SPECIFICATION FIRMS: A VISION FOR THE FUTURE

Jeannette M. Wing

Computer Science Department
University of Southern California
Los Angeles, California 90089-0782 USA

ABSTRACT

The analogies that specifiers are like lawyers and that specifiers are like architects are often drawn. Pursuing these analogies further leads to the conclusion that in the future we may expect to see *specification firms* for which specifiers work and are hired out independently much like lawyers are in law firms and architects are in architectural firms. In this position paper, I justify these analogies, and explore their ramifications to the software engineering process.

1. Introduction

We presume that formal specifications can be and should be practically used in software development. Furthermore, we are interested in specifications of large systems, not just of program modules. For the sake of sticking to one point in this paper we take a simplistic view of the software development process by considering the players involved to be *clients*, *designers*, *specifiers*, and *programmers* and the phases to be *requirements analysis*, *design and specification*, and *implementation*. In practice, of course, other players are involved, players have multiple roles, the entire process is iterative, each phase is iterative, and each phase may feedback to previous phases. Our reasoning and conclusion holds for any of these realistic complications of our simplistic view.

Author's present address: Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213, USA. This work was supported in part by the National Science Foundation under Grant No. ECS-8403905.

2. The Two Analogies

2.1. Basis of the Two Analogies

Specifiers as lawyers

The analogy that specifiers are like lawyers is based on the use of a specification as a contract. The specifier draws up an agreement between the client and the programmers. Specifiers translate the client's informally stated requirements into a formal specification and programmers implement the system according to the specification. Both parties are bound to the specification; the clients can expect no more than what is agreed upon and the programmers must satisfy at least what is agreed upon. As with lawyers and their clients, a good amount of trust must hold between clients and specifiers and between programmers and specifiers.

Specifiers as architects

The analogy that specifiers are like architects is based on the use of a specification as a design. The specifier produces a blueprint as the design of the eventual system to be built. The specification serves as a description of an abstract model of the real system.

In architecture, preliminary sketches approved by a client are further refined into detailed blueprints. Similarly, in an ideal specification process, a preliminary specification would gradually be refined into a detailed design of the system. This detailed design specification would include specifications of the individual system modules that constitute the system. After an initial specification of a system is refined to a level where each system module is specified, the programmers can regard the final specification as a first design of the entire system, and proceed to implement the individual module specifications. Programmers have the advantage over builders who work from blueprints in that after a first implementation, further modifications can be made to improve the system. (Builders do not have the luxury of tearing down their initial building and redoing it.)

CH2138-6/85/0000/0241$01.00

2.2. Justification

Below are four reasons why the two analogies make sense for software engineering. The first two reasons deal with the nature and use of specifications; the second two, with the role of specifiers.

1. A Specification is Written in a Specialized Language.

In both analogies the contract and the blueprint are each written in a highly-specialized language that may be only partially understood by all parties concerned. For example, typically both parties involved in a legal contract do not fully understand legal terms or legal implications of the contract. A formal specification is also written in a highly-specialized language (mathematics) that cannot always be expected to be understood fully by two of its possibly many communities of readers: clients and programmers. Much in the way a lawyer interprets legal jargon for both parties of a contract, specifiers must interpret the mathematics for clients on the one hand, and programmers on the other. Fortunately, with proper training programmers can be taught to read and even write formal specifications since they are already fluent in at least one formal language--their programming language. Much like with a blueprint whose details are presumed to be understood by a builder, we can hope that specifications will reach the stage where programmers can read and understand them.

Thus, most of the work of interpreting a specification by a specifier would have to be aimed at the clients who may not have the mathematical background to understand a formal specification, the time and resources to gain such a background, or the desire to do so. Specifiers must convey the formal meaning of a specification in a natural language description to the clients who can then decide whether the specification captures the intuition properly. Conversely, clients who informally state their system's requirements must make the effort to convey their intuition to the specifier as accurately and completely as possible.

2. A Specification is a Common Point of Reference.

A physical document like a specification is a point of reference common to programmers, clients, and specifiers. Programmers can refer to it to answer questions that arise while doing an implementation. Clients, through the interpretation of specifiers, can refer to it to recall what requirements were requested, and are obligated to have the specifier modify it if any changes or additional requirements arise. The document is useful especially if the client and the programmers are in geographically separate locations so that it would be impractical for programmers to contact the client and discuss issues verbally.

3. A Specifier is an Intermediary.

A specifier serves as an intermediary between the client and the programmer. It should not be left up to only the programmer to discuss the client's desired requirements, to interpret a formal specification for the client, or to demonstrate that the system satisfies the requirements. A specifier should play an important role in each of those functions. One reason is that often programmers may not be able to abstract from implementation details to be able to pose questions of the client at a level to which the client could relate. The specifier serves as a go-between by performing the proper abstraction and interpreter of natural language, the specification language, and even possibly the programming language.

4. A Specifier is a Source of Knowledge.

A specifier is a source of knowledge for (1) evaluating the feasibility of a desired system and (2) investigating whether similar systems to the one requested have already been built. As a lawyer or an architect, the specifier can evaluate the feasibility of building a system according to the constraints imposed by the system's environment. Any contract drawn up by a lawyer, for example, is interpreted with respect to local, state, and federal regulations. Nothing could be included in a contract that would violate any of those regulations. Similarly, if a new house is being designed, structural engineering principles, the availability and cost of materials and supplies, and even local weather conditions may make it difficult or impossible to satisfy all of the client's requirements. Similarly, in the design of a software system, properties of the environment in which the system will be embedded, limitations of the hardware, choice of programming language, availability and cost of resources (hardware, tools, human), and deadlines, are all factors that may necessitate compromising some of the client's requirements.

Specifiers can also build upon previously acquired experience from designing systems similar to the current request. If the software itself cannot be reused, at least parts of the specifications can be. If existing software can be reused, bought, or traded, startup costs and development time can be saved for the client.

3. Implications for the Software Engineering Process

Arguments for the usefulness of formal specifications throughout the software life cycle have been given, e.g., see [1], [2], [3]. Their uses in design, program transformation, testing and debugging, documentation, and maintenance are widely accepted in concept, though rarely used in practice. Instead of focusing on specifications, which are the results of specifying, however, let us focus on the specifiers.

In the immediate future we can expect to see a team of formal specifiers as part of the entire group involved in the development of a large project. This team would enhance, not replace, the usual team of designers and help interface between clients and implementors. Eventually, within a large enough company or corporation, we may then see the existence of a division of formal specifiers who are brought on to assist in various in-house systems and applications-software projects. Groups of specifiers in these divisions may eventually spinoff to form their own private firms. Small companies who cannot afford their own team of specifiers or larger ones who contract out specification work can then rely upon the expertise of the members of these specification firms.

Just as with law firms and architectural firms, specification firms can be specialized. Specialties might break down along classical lines of computer science disciplines, e.g., real-time systems, operating systems, compilers and optimizers, networks, databases, graphics, distributed systems; according to different application areas, e.g., medical systems, digital hardware design, military systems; along other axes, e.g., size (large versus small systems), cost (over $100K projects), time (under 3-months projects); or combinations of these. For example, a specification firm might specialize in dealing with only large military systems or in dealing with only projects with a budget of over $100K and with a one-year delivery deadline.

In general, it should not be surprising that specifiers might be hired for various unrelated projects. They may be hired for their expertise in writing formal specifications and using associated specification tools, for their ability to state formally a set of informally described requirements, for their knowledge of what is and is not feasible, or for their knowledge of what has already been done. People already recognize the benefits of obtaining outside advice and criticism from those who are not intimately related or have a conflict of interest in the project under development.

A more global point can be made: One cannot expect all members of a software project to be talented in all phases of the project. Writing formal specifications for complex systems requires some expertise that not everyone has or is willing to cultivate. We already see software teams break along the lines of the talents of its members; specifiers simply have one other kind of talent to contribute.

4. Closing Remarks

Especially for large, complex systems, people recognize the need to be more rigorous in their design and specification phase. Formal specifications, however, are rarely used in practice. Many reasons including a lack of incentive or motivation to do so (from the engineer's and manager's points of view), a lack of training, and a lack of high-level specification languages and support tools account for the scarce use of formal specifications. Hiring trained specialists would be a feasible way of overcoming these difficulties and increasing the use of specifications in software development. As these specialists gain recognition for their expertise, specification firms may arise, thereby creating sources of specialists willing to do the work that others need done, but do not want to do themselves.

Finally, pursuing the notion of specification firms that hire out experts to write specifications leads to implications beyond the traditional issues of software engineering. If indeed a specification is treated as a contract, i.e., a legal document, then legal issues dealing with software have a tangible piece of evidence over which to base cases. A breach of contract might be legitimate grounds for a lawsuit. Perhaps software engineers would feel more responsible for the robustness of their programs than they do now. We leave up to the reader's imagination further implications of using specifications in the legal process; they are serious and somewhat scary to think about.

References

[1] J.V. Guttag, J.J. Horning, and J.M. Wing, "Some Notes on Putting Formal Specifications to Productive Use," *Science of Computer Programming*, vol. 2, no. 1, pp. 53-68, October 1982.

[2] B.H. Liskov and S.N. Zilles, "Specification Techniques for Data Abstractions," *IEEE Transactions on Software Engineering*, vol. 1, no. 1, pp. 7-19, March 1975.

[3] D.L. Parnas, "The Use of Precise Specifications in the Development of Software," Information Processing 77, B. Gilchrist, Editor, IFIP, North-Holland Publishing Company, pp. 861-867, 1977.