicate potential design or maintainability issues in those specific areas.

Coming to ***Code refactoring***, DBeaver project should reduce the number of God Classes involves breaking down these classes into smaller, more focused classes. This can be achieved by following these steps:

- Identify the classes that have multiple responsibilities.
- Break down these classes into smaller classes, each responsible for a single functionality.
- Determine the relationships between these smaller classes and adjust them accordingly.
- Update the existing codebase to reflect these changes and test the functionality to ensure it still works as expected.

For example, let's say there is a God Class named DatabaseManager that handles both database connection and data manipulation. We can refactor this class into two separate classes: DatabaseConnectionManager and DatabaseDataManipulationManager.

These new classes can be assigned the relevant methods and properties from the original God Class, while also adjusting their relationships with other classes in the codebase. In the long run, this refactoring effort can help improve the code's readability, maintainability, and scalability, ultimately resulting in a more robust and high-quality codebase.

We have the option to refactor code in the JDeodorant tool, for example- In the Feature Envy view the user selects the entry corresponding to the Move Method refactoring to perform. The methodology suggests the selection of the refactoring with the lowest Entity Placement value. however, we can select any of the candidate refactoring’s. The user can press the Apply Refactoring button to perform the selected refactoring on source code.

To maintain or improve modularity during software development, consider the following strategies:

- Follow a consistent coding style and adhere to coding standards to improve code readability and maintainability.
- Implement modular design patterns and architectural styles to enhance the organization and structure of the code.
- Perform regular code reviews and apply appropriate code quality checks to identify and address code smells and potential modularity issues early in the development process.
- Use tools like IDEs and static code analysis tools to automate the identification and remediation of code smells and promote better modularity.
- Continuously learn and adopt new programming paradigms, techniques, and best practices to enhance modularity and improve overall software quality.
- Maintain comprehensive documentation outlining module interactions, responsibilities, and interfaces to facilitate better communication among development teams.

References

- design smells. Softw Eng IEEE Trans 36:20–36. doi: 10.1109/tse.2009.50 Murphy-Hill E, Black A (2010) An interactive ambient visualization for code smells. In: Proceedings of the 5thinternational symposium on software visualization. ACM, pp 5–14
- Zazworka N, Ackermann C (2010) CodeVizard: a tool to aid the analysis of software evolution. In: Proceedings of the 4th international symposium on empirical software engineering and measurement. ACM, article 63
- Tsantalis N, Chaikalis T, Chatzigeorgiou A (2008) JDeodorant: identification and removal of type-checking bad smells. In: Proceedings of the 12th European conference on software maintenance and reengineering. IEEE, pp 329–331
- Paiva T, Damasceno A, Padilha J, Figueiredo E, Sant'Anna C (2015) Experimental evaluation of code smell detection
- tools. In: 3rd workshop on software Visualization, Evolution, and Maintenance (VEM), pp 17–24
- D. Baldwin, F. Sayward, Heuristics for determining equivalence of program mutations, Technical Report Research Report...
- N. Fenton *et al.*
  Software Metrics: A Rigorous and Practical Approach
  (1999)
- V.R. Basili, "Quantitative Evaluation of Software Engineering Methodology," Proceedings of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985.
- [Chidamber and Kemerer, 1994] S. R. Chidamber and C. F. Kemerer, “A metric suite for object oriented design,” IEEE Transactions on Software Engineering, pp. 476–493, 1994.
- Izurieta, C., & Bieman, J. M. (2007, September). How software designs decay: A pilot study of pattern evolution. In Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on (pp. 449-451). IEEE.
- [Fenton and Pfleeger, 1997] Norman E. Fenton, Shari Lawrence Pfleeger. Software Metrics. A rigorous & Practical Approach. 2nd Edition. ITP, International Thomson Computer Press, 1997
- https://sites.pitt.edu/~ckemerer/CK%20research%20papers/MetricForOOD_ChidamberKemerer94.pdf