On Analysis

Robert C. Martin

27 February, 2002

Abstract

The word "analysis" has taken on many different meanings in the software industry. This paper describes the history and etymology of the term as it evolved through the last half of the twentieth century. The paper goes on to describe the definition of analysis in agile software development; i.e. the identification, enumeration, estimation, and prioritization of requirements.

Every software developer is familiar with the term "analysis". Many of us take its meaning for granted. However, the term "analysis" has had many different meanings in the software industry over the course of the last four decades. Here are just a few.

Systems Analysis in the Sixties

In the sixties, the term described the activity of designing a suite of interacting programs and data structures to address a particular customer need. In those days a systems analyst would specify the data sets to be manipulated, describing both their internal data structure and the media, such as tape or disk, that would hold them. They would also describe the way those data sets were manipulated. Some manipulations were accomplished by using standard programs like sorts or merges. Other manipulations required custom programs that the analyst would specify.

There was not a strong temporal constraint between the activities of analysis, design and implementation in those days. Once one of the custom programs had been specified, it could be designed and implemented, even if other parts of the analysis were incomplete. Indeed, in those days there was no discrete design step. The analysis spec was given to the programmer, and the programmer took whatever steps he or she thought necessary to implement it.

Structured Analysis and Design in the Seventies

In the seventies and eighties *Structured Analysis* (SA) was described by Yourdon, Constantine, and DeMarco. These practices became very popular, and by the early eighties had a profound effect upon the definition of analysis. SA was a technique in which the requirements of the customer were broken down into a hierarchy of functions. This breakdown was known as *functional decomposition*. Datasets were specified in abstract (Bacchus-Naur) terms, and their manipulation were depicted through functionally decomposed data flow diagrams.

SA was coupled to another practice known as *Structured Design* (SD). Indeed, the two were often mentioned in the same breath as SASD. SA described the data sets and data transformations implied by the requirements. As such, SA described *what* the system would do, albeit in very technical terms. On the other hand SD described the partitioning of the software into modules, and the flow of data between those modules. Therefore an SD was, more or less, a description of *how* a system would be structured to meet the requirements.

SASD contained a practice known as *The Transform Analysis*, which was used to convert the diagrams representing a Structured Analysis into the diagrams the represented a Structured Design. This practice, and indeed much of the documentation of the period, established the notion that the design was directly derivable from the analysis by applying some simple transformation rules. This meant that the analysis was really a preliminary description of the design, requiring only a mapping operation to complete.

The view of SASD was a remarkable change from the systems analysis of the sixties. In the sixties no such mapping from analysis to design was implied. The analysis simply described data sets and their transformations without implying anything about software structure.

SASD also implied a temporal constraint between analysis and design that was not practiced by the Systems Analysis of the sixties. In SASD it was necessary to finish the analysis before the Transform Analysis could be applied to translate the analysis into a design. Thus, SASD strongly reinforced the waterfall model of development.

Object Oriented Analysis in the Nineties.

Though object-orientation (OO) was invented in the sixties, it languished for two decades before rising to popularity in the nineties. At first, there was just object-oriented programming (OOP). Soon, however, the OO prefix was applied to both analysis (OOA) and design (OOD).

The goal for OOA was to eliminate the Transform Analysis, or its equivalent. Instead of having to translate the finished analysis into a design, an OOA could be *elaborated* into an OOD through the addition of extra detail. This meant that the analysis and design could be performed in the same notation, and at the same time. Indeed, there was a movement in the early nineties to break the temporal constraint between analysis and design and make them *concurrent* activities.

However, the notion of the mapping between analysis and design was maintained, if not strengthened. Now, instead of transforming an analysis into a design, one could simply add detail until the analysis became a design. In a way, the analysis *was* the design in rough form.

Ironically, the relaxing of the temporal constraint did not take hold in the nineties. Too many companies and development teams had already established their development processes around the waterfall. The notion that analysis and design could be done concurrently was foreign at best, and blasphemy at worst. Thus, OOA and OOD remained completely separate activities, with completely separate artifacts that just

happened to use the same notation. Indeed, many teams expended significant effort in separately maintaining both the analysis model and the design model.

Thus, over the last forty years, analysis has changed in two fundamental ways. First, it has become tightly coupled to program structure, to the extent that it is now possible to generate code from analysis diagrams. Secondly, analysis has acquired and maintained a temporal priority over design. In most companies and development teams the OOA must be completed before the OOD is begun.

2001: Agile Analysis

Recently there has been a reversal of opinion about analysis. This reversal is perhaps best typified by the values of the Agile Alliance formed in February of 2001 (agilealliance.org). The agile way of thinking breaks both the temporal constraint, and the structural coupling of analysis. In an agile project the analysis is not done all up front, nor is it directly map-able to the internal structure of the software.

In an agile project, analysis is performed through collaboration between the business stakeholders (possibly represented by business analysts), and the developers. Together they identify features that the system will be required to perform. Features are broken down into small tasks that can be completed in a month or less. The developers estimate these tasks based upon past history in order to establish their cost. Then, the stakeholders decide the priority of the tasks by weighing their cost against their importance to the business. This priority then sets the order in which the tasks will be developed.

Clearly, in order to establish tasks that are less than a month in length, and then estimate those tasks, a considerable amount of design must be performed. This includes at least a rough understanding of the datasets, data transformations, and user interfaces that the application must deal with. Thus, completion of analysis depends upon design. At the same time a significant amount of business analysis must be performed to understand what the features and tasks must be before they can be estimated. Thus, design depends upon analysis. This mutual dependence of analysis and design completely breaks the temporal constraint. Neither can be done without the other.

On the other hand, the design required to break the system into small tasks, and estimate those tasks does not necessarily imply anything about the internal structure of the software. It need not identify classes, relationships, and methods. Rather those tasks, and their estimates, describe external behaviors that are visible and demonstrable to the stakeholders.

The temporal constraint is further weakened because the system can be designed, analyzed, and implemented concurrently. It is not necessary for all the features to be estimated and broken down into tasks before implementation begins. Indeed, the stakeholders could choose a few of the most important features. The team could break them into tasks, estimate them, prioritize them, and choose a month's worth of the highest priority tasks to implement. Indeed, in an agile project they repeat this activity month after month.

Conclusion

In some ways agile analysis is more akin to the systems analysis of the sixties than the OOA of the nineties. They both share the absence of the waterfall time constraint. They both concern themselves more with business value than with program structure. And they both allow implementation to act as feedback to the analysis process.

However agile analysis is not just a throwback to the sixties. There are some important differences between systems analysis and agile analysis. Agile analysis uses the concept of small iterations, and smaller tasks. It uses allows the stakeholders to decide priority based upon business value and estimated cost. It puts estimation in the developers' hands, and bases it upon reasonable design work. It considers the stakeholders and the developers to be a collaborating team.