Rational Design and Documentation

Alireza Namvar and Jeremy Bowers

April 12, 2002

CSE870 Advanced Software Engineering MiniProject
Computer Science and Engineering
Instructor: Dr. B. Cheng
Michigan State University
Contact Information: chengb@cse.msu.edu

Abstract

This paper examine Parnas and Clements paper A Rational Design Process: How and Why to Fake It [1], looks at three papers that build on the work of Parnas and Clements [2, 3, 4], and an additional paper that follows up on the work of Parnas and Clements, but does not cite [1] as a reference [5].

1 Overview of A Rational Design Process: How and Why to Fake It

In "A Rational Design Process: How and Why To Fake It", Parnas and Clements contrast the theoretically ideal software development process with real, practical software design and documentation. Parnas and Clements first discuss what software development would look like if performed by a rational" entity, which always has a reason for every action and does not proceed with an action without such a reason. By definition, it proceeds in a rational and systematic fashion, and the decisions it makes, along with the corresponding reasons, can be documented.

The hypothetical rational design process converts accurate requirements document into an executable program. It starts by writing a development process document, the creates the requirements document, determines and documents the modular decomposition of the project, and keeps the documentation up-to-date during implementation and maintenance.

The process description document documents the project’s definition, satisfaction criteria, and required skills and knowledge. The complete reference of desired behaviors in the requirements document avoids any duplication or inconsistency in the implementation. It also assists in estimating the project cost, insulates the project from staff turnover, and settles arguments among programmers. The semi-precise mathematical model in the requirements document addresses the separation of concerns. The documentation of the modular decomposition is useful during software maintenance. It contains the formal specification of module interfaces, list of invokable programs, access parameters, internal structure, constraints and exceptions in modules.

This hypothetical rational design process follows systematic steps for developing, never backtracks during development because all decisions are made rationally and correctly the first time, and provides a standard way of communicating with the extensive documentation produced, which also facilitates project management and review by outsiders.
Parnas and Clements then observe that following this ideal process is impossible in reality for a rather long list of reasons, including the impossibility of complete domain knowledge, the impossibility of managing all the detail of a large project perfectly correctly, the influence of time and changing requirements, preconceived notions, economic necessities, and other things that our original idealization of a software engineer does not need to worry about. However, it is worthwhile to "fake it": By imagining ourselves in the position of this ideal entity, we can then write the ideal documentation that the entity would have produced. This includes documenting design decisions, mathematical models, constraints, and all of the other things we would like to see in ideal documentation. The result is that in the end we will end up with this ideal documentation, clearly organized and useful, rather then the document that might be produced more naturally, which will be excessively bound to the actual order of events, may contain dead ends, and may have many design decisions left implicit, rather than explicitly stated.

The paper closes with the observation that faking a rational process is used all the time in mathematics, where we are also more interested in results, and effective communication of concepts, than the methods used to arrive at those concepts. Compare the original incarnation of Maxwell's Electromagnetism theory, with hundreds of equations, with the four equations college sophomores learn today, or the one that physics majors learn. Or compare the original proof of Godel's incompleteness theorem with what is taught in computing theory now. We value effectiveness over strict historical accuracy.

This paper fits into the vein of papers such as Fred Brook's famous "No Silver Bullet" essay [6], published at nearly the same time. The importance of papers like Parnas and Clements is it helped the Software Engineering discipline shed some naivety about the difficulty of software engineering, and realize that there will be no simple, magic answer to the problems of software engineering. Brook's attacked the problem of software production. Parnas and Clements attack the problem of useful documentation. Both have the same root message: No ideal solution to these problems exists.

By recognizing that documentation is a hard problem, it opens the door for researching better and easier ways of writing good documentation, rather then keeping a culture with the poisonous subconscious attitude that Real Programmers should find documentation easy to do, where the idea can not be examined and refuted. This opens the door to things like Literate Programming, as described by Knuth [7], using user's manuals as requirement specifications (see our paper later), and other practical documentation techniques research.

2 A Development Method for Multiagent Systems

"A Development Method for Multiagent Systems" defines a system the author calls MASSIVE, for "multiagent systems iterative view engineering", which is proposed as a model for the creation of multiagent systems. MASSIVE is based on the idea of creating and using seven "views" on a system, where a "view" is a projection of the root model of the system with only a certain type of information in it. (The author uses the term "projection" analogously with geometric projections of multi-dimensional objects into lower dimensions, where there are many valid projections of the same object, each highlighting different data and suppressing others about the same object.)

The part of the paper that interests us in this assignment is section three, "Iterative View Engineering", where the author describes how to create the views and models necessary for the multiagent development method. The author combines Roundtrip Engineering [8]and Iterative Enhancement [9] by using Iterative Enhancement as a repeated subprocess to extend Roundtrip Engineering.

Briefly, Roundtrip Engineering consists of creating a model for a problem, implementing the model, fixing the implementation as much as possible, then analyzing the implementation to extract a new model, based on what was learned during implementation. The author of this paper enhances this model by adding iterative enhancement steps to the creation of the model, the implementation, and possibly adding trips around the whole process; We do not know if the original Roundtrip Engineering paper specified more than one trip
through the process, as it appears to be in German.

This extends Parnas's work not by adding to the corpus of work on abstract documentation processes, as the previous two papers we discussed did, but by actually going out and using the observations Parnas and Clements made about rational design processes. The author of this paper explicitly cites Parnas and Clements as an influence: "The advantage of this approach is that it takes into account that any specification is initially incomplete and usually undergoes frequent changes...[1]" We infer from this that [8] did not take this into account.

This demonstrates that Parnas and Clements were not just making useless observations in their paper, but provided useful ideas for the difficult problem of developing documentation, and by extension, developing software.

3 Software Documents: Concepts and Tools

"Software Documents: Concepts and Tools" is written on the premise that all results of the software engineering process can be considered as documents. This includes even source code, which can be considered as the document that finally specifies a program carefully enough to be executed by a computer. The paper considers what an ideal development environment would be for these documents, primarily by focusing on the features we want, and the properties we would want these tools to help us maintain.

One focus of the paper is the issue of maintaining consistency across versions. Document development tools should assist the users in maintaining consistency as various documents are produced, starting with the loosest, informal specifications and moving toward fully formal specifications, ending up with actual source code. The end result resembles an even stronger version of literate programming, where one can view not just the documentation for a given piece of code, but the whole development process of the code fragment.

Another issue is verification of code. By using tools to tie together the original specification, the formal specification, and the actual code, one can improve the ease of formally validating code, which also increases the likelihood that one will actually use formal methods.

The paper calls for the creation of tools to enable this level of connectivity between the various software documents, and also suggests other desirable features. For instance, one suggestion is to use the tools to do automated consistency checks, which might propagate a variable name change across all of the source code. One might even analyze certain changes, and allow an administrator to cause the system to reject them.

This paper describes another concrete way of achieving the goals Parnas and Clements lay out. By improving our tools, we improve our ability to create coherent documents that contain all of the information we may be interested in, from the beginning of the software engineering process through the completion. If the system described in the "Software Documents" paper existed, it would be much easier to produce "Rational Documentation." Unfortunately, the system does not yet exist, and there are significant engineering issues still to be worked out.

4 User’s Manual as a Requirements Specification

"User’s Manual as a Requirements Specification" makes the argument that the the qualities required in a good user's manual are the same qualities that are required in a good requirements specification. A user’s manual concentrates on completely describing the features of the system, how the user will use them, how the user should conceptualize the system, and not mentioning implementation. All these are desirable qualities in a requirements specification. Thus paper proposes that it makes sense to intentionally use a user’s manual as a requirements specification, in addition to its duties as a user’s manual.

The authors also suggest that users manuals provide an excellent source of test cases for the program, as each scenario in the user’s manual can be used as a test case. (There may exist an even deeper level of
granularity; each assertion made in a user’s manual, such "Click the ‘send’ button to submit your report to headquarters", can be used as a separate test case.) This does not completely alleviate the difficulty of maximizing test-case coverage, but it can be a good foundation.

Users manuals also provide a powerful tool for communicating with the client by providing a framework for elicitation, analysis, and validation of the requirements specifications. It provides a focus both the developer and the user can understand, and is a good way to get at least the foundation on which other documents on which other documents will be built down on paper.

The user’s manual also provides concrete uses of the system. If functionality is missing or incorrect, the client is likely to be able to tell, by noticing how that functionality is or is not reflected in the manual. It also provides a way of detecting and eliminating ambiguities in the requirement, because the user's manual is a concrete design for the system, therefore, the act of creating document requires a concrete understanding of the system. This allows the clients to clearly discern where the developers may have resolved an ambiguity in a way other then what the client would have wanted.

Parnas and Clements’ "Rational Design Process" establishes the importance of good documentation, with the qualities they outline in their paper, but the paper concentrates on the end results of a "Rational Design Process", and does not discuss in detail how to achieve the quality documentation they call for. "User’s Manual as a Requirements Specification" builds on Parnas and Clements’ work by exploring a specific and concrete way of achieving quality documentation.

In the process of writing the user’s manual, many conflicts will arise that must be resolved, such as various ways of implementing a feature, or later additions to or subtractions from the user’s required feature set. As these conflicts arise and are resolved, the resolution of these conflicts will be reflected into the user’s manual, which is an example of faking a rational process; we choose to obscure the conflicts in the user manual and just concentrate on the final process, though we may record the conflicts elsewhere.

By providing this method of creating good documentation, this paper also answers the argument that trying to follow the suggestions in "Rational Design Process" might not be practical. Using the user’s manual as a requirements specification is a very practical solution to writing "rational documentation" in the early phases of a project, when there is little code or design to document. Since one must ultimately write a user’s manual for any significant project, it need not necessarily inject much extra work into a project to have the user’s manual do double duty.

The authors of this paper do disagree with Parnas in one interesting way: Parnas says in section V.A.3 that users or their representatives should write the requirements specification. In this paper, the authors have the developers creating the users manual, which is the requirement specifications. This inverts the responsibility for clarification of ambiguities from the way Parnas described, as having the developers write the requirements requires the developers to be extra careful to stay in sync with the user’s perceptions, rather then vice versa.

5 A Pragmatic Approach to Software Documentation

"A Pragmatic Approach to Software Documentation” describes the author’s work on an extension of Literate Programming. Traditional Literate Programming works on a file-by-file basis; the author’s “DosfOp” system extends literate programming to larger systems, and works to address some of the design decisions of the original WEB system (and subsequent derivatives) that have turned out to be suboptimal as time by goes by, such as the difficulty of learning how to deal with what is basically a language on top of a language, and the reduced importance of a meta-language to support code-reordering in the OOP and functional paradigms.

DosfOp starts with the source code of the project, a project database containing additional information about the project and how the various modules are related, and some global configuration data, and prepares documentation in multiple formats. This architecture also allows for using old code without changing the old code, as documentation can be included in the project database, in addition to the traditional inclusion
in the original source code. The project database also includes documentation that may not be attached to a specific file, such as surveys or overviews of the project.

This paper should have cited Parnas and Clements’ work in the introduction of the paper, particularly in the following paragraph from:

In particular, it must be possible to integrate source code that was not prepared specifically for the documentation system. So quick and dirty... can be integrated and then later on be gradually documented. This is certainly not a very pure approach...

This is a specific way of creation documentation that matches what one would get following a Rational Process, as outlined and qualified in the Parnas paper. As such, it should have referenced the Parnas paper.

References


