ECE 3311 Software engineering 1, paper review, Northeastern Univ. Boston, Massachusetts

Nasser M. Abbasi

11/14/1993 Compiled on November 16, 2018 at 11:28am [public]

Contents

Review Of Paper
”Mainaining Object-Oriented Software”
by Norman Wilde, Paul Matthews and Ross Huitt
published in IEEE software, January 1993, pp75-80

1 Review

This paper discusses maintenance problems encountered in OO software.

The paper starts by explaining what maintenance requirements should be, which it lists the following to be such:

1) The ability to make changes easily.

2) The speed at which an in depth understanding of the software structures and behavior is acquired by the maintainers.

The paper then goes on to show how OO software behaves with respect to the above 2 aspects of maintenance, and it shows strengths, but mostly problem areas where OO software makes maintenance harder. (I found this to be surprising result of reading this paper.)

Paper now talks about how OO helps in the ease of making changes in software:

Because real world objects are mapped to the program objects more directly in OO design than in functional design, so it is easier to find what objects need to be changed in the program as a response to changes in the real world.

The paper also point out that encapsulation eases making changes, because changes in one class has no unforeseen side effects in different places of the program, changes should be localized to one class, this is one advantages of using encapsulation. The paper now makes the claim that, even though, OO design does not make programs easier to understand. The mechanisms of inheritance, polymorphism, and dynamic binding may create new difficulties for the maintainers. The paper now goes on to elaborate on this more.

The paper claims that to make changes in an OO based program , the maintainers must grasp the original designers strategy that they had in mind. (I find this surprising to say, since this problem also exist in functional oriented programs, in addition I think the authors should mention here that maintainers should first resort to specifications and design documents to

help them understand the OO program rather than look the code in the hope of so doing. The authors did not mention this in all of their discussion any where in the paper.)

Now the paper mentions that there are tools to help maintainers to understand OO system, one such tool in the use of dependency analysis to show the diverse dependencies between different objects and classes.

Now the paper enters the main subject, the authors have based their analysis on interviews with developers of two medium sized OO software projects, the paper sets about trying to find what special characteristics of the OO design the maintainers need to know to be able to modify these systems as they enter the maintenance phase.

To do so, the authors interviewed the developers themselves , they asked them what aspects of the software and design they think the maintainers need to be most aware of and pay special care to. so as to best be able to maintain the OO programs . (Again, in the following, the authors did not mention as one such source to be the design and specification documents of the project, I find this is to be a weakness from the side of the authors to neglect this source.)

Some of the things that were identified were: 1)Need of maintainers to fully understand the several kinds of class inheritance and be aware of all such class interactions. 2)Maintainers need to identify parallel classes containing similar information that need be modified together at the same time. 3) The paper now says that while it might be easy to read one class related code and to understand that, it was not as easy to understand the complex relations between many classes in large OO software system.

Now the paper discusses aspects relating to analysis of OO code and what problems that might produce. (Again, the authors seem to focus on the code as the only source for the maintainers to help them understand the OO software system, no mention here of why the maintainers can not use specifications and design documents to help them understand the OO system rater than by trying to get this understanding by analyzing the code itself.)

The authors say that a maintainer will first look at the code itself to try to understand what it does, maintainers have the need to fully understand how some piece of code interact with the rest of the system, who calls it and what routines it in turn calls, the authors claim that inheritance complicates such a task by introducing relation dependencies that the maintainer have to also understand . This is because the code being looked at by the maintainer can end up to be operating in a member function of some different subclass than the class being looked at. This introduces an extra dimension of complexity to the maintainers.

The authors have looked at 2 projects as was mentioned before, they found that in the first software project which was written in C++ and was a medium sized project, that the average depth of inheritance was found to be 2-3 with a maximum of depth 6, while in the second OO software project which was prototype in Smalltalk to be 3-4 in inheritance depth.

Another problem with inheritance as far as maintenance is concerned, is that inheritance depth seem to increase with the life cycle of the project, this is due to new features being added to the software system as time passes.

Also, each time a massage is sent to an object, the maintainer may have to examine several levels in the inheritances chain to determine how it could be handled by the receiver member function.

Now the paper focuses on the number and size of member functions in OO systems, and how that is another cause of difficulties to maintainers. the paper says that another problem is that OO systems tends to have large number of methods, the first project evaluated had 1,300 methods. The chances that the larger the number of methods the more relationship dependencies the maintainers had to understand. To understand all the methods effects, a maintainer may have to trace the chains of methods. This problem of functions interrelations, the paper says is also present in functional oriented software system, but the authors claim it is more sever in OO system, since OO design tends to have smaller method sizes (i.e. smaller code in member functions), and hence this has the side effects of having more methods communicating with each other to accomplish the task, leading to larger chain of methods that need to be traced, and hence maintenance is more difficult.

The authors have so far showed 2 areas of difficulties in maintenance of OO system. the first related to inheritance and how that increased classes inter-dependencies, and the other is the

longer chain of methods to traced. the paper points out that in both the above areas there are tools that helps, there are tools for example that analyzes dependency relations between classes.

Now, the paper looks at a different aspects of OO systems , and how that affects maintenance, they look at dynamic binding and polymorphism as potential problem areas for maintainers.

The reason why dynamic binding can cause maintenance problems is that the maintainer might not be able by looking at the code itself to know which one of any member functions will be called at run time. Since the decision of which function is called is made at run time. The authors suggest to help maintainers in this area tools that analyze data flows.

Relating to aspect of OO systems, the authors point out that some OO based libraries sometimes contain as much as 10 methods to implement one given message, and knowing which one will be called at run time can be very difficult, and so it ends up making debugging harder. The paper says that to help reduce this problem, consistent method naming must be adhered to.

The paper claim that dynamic binding and polymorphism are like a two-edge sword for the maintainers, at one point they give flexibility that is the main objective of OO programing, yet on the other hand they make programs harder to understand and hence harder to maintain.

Finally the authors talk about cooperating object classes , they claim that while cooperating object classes is not a bad design idea, they make OO programs harder to understand, the paper suggests the use of more dynamic methods to try to understand the interactions between classes instead of static methods, one method the paper mentions in called object animation, which a dynamic class interactions analyzer.

2 Conclusion of the paper

Designers should keep in mind the maintenance of the software while working on the design, they should limit dynamic binding and polymorphism unless these feature are essential, also designer should recognize the difficulties maintainers have in understanding cooperating classes interactions and so try to reduce this.

The authors finally say that OO methodology is still new, and that we must be prepared to monitor likely problem areas that arises with its use.

3 Critique

I felt this was a very useful paper to read and review, it helped me better understand a very important aspect of OO software system, which is the maintenance problems unique to OO software.

I felt the authors have presented out a good number of issues and problems in OO maintenance in simple and straight forward manner. They looked at the major features of OO software (inheritance, polymorphism, dynamic binding, long chains of methods), and they explained why and how these features make maintenance of OO software harder or more time consuming than non OO based systems.

However it also seems to me that their conclusions and analysis might be too hasty, since the authors themselves say that they based these analysis from looking at only 2 software projects, one in C++ and one in Smalltalk, also the size of these project was medium size (they did not say what the actual size was), it seems to me that they should have looked at more OO based software, and interviewed maintainers who have worked on these projects for longer times.

Still I think the arguments presented in this paper seemed clear and this paper I believe should be read by any one who is working on OO software systems. The authors also give 7 useful reference at the end relating to maintains of OO systems.

This concludes my review and general critique of this paper.