The Smell of Blood: Evaluating Anemia and Bloodshot Symptoms in Web Applications

Zijie HUANG, Junhua CHEN, Jianhua GAO*
Department of Computer Science and Technology, Shanghai Normal University, Shanghai, 200234, China
hzjdev@foxmail.com, {chenjh, jhgao}@shnu.edu.cn

***Abstract*—In web applications that adopt layered architecture, Domain Layer is formed by Domain Model. Without any behavior, Anemic Domain Models contain only data. Those behaviors are dispersed into other layers and causing bloodshot symptoms in them. Most empirical studies suggest that symptoms of anemia and bloodshot degrade the maintainability of web applications, but no quantitative research has been done. This paper evaluates intensities of anemia and bloodshot symptoms based on metrics of three Code Smells, i.e. Data Class, Feature Envy and Blob. Furthermore, correlations of the intensities are evaluated using Spearman's rank correlation coefficient. The achieved results of experiments made on multiple versions of open-sourced projects show that over 65% of the applications are affected by anemia and bloodshot symptoms, and their intensities rarely decrease over time. Correlations of the intensities are also discovered within a single version and among multiple versions.**

Keywords-anemic domain model; code smell; web application; domain driven design

I. INTRODUCTION

Domain Driven Design (DDD) [1] is a model-driven methodology aims to tackle the complexity of software systems. DDD introduced a layered architecture consisting of four layers including Interface, Application (also known as Service [2]), Domain, and Infrastructure Layer. Data in Domain Layer is presented by Domain Models. Fowler [3] defines the Domain Model as an object model of the domain that incorporates both behavior and data, while Anemic Domain Model (ADM) is a Domain Model containing little or no such behavior.

Applying ADMs to Domain Layer triggers the anemia of Domain Layer accompanied by the bloodshot of other layers. Firstly, the domain behaviors are dispersed into other layers notably the Service Layer, but those behaviors still depend on ADMs' data structure, causing tight coupling of the Service Layer to the Domain Layer. As a result, the Service Layer becomes oversized while its cohesion is reduced. Secondly, the object-oriented program degrades into process-oriented program [2] with reduced comprehensibility.

Evans [1] suggests the Service Layer should be kept "thin," while Fowler [2] concludes that ADM is a common anti-pattern and their usage should be avoided. Both of them point out that the core business logic of web applications should be concentrated on the Domain Layer. However, ADMs are still widely adopted in enterprise systems [4]. There have been several discussions questioning whether ADM is an anti-pattern, which suggests the advantages of ADM should be refocused, and in some cases, ADM may be the best practice [5, 6].

* Corresponding Author. The work of this paper was supported by the National Natural Science Foundation of China (Grants 61672355).
DOI reference number: 10.18293/SEKE2019-061

Above-mentioned symptoms and discussions are presented in empirical studies. To the best of our knowledge, the pros and cons of ADMs have not been quantified by any research.

Code Smell is the symptom of poor design and bad implementation choices [7]. Fowler [8] proposes 22 Code Smells for object-oriented programming including God Class (also known as Blob), Feature Envy, and Data Class. Code Smell intensities could be evaluated by proper metrics.

In this paper, we apply metrics of Blob and Feature Envy to quantify bloodshot symptoms, and Data Class for anemia symptoms. The source code of 112 MVC-based Java application together with 96 versions of 10 Java web application are analyzed, while over 65% of them are affected by anemia and bloodshot symptoms. The analysis shows that there is a positive correlation between anemia and bloodshot symptoms, and intensities of these two symptoms rarely decrease over time. The results also reveal that although ADMs are built to separate business logic and data structure completely, most ADM-based applications are not strictly following the design.

The main contributions of this paper are:

1) Fills the gap in the quantification method of anemia and bloodshot symptoms in web applications.

2) Confirms Fowler's empirical discoveries of the negative impact of ADMs on software systems, while the advantages of ADMs have not been proved by any experiment.

3) Reveals the persistence and correlations of anemia and bloodshot symptoms.

The rest of this paper is organized as follows. Section II introduces the background and related works. Section III details the Code Smell detection approach and the quantification methods of anemia and bloodshot symptoms. In Section IV we have done several experiments to verify the accuracy of our approaches and presented our evaluation process together with results. Then, we detail the threats that could affect the validity in Section V. Finally, Section VI concludes the paper.

II. RELATED WORKS

A. Code Smell Detection

This paper evaluates intensities of 3 following Code Smells.

- Blob is for an oversized class with low cohesion, and it implements multiple irrelevant responsibilities [9-11].
- Data Class is a class that contains only data but no behaviors[12].
- Feature Envy describes a method more interested in a class other than the one it is in [9].

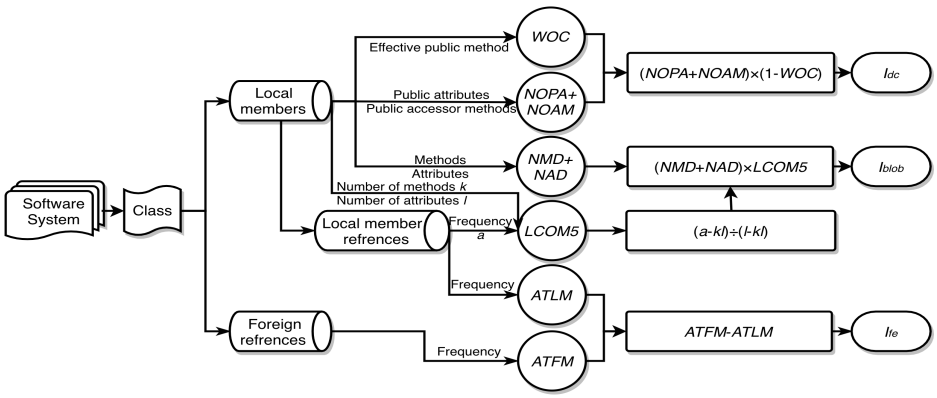

Fig. 1. Overview of Code Smell quantifying method

TABLE I. COMPARISON OF APPROACHES FOR FEATURE ENVY

Name / Approach	Lanza *et al.*[10]	Fokaefs *et al.* [14]
Couples With	Multiple classes	A single class
Precise Coupling Target	No. It detects coupling of a class generally.	Yes. It focus on both couplin g source and target.
Metrics	Access to Foreign Data (ATFD), Local Attribu te Access(LAA), Forei gn Data Provider(FDP)	Access to Distinct Foreign Members, Access to Distinct Local Members
Chosen	No	Yes (Extended to detect mult iple coupling targets)

TABLE II. COMPARISON OF APPROACHES FOR BLOB

Name / Approach	Lanza *et al.*[10]	Moha *et al.* [15]
Metrics	Structural: Access to Foreign D ata (ATFD), Weighted Method C ount (WMC), Tight Class Cohesio n (TCC)	Structural: Number of Methods Defined (NMD), Number of Attr ibutes Defined(NAD), Lack of C ohesion of Methods(LCOM5). Textual: Class name contains M anager, Process, Control, etc. Coupling: Access to at least one Data Class.
Chosen	No	Yes

Lanza *et al.* [10] quantified Code Smells and proposed several metrics and relevant thresholds, which are widely adopted in modern Code Analysis tools such as PMD [13]. JDeodorant [14] and DECOR [15] are notable tools detecting coupling and cohesion Code Smells.

Palomba *et al.* [11] proposed a pure textual detection approach called TACO, which is significantly different to traditional structural approach, aiming to discover conceptional coupling and cohesion problems. Furthermore, Palomba *et al.* [12] built a detection model based on multiple metrics, and they also evaluated the co-occurrence of Code Smells.

While Data Class metric is commonly accepted [10,12], multiple approaches of Feature Envy and Blob detection exists and their compatibilities to tasks in this paper is worth discussing. The difference in detection approaches is listed in Table I and Table II.

For Feature Envy, the main difference of the two approaches is whether an actual coupling target should be detected. Clarifying actual coupling targets is a must, as the identification of layer connections is vital. For Blob, both two methods refer to low cohesion, while Moha *et al.* [15]'s approach is more convenient due to its coupling detection and textual rules. In layered web application, classes and package names follow specific rules, and coupling with a Domain Model is the cause of the bloodshot symptom.

B. Web Application Design Problems

Aniche *et al.* [16] defined several MVC specific Code Smells, and evaluated their variation together with lifecycles based on a public dataset of 120 open-sourced GitHub repositories. This work developed metrics concerning specific web application code components such as Data Access Object (DAO), Repository, Controller and Service.

Cemus *et al.* and Cerny *et al.* [17,18] empirically investigated the negative impacts of ADMs and RDMs, and proposed a generic modeling method to ensure maintainability. According to their case studies, ADMs could cause Information Restatement and Concerns Tangling. RDMs also trigger coupling and cohesion problems, but intensities of design problems among Domains are decreased. The coupling problems within a single Domain could be resolved using Aspect Domain Model (AsDM).

C. Class Role Inference and Layering

Sakar *et al.* [19] generated Dependency Graph of modules and determined layer of each vertex according to their number of indegree and outdegree, while Hayashi *et al.* [20] inferred the role of code component in MVC-based applications using Dependency Graph.

Hickey *et al.* [21] split classes into layers according to their names, this method could lose its accuracy due to different naming strategies. Fokaefs [14] *et al.* and Aniche *et al.* [16] mentioned role detection of code components using class name and annotation name in their works.

III. APPROACH

Several fundamental data, i.e. Code Smell intensities and layers of classes, should be collected before evaluating anemia and bloodshot intensities. The overview of Code Smell quantifying method is illustrated in Fig. 1. In the following paragraphs, we explain each of the steps in detail.

A. Evaluating Data Class Intensity

Data Class is a class with interfaces that (i) provide almost no functionality and (ii) declare data fields. [10]

As shown in Fig. 2, *WMC* metric is used for (i), and *NOPA+NOAM* together with *WOC* are adopted for (ii).

WOC is the number of public methods (with accessors and constructors excluded) divided by the total number of public members. *NOPA* is the number of public attributes of a class, while *NOAM* is the number of accessor methods, i.e. getter and setter. *WMC* sums the complexity of all methods of a class.

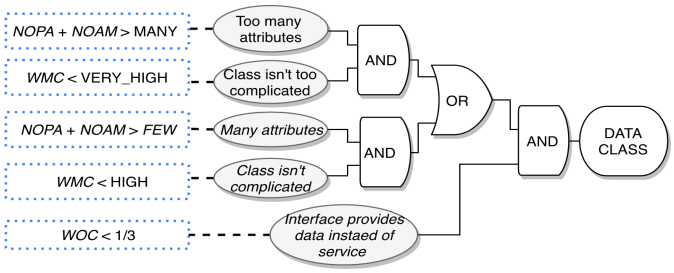

Fig. 2. Data Class detection approach

The *CYCLO* metric is used to calculate method complexity. The calculation approach of *CYCLO* defined in PMD following the standard rules given below is used in this paper:

- Methods have a base complexity of 1;
- +1 for every control flow statement (if, case, catch, throw, do, while, for, break, continue) and conditional expression (?:) ;
- else, finally and default do not count;
- +1 for every Boolean operator (&&, ||) .

Thus, the intensity of Data Class could be calculated as (1):

$$I_{dc}(C)=\left(1-WOC(C)\right)\times\left(NOPA(C)+NOAM(C)\right) \quad (1)$$

B. Evaluating Feature Envy Intensities

A class is affected by Feature Envy if it access members of another class more frequently than its local members. [14]

Given a class $C_{current}$, the approach calculates *ATLM* as the frequency of distinct local member accessed by $C_{current}$. Then, a set *C* for all classes in the software system is formed, for each class C_i in *C*, the frequency of distinct member access from $C_{current}$ to C_i named a_i is evaluated. Finally, the classes in *C* is sorted by a_i in descending order. A class is affected by Feature Envy if the first class C_{top} is not equivalent to $C_{current}$.

For each a_i, calculate $diff=a_i - ATLM(C)$ and treat all negative *diff* value as zero. Then we sum all *diff*s to obtain the result of *ATFM* metric. For each $diff > 0$, the approach treats the related C_i as the coupling target of $C_{current}$.

The intensity is calculated as (2):

$$I_{fe}(C) = ATFM(C) - ATLM(C) \quad (2)$$

C. Evaluating Blob Intensities

As shown in Fig. 3, a Blob class is oversized, with low cohesion, having controller name pattern and couples with Data Classes. [15] We ignore name pattern as it is mentioned in Section D.

Size of a class could be measured according to the sum of *NMD* and *NAD* metrics. The cohesion of class could be evaluated by *LCOM5* metric, the main idea of *LCOM5* is to calculate the rate of access to local members.

Given a class *C*, the approach determines the number of method members *k*, the number of attribute members *l*, and the frequency of access to distinct local members *a*. *LCOM5* could be calculated as (3):

$$LCOM5(C)=\frac{a-kl}{l-kl} \quad (3)$$

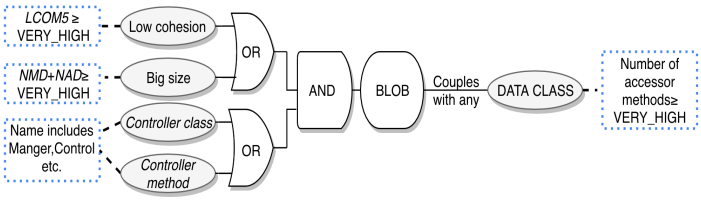

Fig. 3. Blob detection approach

TABLE III. CLASS ROLE INFERENCE APPROACH

Role/ Approach	Domain	Persistence	Service	Interface
Lowercased name includes	{domain, vo, entity, entities}	{dao, repo, repository}	service	{controller, ctrl, api}
Expected Layer	1(bottom)	2	3	4(top)

We pick the 3rd-quartile in Al Dallal's work [22] as threshold of *LCOM5* metric, and the fixed value in Palomba *et al.* 's [12] work as threshold of *NMD+NAD* metric.

The intensity of Blob could be calculated as (4):

$$I_{blob}(C)=\left(NMD(C)+NAD(C)\right)\times LCOM5(C) \quad (4)$$

D. Class Layering

There is no common and generic approach for class layering. Related works mainly consider two features, class dependencies [19,20] and class names [14,16,21]. This paper proposed a mixed layering approach that fits web applications.

First, the approach generates a Directed Acyclic Graph (DAG) according to class dependencies. Given a set of all classes in an application named *C*, a vertex is generated for each class. Then, pairs of vertexes are connected as follows: According to Table III, the likely role of each class could be inferred. For $Ci \in C$, $Cj \in C$ and $Ci \neq Cj$, if *Ci* accessed or called any member of *Cj*, and *Ci* and *Cj* have different inferred roles, an edge from *Ci* to *Cj* should be generated.

Then, the DAG should be split into 4 layers. At the very beginning, the approach splits the DAG into 3 layers. Vertexes with 0 in-degree are moved to the bottom (i.e. Domain) while those with 0 out-degree are moved to the top (i.e. Interface). For the rest of the vertexes kept in a temporary layer, we split them into 2 new layers using the similar method but ignore the connection of middle layer vertexes with the ones in the top and the bottom layer. Vertexes with 0 in-degree are moved to the lower layer (i.e. Persistence classes of Infrastructure), while other vertexes remain in the upper layer (i.e. Service).

Additionally, there exist some exceptions. For the vertexes whose roles cannot be inferred through names, their roles could be determined according to their actual layers as mentioned in Table III. If any Data Class has the naming pattern of Data Transfer Object and are only accessed by Interface Layer classes, they should be excluded from the DAG as their sole function is to normalize data during the transfer process.

Domains of an application could also be detected according to DAG and class name patterns. A set of words could be derived through splitting the Camel-cased Domain class names. If any word in the set appears in the name set of classes in other layers,

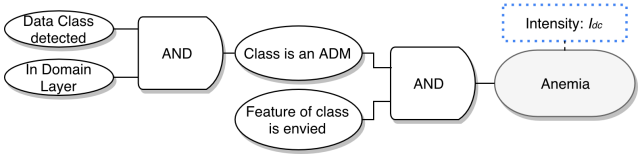

Fig. 4. The strategy of Anemic Class detection

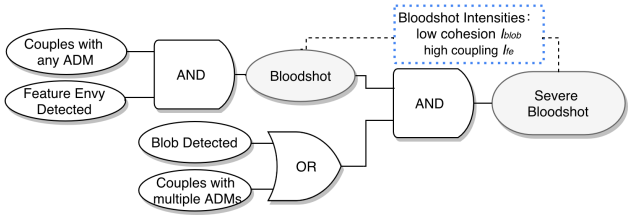

Fig. 5. The strategy of Bloodshot Class detection

and the two vertexes of classes are connected, a domain could be determined.

E. *Quantifying the symptoms of anemia and bloodshot*

For now, we have collected the fundamental data including the layer of each class and their Code Smell intensities. The intensities of symptoms should be evaluated as follows.

As mentioned in Fig. 4, ADM is determined if a Data Class belongs to the Domain Layer, the intensity of anemia is I_{dc}.

As Fig. 5 illustrates, if a non-Domain-Layer class affected by Feature Envy and couples with a Data Class, we consider it bloodshot with intensities of two metrics: the extent of low cohesion I_{blob} and the extent of high coupling I_{fe}. For any bloodshot class with I_{blob} >0 or couples with multiple ADMs, we regard it as a *Severe Bloodshot Class* (**SBC**).

IV. Experiments

This paper implements Code Smell metrics based on PMD [13] with necessary modifications [23]. Statistical data and plots are produced by Python scripts after generating reports of Code Smell intensities. Experiments are conducted to address the following 5 research questions:

- RQ1 **Accuracy**: *Is the approach able to evaluate class layers and Code Smell intensities accurately?*
- RQ2 **Severity**: *How severe is the symptom of anemia and bloodshot in web application?*
- RQ3 **Correlation**: *What is the relationship between the symptom of anemia and bloodshot?*
- RQ4 **Survivability**: *Do intensities of anemia and bloodshot symptom decrease over time?*
- RQ5 **Evaluation**: *What is the effect of applying ADM?*

A. *Accuracy of fundamental data*

A typical web application called military-shop is picked from the dataset [16] consisting of 120 open-sourced Java applications based on MVC architecture from GitHub. RQ1 is to be answered in this section using military-shop as a demo.

Precision and *Recall* metrics are used to evaluate accuracy. The metrics can be calculated as (5) and (6):

$$Precision = \frac{Correct \cap Detected}{Correct} \quad (5)$$

$$Recall = \frac{Correct \cap Detected}{Detected} \quad (6)$$

TABLE IV. Accuracy of Layering Approach

Role/Metric	Domain Model	Persistence	Service	Interface	Utility
Precision	100%	100%	100%	90%	85.71%
Recall	88.57%	100%	85.71%	94.73%	100%
Samples	35	7	14	19	12

TABLE V. Levels of Correlation

Range of ρ	Correlation Level
[0.8,1.0]	Very Strong
[0.6,0.8)	Strong
[0.4,0.6)	Moderate
[0.2,0.4)	Weak
[0.0,0.2)	Very Weak

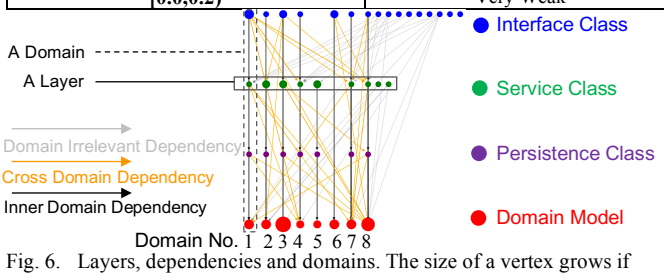

Fig. 6. Layers, dependencies and domains. The size of a vertex grows if overlapped. Edges are hidden if they connect vertexes in the same layer.

where *Correct* denotes the set of items manually identified and *Detected* represents the set of items detected by the heuristic.

Fig. 6 shows the layered DAG of the project, while Table IV lists the accuracy of the layering approach. A few classes with ambiguous patterns were not correctly layered. The approach also detected all 8 domains of the project, and classes with correct inferred role were all placed into proper domain. The domain detection heuristic had a *Precision* of 100% and a *Recall* of 91.58%. Relevant Code Smell intensities were also validated.

Compared with the results of manual detection, we can conclude that fundamental data could be detected correctly. Manual detection is done independently by the first author and a developer having 3 years of enterprise web application development experience. A few disagreements were discussed and resolved later.

B. *The Experiment conducted on single application*

This section use Shopizer [24] as an example to demonstrate the experiment conducted on each project. The results will not be fully presented in this section as they are listed in Table VII and VIII instead.

For Q2, as shown in Table VII, 60.97% of the classes in the Domain Layer were ADMs, while Bloodshot occured in 95.16% of the Interface and Service classes within detected Domain. About 70% bloodshot classes were SBCs.

In order to answer Q3, the correlations of I_{blob}, I_{fe} and I_{dc} should be evaluated. This paper uses the Spearman's rank correlation coefficient [25] with a *P*-value of 0.05 to analyze the correlation between every pair of intensities. The metric takes the input value of two sets of values equal in length and produces the correlation coefficient ρ together with the significance level *P*. With P<0.05, the level of correlation could be considered statically significant with a level presented by ρ ranges in [-1, 1] as shown in Table V. For example, the metric reported a strong

TABLE VI. DETECTION RESULTS OF 59 OPEN-SOURCED PROJECTS

	Class/ Domain Count	SBC rate for bloodshot classes		Bloodshot rate for Services and Interfaces of Domains	ADM contribution to Domain coupling	ADM rate for Domain Layer Classes	ρ of I_{blob}, I_{fe}	ρ of I_{dc}, I_{blob}	ρ of I_{dc}, I_{fe}
		Service	*Interface*						
Mean	645.34/23	65.30%	65.98%	97.76%	49.88%	50.64%	0.68	0.07	0.09
Variance	333845.04/330	0.09	0.09	0.01	0.05	0.05	0.02	0.15	0.11

TABLE VII. PROJECT COMPOSITIONS OF 10 OPEN-SOURCED JAVA WEB APPLICATIONS

Project Name	Commit/Fork/ Release	Latest Version	Class/Domain Count	SBC rate for bloodshot classes		Bloodshot rate for Services and Interfaces of Domains	ADM contribution to Domain coupling	ADM rate for Domain Layer Classes
				Service	*Interface*			
Shopizer	193/941/6	2.2.0	811/31	72.22%	68.18%	95.16%	68.85%	60.97%
OpenLegislation	3192/88/28	2.17	787/36	95.65%	61.11%	98.96%	32.70%	32.04%
LibrePlan	9657/148/32	1.4.1	1294/50	66.67%	100%	99.14%	41.33%	13.91%
OpenCMS	22750/321/228	10.5.4	3382/54	84.61%	76.92%	97.67%	43.63%	18.84%
Thingsboard	1514/676/17	2.1	797/31	100%	63.64%	98.73%	35.67%	8.70%
Sakai	46898/535/21	12.3	4951/45	71.43%	92.00%	100%	38.85%	30.86%
OpenClinica	8521/167/30	4.5.2	1436/38	81.48%	65.06%	96.85%	71.94%	58.41%
Apollo	1944/2873/14	1.0.0	458/24	68.32%	78.27%	100%	57.79%	78.92%
Dataverse	12042/180/30	4.9.2	675/19	33.33%	55.56%	97.43%	44.30%	42.30%
DDDLib	2310/153/18	4.6.1	392/-	0%	0%	-	0%	10.00%

TABLE VIII. P VALUES OF 10 OPEN-SOURCED JAVA WEB APPLICATIONS

Project Name	Analyzed Versions	ρ of I_{blob}, I_{fe}	ρ of I_{dc}, I_{blob}	ρ of I_{dc}, I_{fe}	ρ of ΔI_{blob}, ΔI_{fe}	ρ of ΔI_{dc}, ΔI_{blob}	ρ of ΔI_{dc}, ΔI_{fe}	R_{dec}
Shopizer	6	1.00	-	-	0.78	0.70	0.79	0.70%
OpenLegislation	10	0.81	0.71	0.94	0.67	-	0.31	0.00%
LibrePlan	10	-	-	0.85	-	0.73	0.64	3.84%
OpenCMS	10	0.80	-	-	0.95	0.58	0.47	1.72%
Thingsboard	10	0.99	0.62	-	0.81	0.61	0.74	2.38%
Sakai	10	0.94	-	0.63	0.87	0.51	0.38	1.47%
OpenClinica	10	0.57	-	0.71	0.70	0.28	0.51	0.45%
Apollo	10	1.00	0.96	0.96	0.70	0.36	0.42	1.11%
Dataverse	10	-	-	-	0.78	-	-	0.00%
DDDLib	10	-	-	-	-	-	-	0.00%

correlation between I_{blob} and I_{fe} with ρ=0.71 (P=2.62e-12).

In order to answer Q4, it is necessary to analyze the variation of the 3 concerned intensities (ΔI_{blob}, ΔI_{fe} and ΔI_{dc}) among all 6 release versions. Intensities between two neighboring versions were calculated and normalized to the range [0,1]. The correlations of variations were also evaluated.

For Q5, we calculated the rate of ADMs' contributions to I_{fe} in domains, i.e. the sum of I_{fe} of every domain class coupled with at least 1 ADM divided by the sum of I_{fe} in all domains.

C. Results

Following a similar process of Section B, the experiment was conducted on 120 applications mentioned in Section A, in which 112 projects were available for access on GitHub. Among the 112 projects, 66.96% of them were affected by anemia and bloodshot symptoms(ADM rate > 0% and bloodshot class rate > 0%) and 52.68% of them had at least 1 valid ρ value (P<0.05 and 0<|ρ|<1). Table VI reports the result of the applications with valid ρ value.

The primary cause of the invalid ρ values were: (i) The project was too lightweight, i.e. contained few lines of code. (ii) No such correlation exists. Domains were not detected in some of the applications, in most of the cases they did not have valid ρ values owing to the fact that these projects were not layered or they were not web applications.

This dataset was collected using the filter "exists at least 10 controllers" for the analysis of MVC-based applications, most of them lacked valid release information, which were not capable for analysis based on multiple versions. So a new dataset must be picked.

We selected 10 Java web applications with more than 100 commits, more than 100 classes, and at least one commit in recent 6 months. For each project, we analyzed its latest 10 versions available. 9 out of the 10 projects were affected by anemia and bloodshot symptoms. DDDLib was an exception that follows the DDD specification. Dataverse did not have a clear layering pattern, and the size of each layer was not enough for the metrics to evaluate correlation data.

"The proportion of decreasing anemia or bloodshot intensities" named R_{dec} was calculated as the ratio of "the number of classes with any of the three intensities decreased" to "the number of SBCs and ADMs".

Table VII reports the composition of the applications, while Table VIII lists multiple correlation coefficients (ρ), any ρ with P>0.05 will be marked as unavailable(-).

D. Discussions

For Q2, more than 65% of the 112 projects analyzed by the experiment were affected by anemia and bloodshot symptoms. For web application domains based on ADMs, the proportion of bloodshot classes exceeded 90%, and most of them were SBCs, indicating the symptoms were common and severe.

For Q3, regarding the ρ values of Table VI and Table VIII, the two intensities of bloodshot correlated in most of the cases. Among different versions of the same project, the variations of the three intensities often correlated. We also analyzed the correlation of symptom intensities in different layers within single domains. But we did not find any significant correlation. The cause might be a large number of design problems within the domain are related to other domains instead of itself.

For Q4, as shown in the last column of Table VIII, the symptoms of anemia and bloodshot rarely reduced, which also confirms the conclusion about structural Code Smells that they tend to become more severe and are rarely removed [9,15].

For Q5, the result of our experiment is not showing any advantages of ADM, but confirmed the widely-accepted conclusion that there are a lot of coupling and cohesion problems within an ADM-based domain resulting in SBCs. To our astonishment, applying ADM will not result in a complete separation of data and business logic as it is designed for in most of the cases. The ADM-based applications often contain 30% to 70% of non-ADM domain models, in which domain behaviors are implemented. In conclusion, ADM has an obvious shortage of keeping single responsibilities.

V. Threats to Validity

A threat to Internal Validity is that the layering approach uses name patterns. If the layering pattern in class name is ambiguous, the detection will be completed only according to dependency information, and accuracy will be affected.

Threats to External Validity are listed as follows: (1) Detection process could lose its validity on small applications, as thresholds derive from enterprise applications. (2)There exist a few applications that do not follow layered design. (3) Our approach analyses Java-based web application, the conclusion may not satisfy applications based on weakly-typed languages.

VI. Conclusions and Future work

It has been 15 years since Fowler first proposed the concept of ADM and its negative impacts, but ADM-based domain modeling is still popular. This paper analyzed source code of 112 MVC patterns based Java applications in a public dataset and 96 versions of 10 Java web applications, and concluded that over 65% of applications are affected by anemia and bloodshot symptoms. The analysis also suggests a positive correlation between the two symptoms, and they rarely decrease over time. The shortage of ADM are confirmed by experiment results, furthermore, in most of the cases, the complete separation of data and business logic are not implemented as ADMs are designed for.

Our future work involves the investigation of the impact of commit changes on anemia and bloodshot symptoms, and the application of Deep Learning approaches to improve the accuracy of class role detection is also worth trying.

References

[1] E. Evans, *Domain-driven design: tackling complexity in the heart of softw are*. Boston: Addison-Wesley Professional, 2004.

[2] AnemicDomainModel[Online]. Available: https://www.martinfowler.com/bliki/AnemicDomainModel.html. [Accessed Feb 28, 2019].

[3] M. Fowler, *Patterns of enterprise application architecture*. Boston: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] F. Wang, L. Yan, Z. Peng, S. Wei, and D. Yuan. "The investigation of WEB software system based on domain-driven design." in *International Conference on Web Information Systems and Mining*, Taiyuan, China, 2011, pp. 11-18.

[5] The Anaemic Domain Model is no Anti-Pattern, it's a SOLID design [Online].Available:https://blog.inf.ed.ac.uk/sapm/2014/02/04/the-anaemic-domain-model-is-no-anti-pattern-its-a-solid-design/ [Accessed Feb 28, 2019].

[6] R. Wirfs-Brock. "Are software patterns simply a handy way to package design heuristics?." *in Proceedings of the 24th Conference on Pattern Languages of Programs*, Vancouver, Canada, 2017, p. 3.

[7] M. Tufano, F. Palomba, G.Bavota, R. Oliveto, M. Di Penta, A. De Lucia *et al.*, "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1063-1088, 1 Nov. 2017.

[8] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts. *Refactoring: improving the design of existing code*. Boston: Addison-Wesley Professional, 1999.

[9] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto and A. De Lucia, "The Scent of a Smell: An Extensive Comparison Between Textual and Structural Smells," *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 977-1000, 1 Oct. 2018.

[10] M. Lanza, R. Marinescu. *Object-oriented metrics in practice: Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*, Berlin: Springer Science & Business Media, 2007

[11] F. Palomba, A. Panichella, A. De Lucia, R. Oliveto and A. Zaidman, "A textual-based technique for Smell Detection," *in IEEE 24th International Conference on Program Comprehension,* Austin, TX, USA, 2016, pp. 1-10.

[12] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia and R. Oliveto, "Toward a Smell-Aware Bug Prediction Model," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 194-218, 1 Feb. 2019.

[13] PMD[Online].Available: https://pmd.github.io [Accessed Feb 28, 2019].

[14] M. Fokaefs, N. Tsantalis and A. Chatzigeorgiou, "JDeodorant: Identification and Removal of Feature Envy Bad Smells," in *IEEE International Conference on Software Maintenance*, Paris, France, 2007, pp. 519-520.

[15] N. Moha, Y. Gueheneuc, L. Duchien and A. Le Meur, "DECOR: A Method for the Specification and Detection of Code and Design Smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20-36, Jan.-Feb. 2010.

[16] M. Aniche, G. Bavota, C. Treude, M. Gerosa, and A. Deursen, "Code smells for model-view-controller architectures." *Empirical Software Engineering*, vol. 23, no. 4, pp. 2121-2157, Aug. 2018.

[17] K. Cemus, T. Cerny, L. Matl and J. Michael, "Aspect, Rich, and Anemic Domain Models in Enterprise Information Systems," in *International Conference on Current Trends in Theory and Practice of Informatics*, Harrachov, Czech, 2016, pp. 445-456.

[18] T. Cerny, M. Donahoo, "How to reduce costs of business logic maintenance," in *IEEE International Conference on Computer Science and Automation Engineering*, Shanghai, China, pp. 77-82

[19] S. Sarkar, G. M. Rama and S. R, "A Method for Detecting and Measuring Architectural Layering Violations in Source Code," in *13th Asia Pacific Software Engineering Conference*, Bangalore, India, 2006, pp. 165-172.

[20] S.Hayashi, F.Minami, M.Saeki, "Detecting Architectural Violations Using Responsibility and Dependency Constraints of Components," *IEICE TRANSACTIONS on Information and Systems*,vol.101, no.7, pp. 1780-1789,1 Jul. 2018 .

[21] S. Hickey, M.O. Cinnéide, "Search-Based Refactoring for Layered Architecture Repair: An Initial Investigation,"in *Proceedings of the North American Search Based Software Engineering Symposium,* Dearborn, MI, USA, 2015, pp:1-16.

[22] J. Al Dallal, "Measuring the Discriminative Power of Object-Oriented Clas s Cohesion Metrics," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 788-804, Nov.-Dec. 2011.

[23] Our fork of PMD[Online].Available: https://github.com/CodeSmellD/pmd-mini [Accessed Feb 28, 2019].

[24] Shopizer[Online].Available: https://github.com/shopizer-ecommerce/shopizer [Accessed Feb 28, 2019].

[25] J. H. Zar, "Significance testing of the Spearman rank correlation coefficien t," *Journal of the American Statistical Association*, vol. 67, no.339, pp.578 -580, 1 Oct. 1