https://github.com/Fubar-Cyber/lesson8

lesson8

collabarative assignment

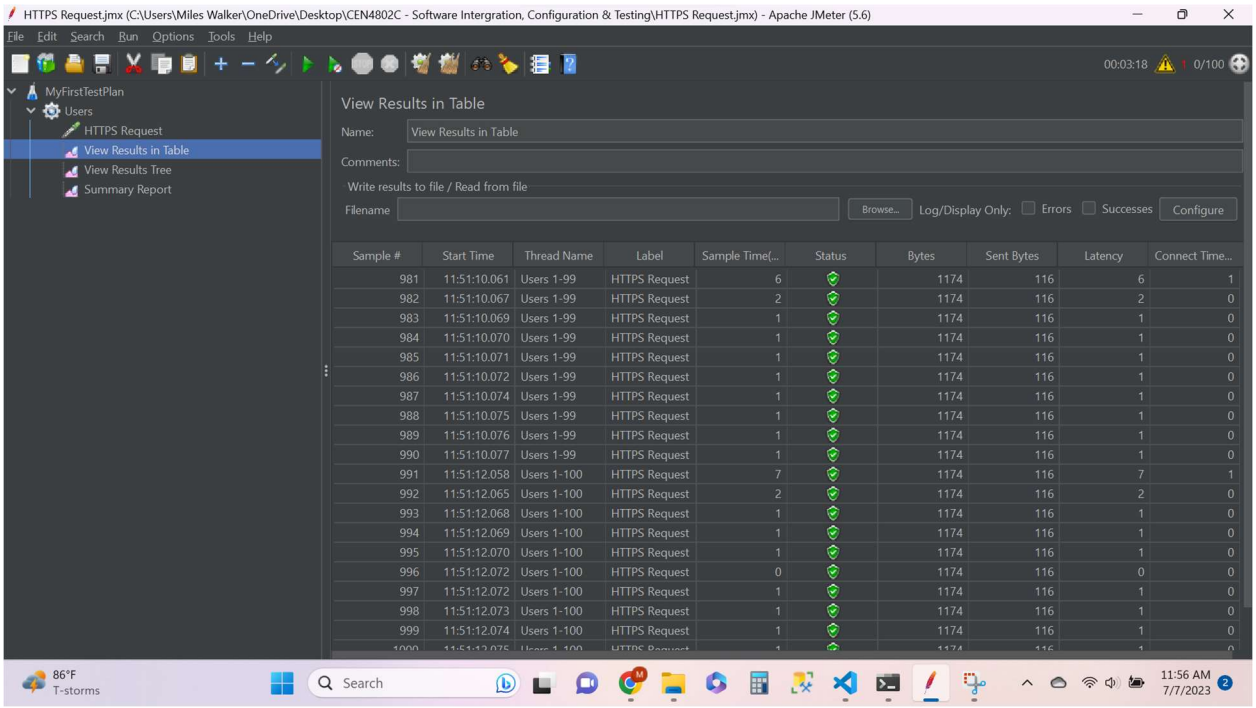
This is a group assignment. Please find a group (either 2 or 3 people for each group). Add yourself to a group under "People --> Module 8 Assignment Groups". Only one person in each group needs to submit the assignment. You can choose which group member's "to do list" application to use for this assignment (i.e., the group member with the most complete application).

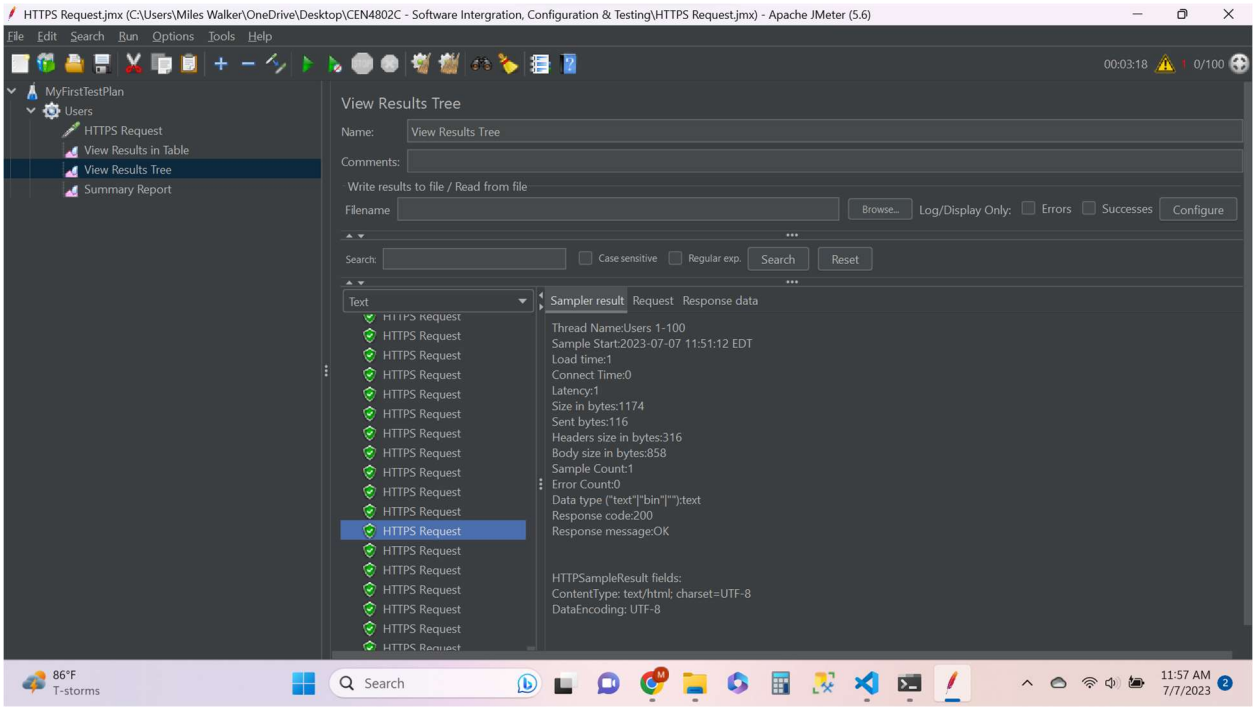
[23 points] Use JMeter to perform load testing on your "To Do List" application's web UI. Submit screen shots of JMeter, showing the resulting performance numbers and graphs.

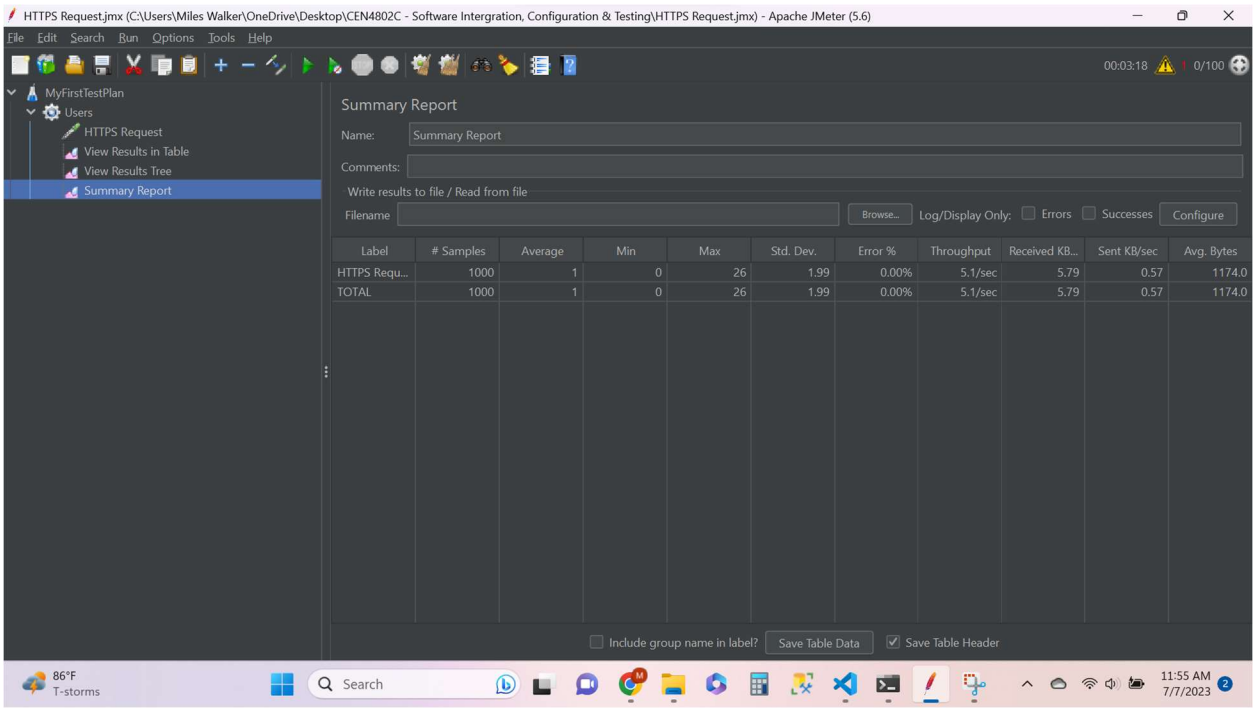
[7 points] Create a table of 10 anti-patterns and 10 code smells that you learned about in the Review section, with a row for each anti-pattern and code smell. It should include the following columns,

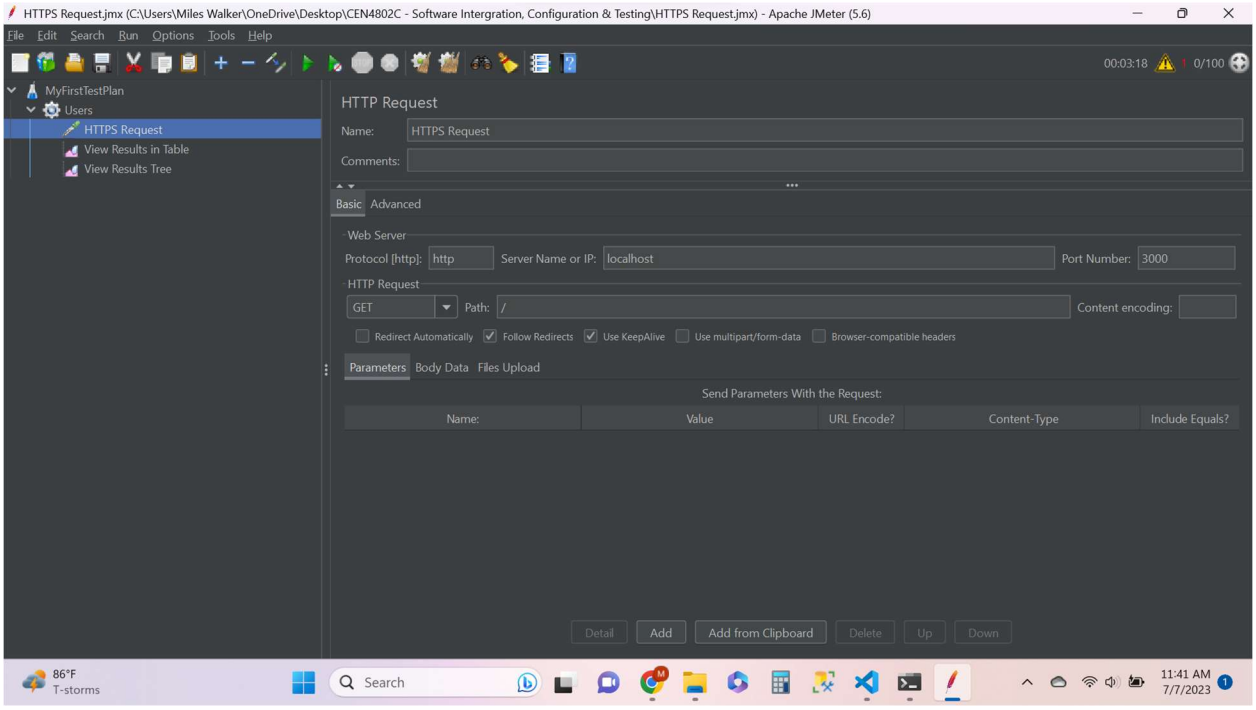
Name Category (architectural, application-level, class-level, etc.) Severity (1-10), with higher severities resulting in worse maintainability Then, answer the following questions,

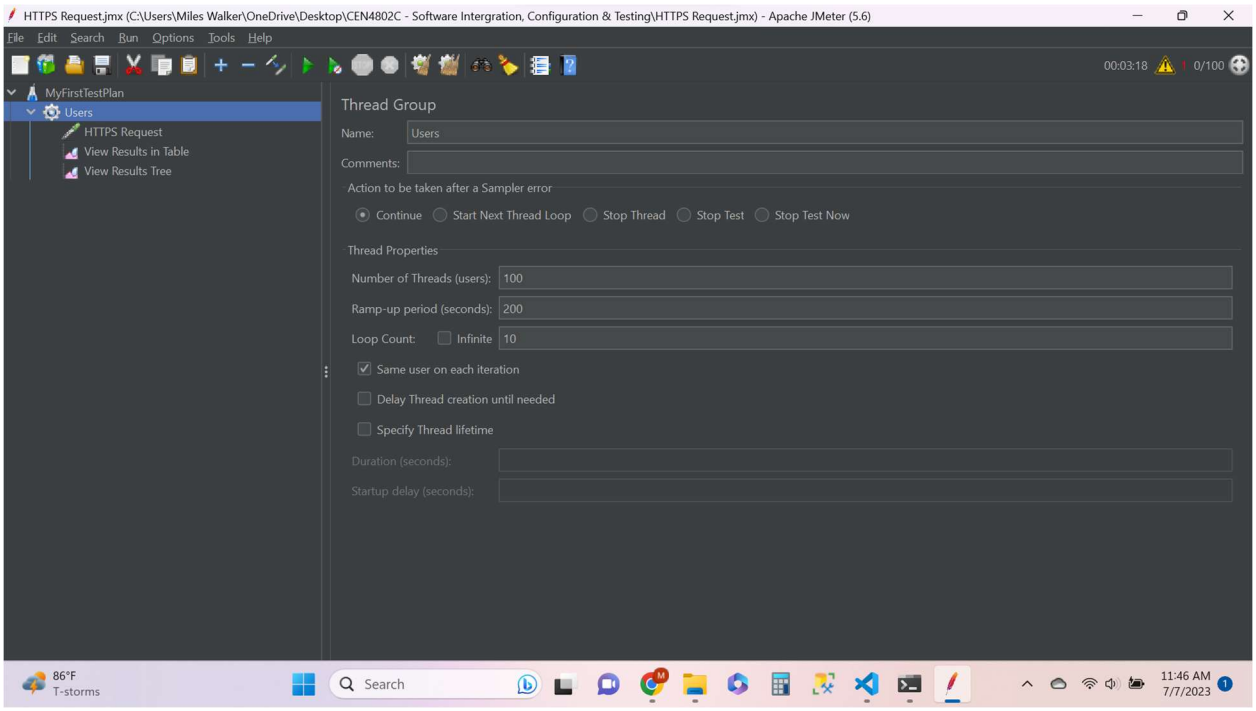
How are code maintainability and simplicity related? Runtime efficiency and code simplicity are often competing goals. How can you deal with this problem? Is it possible to have code that is both simple and efficient?


HTTPS Request.jmx (C:\Users\Miles Walker\OneDrive\Desktop\CEN4802C - Software Intergration, Configuration & Testing\HTTPS Request.jmx) - Apache JMeter (5.6)
File Edit Search Run Options Tools Help
00:03:18 0/100
MyFirstTestPlan
Users
HTTPS Request
View Results in Table
View Results Tree
Summary Report
View Results in Table
Name: View Results in Table
Comments:
Write results to file / Read from file
Filename Browse... Log/Display Only: Errors Successes Configure
Sample # Start Time Thread Name Label Sample Time(... Status Bytes Sent Bytes Latency Connect Time...
981 11:51:10.061 Users 1-99 HTTPS Request 6 1174 116 6 1
982 11:51:10.067 Users 1-99 HTTPS Request 2 1174 116 2 0
983 11:51:10.069 Users 1-99 HTTPS Request 1 1174 116 1 0
984 11:51:10.070 Users 1-99 HTTPS Request 1 1174 116 1 0
985 11:51:10.071 Users 1-99 HTTPS Request 1 1174 116 1 0
986 11:51:10.072 Users 1-99 HTTPS Request 1 1174 116 1 0
987 11:51:10.074 Users 1-99 HTTPS Request 1 1174 116 1 0
988 11:51:10.075 Users 1-99 HTTPS Request 1 1174 116 1 0
989 11:51:10.076 Users 1-99 HTTPS Request 1 1174 116 1 0
990 11:51:10.077 Users 1-99 HTTPS Request 1 1174 116 1 0
991 11:51:12.058 Users 1-100 HTTPS Request 7 1174 116 7 1
992 11:51:12.065 Users 1-100 HTTPS Request 2 1174 116 2 0
993 11:51:12.068 Users 1-100 HTTPS Request 1 1174 116 1 0
994 11:51:12.069 Users 1-100 HTTPS Request 1 1174 116 1 0
995 11:51:12.070 Users 1-100 HTTPS Request 1 1174 116 1 0
996 11:51:12.072 Users 1-100 HTTPS Request 0 1174 116 0 0
997 11:51:12.072 Users 1-100 HTTPS Request 1 1174 116 1 0
998 11:51:12.073 Users 1-100 HTTPS Request 1 1174 116 1 0
999 11:51:12.074 Users 1-100 HTTPS Request 1 1174 116 1 0
86°F T-storms
Search
11:56 AM 7/7/2023


HTTPS Request.jmx (C:\Users\Miles Walker\OneDrive\Desktop\CEN4802C - Software Intergration, Configuration & Testing\HTTPS Request.jmx) - Apache JMeter (5.6)
File Edit Search Run Options Tools Help
00:03:18 0/100
MyFirstTestPlan
Users
HTTPS Request
View Results in Table
View Results Tree
Summary Report
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename Browse... Log/Display Only: Errors Successes Configure
Search: Case sensitive Regular exp. Search Reset
Text
HTTPS Request
Sampler result Request Response data
Thread Name:Users 1-100
Sample Start:2023-07-07 11:51:12 EDT
Load time:1
Connect Time:0
Latency:1
Size in bytes:1174
Sent bytes:116
Headers size in bytes:316
Body size in bytes:858
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK
HTTPSampleResult fields:
ContentType: text/html; charset=UTF-8
DataEncoding: UTF-8
86°F T-storms
Search
11:57 AM 7/7/2023


HTTPS Request.jmx (C:\Users\Miles Walker\OneDrive\Desktop\CEN4802C - Software Intergration, Configuration & Testing\HTTPS Request.jmx) - Apache JMeter (5.6)
File Edit Search Run Options Tools Help
00:03:18 0/100
MyFirstTestPlan
Users
HTTPS Request
View Results in Table
View Results Tree
Summary Report
Summary Report
Name: Summary Report
Comments:
Write results to file / Read from file
Filename Browse... Log/Display Only: Errors Successes Configure
Label # Samples Average Min Max Std. Dev. Error % Throughput Received KB... Sent KB/sec Avg. Bytes
HTTPS Requ... 1000 1 0 26 1.99 0.00% 5.1/sec 5.79 0.57 1174.0
TOTAL 1000 1 0 26 1.99 0.00% 5.1/sec 5.79 0.57 1174.0
Include group name in label? Save Table Data Save Table Header
86°F T-storms
Search
11:55 AM 7/7/2023


HTTPS Request.jmx (C:\Users\Miles Walker\OneDrive\Desktop\CEN4802C - Software Intergration, Configuration & Testing\HTTPS Request.jmx) - Apache JMeter (5.6)
File Edit Search Run Options Tools Help
00:03:18 0/100
MyFirstTestPlan
Users
HTTPS Request
View Results in Table
View Results Tree
HTTP Request
Name: HTTPS Request
Comments:
Basic Advanced
Web Server
Protocol [http]: http Server Name or IP: localhost Port Number: 3000
HTTP Request
GET Path: / Content encoding:
Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data Browser-compatible headers
Parameters Body Data Files Upload
Send Parameters With the Request:
Name: Value URL Encode? Content-Type Include Equals?
Detail Add Add from Clipboard Delete Up Down
86°F T-storms
Search
11:41 AM 7/7/2023



Runtime efficiency and code simplicity are often competing goals. How can you deal with this problem? Is it possible to have code that is both simple and efficient?

There can be a trade-off between runtime efficiency and code simplicity. However, it is possible to strike a balance and have simple and efficient code. Here are some strategies to deal with this problem:

Focus on Algorithmic Efficiency: Before diving into code optimization, consider whether there are more efficient algorithms or data structures that can solve the problem. Choosing the right algorithm can often lead to significant performance improvements without sacrificing code simplicity. Understanding the problem domain and exploring algorithmic alternatives are crucial steps in achieving efficiency.

Prioritize Readability and Maintainability: Start by writing clear, readable, and maintainable code. Prioritizing simplicity allows for easier understanding, debugging, and modification of code. Well-structured and maintainable code provides a solid foundation for future optimizations without sacrificing the overall quality of the codebase.

Profile and Benchmark: Identify critical sections of your code that contribute the most to the overall runtime. Profile your code to identify bottlenecks and areas that can

benefit from optimization. Use appropriate benchmarking techniques to measure the impact of your changes and ensure that your optimizations are effective.

Optimize Bottlenecks: Once you've identified performance bottlenecks, focus your optimization efforts on those specific areas. Analyze the code and algorithms within these bottlenecks and look for opportunities to optimize them. Often, small tweaks or algorithmic improvements in critical sections can yield significant performance gains.

Utilize Efficient Data Structures and Algorithms: Use data structures and algorithms that are known for their efficiency in solving specific problems. For example, using a hash table or a binary search tree instead of a linear search can significantly improve performance. Leverage libraries and frameworks that provide optimized implementations for common operations to avoid reinventing the wheel.

Apply Code-Level Optimizations: Look for opportunities to optimize the code itself without sacrificing readability. Techniques such as loop unrolling, caching frequently accessed data, minimizing unnecessary calculations, and avoiding redundant operations can improve performance without introducing excessive complexity.

Measure and Iterate: Continuously measure the impact of your optimizations and make iterative improvements. Test your code with realistic data and workload scenarios to ensure that the optimizations hold up under real-world conditions. Regularly revisit and review the optimized code to maintain its simplicity and readability.

Remember that the right balance between simplicity and efficiency depends on the specific context and requirements of your project. It's important to consider factors such as the expected input size, system constraints, and the trade-offs acceptable in your particular use case. By employing a systematic approach, you can achieve simple and efficient code, providing a maintainable foundation with optimal performance.

How are code maintainability and simplicity related?

Code maintainability and simplicity are closely related concepts in software development. Simplifying code often leads to better maintainability, and maintaining code becomes easier when it is designed with simplicity in mind. Let's explore this relationship further.

1. Readability: Simple code is typically easier to read and understand. When code is clear and concise, developers can quickly grasp its purpose and functionality. This

improves maintainability because it reduces the time and effort required to comprehend the codebase, which is especially beneficial when multiple developers are working on the same project.

2. Debugging and Troubleshooting: Simple code is less prone to bugs and easier to debug. Complex and convoluted code can introduce unnecessary complexity, making identifying and fixing issues harder. By keeping the code simple, developers can isolate problems more effectively and make changes or enhancements without inadvertently introducing new bugs.

3. Modularity and Reusability: Simplicity often leads to modular code structures, where different components are organized and encapsulated logically. Modular code is easier to maintain because it allows developers to focus on specific sections without affecting the entire codebase. Furthermore, modular code tends to be more reusable, as individual components can be leveraged in different contexts, reducing duplication and improving maintainability.

4. Refactoring: Simplifying code is often an essential step during the refactoring process. Refactoring involves restructuring existing code to improve its design, readability, and maintainability while preserving its functionality. Simplification techniques, such as removing duplication, eliminating unnecessary complexity, and applying design patterns, make codebases more maintainable over time.

5. Documentation: Simple code requires less documentation. When code is self-explanatory and follows established conventions and best practices, it becomes more understandable without extensive documentation. While documentation is still valuable for complex logic or system-wide explanations, reducing the need for excessive documentation improves maintainability because developers can rely more on the code itself for understanding.

6. Collaboration: Simple code promotes collaboration among team members. When code is clear, concise, and follows consistent patterns, it becomes easier for developers to work together, review each other's code, and provide feedback. Collaboration improves maintainability by fostering knowledge sharing, code consistency, and collective ownership of the codebase.

In summary, code maintainability and simplicity are closely intertwined. Simplicity improves code maintainability by enhancing readability, facilitating debugging, promoting modularity and

reusability, supporting refactoring efforts, reducing the need for extensive documentation, and enabling effective collaboration. By striving for simplicity,

developers can create codebases that are easier to understand, modify, and maintain throughout their lifespan.

Module 8 Assignment		
Name	**Category**	**Severity 1-10**
The Blob	Application	3
Lava Flow	Application	5
Functional Decomposition	Application	2
Stovepipe Enterprise	Enterprize-Level	4
Stovepipe System	System-Level	3
Architecture by Implication	System-Level	3
Analysis Paralysis	System-Level	6
Death by Planning	Enterprize-Level	2
Corncob	Enterprize-Level	3
Irrational Management	Enterprize-Level	5
Large Class	Class-Level	3
Data Clumps	Class-Level	3
Switch Statements	Object-Level	5
Alternative Classes with Different Interfaces	Class-Level	4
Shotgun Surgery	Class-Level	6
Parallel Inheritance Hierarchies	Class-Level	4
Lazy Class	Class-Level	3
Dead Code	Class-Level	2
Inappropriate Intimacy	Class-Level	4
Middle Man	Class-Level	2