GodExpo: An Automated God Structure Detection Tool for Golang

Rafed Muhammad Yasir, Moumita Asad, Asadullah Hill Galib, Kishan Kumar Ganguly, and Md. Saeed Siddik

Institute of Information Technology, University of Dhaka, Bangladesh
{bsse0733, bsse0731, bsse0712, kkganguly, saeed.siddik}@iit.du.ac.bd

***Abstract*—God Class is a class that threatens maintainability and understandability of code by performing most of the work alone. Various tools exist that can detect God Class of Java or C++ programs, however, there is no existing tool for detecting God Class(Structure) in Golang. Although Golang is not an object-oriented language, it offers structures which are similar to classes in OOP as they can contain fields and methods. Unlike OOP, methods of a structure can be defined on any file in the package of Golang. This paper presents a tool entitled GodExpo to detect God Structures in Golang programs by calculating metrics namely Weighted Method Count, Tight Class Cohesion, and Access to Foreign Data. In addition, GodExpo can provide version wise result to observe the evolution of God structures. To evaluate GodExpo, an experiment has been conducted on several versions of two open source Golang projects and the tool successfully found God structures in all versions of those projects.**

***Keywords*—God Class, Code Smell, Golang, OOP Metrics**

I. Introduction

Golang is an open source programming language developed and maintained by Google [1]. Since Golang is not an object-oriented language, it does not have polymorphism or inheritance. However, it offers encapsulation with structures (structs) which are similar to classes [2]. This design of Golang results in simpler code [2], which has rapidly increased its popularity [3]. Several renowned projects like Docker and Kubernetes are written in Golang [4]. As Golang's popularity increases, code smell detection tool in Golang is crucially required, which will help developers to maintain code quality [5].

Code smell refers to pattern or aspect of design in a software system that may cause problems for further development and maintenance of the system [6]. Among all other smells, God Class is one of the most studied smells in software engineering literature [7]. According to Lanza and Marinescu, the classes that tend to centralize the intelligence of the system are called God Classes [8]. A God Class performs most of the work, assigns only minor details to a set of trivial classes and uses the data from other classes [8]. It exhibits high complexity, low cohesion and heavy access to foreign classes data. It violates the single responsibility principle - "a class should have only one reason to change" [9]. Olbrich et al. specified that God Classes tend to be very large, which makes the system more difficult to understand [6]. They found that defects and changes occur more frequently in God Classes than other kinds of classes. Therefore, it is important to detect and refactor God Classes(Structs) for improving source code quality and maintainability. Various tools have been proposed to detect God Classes specially in OOP (e.g., JDeodorant [10] for Java, inFusion [5] for C++, etc.). However, there is no existing tool that can detect God classes(Structs) in Golang. In Golang, structs are similar to OOP Classes, which contains both attributes and methods.

This paper presents a tool named GodExpo that can automatically detect God Structs for Golang by calculating Weighted Method Count, Tight Class Cohesion, and Access to Foreign Data metrics. In addition, GodExpo can provide version wise result to observe the evolution of God Structs. GodExpo has been executed on several versions of two Golang projects namely Hugo [11] and Mattermost [12]. It has successfully detected God Structs in all versions of these two projects. The results further show that there is an increase in the number of God Structs with the evolution of different versions of the projects.

II. Technique

GodExpo uses three different metrics namely Weighted Method Count (WMC), Tight Class Cohesion (TCC) and Access To Foreign Data (ATFD) [8] to detect God Structs. These metrics are calculated by static code analysis and used to infer whether a struct is a God or not based on Equation (1) [8].

$$GC(C) = \begin{cases} 1, & \text{((WMC(C)} \geq 47\text{) and ((TCC(C)} < 0.3\text{) and ((ATFD(C)} > 5\text{)} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Where,

- C is the class being inspected.
- Weighted Method Count (*WMC(C)*) is the sum of the cyclomatic complexity of all methods in *C* [13].
- Tight Class Cohesion (*TCC(C)*) indicates the relative number of directly connected methods in *C* [6] presented in Equation (2).

$$TCC(C) = NDC(C)/NP(C). \tag{2}$$

Where,
  - *NP(C)* represents the maximum possible number of direct or indirect connections in a class. Two methods are directly connected if there exists one or more common instance variables between them. Two methods are indirectly connected if they are linked via other directly connected methods [14].

Let, N be the number of methods in a class.

$$NP(C) = N * (N{-}1)/2. \quad (3)$$

– *NDC(C)* is the number of direct connections.

- Access to Foreign Data (*ATFD(C)*) represents the number of foreign class attributes that are directly accessed by class C or via accessor methods.

Instead of class, GodExpo applies Equation (1) on Golang structs.

GodExpo performs The following three steps to detect God Structs as shown in Figure 1:

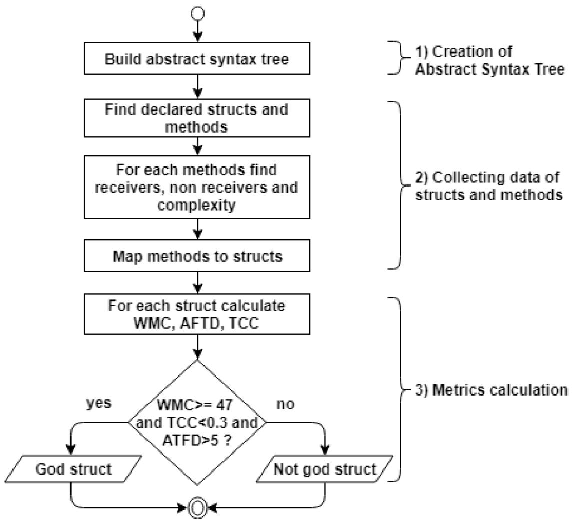
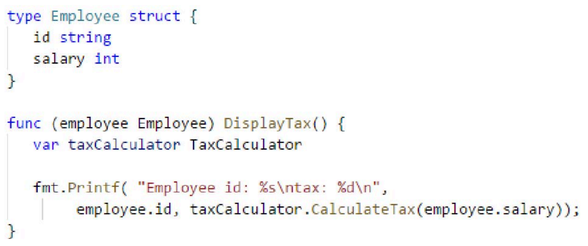

Build abstract syntax tree
1) Creation of Abstract Syntax Tree
Find declared structs and methods
For each methods find receivers, non receivers and complexity
2) Collecting data of structs and methods
Map methods to structs
For each struct calculate WMC, AFTD, TCC
WMC>= 47 and TCC<0.3 and ATFD>5 ?
yes
no
3) Metrics calculation
God struct
Not god struct

Figure 1 Flowchart of GodExpo

1) **Creation of Abstract Syntax Tree:** In this step, all go source files are traversed and an Abstract Syntax Tree (AST) is created from each file. To make this task easier, go/parser[1] package is used. By taking source code as input, it provides an AST as output. The ASTs are used in the next step for identifying structs and methods.



```
type Employee struct {
  id string
  salary int
}

func (employee Employee) DisplayTax() {
  var taxCalculator TaxCalculator

  fmt.Printf( "Employee id: %s\ntax: %d\n",
      employee.id, taxCalculator.CalculateTax(employee.salary));
}
```

Figure 2 Example of Receiver

2) **Collection of Struct and Method Data:** This step extracts all structs and methods from an AST using package go/ast[2] and go/token[3]. For each struct, its name, package, fields, and location in the project are extracted. For each method, its name, corresponding struct, package, receivers and other structs accessed are extracted. Receiver is like a parameter to a function [15] which identifies the corresponding struct of a method. In Figure 2, the *DisplayTax()* method has a receiver named *employee* of type *Employee*. In addition, this method has accessed another struct of type *TaxCalculator*. Unlike OOP, methods of a struct can be defined on any file of the package in Golang [16]. Nevertheless, the complexities of all methods must be added to calculate WMC. Therefore, the methods are mapped to structs by matching their struct name and package name.

3) **Metrics Calculation:** In this step, WMC, TCC and ATFD for a struct are calculated. Based on Equation (1), GodExpo decides whether the struct is God or not.

[1] https://golang.org/pkg/go/parser
[2] https://golang.org/pkg/go/ast
[3] https://golang.org/pkg/go/token

III. Usage

GodExpo has been created as a command line tool. It is publicly available at: https://github.com/rafed123/GodExpo. It comes with the following features:

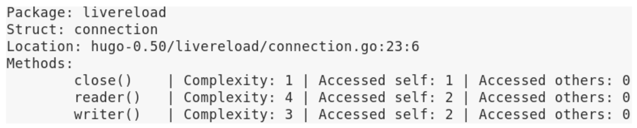
a) Show struct summary.
b) Find God Structs.
c) Show evolution of God Structs.
d) Set custom thresholds for metrics calculation.

A. *Show Struct Summary*

To summarize the structs of a go file, run the following command, where $file.go$ is the file to be analyzed.

```
./godExpo -f file.go
```

The output will show all the structs names, packages, and methods. For each method, its complexity and number of fields accessed are shown. Figure 3 shows a sample output, where the struct *connection* belongs to the *livereload* package. The struct has three methods namely *close()*, *reader()*, and *writer()*. The complexities of the methods *close()*, *reader()*, and *writer()* are 1, 4 and 3 correspondingly. They have accessed 1, 2 and 2 fields respectively from their struct. None of these methods accessed any field from other structs.



```
Package: livereload
Struct: connection
Location: hugo-0.50/livereload/connection.go:23:6
Methods:
        close()    | Complexity: 1 | Accessed self: 1 | Accessed others: 0
        reader()   | Complexity: 4 | Accessed self: 2 | Accessed others: 0
        writer()   | Complexity: 3 | Accessed self: 2 | Accessed others: 0
```

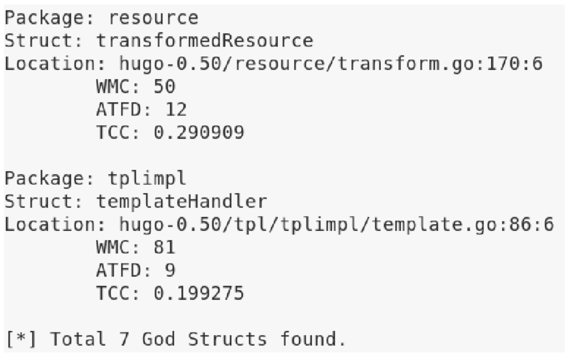
Figure 3 Struct Summary

B. *Find God Structs*

To find God Structs in a project, run the following command, where $directory$ is the project path.

```
./godExpo -d directory
```

The output will show the number of God Structs and their WMC, ATFD and TCC. A sample output is shown in Figure 4, where *transformedResource* struct from *resource* package is a God Struct. Its WMC, ATFD, and TCC are 50, 12, and 0.290909 respectively. This struct can be found at line number 6 of *transform.go* file inside the *hugo-0.50/resource* directory.



```
Package: resource
Struct: transformedResource
Location: hugo-0.50/resource/transform.go:170:6
        WMC: 50
        ATFD: 12
        TCC: 0.290909

Package: tplimpl
Struct: templateHandler
Location: hugo-0.50/tpl/tplimpl/template.go:86:6
        WMC: 81
        ATFD: 9
        TCC: 0.199275

[*] Total 7 God Structs found.
```

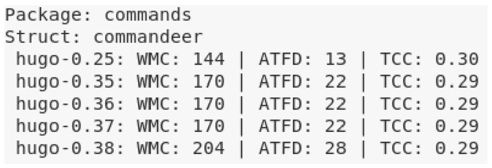
Figure 4 God Structs found in the specified project

C. *Show Evolution of God Structs*

The evolution of God Structs over different releases of the project can be shown by GodExpo. To see the evolution, run the following command:

```
./godExpo -e directory
```

Here, directory contains different releases of a particular project. For each release, the WMC, ATFD, and TCC of a God Struct are shown. Figure 5 shows the evolution of the God Struct *commandeer* from package *commands*. The first column of the output shows the version number (e.g., hugo-0.25) and the next three columns show WMC, ATFD, and TCC of the struct in that particular version. In this case, the WMC and ATFD are increasing, whereas TCC is decreasing.



```
Package: commands
Struct: commandeer
 hugo-0.25: WMC: 144 | ATFD: 13 | TCC: 0.30
 hugo-0.35: WMC: 170 | ATFD: 22 | TCC: 0.29
 hugo-0.36: WMC: 170 | ATFD: 22 | TCC: 0.29
 hugo-0.37: WMC: 170 | ATFD: 22 | TCC: 0.29
 hugo-0.38: WMC: 204 | ATFD: 28 | TCC: 0.29
```

Figure 5 Evolution of a God Struct

D. *Set Custom Thresholds for Metrics Calculation*

The thresholds for WMC, ATFD, and TCC for detecting God Structs are 47, 5 and 0.3 respectively [8]. However, these thresholds may vary from project to project. Schumacher et al. found that in their experiment, they could achieve 100% precision in God Class detection by manually setting ATFD to 10 [17]. Hence, GodExpo has provision for customizing metrics threshold. These thresholds can be changed by adding the following arguments when running GodExpo.

- "*-wmc number*": Set customized WMC threshold
- "*-atfd number*": Set customized ATFD threshold
- "*-tcc number*": Set customized TCC threshold

Here, number represents the customized threshold value.

IV. Evaluation

A. *Dataset Description*

To evaluate the tool, it has been executed on seven different versions of two popular Go projects.

1) **Hugo:** It is a static HTML and CSS website generator written in Go [11].
2) **Mattermost:** Mattermost is an open source, private cloud, Slack-alternative [12].

Table 1 presents information regarding these two projects. Number of files, LOC, and lines of comment have been counted using SLOC tool [4]. The average LOC of Hugo and Mattermost are 45643 and 777008 respectively.

TABLE 1 Information about the Projects

Project name	Number of stars	Version	Number of files	LOC	Comments
Hugo	31704 (as of 10 Jan 2019)	v0.20	188	33956	4725
		v0.25	286	39343	6101
		v0.30	302	41798	6565
		v0.35	313	45345	7258
		v0.40	341	49221	7806
		v0.45	382	53606	8900
		v0.50	417	56230	9584
Mattermost	13886 (as of 10 Jan 2019)	v5.0.0	2325	732649	87393
		v5.1.0	2362	748446	88740
		v5.2.0	2510	779069	97872
		v5.3.0	2518	785248	98693
		v5.4.0	2540	789929	98917
		v5.5.0	2540	789923	98917
		v5.6.0	2614	813789	101235

B. *Findings*

The findings of the projects are summarized in Table 2. GodExpo has successfully detected a number of God Structs in all versions of the two projects.

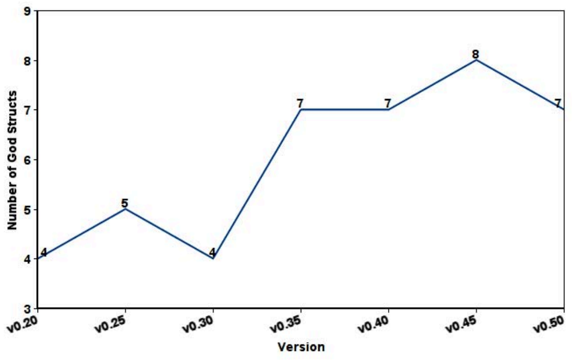
TABLE 2 Findings of the Projects

Project Name	Version	Total Number of Structures	Number of God Structures Detected
Hugo	v0.20	162	4
	v0.25	206	5
	v0.30	224	4
	v0.35	233	7
	v0.40	280	7
	v0.45	343	8
	v0.50	369	7
Mattermost	v5.0.0	4626	51
	v5.1.0	4693	52
	v5.2.0	5151	59
	v5.3.0	5219	59
	v5.4.0	5238	59
	v5.5.0	5238	59
	v5.6.0	5611	60

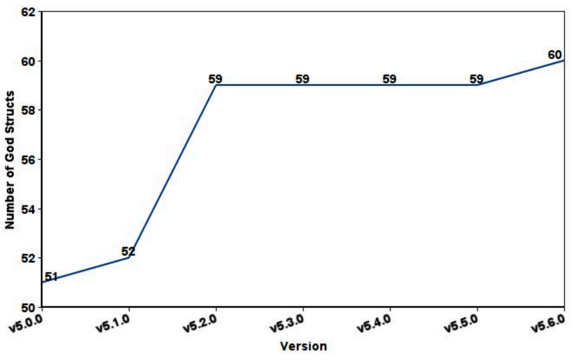
Figure 6 shows that as the number of version increases, the number of God structs generally also increases. This observation conforms to the finding of [18] that the total number of design smells increase with time. Nevertheless, there are two declining points in version 0.30 and version 0.50 of Hugo project. Because, in version 0.25, *commandeer* struct from the *commands* package was a God Struct but two methods were removed from the struct in version 0.30. As a result, its WMC decreased and it was not a God Struct anymore. On the other hand, in version 0.45, *shortcodeHandler* struct was

[4] https://github.com/flosse/sloc

identified as God Struct. Through manual inspection, we found that the struct was no longer a God Struct in version 0.50 due to refactoring.



(a) Hugo



(b) Mattermost

Figure 6 Number of God Structs Detected in Different Versions of the Two Projects

V. Related Work

There exist various tools namely JDeodorant [10] [19], inFusion [5], PMD [5], and JSpIRIT [20] that can detect God Class. JDeodorant is an open source Eclipse plugin for Java. It can detect four code smells: God Class, God Method, Feature Envy, and Type Checking [19]. inFusion is a commercial standalone tool for C, C++ and Java to detect God Class along with 21 other smells [5]. PMD is an open source tool for Java and an Eclipse plugin that detects various smells in Java code, including God Class and God Method [5]. JSpIRIT is an Eclipse plugin for Java that identifies and prioritizes ten code smells, including God Class [20].

To the best of our knowledge, there is no existing tool for God Struct detection of Go programs. Hence, GodExpo is the first tool in Golang to detect God Structs as well as provide version wise result to observe the evolution of God Structs.

VI. Conclusion

This paper presents a command-line tool GodExpo for automatically detecting God Structs in Golang. In addition, GodExpo can provide version wise result to observe the evolution of God Structs. To evaluate GodExpo, it has been executed on seven different versions of two Golang projects namely Hugo [11] and Mattermost [12]. It has successfully detected God Structs in all versions of these two projects. The source code of GodExpo and the dataset are publicly available. Therefore, it will facilitate experiment reproduction as well as conducting comparative studies for further research. In future, GodExpo will be improved to handle anonymous structures as well as provide refactoring suggestions.

References

[1] The Go Project. Retrieved October 20, 2018, from https://golang.org/project/
[2] Seguin, K. (2014). The Little Go Book. Retrieved November 22, 2018, from https://www.openmymind.net/assets/go/go.pdf
[3] The Go Blog. (2017, November 10). Retrieved October 21, 2018, from https://blog.golang.org/8years
[4] Frequently Asked Questions (FAQ). Retrieved March 14, 2019, from https://golang.org/doc/faq#Usage
[5] Paiva, T., Damasceno, A., Figueiredo, E., & SantAnna, C. (2017). On the evaluation of code smells and detection tools. Journal of Software Engineering Research and Development, 5(1), 7.
[6] Olbrich, S. M., Cruzes, D. S., & Sjoberg, D. I. (2010, September). Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. 2010 IEEE International Conference on Software Maintenance (ICSM) (pp. 1-10). IEEE.
[7] Santos, J. A. M., de Mendona, M. G., Dos Santos, C. P., & Novais, R. L. (2014). The problem of conceptualization in god class detection: agreement, strategies and decision drivers. Journal of Software Engineering Research and Development, 2(1), 11.
[8] Lanza, M., and Marinescu, R. (2007). Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media.
[9] Martin, R. C. (2002). Agile software development: principles, patterns, and practices. Prentice Hall.
[10] Fokaefs, M., Tsantalis, N., Stroulia, E., & Chatzigeorgiou, A. (2011, May). JDeodorant: identification and application of extract class refactorings. In 33rd International Conference on Software Engineering (ICSE) (pp. 1037-1039). IEEE.
[11] Hugo. Retrieved January 19, 2019, from https://github.com/gohugoio/hugo
[12] Mattermost. Retrieved January 19, 2019, from https://github.com/mattermost/mattermost-server
[13] Chidamber, S. R., & Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. IEEE TSE, 20 (6), 476-493.
[14] Bieman, J. M., & Kang, B.-K. (1995). Cohesion and Reuse in an Object-Oriented System. Proc. Int'l Symp. Software Reusability, (259-262).
[15] Doxsey, C. (2012). An introduction to programming in Go. CreateSpace.
[16] Aimonetti, M. Go Bootcamp Everything you need to know to get started with Go. Retrieved January 31, 2019, from http://www.golangbootcamp.com/book/methods
[17] Schumacher, J., Zazworka, N., Shull, F., Seaman, C., & Shaw, M. (2010, September). Building empirical support for automated code smell detection. In Proceedings of the 2010 ACM-IEEE Int. Symposium on Empirical Software Engineering and Measurement (p. 8).
[18] Chatzigeorgiou, A., & Manakos, A. (2010, September). Investigating the evolution of bad smells in object-oriented code. In International Conference on the Quality of Information and Communications Technology (pp. 106-115). IEEE.
[19] Tsantalis, N., Chaikalis, T., & Chatzigeorgiou, A. (2008, April). JDeodorant: Identification and removal of type-checking bad smells. In 12th European Conference on Software Maintenance and Reengineering (CSMR) 2008. (pp. 329-331). IEEE.
[20] Vidal, S., Vazquez, H., Diaz-Pace, J. A., Marcos, C., Garcia, A., & Oizumi, W. (2015, November). JSpIRIT: a flexible tool for the analysis of code smells. In 34th International Conference of the Chilean Computer Science Society (SCCC)(pp. 1-6). IEEE.