Software Engineering for Computational Science and Engineering

Chelsea Davis

December 3, 2012

Abstract

There is a need for more effecient and reliable scientific software. We investigated whether or not design patterns are beneficial when writing scientific software. Developers were divided into two groups where one group used design patterns and the other did not. The programs that used design patterns had less fatal errors, greater complexity spread over a greater number of functions, and longer programs. We conclude that design patterns would be beneficial in writing scientific software and making it more effecient.

1 Introduction

Scientific software is often inefficient and too specific for reuse because the software is usually done by the scientist in the subject field and not a software engineer. Scientists also have to worry about producing time efficient and accurate software. To address this problem design patterns have started being used in the development of scientific software. There has been much debate about whether or not design patterns are beneficial or even applicable to the development of scientific software. Patterns are a well-understood methodology for object-oriented software architecture, especially for business applications. But what relevance, if any, do they have to scientific software? [1] Design patterns, as defined by[1], are standard models of a structure or process that can be applied to specific cases in a consistent way. Design patterns have been used for years in object-oriented programming but just recently begun being used for scientific software. There are three common types of patterns; they are creational, structural, and behavioral. Creational patterns typically define different ways of creating objects. Structural patterns typically define relations between classes and objects. Behavioral patterns typically define the behavior of objects. As system size increases there is a greater need for reliable, scalable, extensible, reusable, and maintainable software. [1]

Design patterns help address these needs for large system sizes in scientific software. Patterns are helpful because programmers know that similar code to theirs has been successful in the past for similar problems. Therefore, patterns

provide standard successful code structures. Another issue facing scientific software is the fact that most scientific software is written by a scientist who is an expert in the field needed for the software, and not an expert in software engineering. For example, a biologist would write the program to model cell division because they understand cell division. While this program may work it probably wont be very efficient because most biologists dont have extensive knowledge of software engineering. [2] says "because CSE projects often investigate new scientific findings, the software's expected output is sometimes unknown, making it difficult or impossible to define test oracles." Common design patterns would help a biologist write a more efficient and reliably correct program. So scientists need to be confident that their prorams are producing possible results. More communication between scientists and software engineers would also be beneficial. [3] claims "the conclusion is that there is substantial potential for research and collaboration between scientists writing software and the software engineering community." During our research my fellow researchers and I learned about design patterns and then implemented them.

2 Methods

To test if design patterns are truly beneficial in scientific software, we analyzed their use in the development of the program the Game of Life. Two researchers intentionally used design patterns and two did not when writing their code individually. We used MPI, Message Passing Interface, to implement the program and test it using different problem sizes and nodes. The Game of Life is modeled using cellular automata. Each cell is represented by an element in a matrix as either a zero or one. A zero is a dead cell and a one is a live cell. At each step, a cells state is determined by its neighbors states and its own state. The rules for a cells state change are below:

1. A live cell with zero or one live neighbors dies from loneliness.
2. A live cell with four or more live neighbors dies from overpopulation.
3. A dead cell with two or three live neighbors becomes alive.
4. Otherwise, a cells state stays unchanged.

All four programs accepted three command line arguments. Two dimension values, M and N, for the dimension of the matrix, and the number of iterations, K, the game will go through. Once the dimensions of the matrix are inputted, the cells initial states are randomly assigned using a random number generator. We ran this program on a cluster of four computers, so that we would have four process nodes, a master node and three children nodes. The master node sent different rows of the matrix to the children nodes, received the updated cells, and printed the resulting matrix. Then we collected data from the four programs and compared them based on who used design patterns and who did not. Figure 1 is sample output of the program.

Figure 1: Sample matrices of program

Start:	Round 0:	Round 1:
0 0 0 1	0 0 1 0	0 1 0 1
0 0 0 1	0 0 1 0	1 0 1 1
0 1 0 0	1 1 0 1	1 0 0 1
1 1 1 0	1 0 1 0	1 0 1 1
0 1 0 0	0 0 0 0	1 0 1 1
0 1 0 0	1 0 1 0	0 1 0 0

3 Results

Through the graphs we can see that using design patterns seems to have caused a great difference in terms of: time spent coding, code length, error types, and complexity distribution. Figure 2 shows design pattern programs took a longer amount of time to write and thus, were compiled more often. Figure 3 shows design pattern programs had greater total complexity and function counts, but lower average complexity. The benchmark we used to calculate the total complexity was based off the number of methods and length of the programs. Figure 4 shows design pattern programs had a lot more lines of code which included comments. Figure 5 shows the types of errors that occurred when writing the code and the percentage of each type out of the group's total number of errors.

4 Conclusions

Based on these results, we can conclude that design patterns do help with the development, complexity, and time efficiency of scientific software. However, we do need to explain other possible reasons for the differences. The results may have been influenced by one design pattern developer using bitwise memory management, leading to increased program difficulty. Bitwise memory management is using the computer's natural binary storage to store the matrix of zeros and ones. This method saves a lot of space in memory but is very difficult to code, especially if the programmer is not familiar with it. Also, this method ran and produced results faster as a consequence. Plus, the researchers coding abilities with C would have affected the results, especially the error results. None of the researchers had coded in C before this experiment. There are many factors that could have influenced this experiment that we cannot measure to accurately take into account for and explain different results. Also, the Game of Life program is not a very complex problem so doing this experiment again with a more complex program would better show the effect using design patterns has and how they can be useful for complex scientific programs. We would also want to have a larger sample of programmers to more conclusively determine

Figure 2: Shows that those who used design patterns spent more time on their program and compiled more often.

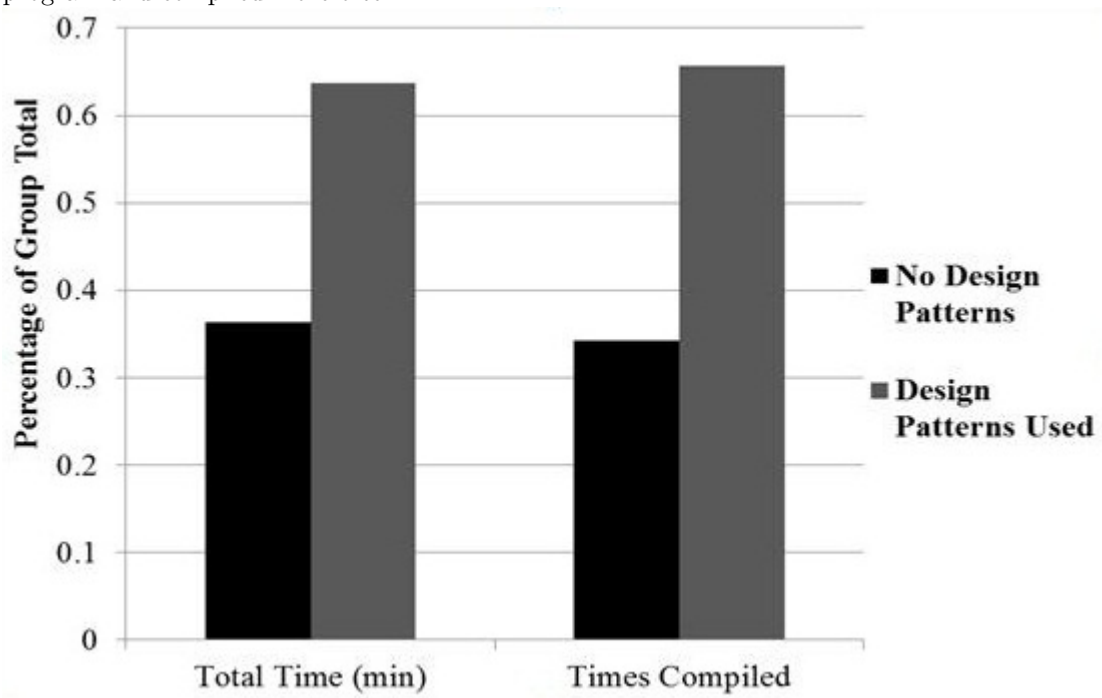


Figure 3: This graph shows that design patterns resulted in program complexity spreading over a greater number of functions.

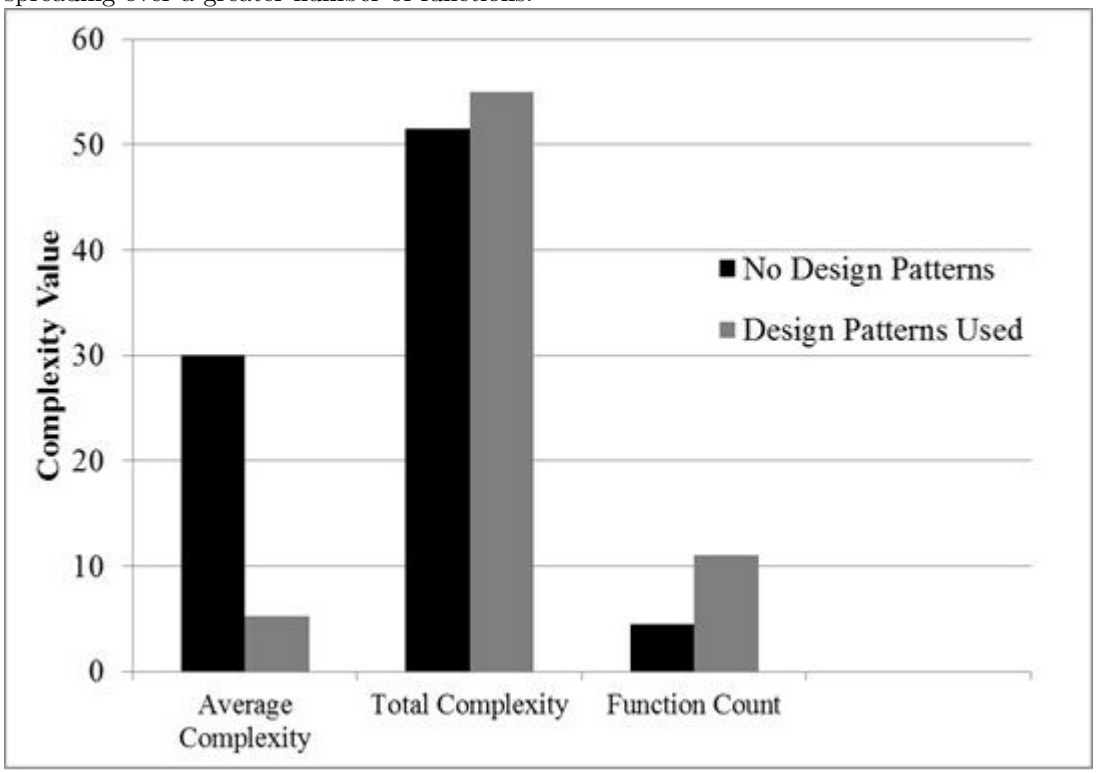


Figure 4: This graph charts code progression as total lines of code increased and decreased. Developers who used design patterns had longer code overall.

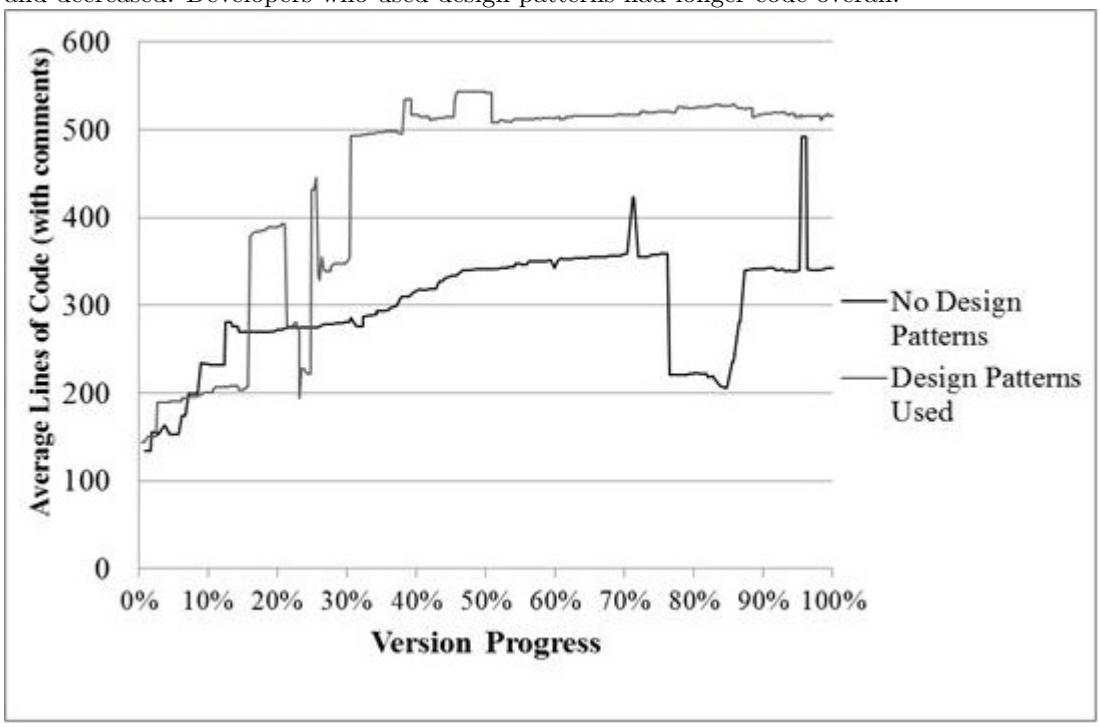
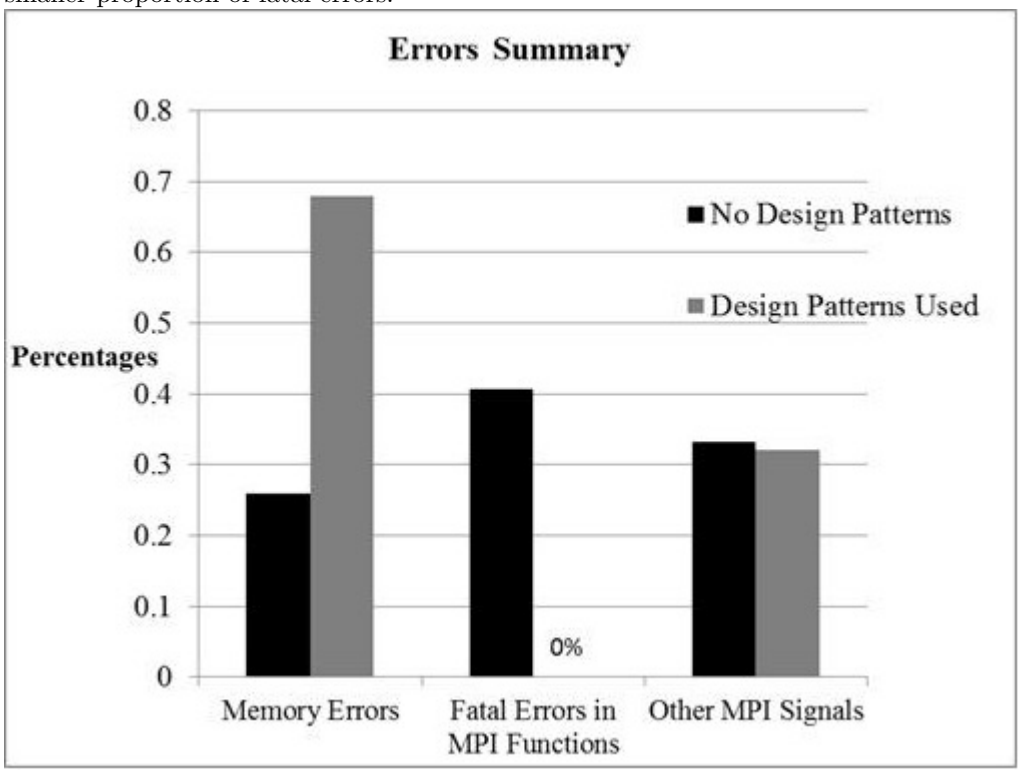


Figure 5: Design patterns caused a greater proportion of memory errors and smaller proportion of fatal errors.



if design patterns helped or not. A diverse group of programmers with varying degrees of knowledge of C would be a good idea too. While our research does show that design patterns help in the development and complexity of code, a more detailed experiment would be beneficial to further prove these results.

References

[1] Charles Blilie. Patterns in scientific software: An introduction. *Computing in Science and Engineering*, 4(3):48–53, 2002.

[2] Jeffrey Carver. Report: The second international workshop on software engineering for cse. *Computing in Science and Engineering*, 11(6):14–19, 2009.

[3] Diane Kelly and Rebecca Sanders. Assessing the quality of scientific software. *Proc of the First International Workshop on Software Engineering for Computational Science and Engineering*, pages 1–4, 2008.