Practical Complexity Analysis
. . . and Reduction

Wolfgang Spahn

2022-11-14

Figure 1: Photo by Ola Dapo (Pexels)

Over time, as part of any development cycle, projects accumulate unnecessary complexity that should be reduced by people's efforts, but it is not. Under pressure, these activities suffer, so the depth of over-complexity and under-documentation is accumulated, which becomes more and more difficult to recover.

When activity grows bigger, it can be increasingly difficult to know where and how to apply changes. Decisions get costly and risky. For sure some complexity growth is unavoidable as more features are offered to the market. But over time knowledge is lost when people are leaving the team, implicitly competing concepts have been implemented or abstraction gets leaky. Over-complexity and lack of knowledge on status quo is accumulating. It generates a general loss of coherence.

Without counter measure, the sand in the gearbox spreads, creates frustration of all involved, it hinders objective reasoning on recovery means. Reasoning starts to lack evidence, it can be applied only partly. In general arguments get more and more opinionated.

As a natural tendency, when degradation has pilled up, there is the call to redo the whole application in the hope that now the evolution will be different. For long living mission critical apps this is very painful as robustness needs to regained from day one. There is quite a risk, when having reached the same depth of feature and robustness, amount of non-coherence is similar. Just it is now in different areas. In both scenarios, re-emplementation or not, learning to tackle the complexity challenge is a necessity to get a different outcome.

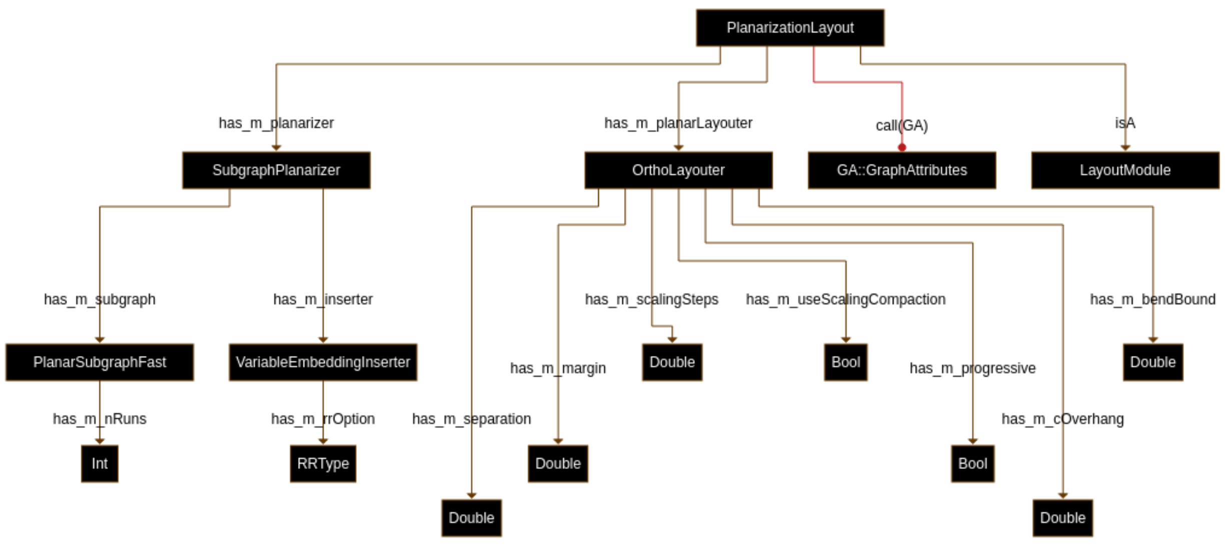

Figure 2: *Automatically generated software knowledge graph to regain the big picture*

Over-complexity in the software is often paired with a lack of big picture visibility. It is a good idea, to invest some capacity and take a step back and to create a verified, complete big picture of the software to be omnipresent in the team: What is the meaning and purpose of the individual elements, what is its structure and architecture? What concepts are localized, what are delocalized? How do all of the pieces fit together? With this improved overview detailed analysis can follow, team can propose and agree on changes and finally recover the situation.

To create a verified big picture and control the details with it, we can learn from best practices in SW documentation, Domain Driven Design, and verified machine-readable knowledge bases (see fig. 2).

1 Recover specifications

Typically, in situations of lost knowledge, the visibility of software specifications is also reduced. Descriptions are missing, ambiguous or imprecise, resulting in extra development effort.

Especially in complex, mission-critical applications, the specification of distributed behavior is very important in the daily work of developers. However, documentation is often lacking, forcing developers to speculate about behavior. Even when supported by code comprehension analysis and traces on example cases, it is to be avoided as it is very time consuming and error prone.

Practical formal requirements tools used by the Aerospace, Avionics, Healthcare and Fin TEC industries can help restore a common understanding of the system specification and its mapping into code.

To understand your status quo, an analysis of verification and field failures is recommended. Whenever they can only be recovered by reengineering the code behavior for better understanding, you have a good indicator that improving specification coverage should be considered.

2 Finding the essence of the application

Applying recent methods from the theory of software design, which understands a software system as a collection of interacting concepts, can restore the understanding of the system at hand.

It brings the techniques of natural language understanding used by conversational applications like Siri into the realm of code and system understanding.

3 Use Static Analysis to scale

When the system is already years old and the code lines are large, some automatic methods must be used to get practical answers with affordable effort.

Using machine based static analysis allows to automatically analyze a large amount of code and specifications. It offers interesting options to drastically reduce the effort to heal the accumulated degradation, guiding team to implement pratical complexity reduction.

4 Conclusion

Over time projects collect unnecessary complexity, which should be reduced by peoples effort, as part of every development cycle, but they are not. Under pressure these activity suffer, so depth on over-complexity and under-documentation is collected which is more and more difficult to recover.

By the advancements of artificial intelligence conversation systems language understanding is now wide schema of usage. This technologies can be used as well for specification and code/systems understanding.

The use of machine-based static analysis allows large amounts of code and specifications to be analyzed automatically. It offers interesting options to drastically reduce the effort to heal the accumulated degradation, guiding the team to implement practical complexity reduction.