2009 Brazilian Symposium on Aerospace Eng. & Applications
Copyright © 2009 by AAB

3rd CTA-DLR Workshop on Data Analysis & Flight Control
September 14-16, 2009, S. J. Campos, SP, Brazil

DEALING WITH LEGACY SOFTWARE SYSTEMS IN SPACE PROJECTS

Miriam B. Alves, miriamalves@iae.cta.br
Martha A. Abdala, martha@iae.cta.br
Instituto de Aeronáutica e Espaço - IAE
Praça Marechal Eduardo Gomes, 50
Vila das Acácias, Campus do CTA
Sao José dos Campos, SP - Brazil

Abstract. *Space software systems are usually employed in several space missions repetitively and, as a consequence, have a long life-cycle. These legacy systems are still being employed in important space projects and, in most of the cases, were designed using old fashioned structured analysis techniques and aged development platform. However, they cannot be overlooked. This paper describes an ongoing work at the Institute of Aeronautics and Space- IAE to conduct a process for updating legacy space software systems, considering a balanced approach when employing new technologies, still keeping traceability with the old models, even thought different techniques are applied. This transition aims not only to update the software but also incorporate new requirements derived from new space mission goals. Considering that technologies related to such software systems are in continuous progress, this initiative has two main benefits: bringing to the legacy systems and space projects technological innovations that can facilitate and improve their maintenance process, and keeping active systems that have proven to be cost effective and reliable. A case study was conducted using part of a flight control software system whereas old models were revised to reflect new requirements and new models were elaborated to complement the old ones. As a result, new tools and techniques could be used to improve the understanding of the software system, and to bring advances for the verification and validation process.*

***Keywords**: space software system, legacy systems, structured analysis, models, UML.*

1. INTRODUCTION

Space software systems play a fundamental role in space missions. In the case of spacecrafts, these systems are frequently involved in complex tasks as flight commands, telemetry, power, and attitude control. There are several scientific instruments on board the spacecraft that are also controlled by software. The flight control software of a satellite launcher, for example, is responsible for the control of the launcher – apart from the launcher's destruction – from a few minutes preceding the lift-off until the satellite has been deployed into the Earth's orbit. This implies that the software has to deal, in real time, with events that often occur asynchronously and these events can affect the process of control itself. In the case of a embedded system, the situation is more complex due to the hardware design dependency. The software requirements can only established, documented, and reviewed when the hardware design is almost established. On the top of all these issues, there is still the essential question of how the software system will be verified, validated and tested (Pisacane, 2005).

The process of engineering a space software system includes at least five phases, starting with the requirements definition followed by preliminary design, detailed design, implementation, and qualification tests. There is also the verification and validation process that occurs in parallel to all of these phases to assess the development resulting products. Each of these phases produces semantic and logical models, whereas the level of details increases as the software development approaches its end. Techniques are applied to elaborate the models, and these techniques are usually associated with a well defined methodology. As a result, the models will reflect some specific way to break down the complexity of the requirements in smaller and more manageable pieces, which basically will be oriented by data, function, object, or tasks. The resulting models will, occasionally, have a combination of these methodological approaches, normally with a predominant one.

As these models become more accurate in details, they can be transformed in code that will be compiled and, consequently will be converted into an executable program. If the software is embedded in some hardware, the choice of the programming language is made in advance because the software development environment must be able to generate code that will run in the selected on-board processors. This fact may limit the choice. A structured language is the most frequent choice for such software systems, due its facility to modify, test, and implement.

The choices of the programming language and the available techniques and tools have driven the methodology employed in most of the legacy space software systems that are still in use today. The majority of the models were elaborated using Structured Analysis and Design, with some variations to include real time characteristics (DeMarco, 1978; Ward and Mellor, 1981; Hatley and Pirbhai, 1987).

Legacy systems surprisingly have a big position in the space systems trade analysis. Many software systems and equipments remains the same and they are still fulfilling their purpose. However, new technology opens a door to a range of new services and capabilities. Therefore software engineers have to be creative in terms of considering next-generation solutions while at the same time maintaining and sustaining old legacy systems for established projects.

According to (Comella-Dorda et al, 2000), system evolution activities include maintenance, modernization, and replacement and they are applied at different phases of the system life cycle. Space software systems have very specific characteristics and a thorough analysis of such systems needs to be done before choosing the evolutionary activity that is most appropriate at different points of their life-cycle. Our experience has shown that the best choice is a balanced combination of maintenance, modernization, and replacement

Most of the modernization techniques related to legacy systems found in the literature are connected to legacy information systems (Chowdhury and Iqbal, 2004), which are very different from space software systems in complexity, use and real time restrictions.

The management of a space software system evolution is a continuous challenge that has to cope with new mission requirements changes and new technology as it becomes available. In the majority of the cases, the existence of new technology cannot pass unnoticed since its non- adoption might cause the impossibility of legacy system maintenance.

This paper relates the efforts to conduct an updating and a modernization of space legacy systems at IAE, considering the importance of such systems in the actual space projects. A balanced approach is adopted in order to keep up with new technology, still maintaining the traceability with the legacy system models and code, even thought different techniques are applied. This transition aims not only to update the software but also incorporate new requirements derived from new space mission goals.

The next section explains the big role that legacy software systems play in the context of space systems design. Section 3 presents our approach to update and maintain legacy systems, considering a balanced insertion of new technology encompassing models and code, reflecting in the improvement of the verification and validation process as well. Section 4 presents a case study, whereas part of a flight control software system is used to show the benefits and insights of such approach. Section 5 brings up some important conclusions.

2. THE ROLE OF LEGACY SOFTWARE SYSTEMS IN SPACE PROJECTS

Most of embedded space software systems provide the intelligence needed to bring up the required functionality of a specific hardware or subsystem. Typically, a satellite launcher has an on-board computer with embedded flight control software in charge of all the flight control events until the satellite is put on Earth's orbit. Besides, the software is responsible for the checkout of various launcher systems, including inertial platforms, autopilot chain and sequencing chain. Each flight has different characteristics and the software has to be prepared to incorporate new requirements, such as a new type of inertial measurement unit.

Devices play an important role in space systems doing specialized tasks, as sensors and actuators. They are usually controlled by embedded software. These devices are fundamental for the spacecraft performance and the software sends information to them, according to some strategy of control for the mission. Most of these devices have real-time operational restrictions, which mean that it is necessary to guarantee the execution of code within a certain time window.

2.1. Models

Most of the legacy systems still in use were specified using Structured Analysis and Design techniques. Structured Analysis (DeMarco, 1978) and its subsequent variations were the result of an evolutionary process. This process began with the need to write larger and more complex software systems which resulted in the use of the concept of a module as a form to help control complexity.

By that time, the focus of attention was specifically the dual issues of correctness and maintainability. The definition or specification of the problem to be developed was an additional concern addressed by Structured Analysis. Later, modifications in the Structured Analysis methods have been done in an attempt to address real time systems (Ward and Mellor, 1981; Hatley and Pirbhai, 1987). Another extension of Structured Analysis is referred to as HRT/DM- Hard Real Time (Peters, 1989).

In Structured Analysis (SA), an application is modeled in terms of abstractions of its processing, data storage, data flow, and external elements. The central specification tool of SA is the data flow diagram (DFD), which provides a pictorial, data flow representation of an application. The creation and interpretation of DFDs and their supporting information are based on syntactic and semantic notions that are not formally defined, thus the SA specification method and its products are referred as informal models. An application model is represented by: a hierarchy of DFDs and a set of data object definitions

However, the need to state application requirements unambiguously and to rigorously reason about specified application properties has generated much interest in formal specification techniques. A formal specification technique consists of a precise specification language and a reasoning system that can be used as a basis for rigorous specification analysis (France and Docker, 1993). Nowadays these techniques are becoming more popular amongst the software engineers, and they are frequently used to specify critical parts of a system.

2.2. Code

The reliability of computer hardware is as important as the reliability of embedded software. A simple bit inversion may cause the loss of the mission. There is a high software dependency on hardware, including reliable proprietary protocol. There is a natural resistance in abandoning a system code that performs well and that has proven to be efficient and correct.

The costs of updating and sustaining legacy systems are quite tangible, while the benefits of employing new technologies is hard to prove and measure at first. The adoption of new technologies to update a legacy system and deal with hardware obsolescence has its drawbacks. As legacy systems are converted to use modern object-oriented languages and techniques, it is necessary to convert the existing Structured Analysis and Design models to Unified Modeling Language - UML (Douglass, 2004; Douglass, 2000), but the code is still kept in a structured language, like C for example. In doing so, it is necessary to keep the traceability of the models. The need for backwards compatibility has been greater in space systems, considering the long cycle of life of these systems.

Another problem with higher level languages for real-time systems, such as Java, is that the more you abstract from the hardware, the harder it is to control the timing of execution. Workstations are not the same as an on-board computer for spacecrafts.

3. THE BALANCED INSERTION OF NEW TECHNOLOGY

This section presents the experimental approach to sustain legacy systems and its foundation. The objective is to offer a set of rules and suggestions for enhancing the models and getting the code up-to-date with new space mission requirements, restrictions, and technology evolution.

The insertion of new technology in space software systems ought to be a careful process that has to consider all the implication at system engineering level. One option could be replace the legacy system. However, replacement is essentially building a new system and, space software systems are very resource intensive, and require extensive testing using specific laboratories to reproduce the space mission environment and conditions. Besides, to get the software totally tested and tuned, demands considerable experience and expertise.

When evaluating the pros and cons of sustaining legacy systems versus upgrading them or developing new ones, cost is one of the guiding forces, but the point is whether costs can pay back through extensions of the software system caused by new mission requirements and prolonged use. Cost is one part of the tradeoff analysis which also takes into account a system with reliable use evidence.

Other issues are related to personal training and turnover as well as technical obsolescence. Finding needed expertise in out-of-date technologies becomes increasingly difficult and expensive. It is often a challenge to keep certain key competencies that are fundamental to maintain software legacy systems. As the projects usually have a long life-cycle, it happens that experienced and senior professionals retire or leave the projects to assume other managerial positions in higher level of the organizational structure. The solution has to be flexible and sometimes creative to accommodate this new situation. The software team will be composed by some veteran and new personal, and the new personal are not experienced professionals neither in the techniques applied to develop the legacy system nor have a complete understanding of the system, but they can substantively help to make use of new technology to improve system's performance, incorporate new mission requirements and update the system's semantic models.

3.1. Models

Models, as a form of graphical documentation, can help software engineers, providing a comprehension of the system before alterations can be implemented, making these changes limited and predictable (Tilley, S. and Huang, S., 2003).

There are some radical proposal to convert non-object structured analysis and structured design models of legacy systems into UML artifacts. The approach proposed in (Fries, 2006) does not take into account the complexity and real time characteristics of critical systems, and considers a simplistic approach that the structured analysis and design will only produce a set of DFDs and entity relationship diagrams. In fact, this is not actually true in the realm of space software systems. There is another factor not considered: most of the real time software systems are implemented using a structured language. So, such kind of proposed approach may work well when documenting simple information systems.

UML diagrams, as any other diagrams, offer a pictorial view of certain aspects of the system and they have their weakness as well. Tilley and Huang (2003) pointed out that experiments have shown that UML's efficacy is limited by its syntax and semantics and also by its domain representation. They added that there is no scientific evidence that shows that certain types of diagrams are better or more appropriate than others to help in the system understanding. There is no real proof that object-oriented techniques are better than structured methods (Fernandes and Lilius, 2004).

Instead of replace all the structured models by objected-oriented artifacts, there is also a possibility of integrating DFDs into an object-oriented development strategy based on UML as proposed by (Fernandes and Lilius, 2004). This integration is done by mapping the DFDs into UML concepts, which has its pros and cons. The alleged positive aspect of restricting the models in UML meta-models is the use of CASE tools or back processing systems for model transformation, validation, and code generation. However it is uncertain how trustful these tools can be in the matter of generating code and validation tests for real time and critical mission software systems.

The idea proposed by Fernandes and Lilius takes DFDs as an extension or adaptation of an existing UML diagram (Truscan, Fernandes and Lilius, 2003). They suggest that DFDs should be used in the development of embedded systems as a complementary diagram, transforming them in a UML diagram, or giving an UML diagram a DFD flavor. Therefore, the DFD meta-model must be mapped in UML concepts. Although this seems to be an interesting alternative from an academic point of view, it seems to be confused for practical use, deforming well established concepts already incorporated in legacy systems.

The approach proposed by this work is slightly different from the ones mentioned before, but the practical results have proven to be an adequate one for legacy software systems within space projects. The main idea is to establish a strategic combination of both paradigms, structured and object-oriented ones, having in mind that all the design models will inevitably be transformed in structured code. In this sense there is no point of transforming DFDs in UML concepts, but instead use them as complementary artifacts. Besides, there are not only DFDs to be used in real time structured analysis. Although they are the central diagrams when this approach is adopted, task diagrams and structured charts are also used to model different perspectives of the same software system.

The UML models may be used to complement the modeling process and enrich the legacy system documentation. Typically in structured analysis legacy models, the first DFD to be elaborated contains the main data flows and data controls that link the external interfaces to the system and, it also shows the system's principal functions. However, it is difficult to get from a DFD the dynamic aspect of the different interactions during the system operation scenarios. A use case diagram should then be elaborated to catch this aspect of the software system. The elaboration of use cases does not automatically mean that an object-oriented paradigm will be adopted as well.

Once the use case diagram is created, activity diagrams can be elaborated for each use case to show the different interactions amongst them. These activity diagrams can be confronted with the lower level DFD's for correctness and complementarities. Message Sequence Charts (MSCs) should be also elaborated to illustrate different scenarios associated to each use case, and, therefore, take into account other elements of the system. These elements are present in other legacy diagrams, from the structured analysis, like subsystems, main procedures, tasks and data components. In that way, a thorough traceability between the models can be obtained.

The approach adopted in this work uses new techniques for modeling, as UML diagrams, in order to improve the understanding of the internal system operation, its components, and their relationships, complementing the old models that represent the system at a higher level of abstraction. New models can also be used to better analyze the inputs and outputs of a legacy system inside its operating context, resulting in a gain of the understanding of the system interfaces. Message sequence charts, collaboration diagrams and activity diagrams enrich the understanding of the system interfaces and also the communication between different elements in the system, enhancing, and improving the information about the behavioral model, functional model, control model, and data model.

3.2. Code

Legacy software systems are continuously updated in order to correct errors, to provide new functionality, or portability to new operational systems and modern hardware platforms. However, software re-engineering activities should be conducted carefully, and it is important to incorporate non-functional requirements as part of this process.

Translating the code semantics of the often procedural legacy system to the hierarchic semantics of an object-oriented language can be a difficult task and can add an overhead in the execution not admitted for real-time systems. Besides, the creation and destruction of objects and the garbage collection is uncontrolled in such implementation, which can compromise the performance and priority of the critical functional tasks. There is not any available example yet of some successful use of object-oriented techniques during a whole life cycle of a space software system.

Automatic code generation is rarely used for space software system because the complexity of such systems requires experienced personal to write code. The implementation models include code understanding which involves the understanding of the underlying system structure. This process requires knowledge of the internals of a legacy system. The system's value must be preserved, so that the updated system must be as robust or functional as the old one.

Once modified, space software legacy systems has to undergo to a rigorous qualification process, which includes meticulous tests, since recent events in space showing significant failures have stimulated the creation of more rigorous testing procedures and processes. There will be more pressure on software engineers to justify the added expense of qualifying redesigned software using new components and new technology. As hardware technology evolves very quickly today, sometimes in the case of hardware obsolescence it costs more to maintain legacy hardware than to buy modern and more capable one. But updating hardware implies to refresh the software as well.

Code modernization requires a more traditional and conservative strategy.

4. EXPERIENCING THE EVOLUTION: A CASE STUDY

This section presents as a case study part of a flight control software for a satellite launcher. The real diagrams are not shown, but the adopted strategy and the resulting benefits. Figure 1 shows a schema of a generic flight software control. This software is connected to specialized hardware outside of the on-board computer and provides attitude control. The on-board computer usually has no peripherals, so it is hard to know what is happening during the tests.

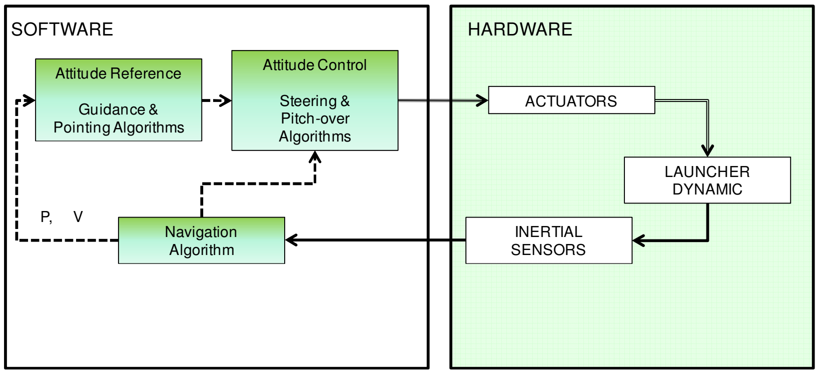

Figure 1. The generic main functions of the flight control software.

The documentation of this system consists of Structured Analysis and Design models and it is implemented in a procedural language. New requirements were incorporated in the system, resulting from space project changes and new mission goals. The introduction of new incremental improvements was done by creating new models based on UML, and afterwards, correlating these new models to the old structured and data flow driven models elaborated before. The old models were also revised to reflect the changes.

A pictorial view of the adopted strategy for dealing with this specific case study is showed in the figure 2. The several levels of DFDs were numbered starting with 0, indicating the highest level.

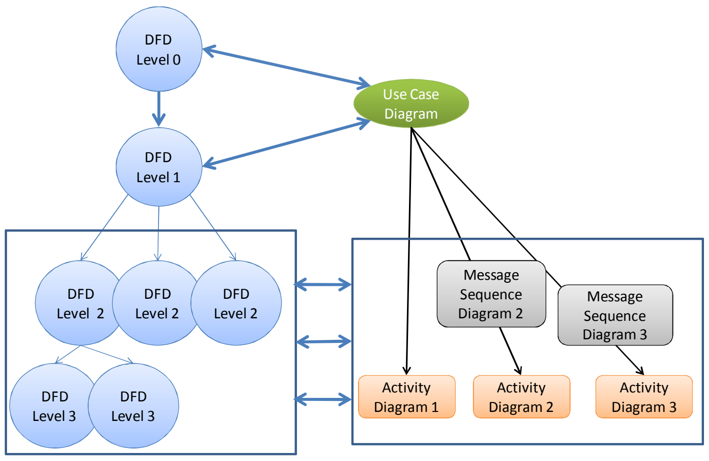

Figure 2. The strategy adopted for the case study models.

A use case diagram was elaborated considering the two first levels DFDs as references, and also taking into account the information of each process specification (PSPEC) within the DFDs. For each use case identified in the diagram, an activity diagram was constructed and, in some cases, a message sequence diagram was also elaborated.

The actors of the use cases were taken from the DFD level 0. They actually correspond to the external interfaces identified previously in highest level DFD. The message sequence charts used elements already identified in some old task diagrams, processes and structured charts. The processes of DFD level 1 became part of the use case diagram and the PSPECs helped to construct the activities diagrams. In this way, the traceability of the models could be kept and the integrity as well.

In order to exemplify the strategy, a specific functionality will be tracked. In the DFD level 1 there is a process responsible for controlling the flight events, called “Control Flight Events” as shown in Figure 3.

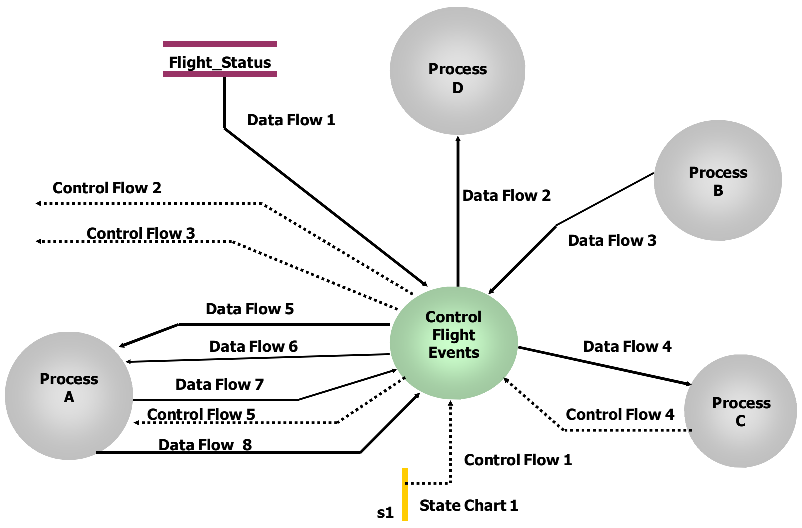

Figure 3. "Control Flight Events" process in the partial DFD level 1.

This process was also identified as a use case in the use case diagram as show in figure 4. The other use cases in the figure 4 are also related to the process A, B, C and D illustrated in Figure 3. It is interesting to observe that new relationships amongst these processes appear now in the use case diagrams that are not explicit in the DFD level 1. This fact reinforces the enrichment of the models and helps to better understand the intricate complexity of the system, by putting together different perspectives, functional and scenario-driven.

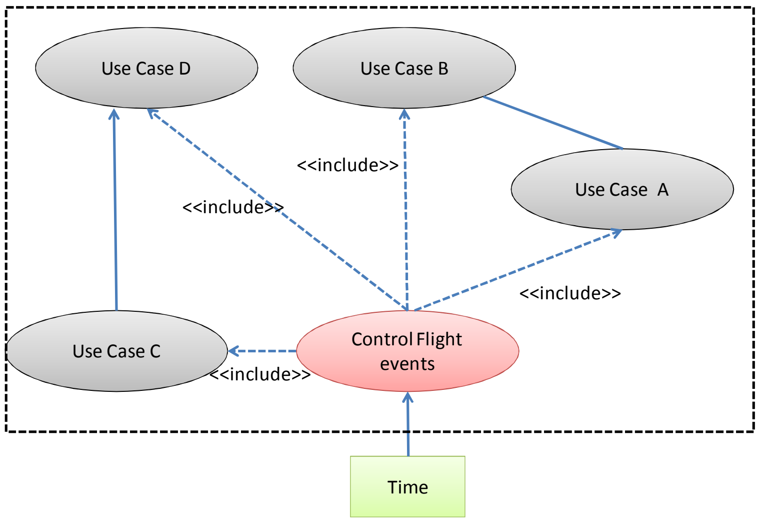

Figure 4. The use case "Control Events Flight" in the partial Use Case Diagram.

The next step was the elaboration of the activity diagram for the use case "Control Flight Events" based on the specification of the correspondent process and the new relationships with the other use cases identified in the main use case diagram. The resulting diagram showed a much better behavioral view of the process itself. Message sequence charts were built latter in order to study different scenarios associated with this use case in particular and other elements in the models.

This strategy was employed for all the processes and use cases and the resulting improvement has overcome the weakness of the structured analysis for the real-time modeling. During the process, the old models were also used as a reference to verify the new ones, based on the fact that the legacy system is an implemented image of old models. The gained insights helped planning a better verification and validation of the system and, as a consequence, sustaining its active life.

5. CONCLUSIONS

There is a growing alertness to provide evidence to support the practices of software engineering considering that there is not any solid audit trail that can provide a validation of these practices, linking theory and concepts. This work relates the knowledge acquired in such practices, during the several years of experience in the development of flight control software for the Brazilian Satellite Launcher Program. Nowadays, legacy systems unexpectedly have a big position in the space systems trade analysis and many software systems and equipments are still fulfilling their purposes very well. However, new technology opens a door to a range of new services and capabilities.

This paper presented a strategic approach to deal with legacy software systems updating and modernization. This approach defends the use of new technology in a complementary way, considering that there is not only a natural resistance to replace a system that has already proven to be correct but also high associated costs. The approach indicates the establishment of a strategic combination of both paradigms, structured and object-oriented ones, having in mind that all the design models will inevitably be transformed in structured code. The old models are used as a reference to verify the new ones based on the fact that the operational legacy software system is an implemented image of old models. The case study brought excellent results, especially related to the improvement of the system understanding for new personal and the verification and validation process. We seek to use this approach for large-scale software development projects, such as the Launching Preparation Ground System, to bring improvements to the strategy and collect metrics to prove its effectiveness and efficacy. There is no right recipe that could be used for all kinds of legacy software systems actualization. The proposed solution is a customized recipe that takes into account experience and expertise, domain knowledge, environment and perspective of life time.

6. REFERENCES

Chowdhury, M. W. and Iqbal, M. Z., 2004, "Integration of Legacy Systems in Software Architecture", In Proceedings of 2004 Specification and Verification of Component-Based Systems, Newport Beach, California.

Comella-Dorda, S., Wallnau, K., Seacord, R. C., Robert, J. 2000, "A Survey of Legacy System Modernization Approaches", Technical Note CMU/SEI-2000-TN-003, Carnegie Mellon University. Available at http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tn003.pdf.

De Marco, T., 1978, "Structured Analysis and System Specification", Yourdon Press, New York, N.Y.

Douglass, B. P., 2000, "Real-time UML – Developing efficient objects for embedded systems", Addison-Wesley, 2nd Edition, 328 p.

Douglass, B. P., 2004, "Real-time UML – Advances in the UML for real-time systems", Addison-Wesley, 3rd Edition, 693 p.

Fernandes, J. M., Lilius, J., 2004, "Functional and object-oriented views in embedded software modeling". In Proceedings of 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04).

France, R.B., Docker, W. G., 1993,"Towards CASE tool support for rigorous structured analysis", 0-8186-4212-2/93, IEEE.

Fries, T. P., 2006, "A framework for transforming structured analysis and design artifacts to UML", In Proceedings of SIGDOC'06, October 18-20, Myrthe Beach, South Carolina, USA.

Hatley, D, Pirbhai, I., 1987, "Strategies for Real Time System Specification" Dover Press, New York, N.Y.

Peters, L., 1989, "Timing Extensions to Structured Analysis For Real Time Systems", Software Consultants International, Limited, Kent, Washington.

Pisacane, V., 2005, "Fundamentals of Space Systems", Oxford University Press, New York, NY, 828 p.

Tilley, S., Huang, S., 2003, "A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding", In Proceedings of SIGDOC'03, October 12–15, San Francisco, California, USA.

Ward, P, Mellor, S., 1981, "Structure Analysis of Real Time Systems" , Yourdon Press, NewYork, N.Y.