Contributions of Object Oriented Software Design towards Limiting the Problems Caused by a Lack of Software Engineering.

Jennifer Bevan

May 14, 2002

Abstract

The Radio Science Validation and Processing (RSVP) software suite was created to replace an outdated and disorganized set of legacy software. The development process applied no formal software engineering methods, but did employ an object-oriented approach during the design of the initial phase. The modules developed during this initial phase were later recognized to be the most stable and easily maintained portion of the resulting software suite. This paper discusses some of the problems frequently caused by a lack of software engineering, and examines how an object-oriented foundation mitigated the effects of these problems on the overall quality of the RSVP project.

Contents

List of Figures

List of Tables

1 Introduction

While the benefits of software engineering are no longer considered to be strictly hypothetical quality improvements [1], the consistant use of software engineering principles is still far from widespread. While software engineering baccalaureate programs worldwide have increased from none in 1994 to more than 18 in 1999 [2], object-oriented languages have generally become the rule rather than the exception in the required programming courses for computer science baccalaureate degrees. This education is creating a growing pool of programmers in the workforce who are already familiar with the basics of object-oriented design, yet who are not familiar with software engineering principles. Given the obsession with lowering a product's time-to-market (TTM), if it seems possible to create that product using familiar techniques, commercial developers will use what they know. New learning generally occurs only when a sufficient benefit is clearly shown to and approved by project management.

However, the use of object-oriented principles in software design is certainly not mutually exclusive with the use of software engineering principles, and in fact helps to ensure that some basic software engineering issues are addressed. This report identifies common problems incurred by the lack of software engineering, discusses the aspects of object-oriented design which overlap software engineering concerns, and examines how an object-oriented foundation mitigated the effects of these problems on the overall quality of a specific scientific data processing tool.

2 Development Overview

The development of the Radio Science Validation and Processing (RSVP) software suite was divided into three phases, the first of which allowed the extraction of science data from multiple input formats. The legacy software which mathematically manipulated the data was updated later, during the second and third phases; the requirement that the first phase produce output that was acceptable input to the legacy modules restricted the early objectification of the project as a whole. While later versions of RSVP started to incorporate object-oriented principles in these later modules, only the "core code" developed in the first phase can currently be considered object oriented.

In 1994, there were 6 input formats that the software was required to accept. By 1999, this number had grown to 10. Within these formats, data of the same *data type* (such as received frequency, time-stamp, or other format elements) were not always represented in identical units or encodings. The RSVP core code uses a generic *Record* class which holds all of the common data across the different formats, as well as data search functions. Format-specific classes inherit from this *Record* class and add the necessary data types and flags to correctly control the processing control and output. Virtual functions in the *Record* class lead to format-specific subroutines in the appropriate class, subroutines which generally consist of bit-level unpacking of data formats. The core code also consists of an *ioType* class which provides a base class to multiple media-specific classes; The half-inch magnetic tape media used by the legacy code was discontinued the same year RSVP was commissioned, and those data files were archived onto Exabytes and later, onto CD/ROMs. While the *ioType* class was being created to handle both types of high-capacity magnetic media, additional

type-specific classes were added to handle byte streams and arrays. After the core code was initially released, additional format- or media- specific classes only entailed creating a new class and updating the case/switch statements which took the user's type specifications and created the appropriate instantiations.

3 Common Problems Involved with Informal Product Development

The purpose of software engineering is essentially to provide methods by which the probability of creating the right product, in the best possible way, is maximized. Checkpoints are inserted into well-defined developmental stages which attempt to discover bugs and other "features" as early as possible, thereby reducing the number of problems found after product deployment. Software engineering methods encourage the development of software that is easily upgraded or otherwise maintained. In other words, the goal is to produce software with as much thoughtful planning and execution as is commonly applied in other engineering disciplines [3].

Some common problems which frequently result from a lack of such formal development techniques include an incomplete or incorrect understanding of the product requirements and designs that, even if formally established, are not flexible in the direction of the most likely upgrade path. Additionally, implementations are created that require system-wide changes for maintenance or upgrade modifications; these changes are frequently of the type that cause unintended side effects in unexpected parts of the code. While software engineering does not guarantee an error-free product development cycle, it certainly does provide a framework which aids developers in avoiding these problems.

4 Software Engineering Issues Addressed by Object-Oriented Design

Object-oriented design, while not a process that can be applied to a product's entire life-cycle, certainly addresses some of the same concerns as software engineering. The re-classification of a problem domain into objects enforces a fairly in-depth examination of the requirements of a project. The object-oriented design state creates a well-defined model for functional and object-based upgrade paths. Object-oriented implementation is more tightly connected with a verified design, and the software engineering metrics of high cohesion and low coupling are analogous to "good" object-oriented design techniques: these techniques promote the localization of the scope of bug fixes.

5 Object-Oriented Design Principles and RSVP

The object-oriented approach in the development of RSVP addressed several software engineering aspects. Each of these aspects, and the extent to which this approach acheived the

goals of the aspects, is discussed in turn.

5.1 Requirements Engineering

Given the dynamic nature of the set of input formats that RSVP would be required to process, the creation of an object-based representation ensured that the relationship between the data-processing software and the input data was thoroughly examined. This process uncovered the fact that the same nominal data type across different formats might have different encodings, use different units of measurement, or allow a different range of valid data values. It also identified exactly which data types were required by the processing, in what format they had to be presented to the processing software, and in which data formats this data would appear. Links between the input format and the allowable set of processing functions could then be created without modification of actual data processing code, which for the majority of the development process remained legacy code. Data value or format modification could be done by the format-specific subclass before the extracted data were presented to the rest of the code.

5.2 Design

After the realization that all of the input formats could be represented as a data block nested in three layers of variable-length header and trailer block pairs, the *Record* class was created as the "parent" of a number of format-specific subclasses. This decision was to specifically ease the expected workload of adding a new class every one or two years. The *ioType* class was created for the same purpose (the use of new media was an expected upgrade path), and used the same parent-child relationship. The inheritance depth of RSVP is only one, between these parents and their specific subclasses: the project was not objectified to the extent that an encompassing *DataProcessing* class was created.

5.3 Implementation

The *Record* class, as the central access point for all input data, publicly owns the most common data types (e.g. spacecraft identifiers, antenna and ground system identifiers, time-stamps). Format-specific data types, where either the data type itself was not common or where the representation was not format-unique, were isolated in format-specific subclasses. The *Record* class provided the means to traverse the data records and the procedures for to output the data to the scientific processing programs, while the format-specific subclasses manipulated the data types to provide a uniform representation and generic valid-data flags.

The addition of new input format subclasses do not affect any existing code, with the exception of a single localized modification to the *Record* class initialization procedure. Because any instance of a *Record* has the same interface, the high-level code to manipulate data records did not need to change over the later, updated, versions of RSVP. Similarly, new subclasses of *ioType* did not affect the high-level code, for exactly the same reasons.

6 Object-Oriented Induced Quality Aspects of RSVP

The development of RSVP is currently considered complete, with respect to the original project goals. While it is certainly not a perfect product, especially with respect to mathematical problems in specific scientific data processing stages, the overall stability and flexibility of the software is very satisfactory. New input media and data formats are not a cause for concern to the scientists using the product: they have worked through several format additions and found no adverse effects unintentionally introduced. The "core code" has not changed beyond the addition of initialization routines for new subclasses. Modifications to the data-extraction code does not affect the data-processing code, and vice-versa. While there are certainly some quality aspects which a formal software engineering process would have improved, those aspects affected by the object-oriented design process are recognized by the users as of high quality.

Given that there are very few specific, universally applicable standards for software quality, the best source of quality information is from the users. The quality of RSVP is generally divided into three categories by its users. The first is the ability to quickly adapt to new formats with respect to simple data-extraction and reformatting issues. The second is the ability to protect working portions of the project through numerous maintenance-level changes. The third is mathematical correctness. The first two issues were directly affected by the use of object-oriented design in the initial phase of development. The third issue, on the other hand, was not covered by object-oriented design techniques; the legacy code was converted in later, non-object-oriented development phases, and due to a lack of formal verification and validation methods, suffered during the translation.

6.1 Simple Modification to Include New Formats

The simple, yet very structured design of single-point data access via the *Record* class allows for new formats to be added easily and quickly. The initial step, that of creating the format-specific subclass, generally took only a few hours after the new format documentation was provided to the developer. If verification of the new subclass with test or actual data uncovered any implementation-level errors, they could generally be fixed in a matter of minutes or hours. Because this stage of the data processing generally involves only data extraction and necessary modifications to conform to the uniform data type representations, all such errors could only occur within the subclass definition. This localization of potential errors greatly increased the speed with which bugs could be found and fixed.

6.2 Localization of Code Modification Effects

Because RSVP was developed in a three planned phases, changes to each phase were incorporated with the expected work of the next phase. The deployment environment was limited to a "released" version and a "beta" version, which was essentially the developer's copy with minor privilege restrictions. Scientists who had requested a modification could, if necessary, only wait for as long as it took the developer to implement that single modification in the beta tree, then run that version to see if the modification made a difference to the processed

data. During these well-defined yet very informal maintenance phases, it was crucial that a modification not be able to affect calculations in unintended parts of the code.

The object-oriented structure of the code, which inherently promotes the low coupling and high cohesion principles, increased the quality of RSVP in this respect. Modifications to the transformation routines of format-specific data types were forcibly restricted affecting only that subclass. Modifications to one portion of the data processing code, while not created under an object-oriented paradigm, were nonetheless called on object-oriented data. This shielding allowed the scientists to get the earliest possible returns on modification requests without worry that a given change could affect the data outside the modification "effect field".

7 Conclusions

The creation of RSVP was the work of a single programmer who, like many in the workforce now, had neither heard of software engineering nor understood its principles and yet was considered a "good" programmer. As is typical in situations where there is a push to produce working code in a short time frame (in this case due to the unreliable nature of the hardware on which the legacy code could run), the programmer applied the principles she did understand, those of object-oriented design. While the overall quality and acceptance of the RSVP package by its users is considered good, certain important aspects were overlooked due to the total absense of formal software engineering techniques. However, the quality of RSVP is not accidentally, nor coincidentally, good. The object-oriented design used in the initial phase, where the legacy code had no impact on the new code except in determining output formats, created the most stable and reliable portion of the entire RSVP package. While this design process did not address all of the issues which should have been addressed, the quality of RSVP is almost certainly much higher than it would have been without the benefit of an object-oriented foundation.

References

[1] M. Paulk, B. Curtis, M. Chrissis, and C. Weber, "Capability maturity model for software.," *Software Engineering*, pp. 427–438, 1997.

[2] D. Hew, E. Sinderson, and L. Spirkovska, "The state of software engineering: Body of knowledge, education, certification, and licensing.." Final Project, Software Engineering Graduate Course, Nov. 1999.

[3] F. Bauer, "Foreword: Software engineering-a european perspective.," *Software Engineering*, p. 75, 1993.