Name: 1.Harshavardhan.Pullakandam
2.Anil kumar Banoth

Empirical study: Effect of class size on software maintainability

Section 1 : Introduction

Maintainability is a key part of the success of software projects. It is the power to change software without introducing bugs. Class size is a programme metric that counts the number of methods in a class. Larger classes tend to be harder to maintain because they are more complicated and have more code to understand.

This study looked at how the size of a class affects how easy software is to manage. A set of 100 software programmes was collected. The WMC and CC of each class in each software system were tested. During upkeep, the number of bugs found in each software system was also recorded.

Statistical analysis was used to see if there was a link between the size of the class and the number of mistakes found. The results of the study showed that there is a link between the size of the class and the number of problems found. This means that bigger classes are more likely to have flaws than smaller classes.

The results of this study show that developers should avoid making classes that are too big. This will help to make their tools easier to update.

Here are some more information that you may want to include in your abstract:

* The methods used to gather data on class size, WMC, CC, and the number of flaws found. The methods of statistical analysis that were used to find out if there was a link between the size of the class and the number of problems found.
* The limitations of the study, such as the size of the sample and the types of software tools that were used in the study. What the study means for software writers and people who make sure the quality of software.

OBJECTIVE:

To find out how the size of a class affects the software maintainability, how easy it is to manage software.

In particular, we will try to answer the following research questions using our study:

- Does the size of a class have a big effect on how easy it is to manage software?
- What is the link between class size and things like depth of inheritance(DIT), Coupling between Objects(CBO), and cohesion that affect how easy it is to keep software running?
- What do these results mean for software developers and people in the context of software engineering?

We will use a quantitative research method to look at a group of software projects and collect data from them. Statistical tools, like regression analysis, will be used to look at the data.
The results will add to what we know about the maintainability of softwares and will help software developers and software engineering managers to make their software systems easier to maintain.

Here are some of the ways this study could help:

- The results can help software developers write code that is easier to manage by making their classes smaller.
- The results of this study can help software engineering managers better use their resources by letting them know which classes are likely to be hard to keep.
- The results of this study could help software development companies improve their processes by putting in place methods that make classes smaller.

GQM APPROACH:

The Goal-Question-Metric (GQM) model is a way to find, define, and collect software metrics in a systematic way. It's based on the idea that metrics should be used to answer specific questions about the software development process and result.
Three steps make up the GQM method:
1. **Set a goal**: The first step is to set a goal for the measuring project. This could be to improve the quality of the programme, shorten the time it takes to make it, or make customers happier.
2. **Define the questions** : Once the goal is clear, the next step is to figure out what questions need to be answered to reach the goal. These questions should be clear, measurable, important, and set within a certain time frame.
3. **Choose which measures to use**: The last step is to choose the measurements that will be used to answer the questions. The metrics should be correct and reliable, and they should be related to the goal and the questions.

The GQM method is a useful tool for software developers who want to make their goods better. It makes it easy to find, describe, and collect metrics that can be used to measure how well the software development process is working.

Here are some of the good things about using GQM:

- It helps make sure that metrics are used to answer specific questions about the software development process and product.
- It also helps make sure that metrics are important to the goal of the measurement effort. It helps make sure that measurements are correct and trustworthy.
- It helps make sure that measurements are used to improve both the process of making software and the software itself.

SECTION 2: SUBJECT PROGRAMS/DATA SET

Programs To Be Studied:

1.Teknoko Master : The TEKNOKO application assists store employees in carrying out store management processes such as recording daily transaction reports, checking stock items, data on incoming and outgoing transactions, and overseeing inventory management issues in the warehouse. Java based desktop application for store management.
2. E-commerce Website : Java based project of an e-commerce website to buy stuff, add to carts and much more
3. Cinema Ticket Booking System: An application based project for seat reservation in an cinema hall
4. Linked Lists: Data structures projects the involves the implementation of following linked lists - DLDHDTList, SLFLList and SLList
5. Design Patterns Master - Design pattern project demonstrating approach to different recurring problems

SECTION 3: TOOL USED (CK Java Metric)

The CK Java metric is a set of object-oriented metrics that can be used to judge the design and quality of Java programmes. Chidamber and Kemerer came up with the metrics in 1994, and they have been used a lot in study and in the real world.

The 14 measures that make up the CK Java metric are grouped into four groups:

Metrics at the class level measure the size and complexity of classes.

- Coupling metrics: These measure how much two classes depend on each other.
- Cohesion metrics: These metrics measure how much a class's methods and attributes are linked to each other.
- Inheritance metrics: These measurements show how deep and wide inheritance groups are.

The CK Java indicator can be used in a number of ways to judge the quality of Java programmes. For example, it can be used to find classes that are likely to have bugs or to figure out how easy a programme is to fix. You can also use the measure to compare how good different Java programmes are.

The CK Java metric is a useful study tool for learning about how Java programmes are made and how good they are. It can be used to find possible problems early on in the development process and to measure the quality of Java programmes after they have been made.

We have some of the reasons why the CK Java measure is a good tool for research:

- We can use it to figure out how big, complicated, linked, cohesive, and inherited a Java programmes.
- It can be used to find classes that are likely to cause problems in Java programmes.
- It can be used to find out how good a Java programme is. It can be used to see how different Java programmes measure up.

The CK Java metric is a useful tool that can help make Java programmes better. It is a useful tool for both developers and experts.

Section 4 : Metrics to be studied:

- **Depth Inheritance Tree(DIT) -** DIT is a code metric that tracks the longest path between a node in a class hierarchy and the root node. If the DIT is higher, it means that the class is more complicated because it has taken things from more classes. This can make it harder to understand and keep up with the class because it has more code to its name. But this can also make the class more useful because it can take features from its parents.

- **Coupling Between Objects(CBO) -** CBO is a software parameter that shows how much classes in a software system depend on each other. A high CBO means that the connections between classes are strong, which can make the system hard to understand, keep up, and add to. By reducing CBO, software developers can create systems that are more modular, flexible, and easier to maintain.
- **Line of Code(LOC) -** lesson size is a way to figure out how hard a lesson is. The amount of lines of code in a class is used to figure it out. A class with more members is more complicated, which can make it harder to understand, keep up with, and add to.

Process

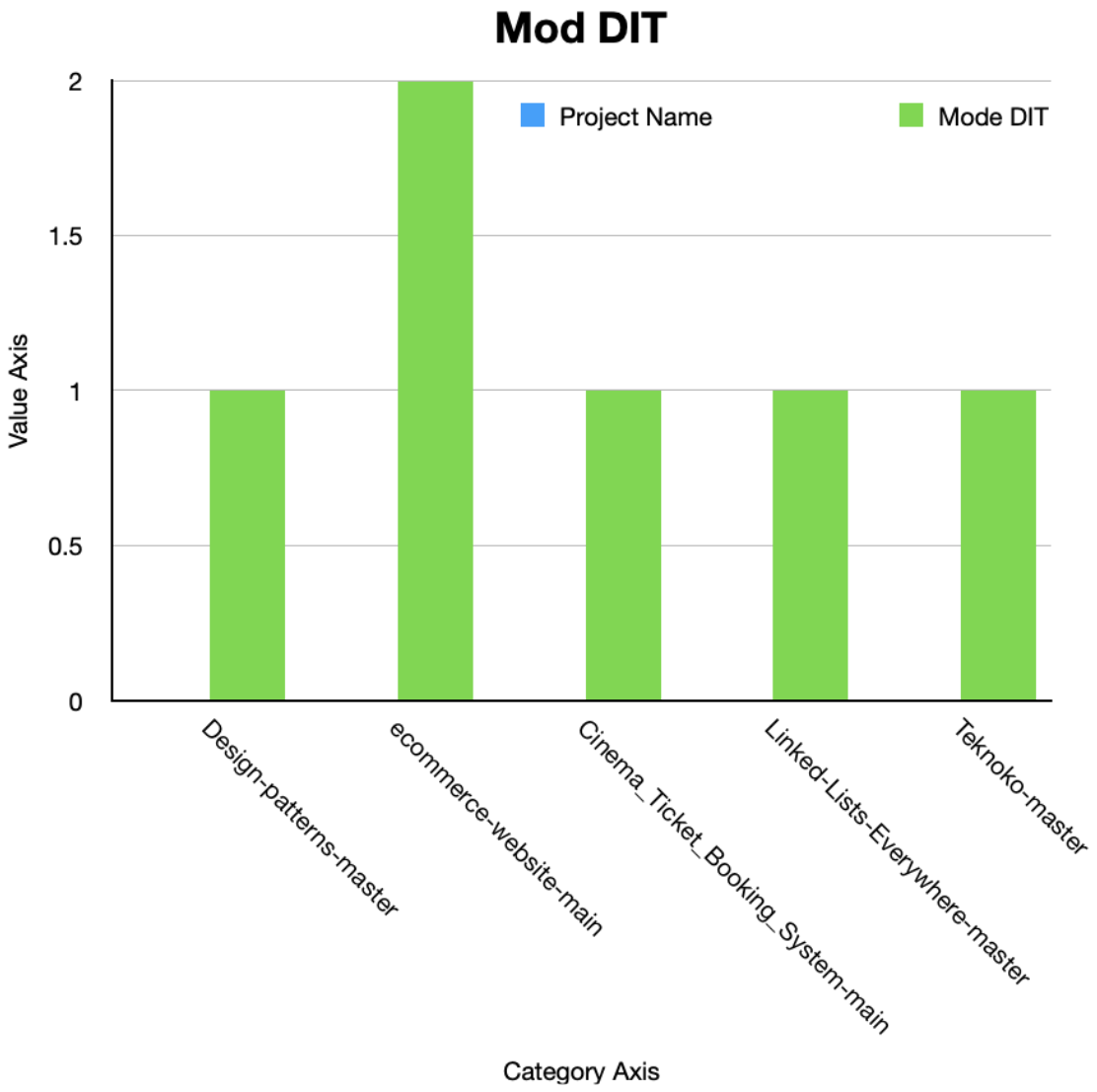
CK Java Metric is a set of instruction and stand alone program which when executed exports a set of csv files from the selected project that includes different metrics on which the the software maintainability depends and out of which DIT and CBO are the significant metrics

Analysis:

- **Depth Inheritance Tree**

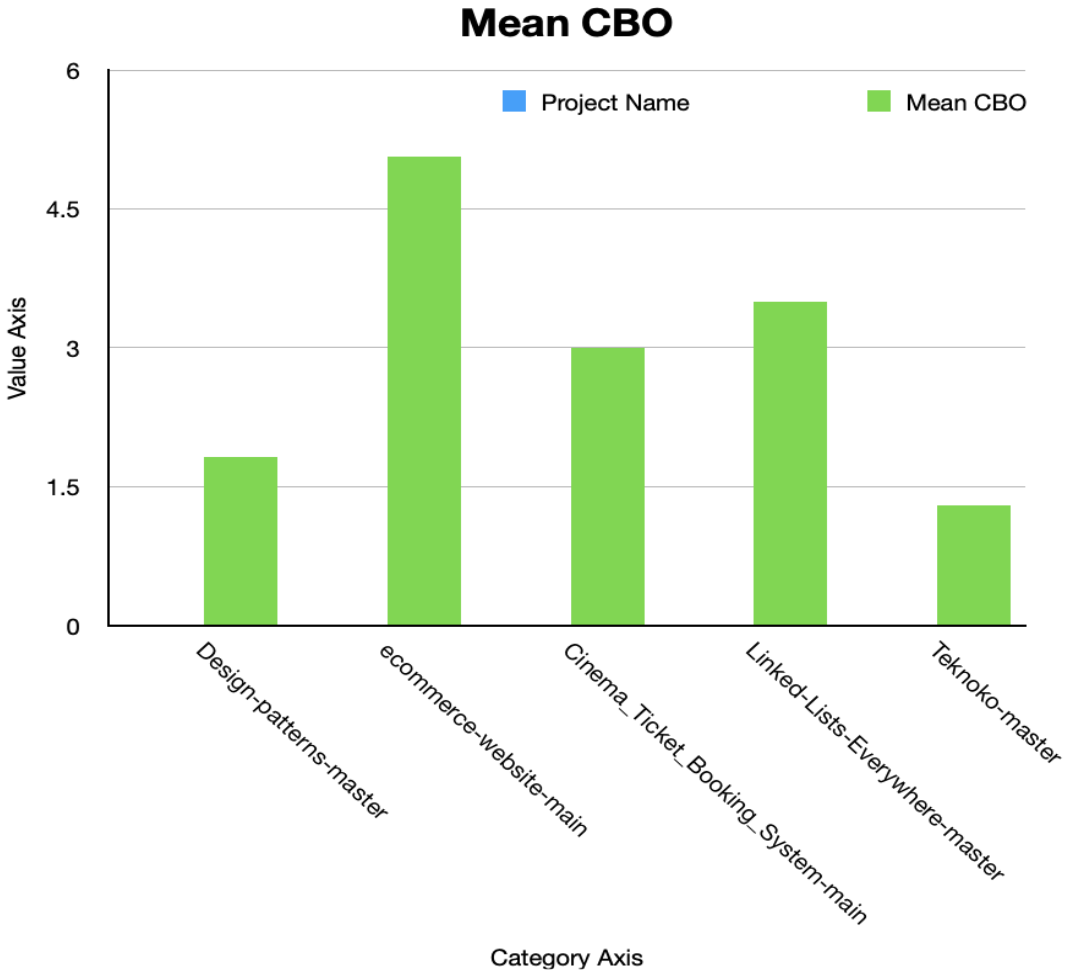
Sno	Project Name	No of classes	Mode DIT	Max DIT
1	Design-patterns-master	67	1	2
2	ecommerce-website-main	31	2	2
3	Cinema_Ticket_Booking_Syst	14	1	2

	em-main			
4	Linked-Lists-Everywhere-master	22	1	2
5	Teknoko-master	65	1	6



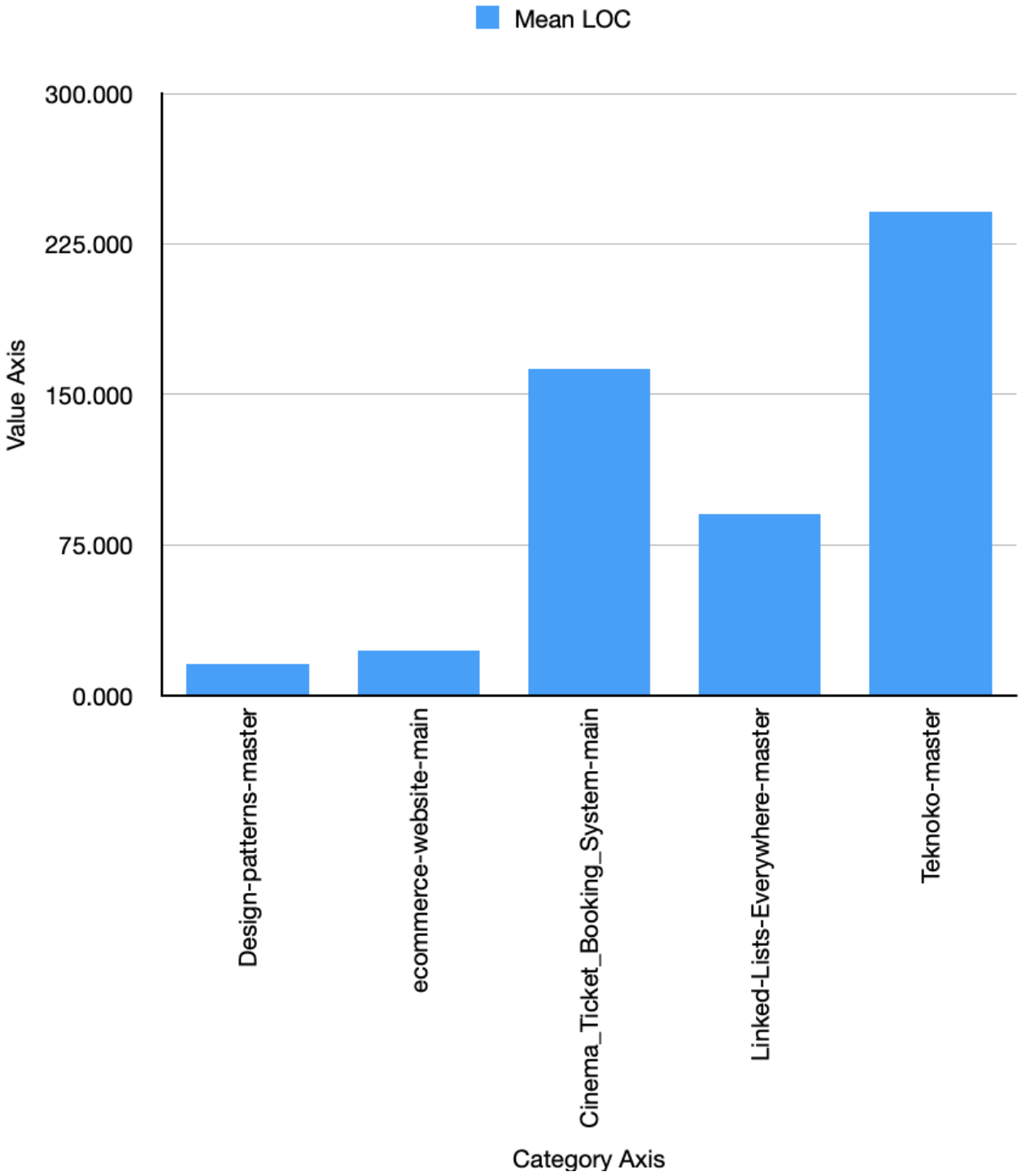
- **Coupling Between Objects**

Sno.	Project Name	CBO(min)	CBO(max)	Mean CBO
1	Design-patterns-master	0	9	1.8208
2	ecommerce-website-main	0	11	5.0645
3	Cinema_Ticket_Booking_System-main	0	12	3
4	Linked-Lists-Everywhere-master	0	6	3.5
5	Teknoko-master	0	9	1.2923



- Class Size

Sno	Project Name	No of classes	Max LOC	Mean LOC
1	Design-patterns-master	165	73	15.890
2	ecommerce-website-main	33	156	22.486
3	Cinema_Ticket_Booking_System-main	64	529	162.65
4	Linked-Lists-Everywhere-master	130	255	90.3
5	Teknoko-master	174	752	241.183

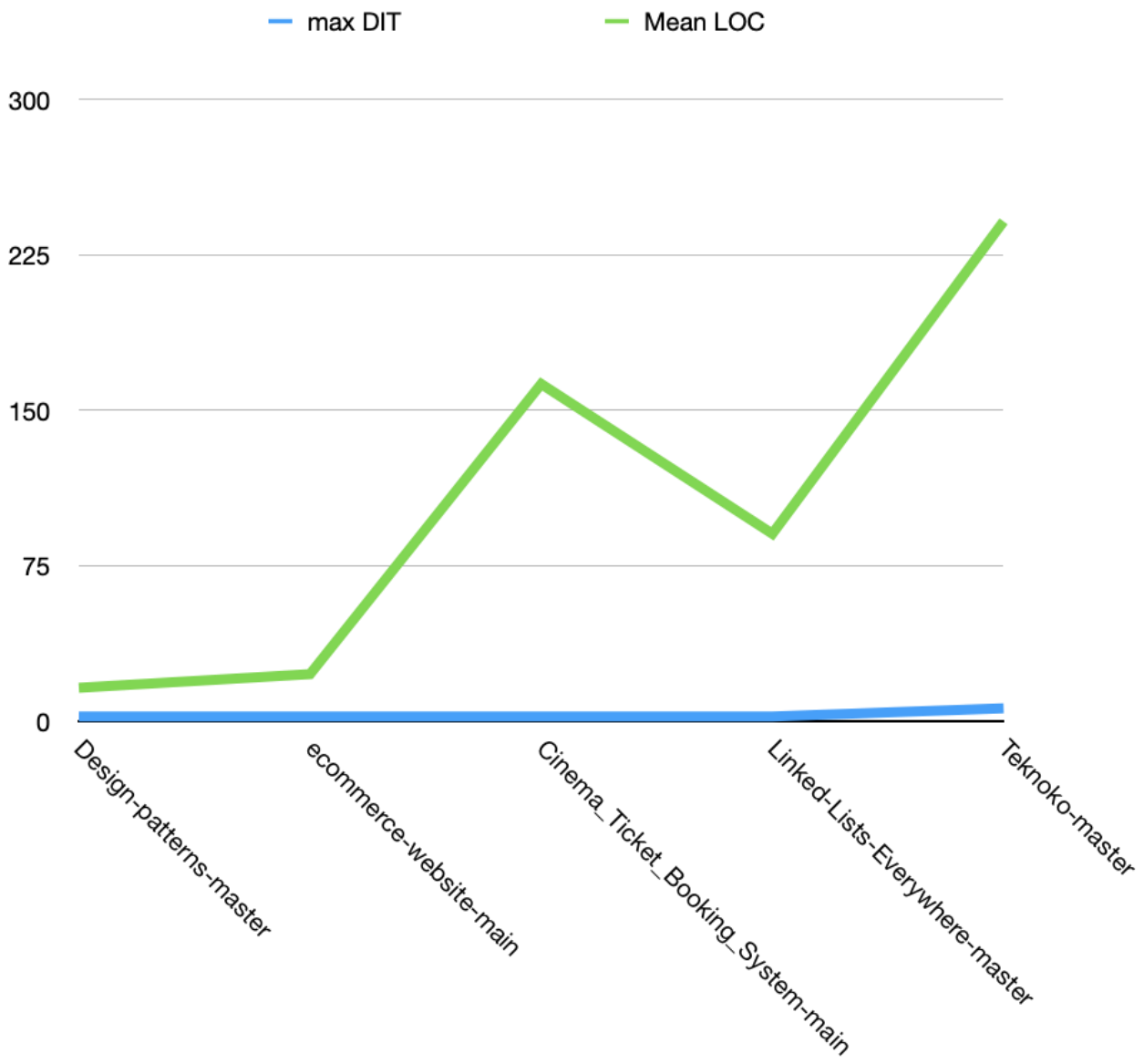


These key findings will help us plot a relation on how the the metrics affect the software maintainability

How Class Size and other metrics are affecting Software Maintainability

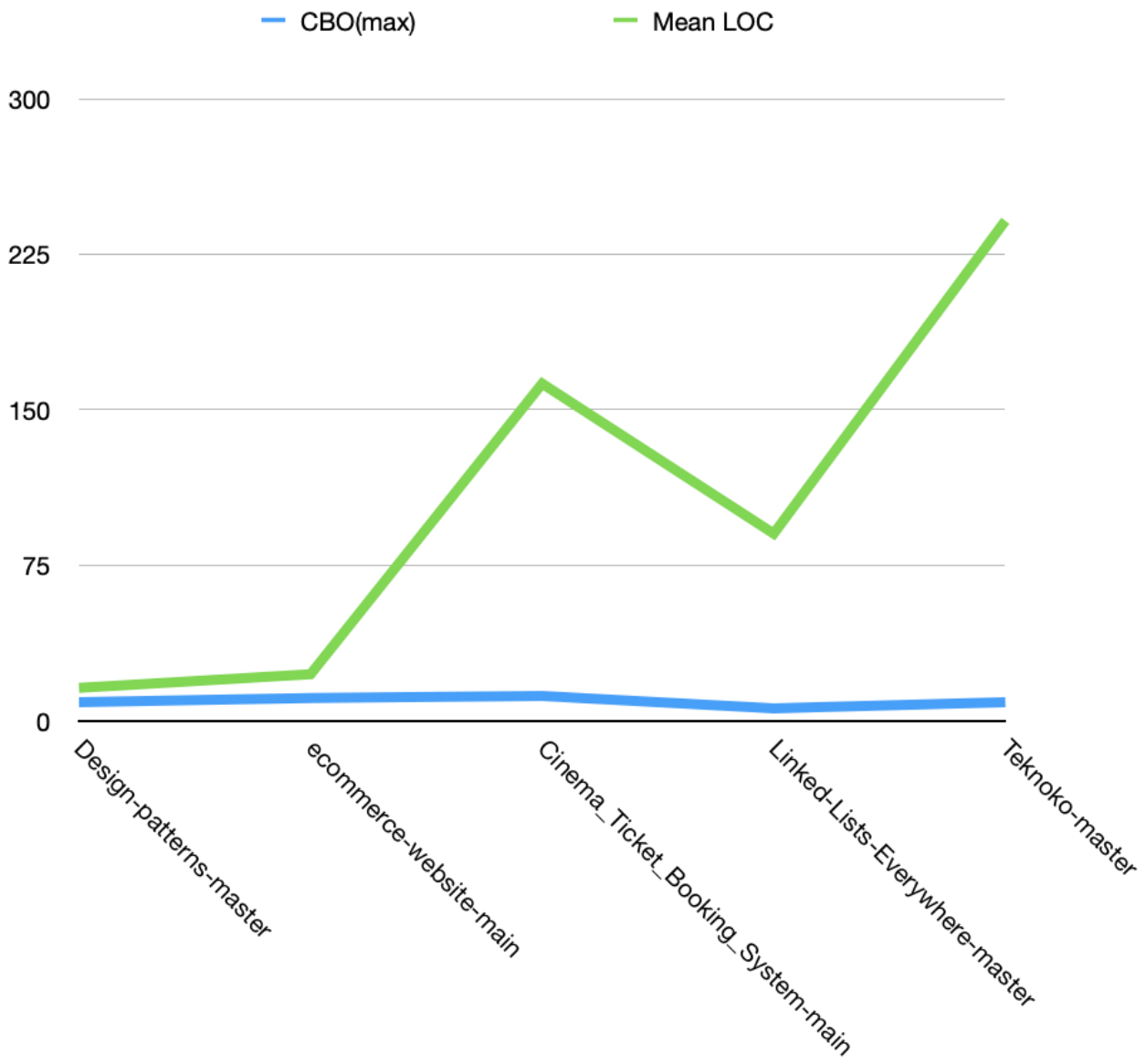
Let’s recap on what we know about ck metric and try exploring few metric which and how do they affects the software maintainability
CK metric is set of different metrics that holds different properties/characteristics of the classes in a java project, The metrics are WMC,DIT,NOC,CBO,LCOM
We already know, The number of lines of code (LOC) in a software system is a common metric used to measure its complexity. In general, a system with more LOC is more complex and therefore more difficult to maintain. This is because more LOC means more code to read, understand, and modify.

Correlation between LOC and DIT
The connection between LOC and DIT is important because it can help find software systems that may be hard to understand and keep up with. A high DIT can be a sign of code bloat, which is when more code is added than is needed to make a feature work. Code bloat can make a software system harder to understand and keep up to date, and it can also slow it down.
We will try plotting a comparison chart between LOC and DIT for the selected projects



Correlation between LOC and CBO

LOC and CBO can be correlated for reasons like, more LOC means more object creation. Each new object will likely interact with other objects to complete its tasks. Second, more LOC increases the likelihood of object modifications. This can modify object interactions, increasing coupling.
LOC-CBO correlations can reveal code base issues. A code base with a high CBO may be hard to understand and maintain. Defects and maintenance expenses may rise.
We will try plotting a comparison chart between LOC and CBO for the selected projects



We can use these metrics to figure out how easy a software system is to maintain, since higher numbers for these metrics can mean that the system is more complicated and therefore harder to maintain.

DIT counts how many times a class inherits from its mother class. If a class has a bigger DIT value, it means that it is deeper in the inheritance hierarchy. This can make it harder to understand the code and behavior of the class, since it may get methods and properties from more than one source class.

LOC is a way to figure out how big a piece of software is. A system is bigger and more complicated if it has a higher LOC number. This can make it harder to understand the architecture and design of the system and harder to find and fix bugs.

CBO counts how many other classes a class has direct interactions with. A class is more closely linked to other classes if its CBO number is high. This can make it harder to change the code of the class, since any changes could affect other classes that it works with.

In summary, the higher the values of the metrics, lower will be the software maintainability

Note : In the analysis mentioned above, we plotted charts for different metrics like DIT, CBO and LOC followed by understanding how they affect the maintainability and complexity of a software. Conclusively we can understand and examine each classes/project's complexity through the plotted charts and values

Section 5 : Conclusion

The conclusions we can draw from the study are

- "Class size has a negative effect on how easy it is to maintain software." Classes that are bigger are harder to understand, change, and test. This is because they have more code and more complicated links between the parts of the code.
- The depth of inheritance (DIT) is another thing that can make software hard to manage. Classes with a higher DIT are harder to understand because they inherit from more classes. This can make it hard to change the code or find bugs in it.
- The amount of class-to-class dependencies (CBO) is another thing that can hurt the maintainability of software. Classes with a higher CBO are harder to understand because they depend on other classes more closely. This can make it hard to change the code or find bugs in it.

We used the GQM approach and the ck metric tool, which led us to these findings. The GQM method is a way to define and measure software quality in a systematic way. The ck metric tool is a software metrics tool that can be used to measure a software system's classes' size, level of inheritance, and number of dependencies between classes.

Based on the results of the study, software developers should take steps to reduce the size, amount of class-to-class dependencies, and depth of inheritance of their software systems' classes. This can be done by using good software design practices, such as making classes smaller, making inheritance structures flatter, and making classes less tightly linked to each other. By doing these things, developers can make their software systems easier to manage, which can lead to lower maintenance costs and faster time to market.

Here are some more suggestions for people who make software:

- Use a range of software design techniques to make classes that are smaller and work better together. * Don't use deep inheritance hierarchies.
- Use loose coupling between classes to make changes to one class less likely to affect other classes.
- Use a variety of software testing methods to make sure the code is easy to understand and change. Document the code well to make it easier to understand and manage.

References

- Henderson-Sellers, Brian, Larry L. Constantine and Ian M. Graham. “Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design).” Object Oriented Systems 3 (1996): 143-158.
- @manual{aniche-ck,
  title={Java code metrics calculator (CK)},
  author={Maurício Aniche},
  year={2015},
  note={Available in https://github.com/mauricioaniche/ck/}
  }
- Wedyan, F., Alrmuny, D., & Bieman, J. M. (2009). The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction. 2009 International Conference on Software Testing Verification and Validation.
- Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476-493
- Github Repos
    - Design-patterns-master
    - Ecommerce-website-main
    - Cinema_Ticket_Booking_System-main
    - Linked-Lists-Everywhere-master
    - Teknoko-master